



Amin Majd

Distributed and Lightweight Meta-heuristic Optimization Method for Complex Problems

TURKU CENTRE *for* COMPUTER SCIENCE

TUCS Dissertations
No 244, October 2019

Distributed and Lightweight Meta-heuristic Optimization method for Complex Problems

Amin Majd

To be presented, with the permission of the Faculty of Science and Engineering, for public criticism in the Auditorium of Agora XXII in October 5th, 2019 at 12:00.

University of Turku
Department of Future Technologies
20014 TURUN YLIOPISTO

2019

Supervised by

Professor Hannu Tenhunen
Department of Future Technologies
University of Turku
Finland

Professor Juha Plosila
Department of Future Technologies
University of Turku
Finland

Associated Professor Masoud Daneshtalab
Department of Embedded Systems, division of Intelligent Future Technology
Mälardalen University
Sweden

Reviewed by

Professor Amir Hossein Gandomi
Faculty of Engineering & Information Technology
University of Technology Sydney
Ultimo, NSW, Australia

Professor Ning Xiong
School of Innovation, Design and Engineering
Mälardalen University
Västerås, Sweden

Opponent

Senior Lecturer Amir Pourabdollah
Department School of Science and Technology
Nottingham Trent University
Nottingham, England

Painosalama Oy, Turku

ISBN 978-952-12-3864-2

ISSN 1239-1883

Abstract

The world is becoming more prominent and more complex every day. The resources are limited and efficiently use them is one of the most requirement. Finding an Efficient and optimal solution in complex problems needs to practical methods. During the last decades, several optimization approaches have been presented that they can apply to different optimization problems, and they can achieve different performance on various problems. Different parameters can have a significant effect on the results, such as the type of search spaces. Between the main categories of optimization methods (deterministic and stochastic methods), stochastic optimization methods work more efficient on big complex problems than deterministic methods. But in highly complex problems, stochastic optimization methods also have some issues, such as execution time, convergence to local optimum, incompatible with distributed systems, and dependence on the type of search spaces.

Therefore this thesis presents a distributed and lightweight meta-heuristic optimization method (MICGA) for complex problems focusing on four main tracks. 1) The primary goal is to improve the execution time by MICGA. 2) The proposed method increases the stability and reliability of the results by using the multi-population strategy in the second track. 3) MICGA is compatible with distributed systems. 4) Finally, MICGA is applied to the different type of optimization problems with other kinds of search spaces (continuous, discrete and order based optimization problems).

MICGA has been compared with other efficient optimization approaches. The results show the proposed work has been achieved enough improvement on the main issues of the stochastic methods that are mentioned before.

Tiivistelmä

Maailmasta on päivä päivältä tulossa yhä monimutkaisempi. Resurssit ovat rajalliset, ja siksi niiden tehokas käyttö on erittäin tärkeää. Tehokkaan ja optimaalisen ratkaisun löytäminen monimutkaisiin ongelmiin vaatii tehokkaita käytännön menetelmiä. Viime vuosikymmenien aikana on ehdotettu useita optimointimenetelmiä, joilla jokaisella on vahvuutensa ja heikkoutensa suorituskyvyn ja tarkkuuden suhteen erityyppisten ongelmien ratkaisemisessa. Parametreilla, kuten hakuavaruuden tyypillä, voi olla merkittävä vaikutus tuloksiin. Optimointimenetelmien pääryhmistä (deterministiset ja stokastiset menetelmät) stokastinen optimointi toimii suurissa monimutkaisissa ongelmissa tehokkaammin kuin deterministinen optimointi. Erittäin monimutkaisissa ongelmissa stokastisilla optimointimenetelmillä on kuitenkin myös joitain ongelmia, kuten korkeat suoritusajat, päätyminen paikallisiin optimipisteisiin, yhteensopimattomuus hajautetun toteutuksen kanssa ja riippuvuus hakuavaruuden tyypistä.

Tämä opinnäytetyö esittelee hajautetun ja kevyen metaheuristisen optimointimenetelmän (MICGA) monimutkaisille ongelmille keskittyen neljään pääavoitteeseen: 1) Ensisijaisena tavoitteena on pienentää suoritusaikaa MICGA:n avulla. 2) Lisäksi ehdotettu menetelmä lisää tulosten vakautta ja luotettavuutta käyttämällä monipopulaatiostrategiaa. 3) MICGA tukee hajautettua toteutusta. 4) Lopuksi MICGA-menetelmää sovelletaan erilaisiin optimointiongelmiin, jotka edustavat erityyppisiä hakuavaruuksia (jatkuvat, diskreetit ja järjestykseen perustuvat optimointiongelmat).

Työssä MICGA-menetelmää verrataan muihin tehokkaisiin optimointimenetelmiin. Tulokset osoittavat, että ehdotetulla menetelmällä saavutetaan selkeitä parannuksia yllä mainittuihin stokastisten menetelmien pääongelmiin liittyen.

Acknowledgements

I would like to express my special appreciation and thanks to my supervisors: Associated Professor Masoud Daneshtalab, Professor Juha Plosila, and Professor Hannu Tenhunen that without their supervision, and scientific support this doctoral thesis would not have been possible. I want to especially thank Associated Professor Masoud Daneshtalab for his comprehensive support when I entered the University of Turku and during my studies.

I am thankful for the Nokia Foundation, and CIMO Fellowships programme for financial support.

I would like to thank my colleagues and co-authors: Associate Professor Elena Troubitsyna, Golnaz Sahebi, Dr Shahriar Lotfi, Dr Nima Khalilzad, Mahdi Abdollahi, Mohammad Loni, and Maghsoud Salimi.

I would also like to thank Professor Amir Hossein Gandomi and Professor Ning Xiong for accepting to be the reviewer of my thesis. I also thank Senior Lecturer Amir Pourabdollah for accepting to be the opponent of my thesis.

I would like to especially thank my mother, Sorour, and my father, Hossein, for all the supports that they provided during my life.

Also, I would like to thank my daughter, Diana, because I have spent some of her time to complete my studies.

Finally, I would like to express my deepest appreciation to my brilliant wife, Golnaz Sahebi, that without her support and inspiration this journey would not have been started. She was my best friend, colleague, and supporter over the last 15 years.

List of original publications

The work discussed in this dissertation is based on the publications listed below:

- | | |
|-----------|--|
| Paper I | A. Majd , Sh. Lofti, G. Sahebi, M. Daneshtalab and J. Plosila, “PICA: Multi-Population Implementation of Parallel Imperialist Competitive Algorithms,” 24th Euromicro International Conferences on Parallel, Distributed and Network-Based Processing, PDP 2016. |
| Paper II | A. Majd , G. Sahebi, M. Daneshtalab, J. Plosila, Sh. Lotfi, H. Tenhunen, “Parallel Imperialist Competitive Algorithms” Concurrency and Computation Practice and Experience, WILEY, 2018. |
| Paper III | A. Majd , M. Abdollahi, G. Sahebi, D. Abdollahi, M. Daneshtalab, J. Plosila and H. Tenhunen, “Parallel Imperialist Competitive Algorithm Based on Multi-Population Technique for Solving Systems of Nonlinear Equation,” The 2016 International Conference on High Performance Computing & Simulation (HPCS), 2016. |
| Paper IV | M. Salimi, A. Majd , M. Loni, C. Seceleanu, T. Seceleanu, M. Sirjani, M. Daneshtalab, and E. Troubitsyna, “Finding Near-Optimal Task Scheduling for Distributed Real-Time Environments” 6th Conference on the Engineering of Computer Based Systems (ECBS19), 2019. |
| Paper V | A. Majd , G. Sahebi, M. Daneshtalab, J. Plosila and H. Tenhunen, “ Placement of Smart Mobile Access Points in Wireless Sensor Networks and Cyber-Physical Systems using Fog Computing,” The 16th IEEE International Conference on Scalable Computing and Communications (Smart World 2016). |
| Paper VI | A. Majd , G. Sahebi, M. Daneshtalab, J. Plosila and H. Tenhunen, “ Hierarchal Placement of Smart Mobile Access Points in Wireless Sensor Networks using Fog Computing, ” 25th Euromicro International Conferences on Parallel, Distributed and Network-Based Processing, PDP 2017. |

- Paper VII | **A. Majd**, G. Sahebi, M. Daneshtalab, E. Troubitsyna, “Optimizing Scheduling for Heterogeneous Computing Systems using Combinatorial Meta-heuristic Solution” The 17th IEEE International Conference on Scalable Computing and Communications (Smart World 2017), 2017.
- Paper VIII | **A. Majd**, M. Daneshtalab, J. Plosila, N. Khalilzad, G. Sahebi, and E. Troubitsyna, “NOMeS: Near-Optimal Metaheuristic Scheduling for MPSoCs”, 19th International Symposium on Computer Architecture and Digital Systems (CADS’17), 2017.

Contents

I	Research Summary	1
1	Introduction	3
1.1	Contributions of Thesis	4
2	Related Works	7
2.1	Genetic Algorithm (GA)	7
2.2	Imperialist Competitive Algorithm (ICA)	8
2.3	Parallel Evolutionary Algorithms	9
2.3.1	Parallel Genetic Algorithm	9
2.3.2	Parallel Ant Colonies	10
2.3.3	Parallel ABC	10
2.3.4	Parallel PSO	11
2.3.5	Parallel Memetic Algorithm	11
3	Thesis Contributions and Case Studies	13
3.1	Thesis Contributions and the Hybrid Evolutionary Algorithm	13
3.2	list of case studies	15
3.2.1	Synthetic benchmarks	15
3.2.2	Nonlinear equations	15
3.2.3	Task Scheduling	17
3.2.3.1	Task graph scheduling	18
3.2.3.1.1	Order-based country (OBC)	19
3.2.3.2	Scheduling a pack of application	20
3.2.3.3	Task scheduling with more constraints on industrial application	20
3.2.4	Placement of Swarm of drones	22
3.2.4.1	Placement of SMAPs	25
3.2.4.2	Hierarchical Placement of SMAPs	25
4	Experimental Results	27
4.1	Synthetic benchmarks	27
4.2	Nonlinear equations	29
4.3	Task Graph Scheduling	31
4.3.1	Single Objective Optimization	35
4.3.2	Multi-Objective Optimization	36
4.4	Placement of Swarm of drones	38
4.4.1	Single SMAP	39
4.4.2	Multiple SMAPs	41

5	Discussion and Conclusion		45
6	Overview of Original Publications		47
6.1	Paper I	PICA: Multi-Population Implementation of Parallel Imperialist Competitive Algorithms	47
6.2	Paper II	Parallel Imperialist Competitive Algorithms	47
6.3	Paper III	Parallel Imperialist Competitive Algorithm Based on Multi-Population Technique for Solving Systems of Nonlinear Equation	47
6.4	Paper IV	Finding Near-Optimal Task Scheduling for Distributed Real-Time Environments	48
6.5	Paper V	Placement of Smart Mobile Access Points in Wireless Sensor Networks and Cyber-Physical Systems using Fog Computing	48
6.6	Paper VI	Hierarchical Placement of Smart Mobile Access Points in Wireless Sensor Networks using Fog Computing	48
6.7	Paper VII	Optimizing Scheduling for Heterogeneous Computing Systems using Combinatorial Meta-heuristic Solution	49
6.8	Paper VIII	NOMeS: Near-Optimal Metaheuristic Scheduling for MPSoCs	49

II Original Publications

Symbols and Abbreviations

2-D	Two dimensional
3-D	Three Dimensional
ACO	Ant colony optimization
C-PPSO	Course-grained PSO
CD	Conjugate direction method
COA	Cuckoo optimization algorithm
CP	Critical path of task graph
CPS	Cyber physical systems
DAG	Directed acyclic graph
DSC	Dominant sequence clustering algorithm
EA	Evolutionary algorithm
EC	Evolutionary computing
GA	Genetic algorithm
GWO	Grey wolf optimization
HSMAP	Hierarchical smart mobile access point
ICA	Imperialist competitive algorithm
IoT	Internet of Things
ISA	Interior search algorithm
KH	Krill herd
MA	Memetic algorithm
MD	Mobility directed algorithm
MICGA	Multi-population ICA and GA
MOP	Multi-objective Problem
MPGA	Multi-population GA
MPI	Message passing interface
MPICH2	An implementation of MPI
MPQGA	Multiple priority queues GA
MPSO	Multi-population PSO
MPSoCs	Multiprocessor System-on-Chips
MRPSO	Multi-population Repulsive PSO
MSEOA	Social-emotional optimization algorithm
NoC	Network on a chip
NP	Nondeterministic polynomial time
OBC	Order-based country
PABC	Parallel artificial bee colony
PACO	Parallel ant colony optimization
PACO-CGD	Parallel ant colony-coarse grained
PARME	Parallel memetic algorithm

PICA	Multi-population ICA
PMC	Priority-based multi-chromosome
Pp	Number of parts in each pack
PSO	Particle swarm optimization
SA	Simulated annealing
SD	Standard division
SMAP	Smart mobile access point
UAV	Unmanned aerial vehicles
WSN	Wireless sensor network

Part I
Research Summary

Chapter 1

Introduction

As the level of computing complexity become larger, artificial intelligence has shown enormous power in decreasing this complexity.

Generally, improving one parameter in a complex problem (e.g. accuracy in the machine learning problem), might have some side effects on other parameters. Therefore, optimization techniques in solving complex problems are the best solution to reduce such side effects. Therefore, evolutionary algorithms are selected as a meta-heuristic approach to optimization problems.

Evolutionary algorithms (EAs) are population-based search optimization methods that mimic the processes of natural selection and evaluation. They work based on the combination of some random operations with guided ideas. The guided ideas Inspired from understandable samples in natural competitions. They do not need particular assumptions like differentiability or continuity. They are really suitable for dealing with multi-objective-problems (MOP). In the past few decades, plenty of evolutionary algorithms have been proposed by researchers.

The main aim of an EA is to find the nearest optimal solution for some real-world problems [1]. The algorithm keeps on producing multiple generations of solutions until a generation achieves the most optimal solution. EA's first step is to establish a convertible link between the real world and a computational world of EA. This step relies on two fundamental concepts: phenotype and genotype.

A candidate solution (called an individual) is phenotype in the real world and the corresponding sample in the algorithm is a genotype that is called a chromosome in the genetic algorithm (GA). If the algorithm wants to optimize an equation integer variable, the algorithm can make a binary code of integer digits as a genotype. Also, a representation of a phenotype by a genotype is called encoding, while mapping of a genotype to a phenotype is called decoding.

In fact, the performance of EAs highly depends on multiple factors like a search space area, the number of optimization objects, the behaviour of

the problem (dynamic and static), and the complexity to evaluate a solution. So such factors are the origin of emerging new EAs.

The majority of EA starts from the stochastic generation of the initial population of genotypes in the overall search space according to a specific probability distribution. We can evaluate the fitness function for each genotype, which describes the requirements to which the population should adopt. A fitness function allocates quality measures to the genotypes and forces the population improvement. An evaluation of a fitness function typically requires decoding of a genotype into the corresponding phenotype and computing a certain quality measure.

An EA estimates the fitness function for all genotypes of a given population. The genotypes with the greater values of the fitness function take the higher probability to be accepted as the parents of the next generation. The chosen parents undergo variation to create offsprings. A variation consists of mutation and recombination [2]. The mutation is a unary operator applied to a genome to create a mutated mutant – a child (offspring). The mutation is stochastic, for instance, the child depends on the outcomes of random choices. EAs usually follow the same structures but they make a change by the ability of the operations and competition idea between populations.

In this thesis, different EAs have been reviewed while the behaviour of each EA is explored within the different context of problems.

1.1 Contributions of the Thesis

This thesis has dealt with the following open problems:

- EAs are not enough fast to apply for extremely complex problems and real-time applications. How can we reduce their execution time?
- EAs can converge to local optimum in complex problems. How can we improve the reliability of their results?
- Usually, EAs work based on the main population. How can we make them fit for distributed systems?
- EAs have different performance on different problems. How can we introduce an EA to apply to different applications?

To tackle these problems, the following solutions have been presented in this thesis:

- To reduce the execution time, we have implemented the parallel implementations of ICA and GA on Paper I and II.

- To improve the accuracy of our parallel method, we have introduced Paper II and tested our methods in two different areas.
- To make it fit for distributed systems we have selected multi-population method and tested it on different application domains on papers III, IV, V and VI.
- Finally, we have introduced a hybrid method, an efficient combination of multi-population ICA (PICA) and Multi-population GA (MPGA), to reduce the execution time, improve the accuracy of results, and make the ability to apply on different application domains and have tested on Paper VII and VIII.

Figure 1.1 illustrates the general overview of the works accomplished in this thesis and the cohesion of the publications.

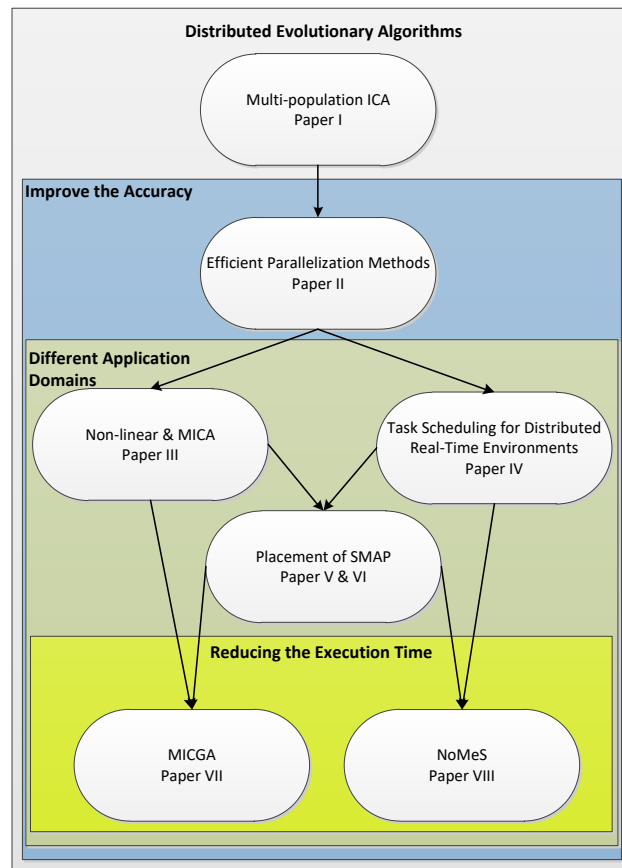


Figure 1.1 Illustration of papers cohesion. Each category is labelled with different colors.

Chapter 2

Related Works

Optimization problems have different levels of complexity and various search spaces. To optimize NP problems, the execution time is the primary objective. By growing the size of NP problems, optimization takes lots of time, so generally, ECs are more popular than other optimization problems because they are faster than other methods. In the last decades, many different ECs have been presented such as genetic algorithm (GA) [4], particle swarm optimization (PSO) [79], ant colony optimization (ACO) [18], simulated annealing (SA) [80], and grey wolf optimization (GWO) [81]. Also, krill herd (KH) [82] is proposed for solving optimization tasks. The KH algorithm is based on the simulation of the herding behaviour of krill individuals. The minimum distances of each individual krill from food and from the highest density of the herd are considered as the objective function for the krill movement [82]. The interior search algorithm (ISA) [83] as a novel method for solving optimization tasks achieves high accurate results.

Among them, GA is selected as an efficient and straightforward method for discrete problems, and imperialist competitive algorithm (ICA) [6] is chosen as a highly accurate method for continuous problems. Also, the parallel implementation of different EC methods has been studied to analyze how the execution time and accuracy of an EC method can be improved. This chapter introduces GA, ICA and some of the well-known parallel EA methods.

2.1 Genetic Algorithm (GA)

Genetic algorithms are population-based search methods that mimic the process of natural selection and evolution, which some of their characteristics can help researchers for optimization [3]. Each serial GA has an initial population (several random chromosomes that each one is an individual) and executes routine operations, such as selection, crossover, mutation, and replacement [3]. All operations are repeated until leading to a proper result or ending in a specific generation.

The simple model of GA was presented in 1970 by Goldberg and Holland [4]. In the last 40 years, plenty of research studies have been carried out to improve GA. These improvements make the GA more applicable to complex problems. The standard GA works with binary coding [5], which is suitable for discrete problems [3]. However, the performance on continuous data and convergence to local optimums are the key bottlenecks of GA.

2.2 Imperialist Competitive Algorithm (ICA)

In 2008, the Imperialist Competitive Algorithm (ICA) was introduced by E. Atashpaz and C. Lucas [6], inspired by imperialistic competition. ICA is an EA to optimize the linear and nonlinear NP-complete problems. It is a population-based method in which each possible solution is a country, corresponding to the chromosome concept in the genetic algorithm [6]. The algorithm starts by generating a set of countries, as the initial population. Then all countries are separated into two classes: imperialist and colonies.

Imperialistic competition is the main operation of this algorithm, and the expectation is that the colonies converge to the global optimum of the cost function, or at least very close to this optimum.

The primary sorting of the countries is based on their fitness function values. The best countries are elected to be the imperialists, and the rest of the countries will be the colonies of these imperialists (Figure 2.1, step 1) and they (imperialist and colonies) make different empires.

After distributing all colonies among the imperialists, colonies move toward their relevant imperialist countries. This movement operated by assimilation and revolution operations (Figure 2.1, step 2). In the assimilation, colonies change their positions in the search space to some places that are closer to their imperialist. This operation is similar to recombination operation in EAs and crossover in GA. Ordinarily, assimilation does exploitation in the search space and helps to have a more accurate search around the solutions. Using the assimilation (exploitation) alone as the recombination operation in the EAs can push our solution to the local optimum. As the best solution for this problem, the revolution operation as an exploration method can force the algorithm to try another part of the search area and avoid converging to a local optimum.

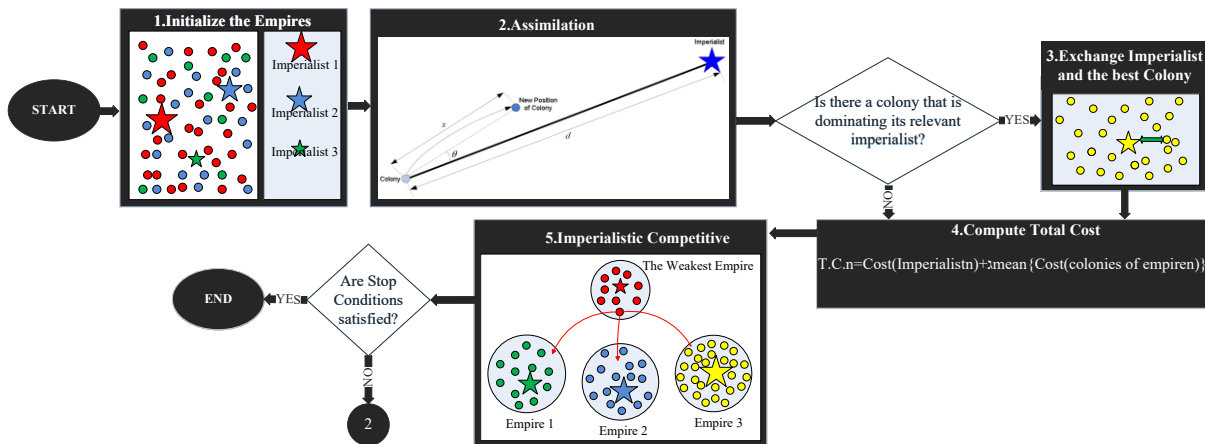


Figure 2.1 ICA Flowchart

For each colony in each iteration, two random real numbers ranging between zero and one are generated. Then these values are compared with the probabilities of assimilation and revolution (i.e. revolution rate). If the generated numbers are lower than these probabilities, the procedure of assimilation or revolution is performed. In the next step, ICA calculates the total power of each empire and the imperialistic competition begins. The weakest empire loses its weakest colony and randomly one imperialist

capture this colony. These steps are repeated until the termination condition is satisfied.

The ICA is a suitable method for a variety of optimization problems, but there exist certain challenges concerning the evolutionary algorithms in general such as convergence to local optimum or long execution time.

2.3 Parallel Evolutionary Algorithms

EAs can help to solve various problems, but there are some disadvantages associated with them. For instance, if the search space is very large, it is possible that an EA does not converge towards the global optimum or even to near-optimal solutions. To improve the result in such cases, the initial population should be expanded. The execution time of EAs is another challenge which can be intolerably high in complex cases. Parallelization can enhance the quality of decisions and reduce the time of obtaining results.

Several parallel EA methods have been proposed to achieve better results (e.g., parallel genetic algorithm [3][7][8][9], parallel ant colony optimization (PACO) [10][11][12], parallel memetic algorithm [13], parallel ABC (PABC) [14][15], and parallel PSO [16][17]). Parallel EAs are categorized into four main groups [3], master-slaves, multi-population, fine-grain and hierarchical methods. They have different usage based on different problems, goals, and computational resources. The master-slave method applies to problems that have a complex fitness function [18] to improve computation time (execution time). The execution time of EAs is not the only concern, but the reliability of the results and avoiding to converge to local optimums are two important challenges in EAs which can be enhanced by Multi-population (Coarse-grain) method. The fine-grain parallel method is similar to multi-population, but require huge computational resources [3]. The hierarchical method can be designed as a combination of the other three methods (e.g. using the multi-population strategy on the top layer and master-slaves on the lower level). Both the fine-grain and hierarchical methods are resources hungry.

2.3.1 Parallel Genetic Algorithm

Here we will explore different types of parallel GA [3]. In coarse-grain methods, several processors are available and each of which has independent initial populations; and each processor runs a serial GA. After a definite number of iterations, all processors will stop and transfer some chromosomes (the migration operation) with a predefined strategy, e.g., worse or best. By this operation, the processors share the results of solutions among each other. In the coarse-grain, the most three important parameters are migration gap (the number of iterations between two migration), migration rate (the number of chromosomes that migrate at each the migration time), and interconnection topologies such as the ring or fully connected topology.

Another parallel GA method is master-slave. In this method, one processor is specified as a “master” to do the more complex computations of GA, such as replacement, and selection, whereas the other processors (slaves) evaluate the fitness function and transfer the results to the master

processor. The fitness function defines the quality of each individual (chromosome). Master-slave nodes can be implemented either synchronously or asynchronously. In the asynchronous method, the master processor proceeds with its work without waiting for the transferred data of slave nodes. In the synchronous method, the master node holds until it receives the results of all tasks from slaves.

Fine-grain methods are suitable for parallel computing with a massive number of processors where each processor is able to communicate with the neighboring processors where each chromosome can recombine with each chromosome on the neighborhood of processors. This method is faster than the master-slave method but the required resource (such as the number of processors and communication rate) is too high.

The hybrid method is composed of 2 levels: the upper level uses the coarse-grain method while the lower level utilizes either the master-slave, the multi-population, or the fine-grain method. This is more efficient and faster than other methods because it can utilize the advantages of two methods at the different levels of hierarchy.

2.3.2 Parallel Ant Colonies

Ant colony optimization (ACO) [19] is a technique for approximate optimization. The inspiring source of ACO algorithm is real ant colonies. More specifically, ACO is inspired by the ants' foraging behaviour. At the core of this behaviour is the indirect communication between the ants by means of chemical pheromone trails, which enables them to find short paths between their nest and food sources. This characteristic of real ant colonies is exploited in ACO algorithms in order to solve, e.g., discrete optimization problems. Two main parallel implementations of ACO are PACO [11] and PACO-CGD [10].

PACO is applied to different optimization problems and is based on the multi-population method, where each processor has an independent population and runs the standard ACO independently. After a specified number of generations, a processor transfers some useful information to other processors. In PACO-CGD, the constructor graph decomposes into smaller pieces and each part is assigned to a processor, and then each processor runs the ACO method by itself; this method is faster than the PACO.

2.3.3 Parallel ABC

Artificial Bee Colony (ABC) [20] is inspired by honeybees. The ABC, as an optimization mechanism, provides a population-based search function in which the artificial bees modify individuals called food positions as time passes. The bees aim to explore the places of good quality and food sources with high nectar amounts and finally the one with the highest nectar amount. ABC can be parallelised in three ways: based on either the coarse-grain, master-slave, or hybrid method. Similar to the master-slave method for GA, one processor is selected as the master and others as the slaves. There is also the same similarity between multi-population and hybrid method of ABC with GA.

2.3.4 Parallel PSO

Many applications have taken advantage of particle swarm optimization (PSO) [21]. The PSO emulates the behaviour of animal societies that do not have any unique leader in their swarm, such as fish schooling and bird flocking. The PSO is used efficiently for continuous search space problems. Parallel PSO is an efficient method for optimal task scheduling in distributed systems [16], which has been implemented with two different multi-population techniques: Multi-population PSO (MPSO) and Multi-population Repulsive PSO (MRPSO).

MPSO runs like other coarse-grain methods where each processor has an independent population, and the simple PSO runs on its population independently [16]. The migration operation is also similar to GA.

The MRPSO is the enhanced MPSO method that combines an extra component to MPSO called "repulsive component". Trying to make a diverse population in each processor is the primary rule of this component. This causes a high degree of diversity in PSO which helps obtain better results.

2.3.5 Parallel Memetic Algorithm

Memetic algorithms (MAs) [22] are population-based and heuristic search approaches for optimization problems similar to GAs. GAs, however, rely on the concept of biological evolution but MAs mimic cultural evolution. Parallel MA [13] is implemented as a coarse-grain approach called PARME. It is used on optimization problems in the work of Vanneschi. Typically, parallel MA is implemented as a coarse-grain method called PARME [13].

The PARME is a coarse-grain method that uses a different population in each processor that runs a serial MA independently. One of the processors is selected as the master processor. This is the main difference between PARME and the other methods as mentioned earlier in parallel GAs. The master processor controls the tasks of other processors and generates a task table in each iteration and shared it with other processors. This task table has the information of significant parameters such as the fitness value of the best and the worst populations while the table will update in each iteration.

Chapter 3

Thesis Contributions and Case Studies

It is clear that different problems should be solved by different optimization methods. Here, we have improved our optimization method and use different types of coding, encoding, fitness functions and other needed operations based on the requirement of problems.

3.1 Thesis Contributions and the Hybrid Evolutionary Algorithm

As mentioned before, there are several open questions about the EAs that this thesis has addressed some of them. Today, due to the complexity of modern applications in IoT (internet of things) and CPS (cyber physical systems) domains, optimization methods have emerged as a vital solution. Generally, EAs are time-demanding computational models for complex problems, particularly when search spaces, and the number of objectives are increased. So, in the first effort, we did some research to select a fast evolutionary method, and we selected ICA due to its performance and accuracy.

ICA is an optimization method based on imperialistic competition. In this method, all countries (same as chromosomes in the GA) are divided into two categories: colonies and imperialist states. The main part of this method is imperialistic competition which causes the colonies to converge to the global minimum.

ICA can converge to the global optimum in the smaller iteration compared to other EA methods but was not fast enough for our use-cases. To improve the execution time of ICA, we decided to find a parallel solution where we have presented two different parallel implementations of ICA. In the first try (Paper I), we have proposed a multi-population implementation of ICA. We have connected the processors with the ring topology and tested our method on some well-known benchmarks. The results of these implementations show that we got considerable improvement in the execution time and accuracy in which super-linear performance [23] and system of nonlinear equations (in Paper III) achieved the best results. This improvement indicates that the migration operation has enough effect on getting the algorithm out from the local optimum and help the algorithm to have more exploration and exploitation in the search space.

Next, we implemented the multi-population and master-slave models of ICA in Paper II and evaluated it with different benchmarks. The results show that the master-slave can only improve the performance, while the multi-population can increase the accuracy at the same time.

We evaluated this method with another case study as a multi-objective problem. We introduced the concept of smart mobile access points

(SMAPs) to enhance node placement in wireless sensor networks (WSN). Finding the best placement of SMAPs is a multi-objective problem, and our algorithm should obtain the most coverage and connectivity of the network at the same time. The search space simulated as a 3-D mesh and each SMAP can select one point in the mesh. We used PICA to solve this problem (Paper V). However, as this problem requires a discrete-based EA, we presented an MPGA (Paper VI). With this, the execution time has been improved by using fast and straightforward exploration and exploitation operations (crossover and mutation) but could not guarantee the convergence to global optimum as PICA is able to do. This motivated us to discover why GA-based solutions cannot converge easily and accurately to the global optimum.

The GA operations are fundamentally suitable for discrete problems therefore not suitable for the convergence strategy in continuous problems is not useful [24], because the selection operation in GA misses some chromosomes that have a probability to become potential genes for obtaining the best results. However, In the ICA, we use countries as the population, all the countries will be available in all iterations; they may only move to other places in the search space. However, the selection operation highly depends on random functions which can easily converge to a local optimum. Therefore, we use an efficient convergence strategy in order to improve the reliability of results.

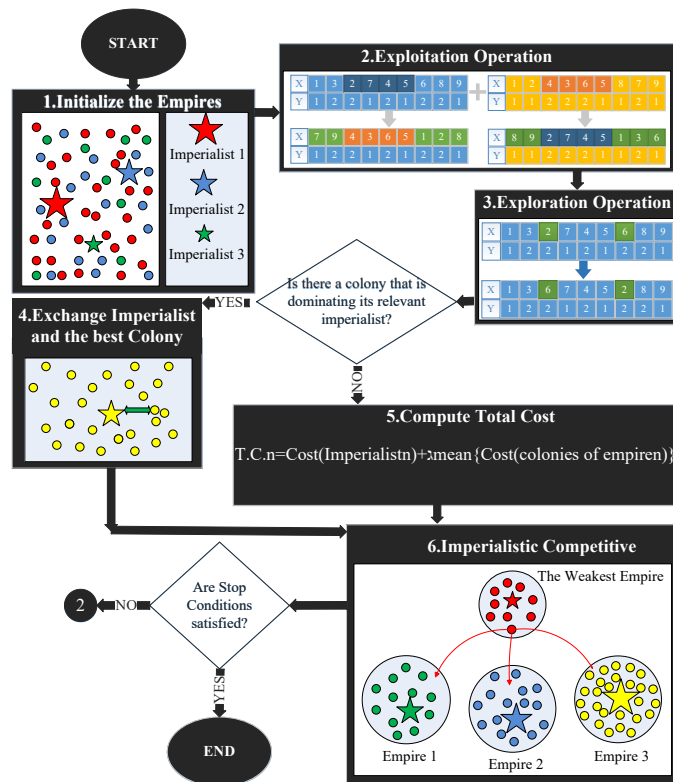


Figure 3.1 Flowchart MICGA

To make this combination, we have selected and combined the best parts of each method (ICA and GA) and proposed MICGA (PICA & GA). MICGA uses the following steps from ICA: generates countries and creates

empires and imperialistic competition. Then the crossover and mutation will be executed from GA. With this combination, we take advantage of ICA's efficiency in the convergence of global-optimum along with the fast exploration and exploitation operation of GA. We have tested the proposed method on task graph scheduling problem as another multi-objective optimization problem (in Paper VII and IV). The results show that we have a considerable improvement in accuracy and performance. Finally, we adopted the proposed hybrid method on task scheduling for real and complex applications [Paper VIII].

3.2 list of case studies

In this thesis, different types of case studies with various level of complexity have been used.

3.2.1 Synthetic benchmarks

Generally, several well-known benchmarks have been used to assess EC methods. In the Paper I and II, eight test functions (mathematical benchmark functions) along with three case studies have been used and presented in Table 2.1, in order to analyse and compare the proposed method with the sequential ICA and some other parallel EC methods.

Table 3.1 MATHEMATICAL BENCHMARKS

	Name	Equation	Min. Value	Bounds
f_1	G_1	$x \cdot \sin(x) + 1.1y \cdot \sin(2y)$	-18.5547	$0 < x, y < 10$
f_2	G_2	$0.5 + \frac{\sin\sqrt{x^2 + y^2} - 0.5}{1 + 0.1(x^2 + y^2)}$	-0.5231	$-\infty < x, y < \infty$
f_3	Sphere	$\sum_{i=1}^D x_i^2$	0	$-\infty < x_i < \infty$
f_4	Rosenbrock	$\sum_{i=1}^{n-1} 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2$	0	$x_i \in [-50, 50]$
f_5	Rastrigin	$\sum_{i=1}^n x_i^2 - 10 \cos(2\pi x_i) + 10$	0	$x_i \in [-5.2, 5.2]$
f_6	Akley	$20 + e - 20 \exp(-0.2 \sqrt{\frac{1}{D} \sum_{i=1}^D x_i^2} - \exp(\frac{1}{D} \sum_{i=1}^D \cos(2\pi x_i)))$	0	$x_i \in [-32, 32]$
f_7	Ellipse	$\sum_{i=1}^D 10^{4 \frac{i-1}{D-1}} x_i^2$	0	$x_i \in [-5, 5]$
f_8	Griewank	$\frac{1}{4000} \sum_{i=1}^n x_i^2 - \prod_{i=1}^n \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	0	$x_i \in [-600, 600]$

3.2.2 Nonlinear equations

Solving nonlinear equations are one of the NP-Hard problems [25], which match the categories of multi-objective optimization problems. Systems of nonlinear equations are used in a large range of engineering applications, such as petroleum geological prospecting, weather forecast, control fields and computational mechanics. In the classic methods, the quality of answers depends on the initial guess of the solution. The Newton-type methods are an example of classic methods. However, choosing suitable initial solutions for the systems of nonlinear equations is extremely complex and takes a lot of time and computations.

Therefore, to improve the performance several methods have been introduced for optimization problems. These methods can be categorized into two major categories: mathematical methods and evolutionary

computing (EC) methods where a large group of them are implemented in a serial mode and some of them are in a parallel mode.

Serial methods are more popular than other mathematical methods in nonlinear equations. However, the serial implementation does not always provide sufficient accuracy while parallel implementation can improve accuracy.

First, we explore some of the serial implementations of EAs. El-Emary and El-Kareem used Gauss-Legendre integration as a serial method to the system of nonlinear equations and used GA to find the results without turning the nonlinear equations to linear equations [26]. Mastorakis applied GA to solve a nonlinear equation as well as systems of nonlinear equations [27].

For solving a set of nonlinear equations, Li and Zeng [28] trained a neural-network algorithm. A simple gradient descent rule with variable step-size levels used to carry out the computation [28].

Huan-Tong et al. introduced an improved evolution strategy based on a probability ranking system to solve difficult nonlinear systems of equations problems [29]. ICA was implemented for solving nonlinear systems of equations by M. Abdollahi et al. [30]. The PSO method focuses more on "exploration", but the Nelder-Mead simplex system adjusts on "exploitation" [31]. Wu et al. used a new turn of the social-emotional optimisation method called MSEO, principally inspired by the Metropolis Rule [32]. In another try, M. Abdollahi et al. used a cuckoo optimization for solving nonlinear systems equations [33] [34].

Luo et al. introduced an efficient combination of the Newton-type method and Chaos search [35]. Grosan and Abraham used a new perspective of the EA [36], Mo et al. introduced a simple combination of the conjugate direction method (CD) [37], and M. Jaberipour used the standard implementation of PSO [38]. Pourjafari et al. [39], and Henderson et al. [40] proposed a methodology based on a polarization and novel optimization methods using Invasive Weed Optimization for finding all roots of a system of nonlinear equations.

Wu and Kang applied a parallel elite subspace EA for solving systems of nonlinear equations [12]. The initial guess can have an undeniable effect on the results of mathematical methods. On the other hand, a large and enough well-distributed population of the EAs can improve the convergence of them to the global optimum, which makes them slow. The EAs are ineffective for large-scale problems, like systems of nonlinear equations because of massive memory requirements and their high linear algebra costs. Based on all issues presented above, finding an efficient method for solving systems of nonlinear equations is necessary.

To solve nonlinear equations (in Paper III), we have utilized a coarse-grained method to parallelize ICA (PICA). Among parallelized techniques of EAs, the multi-population method has faster convergence and more reliable results than other parallel methods.

In the multi-population method, each processor has an independent population, each exploring its area of the search space. Their area can be separated or merged, and the ICA is run in each processor independently.

Each processor can use different values for parameters. Different rates of exploration and exploitation can help avoid converging to local optimums.

The migration operation in the multi-population method can significantly increase the performance of solving nonlinear equations so that it is the most critical parallel operation in the coarse-grain implementation. The processors share the best results together to discover better results in a shorter number of iterations. The proposed method selects the best imperialist to be migrated between processors, and the migrated imperialist replaces the worst colonies.

In this problem, each colony or imperialist is a possible solution for the nonlinear equation. Therefore, the proposed method creates the initial populations in each processor randomly, which are the possible solutions. Some of the countries are selected to be processed by our proposed method to converge them towards better results at each iteration.

The multi-population architecture has been made by connecting the processors as ring topology by message passing based communication. The low-cost communication and simplicity of the ring topology are the main reason that it has used in our implementation. The initial population happens for each processor independently. The initial population size is the same, and the serial ICA runs independently on them. After some iterations, the best imperialist migrates from each processor P_i to the next processor P_{i+1} in the ring and replaces the worst colony in P_{i+1} .

In the proposed method, growing the initial population size increases the selection pressure to converge to the global optimum faster. Therefore it is advantageous to increase the number of countries.

3.2.3 Task Scheduling

Over the last two decades, parallel processing in contemporary Multiprocessor System-on-Chips, or MPSoCs, in a wide variety of applications, is the result of many breakthroughs. Based on this development of embedded MPSoCs, many application domains such as video and audio processing, health monitoring, and autonomous vehicles developed to use in real-time environments. To improve the response time of these applications, the data and the computational tasks of these applications are distributed on all computational resources such as available multiple cores. By using an efficient task partitioning and scheduling strategies, the performance of such parallel systems can be improved.

In [41], the modified critical path algorithm is introduced, based on the latest possible start time of a task. A task's latest possible starting time is defined through the as-late-as-possible binding by crossing the task graph upward from the final tasks to the entry tasks while pulling the tasks' start times downwards as much as possible. The delayed possible start time of the task itself is followed by decreasing order of the latest possible start times of its follower tasks.

Moreover, the dominant sequence clustering algorithm (DSC) is performed in [41] that it works based on the dominant sequence, and at each step computation of the critical path of the partially scheduled task graph is necessary. At every track, DSC checks whether the greatest the critical path of task graph (CP) node that is a ready node. DSC assigned it to a processor

allowing the smallest start time. Such a smallest start time may be obtained by rescheduling some of the node's predecessors to the same processor. If the greatest CP task is not an immediate task, DSC does not choose it for scheduling. Instead, it keeps the highest task which lies on a path reaching the CP for scheduling. Furthermore, the mobility directed algorithm (MD) is presented in [41]. MD selects a task at each iteration based on relative movement which is determined as the difference between a task's earliest start time and latest start time. The earliest possible start time is allocated to each task through the as-soon-as-possible (ASAP) binding that is similar to the ALAP binding. This is executed by traversing the task graph downward from the entry nodes to the exit nodes while pulling the nodes upward as much as possible. Furthermore, relative mobility is achieved by distributing mobility with the task's computation cost. A coarse-grain implementation of the genetic algorithm method (MPGA) is introduced in [42] which outperforms nondeterministic and deterministic methods reported in [43], [44].

A new encoding method with a multi-functional chromosome is performed that uses a priority representation of chromosomes. This priority method is called priority-based multi-chromosome (PMC) [45]. In this method, GA uses to obtain near-optimal scheduling. Lei et al. proposed a GA mapping algorithm to minimize application execution time [46]. The target architecture is network on a chip (NoC) and graphs represent applications. Wu et al. also used genetic algorithms [47]. They combined a dynamic voltage scaling method with mapping and obtained 51% savings in energy consumption. Murali et al. used the tabu search (TS) algorithm to task mappings for several applications in NoC design [48]. Manolache et al. tried to guarantee packet latency by using task mapping in NoCs [49]. Hu et al. introduced a branch-and-bound algorithm to map a set of IP cores (IPs) onto a NoC with bandwidth reservation [50]. Their energy savings in the communication architecture is 51.7%. Marcon et al. compared several methods, using an architecture that characterizes applications by their inter-task communication volume. Xu et al. [51] used a multiple priority queues GA (MPQGA) for task scheduling problem on heterogeneous systems.

3.2.3.1 Task graph scheduling

In Paper VIII, we used MICGA (the combination of the GA and ICA) and customised this hybrid method for the scheduling problem. Operations of The GA primarily fit for scheduling, but as mentioned before, GA's convergence strategy to obtain more accurate and reliable results is not efficient enough [52]. Based on our previous knowledge in the ICA, The population contains countries as the chromosome.

The ICA improves the convergence operation quality by keeping all the countries available in all iterations. They may only relocate to other positions in the search space. On the other hand, the selection operation in the GA extremely depends on random functions which can quickly converge to a local optimum. Therefore, MICGA uses a more efficient convergence strategy (the convergence idea of ICA) to improve the reliability of results.

All the scheduled tasks in a directed acyclic graph (DAG) should satisfy the priority relations so that we applied an order based coding mechanism fitted for multiprocessor task scheduling. Therefore we had to fulfil the following rules:

Before starting the execution of the task, all the predecessors of the task must have completed their execution. Based on a DAG, all the tasks must be executed at least once.

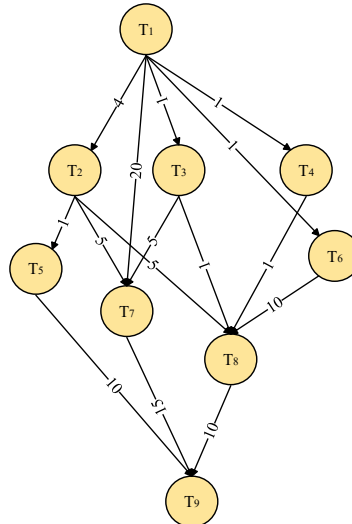


Figure 3.2 DAG Task graph

3.2.3.1.1 Order-based country (OBC)

The multiprocessor scheduling problem was solved by several methods [53], [54], [55] that they have used GA. Encoding a solution to the problem into a chromosome is a fundamental issue here.

Two critical angles that should be satisfied are 1) converting the problem from the genotype space to the phenotype space when chromosomes (countries) are decoded into solutions [56], and 2) metamorphosis properties when chromosomes are manipulated by GA operators [57]. Typically, the two common difficulties with respect to the encoding problem are the following:

- 1) Needs for storing a huge number of chromosomes (countries) corresponding to different suggested schedule.
- 2) Very complex operations (exploration and exploitation) in the case of a large number of tasks.

To succeed these difficulties we use the concept of an order based country (OBC) that strings the present task nodes in the DAG order of task nodes with the corresponding processors simultaneously. Figure 3.4 is a simple example that shows an OBC that represents nine tasks along with two processors for the DAG shown in Figure 3.3.

Priority	1	2	3	4	5	6	7	8	9	
X1	1	3	2	7	4	5	6	8	9	
Y1	1	2	2	1	2	1	3	2	1	
X2	1	2	3	4	6	7	5	8	9	
Y2	2	2	3	1	3	2	1	3	1	
P1-Chain	1	7	5	9	4	5	9			
	X1				X2					
P2-Chain	3	2	4	8	1	2	7			
	X1				X2					
P3-Chain	6	3	6	8						
	X1		X2							

Figure 3.3 Chain of tasks based on the presented country

3.2.3.2 Scheduling a pack of application

In Paper VII, we have improved the previous work in task scheduling for a pack (more than one) of jobs instead of only one job.

In normal task scheduling, some computational resources are useless and with the pack of applications, the maximum capacity of resources can be utilized.

Finding an encoding which can explain the priority constraint efficiently among the jobs is a significant step. The suggested encoding order is based on the priority of tasks, i.e. the pipelined execution representation of applications at the same time. Generally, in the pipeline model, there is a large data as input. This large data are divided into several small parts then these parts are then one by one used as inputs to an application. We make a pack of the parts and find the best schedule for this pack of data. Each part is an application such as Sobel Application (Figure 4.11), and the number of parts (applications) in each pack is. A proposed schedule of six Sobel applications (pp=6) on four processors is illustrated in Figure 3.5.

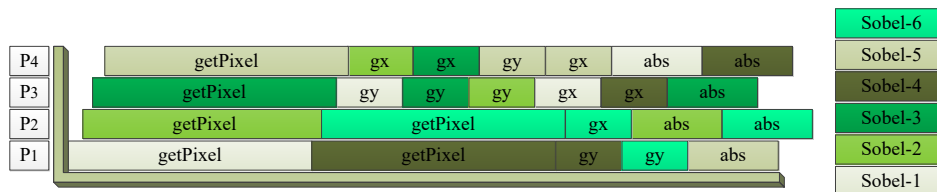


Figure 3.4 A proposed schedule of six Sobel application on four processors (pp=6)

3.2.3.3 Task scheduling with more constraints on industrial application

In Paper IV, an industrial application has been chosen to improve by a heuristic task graph scheduling. Industrial applications often require guaranteeing real-time execution, fault-tolerant implementations and providing reliable functionality. In general, it is impossible for a single machine architecture to satisfy all these needs. However, a distributed processing environment provides a variety of computational capabilities, which can be utilized to perform an application that has diverse execution requirements. An application job can be decomposed into subtasks. Subtask may have different computational requirements, where existing data

dependences is possible among these subtasks. For distributing subtasks, the following decisions should be made respectively:

- 1) Subtask matching, i.e. assigning subtasks to processing machines, and
- 2) subtasks scheduling, i.e. defining subtask execution order and the order of data transfers among machines.

The general goal of using distributed environments is to minimize the end-to-end cost of computation, i.e. minimizing the overall response time of the application, minimizing the number of processing machines, or both.

Performance of such parallel systems can be optimized by employing an efficient task matching and scheduling approach, however, the matching and scheduling problem is an NP-Complete [58]. Using exhaustive approaches for finding an optimal solution is time-consuming and is impossible in practice. Many heuristic task scheduling strategies have been proposed [56], [59] to find a near-optimal solution in a reasonable amount of time.

To only focus on the matching and scheduling problem, we made the following assumptions. First of all, we assumed that each subtask is written in a machine-independent language. Moreover, it is assumed that an application job is decomposed into multiple subtasks and we know all the data dependencies among subtasks before the execution. The load complexity of all the subtasks and their execution time on each machine is known a priori. It is assumed that for each subtask, there is a couple of input nodes that produce raw input data (sensors) and there are some output nodes which consume subtasks processing results (Actuator). Obviously the input of subtask is coming from sensors or the output of other subtasks and similarly, the output results of each subtask will be consumed by actuators or other subtasks. The distributed processing platform is non-uniform that consists of multiple homogeneous machines with various processing potential. All the processes on machines are non-primitive meaning each machine completes the current task before calling of the next task.

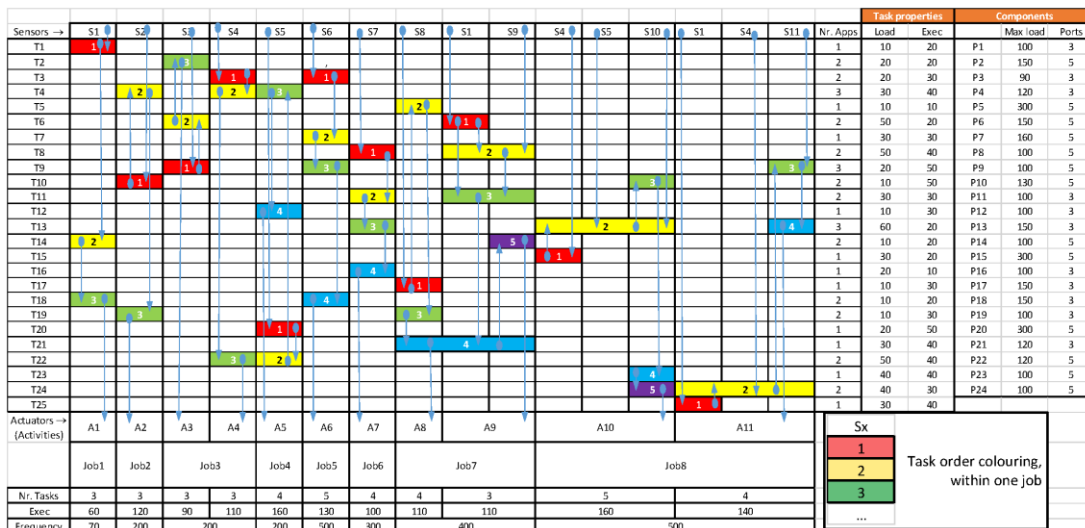


Figure 3.5 Representing the use case including jobs, intra-task dependencies, tasks load complexity, real-time deadlines and processing unit specifications

Also, all input data items of a subtask must be received before its execution can begin, and none of its output data items is available until the execution of this subtask is finished. A data conditional is based on input data, it is assumed to be contained inside a subtask. A loop that uses an input data item to determine one or both of its bounds is also assumed to be contained inside a subtask. When two communicating tasks are mapped onto the same processor we assume that the communication delay is zero. However, when they are mapped onto different processors a finite communication delay is assumed and modelled. The task is represented by a DAG, where each edge goes from a producer to a consumer. Figure 3.7 shows a DAG of an application job, seventh job, from the industrial use case (Figure 3.6).

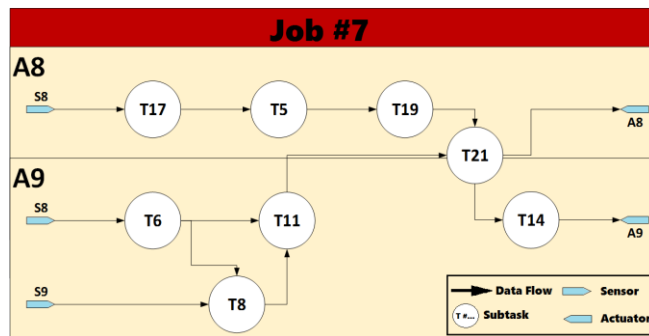


Figure 3.6 Dataflow of Job #7

3.2.4 Placement of Swarm of drones

Wireless sensor networks (WSN) and cyber-physical systems (CPS) are two critical current fields of technology that are tightly intertwined [60], [61]. The combination of WSNs and other modern technologies, such as unmanned aerial vehicles (UAV) and mobile robots, has created a novel revolution in this area. An efficient combination of WSNs with mobile nodes can obtain better performance by using mobile access points embedded in UAVs or mobile robots [62], [63].

Smart mobile access points (SMAPs) can make a cluster together for distributed computing and make decisions for improving the quality of the network. SMAPs change their positions their decisions based on the current situation in the environment. Therefore, placement of SMAPs is a critical problem in this concept.

The Placement problem is a multi-objective optimization problem. There are different kinds of methods such as static or dynamic methods used to solve this problem [64]. The general method to solve the initial placement, a static placement problem, is the evolutionary algorithm [9], [65] such as PSO.

SMAPs are mobile access points that are able to create a dynamic and smart sensor network. The main task of SMAPs is to monitor and predict the behaviour of the network and decide the best method to maintain the network at any given time. SMAPs collect and broadcast signals such as battery levels, and help requests, from and to the other network nodes. Based on the collected information, SMAPs predict the next actions based on this information and run these actions.

Normally, a SMAP uses previous information and operations or generates a new solution by learning or evolving through machine learning algorithms. The most major tasks of SMAPs are the following:

1) Compute new (near-optimal) locations for access points to obtain greater coverage of sensors.

2) Making decisions about how the access point must move to new areas [66]

3) Participating in distributed communication and computation tasks (fog computing [67]) other than the processing carried out as an inherent part of the decision making in 1) and 2).

A SMAP can move to a new position based on different situations, such as reconfiguration of the network, supporting specific tasks (e.g. tolerating faults by replacing a faulty access point with a functioning one) or covering a missed area of sensors that are not covered currently by any other access point. SMAPs increase quality and flexibility of WSNs with the following advantages:

- Improving the system by increasing the coverage in large scale networks.
- Solving hotspot problems in the network.
- Handling dynamic reconfiguration of WSNs.
- Replacing faulty access points.
- Forming a grid network to run real-time tasks in parallel.

Therefore, an optimised placement of SMAPs has a significant effect on the ability and performance of the network. Therefore, we focused on the initial placement to illustrate the efficiency of our method. Also, SMAPs should have the best initial placement and clustering [68], [69]. Additionally, they should have the ability to move dynamically to new locations, because the behaviour of the network can be changed at run-time. This network reconfiguration method is a response to the current status of the network, and SMAPs should update their locations for the new configuration. Our method is based on the following:

Each access point or cluster centre always has a distinct weight. These weights are equal to the probability of the request for reconfiguration. The Weights can be calculated based on various effective parameters, such as the current traffic of network at these access points, properties of the covered area or the lifetime of their batteries.

In a dynamic WSN, mobile nodes can be repositioned to new positions e.g. a sensor can change its location to the new location to collect extra information from the environment, or it can be replaced with another sensor that has a different performance on sensing physical phenomena.

Dynamic replacement of nodes during network operation is very complex. It is clear that there is a significant difference between static and dynamic placement. In the dynamic replacement, more complex parameters such as the environment base, mobile targets, and lifetime of sensors can have an effect on the placement problem. Furthermore, the dynamic placement of a node needs very accurate handling because it can probably cause a disruption in data connectivity and delivery.

Also, sensors can migrate to new locations to obtain better readings or to sense extra information. Therefore, dynamic placement is a really challenging problem particularly when the following two aspects are also taking into consideration:

- 1) The ability of nodes to migrate and reposition to an infinitive number of positions.
- 2) The responsibility of the network to keep full connectivity of covering all sensors.

Placement in WSNs can be categorised to two main groups: placement of sensors and placement of access points that in this case study, we only focused on the placement of access points.

The Placement problem of access points is a complex multi-objective optimization problem that all goals must be satisfied. The two more critical goals are:

- 1) To guarantee continuous network connectivity. In fact, at least each sensor and a node should have a connection to one access point that is connected to the network at any given time.
- 2) Implementing a new reliable communication in the case of access point failures, meaning that other nodes in the network should find a new configuration to keep their communication.

The best solution to achieve this goal is to use extra (redundant) access points which increases the number of feasible access points for sensors in the network.

Heterogeneous WSNs are more popular and they have three different kinds of nodes: sensors, access points, and gateways.

Sensors communicate with their corresponding access points. Access points can communicate with all other types of nodes. Additionally, access points have extra memory and computing capacity than sensors.

Finding a near-optimal placement to improve reliability, connectivity, and energy consumption is an NP-hard problem [70]. Different heuristic methods used to solve this problem, which these methods are presented in [66], [71], [72].

Resolving the problem of failing access points is necessary for real large-scale WSNs. Also, it is apparent that using regular methods to achieve more redundancy (i.e. adding fixed/stationary access points) is very costly. SMAP can be used to solve this problem at a lower cost and with more flexibility.

As mentioned before, the placement of SMAPs is a complex problem. During our research, we have worked in this area and test our methods to find a near optimal placement for SMAPs. At first in Paper V, we have proposed an efficient optimization method to solve the placement problem. Then in the next publication (Paper VI), we introduced a hierarchical structure to improve the scalability of SMAPs. To address the scalability of SMAPs, we proposed a complex coding to have the ability to optimize a multi-layers placement.

3.2.4.1 Placement of SMAPs

In the first method [Paper V], a multi-population implementation of ICA is performed to improve the efficiency of solving the initial placement problem in SMAP based systems. We have categorized our method into two critical layers: an architectural layer and an algorithmic layer. The architectural layer is more related to the structure of SMAPs and there is not any relation to the optimization concept and only describing the algorithm part.

In the proposed method, all access points are categorized into two main categories, SMAPs and normal access points, to have the best placement of SMAPs. Normal access points are static and they only transfer data and control connectivity in the network.

SMAPs have an extra ability compared to normal access points. They can move around the network area to improve the total performance of the network. Normal access points (static access points) have been selected as objects that should be covered by SMAPs. Each static access point has its weight and this weight indicates the probability of requiring support from SMAPs. The weight can be derived from various parameters such as battery lifetime and communication traffic. To simplify the work, the rate of communication traffic is picked for this purpose. More details about the modelling of the problem are available on Paper V. The PICA has been used to solve this problem.

Like PICA, processors are connected in a ring topology and we use message passing to implement this idea.

First, independent populations are generated in each processor by running the sequential ICA independently. After each migration gap (chapter 1.3.1) (some certain iterations) the best imperialist moves (migrates) from the processor P_i to P_{i+1} in the ring topology and replaces the worse colony in the weakest empire in P_{i+1} . This operation has been run synchronously. Different migration strategies can provide different results.

When a sparse connection topology, like a ring topology, is used low migration rates and short migration gap can work better. In turn, in a fully connected topology such as the star topology, the best optimal solution are obtained when the migration rates are higher. A processor does not execute any other operations during the migration operation.

MICA is an efficient parallel method which improves the execution time, stability, and reliability of results (Paper I). This is sufficient motivation to apply PICA to the placement of SMAPs. In order to appropriately modify the method, each SMAP is considered a processor, and a ring topology is assumed.

3.2.4.2 Hierarchical Placement of SMAPs

In Paper VI, we have presented a hierarchical placement structure for SMAPs to satisfy the scalability condition of them. In the proposed method, a parallel implementation of GA (MPGA) is implemented to improve an efficient method for solving the initial placement problem in hierarchical SMAPs based systems.

Here again, the focus is on the algorithmic layer rather than the architectural layer. To simplify the placement problem, all access points are divided into two types which are normal access points and SMAPs. Normal access points and SMAPs are the same as a previous case study in terms of their ability (static and dynamic), but both of them transfer data and maintain connectivity in the network (Paper V).

Normal access points have been chosen as objective points to cover by SMAPs in the lowest layer. In each layer in the hierarchical method, SMAPs are distributed into several clusters, and each cluster is chosen as objective points (same as normal access points for the first layer) for other SMAPs in the higher level.

We let that SMAPs works on a 3-D mesh and they can select a position from this mesh. Also, our method should work on a discrete space; therefore, a GA is a fit option to solve the placement problem [9], (Paper I), (Paper V). Since, The multi-population GA is more efficient than a serial GA for a complex problem, we have utilized it to achieve the best possible results for choosing the best positions.

The critical points of MPGAs, regarding their practical usage in this problem, are 1) Making the better diversity of the initial population, 2) Increasing the selection pressure and 3) Migration operator [9], (Paper V).

Chapter 4

Experimental Results

In this chapter, the results of thesis contributions are presented. Only some of the results from the original papers have been selected, and they are presented based on the order of the published papers. The results are based on convergence stability diagrams (show the convergence and reliability of our methods), the statistical results (shows the accuracy of our methods) and performance.

We used the multi-population approach to parallelise our methods via the message passing interface (MPI) [73] using MPICH2 [74] which allows distributing the algorithm on several processors which are connected in a ring topology. The proposed methods have been tested on an Intel Core i5-45705 desktop computer clocked at 2.90 GHz (64-bit) with 24GB of memory. The implementation parameters are presented in Table 4.1.

Table 4.1: parameters of implementation

Parameters	Values
Number of Countries	100
Number of Empires	5
Termination Condition	20 Iterations
Number of Processors	5
Exploitation Rate	0.8
Exploration Rate	0.3
Migration Rate	1 Chromosome

In the rest of this chapter, the results of benchmarks and case studies are presented.

4.1 Synthetic benchmarks

In Papers I and II, we introduced two parallel methods of ICA (PICA and Master-Slave ICA) implemented by MPI. We assessed them on eight mathematical benchmarks presented in Table 2.1. The obtained results are compared with relevant state-of-the-art methods such as the sequential ICA, PABC, Coarse Grain parallel PSO, Multi-Population Genetic Algorithm, Dynamic Neighborhood Structures in Parallel Evolution Strategies (Neighborhood GA), cuckoo optimization algorithm (COA), course-grained PSO (C-PPSO) [30] and PSO-TM.

The results have been assessed based on our four principal parameters which are speed up, stability, accuracy and ability to apply on different areas. Regarding the performance (execution time) the results shown in Figures 4.1, 4.2, 4.3 and Table 4.2, reflect that both of our methods are faster

but PICA is superior to others. Table 4.2 shows the speedup and efficiency of PICA and C-PPSO. C-PPSO has been selected as the best-related works, and the results illustrate that PICA is three times faster and more efficient than C-PPSO.

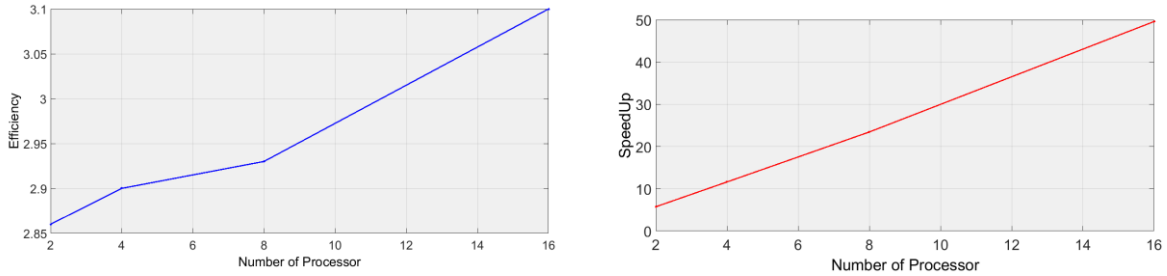


Figure 4.1 Speed up diagram and parallel Efficiency diagram for f_3

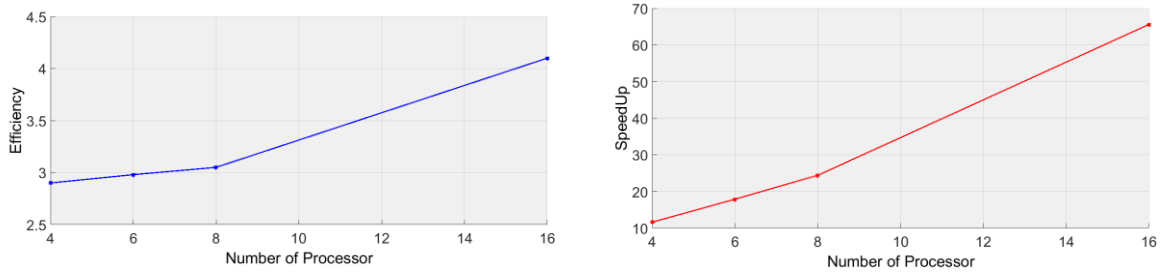


Figure 4.2 Speed up diagram and parallel Efficiency diagram for f_4

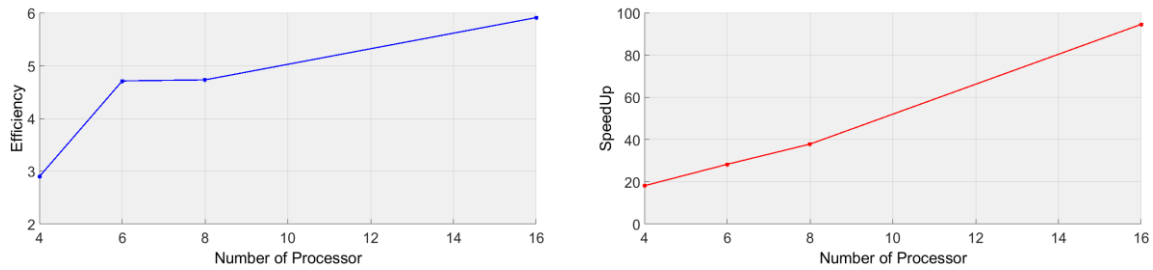


Figure 4.3 Speed up diagram and parallel Efficiency diagram for f_5

Table 4.2 Values of Speedup and Efficiency on Some Benchmarks by PICAs and C-PPSO

	Multi-Population PICA						C-PPSO					
	CPU=4		CPU=6		CPU=8		CPU=4		CPU=6		CPU=8	
	Speed up	Efficiency	Speed up	Efficiency	Speed up	Efficiency	Speed up	Efficiency	Speed up	Efficiency	Speed up	Efficiency
Sphere	12.2	3.05	18.6	3.10	25.12	3.14	3.9029	0.9757	5.7555	0.9593	6.7519	0.8440
Rosenbrock	11.6	2.90	17.88	2.98	24.4	3.05	3.9572	0.9893	5.7793	0.9632	5.9724	0.7465
Rastrigin	18.1	4.52	28.26	4.71	37.84	4.73	3.9580	0.9895	5.7774	0.9629	7.7703	0.9713
Griewank	16.3	4.07	25.32	4.22	34.08	4.26	3.9114	0.9778	5.9128	0.9855	7.3851	0.9231

The stability diagrams and statistical results confirm more reliable results are obtained from different runs. The stability diagram shows the results of a method in different runs and the method is more accurate if the results have a lower standard deviation. As an example, the stability diagram of PICA on Akley as a most complex benchmark in Table 2.1 is illustrated in Figure 4.4, and statistical results are presented in Table 4.3 and Table 4.4. PABC has been selected as the most reliable method among others. Table 4.4 presents values of different statistical results. Values of standard division (SD) shows the variation or dispersion of final results in different runs. Based on the presented values, the SD values of PABC are at least two times larger than the SD values of PICA, which shows that PICA is more reliable than other related methods.

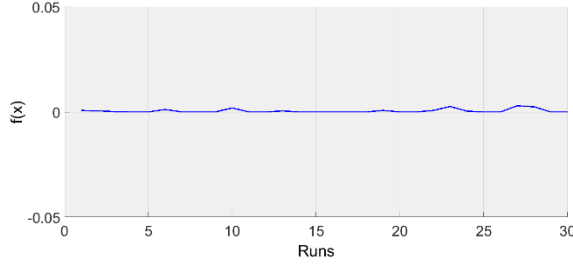


Figure 4.4 Stability diagrams of f_8 with PICA

Table 4.3 Statistical table of multi-population PICA on f_3, f_4, f_5, f_6, f_7 and f_8 with PICA with 30 runs

	Mean	SD	STE	Median	Worse	Best
f_3	0.0018104333333333	0.002970555267766	5.423467094903584e-004	0	0.0018104333333333	0
f_4	0.0070844333333333	0.009400134634705	0.001716221927671	0.00158750	0.028145000000000	0
f_5	0.0015893666666667	0.002425782099026	4.28851917428874e-004	0	0.007724000000000	0
f_6	8.014333333333334e-004	0.001179387178196	2.153256538433475e-004	0	0.003326000000000	0
f_7	0.006058700000000	0.008992364052446	0.001641773545608	0	0.028145000000000	0
f_8	4.893666666666666e-004	8.566241876403997e-004	1.563974636250617e-004	0	0.002942000000000	0

Table 4.4 Results obtained for the PICA and PABC algorithms on some benchmark functions.

	f	D	MCN	PICA					
				Master-slave		Multi-Population			
				P=4		P=4		P=16	
				Mean	SD	Mean	SD	Mean	SD
PICA	f_3	30	2000	2.51354E-16	3.66295E-17	1.64297E-16	2.28976E-17	1.52341E-16	2.18497E-17
	f_4	30	2000	2.322891E-02	3.462278E-02	1.876391E-02	2.964761E-02	4.823789E-03	5.213874E-03
	f_5	30	2000	1.993628E-16	4.736478E-17	1.862283E-16	4.732892E-17	1.722862E-16	4.378971E-17
	f_8	30	2000	4.728617E-18	6.598321E-19	3.927344E-18	5.668102E-19	3.367451E-18	4.962713E-19
PABC [15]	f_3	30	2000	-	-	2.49479E-16	4.043068E-17	2.467389E-16	4.100729E-17
	f_4	30	2000	-	-	2.182352E-02	3.250047E-02	2.282869E-02	2.585128E-02
	f_5	30	2000	-	-	1.946071E-16	4.615336E-17	1.931904E-16	5.386725E-17
	f_8	30	2000	-	-	4.896980E-18	7.036649E-19	4.756034E-18	6.995602E-19

4.2 Nonlinear equations

To examine the accuracy of PICA, we utilized three nonlinear equations. The results show that PICA obtained the most accurate results in the continuous search spaces problems.

In the first case study (nonlinear equation), the results compared with [38] and [75] have been obtained with 120 generations with an unknown number of population sizes. In [30], the parameters of ICA have been set to 50 iterations with 250 countries. As shown in Table 4.5, PICA achieved better and more accurate results than the previous works. The accuracy of PICA is more than e^{-32} instead of e^{-30} that obtained by serial ICA and C-

PPSO. The related convergence and stability diagram have been presented in Figures 4.5, and 4.6, respectively; and statistical results are presented in Table 4.6.

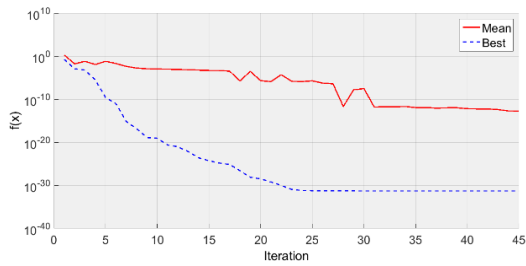


Figure 4.5 The convergence history of case 1 with PICA

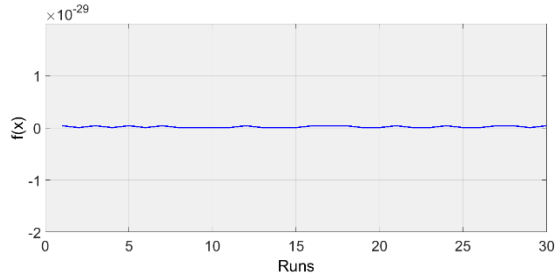


Figure 4.6 The stability chart of case 1 with PIC

Table 4.5 Comparison Results of PICA for Case 1 with [38], [75], [33] and [30]

Methods	x_1	x_2	$f(x)$
PPSO [38] and Gyurhan [75]	-0.29051455550725	1.08421508149135	4.686326815078573e-029
PPSO [38] and Gyurhan [75]	-0.793700525984100	-0.793700525984100	1.577721810442024e-030
COA [33]	1.08421508149135	-0.29051455550725	4.686326815078573e-029
COA [33]	-0.29051455550725	1.08421508149135	4.686326815078573e-029
ICA [30]	1.084215081491351	-0.290514555507251	3.562200025138631e-030
ICA [30]	-0.793700525984100	-0.793700525984100	1.577721810442024e-030
ICA [30]	-0.290514555507251	1.084215081491351	3.562200025138631e-030
PICA (present study)	1.0842150814913511	-0.2905145555072514	4.9303806576313238e-032
PICA (present study)	-0.79370052598409995582	-0.79370052598409995582	3.9443045261050590e-031
PICA (present study)	-0.2905145555072514	1.0842150814913511	4.9303806576313238e-032

Table 4.6 Statistical results of PICA

Problem	N	Mean	Std. Deviation	Std. Error Mean	Worst	Best
Case 1	30	1.988586000000001e-031	1.739458967563944e-031	3.175803047964731e-032	3.944300000000000e-031	4.930400000000000e-032
Case 2	30	1.046312443884771e-026	1.511543708264576e-026	2.759688618905053e-027	4.414500000000001e-026	0.0
Case 3	30	6.898049999999997e-037	1.150107106181705e-037	2.099798685342363e-038	9.039099999999999e-037	5.800000000000000e-037

In case 2, the best results in [77], [76], [30], and [33] with 50 iterations and 250 population sizes were compared with PICA (Paper III). PICA obtained these results with 250 countries and 35 decades. Also, the convergence and stability diagram are illustrated in Figures 4.7 and 4.8.

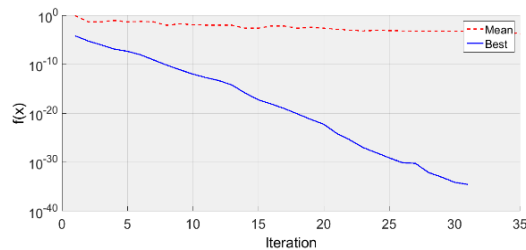


Figure 4.7 The convergence history of case 2 with PICA

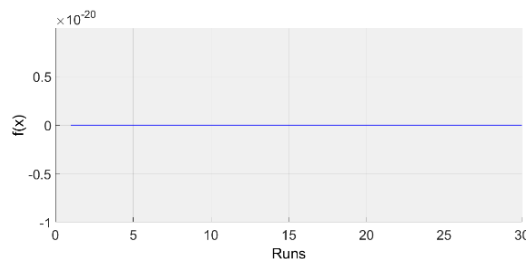


Figure 4.8 The stability chart of case 2 with PICA

Case 3 has been solved by the filled function method in [77] and has been proposed as a problem in [76] and [30]. The presented methods in [77], [76], and [30] have been tested for 1000 iterations, and their population size has been set to 300. The results of the PICA are illustrated in Table 4.6.

The convergence diagram of PICA has been presented in Figures 4.9. Figure 4.10 shows the stability diagram of PICA for Case 3. The speedup and efficiency results of all three cases have been listed in Table 4.7. Achieving the super-linear performance demonstrates that PICA has an excellent performance gain for continuous problems.

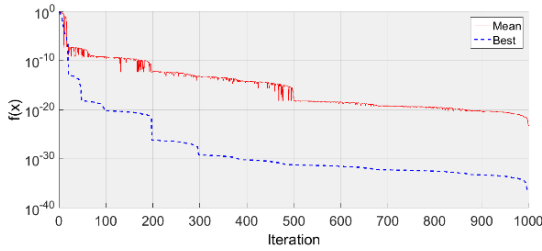


Figure 4.9 The convergence history of case 3 with PICA

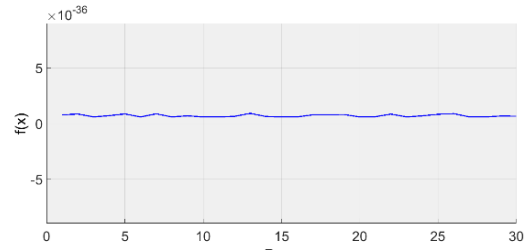


Figure 4.10 The stability chart of case 3 with PICA

Table 4.7 The Comparison Statistical Results of Serial ICA, and PICA

Problem	Speed Up	Efficiency	Serial ICA time	PICA time	#processors	Super linear performance?
Case 1	2.82	1.41	0.0341	0.012	2	Yes
Case 2	5.1	2.55	2.1	0.411	2	Yes
Case 3	6.24	3.12	6.78	1.08	2	Yes

4.3 Task Graph Scheduling

In Paper VII and VIII, four well-known real applications, Sobel, SUSAN, RASTA-PLP and JPEG encoder [78] (Figure 4.11) have been used to exhibit the performance of MICGA. The collected results have been compared with those of the other EC methods that have used these applications in their evaluations such as MPGA [42], PMC [45] and MPQGA [51].

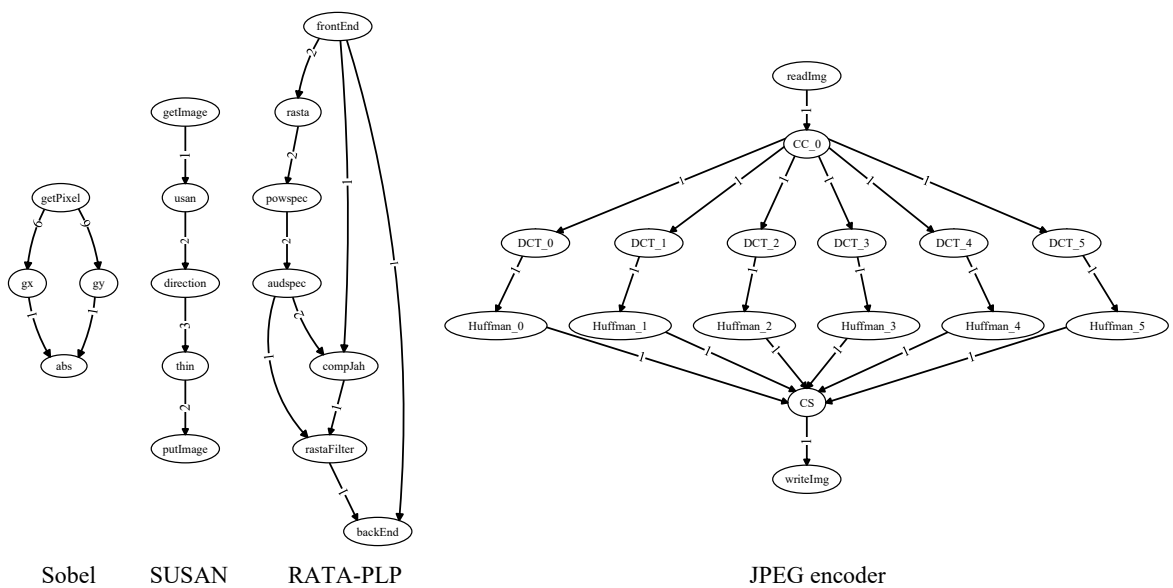


Figure 4.11 Task graph scheduling benchmarks

First, we present the results of MICGA reported in the paper VIII (NoMeS). MICGA is named NoMeS in the paper. Figure 4.12, Figure 4.13, Figure 4.14 and Figure 4.15 show the end to end execution times of the benchmarks for all optimization methods with various numbers of processors. Based on the results, MICGA decreases end to end execution time more than 29.5%, 68.1%, 47.4%, and 10.1% for the Sobel, SUSAN, RASTA-PLP, and JPEG encoder applications, respectively.

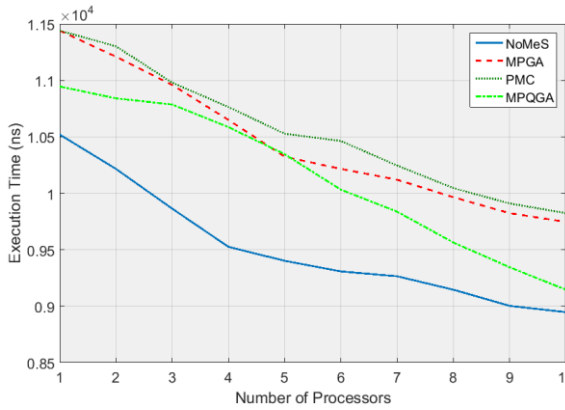


Figure 4.12 The Execution Time of Sobel filter with different numbers of processors

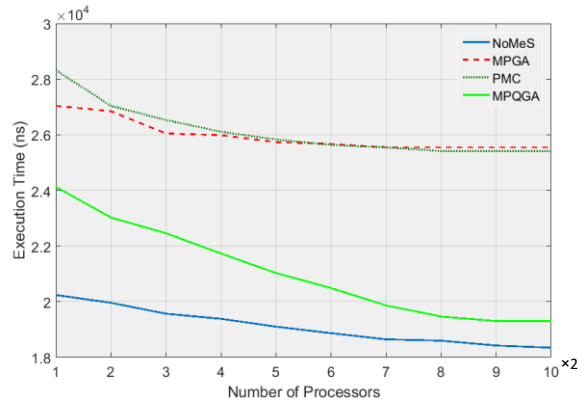


Figure 4.14 The Execution Time of RASTA-PLP filter with different numbers of processors

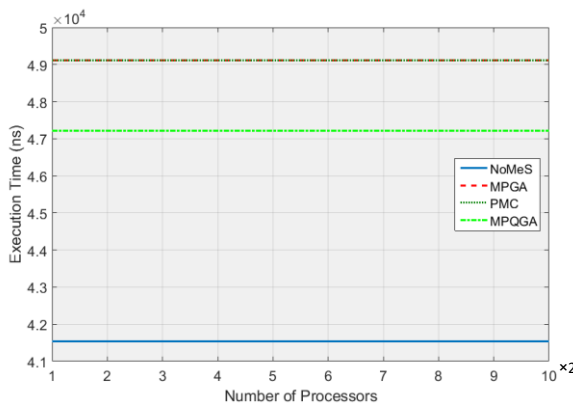


Figure 4.13 The Execution Time of Susan filter with different numbers of processors

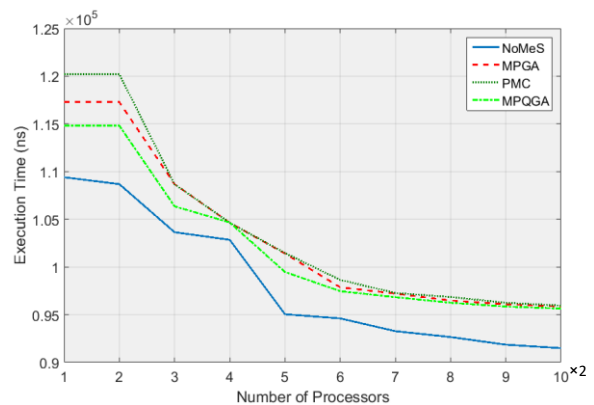


Figure 4.15 The Execution Time of JPEG-encoder with different numbers of processors

Another critical factor to select the best method is the stability and reliability of the experimental results. The heuristic and metaheuristic methods cannot converge to the best results in all execution runs. Therefore, each method is executed several times and their results in all iterations show the stability. Figure 4.16 illustrate the stability diagram of each method. The results confirm that our method is more reliable with fewer errors values than the others. Figure 4.17 illustrate the convergence diagrams of MICGA on all applications. Based on the quality and accuracy of the results, MICGA is a prominent candidate for task graph scheduling problems.

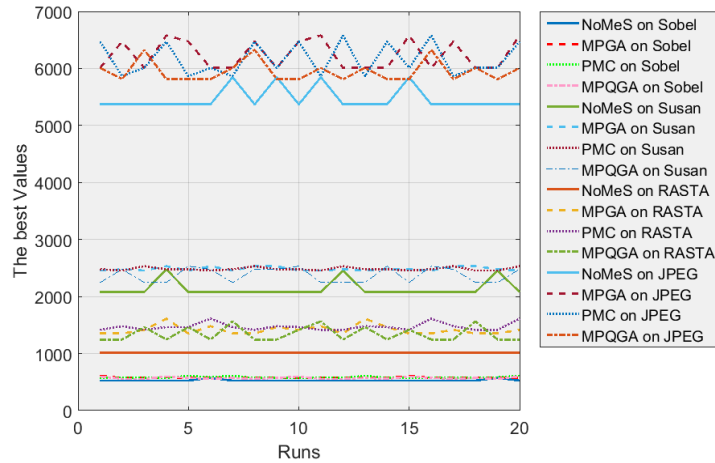


Figure 4.16 The stability diagram of MICGA, MPGA, PMC and MPQGA on Sobel, SUSAN, RASTA-PLP and JPEG encoder with 2 processors

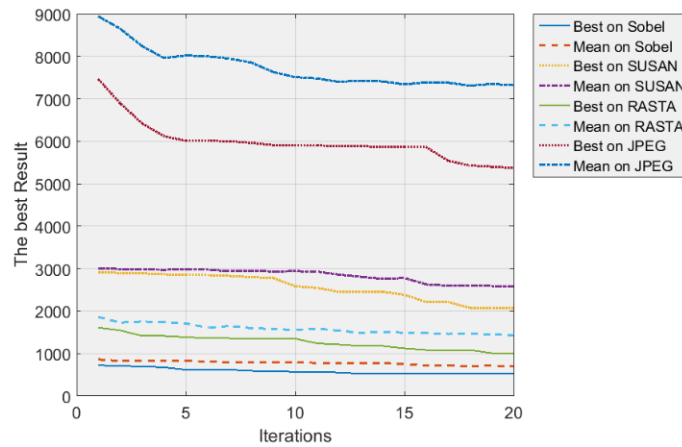


Figure 4.17 The convergence diagram of MICGA on Sobel, SUSAN, RASTA-PLP and JPEG encoder with 2 processors.

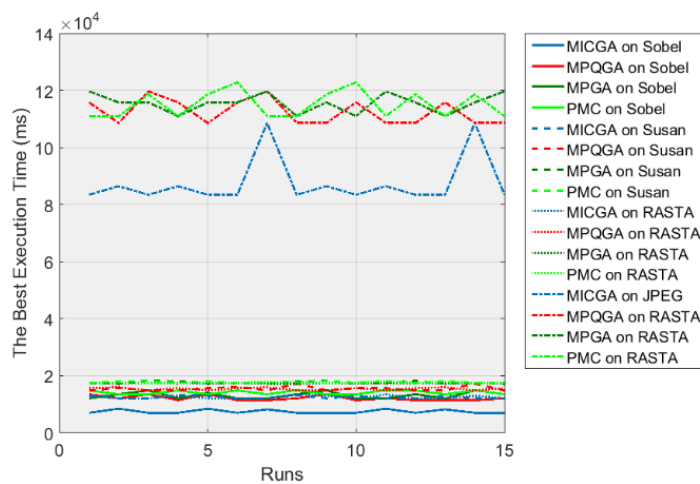


Figure 4.18 The stability diagram of MICGA, MPGA, PMC and MPQGA on Sobel, SUSAN, RASTA-PLP and JPEG encoder with 4 processors

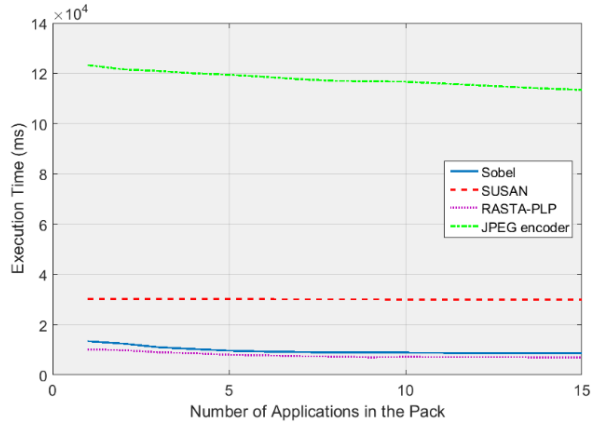


Figure 4.19 The execution time of MICGA on all benchmarks with different number of pp on six processors

Here, we report the results of the paper VII related to the idea of a pack of tasks. The results obtained using the same applications to show the stability of each method in Figure 4.18. MICGA is compared with MPGA [42], PMC [45] and MPQGA [51] in the same conditions and other methods.

Figure 4.20, Figure 4.21, Figure 4.22 and Figure 4.23 illustrate the end to end execution time of all method on the applications on six processors. MICGA decreases the execution time by more than 46%, 8%, 15%, and 18% for the Sobel, SUSAN, RASTA-PLP, and JPEG encoder applications, respectively. It seems that the used packing method decreases the execution time.

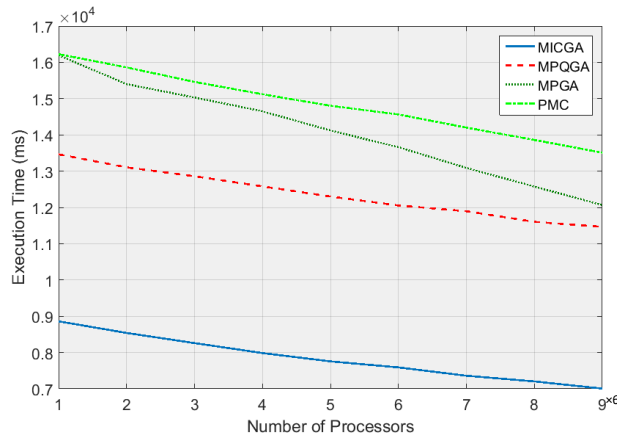


Figure 4.20 The execution time of all methods on 100 of Sobel application (pp=100)

One of the main limitations of evolutionary algorithms is that they decrease the convergence speed by increasing the number of iterations leading to non-convergent results in low iterations. In the Paper IV, Figure 4.24 and Figure 4.27 represent the convergence of fitness functions for both single and multi-objective optimization, respectively. It can be easily observed from the convergence figures that both strategies are highly convergent toward the optimal results by a contentious reduction in fitness functions (see Equation (1) and Equation (2)).

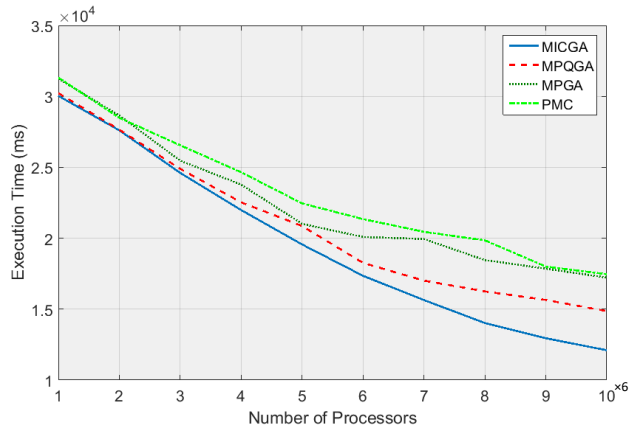


Figure 4.21 The execution time of all methods on 100 of SUSAN application (pp=100)

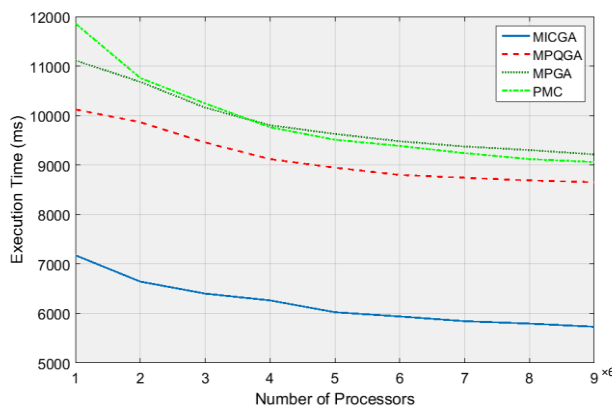


Figure 4.22 The execution time of all methods on 100 of RASTA-PLP application (pp=100)

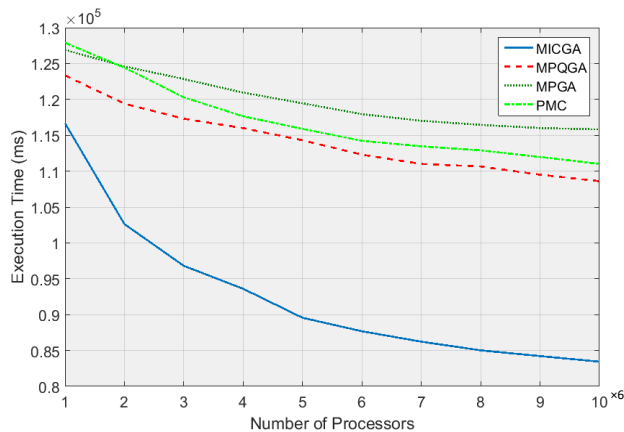


Figure 4.23 The execution time of all methods on 100 of JPEG-encoder application (pp=100)

4.3.1 Single Objective Optimization

Figure 4.25 illustrates the variation trend of the total number of utilized processing units by increasing the number of iterations. As mentioned before, the aim of single objective optimization is to decrease the number of processing units used in jobs scheduling. Figure 4.26 shows considerable improvement in finding scheduling with lower required processing units. According to the results of Figure 4.25, we need 22 processing units for scheduling in the first iteration, while by proceeding

the exploration algorithm; we found a solution with only seven required processing units. Although there exist some breaks in continuous improvement, the overall trend moves toward improvement.

4.3.2 Multi-Objective Optimization

As mentioned earlier, the total number of processing units and end-to-end run-time for all the jobs are the two main objectives of MPGA. Figure 4.27 and Figure 4.28 illustrate the convergence figures of required processing units for scheduling and end-to-end run-time for all the scheduled jobs, respectively. We can conclude from the figures that both the objectives are approaching toward optimized results. Although there are some failures or stops in achieving better results in each iteration, the overall progression of MPGA always approaches toward superior outcomes.

Table II shows three different solutions on the Pareto frontier of the last Population. We have a variety of options based on user needs. Solution 1 is scheduling with a minimized number of processing units (7 processing units) while takes more time, 210tu, for running. On the other hand, Solution 3 provides the minimum elapsed end-to-end run-time (160tu), while needs 9 processing units for running.

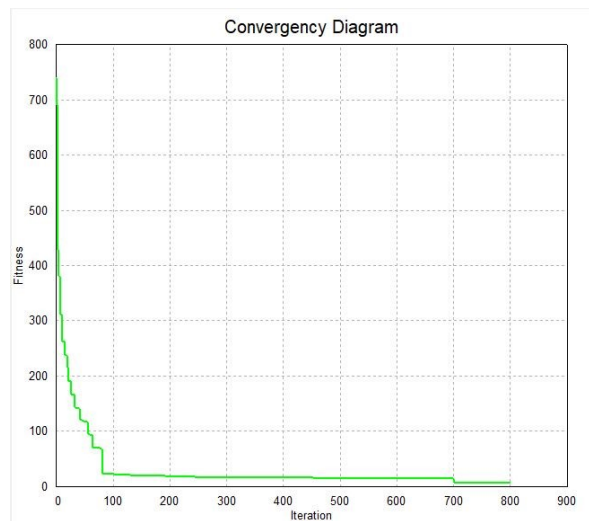


Figure 4.24 Convergence diagram of single objective optimization (# Processing units).

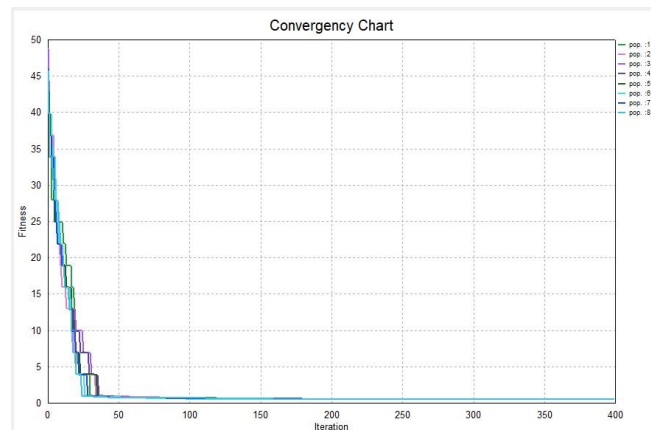


Figure 4.25 Convergence diagram of MPGA (# processing units, end-to-end runtime).

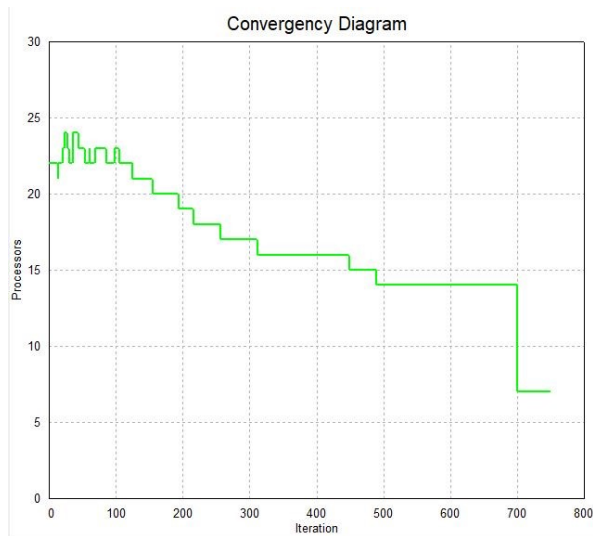


Figure 4.26 Convergence diagram of the variations # processing units based on in single objective optimization.

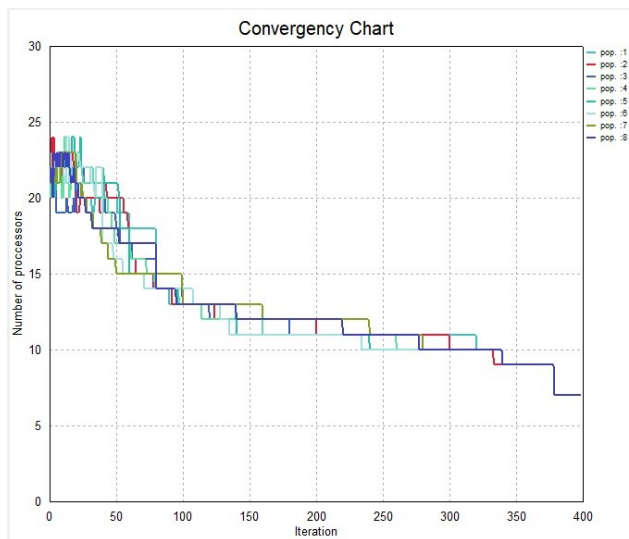


Figure 4.27 Convergence diagram of # processing units in multi-objective optimization by using MPGA approach.

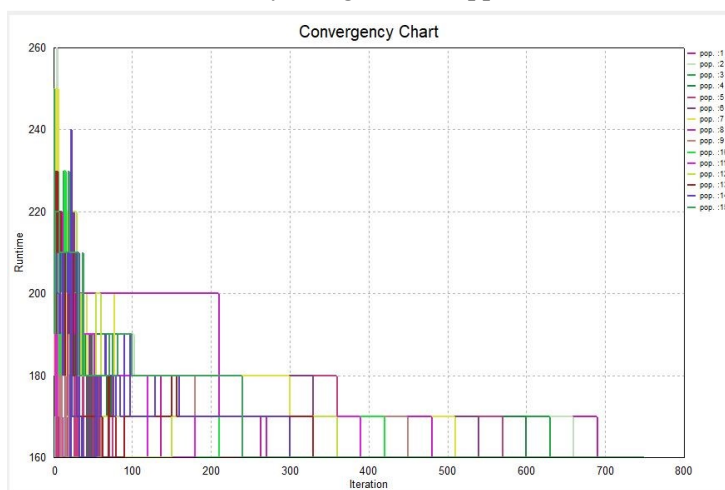


Figure 4.28 Convergence diagram of end-to-end run-time in multi-objective optimization by using MPGA approach.

Figure 4.9 presents the allocation and scheduling results of the use case (Figure 3.6) described in Subchapter 3.2.3.3. The results of optimizing the use-case with MPGA are shown in Figure 4.9 valid scheduling for all jobs and their related tasks with the minimum number of processing units (Solution1).

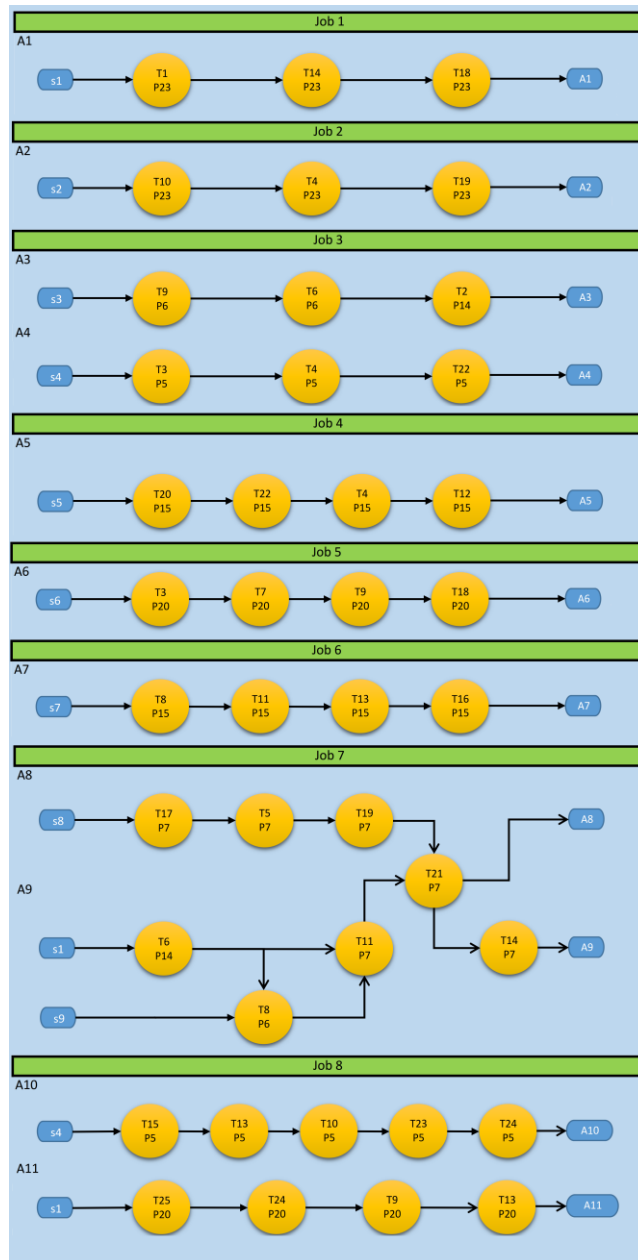


Figure 4.29 The best solution for multi-objective optimization with a minimum number of processing units (solution1)

4.4 Placement of Swarm of drones

Placement of SMAPs, described in Subchapter 3.2.4 has been implemented in two different works (Paper V and Paper VI). Paper V presents an efficient placement of SMAPs while Paper VI proposed a hierarchical placement of SMAPs.

In Paper V, we used PICA for finding the optimal placement using four well-known 2-D benchmarks and all the collected results are compared with the serial ICA, deterministic and random method [70], for the placement problem.

4.4.1 Single SMAP

The initial placement in the first three benchmarks is coordinated at (0, 0). The first three benchmarks are used to find the best placement for only one SMAP. But, the fourth benchmark is considered for more than one SMAP. All the methods have been run for 40 times on each benchmark,

In the first benchmark, eight static access points have been placed around the SMAP in a circular manner. The radius of the circle is 500 meters. The second benchmark has eight static access points that are symmetric based on the center point.

In the third benchmark, there are 50 access points that are randomly placed around the SMAP. The weights of all static access points are equal in all benchmarks. PICA and ICA have been run in 100 iterations and their population size are equal to 500. The placements that has been selected by PICA is shown in Figure 4.30, Figure 4.32, Figure 4.34. The results show that our placements are very close to the optimal point based on the best and the worst distance in Table 4.8.

Table 4.8 Statistical Results

	Benchmarks	The Best Distance	The Worst Distance	Correct Placement Count
PICA	3	0	2.8635	35
Random		13.2549	78.1004	0
Serial ICA		0	11.5873	7
Mathematical		0	0	40
PICA	2	0	61.4191	36
Serial ICA		0	127.3185	4
Random		52.9906	1.0572e+03	0
Mathematical		0	0	40
PICA	1	0	475.0343	21
Serial ICA		0	4.6043e+03	2
Random		846.77	1.0447e+04	0
Mathematical		0	0	40

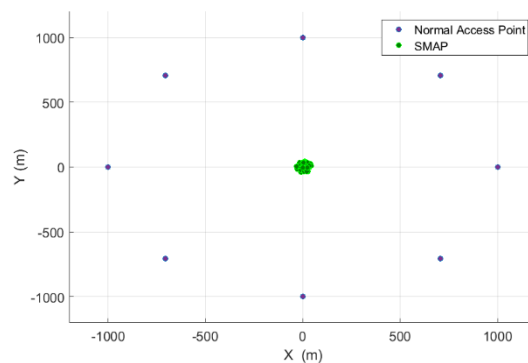


Figure 4.30 The placement of PICA on benchmark 1.

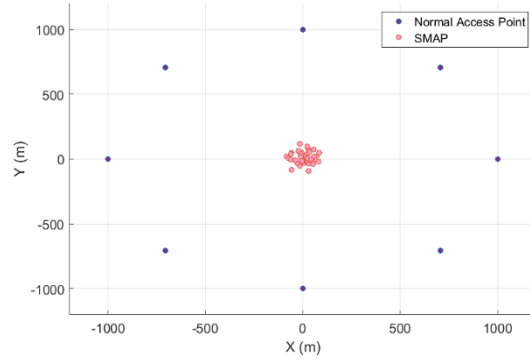


Figure 4.31 The placement of ICA on benchmark 1.

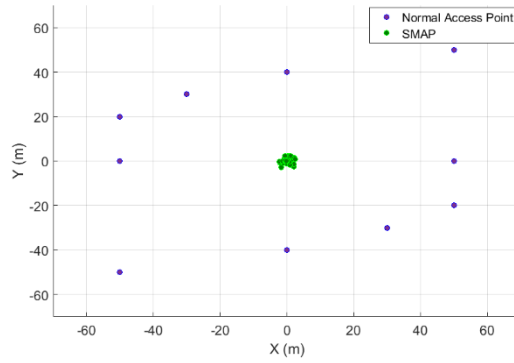


Figure 4.32 The placement of PICA on benchmark 2

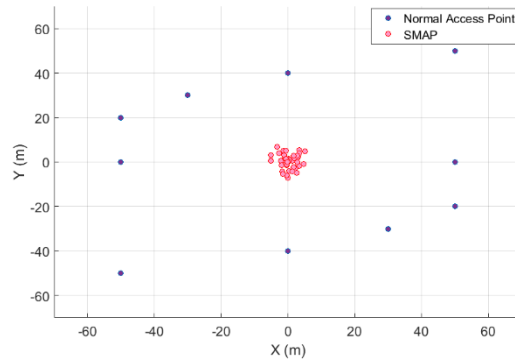


Figure 4.33 The placement of ICA on benchmark 2

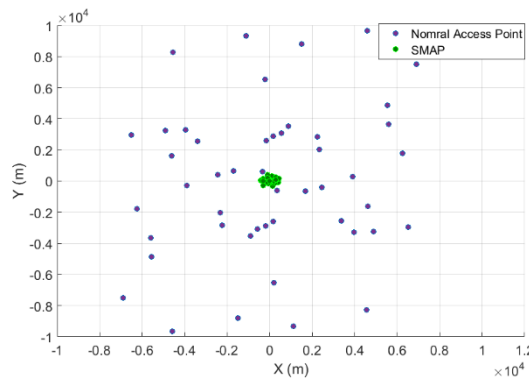


Figure 4.34 The placement of PICA on benchmark 3.

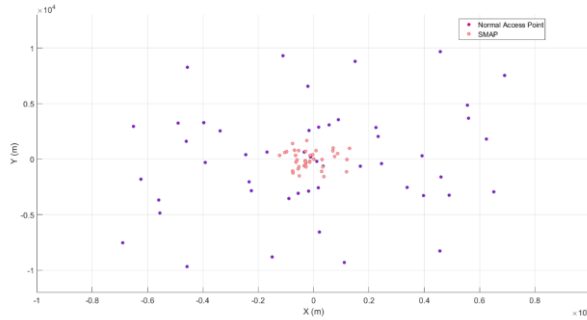


Figure 4.35 The placement of ICA on benchmark 3.

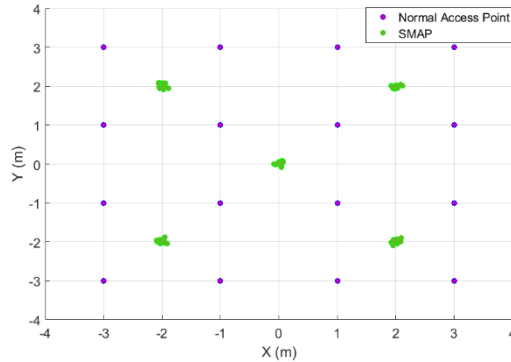


Figure 4.36 The placement of PICA on benchmark 4.

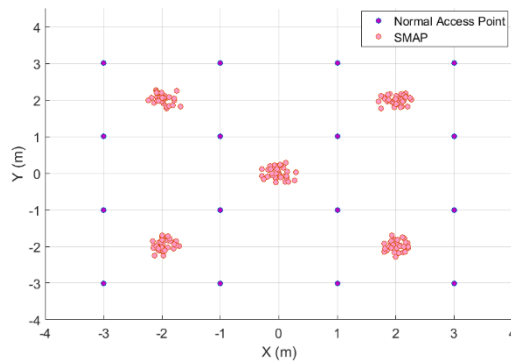


Figure 4.37 The placement of ICA on benchmark 4

4.4.2 Multiple SMAPs

The fourth benchmark considers 5 SMAPs and 16 standard access points.

To check the reliability of the results of the PICA and serial ICA, for each method we run 40 times each with 100 iterations. The initial country is equal to 500. Figures 4.36 show the placement of the proposed method on the fourth benchmark.

The overall results confirm that the proposed fitness function is extremely efficient, letting PICA and ICA obtain the best placement also in the complex placement cases. The results also demonstrate that PICA performs much better than ICA.

In the hierarchal placement (Paper VI), three well-known 2-D benchmarks have been utilized. The static-access points have different weights in the second and the third benchmarks; hence, benchmarks 2 and 3 are more complex than the first one. The results of MPGA have been compared with sequential GA and standard mathematical method.

In these benchmarks, the proposed method should choose the best placement for several SMAPs in two layers. In the first benchmark, twenty-four normal access points and nine SMAPs have been divided into two layers (six SMAPs in the first layer and three SMAPs in the second layer). SMAPs should have different positions. The results also show that MPGA performs much better than GA.

The main aim is to obtain the best placement for multiple SMAPs in two different layers. The first benchmark includes twenty-four static access points, which have a regular arrangement. All normal access points have the same weights.

The second benchmark has thirty-two static access points and divided into two subsets (i.e., A and B).

The third benchmark has sixty static access points, stayed on two hyperbolic curves (a horizontal curve and a vertical one). They also indicate that the proposed MPGA method is successful in both simple and complex case studies.

TABLE 4.9 The stability diagram of all methods on case study 3. Statistical Results

	Case	The Best Distance	Correct Placement Count	The Worst Distance
PGA	3	0	27	17.436
Serial GA		0	11	43.261
Mathematical		-	0	-
PGA	2	0	26	11.814
Serial GA		0	7	34.899
Mathematical		-	0	-
PGA	1	0	28	9.465
Serial GA		0	13	18.927
Mathematical		-	0	-

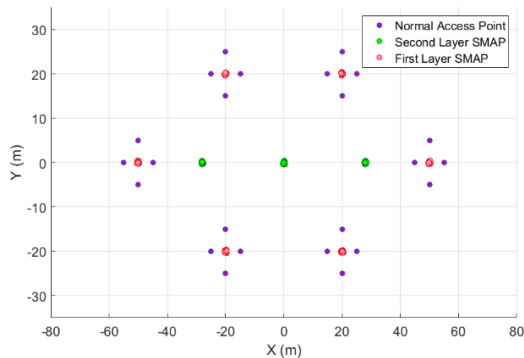


Figure 4.38 The placement of MPGA on case study 1.

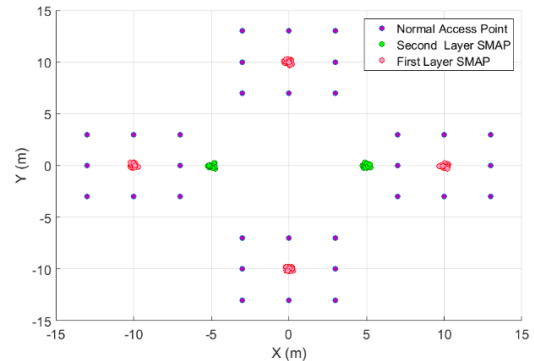


Figure 4.40 The placement of MPGA on case study 2.

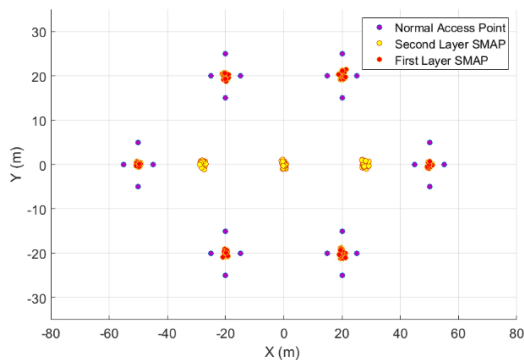


Figure 4.39 The placement of serial GA on case study 1.

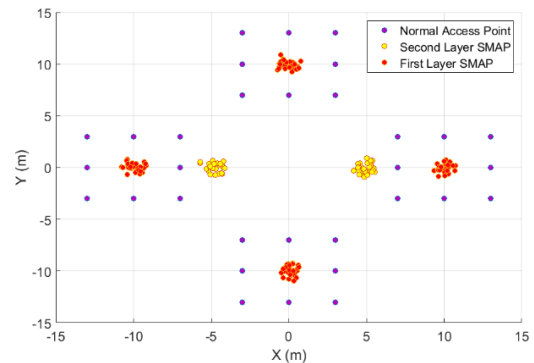


Figure 4.41 The placement of serial GA on case study 2.

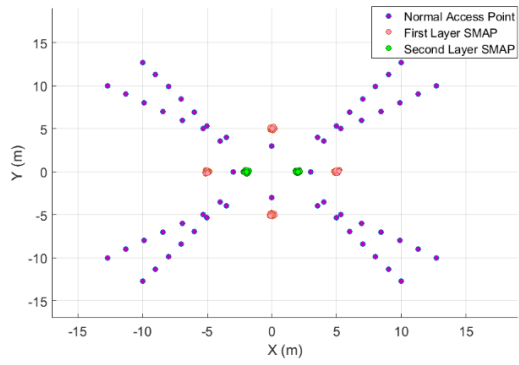


Figure 4.42 The placement of serial MPGA on case study 3

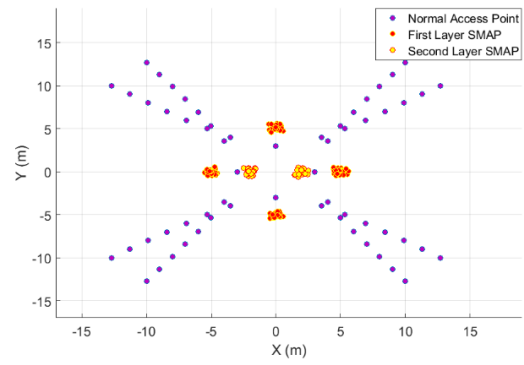


Figure 4.43 The placement of serial GA on case study 3.

Chapter 5

Discussion and Conclusion

In conclusion, this thesis improved the execution time and accuracy of meta-heuristic methods for optimization problems. Commonly, in really complex optimization problems, Evolutionary Computing (ECs) are popular ones. Although ECs show a better performance in comparison with other optimization methods, convergence to local optimums and large execution time motivated us to improve them.

The thesis contribution is threefold as follows:

Reducing the execution time of ECs to apply for extremely complex problems and real-time applications.

Changing the exploration and exploitation to prevent convergence to local optimum in complex problems, and raise the reliability of results. Developed a framework to adapt ECs running in a distributed manner.

Therefore, the main conclusions of this study are summarized as follows:

1) We have selected the GA and ICA as two popular optimization methods for discrete and continuous problems. To improve the execution time of ECs methods we have parallelized GA and ICA, and our experimental results show that not only we have achieved an improvement on the execution time, but the reliability of the results have also been improved by the multi-population technique. The statistical results and stability diagrams show these successes. In Paper I, multi-population ICA (PICA) has been presented, and its results compare with some other methods based on the synthetic benchmarks.

The results show a faster convergence, more reliability and super-linear performance. The improvement of convergence after each migration indicates that the migration operation has a significant effect of converging method to the global optimum.

In Paper II, master-slave ICA and PICA have been presented. Improving the convergence and reliability of results were the primary concern in this paper. These methods and some other related methods have been compared together on more synthetic benchmarks. The results explain that multi-population improves the execution time and reliability of the results, but master-slave only reduces the execution time.

After improving the convergence in Paper I and II, the PICA applied on non-linear equations to test the accuracy of results. The proposed method was compared with other related methods. More accurate results obtained by multi-population ICA.

All the benchmarks and non-linear equations have continuous search spaces based on the behaviour of serial ICA.

In Paper IV, a task scheduling problem has been selected as an order based problem (a type of discrete problems), and MPGA was chosen to apply on it. GA works efficiently on discrete problems. The results confirm the obtained results are reliable, but the convergence does not have enough improvement. Then, in Paper V, PICA applied on complex placement problems (a discrete problem) and

the convergence and reliability of results were improved, but the execution time was not changed.

2) Missing convergence improvement in MPGA (Paper IV) and execution time in PICA (Paper V) have motivated us to find the main reason for these problems. Therefore, MICGA has been proposed from the combination of PICA and MPGA. In MICGA the convergence strategies of PICA and exploration (mutation) and exploitation (crossover) operations of MPGA work together as an efficient optimization method.

In Paper VI, MICGA has been applied on more complex placement problem (Hierarchical placement of smart mobile access points). The results of Paper VI show that we have achieved a faster convergence, less execution time and more reliability. The results show that MICGA has achieved most of our goals in this thesis. In Paper VII and VIII, MICGA is applied on two more complex scheduling problems and their results illustrate that MICGA can satisfy all mentioned objects. We combined ICA and GA, named MICGA, which takes advantage of the distributed, and convergence strategies of ICA, along with the crossover and mutation operations of GA. With this combination, MICGA not only increases the execution of PICA and GA but also has a great potential to be used for both continuous and discrete problems.

3) We adapted PICA, MPGA and MICGA to be executed in a distributed manner using a ring topology (minimum connection) with a low rate of migration (low communication cost) using message passing systems. The experimental results show that we achieved considerable efficiency in comparison with central computation systems.

Chapter 6

Overview of Original Publications

Articles published including the results and analysis from the thesis are summarized below.

6.1 Paper I: PICA: Multi-Population Implementation of Parallel Imperialist Competitive Algorithms

In this paper, a multi-population implementation of ICA is proposed to improve the execution time of ICA. The processors are connected as a ring topology to communicate together. The proposed work starts its process by initializing some independent populations. This helps avoid converging to a local optimum in different populations at the same time.

The key contribution of this paper is the migration operation. By the migration operation, some chromosomes from each processor migrate to another processor to share their genome.

The migration operation helps different populations to change their position from local optimums. The evaluation shows our solution improves the execution time and the stability.

6.2 Paper II: Parallel Imperialist Competitive Algorithms

In this paper, two different types of parallel ICA are introduced, multi-population and master-slave. One processor is selected as a master processor and manages the tasks of other processors. Slaves run some tasks such as chromosome evaluation and return the corresponding values to the master processor.

In this work, our proposed works are evaluated in eight well-known benchmarks, and their outcomes compare with some other parallel EC methods.

The results show that the master-slave ICA improves only the execution time, but PICA promotes reliability and execution time.

6.3 Paper III: Parallel Imperialist Competitive Algorithm Based on Multi-Population Technique for Solving Systems of Nonlinear Equation

In this paper, we have used PICA to solve nonlinear equation systems. Vast uses of non-linear equations are undeniable. Some of their applications are in chemistry, economics, mechanics, engineering, medicine, and robotics.

The complexity of nonlinear equations proves the performance of our method. To qualify our method, we applied PICA on three well-known nonlinear equations. The presented results, such as stability diagram, convergence diagram, statistical results and table of speed up, illustrate that our method has a significant achievement. The accuracy of our results confirms the reliability of PICA. Also, we obtain the super-linear performance and shows the efficiency of our method on continuous problems.

6.4 Paper IV: Finding Near-Optimal Task Scheduling for Distributed Real-Time Environments

In this paper, we proposed an MPGA approach for near-optimal scheduling optimization that guarantees end-to-end deadlines of tasks in distributed processing environments. We analyse two different exploration scenarios, including single and multi-objective explorations.

The principal goal of the single objective exploration algorithm is to achieve a minimal number of processing units for all tasks, whereas a multi-objective optimization tries to optimize two conflicting objectives simultaneously considering the total number of processing units and end-to-end finishing time for all the jobs. The potential of the proposed approach is demonstrated by experiments based on a use case for mapping a number of jobs covering industrial automation systems, where each of the jobs consists of several tasks in a distributed environment.

6.5 Paper V: Placement of Smart Mobile Access Points in Wireless Sensor Networks and Cyber-Physical Systems using Fog Computing

In this paper, a PICA is applied to develop an efficient method for planning the initial placement problem in SMAP based systems. In PICA, we have divided our method into two layers: an algorithmic layer and an architectural layer. To have the optimal placement of SMAPs, all access points are divided into two major groups that are SMAPs and static access points. Static access points only transfer data and maintains the connectivity in the network. SMAPs can work like static access points, but they also can accomplish other tasks, like maintaining other access points when required. To clarify the problem, static access points have been elected as target points for SMAPs, as we are considering the initial placement only. Also, each static access point has an independent weight. The weight of each access point shows the probability of requiring support from SMAPs.

PICA has been tested on four Well-known 2-D benchmarks. The achieved results have been compared with the ICA, and other similar placement methods.

6.6 Paper VI: Hierarchical Placement of Smart Mobile Access Points in Wireless Sensor Networks using Fog Computing

In this paper, the idea of a hierarchical smart mobile access point (HSMAP), which is a significant building block for an intelligent network, has been proposed. The placement of SMAPs is a fundamental factor in the dynamic network. Since the placement of HSMAPs is an NP problem, we solve the initial placement of them using an MPGA with an efficient evaluation.

Also, since SMAPs can be used in a large scale WSN, scalability of SMAPs placement is considerably important. Therefore, we have presented a hierarchical implementation of SMAPs to solve the scalability problem. The proposed method has been tested on four Well-known 2-D benchmarks. The achieved results have been compared with the serial ICA, and other similar placement methods.

6.7 Paper VII: Optimizing Scheduling for Heterogeneous Computing Systems using Combinatorial Meta-heuristic Solution

In this paper, a new task graph scheduling approach is presented. To perform our work more complicated, NoC is assumed as a heterogeneous multiprocessor system. We tackle this task scheduling problem by MICGA. In this paper, the main goal is to improve execution time and reliability. The presented encoding method is based on task priority, and our work focuses on the pipelined execution model of applications. In the pipeline model, input data is divided into several parts, and these parts are then one by one used as inputs to an application. We make a pack of the parts and find the best schedule for this pack of data. To test and compare our method, four commonly explored real applications have been utilized to demonstrate the performance of the MICGA.

6.8 Paper VIII: NOMeS: Near-Optimal Metaheuristic Scheduling for MPSoCs

In this work, we use a stochastic model for the task-scheduling problem, where the data communication times between tasks and the execution times of tasks are known. The task scheduling problem based on a directed acyclic task graph (DAG) that specifies the preference relations of the tasks is known to be an NP-hard problem. In general, priority constraints between tasks can be non-uniform, but we consider here, for simplicity, that the MPSoC platform is uniform (a homogeneous multiprocessor system) and non-primitive (each processor completes the current task before starting the execution of the next task).

In this paper, we tackle this problem by introducing a perfect combination of a GA and the PICA. Moreover, we explicitly estimate the communication delays between processors. When two communicating tasks are mapped onto the same processor, we assume that the communication delay is zero. However, when they are mapped onto different processors, a finite communication delay is assumed and modelled.

We have used the concept of an order-based country (OBC) as an extension of the order-based coding method. This coding defines the order of tasks and the selected processor to run each task. To improve the outcome of the optimization process, we used MICGA which takes advantage of both ICA and GA. This combination improves the convergence policy and selection pressure, by keeping all countries in all iterations and avoiding the use of a general selection operation such as tournament schemes and roulette wheel.

Bibliography

- [1] A. Majd, E. Troubitsyna, and M. Daneshtalab, "Safety-Aware Control of Swarms of Drones," 12th ERCIM/EWICS/ARTEMIS Workshop on Dependable Smart Embedded Cyber-physical Systems and Systems-of-Systems at SAFECOMP 2017, (DECSoS '17), 2017
- [2] A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computing. Springer, 2003
- [3] E. Cantú-Paz, "A Survey of Parallel Genetic Algorithms," Department of Computer Science and Illinois Genetic Algorithms Laboratory University of Illinois at Urbana-Champaign, 1997.
- [4] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning. Machine Learning" , 1988
- [5] Gen M, Cheng R. Genetic algorithm and engineering optimization. NewYork:Wiley; 2000
- [6] E. A. Gargari, and C. Lucas, "Imperialist Competitive Algorithm: An Algorithm for Optimization Inspired by Imperialistic Competition," Congress on Evolutionary Computation, IEEE, 2007
- [7] H. Lotfi, A. Boroumandnia and Sh. Lotfi, "Task Graph Scheduling in Multiprocessor Systems Using a Coarse Grained Genetic Algorithm," 2nd International Conference on Computer Technology and Development, IEEE, 2010.
- [8] H. Lotfi, Sh. Lotfi and A.Boroumandnia, "Task Graph Scheduling in Multiprocessor Systems Using a Two Population Genetic Algorithm," International Conference on Software and Computing Technology, IEEE, 2010.
- [9] A. Majd, Sh. Lotfi and G. Sahebi, "Review on Parallel Evolutionary Computing and Introduce Three General Framework to Parallelize All EC Algorithms," The 5th Conference on Information and Knowledge Technology, IEEE, pp. 61-66, 2013
- [10] H. Liu, P. Li and Y. Wen, "Parallel Ant Colony Optimization Algorithm," World Congress on Intelligent Control and Automation. China, 2006.
- [11] Z. Yang and B. Yu, "A Parallel Ant Colony Algorithm for Bus Network Optimization," Computer-Aided Civil and Infrastructure Engineering, vol. 22, pp. 44-55, 2007.
- [12] A. Mousa, W. Wahed and R. Allah, "A Hybrid Ant Colony Optimization Approach Based Local Search Scheme for Multi Objective Design Optimizations," Electric Power Systems Research, vol. 81, pp. 1014-1023, ELSEVIER, 2011.

- [13] J. Digalakis and K. Margaritis, "A Parallel Memetic Algorithm for Solving Optimization Problems," 4th Metaheuristics International Conference. Parallel Distributed Processing Laboratory, Greece, 2001.
- [14] R. Parpinelli, C. Benitez and S. Lopes, "Parallel Approaches for the Artificial Bee Colony Algorithm," Handbook of Swarm Intelligence, vol. 8, pp. 329-345, 2010.
- [15] H. Narasimhan, "Parallel Artificial Bee Colony (PABC) Algorithm," World Congress on Nature & Biologically Inspired Computing, pp. 306-311. IEEE, 2009.
- [16] L. Vanneschi, D. Codecasa and G. Mauri, "A Comparative Study of Four Parallel and Distributed PSO Methods," New Generation Computing. vol. 29, pp. 129-161, 2011.
- [17] P. Y. Yin, S. S. Yu, P. P. Wang and Y. T. Wang, "A Hybrid Particle Swarm Optimization Algorithm for Optimal Task Assignment in Distributed Systems," Computer Standards & Interfaces, vol. 28, pp. 441–450, ELSEVIER, 2006.
- [18] E. Alba, F. Luna, A. J. Nebro and J. M. Troya, "Parallel Heterogeneous Genetic Algorithms for Continuous Optimization," Parallel Computing, vol. 30, pp. 699–719, ELSEVIER, 2004.
- [19] M. Dorigo and G. Di Caro, "Ant colony optimization: a new meta-heuristic" The IEEE Congress on Evolutionary Computation-CEC99, 1999.
- [20] D. Karaboga, and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm," Journal of Global Optimization, Vol. 39-3, pp. 459–471, 2007.
- [21] R. Eberhart ; J. Kennedy, "A new optimizer using particle swarm theory," The Sixth IEEE International Symposium on Micro Machine and Human Science, 1995.
- [22] C. Cotta, L. Mathieson and P. Moscato, "Memetic Algorithms," Handbook of Heuristics, pp 1-32, 2017.
- [23] E. Alba, "Parallel Evolutionary Algorithms Can Achieve Super-Linear Performance," Information Processing Letters, vol. 82, pp. 7–13, ELSEVIER, 2002.
- [24] E. Alba, F. Luna, A. J. Nebro and J. M. Troya, "Parallel Heterogeneous Genetic Algorithms for Continuous Optimization," Parallel Computing, vol. 30, pp. 699–719, ELSEVIER, 2004.
- [25] M. Abdollahi, A. Isazadeh, and D. Abdollahi, "Imperialist competitive algorithm for solving systems of nonlinear equations," Comput. Math. Appl., vol. 65, pp. 1894-1908, 2013.

- [26] Ibrahiem M. M. El-Emary and Mona M. Abd El-Kareem, Toward Using Genetic Algorithm for Solving Nonlinear Equation Systems, *World Appl. Sci. J.* 5 (2008) 282-289.
- [27] Nikos E. Mastorakis, Solving Non-linear Equations via Genetic Algorithms, *Proceedings of the 6th WSEAS Int. Conf. on Evolutionary Computing*, Lisbon, Portugal, June 16-18 (2005) 24-28
- [28] G. Li, Zh. Zeng, A neural-network algorithm for solving nonlinear equation systems, *IEEE International Conference on Computational Intelligence and Security*, CIS08 (2008) 20-23.
- [29] G. Huan-Tong, S. Yi-Jie, S. Qing-Xi, W. Ting-Ting, Research of Ranking Method in Evolution Strategy for Solving Nonlinear System of Equations, *IEEE International Conference on Information Science and Engineering*, ICISE09 (2009) 348-351.
- [30] M. Abdollahi, A. Isazadeh, D. Abdollahi, Solving systems of nonlinear equations using imperialist competitive algorithm, *The 8th International Industrial Engineering Conference*, 8 (2012) 1-6.
- [31] A. Ouyang, Y. Zhou, Q. Luo, Hybrid Particle Swarm Optimization Algorithm for Solving Systems of Nonlinear Equations, *IEEE International Conference on Granular Computing*, GRC09 (2009) 460- 465.
- [32] J. Wu, Zh. Cui, J. Liu, Using Hybrid Social Emotional Optimization Algorithm with Metropolis Rule to Solve Nonlinear equations, *IEEE International Conference on Cognitive Informatics & Cognitive Computing*, ICCI*CC'11 (2011) 405-411
- [33] M. Abdollahi, Sh. Lotfi, D. Abdollahi, Solving systems of nonlinear equations using cuckoo optimization algorithm, *The 3rd International conference on The Contemporary Issues in Computer Sciences and Information Technology (CICIS)*, 3 (2012) 191-194.
- [34] M. Abdollahi, A. Bouyer, D. Abdollahi, Improved cuckoo optimization algorithm fo solving systems of nonlinear equations, *J. Supercomput.* 72 (2016) 1246-1269.
- [35] Y.Z. Luo, G.J. Tang, L.N. Zhou, Hybrid approach for solving systems of nonlinear equations using chaos optimization and quasi-Newton method, *Appl. Soft. Comput.* 8 (2008) 1068-1073. J. Clerk Maxwell, *A Treatise on Electricity and Magnetism*, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
- [36] C. Grosan, A. Abraham, A New Approach for Solving Nonlinear Equations Systems, *IEEE Trans. Syst. Man Cybern. A* 38 (3) (2008) Senior Member, IEEE.

- [37] Y. Mo, H. Liu, Q. Wang, Conjugate direction particle swarm optimization solving systems of nonlinear equations, *Comput. Math. Appl.* 57 (2009) 1877-1882.
- [38] M. Jaberipour, E. Khorram, B. Karimi, Particle swarm algorithm for solving systems of nonlinear equations, *Comput. Math. Appl.* 62 (2011) 566-576.
- [39] E. Pourjafari, H. Mojallali, Solving nonlinear equations systems with a new approach based on invasive weed optimization algorithm and clustering, *Swarm Evol. Comput.* 4 (2012) 3343.
- [40] N. Henderson, W. F. Sacco, G. Mendes Platt, Finding more than one root of nonlinear equations via a polarization technique: An application to double retrograde vaporization, *Chem. Eng. Res. Des.* 88 (2010) 551-561
- [41] Adam TL, Chandy KM, Dicksoni JR. A comparison of list schedules for parallel processing systems. *Communications of the ACM* 1974;17(12):685–90.
- [42] R. Moradi and D. Dal, A Multi-Population Based Parallel Genetic Algorithm for Multiprocessor Task Scheduling with Communication Costs, 2016 IEEE Symposium on Computers and Communication (ISCC).
- [43] Wu MY, Gajski DD. Hypertool: a programming aid for message-passing systems. *IEEE Transactions on Parallel and Distributed Systems* 1990;1(3):330–43.
- [44] Hou ESH, Ansari N, Hong R. A genetic algorithm for multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems* 1994;5(2):113–20.
- [45] R. Hwang, M. Gen and H. Katayama, “A comparison of multiprocessor task scheduling algorithms with communication costs” *Computers & Operations Research*, Vol. 35, pp. 976 – 993, ELSEVIER, 2008.
- [46] T. Lei and S. Kumar, “A Two-Step Genetic Algorithm for Mapping Task Graphs to a Network on Chip Architecture,” *Proc. Euromicro Symp. Digital System Design (DSD 03)*, IEEE Press, 2003, pp. 180-187.
- [47] D. Wu, B. Al-Hashimi, and P. Eles, “Scheduling and Mapping of Conditional Task Graphs for the Synthesis of Low Power Embedded Systems,” *Proc. Design, Automation and Test in Europe (DATE 03)*, IEEE CS Press, 2003, pp. 90-95.
- [48] S. Murali and G. De Micheli, “Bandwidth-Constrained Mapping of Cores onto NoC Architectures,” *Proc. Design, Automation and Test in Europe (DATE 04)*, IEEE CS Press, 2004, pp. 896-901.
- [49] S. Manolache, P. Eles, and Z. Peng, “Fault and Energy-Aware Communication Mapping with Guaranteed Latency for Applications Implemented on NoC,”

- Proc. 42nd Annual Design Automation Conf. (DAC 05), ACM Press, 2005, pp. 266-269.
- [50] J. Hu and R. Marculescu, "Energy- and Performance- Aware Mapping for Regular NoC Architectures," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 4, 2005, pp. 551-562.
- [51] Y. Xu, K. Li, J. Hu and K. li, "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," *Information Sciences*, Vol.270, pp. 255-287,Elsevier,2014.
- [52] Wu AS, Yu H, Jin S, Lin K-C, Schiavone G. An incremental genetic algorithm approach to multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems* 2004;15(9):824–34.
- [53] Yang T, Gerasoulis A. DSC: scheduling parallel tasks on an unbounded number of processors. *IEEE Transactions on Parallel and Distributed Systems* 1994;5(9).
- [54] T hanalapati T, Dandamudi S. An efficient adaptive scheduling scheme for distributed memory multicomputers. *IEEE Transactions on Parallel and Distributed Systems* 2001;12(7):758–68
- [55] Corbalan J, Martorell X, Labarta J. Performance-driven processor allocation. *IEEE Transactions on Parallel and Distributed Systems* 2005;16(7):599–611.
- [56] Adam TL, Chandy KM, Dicksoni JR. A comparison of list schedules for parallel processing systems. *Communications of the ACM* 1974;17(12):685–90.
- [57] Hwang RK, Gen M. Multiprocessor scheduling using genetic algorithm with priority-based coding. *Proceedings of IEEJ conference on electronics, information and systems; 2004.*
- [58] R. F. Freund, "Optimal selection theory for superconcurrency," *Proc. Supercomputing 89*. IEEE Computer Society, Reno, NV, pp. 699703. 1989.
- [59] M. Wu, and DD. Gajski, "Hypertool: a programming aid for messagepassing systems," *IEEE Transactions on Parallel and Distributed Systems*, pp. :33043, 1990.
- [60] Fang-Jing Wua , Yu-Fen Kao b , Yu-Chee Tseng a, "From wireless sensor networks towards cyber physical systems," *Pervasive and Mobile Computing*, Elsevier, Vol. 7, pp. 397–413, 2011.
- [61] Priyanka Pandit1 , Swarupa Kamble2, "A Survey on Knowledge Extraction from WSN," *International Journal of Science and Research (IJSR)*, Vol. 6, pp. 384-387, 2016.

- [62] N. Heo and P. K. Varshney, "Energy-Efficient Deployment of Intelligent Mobile Sensor Networks," IEEE Transactions on Systems, Man, Cybernetics, Part A, Vol. 35, No. 1, pp. 78-92, January 2005.
- [63] A. Kansal et al., "Controlled Mobility for Sustainable Wireless Sensor Networks," in the Proceedings of IEEE Sensor and Ad Hoc Communications and Networks (SECON'04), Santa Clara, CA, October 2004.
- [64] Y. J. Cha, A. Raich, "Optimal placement of active control devices and sensors in frame structures using multi-objective genetic algorithms," Structural Control and Health Monitoring, Vol. 20, pp. 16-44, 2013
- [65] A.Majd and G.Sahebi, "A Survey on Parallel Evolutionary Computing and Introduce Four General Frameworks to Parallelize all EC Algorithms and Create New Operation for Migration," Journal of Information and Computing Science, vol. 9, pp.97-105,2014.
- [66] Y. T. Hou, Yi Shi, and Ha. D. Sherali, "On Energy Provisioning and Relay Node Placement for Wireless Sensor Networks," In IEEE Trans. on Wireless Comm., (4)5:2579-2590, Sep. 2005.
- [67] J. Zhu, D. S. Chan, M. S. Prabhu, P. Natarajan, H. Hu and F. Bonomi, "Improving Web Sites Performance Using Edge Servers in Fog Computing Architecture," IEEE Seventh International Symposium on Service-Oriented System Engineering, pp. 320-323, 2013.
- [68] ed. J. Wu. Auerhach, "Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks" Publications, 2006.
- [69] G. Karypis, E. H. Han, and V. K. Chameleon, "Hierarchical Clustering Using Dynamic Modeling," IEEE Computer, Aug 1999.
- [70] J. Suomela, "Computational Complexity of Relay Placement in Sensor Networks" In SOFSEM 2006, LNCS 3831, pp. 521-529. Springer-Verlag, 2006.
- [71] S. Toumpis and G. A. Gupta, "Optimal Placement of Nodes in Large Sensor Networks Under a General Physical Layer Model," In Proc of IEEE SECON, September 2005.
- [72] N. Heo and P. K. Varshney, "Energy-Efficient Deployment of Intelligent Mobile Sensor Networks," IEEE Transactions on Systems, Man, Cybernetics, Part A, Vol. 35, No. 1, pp. 78-92, January 2005.
- [73] <https://www.open-mpi.org/>
- [74] <https://www.mpich.org/>

- [75] H. Gyurhan and A. Nedzhibov, “family of multi-point iterative methods for solving systems of nonlinear equations”, *Journal of Computational and Applied Mathematics*, Vol. 222, pp. 244-250, 2008.
- [76] C. A. Floudas, P. M. Pardalos, C. S. Adjiman, W. R. Esposito, Z. H. Gumus, S. T. Harding, J. L. Klepeis, C. A. Meyer, and C. A. Schweiger, “Handbook of Test Problems in Local and Global Optimization,” Kluwer Academic Publishers, Dordrecht, the Netherlands, 1999.
- [77] C. Wang, R. Luo, k. Wu, and B. Han, “A new filled function method for an unconstrained nonlinear equation,” *Journal of Computational and Applied Mathematics*, Vol. 235, pp. 1689-1699, 2011.
- [78] N. Khalilzad, K. Rosvall and I. Sander, A Modular Design Space Exploration Framework for Multiprocessor Real-Time Systems, *Forum on specification and Design Languages (FDL'16)*, 2016.
- [79] James Kennedy, “Particle Swarm Optimization,” *Encyclopedia of Machine Learning*, Springer, 2010.
- [80] P. J. M. Laarhoven, and E. H. L. Aarts, “Simulated annealing,” *Journal of Simulated Annealing: Theory and Applications*, Springer, pp.7-15, 1987.
- [81] S. A. Mirjalili, S. M. Mirjalili, and A. Lewis, “Grey Wolf Optimizer,” *Journal of Advances in Engineering Software*, Vol. 69, Pages 46-61, 2014.
- [82] A. H. Gandomi, and A. H. Alavi, “Krill herd: A new bio-inspired optimization algorithm,” *Journal of Communications in Nonlinear Science and Numerical Simulation*, Vol. 17, p.p 4831-4845, 2012.
- [83] A. H. Gandomi, “Interior search algorithm (ISA): A novel approach for global optimization,” *ISA Transactions*, Vol. 53, Issue 4, p.p 1168-1183, 2014.

Part II
Original Publications

Paper I

PICA: Multi-Population Implementation of Parallel Imperialist Competitive Algorithms

A. Majd, Sh. Lofti, G. Sahebi, M. Daneshtalab and J. Plosila

PICA: Multi-Population Implementation of Parallel Imperialist Competitive Algorithms

Amin Majd
Department of
Information Technology
University of Turku
Turku, Finland
amin.majd@utu.fi

Shahriar Lotfi
Department of
Computer Sciences
University of Tabriz
Tabriz, Iran
shahriar_lotfi@tabrizu.ac.ir

Golnaz Sahebi
Department of
Information Technology
University of Turku
Turku, Finland
golnaz.sahebi@utu.fi

Masoud Daneshtalab
Royal Institute of
Technology (KTH),
Stockholm, Sweden,
masdan@kth.se

Juha Plosila
Department of
Information Technology
University of Turku
Turku, Finland
juplos@utu.fi

Abstract—The importance of optimization and NP-problems solving cannot be over emphasized. The usefulness and popularity of evolutionary computing methods are also well established. There are various types of evolutionary methods that are mostly sequential, and some others have parallel implementation. We propose a method to parallelize Imperialist Competitive Algorithm (Multi-Population). The algorithm has been implemented with MPI on two platforms and have tested our algorithms on a shared- memory and message passing architecture. An outstanding performance is obtained, which indicates that the method is efficient concern to speed and accuracy. In the second step, the proposed algorithm is compared with a set of existing well known parallel algorithms and is indicated that it obtains more accurate solutions in a lower time.

Keywords- optimization; evolutionary computing; parallel approaches; ICA; parallel programming; multi-population; super linear performance.

I. INTRODUCTION

Optimization algorithms can be categorized into two categories: heuristic and meta heuristic. In the class of heuristic algorithms, there are some constructions and improvement algorithms such as the meta heuristic algorithms manage a chain or flow of executions of classical heuristics, e.g. tabu search, simulated annealing, genetic or memetic algorithms.

Computability is a significant problem for many researchers especially in NP-hard problems, which do not have suitable solutions, which are able to find the best answers in a limited time. There are several techniques to solve some NP-hard Problems, but Evolutionary Computing (EC) are the better and more popular ones. Different types of EC methods are useful for various kinds of problems, for instance, genetic algorithms are an old method that is suitable for discrete problems. Genetic algorithms are population-based search methods that mimic the process of natural selection and evolution, which some characteristics of them help researchers to optimize their problems. Particle Swarm Optimization (PSO) is another

evolutionary method that mimics behavior of birds when they migrate to other places.

EC methods enhance to solve different problems, but there are some disadvantages associated with them. For example, it is impossible for some algorithms, which have a large search space, to converge to optimum solutions. Hence, the initial population should be increased to improve the results. Speed of algorithms is the other challenge in this area. Sometimes, answers are found taking a long time. Parallel algorithms are the proper solutions that enhance to improve the quality and time of obtaining results.

Previously, researchers have utilized several parallel EC techniques to achieve better results (e.g. parallel ant colony optimization (PACO) [4], parallel genetic algorithms [3], parallel ABC (PABC) [5], parallel memetic [7], and parallel PSO [6]). We know from [15] that some parallel EC methods can achieve super-linear performance. Super-linear performance is an expression for some scarce parallel algorithms that their efficiency is more than one (efficiency value normally is between zero and one).

Recently, several EC methods have been created, but some of them have not been yet parallelized. For example, Imperialist Competitive Algorithm (ICA) is an efficient method that is outstanding for continuous problems. In this paper, a parallel implementation of ICA (PICA) is proposed. The PICA is implemented in multi-population (coarse-grain) strategy. We have implemented PICA and tested it on four mathematical benchmarks; meanwhile it has obtained a super-linear performance. They have been implemented on two different platforms and have been utilized the Message Passing Interface (MPI) instructions on the ring connection topology.

In Section 2, a parallel multi-population implementation of ICA will be introduced. In Section 4, PICA will be compared with ICA then some algorithms, such as PICA, PABC, GPU Based PSO-TM, and C-PPSO will be compared.

II. PARALLEL ICA

A. Imperialist Competitive Algorithm (ICA)

ICA is an optimization method based on imperialistic competition. ICA is a new EC algorithm and optimizes results of problems. In this algorithm, all countries are divided into two categories: colonies and imperialist states. The major part of this algorithm is imperialistic, and hopefully causes the colonies to converge to the global minimum. In the first step, the algorithm creates some countries and sorts them, then selects the best countries to be imperialists, and the remaining countries form the colonies of these imperialists. These colonies start moving toward their relevant imperialist after dividing all colonies among imperialists [10]. The next step computes the power of each imperialist and the imperialistic competitive step follows. The weakest imperialist loses its weakest colony and the selected imperialist obtains this colony and these steps are repeated until reaching a termination condition. The termination condition can be different, for example, the ICA stops when we have one imperialist with all colonies as members of this imperialist. This algorithm starts by creating colonies in initial population and in the next step it sorts all colonies and divides them among imperialists with the best evaluation; in the next step, imperialistic competition will start. In this step, some operations will be repeated until arriving at the termination condition.

ICA is a suitable method to optimization problems, but there would be similar problems in ICA as in the other EC algorithms; in case we confront with a big problem with a far-reaching search area, we need a big initial population to obtain a more accurate and reliable result, but with a processor we cannot realize this requirement. In another case, when we are confronted with a complex problem that needs complex computation, the run time will increase. Therefore, we need to utilize some new methods to improve speed and efficiency. A parallel computing method for ICA presented here to improve its speed and efficiency. The proposed method is a multi-population implementation of the ICA.

Sequential ICA algorithm has a parallel structure, but there is no parallel implementation for the ICA. In the ICA, each imperialist and colonies work independently and after a decade a colony moves to another imperialist; so this algorithm works like a multi-population method that executes on a processor. In the following, we implement a multi-population method for the ICA.

B. Multi-population PICA

In our work, we utilize a multi-population model to implement the PICA using selective local search strategy. In our implementation, we have several processors that are connected together on a ring

topology in the message passing method, but in the shared-memory method there is not any topology. In each processor, we first initiate independent countries and run the ICA independently, and after some decades (which is different on different runs) the best country is migrated from processor P_i to P_{i+1} and replaces it on the worse country. We utilize the ring topology to connect processors together. The numbers of whole countries in all processors are equal even when countries migrate to other processors. All migrations between processors are done synchronously. We know the migration strategy can be different leading to different results. For example, when we use a sparse connection topology, it is better that a migration strategy is utilized with low rates, and the distance of migration should be short. The important problem is the time of routing. In a fully connection topology, the best results are obtained when the migration rates are high. Therefore, we know that a migration operation is useful with suitable rates; we have a sparse connection topology (ring topology), so we use the migration operation with low rates. In figure 1, the pseudo code for the multi-population ICA is presented.

In the multi-population ICA, we increase the number of all countries and increase the selecting pressure; therefore, it helps us to obtain more accurate results in the shortest time, and convergence to results is faster than in the sequential ICA.

Processor P_i :

1. Create independent initial countries.
2. Run ICA algorithm independently.
3. If now is time of migration do
 - 3.1 Wait when all processors arrive to this point.
 - 3.2 Send the best country to processor $(P_{i+1}) \bmod$ (number of processors).
 - 3.3 Receive a country from $(P_{i-1}) \bmod$ (number of processors) and replace it with the worst country.
4. If termination condition is obtained then terminate algorithm Else go to 2.
5. Show the best country.
6. End.

Figure 1. Pseudo code of Multi-Population ICA.

III. TEST FUNCTIONS AND BENCHMARKS

In this paper, we use four test functions or mathematical benchmark functions to compare our algorithm (PICA) with sequential ICA and some other parallel EC methods. All these problems are minimization problems. We can show mathematical benchmarks on table 1.

IV. EVALUATION AND EXPERIMENTAL RESULTS

We use all four test functions with different conditions and different kinds of platforms. The algorithms are implemented on the share memory and message passing structure. First, we compare our

algorithms with sequential ICA and follow our comparison with some other parallel EC methods.

We implement parallel ICA on share memory and message passing model. Also we utilize MPI to parallelize our algorithms and MPICH2 to run the algorithms. In the multi-population ICA, we connect processors in a ring topology with different processors on different tests. We test our algorithms on two platforms; the first platform's specification is on table 2 and the second platform is an Intel core i3-330M, processors 2.13 GHz(64-bit), memory 4 GB.

TABLE I. MATHEMATICAL BENCHMARKS

	Name	Bounds	Equation	m i n
f_4	Rosen brock	[-50,50]	$\sum_{i=1}^{n-1} 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2$	0
f_5	Rastrigin	[-5.12,5.12]	$\sum_{i=1}^n x_i^2 - 10 \cos(2\pi x_i) + 10$	0
f_6	Akley	[-32,32]	$20 + e - 20 \exp(-0.2 \sqrt{\frac{1}{D} \sum_{i=1}^D x_i^2}) - \exp(\frac{1}{D} \sum_{i=1}^D \cos(2\pi x_i))$	0
f_8	Griewank	[-600,600]	$\frac{1}{4000} \sum_{i=1}^n x_i^2 - \prod_{i=1}^n \cos(\frac{x_i}{\sqrt{i}}) + 1$	0

TABLE II. SPECIFICATION OF 8 PROCESSORS PC CLUSTER SYSTEM.

CPU	Intel Core i7 2.67 GHz
Mother Board	Gigabyte EX-58 EXTREME
RAM	6 GB DDR3
HDD	500 GB
NIC	Gigabyte
Network Switch	Cisco Catalyst 3750
Operating System	WINDOWS XP 64-bit
MPI Library	MPICH2 1.4
Compiler	Visual C++ 6.0
GPU	NVIDIA GeForce 9600 GT

A. Convergence

In this section, we show convergence diagrams of benchmarks and we can compare our results and diagrams with the results and diagrams of [8], [10], [23], [24], [25] and [26]. These results obtained with different number of processors and different numbers of population size on multi-population PICA.

After the comparison between our algorithm convergence diagram and those of [8], [10], [23], [24], [25] and [26], we can claim that our algorithm convergence obtain results faster than other methods in lower iteration. Convergence diagrams are shown in figures 2, 3, 4 and 5.

B. Stability

In this section we want to show that our algorithms are stable; so we test it on benchmarks and drawing stability diagrams of them. Stability diagrams shown in figures 6, 7, 8 and 9 and the results claim that our

algorithms are stable and accurate. Our results show that our algorithms are more stable and more accurate than that of other methods.

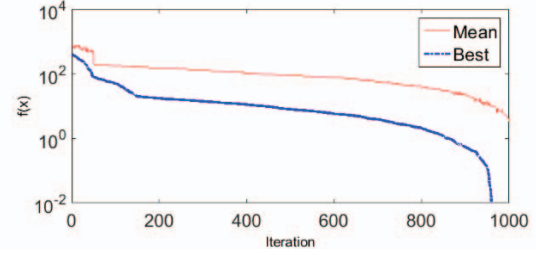


Figure 2. Convergence diagrams of f_4

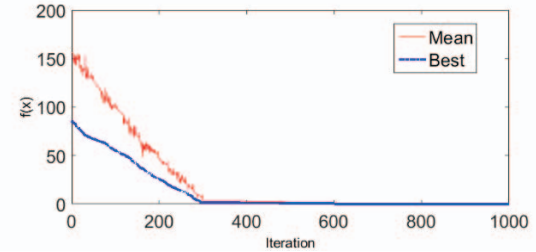


Figure 3. Convergence diagrams of f_5

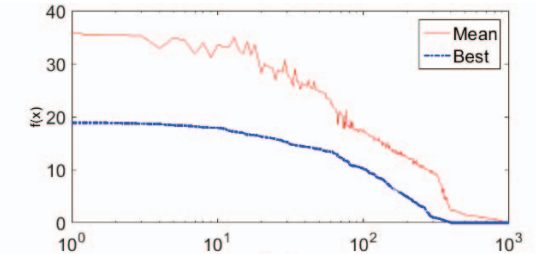


Figure 4. Convergence diagrams of f_6

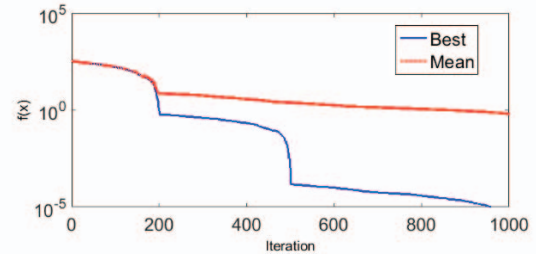


Figure 5. Convergence diagrams of f_8

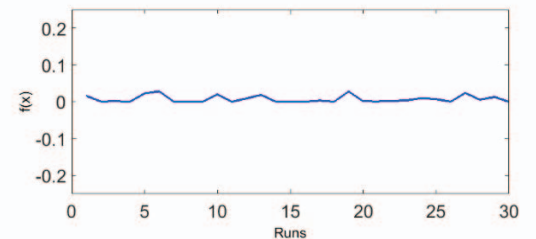


Figure 6. Stability diagrams of f_4

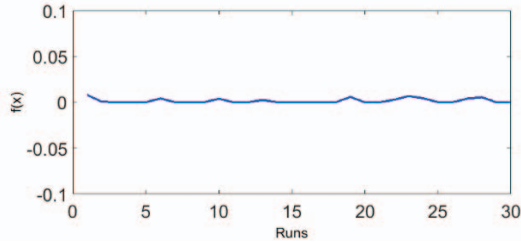


Figure 7. Stability diagrams of f_5

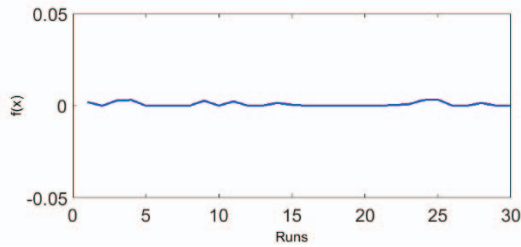


Figure 8. Stability diagrams of f_6

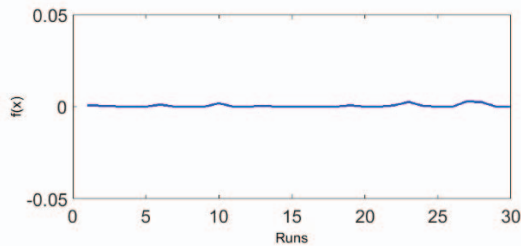


Figure 9. Stability diagrams of f_8

C. Speedup and Efficiency

Speedup value is an important parameter to check quality of parallelization that is obtained from this function and m is the numbers of processors:

$$\begin{aligned} \text{Speedup} &= \frac{\text{Execution time on one processor core}}{\text{Execution time on } m \text{ processor cores}} \\ &= \frac{T(1)}{T(m)} \end{aligned} \quad (1)$$

And the efficiency (normalized speedup) was calculated using:

$$\text{Efficiency} = \frac{\text{Speedup}}{m} \quad (2)$$

These are suitable parameters for deciding whether parallel algorithms are suitable or not. We could use these parameters to compare the speed of our algorithms with that of other parallel EC methods without similar platforms. In this method, we know from [18], [19] and [20] that serial ICA is faster than GA, ES, ABC and PSO. Therefore, if our speed up and efficiency values are bigger than the parallel implementation of these EC algorithms, then our algorithm is faster than those of parallel EC methods. The reason for this decrease is the behavior of the benchmark; in the ICA, the best results are obtained

after 10 iterations and in the PICA after 3 iterations; so this benchmark is very simple and when we find results with 6 processors after 3 iterations, it is not useful to use more processors.

In figures 10, 11, 12 and 13 we show the efficiency and speed up with an increase in the number of processors, and they show that the parallel implementations are useful.

Super linear performance is the main Achievement of our algorithms. In all benchmark diagrams, we can show that we obtain the super linear performance. So we can claim that our implementation and our algorithms are efficient, and the parallel ICA is a better choice for solving problems.

D. Comparison between ICA and Parallel ICA

In this section, we compare them with each other; we use two different type kinds of comparisons: we use the convergence diagram, decades of convergence, and the table of speed up to compare the multi-population PICA and the ICA; and we use the speed up diagram to compare between PICA and ICA. We use two important benchmarks that are used in [10] and we can illustrate the results of them with our algorithm.

In this experiment, we use two platforms; when we use 2 processors we implement our algorithm on Plat1 platform and when we use 6 processors we implement our algorithm on Plat2 platform, and we use message passing architecture. In our algorithms we have 100 countries and 8 imperialists in each processor. Revolution's rate is 0.4 and zeta parameter is 0.1; we know that we can obtain different result by changing these parameters, but we test our algorithms using a constant value. Our connection strategy is the ring topology that is a spars connection strategy.

We can reveal some good points for our algorithms. It is clear that the multi-population method of PICA converge faster than ICA. The speed of the convergence of PICA is better than that of sequential method and our algorithms converge to the best result faster than those of the sequential methods. Other good points of our algorithms are the stability of them that are illustrated in figures 6, 7, 8 and 9 and the assurance of the PICA results. We illustrate that the results and the convergence, when used by 6 processors, are better than when we use 2 processors; of course, both of them are better than that of sequential methods. There are two other criteria: speed up and efficiency. The speed up values are shown in tables 3, 4 and 7. We can conclude that when we use more processors we can obtain better results, but it is essential to illustrate how many processors are useful? This answer is obtained from the efficiency values. We have illustrated that the speed up of PICA is better than that of ICA and it has a very good advantage, but now we need to discuss another view of parallelization, which is the efficiency diagram. The results of this diagram are very fantastic

and illustrate that PICA is suitable and can be used to solve and optimize the complex and big problems that have a widespread search space area. We know that the speed up of 6-processor PICA is higher than that of 2-processors, and the efficiency of 6-processor PICA is higher than that of 2-processor, so the 6-processor method is more efficient than the 2-processor one, but both of them are very efficient because their efficiency is bigger than 1, and this illustrates that PICA is an extremely good and efficient method.

We have used a different gap of migration and found good results. When we increase the migration gap we obtain better results because it is a fact that migration is useful, but in different papers of other parallel EC methods the communication time of migration is an important parameter that has important influence on the run time of algorithm, and it has side effects in the other methods as a balance between the migration rate and communication time should be created. But in our algorithm, the communication time is very low because, in each migration time, only one country (imperialist) moves to another processor; so, the communication time is little.

As a result, we have illustrated that the Multi-population PICA is a suitable method and we can obtain better results by increasing the number of processors.

E. Comparison between PICA and Parallel ABC

The proposed parallel algorithm was evaluated on three well-known benchmark functions [8]. In Multi-Population PICA we have created 160 independent countries in all processor and each processor has 8 imperialists. We tested it on Plat2 and the migration interval was done every 100 decades. The connection topology was the ring.

We have implemented our algorithms and their parameters like PABC and run them in similar conditions and compared them in table 3 and 6. We have illustrated that the results of Multi-Population PICA are better than that of PABC. A has a better SD value.

TABLE III. SPEEDUP AND EFFICIENCY VALUES ON f_8 BY MULTI-POPULATION ICA AND PABC

Number of threads	Multi-Population PICA		PABC[8]	
	Speed up	Efficiency	Speed up	Efficiency
	2	11.8	5.9	1.990
3	14.3	4.7	2.965	0.988
4	16.1	4.0	3.934	0.983

Multi-Population PICA has the best mean value and SD value. The Main factors for this success are migration in a small gap and the parallel characteristic of ICA that works like a parallel method and we have

succeeded to create a coarse grain parallel method while using all the parallelized potential of the ICA.

We have shown that PICA is more successful than PABC and it obtains better results. We can't compare the number of evaluation fitness function because there is not any information about it in [8].

The results of tables 3, 4, 5, 6 and 7 illustrate that our algorithms are efficient and suitable and can be used to solve different complex problems. We have shown that the increase of processors' number is efficient and obtains better speedup and efficiency values.

Now, we need to compare the speed of PICAs and PABC, but our platform is different from PABC platform; so, we cannot compare them in a normal way and we should use the speed up for comparison. We know from [20][21][22] that ICA is faster than ABC and if our algorithms have higher speed up values then we can claim that our algorithms are faster than that of PABC. We have tested our algorithms on Griewank on different numbers of processors, 2, 3 and 4 [8]. Our algorithms are faster than PABC. Therefore, we can claim that PICAs are faster than PABC.

F. Comparison between PICA and GPU based on PSO-TM

In this section, we want to compare the PICA and a parallel PSO-TM based on GPU. Our algorithms implemented on platform 1. We ran our algorithm on two different population sizes: 1024 and 8192 on four processors.

We know from [20][21][22] that the sequential ICA is faster than the sequential PSO and we can compare the speed of the PICA and the GPU PSO-TM with a comparison between their speed up. We ran algorithms on Akley, Rastrigin and Rosenbrock. In table 4, we have illustrated that the speed ups of our algorithm are higher than that of GPU PSO-TM and we can claim that our algorithm is faster than the GPU PSO-TM.

TABLE IV. VALUES OF SPEEDUP AND EFFICIENCY ON SOME BECHMARKS BY PICAS AND GPU PSO-TM

	Multi-population PICA		GPU PSO-TM[25]	
	N=1024	N=8192	N=1024	N=8192
	Speedup	Speedup	Speedup	Speedup
Akley	12.2	16.4	8.2	14.6
Rastrigin	18.1	28.9	16.6	25.5
Rosenbrock	11.6	15.2	8.2	16.9

G. Comparison between PICA and C-PPSO

In this section, we want to compare the PICA and a Coarse grain Parallel PSO-TM (C-PPSO). We have tested our algorithms on a similar state and each experiment was repeated 30 times and the maximum iteration number was 10000 for all functions. In the C-PPSO, we investigated the coarse-grain models with

different numbers of subpopulations and the entire population consisted of 100 individuals. The results of the C-PPSO model were compared with a standard PSO.

It is shown, in table 7 that our algorithms have higher speed up and efficiency values and we can claim that our algorithms are faster and more efficient than that of C-PPSO.

V. CONCLUSION

In this paper, we introduced a parallel methods of ICA and implemented it with MPI instructions (Multi-Population PICA). We also tested them on two different platforms, on four mathematical benchmarks, and compared them with the sequential ICA, PABC, Coarse Grain parallel PSO, and GPU-Based PSO-TM. We have found some considerable results that show PICAs are very efficient in solving different kinds of complex problems and they are faster and more efficient than other methods. We have illustrated in figures 6, 7, 8 and 9 that our algorithms are stable and the speedup illustrated in figures 2, 3, 4 and 5 showed that our algorithms convergence to the best results,

that is they are faster than the other methods in the lower iteration.

This fact is derived from the characteristic of the ICA and the ability of the migration operation. It also evaluated less fitness; and obtained suitable run times. In figures 6, 7, 8 and 9, we have illustrated the stability diagrams of Multi-Population ICA; their results have proven to show that our parallel methods truly work. In addition, we compared the PICA with the PABC, GPU Based PSO-TM and C-PPSO and utilized four mathematical benchmarks. We illustrated table 6 and compared the accuracy of the results, it is illustrated that the results of the Multi-Population PICA are more accurate than that of the PABC.

We have compared the speed and efficiency of PICAs, GPU-Based PSO-TM and C-PPSO in tables 4 and 7 illustrating that PICAs are faster than the others and they converge faster than the other methods in the lower iteration.

As a result, we claim that the PICA methods are so fast and accurate that they can be used to solve and improve complex problems, and they try to obtain the best results to the problems. In this article, we have illustrated that our algorithms achieve a super-linear performance.

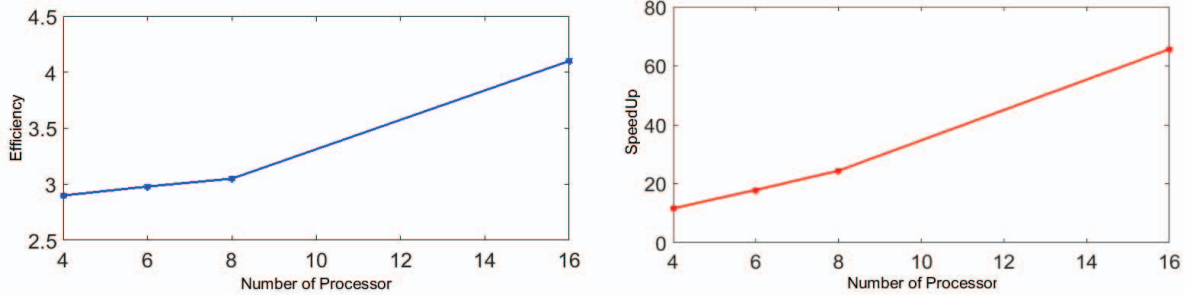


Figure 10. Speed up diagram and Efficiency diagram for f_4

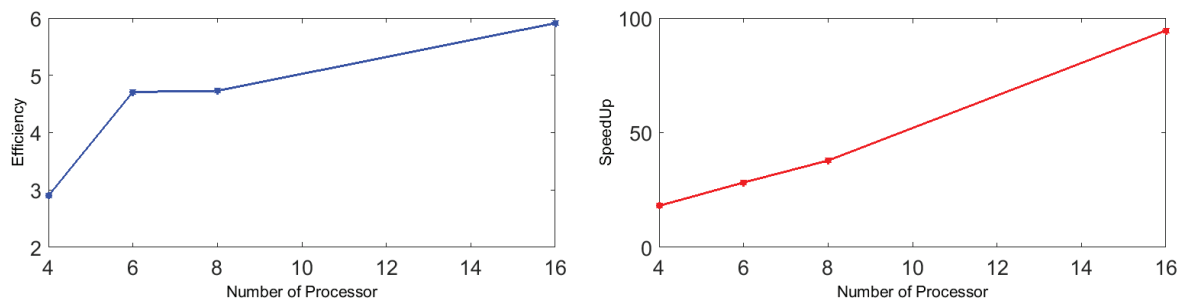


Figure 11. Speed up diagram and Efficiency diagram for f_5

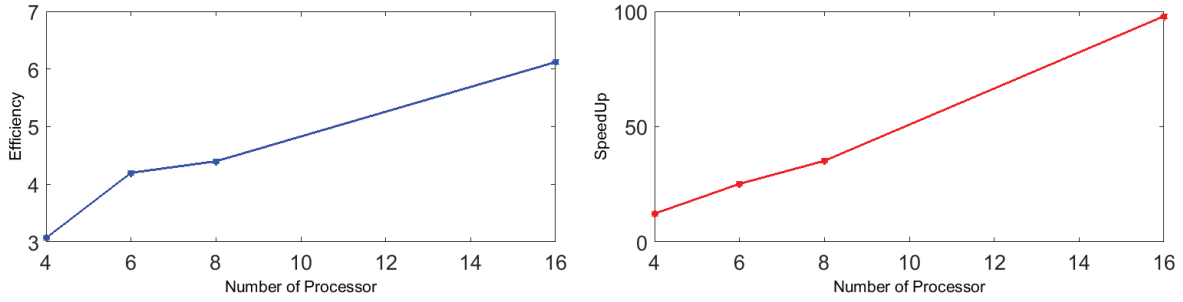


Figure 12. Speed up diagram and Efficiency diagram for f_6

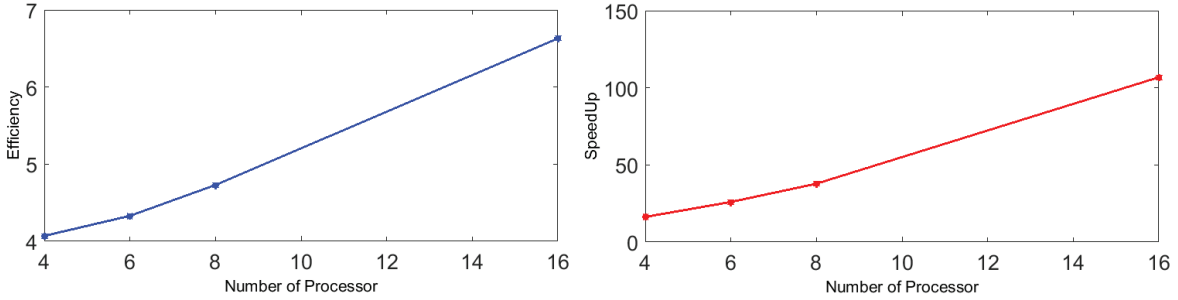


Figure 13. Speed up diagram and Efficiency diagram for f_8

TABLE V. STATISTICAL TABLE OF MULTI-POPULATION PICA ON f_4, f_5, f_6 and f_8

	N	Mean	SD	STE	Median	Worst	Best
f_4	30	0.0070844333333333	0.009400134634705	0.001716221927671	0.0015875000000000	0.0281450000000000	0
f_5	30	0.0015893666666667	0.002425782099026	428851917428874e-004	0	0.0077240000000000	0
f_6	30	8.014333333333334e-004	0.001179387178196	2.153256538433475e-004	0	0.0033260000000000	0
f_8	30	4.893666666666666e-004	8.566241876403997e-004	1.563974636250617e-004	0	0.0029420000000000	0

TABLE VI. RESULTS OBTAINED FOR THE PICA AND PABC ALGORITHMS ON SOME BENCHMARKS FUNCTION

f	D	MCN	PICA				PABC[8]			
			Multi-Population				P=4		P=16	
			P=4		P=16		Mean	SD	Mean	SD
		Mean	SD	Mean	SD	Mean	SD	Mean	SD	
f_4	30	2000	1.876391E-02	2.964761E-02	4.823789E-03	5.213874E-03	2.182352E-02	3.250047E-02	2.282869E-02	2.585128E-02
f_5	30	2000	1.862283E-16	4.732892E-17	1.722862E-16	4.378971E-17	1.946071E-16	4.615336E-17	1.931904E-16	5.386725E-17
f_8	30	2000	3.927344E-18	5.668102E-19	3.367451E-18	4.962713E-19	4.896980E-18	7.036649E-19	4.756034E-18	6.995602E-19

TABLE VII. VALUES OF SPEEDUP AND EFFICIENCY ON SOME BENCHMARKS BY PICAS AND C-PPSO

	Multi-Population PICA						C-PPSO[26]					
	CPU=4		CPU=6		CPU=8		CPU=4		CPU=6		CPU=8	
	Speed up	Efficiency	Speed up	Efficiency	Speed up	Efficiency	Speed up	Efficiency	Speed up	Efficiency	Speed up	Efficiency
Rosenbrock	11.6	2.90	17.88	2.98	24.4	3.05	3.9572	0.9893	5.7793	0.9632	5.9724	0.7465
Rastrigin	18.1	4.52	28.26	4.71	37.84	4.73	3.9580	0.9895	5.7774	0.9629	7.7703	0.9713
Griewank	16.3	4.07	25.32	4.22	34.08	4.26	3.9114	0.9778	5.9128	0.9855	7.3851	0.9231

REFERENCES

- [1] H. Lotfi, A. Boroumandnia and Sh. Lotfi, "Task Graph Scheduling in Multiprocessor Systems Using a Coarse Grained Genetic Algorithm," 2nd International Conference on Computer Technology and Development, IEEE, 2010.
- [2] H. Lotfi, Sh. Lotfi and A. Boroumandnia, "Task Graph Scheduling in Multiprocessor Systems Using a Two Population Genetic Algorithm," International Conference on Software and Computing Technology, IEEE, 2010.
- [3] E. Cantú-Paz, "A Survey of Parallel Genetic Algorithms," Department of Computer Science and Illinois Genetic Algorithms Laboratory University of Illinois at Urbana-Champaign, 1997.
- [4] H. Liu, P. Li and Y. Wen, "Parallel Ant Colony Optimization Algorithm," World Congress on Intelligent Control and Automation, China, 2006.
- [5] R. Parpinelli, C. Benitez and S. Lopes, "Parallel Approaches for the Artificial Bee Colony Algorithm," Handbook of Swarm Intelligence, vol. 8, pp. 329-345, 2010.
- [6] L. Vanneschi, D. Codecasa and G. Mauri, "A Comparative Study of Four Parallel and Distributed PSO Methods," New Generation Computing, vol. 29, pp. 129-161, 2011.
- [7] J. Digalakis and K. Margaritis, "A Parallel Memetic Algorithm for Solving Optimization Problems," 4th Metaheuristics International Conference, Parallel Distributed Processing Laboratory, Greece, 2001.
- [8] H. Narasimhan, "Parallel Artificial Bee Colony (PABC) Algorithm," World Congress on Nature & Biologically Inspired Computing, pp. 306-311. IEEE, 2009.
- [9] A. Majd, Sh. Lotfi and G. Sahebi, "Review on Parallel Evolutionary Computing and Introduce Three General Framework to Parallelize All EC Algorithms," The 5th Conference on Information and Knowledge Technology, IEEE, pp. 61-66, 2013.
- [10] E. A. Gargari, and C. Lucas, "Imperialist Competitive Algorithm: An Algorithm for Optimization Inspired by Imperialistic Competition," Congress on Evolutionary Computation, IEEE, 2007.
- [11] H. Narasimhan, "Parallel Artificial Bee Colony (PABC) Algorithm," World Congress on Nature & Biologically Inspired Computing (NaBIC), IEEE, 2009.
- [12] Z. Yang and B. Yu, "A Parallel Ant Colony Algorithm for Bus Network Optimization," Computer-Aided Civil and Infrastructure Engineering, vol. 22, pp. 44-55, 2007.
- [13] E. Alba, "Parallel Evolutionary Algorithms Can Achieve Super-Linear Performance," Information Processing Letters, vol. 82, pp. 7-13, ELSEVIER, 2002.
- [14] L. Vanneschi, D. Codecasa and G. Mauri, "A Comparative Study of Four Parallel and Distributed PSO Methods," New Generation Computing, vol. 29, pp. 129-161, Ohmsha Ltd. and Springer, 2011.
- [15] J. Digalakis and K. Margaritis, "A Parallel Memetic Algorithm for Solving Optimization Problems," 4th Metaheuristics International Conference, MIC, 2001.
- [16] Z. Yang and B. Yu, "A Parallel Ant Colony Algorithm for Bus Network Optimization," Computer-Aided Civil and Infrastructure Engineering, vol. 22, pp. 44-55, 2007.
- [17] E. C. G. Wille and E. Y. H. S. Lopes, "Discrete Capacity Assignment in IP networks using Particle Swarm Optimization," Applied Mathematics and Computation, vol. 217, pp. 5338-5346, ELSEVIER, 2011.
- [18] A. Mousa, W. Wahed and R. Allah, "A Hybrid Ant Colony Optimization Approach Based Local Search Scheme for Multi Objective Design Optimizations," Electric Power Systems Research, vol. 81, pp. 1014-1023, ELSEVIER, 2011.
- [19] P. Y. Yin, S. S. Yu, P. P. Wang and Y. T. Wang, "A Hybrid Particle Swarm Optimization Algorithm for Optimal Task Assignment in Distributed Systems," Computer Standards & Interfaces, vol. 28, pp. 441-450, ELSEVIER, 2006.
- [20] H. Bahrami, K. Faez and M. Abdechiri, "Imperialist Competitive Algorithm using Chaos Theory for Optimization (CICA)," 12th International Conference on Computer Modeling and Simulation, 2012.
- [21] M. Abdechiri, K. Faez and H. Bahrami, "Adaptive Imperialist Competitive Algorithm (AICA)," 9th IEEE International Conference on Cognitive Informatics (ICCI), 2010.
- [22] H. Bahrami, M. Abdechiri and M. Meybodi, "Imperialist Competitive Algorithm with Adaptive Colonies Movement," IJ. Intelligent Systems and Applications, vol. 2, pp. 49-57, 2012.
- [23] E. Alba, F. Luna, A. J. Nebro and J. M. Troya, "Parallel Heterogeneous Genetic Algorithms for Continuous Optimization," Parallel Computing, vol. 30, pp. 699-719, ELSEVIER, 2004.
- [24] K. Weinert, J. Mehnen and G. Rudolph, "Dynamic Neighborhood Structures in Parallel Evolution Strategies," Complex Systems Publications, vol. 13, pp. 227-243, 2002.
- [25] Y. Zhou and Y. Tan, "Particle Swarm Optimization with Triggered Mutation and its Implementation Based on GPU," Proceedings of the 12th annual conference on Genetic and evolutionary computation, ACM, pp. 1-8, 2011.
- [26] A. Basturk, R. Akay and A. Kalinli, "Comparison of fine-grained and coarse-grained parallel models in particle swarm optimization algorithm" 2nd World Conference on Information Technology (WCIT), 2011.
- [27] A. Majd and G. Sahebi, "A Survey on Parallel Evolutionary Computing and Introduce Four General Frameworks to Parallelize all EC Algorithms and Create New Operation for Migration," Journal of Information and Computing Science, vol. 9, pp. 97-105, 2014.


Paper II

Parallel Imperialist Competitive Algorithms

A. Majd, G. Sahebi, M. Daneshtalab, J. Plosila, Sh. Lotfi, H. Tenhunen

SPECIAL ISSUE PAPER

Parallel imperialist competitive algorithms

Amin Majd¹  | Golnaz Sahebi⁴ | Masoud Daneshtalab² | Juha Plosila⁴ |
Shahriar Lotfi³ | Hannu Tenhunen⁴

¹Åbo Akademi University, 20500 Turku, Finland
²Mälardalen University, 721 23 Västerås, Sweden
³University of Tabriz, Tabriz 51666 16471, Iran
⁴University of Turku, 20500 Turku, Finland

Correspondence

Amin Majd, Åbo Akademi University, 20500 Turku, Finland.
Email: amin.majd@abo.fi

Summary

The importance of optimization and NP-problem solving cannot be overemphasized. The usefulness and popularity of evolutionary computing methods are also well established. There are various types of evolutionary methods; they are mostly sequential but some of them have parallel implementations as well. We propose a multi-population method to parallelize the Imperialist Competitive Algorithm. The algorithm has been implemented with the Message Passing Interface on 2 computer platforms, and we have tested our method based on shared memory and message passing architectural models. An outstanding performance is obtained, demonstrating that the proposed method is very efficient concerning both speed and accuracy. In addition, compared with a set of existing well-known parallel algorithms, our approach obtains more accurate results within a shorter time period.

KEYWORDS

evolutionary computing, ICA, multi-population, parallel approaches, parallel programming, optimization, super-linear performance

1 | INTRODUCTION

Optimization algorithms can be divided into 2 categories: heuristic and metaheuristic methods. Heuristic algorithms are problem dependent and are often greedy and prone to get stuck in local optima, failing to obtain the global optimum or even a near-optimal solution. Metaheuristic methods such as tabu search, simulated annealing, and genetic or memetic algorithms are problem-independent techniques or frameworks that improve performance of a heuristic search by allowing more thorough exploration of the search space and avoiding local optimum traps.

Computability is a significant challenge especially in NP-hard problems; there are no guarantees that such problems can be solved in a satisfactory manner in a limited time. Several techniques have been proposed to improve solving of NP-hard problems. Among these, evolutionary computing (EC) methods are the most prominent and popular. The EC methods are useful for solving various kinds of problems. For instance, well-known genetic algorithms (GA) are very suitable for discrete problems.¹ They are population-based search methods that mimic the process of natural selection and evolution, as some characteristics of this process can be utilized in optimization problems. Particle swarm optimization (PSO) is another population-based evolutionary method that mimics the flocking behavior of birds when they migrate from a place to another.²

EC methods can enhance solving of different problems but there are some disadvantages associated with them. For example, if the search space is very large, it is possible that an evolutionary algorithm does not converge towards the global optimum or even to near-optimal solutions. To improve the outcome in such cases, the initial population should be increased. The execution time of algorithms is another challenge in this area; it can be intolerably high in some cases. Parallel approaches provide a viable means to enhance the quality of results and reduce time of obtaining results.

Previously, several parallel EC techniques have been proposed to achieve better results (eg, parallel ant colony optimization (PACO),³ parallel genetic algorithm,¹ parallel ABC (PABC),⁴ parallel memetic algorithm,⁵ and parallel PSO⁶). According to the works of Digalakis and Margaritis and Alba,^{7,8} some parallel EC methods can even achieve super-linear performance. Super-linear performance means that parallel efficiency, ie, speedup per processor, is higher than one. Normally, it is between zero and one.

Among the EC methods, the Imperialist Competitive Algorithm (ICA)⁹ is an efficient approach for continuous problems. In this paper, a parallel implementation of the ICA (PICA – Parallel ICA) is proposed. We extend our previous works (see the works of Majd et al^{9,10}) by presenting two different parallel implementations of PICA (multi-population and master-slave) to improve the speed and accuracy. The results are extensively analyzed

with eight benchmarks and three case studies. The PICA is based on a multi-population (coarse-grain) strategy, and to the best of our knowledge, this is the first attempt towards parallelizing the ICA. We evaluate the PICA under eight mathematical benchmarks, showing that a super-linear performance is achieved. The experimental results are obtained through simulations on two different computer platforms, using Message Passing Interface (MPI) instructions and a ring connected system model.

The remainder of this paper is organized as follows. In Section 2, a review of different parallel evolutionary algorithms is presented, and in Section 3, parallel master-slave and multi-population implementations of the ICA are introduced. The experimental results are provided in Section 4, and concluding remarks are given in Section 5.

2 | REVIEW OF PARALLEL EVOLUTIONARY ALGORITHMS

In this section, some of the parallel EC methods and their common features, which have been utilized in our work, will be explored.

2.1 | Parallel genetic algorithms

Genetic algorithms are population-based search methods that mimic the processes of natural selection and evolution, as some characteristics of these processes can be utilized in solving optimization problems. Each GA has an initial population (several random chromosomes each of which is an individual) and executes frequent operations such as selection, crossover, mutation, and replacement. All operations are repeated until reaching a suitable result or ending in a certain generation. In multi-population methods,¹ there are several processors each of which has independent populations, and each processor runs a simple GA. After a certain number of generations, all processors will stop and send some chromosomes (migration operation) with a certain strategy, eg, best or worse, and share the results of solutions among each other. In this method, some of the important parameters are: migration rate (number of countries that migrate at each migration time), migration gap (number of iterations between two migration events), and interconnection topologies.

Master-slave is another approach used for different problems, eg, solving a task graph scheduling with coarse-grain GA like in the works of Lotfi et al.^{11,12} In this method, one processor is assigned as a “master” to do the important operations of GA, such as crossover, mutation, replacement, and selection, whereas the other processors that are called “slaves” evaluate the fitness function and send back the results to the master processor. The fitness function is an equation that defines the quality of each chromosome. These methods can be implemented either synchronously or asynchronously. In the synchronous method, the master processor sends tasks to slave processors and waits until it receives results of all tasks from slave processors. In the asynchronous method, the master processor continues its work without waiting for results of slave processors.¹³

Fine-grain methods are suitable for parallel computing with a massive number of processors where each processor is able to communicate with the adjacent processors and each individual can recombine with each individual on the neighborhood of processors. The execution time of this method is considerable but the resource (such as number of processors and communication rate) footprint is too high.

Hybrid GAs are compound methods composed of 2 levels: the upper level uses a multi-population method and the lower level utilizes either a multi-population, master-slave, or fine-grain method. This approach is more efficient and faster than other methods because it can exploit the strengths of different methods at the different levels of hierarchy.

2.2 | Parallel ant colony optimization

Ant colony optimization (ACO) is a technique for approximate optimization.¹⁴ The inspiring source of ACO algorithms are real ant colonies. More specifically, ACO is inspired by the ants’ foraging behavior. At the core of this behavior is the indirect communication between the ants by means of chemical pheromone trails, which enables them to find short paths between their nest and food sources. This characteristic of real ant colonies is exploited in ACO algorithms in order to solve, eg, discrete optimization problems.^{3,15} There are two parallel implementations of ACO: PACO and PACO-CGD.

PACO is used on different optimization problems and is based on multi-population method, where each processor has an independent population and runs the sequential ACO independently. After a certain number of iterations, a processor sends some useful information to other processors, which is called “migration.” In PACO-CGD, the constructor graph decomposes into smaller parts and each part is sent to a processor, and then each processor can run the ACO method by itself; the execution time of this approach is better than that of PACO.¹³

2.3 | Parallel artificial bee colony algorithm

Artificial Bee Colony (ABC) is motivated by the intelligent behavior of honey bees. It is as simple as particle swarm optimization (PSO) and differential evolution (DE) algorithms and uses only common control parameters such as colony size and maximum cycle number.¹⁶ ABC, as an optimization

tool, provides a population-based search procedure in which individuals called food positions are modified by the artificial bees as time passes, and the bees' aim is to discover the places of food sources with high nectar amounts and finally the one with the highest nectar amount. In the ABC system, artificial bees fly around in a multidimensional search space and some (employed and onlooker bees) choose food sources depending on the experience of themselves and their nest mates and adjust their positions accordingly. Some (scouts) fly and choose the food sources randomly without using experience. If the nectar amount of a new source is higher than that of the previous one in their memory, they memorize the new position and forget the previous one. Thus, the ABC system combines local search methods carried out by employed and onlooker bees with global search methods, managed by onlookers and scouts, attempting to balance exploration and exploitation processes. Artificial bee colony can be parallelized in three ways: based on either a master-slave, multi-population, or hybrid method. In the master-slave method for ABC, similar to the master-slave method for GA, one processor is assigned as a master in order to run the repetitive operations like an evaluation operation.

In multi-population ABC, each processor has an independent population and runs the sequential ABC method on its population with parameters such as migration gap, migration rate, and network topology.

Hybrid ABC is a mixed method that, in the high level, uses the multi-population method, and in the low level, exploits the master-slave method, which works similarly to the hybrid GA.¹³

2.4 | Parallel particle swarm optimization

Theory of particle swarm optimization (PSO) has been advancing rapidly. PSO has been used by many applications on several problems. The algorithm of PSO emulates the behavior of animal societies that do not have any leader in their group or swarm, such as bird flocking and fish schooling. Typically, a flock of animals that has no leaders will find food randomly, following one of the members of the group that has the closest position with respect to a food source (potential solution). The flocks achieve their best condition simultaneously through communication among members who already have a better situation. PSO is used for continuous problems, whereas parallel PSO is an efficient solution for optimal task assignment in distributed systems,¹⁷ which can be implemented with two parallel techniques: MPSO and MRPSO.

Multi-population PSO works like other multi-population methods where each processor has a different population and runs a simple PSO on its population independently; a migration operation is also available.

MRPSO is an improved version of MPSO that adds an extra component to MPSO, called a repulsive component. This component in each processor tries to make a diverse population. Particles that migrate between the swarms should be as different as possible from the particles already contained in these swarms. The high degree of diversity in EC methods is very useful and helps obtaining better results.¹³

2.5 | Parallel memetic algorithms

Memetic algorithms (MAs) are population-based and heuristic search approaches for optimization problems similar to GAs. GAs, however, rely on the concept of biological evolution but MAs mimic cultural evolution. Parallel MAs are implemented as a coarse-grain approach called PARME. It is used on optimization problems in the work of Vanneschi.¹⁸

PARME is a multi-population method that uses an independent population in each processor that runs MA independently. There is a big difference between PARME and the other aforementioned methods like parallel GAs. The master processor controls the behavior of other processors and creates an operation table in each iteration and sends it to other processor nodes. This operation table has the values of critical parameters such as values of the best and the worst populations. The table will change in each iteration.¹³

3 | PROPOSED METHODS

The Imperialist Competitive Algorithm (ICA), introduced by Gargari and Lucas,¹⁹ was inspired by imperialistic competition as its name suggests. The ICA belongs to the class of evolutionary algorithms and is meant for solving linear and nonlinear NP-complete optimization problems. It is a population-based method in which each possible solution is a country, corresponding to the chromosome concept in a genetic algorithm. The algorithm first generates a set of countries, ie, the initial population. Then, all countries are divided into two types: imperialist states and colonies. Imperialistic competition is the main instrument of this algorithm, and the expectation is that the colonies converge to the global minimum of the cost function, or at least very close to this minimum. The initial sorting of the countries is based on their fitness function values. The best countries are selected to be the imperialists, and the rest of the countries form the colonies of these imperialists (Figure 1, step 1). After dividing all colonies among the imperialists, these colonies start moving toward their relevant imperialist countries. This takes place by revolution and assimilation operations (Figure 1, step 2). In each iteration, two random real numbers varying between zero and one are generated for every colony. Then, these values are compared with the predetermined assimilation (ie, Zeta¹⁹) and revolution probabilities (rates). If the random numbers are lower than these probabilities, the procedure of assimilation or revolution is performed. In the next step, the ICA computes the power of each imperialist and the imperialistic competition begins. The weakest imperialist loses its weakest colony and the selected imperialist captures this colony (Figure 1, step 5).

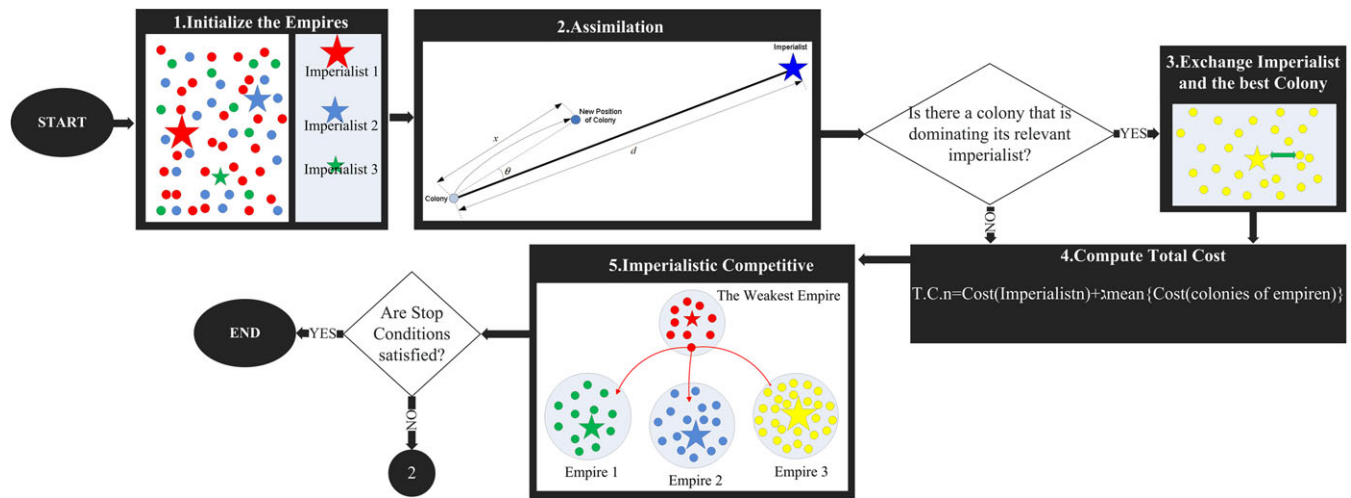


FIGURE 1 Imperialist competitive algorithm

These steps are then repeated until the termination condition is satisfied. The termination condition can be defined in different ways. For example, the ICA could be set to stop after a certain number of iterations, or when all countries have become colonies of one imperialist, ie, when there is only one empire left (see Figure 1).

The ICA is a suitable approach for a variety of optimization problems but there exist certain challenges concerning the evolutionary algorithms in general. For example, when we are dealing with a large search space, we need a large initial population to obtain acceptable accuracy of results; but in a resource constrained processing environment, we may not be able to satisfy this requirement. Also, in the case of a complex problem that needs complex computations, the run time of the algorithm will increase and may become intolerable. Therefore, we need to find an efficient approach to improve the speed, stability, and accuracy of the method.

The ICA has an inherent parallel structure, and, consequently, parallel implementation provides a viable path to enhance the performance of the algorithm. In the ICA, each imperialist and its colonies work independently, and after a decade (iteration), a colony moves to another imperialist. The behavior is similar to that of a multi-population method running on a single processor. We will look at two different approaches for PICA implementation.

3.1 | Master-slave PICA

In this method, we have several processors that are connected in a star topology using message passing but in a shared memory setup. One of the processors (P_0) is the master and the others are slaves. The master processor is the manager of the algorithm coordinating all involved operations and dividing tasks among slave processors. The slave processors perform the tasks given to them, assisting the master processor. The master processor initiates the algorithm and determines how many countries should be created by the processor P_i ($0 < i < \text{number of processors}$), and which fitness function evaluations P_i is to execute. It also divides the imperialists among the slave processors to assimilate colonies to imperialists and carries out a revolution operation on all colonies, enabling them to obtain new positions. The assimilation and revolution operations are, respectively, exploitation and exploration operations from the EC perspective. Moreover, the master processor divides the imperialists among all slave processors to calculate the total cost of each empire. It performs imperialistic competition, selects the weakest imperialist and its weakest colony, and then moves this weakest colony to the strongest (winner) imperialist. The master processor repeats the aforementioned operations until it reaches the set termination condition.

The master-slave method is especially useful for complex problems or problems that have a complex fitness function. It can provably speed up the optimization process. The implementation of the method is based on the synchronous paradigm, and its operation is in essence similar to that of the sequential ICA with improved performance due to the parallel-operating slave processors. For this method, the shared memory setup is more efficient than the message passing approach because the communication costs between the master and slave processors would be too high with the message passing scheme. The pseudocode for the master-slave PICA is presented in Figure 2.

3.2 | Multi-population PICA

The multi-population approach enables us to implement the PICA with a selective local search strategy. Our underlying system model consists of a number of processors that are connected in a ring topology and use a message passing scheme for communication. In each processor, we first

Master processor P₀:

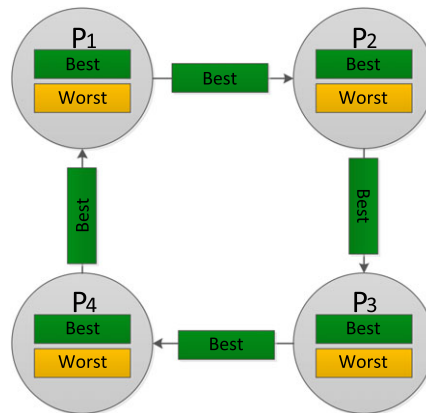
1. Start algorithm.
2. Set parameters.
3. Determine numbers of countries that each slave processors should create.
4. Divide countries (colonies) between slave processors to evaluate fitness functions.
5. Sort all colonies and select best countries to be imperialists.
6. Divide countries between slave processors to create imperialists.
7. Divide imperialists between slave processors to do assimilate operation on their colonies.
8. Divide imperialists between slave processors to do revolution operation on their colonies.
9. Divide imperialists between slave processors to calculate the total sum of their colonies.
10. Run imperialistic competition between all imperialists and select the worst colony on the worst imperialist to move it to the winner imperialist.
11. If termination condition is obtained then terminate
Else go to 7.
12. Show the best imperialist.
13. End.

Slave processor P_i (1 ≤ i ≤ number of processor) :

1. Start.
2. Create countries that master processor has determined.
3. Evaluate fitness functions of countries that master processor has determined.
4. Create an imperialist with its colonies that master processor has determined.
5. Run the assimilate operation on imperialist's colonies that the master processor has determined.
6. Run the revolution operation on imperialist's colonies that the master processor has determined.
7. Calculate the total sum of imperialists that the master processor has determined them.
8. End.

FIGURE 2 Pseudocode of master-slave PICA

initiate an independent set of countries and run the ICA independently of the other processors. Then, after some decades (iterations), the best country migrates from the processor P_i to the processor P_{i+1} (for all i) and replaces the worse country in P_{i+1} . The migration interval varies from a run to another. All migrations between the processors take place synchronously (simultaneously), and, therefore, the numbers of the countries in all processors are equal after each migration event.⁹ The behavior of the multi-population PICA is illustrated in Figure 3, and the pseudocode is presented in Figure 4. The sparse connected ring topology enables a low migration rate due to the simple routing scheme and the short (single-hop) migration distance.

**FIGURE 3** Multi-population migration operation¹⁰

```

Processor Pi:
Begin
Create independent initial countries.
Run ICA algorithm independently.
If (#iteration mod migration gap=0)do
begin
Wait when all processors arrive to this point.
Send the best country to processor () mod (# processors).
Receive a country from () mod (number of processors) and replace it with the worst country.
end;
If termination condition is obtained then terminate algorithm.
Show the best country.
End.

```

FIGURE 4 Pseudocode of multi-population PICA⁹

TABLE 1 Mathematical benchmarks

Name	Equation	Min. Value	Name	Equation	Min. Value		
f_1	G_1	$x \cdot \sin(x) + 1.1y \cdot \sin(2y)$	-18.5547	f_5	Rastrigin	$\sum_{i=1}^n x_i^2 - 10 \cos(2\pi x_i) + 10$	0
f_2	G_2	$0.5 + \frac{\sin \sqrt{x^2+y^2}-0.5}{1+0.1(x^2+y^2)}$	-0.5231	f_6	Akley	$20 + e - 20 \exp(-0.2 \sqrt{\frac{1}{D} \sum_{i=1}^D x_i^2}) - \exp(\frac{1}{D} \sum_{i=1}^D \cos(2\pi x))$	0
f_3	Sphere	$\sum_{i=1}^D x_i^2$	0	f_7	Ellipse	$\sum_{i=1}^D 10^{4 \frac{i-1}{D-1}} x_i^2$	0
f_4	Rosenbrock	$\sum_{i=1}^{n-1} 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2$	0	f_8	Griewank	$\frac{1}{4000} \sum_{i=1}^n x_i^2 - \prod_{i=1}^n \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	0

TABLE 2 Specification of platforms:

	Platform 1	Platform 2
CPU	4 x Intel Core i7 2.67 GHz	Intel Core i3-330M, 2.13 GHz
Mother board	Gigabyte EX-58 EXTREME	-
RAM	6 GB DDR3	4 GB
HDD	500 GB	300 GB
NIC	Gigabyte	-
Network switch	Cisco catalyst 3750	-
Operating system	WINDOWS XP 64-bit	WINDOWS XP 64-bit
MPI library	MPICH2 1.4	MPICH2 1.4
Compiler	Visual C++ 6.0	Visual C++ 6.0
GPU	NVIDIA GeForce 9600 GT	-

By using the multi-population PICA, we increase both the size of the overall population, ie, the total number of countries and the selection pressure, which helps obtaining more accurate results within a fewer number of iterations. Indeed, convergence to results takes place much faster than in the case of the sequential ICA, as will be demonstrated in Section 4.

4 | EVALUATION AND EXPERIMENTAL RESULTS

We use eight test functions, ie, mathematical benchmarks, presented in TABLE 1, to compare our algorithms, ie, the two implementations of PICA presented in Section 3, with the sequential ICA and some other parallel EC methods. All the considered problems are minimization problems.

In addition to the eight mathematical benchmarks, we will also have three case studies in the field of nonlinear equations. The algorithms are implemented based on both shared memory and message passing communication models. First, we compare our algorithms with the sequential ICA, followed by a comparison with a set of other well-known parallel EC methods. We utilize the MPI to parallelize our algorithms and MPICH2²⁰ to run the algorithms. In the multi-population PICA, we connect processors in a ring topology with different numbers of processors on different tests. We test our algorithms on two computer platforms, named Platform 1 (higher-end platform) and Platform 2 (lower-end platform), that are specified in Table 2.

4.1 | Comparison between multi-population parallel ICA and ICA

In this section, we compare the multi-population version of the PICA and the serial ICA. The comparison of these algorithms is based on their convergence diagram and potential speedup (Tables 3, 4, 5, 10, 11, 13, and 18). We employ 2 important benchmarks from the work of Gargari and Lucas¹⁹ in our tests, ie, f_1 and f_2 , as presented in Table 1.

In the experiment, we use two system models and two platforms (Table 2); in the case of a two-processor system model, we implement our algorithm on Platform 2, and in the case of a six-processor system model, we implement it on Platform 1. Message passing based communication is utilized in both cases. We have 100 countries and 8 imperialists in each processor. The revolution rate is set to 0.4 and the Zeta parameter to 0.1, corresponding to the exploration and exploitation rates in any EC method, respectively. We would obtain different results by varying these parameters but we choose to test our algorithms using the mentioned constant values for simplicity. The architecture is based on the ring topology.

TABLE 3 Speedup and efficiency values on f_8 by multi-population ICA and PABC

Number of threads	Multi-population PICA		PABC ¹⁶	
	Speedup	Efficiency	Speedup	Efficiency
2	11.8	5.9	1.990	0.995
3	14.3	4.7	2.965	0.988
4	16.1	4.0	3.934	0.983

TABLE 4 Values of speedup and efficiency on some benchmarks by PICAs and GPU PSO-TM²

	Multi-population PICA		GPU PSO-TM ²	
	N=1024	N=8192	N=1024	N=8192
	Speedup	Speedup	Speedup	Speedup
Akley	12.2	16.4	8.2	14.6
Rastrigin	18.1	28.9	16.6	25.5
Ellipse	11.2	14.8	7.2	12.8
Rosenbrock	11.6	15.2	8.2	16.9

TABLE 5 Values of speedup and efficiency on problems f_1 and f_2

	PICA							
	Master slave				Multi-population			
	P=2		P=6		P=2		P=6	
	Speedup	Efficiency	Speedup	Efficiency	Speedup	Efficiency	Speedup	Efficiency
G_1	1.72	0.86	4.98	0.83	12.2	6.1	67.7	9.45
G_2	1.82	0.91	5.52	0.92	14.9	7.45	73.6	12.26

TABLE 6 Statistical table of multi-population PICA on problem f_1

#P	N	Mean	SD	STE	Median	Worse	Best
2	100	-18.5350	0.0466	0.0047	-18.5544	-18.2159	-18.5547
6	100	-18.5473	0.0149	0.0015	-18.5543	-18.4892	-18.5547

The aforementioned analysis seems to indicate that the more processors we use, the better results we get. Closely related to this, let us now look into another aspect of parallelization, ie, the parallel efficiency diagram (Figures 31 to 38).

The results demonstrate that the multi-population PICA is very suitable for solving and optimizing complex problems that have a large search space. Based on the analysis, we know that the speedup of the six-processor PICA is higher than that of the two-processor PICA, and that the parallel efficiency of the six-processor PICA is higher than that of the two-processor one, so the six-processor implementation clearly outperforms the two-processor implementation. However, both of these implementations are actually very efficient because their parallel efficiency values are larger than 1. This highlights the observation that the PICA is an extremely competitive and capable method, independently of the number of processors used. With two processors, the parallel efficiency values are 6.1 and 7.45 for f_1 and f_2 , respectively, and with six processors, they are 9.45 and 12.26 for f_1 and f_2 , respectively. In other words, the results are relatively very close to each other. How many processors should be used in a real-life application depends on the application-specific requirements and constraints.

We have also examined different migration gaps (intervals) and found interesting results. Having the migration operation as part of our method is, in general, very beneficial; it improves the performance and outcome of the algorithm. However, in other parallel EC methods, the communication time of migration is an important factor that has a significant influence on the run time of the algorithm, and therefore it has negative side effects in these methods. This means in practice that a balance between the migration rate and communication time has to be found. In our algorithm, in turn, the communication time is very small because in each migration event, every processor sends a single country (the best one) to the next (adjacent) processor, as a single synchronous step.

As an overall result, we have demonstrated that the multi-population PICA is a prominent method and that we can improve the performance and the quality of results by increasing the number of processors.

TABLE 7 Statistical table of multi-population PICA on f_3, f_4, f_5, f_6, f_7 , and f_8

	N	Mean	SD	STE	Median	Worse	Best
f_3	30	0.0018104333333333	0.002970555267766	5.423467094903584e-004	0	0.0018104333333333	0
f_4	30	0.0070844333333333	0.009400134634705	0.001716221927671	0.00158750	0.028145000000000	0
f_5	30	0.0015893666666667	0.002425782099026	4.28851917428874e-004	0	0.007724000000000	0
f_6	30	8.014333333333334e-004	0.001179387178196	2.153256538433475e-004	0	0.003326000000000	0
f_7	30	0.006058700000000	0.008992364052446	0.001641773545608	0	0.028145000000000	0
f_8	30	4.893666666666666e-004	8.566241876403997e-004	1.563974636250617e-004	0	0.002942000000000	0

4.2 | Comparison between master-slave PICA and ICA

In this section, we compare the master-slave PICA with the serial ICA. Like in the previous experiment, we use two platforms (Table 2). In the case of a two-processor system model, we implement our algorithm on Platform 2, and when the model has more than two processors, we implement it on Platform 1. We utilize both message passing and shared memory architectures and use the MPI instructions with MPICH2-1.4. Furthermore, we have 100 countries and 8 imperialists in each processor. The revolution rate and the Zeta parameter are set to the constant values of 0.4 and 0.1, respectively, like in the previous experiment. In our system model, we have a master processor whose rank is 0, and the other processors are slaves. The master processor runs all the important tasks and manages the algorithm. The slave processors run all the tasks that are determined by the master processor. The architecture is based on a fully topology in which the master processor is the central node.

In the master-slave setup, the parallel implementation of an EC algorithm, when implemented synchronously, is essentially similar to the sequential implementation on a single processor from the behavioral perspective. So, the convergence speeds of the serial ICA and the master-slave PICA are the same. However, as illustrated in Table 5, the speedup of the master-slave PICA is significant (ie, clearly higher than 1) because the slave processors are computing tasks in parallel. Moreover, the speedup gets actually higher when the complexity of the problem increases, ie, the benchmark f_2 is more complex than f_1 .

We can also see that increasing the number of processors leads to better results, as well as higher speedup and parallel efficiency. However, even though the master-slave PICA is an efficient method, the multi-population PICA clearly outperforms it in this respect.

4.3 | Comparison between PICA and parallel ABC

The proposed parallel algorithms are next evaluated on a set of well-known benchmark functions that were used in the work of Narasimhan¹⁶ to analyze the PABC method. In the case of the multi-population PICA, we create 160 independent countries in every processor, and each processor has 8 imperialists. We test it on Platform 1 with a migration operation taking place every 100 decades (iterations). The connection topology is the ring. In the case of the master-slave PICA, we create 160 countries in the master processor and have 7 slave processors that run parallel operations controlled by the master processor.

In the PABC experiments presented in the work of Narasimhan,¹⁶ the size of the bee colony was chosen to be 160 with 50% employed bees and 50% onlooker bees (SN = 80). A maximum of one scout bee is produced per cycle. The PABC algorithm was run for 2000 iterations on a set of benchmark functions. The algorithm was simulated for a different number of processors. The best mean values and the standard deviation (SD) were recorded over 30 runs for 4, 8, and 16 processors.

In our experiments on the PICA, we mimic the test conditions used in the work of Narasimhan¹⁶ to enable fair comparison between the PICA and the PABC method. The results are listed in Tables 8, 9, and 12. They indicate that the multi-population PICA performs better than PABC and the master-slave PICA but the results of PABC and the master-slave PICA are almost the same. The PABC method has a better mean value but the master-slave PICA has a better SD value (in Table 8).

The multi-population PICA has the best mean and SD values, and can therefore be considered the most prominent method of the three. The main factors for this success are the efficient migration strategy and process and the inherent parallel characteristics of the ICA that seamlessly facilitate efficient parallel implementation of the algorithm. We cannot compare the numbers of fitness function evaluations because there is no information about it in the work of Narasimhan.¹⁶

We also need to compare the execution speeds of the PICA and PABC but our platform is different from the platform used in the work of Narasimhan¹⁶ for analyzing PABC. Therefore, we cannot compare the speeds directly but we need to use speedup values for indirect comparison. We know from other works²¹⁻²³ that the sequential ICA is faster than the sequential ABC method; so, if our parallel algorithms have higher speedup values, then we can claim that our proposed algorithms are faster than PABC. We have tested our algorithms on the Griewank benchmark (Table 1) with 2, 3, and 4 processors, like what was done in the work of Narasimhan¹⁶ for PABC. The speedup values are shown in Table 9. Comparing these, we can see that our methods have better speedup values. In the case of the multi-population PICA, the difference is very clear, whereas in the case

TABLE 8 Results obtained for the PICA and PABC algorithms on some benchmark functions

f	D	MCN	Master-slave P=4		PICA P=4		Multi-population P=16		P=4		PABC ¹⁶ P=16	
			Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
f ₃	30	2000	2.51354E-16	3.66295E-17	1.64297E-16	2.28976E-17	1.52341E-16	2.18497E-17	2.49479E-16	4.043068E-17	2.467389E-16	4.100729E-17
f ₄	30	2000	2.322891E-02	3.462278E-02	1.876391E-02	2.964761E-02	4.823789E-03	5.213874E-03	2.182352E-02	3.250047E-02	2.282869E-02	2.585128E-02
f ₅	30	2000	1.993628E-16	4.736478E-17	1.862283E-16	4.732892E-17	1.722862E-16	4.378971E-17	1.946071E-16	4.615336E-17	1.931904E-16	5.386725E-17
f ₈	30	2000	4.728617E-18	6.598321E-19	3.927344E-18	5.668102E-19	3.367451E-18	4.962713E-19	4.896980E-18	7.036649E-19	4.756034E-18	6.995602E-19

TABLE 9 Speedup and efficiency values on f_8 for PICA and PABC

Number of threads	Master-slave PICA		Multi-population PICA		PABC ¹⁶	
	Speedup	Efficiency	Speedup	Efficiency	Speedup	Efficiency
2	1.936	0.968	11.8	5.9	1.990	0.995
3	2.967	0.989	14.3	4.7	2.965	0.988
4	3.964	0.991	16.1	4.0	3.934	0.983

TABLE 10 Values of speedup and efficiency on problem f_3 by PICAs, multi-population GA and Neighborhood GA

Problem	Master-slave PICA		Multi-population ICA		Multi-population GA ²⁴		Neighborhood GA ²⁵	
	P=8		P=8		P=8		P=8	
G_3	Speedup	Efficiency	Speedup	Efficiency	Speedup	Efficiency	Speedup	Efficiency
	7.52	0.94	23.44	2.93	3.28	0.41	4.7	0.58

of the master-slave PICA, the difference is very small. We can therefore conclude that the multi-population PICA is the champion among the three methods by a clear margin.

4.4 | Comparison between PICA and multi-population parallel genetic algorithm

In this section, we compare both PICA versions with the multi-population PGA. The comparison is based on the benchmark *Sphere* (Table 1, function f_3). We focus on the speed of these algorithms. To compare the speeds straightforwardly, we would need to run the PICA and PGA on the same platform. To compare these algorithms without actually implementing the PGA (ie, by using the results from an earlier publication instead), we utilize another approach that is based on the speedup and parallel efficiency, like the aforementioned statements where we compared the speeds of the PICA and the PABC method. In other words, we divide the task into two steps. In the first step, we compare the sequential ICA and the sequential GA on the same platform and on the same benchmark. In the second step, we compute the speedups of the PICA and the PGA to find out which algorithm is faster. First, we know from other works²¹⁻²³ that the sequential ICA is faster than the sequential GA on the Sphere benchmark but let us assume here that their speed is the same just to be on the safe side. Second, we compute the speedup and parallel efficiency of the PICA on the Sphere benchmark; the results are given in Table 10. Then, we take the PGA's speedup and parallel efficiency values from the work of Alba et al²⁴ and insert them in Table 10. Now, we can compare the ICA and the PGA; the one that has a higher speedup and parallel efficiency is the winner. We have run our 2 PICA versions 100 times under the same conditions to obtain the results shown in Table 10.

Based on the results (Table 10), we can conclude that both PICA versions outperform the multi-population PGA by a clear margin, the multi-population PICA being superior among the 3 candidates.

4.5 | Comparison between PICA and dynamic neighborhood structures in parallel evolution strategies (Neighborhood GA)

In this section, we compare our 2 PICA versions with an approach known as the Dynamic Neighborhood Structures in parallel ES (ie, the Neighborhood GA). We use the Sphere benchmark (Table 1) and the same speedup and parallel efficiency based comparison method, as aforementioned. We run our algorithms 100 times under the same conditions.

We know that the sequential ICA converges to the best result faster than the sequential GA does¹⁹ but let us assume modestly that their speeds are equal. In Tables 10 and 11, we show the simulation results for our multi-population and master-slave PICAs and the results for the Neighborhood

TABLE 11 Values of speedup and efficiency on problem f_3 by PICAs and Neighborhood GA

Problem	Master-slave PICA		Multi-population ICA		Neighborhood GA ²⁵	
	P=4		P=4		P=4	
G_3	Speedup	Efficiency	Speedup	Efficiency	Speedup	Efficiency
	3.84	0.96	11.6	2.9	3.1	0.75

GA that are obtained from the work of Weinert et al.²⁵ By comparing the results (Tables 10 and 11), we can conclude that both PICA versions clearly outperform the Neighborhood GA from both speedup and parallel efficiency perspectives.

4.6 | Comparison between multi-population PICA and GPU-based PSO-TM

In this section, we compare the multi-population PICA with the parallel PSO-TM method implemented on a GPU.² The GPU-PSO-TM was deployed in the work of Zhou and Tan² on the CUDA platform based on an Intel Core 2 Duo 2.20 GHz CPU, 3.0 GB RAM, with the NVIDIA GeForce 9800GT display adapter and the Windows XP operating system. Our multi-population PICA is implemented on Platform 1 (Table 2). We run our algorithm on two different population sizes, ie, 1024 and 8192, using a four-processor system model with the ring topology.

Based on other works,²¹⁻²³ we know that the sequential ICA is faster than the sequential PSO, and therefore we can compare the speeds of the PICA and the GPU-PSO-TM method by comparing their speedup values like we did with some other methods in the previous subsections. We run the multi-population PICA on the Akley, Rastrigin, Ellipse, and Rosenbrock benchmarks (Table 1). The results are shown in Table 4. We can see that the speedup of the PICA is generally higher than that of the GPU-PSO-TM approach, especially with the smaller population size. From the speedup perspective, our algorithm outperforms the GPU-PSO-TM method in all tests except for 1 case. With the larger population size, the GPU-optimized PSO-TM solution becomes relatively stronger, even surpassing our multi-population PICA's speedup value in one of the tests.

4.7 | Comparison between multi-population PICA and C-PPSO

In this section, we compare the multi-population PICA with the coarse-grain parallel PSO-TM method (C-PPSO).²⁶ We have tested our PICA on the Sphere, Rosenbrock, Rastrigin, and Griewank benchmarks (Table 1) using four-, six-, and eight-processor system models (with the ring topology). Each experiment was repeated 30 times, and the maximum iteration number was 10 000 for all benchmarks. In the case of the C-PPSO approach, based on the work of Basturk et al,²⁶ we investigated the coarse-grain models with different numbers of subpopulations, and the entire population consisted of 100 individuals. The results of the C-PPSO model were compared with the results of a standard PSO to determine the speedup and parallel efficiency values.²⁶

The results shown in Table 13 demonstrate that our algorithm has higher speedup and parallel efficiency values, and therefore we can conclude that our multi-population PICA is faster and more efficient than the C-PPSO method. The difference is clear and becomes even clearer when the number of processors increases.

4.8 | Convergence

In this section, we show the convergence diagrams of the benchmarks in order to compare our results and diagrams with the results and diagrams presented in other works.^{2,16,19,24-26} These results have been obtained with different numbers of processors and different population sizes on the multi-population PICA method. The convergence diagrams for our approach are shown in Figures 5 to 12, 23, 26, and 29. After the comparison between our algorithm's convergence diagrams and those of the mentioned other works, we can claim that our algorithm converges faster than the other methods within a lower number of iterations.

4.9 | Stability

In this section, we demonstrate that our multi-population PICA is stable (producing similar results in different runs) and accurate (producing results that are near to the global optimum) by testing it on the different benchmarks and drawing the corresponding stability diagrams for it. Each stability diagram shows the behavior of the method in different runs for a given benchmark. These diagrams are illustrated in Figures 13 to 20, Figure 24, Figure 27, and Figure 30. The minor fluctuation of the graphs indicates that our algorithm is indeed stable and accurate. Moreover, the statistical results listed in Tables 6, 7, 8, 12, and 16 also indicate that the multi-population PICA is more stable and accurate than the other considered methods. In these tables, there are five important parameters. The standard deviation (STD) is a measure that is utilized to quantify the amount of variation or dispersion of a set of data values. A method cN runs. The best and the worst are the best and the worst values of all N . The mean is the average value of all the best results in all N runs. A method can be considered the most accurate one when it has the lowest values of STD and the mean and the lowest values of the best and the worst. We can conclude that our algorithm indeed is more accurate, with fewer errors, than the other considered methods.

TABLE 12 Results obtained for the PICA and PABC algorithms on some benchmarks function

<i>f</i>	D	MCN	PICA				PABC ¹⁶			
			Multi-population		P=16		P=4		P=16	
			Mean	SD	Mean	SD	Mean	SD	Mean	SD
<i>f</i> ₄	30	2000	1.876391E-02	2.964761E-02	4.823789E-03	5.213874E-03	2.182352E-02	3.250047E-02	2.282869E-02	2.585128E-02
<i>f</i> ₅	30	2000	1.862283E-16	4.732892E-17	1.722862E-16	4.378971E-17	1.946071E-16	4.615336E-17	1.931904E-16	5.386725E-17
<i>f</i> ₈	30	2000	3.927344E-18	5.668102E-19	3.367451E-18	4.962713E-19	4.896980E-18	7.036649E-19	4.756034E-18	6.995602E-19

TABLE 13 Values of speedup and efficiency on some benchmarks by PICAs and C-PPSO

Efficiency	CPU=4	Multi-population PICA				C-PPSO ²⁶						
		Speedup	Efficiency	Speedup	Efficiency	Speedup	Efficiency	Speedup	Efficiency			
Sphere	12.2	3.05	18.6	3.10	25.12	3.14	3.9029	0.9757	5.7555	0.9593	6.7519	0.8440
Rosenbrock	11.6	2.90	17.88	2.98	24.4	3.05	3.9572	0.9893	5.7793	0.9632	5.9724	0.7465
Rastrigin	18.1	4.52	28.26	4.71	37.84	4.73	3.9580	0.9895	5.7774	0.9629	7.7703	0.9713
Griewank	16.3	4.07	25.32	4.22	34.08	4.26	3.9114	0.9778	5.9128	0.9855	7.3851	0.9231

TABLE 14 Comparison results of PICA for Case 1 with other works^{25,27,28,30}

Methods	x_1	x_2	$f(x)$
PPSO ²⁷ and Gyrhan ²⁸	-0.29051455550725	1.08421508149135	4.686326815078573e-029
PPSO ²⁷ and Gyrhan ²⁸	-0.793700525984100	-0.793700525984100	1.577721810442024e-030
COA ³⁰	1.08421508149135	-0.29051455550725	4.686326815078573e-029
COA ³⁰	-0.29051455550725	1.08421508149135	4.686326815078573e-029
Ica ²⁹	1.084215081491351	-0.290514555507251	3.562200025138631e-030
Ica ²⁹	-0.793700525984100	-0.793700525984100	1.577721810442024e-030
Ica ²⁹	-0.290514555507251	1.084215081491351	3.562200025138631e-030
PICA (present study)	1.0842150814913511	-0.2905145555072514	4.9303806576313238e-032
PICA (present study)	-0.79370052598409995582	-0.79370052598409995582	3.9443045261050590e-031
PICA (present study)	-0.2905145555072514	1.0842150814913511	4.9303806576313238e-032

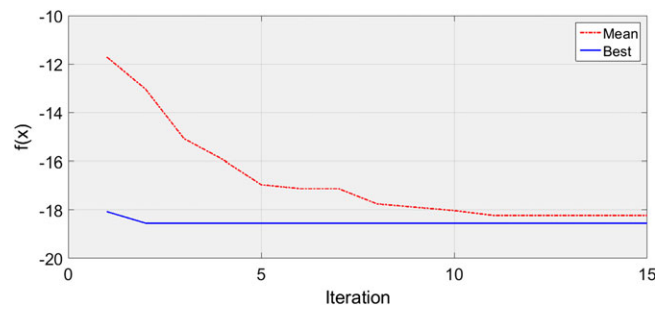


FIGURE 5 Convergence diagram of multi-population PICA with 2 processors in problem f_1

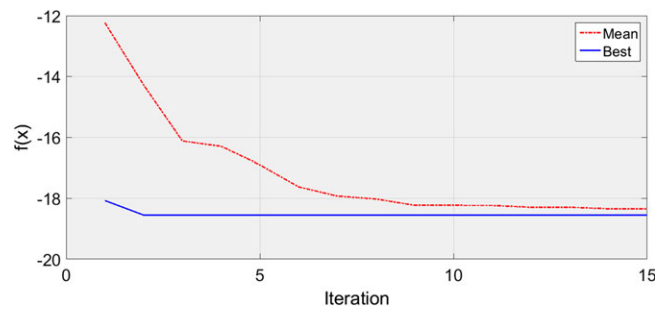


FIGURE 6 Convergence diagram of multi-population PICA with 6 processors in problem f_1

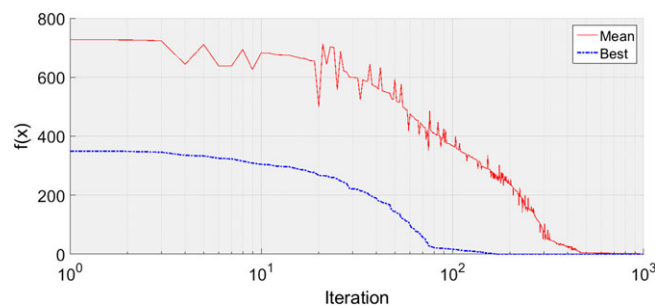


FIGURE 7 Convergence diagrams of f_3

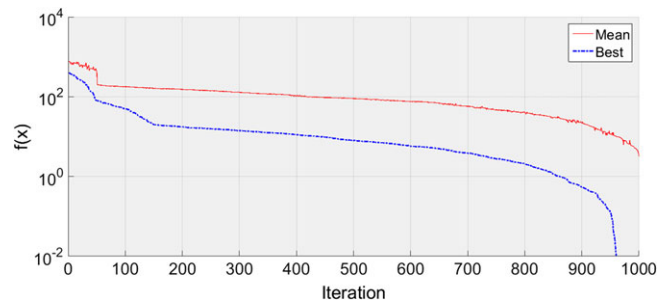


FIGURE 8 Convergence diagrams of f_4

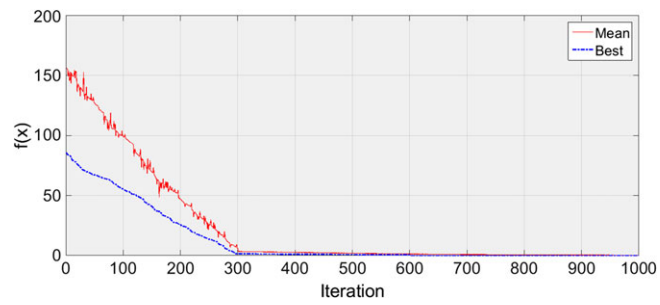


FIGURE 9 Convergence diagrams of f_5

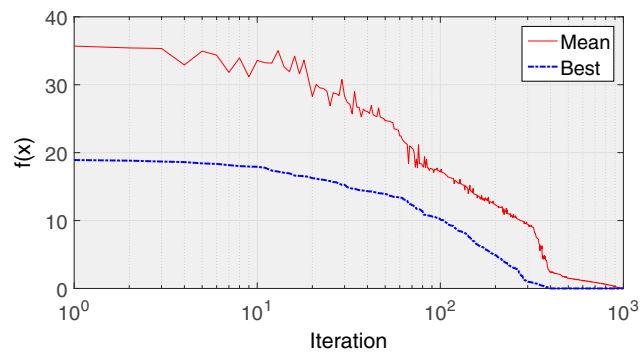


FIGURE 10 Convergence diagrams of f_6

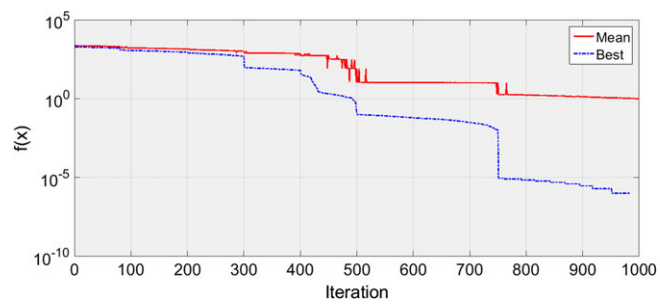


FIGURE 11 Convergence diagrams of f_7

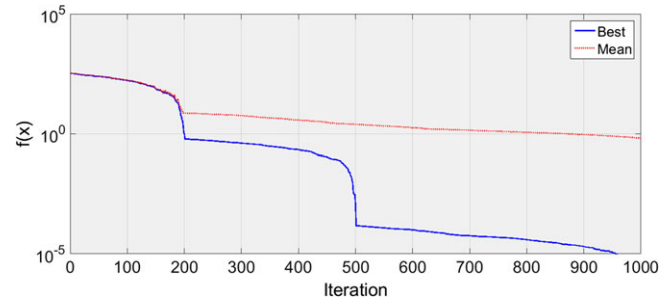


FIGURE 12 Convergence diagrams of f_8

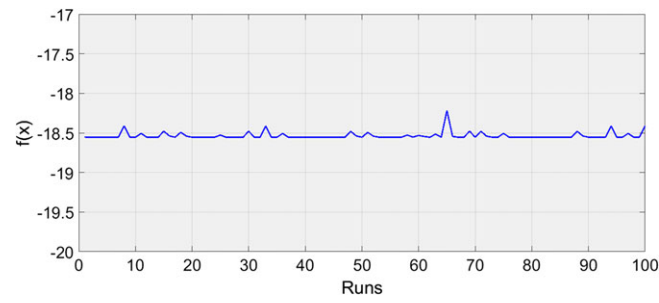


FIGURE 13 Stability diagram for 100 runs by multi-population PICA with 2 processors in problem f_1

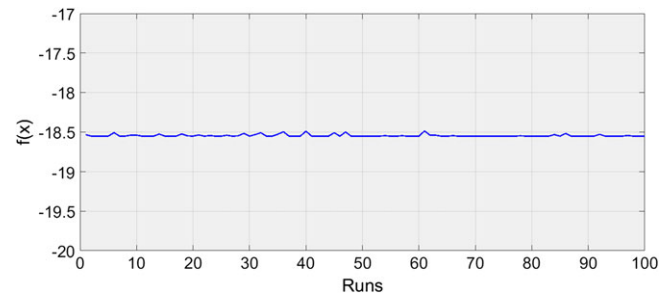


FIGURE 14 Stability diagram for 100 runs by multi-population PICA with 6 processors in problem f_1

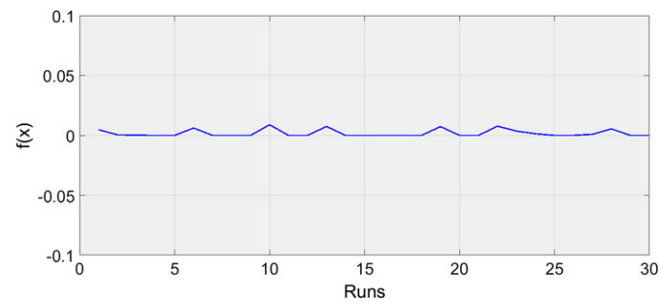


FIGURE 15 Stability diagrams of f_3

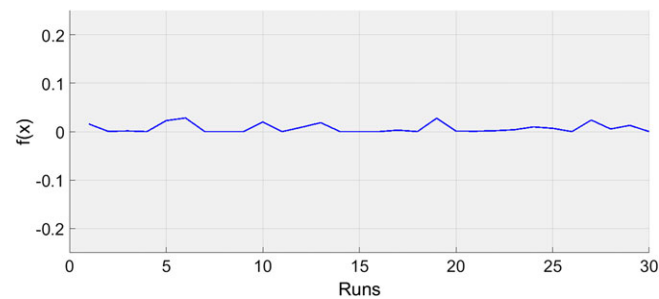


FIGURE 16 Stability diagrams of f_4

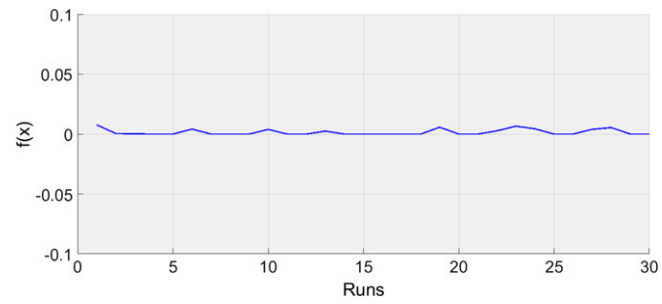


FIGURE 17 Stability diagrams of f_5

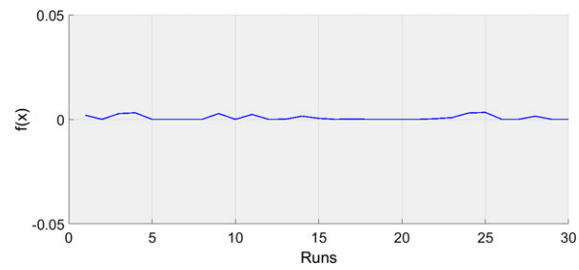


FIGURE 18 Stability diagrams of f_6

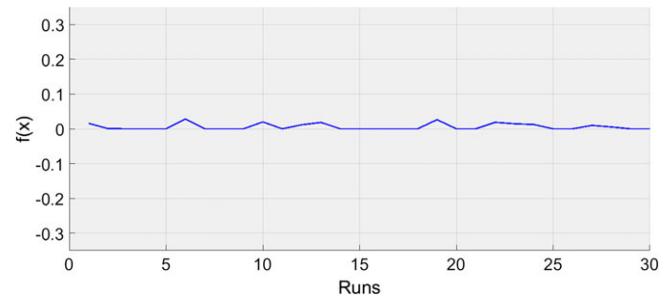


FIGURE 19 Stability diagrams of f_7

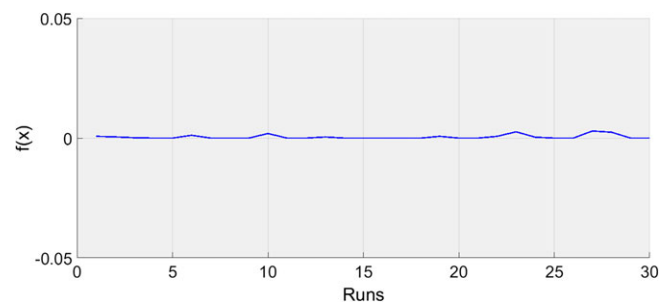


FIGURE 20 Stability diagrams of f_8

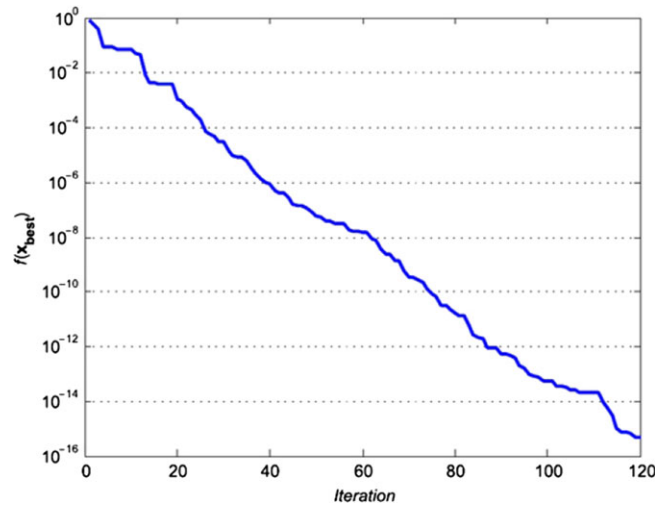


FIGURE 21 The convergence history of case 1 (from the work of Jaberipour et al²⁷)

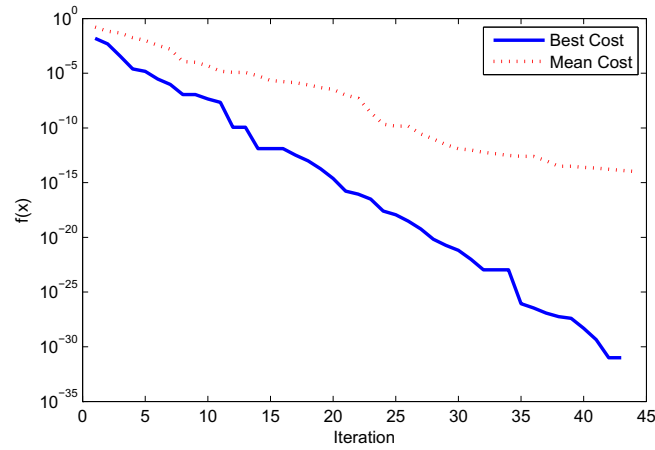


FIGURE 22 The convergence history of case 1 (from the work of Abdollahi et al²⁹)

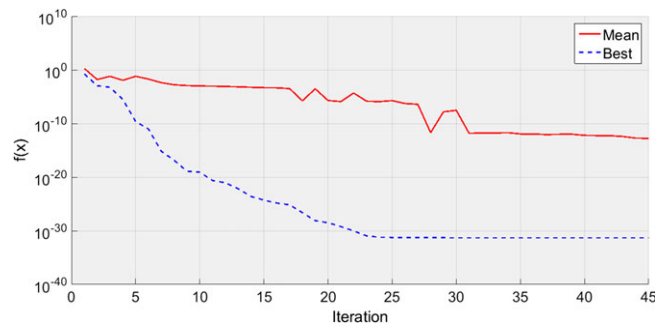


FIGURE 23 The convergence history of case 1 with PICA

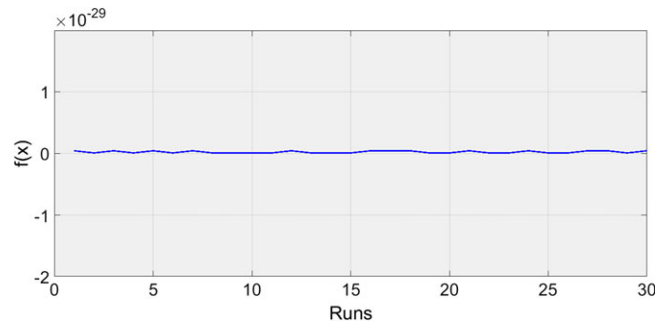


FIGURE 24 The stability chart of case 1 with PICA

TABLE 15 Comparison results of Case 2

Methods	X	Variable values	f	Functions values	F(x)
The best in the work of Wang et al ³¹	x_1	0.500432850000000	f_1	-0.000238520000000	7.693745216994211e-008
	x_2	3.141863170000000	f_2	0.000141590000000	
The best in the work of Floudas et al ³²	x_1	0.299450000000000	f_1	6.139739265609290e-007	1.014347133848949e-012
	x_2	2.836930000000000	f_2	-7.983627943186633e-007	
	x_1	0.500000000000000	f_1	2.111655261760603e-007	5.316365008296489e-012
	x_2	3.141590000000000	f_2	-2.296034435467220e-006	
The best in COA ³⁰	x_1	0.299300000000000	f_1	-7.128922385554737e-005	5.792081721117691e-009
	x_2	2.836600000000000	f_2	2.664447941302939e-005	
The best in ICA ²⁹	x_1	0.299448692495720	f_1	1.305289210051797e-012	5.631272867601562e-024
	x_2	2.836927770471037	f_2	2.284838984678572e-013	
	x_1	0.500000000000000	f_1	0	0
	x_2	3.141592653589794	f_2	0	
The best of PICA	x_1	0.29944869249092598	f_1	-1.387778780781446e-016	6.856310602018560e-032
	x_2	2.8369277704589400	f_2	2.220446049250313e-016	
	x_1	0.500000000000000	f_1	0	0
	x_2	3.141592653589794	f_2	0	

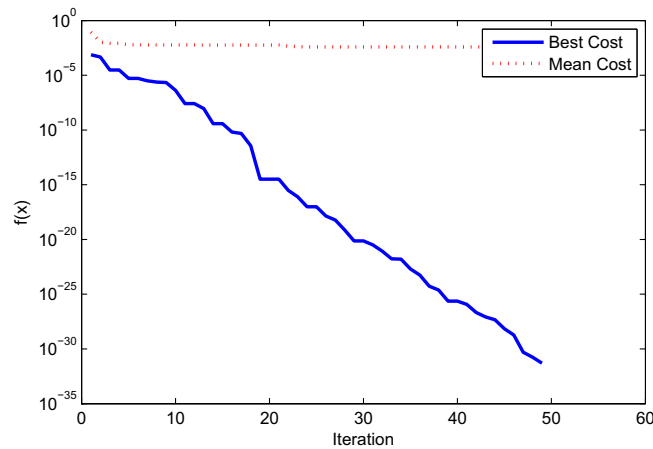


FIGURE 25 The convergence history of case 2 (from the work of Abdollahi et al²⁹)

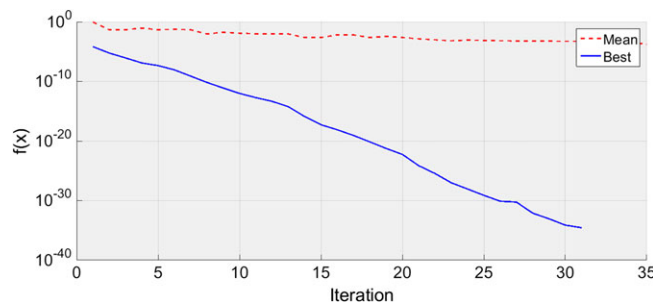


FIGURE 26 The convergence history of case 2 with PICA

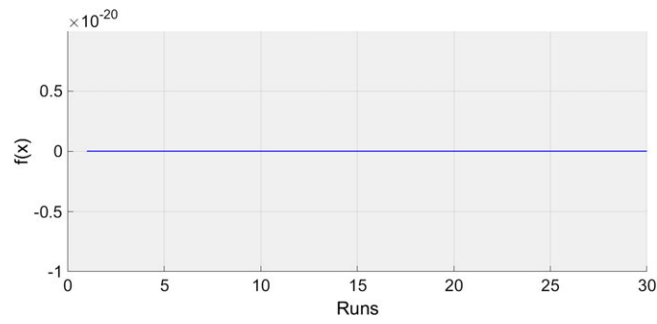


FIGURE 27 The stability chart of case 2 with PICA

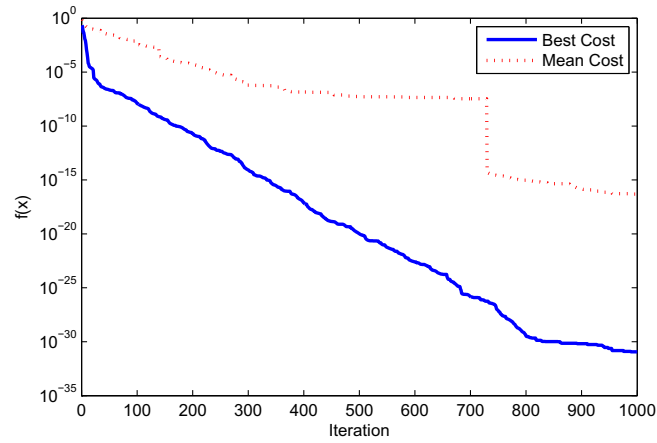


FIGURE 28 The convergence history of case 3 (from the work of Abdollahi et al²⁹)

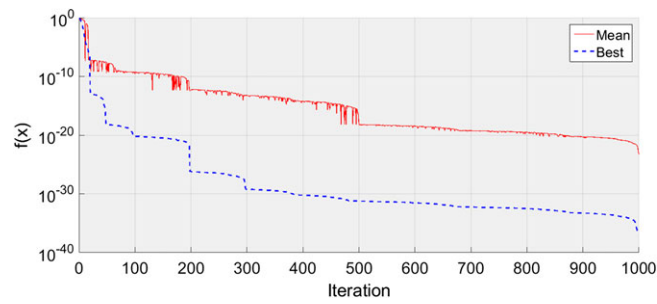


FIGURE 29 The convergence history of case 3 with PICA

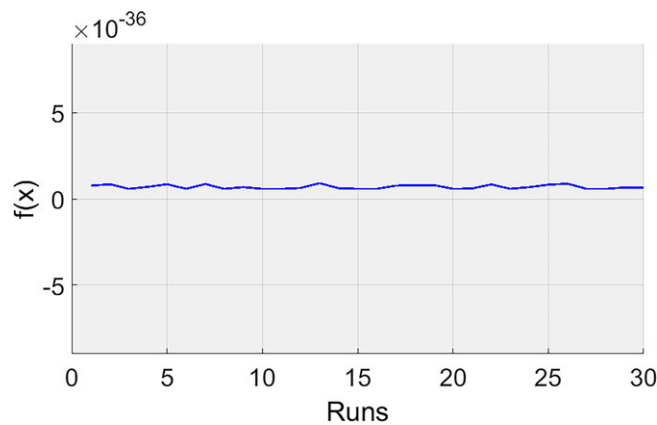


FIGURE 30 The stability chart of case 3 with PICA

TABLE 16 Statistical results of multi-population ICA

Problem	N	Mean	Std. deviation	Std. error mean	Worst	Best
Test 1	30	0.0	0.0	0.0	0.0	0.0
Test 2	30	-1.2159819999999999e+002	7.226896453227138e-014	1.319444736062194e-014	-1.2159820000000000e+002	-1.2159820000000000e+002
Case 1	30	1.9885860000000001e-031	1.739458967563944e-031	3.175803047964731e-032	3.9443000000000000e-031	4.9304000000000000e-032
Case 2	30	1.046312443884771e-026	1.511543708264576e-026	2.759688618905053e-027	4.4145000000000001e-026	0.0
Case 3	30	6.898049999999997e-037	1.150107106181705e-037	2.099798685342363e-038	9.039099999999999e-037	5.8000000000000000e-037

TABLE 17 Comparison results of Case 3

Methods	X	Variables values	f	Functions values
The best in the work of Wang et al ³¹	x_1	0.67154465	f_1	-0.00000375
	x_2	0.74097111	f_2	0.00001537
	x_3	0.95189459	f_3	0.00000899
	x_4	-0.30643725	f_4	0.00001084
	x_5	0.96381470	f_5	0.00001039
	x_6	-0.26657405	f_6	0.00000709
	x_7	0.40463693	f_7	0.00000049
	x_8	0.91447470	f_8	-0.00000498
The best in the work of Floudas et al ³²	x_1	0.1644	f_1	-8.8531e-005
	x_2	-0.9864	f_2	3.5894e-005
	x_3	-0.9471	f_3	6.6216e-006
	x_4	-0.3210	f_4	2.1560e-005
	x_5	-0.9982	f_5	1.2320e-005
	x_6	-0.0594	f_6	3.9410e-005
	x_7	0.4110	f_7	-6.8400e-005
	x_8	0.9116	f_8	-6.4440e-005
The best of ICA ²⁹	x_1	0.164431665854327	f_1	2.775557561562891e-016
	x_2	-0.986388476850967	f_2	-1.110223024625157e-016
	x_3	0.718452601027603	f_3	-1.110223024625157e-016
	x_4	0.718452601027603	f_4	1.734723475976807e-018
	x_5	0.997964383970433	f_5	0
	x_6	0.063773727557003	f_6	0
	x_7	-0.527809105283546	f_7	0
	x_8	-0.849363025083964	f_8	0
The best of PICA	x_1	0.164431665854327405	f_1	5.368529659036217811e-019
	x_2	-0.986388476850967110	f_2	2.548307417523678423e-019
	x_3	0.718452601027603350	f_3	-3.378192205891815512e-019
	x_4	-0.695575919707310931	f_4	3.389211820587187123e-019
	x_5	0.997964383970432520	f_5	0
	x_6	0.063773727557002571	f_6	0
	x_7	-0.527809105283546241	f_7	0
	x_8	-0.849363025083964123	f_8	0

TABLE 18 The comparison statistical results of serial ICA²⁵ and PICA

Problem	Speedup	Efficiency	Serial ICA time	PICA time	#processors	Super linear performance?
Case 1	2.82	1.41	0.0341	0.012	2	Yes
Case 2	5.1	2.55	2.1	0.411	2	Yes
Case 3	6.24	3.12	6.78	1.08	2	Yes

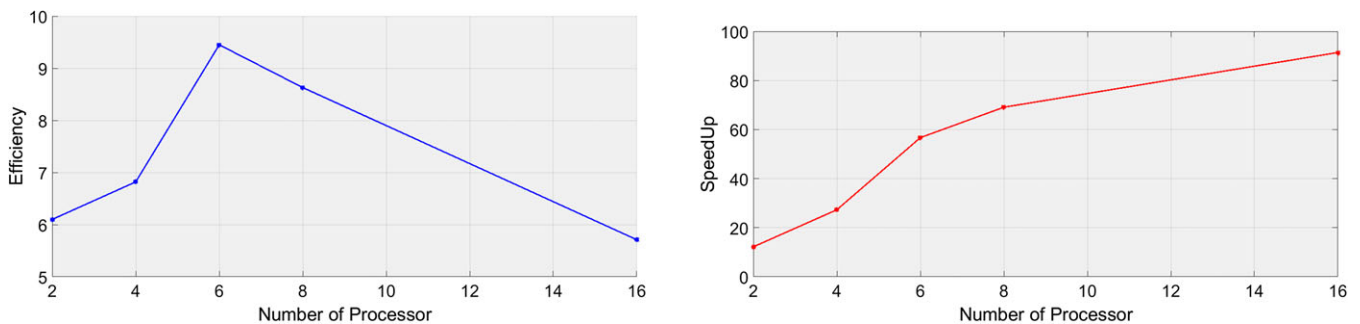


FIGURE 31 Speedup diagram and parallel efficiency diagram for f_1

4.10 | Case studies

In this section, three commonly explored systems of nonlinear equations have been used to demonstrate the performance of the proposed method, and the obtained results have been compared with the other known methods.

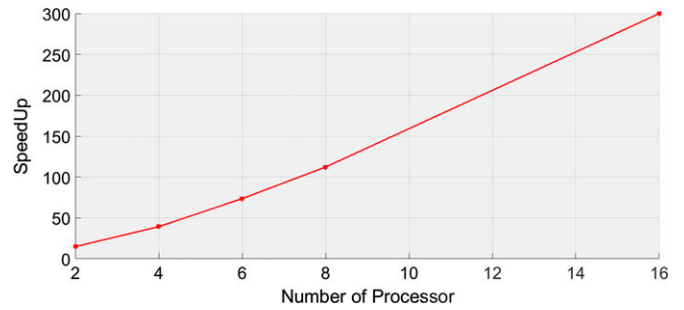
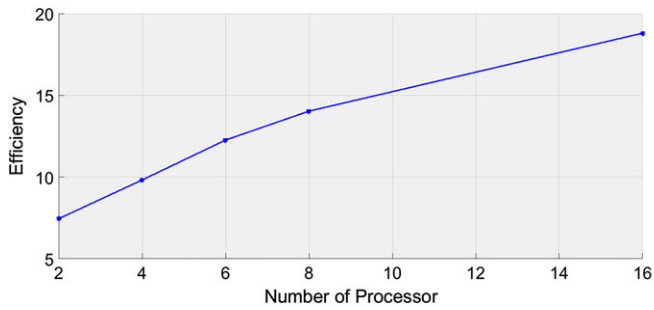


FIGURE 32 Speedup diagram and parallel efficiency diagram for f_2

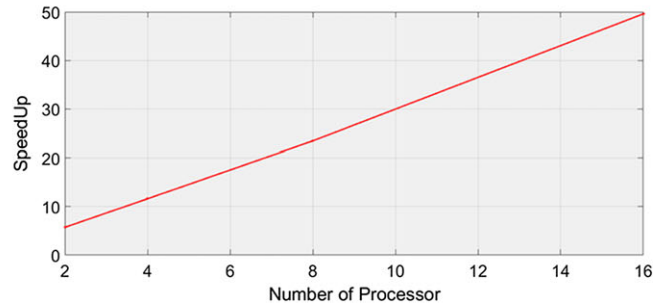
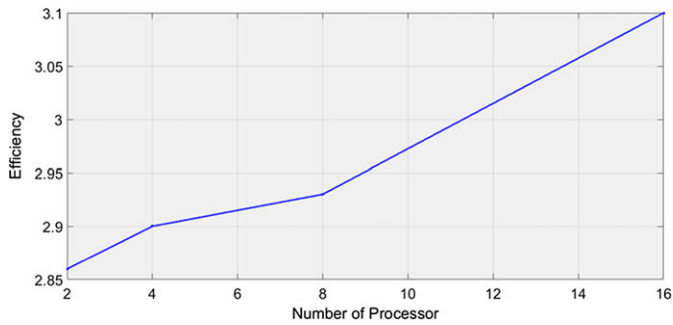


FIGURE 33 Speedup diagram and parallel efficiency diagram for f_3

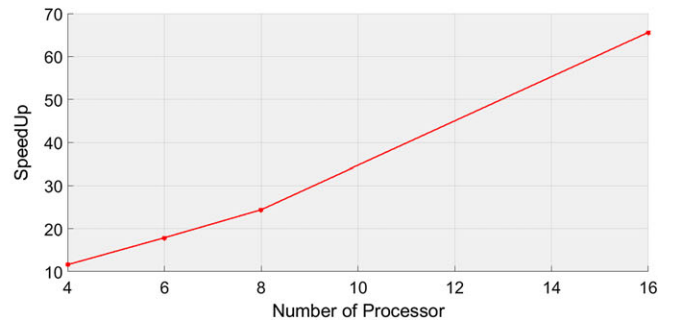
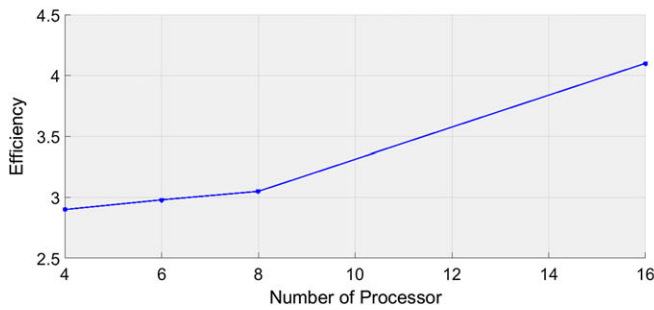


FIGURE 34 Speedup diagram and parallel efficiency diagram for f_4

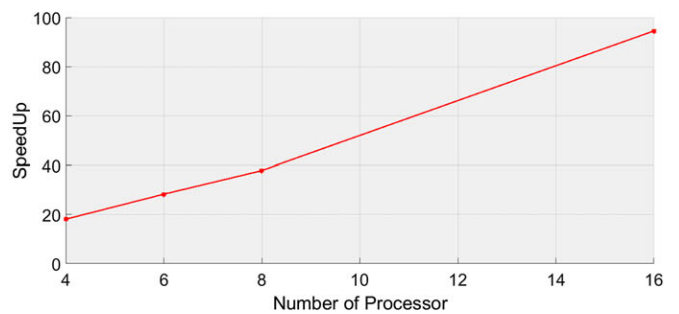
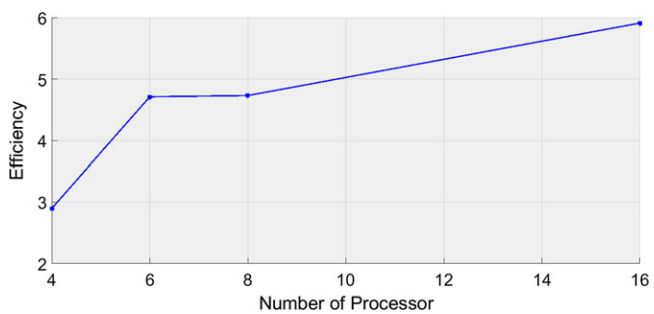


FIGURE 35 Speedup diagram and parallel efficiency diagram for f_5

Case 1. This example has been given in other works²⁷⁻³⁰

$$\begin{cases} x_1 - 3x_1x_2^2 - 1 = 0 \\ 3x_1^2x_2 - x_2^3 + 1 = 0. \end{cases} \quad (1)$$

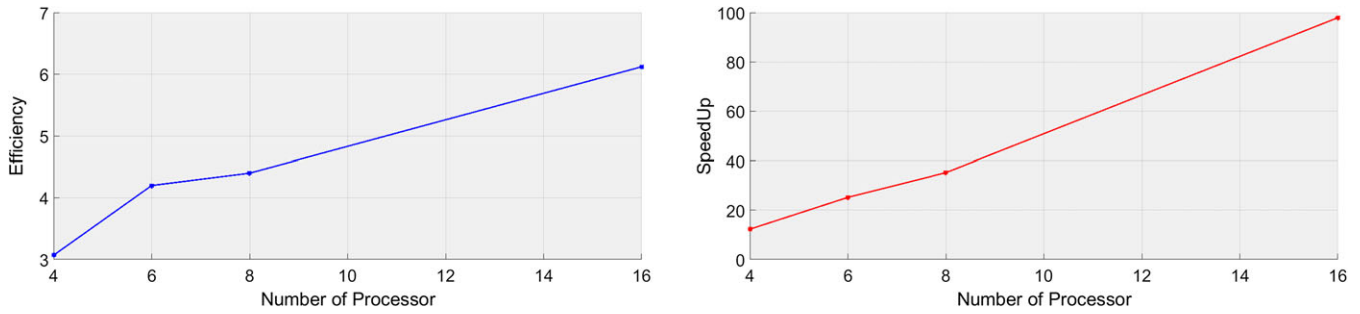


FIGURE 36 Speedup diagram and parallel efficiency diagram for f_6

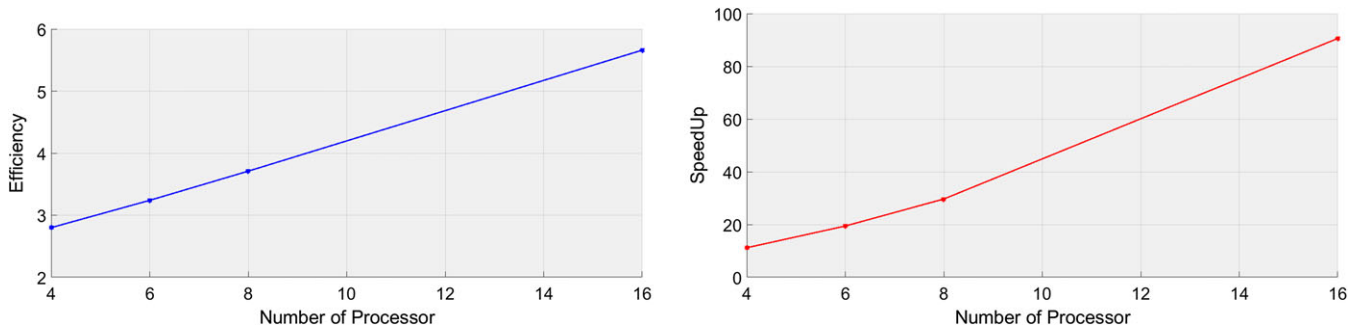


FIGURE 37 Speedup diagram and parallel efficiency diagram for f_7

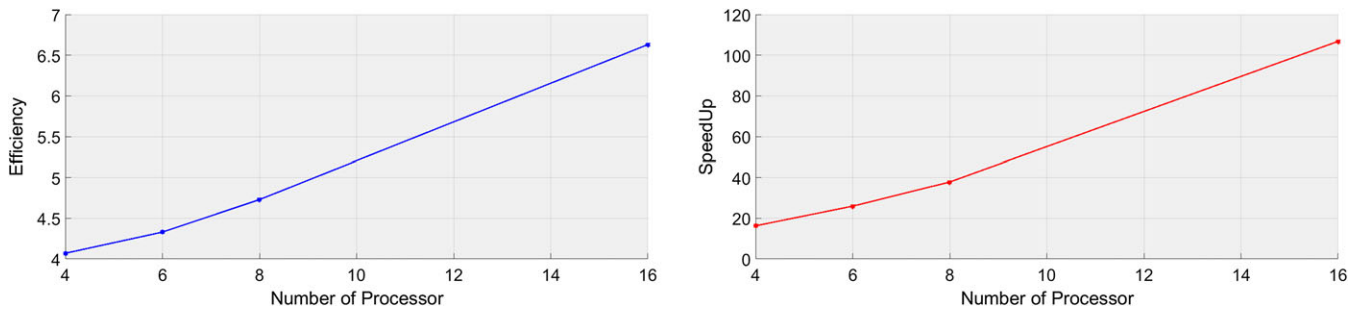


FIGURE 38 Speedup diagram and parallel efficiency diagram for f_8

The solutions in the works of Jaberipour et al²⁷ and Gyurhan and Nedzhibov²⁸ have been obtained with 120 iterations with unknown population sizes. The parameters of the sequential ICA method²⁹ have been set to 50 iterations with 250 countries. The results of case 1 are compared with the PICA in Table 14. The obtained solutions by the PICA are better and more accurate than those of the previous works. Figures 21 to 23 indicate the convergence history of Case 1. Figure 24 shows the stability diagram of this case.

Case 2. (Problem 2 in the work of Wang et al,³¹ Test Problem 14.1.4 in the work of Floudas et al,³² and Case Study in the works of Abdollahi et al^{29,30})

$$\begin{aligned}
 f_1(x_1, x_2) &= 0.5 \sin(x_1 x_1) - 0.25 x_2 / \pi - 0.5 x_1 = 0 \\
 f_2(x_1, x_2) &= (1 - 0.25 / \pi) ((\exp(2x_1) - e) + e x_2 / \pi - 2e x_1) = 0.
 \end{aligned}
 \tag{2}$$

The results of case 2 in other works²⁹⁻³² with 50 iterations and the population size of 250 are compared with the PICA in Table 15. The obtained solutions show that PICA outperforms the mentioned methods with 250 countries and 35 iterations. The results are illustrated in Figures 25 to 27.

Case 3. (Problem 6 in the work of Wang et al³¹ and Test Problem 14.1.6 in the work of Floudas et al³²)

Case 3 has been solved by the filled function method in the work of Wang et al³¹ and has been proposed as a problem in the work of Abdollahi et al.^{29,32}

$$\begin{aligned}
 4.731 \times 10^{-3}x_1x_3 - 0.357x_2x_3 - 0.1238x_1 + x_7 - 1.637 \times 10^{-3}x_2 - 0.9338x_4 - 0.3 &= 0 \\
 0.2338x_1x_3 + 0.7623x_2x_3 + 0.2638x_1 - x_7 - 0.07745x_2 - 0.6734x_4 - 0.6022 &= 0 \\
 x_6x_8 + 0.3578x_1 + 4.731 \times 10^{-3}x_2 &= 0 \\
 -0.7623x_1 + 0.2238x_{20,3461} &= 0 \\
 x_1^2 + x_2^2 - 1 &= 0 \\
 x_3^2 + x_4^2 - 1 &= 0 \\
 x_5^2 + x_6^2 - 1 &= 0 \\
 x_7^2 + x_8^2 - 1 &= 0 \\
 -1 \leq x_i \leq 1, \quad i = 1, 2, \dots, 8.
 \end{aligned} \tag{3}$$

The number of iterations for this problem in other works^{29,31,32} is 1000 and the population size is 300. Our results with the same numbers of iterations and countries have been compared in Table 17. The convergence history of ICA²⁹ and PICA is shown in Figures 28 and 29, respectively. Figure 30 shows the stability diagram of PICA for Case 3. The statistical results of the tests and cases are illustrated in Table 16. The comparison between the statistical results of the serial ICA and the parallel ICA is given in Table 18.

4.11 | Speedup and parallel efficiency

As already discussed above, speedup and parallel efficiency, defined in Equations 4 and 5, are important parameters, indicating the quality of parallelization. They reveal whether parallel algorithms are suitable or not for given problems. As already demonstrated by the above analyses, we can also use these parameters to compare the speed of our algorithms with that of another parallel EC method. For example, we know based on other works^{15,21,33} that the serial ICA is faster than the GA, ES, ABC, and PSO. Therefore, if our speedup and parallel efficiency values for the PICA are higher than those of the parallel implementations of the other mentioned EC algorithms, then our parallel algorithm is provably faster than the others.

$$\text{Speedup} = \frac{\text{Execution time on one processor core}}{\text{Execution time on } m \text{ processor cores}} = \frac{T(1)}{T(m)} \tag{4}$$

$$\text{Parallel Efficiency} = \frac{\text{Speedup}}{m} \tag{5}$$

For the benchmark f_1 (Table 1), the sequential ICA obtains the best results after 10 iterations, while the multi-population PICA obtains them after three iterations. So, this benchmark is very simple, and when we find the results with six processors after three iterations, there is clearly no need for a larger number of processors. In Figures 31 to 38, we show the parallel efficiency and speedup as a function of the number of processors on all the benchmarks; they convincingly demonstrate efficiency and usefulness of the parallel implementations. Super-linear speedup means that using M processors leads to an algorithm that runs more than M times faster than the sequential version. However, reporting super-linear speedup is somewhat controversial, especially for the “traditional” research community, since some non-orthodox practices could be thought of being the cause for this result.⁸ Super-linear performance is nevertheless the main achievement of our algorithms. Indeed, in all benchmark diagrams, we can see that we obtain super-linear performance. Based on our extensive experiments, the PICA is the best choice for solving the considered problems.

5 | CONCLUSION

We have introduced two parallel versions of the ICA, ie, the multi-population PICA and the master-slave PICA, utilizing MPI instructions in their implementations. We tested them on two different computer platforms, on eight mathematical benchmarks and three nonlinear case studies, and compared them with the sequential ICA and a large set of other parallel evolutionary computing methods. Our extensive experiments show that both PICA implementations are very efficient in solving different kinds of complex problems and they are in general faster and more efficient and stable than the other parallel methods considered. The multi-population PICA has been proven to be especially prominent approach. In fact, both approaches have been shown to achieve a super-linear performance. Figures 13 to 20, Figure 23, Figure 27, and Figure 30 highlight the stability of the proposed multi-population PICA, and Figures 21 to 23, Figure 25, Figure 26, Figure 28, Figure 29, and Figures 31 to 38 demonstrate that our multi-population algorithm converges fast to the best results and has very high speedup and parallel efficiency values. Overall, compared with the

competing parallel methods, its run time is significantly lower (there is a smaller number of fitness function evaluations) and its accuracy is better. Indeed, the proposed multi-population PICA is the champion among the analyzed evolutionary methods by a clear margin.

ORCID

Amin Majd  <http://orcid.org/0000-0001-7256-6618>

REFERENCES

1. Cantú-Paz E. *A Survey of Parallel Genetic Algorithms*. Urbana, IL: Department of Computer Science and Illinois Genetic Algorithms Laboratory University of Illinois; 1997.
2. Zhou Y, Tan Y. Particle swarm optimization with triggered mutation and its implementation based on GPU. Paper presented at: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation; 2011; Portland, OR.
3. Liu H, Li P, Wen Y. Parallel ant colony optimization algorithm. Paper presented at: The 6th World Congress on Intelligent Control and Automation; 2006; Dalian, China.
4. Parpinelli R, Benitez C, Lopes S. Parallel approaches for the artificial bee colony algorithm. *Handbook of Swarm Intelligence*. Vol 8. Berlin, Germany: Springer; 2010: 329-345.
5. Digalakis J, Margaritis K. A parallel memetic algorithm for solving optimization problems. Paper presented at: 4th Metaheuristics International Conference. Parallel Distributed Processing Laboratory; 2001; Athens, Greece.
6. Vanneschi L, Codecasa D, Mauri G. Comparative study of four parallel and distributed PSO methods. *N Gener Comput*. 2011;29:129-161.
7. Digalakis J, Margaritis K. A parallel memetic algorithm for solving optimization problems. Paper presented at: 4th Metaheuristics International Conference, MIC; 2001; Thessaloniki, Greece.
8. Alba E. Parallel evolutionary algorithms can achieve super-linear performance. *Inf Process Lett*. 2002;82:7-13.
9. Majd A, Lotfi Sh, Sahebi G, Daneshtalab M, Plosila J. PICA: Multi-population implementation of parallel imperialist competitive algorithms. Paper presented at: 24th Euromicro International Conferences on Parallel, Distributed and Network-Based Processing PDP; 2016; Heraklion, Greece.
10. Majd A, Abdollahi M, Sahebi G, et al. Parallel imperialist competitive algorithm based on multi-population technique for solving systems of nonlinear equation. Paper presented at: The 2016 International Conference on High Performance Computing & Simulation (HPCS); 2016; Innsbruck, Austria.
11. Lotfi H, Boroumandnia A, Lotfi Sh. Task graph scheduling in multiprocessor systems using a coarse grained genetic algorithm. Paper presented at: 2nd International Conference on Computer Technology and Development; 2010; Cairo, Egypt.
12. Lotfi H, Lotfi Sh, Boroumandnia A. Task graph scheduling in multiprocessor systems using a two population genetic algorithm. Paper presented at: International Conference on Software and Computing Technology; 2010; Kunming, China.
13. Majd A, Lotfi Sh, Sahebi G. Review on parallel evolutionary computing and introduce three general framework to parallelize all EC algorithms. Paper presented at: 5th Conference on Information and Knowledge Technology; IEEE; 2013; Shiraz, Iran.
14. Majd A, Sahebi G. A survey on parallel evolutionary computing and introduce four general frameworks to parallelize all EC algorithms and create new operation for migration. *J Inf Comput Sci*. 2014;9:97-105.
15. Mousa A, Wahed W, Allah R. A hybrid ant colony optimization approach based local search scheme for multi objective design optimizations. *Electr Power Syst Res*. 2011;81(4):1014-1023.
16. Narasimhan H. Parallel artificial bee colony (PABC) algorithm. Paper presented at: World Congress on Nature & Biologically Inspired Computing; IEEE; 2009; Coimbatore, India.
17. Willie ECG, Lopes EYHS. Discrete capacity assignment in IP networks using particle swarm optimization. *Appl Math Comput*. 2011;217(12):5338-5346.
18. Vanneschi L, Codecasa D, Mauri G. A comparative study of four parallel and distributed PSO methods. *New Generation Computing*. Vol 29. Tokyo, Japan: Ohmsha Ltd. and Springer; 2011:129-161.
19. Gargari EA, Lucas C. Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. Paper presented at: IEEE Congress on Evolutionary Computation; 2007; Singapore, Singapore.
20. <https://www.mpich.org/>
21. Bahrami H, Faez K, Abdechiri M. Imperialist competitive algorithm using chaos theory for optimization (CICA). Paper presented at: 12th International Conference on Computer Modeling and Simulation; 2012; Cambridge, UK.
22. Abdechiri M, Faez K, Bahrami H. Adaptive imperialist competitive algorithm (AICA). Paper presented at: 9th IEEE International Conference on Cognitive Informatics (ICCI); 2010; Beijing, China.
23. Bahrami H, Abdechiri M, Meybodi M. Imperialist competitive algorithm with adaptive colonies movement. *I J Intell Syst Appl*. 2012;2:49-57.
24. Alba E, Luna F, Nebro AJ, Troya JM. Parallel heterogeneous genetic algorithms for continuous optimization. *Parallel Comput*. 2004;30:699-719.
25. Weinert K, Mehnen J, Rudolph G. Dynamic neighborhood structures in parallel evolution strategies. *Complex Syst Publ*. 2002;13:227-243.
26. Basturk A, Akay R, Kalinli A. Comparison of fine-grained and coarse-grained parallel models in particle swarm optimization algorithm. Paper presented at: 2nd World Conference on Information Technology (WCIT); 2011; Antalya, Turkey.
27. Jaberipour M, Khorram E, Karimi B. Particle swarm algorithm for solving systems of nonlinear equations. *Comput Math Appl*. 2011;62:566-576.
28. Gyurhan H, Nedzhibov A. A family of multi-point iterative methods for solving systems of nonlinear equations. *J Comput Appl Math*. 2008;222(2):244-250.
29. Abdollahi M, Isazadeh A, Abdollahi D. Imperialist competitive algorithm for solving systems of nonlinear equations. *Comput Math Appl*. 2013;65:1894-1908.

30. Abdollahi M, Lotfi SH, Abdollahi D. Solving systems of nonlinear equations using cuckoo optimization algorithm. Paper presented at: Proceedings of the 3rd International Conference on Contemporary Issues in Computer and Information Sciences (CICIS), Vol 3; 2012; Zanjan, Iran.
31. Wang C, Luo R, Wu K, Han B. A new filled function method for an unconstrained nonlinear equation. *Comput Appl Math*. 2011;235:1689-1699.
32. Floudas CA, Pardalos PM, Adjiman CS, et al. *Handbook of Test Problems in Local and Global Optimization*. Dordrecht, The Netherlands: Kluwer Academic Publishers; 1999.
33. Yin PY, Yu SS, Wang PP, Wang YT. A hybrid particle swarm optimization algorithm for optimal task assignment in distributed systems. *Comput Stand Interface*. 2006;28:441-450.

How to cite this article: Majd A, Sahebi G, Daneshtalab M, Plosila J, Lotfi S, Tenhunen H. Parallel imperialist competitive algorithms. *Concurrency Computat Pract Exper*. 2018;e4393. <https://doi.org/10.1002/cpe.4393>

Paper III

Parallel Imperialist Competitive Algorithm Based on Multi-Population Technique for Solving Systems of Nonlinear Equation

A. Majd, M. Abdollahi, G. Sahebi, D. Abdollahi, M. Daneshtalab, J. Plosila and H.Tenhunen

Multi-Population Parallel Imperialist Competitive Algorithm for Solving Systems of Nonlinear Equations

Amin Majd
Department of Information
Technology
University of Turku
Turku, Finland
Amin.Majd@utu.fi

Mahdi Abdollahi
Department of Computer
Sciences
University of Tabriz
Tabriz, Iran
m.abdollahi89@ms.tabrizu.ac.ir

Golnaz Sahebi
Department of Information
Technology
University of Turku
Turku, Finland
Golnaz.Sahebi@utu.fi

Davoud Abdollahi
Department of Mathematic
Sciences
University College of
Daneshvaran
Tabriz, Iran
abdollahi_d@daneshvaran.ac.ir

Masoud Daneshtalab
Royal Institute of Technology (KTH)
Stockholm, Sweden,
masdan@kth.se

Juha Plosila
Department of Information Technology
University of Turku
Turku, Finland
juplos@utu.fi

Hannu Tenhunen
Royal Institute of Technology
Stockholm, Sweden
University of Turku, Finland
Hannu@kth.se

Abstract— the widespread importance of optimization and solving NP-hard problems, like solving systems of nonlinear equations, is indisputable in a diverse range of sciences. Vast uses of non-linear equations are undeniable. Some of their applications are in economics, engineering, chemistry, mechanics, medicine, and robotics. There are different types of methods of solving the systems of nonlinear equations. One of the most popular of them is Evolutionary Computing (EC). This paper presents an evolutionary algorithm that is called Parallel Imperialist Competitive Algorithm (PICA) which is based on a multi-population technique for solving systems of nonlinear equations. In order to demonstrate the efficiency of the proposed approach, some well-known problems are utilized. The results indicate that the PICA has a high success and a quick convergence rate.

Keywords— parallel imperialist competitive algorithm (PICA); multi-population technique; evolutionary computing (EC); super linear performance; nonlinear equations; multi-objective optimization;

I. INTRODUCTION

Systems of nonlinear equations are one of the NP-Hard problems, which resemble the multi-objective optimization problems. Systems of nonlinear equations are utilized in a range of engineering applications, such as weather forecast, petroleum geological prospecting, computational mechanics, and control fields. The quality of answers of the classical methods, like the Newton-type methods, depends on the initial guess of the solution. However, selecting suitable initial solutions for the most systems of nonlinear equations is extremely difficult.

So far, several methods have been proposed for optimization problems. They can be classified into two major classes: mathematical methods and evolutionary computing (EC) methods. There are different types of EC methods, most of them are implemented in a sequential mode and some in a parallel mode.

Sequential EC methods are more popular than other mathematical methods for solving nonlinear equations, but they do not always provide sufficient accuracy. There are different kinds of parallel EC methods that are capable of improving the accuracy of results. In this work, we utilize a multi-population EC method that improves the results of the used benchmarks.

The rest of the paper is organized as follows: In Section II, the imperialist competitive algorithm is reviewed. Section III introduces the parallel implementation of the ICA based on the

multi-population technique. In Section IV, the proposed algorithm is compared with the related previous works. Finally, Section V concludes the paper and indicates the future works.

II. RELATED WORK

Let us first look into the sequential algorithms that have been proposed for solving systems of nonlinear equations. El-Emary and El-Kareem employed Gauss-Legendre integration as a technique to solve the system of nonlinear equations and used genetic algorithm (GA) to discover the results without converting the nonlinear equations to linear equations [15]. Mastorakis employed genetic algorithm (GA) to solve a non-linear equation as well as systems of non-linear equations [17]. Li and Zeng [18], used a neural-network algorithm for solving a set of nonlinear equations. The computation is carried out by a simple gradient descent rule with variable step-size levels [18]. Huan-Tong et al. proposed a modified evolution strategy (ES) based on a probability ranking method to solve complicated nonlinear systems of equations (NSE) problems [19]. M. Abdollahi et al. applied the imperialist competitive algorithm for solving nonlinear systems of equations [16] Ouyang et al. employed a hybrid particle swarm optimization (HPSO) algorithm. The particle swarm optimization (PSO) method focuses on "exploration", and the Nelder-Mead simplex method (SM) focuses on "exploitation" [20], while Wu et al. used a new variation of the social emotional optimization algorithm called MSEO, mainly inspired by the Metropolis Rule [21]. M. Abdollahi et al. proposed a cuckoo optimization algorithm for solving nonlinear systems equations [25], [29]. Luo et al. applied a combination of the chaos search and Newton type methods [1]. Grosan and Abraham employed a new perspective of the evolutionary algorithms (EA) [7], Mo et al. proposed a combination of the conjugate direction method (CD) [2], and M. Jaberipour used the particle swarm algorithm [3]. Henderson et al. [22], and Pourjafari et al. [23] introduced a methodology based on a polarization technique and a novel optimization method based on Invasive Weed Optimization (IWO), respectively, for finding all roots of a system of nonlinear equations.

In past years, researchers have utilized some parallel EC methods for optimization problems such as Parallel Genetic Algorithms [4], and Parallel PSO [9], Parallel ABC (PABC) [6], Parallel Ant Colony Optimization (PACO) [5], and Parallel Memetic [10] algorithms. Wu and Kang used a parallel elite-subspace evolutionary algorithm for solving systems of

nonlinear equations [14]. Some parallel EC methods can achieve super-linear performance [12], where each one is implemented with different techniques and hardware platforms. For example, Parallel Genetic Algorithms are implemented in the following four categories [4]: Master-Slave Genetic Algorithms, Coarse Grain Genetic Algorithms (Multi-Populations Genetic Algorithms), Fine Grain Genetic Algorithms, and Hybrid Genetic Algorithms [27].

The obtained results of mathematical methods are sensitive to the initial guess of the solution. The population size of the evolutionary algorithms is large and the convergence of the evolutionary methods to the global minimum is slow. The EC methods are impractical for large-scale problems, like systems of nonlinear equations because of their high linear algebra costs and large memory requirements. For this reason, it is necessary to find an efficient algorithm for solving systems of nonlinear equations. Let systems of nonlinear equations be of the form:

$$\begin{cases} f_1(x_1, x_2, \dots, x_n) = 0 \\ f_2(x_1, x_2, \dots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) = 0 \end{cases} \quad (1)$$

In order to transform (1) to an optimization problem, we will use the auxiliary function:

$$\min f(x) = \sum_{i=1}^n f_i^2(x), \quad x = (x_1, x_2, \dots, x_n) \quad (2)$$

Where $f(x)$ is the global minimum that will be minimized.

In this paper, a parallel imperialist competitive algorithm (PICA) based on the multi-population technique for solving systems of nonlinear equations problems is presented. The proposed method overcomes the mentioned weaknesses of evolutionary methods for solving the systems of nonlinear equations problems. We have also selected some well-known problems for the evaluation.

III. BACKGROUND

The Imperialist Competitive Algorithm (ICA) was introduced by E. Atashpaz and C. Lucas [26], and is inspired by imperialistic competition. ICA is an evolutionary algorithm and optimizes results of problems. In this algorithm, all countries are divided into two types: imperialist states and colonies. Imperialistic competition is the main part of this algorithm, and the expectation is that the colonies converge to the global minimum of the cost function. In the first step, the algorithm creates some countries and after sorting them the best countries are selected to be imperialists and the rest of the countries form the colonies of these imperialists (Fig 1, step 1). After dividing all colonies among imperialists, these colonies start moving toward their relevant imperialist countries (Fig 1, step 2). In the next step, the ICA computes the power of each imperialist and the imperialistic competition begins. The weakest imperialist loses its weakest colony and the selected imperialist captures this colony (Fig 1, step 5). These steps are then repeated until the termination condition is satisfied. The termination conditions can be different. For example, the ICA algorithm could stop after certain number of iterations or when all colonies have become members of one imperialist (see Fig. 1).

ICA is a suitable method for optimization problems, but there exist some challenges concerning the evolutionary algorithms.

For example, when we are considering a far-reaching search area, we need a large initial population to obtain an appropriate result, but with a resource constrained processor we may not be able to satisfy this requirement. Also, when we face a complex problem that needs complex computations, the run time will increase, and therefore we need to utilize an efficient method to improve the speed, stability, and accuracy.

The sequential ICA algorithm inherently has a parallel structure, and therefore, a parallel ICA implementation is a viable solution to improve the ICA. In the ICA, each imperialist and colonies work independently, and after a decade a colony moves to another imperialist. Hence, this algorithm works similarly as a multi-population method that works on a processor. In the next section, we utilize a multi-population ICA to solve some complex problems.

IV. THE PROPOSED METHOD

In this work, we utilize a multi-population model to implement the Parallel Imperialist Competitive Algorithm (PICA), by applying a selective local search strategy, in order to solve systems of nonlinear equations. We intend to utilize the full capacity of evolutionary algorithms (e.g. faster convergence, run time speed, and accuracy) for solving such problems. There are different approaches for parallelizing evolutionary algorithms, such as the master slave, multi-population, fine-grain, and hybrid methods; but multi-population method has better convergences and has more accurate results than other parallel methods. Of these, we use the multi-population method. In the multi-population method, there are some independent populations in different processors, each covering its own independent area of the search space. Each processor runs the ICA on its population with independent parameter values. Due to this independency, different levels of exploration and exploitation can be utilized, and therefore the application can get out from local optimums. The other advantage of the multi-population method is its ability for migration, which can significantly improve performance of solving nonlinear equations. It is the most important parallel operation in the multi-population implementation, as letting processors to share the best results and investigate areas with other processors help discover better results in a smaller number of iterations. There are different kinds of migration techniques, for example, each processor can select the best or worst countries to be sent to the other processors or can randomly select some countries. In our work, we select the best countries to be migrated between processors to share the best results together, and the processors replace their worst countries with the received best countries. Receiver processors are selected based on the connection topology used. Figure 1 illustrates the migration behaviour. For example, for a network with the fully connected topology, each processor can receive countries from any other processor. It can be very useful if all processors receive all migrated countries, but in practice we have to find balance between data communication cost (communication time) and achieved improvement in results.

In this work, each country is a possible solution for the selected nonlinear equation. Hence, the algorithm randomly creates the initial populations, which are the possible solutions. Some of them are then selected, at each iteration step, to be processed by the ICA in order to converge them towards better results.

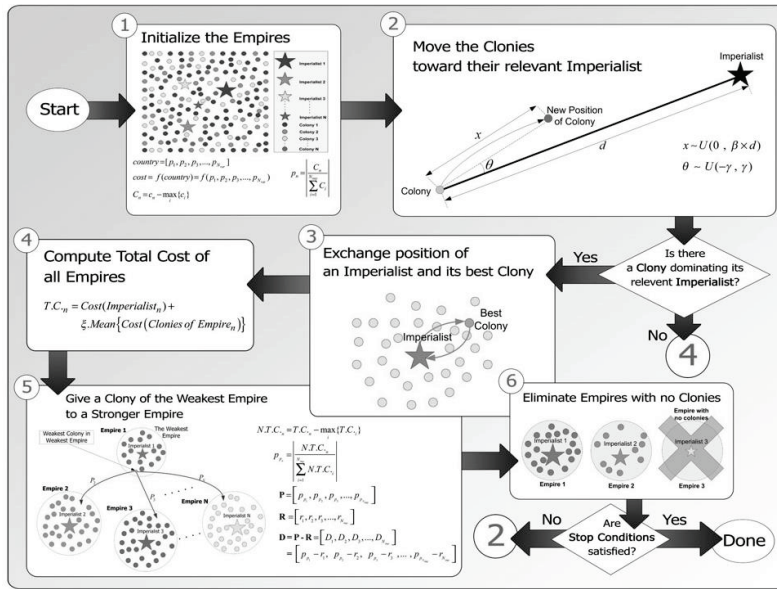


Fig. 1. Imperialist Competitive Algorithm [28]

In our implementation, several processors are connected together using a ring topology and message passing based communication. The ring topology has been selected because of its low communication cost and simplicity. Each processor is first initialized with a set of independent countries (the number of countries in each processor is the same) and the ICA to be independently run on it. After some decades (the period varies from an execution to another), the best country migrates from each processor P_i to the next processor P_{i+1} in the ring and replaces the worst country in P_{i+1} . Since we utilize the ring topology to connect processors together, and because the migration takes place in all processors synchronously, the numbers of countries in any two processors are equal at any given time. The migration strategy can affect the result as well. Generally, it is better to establish a balance between the migration rate and data communication. The chosen ring topology is utilized to reduce the migration rate and to decrease the distance of the migrations.

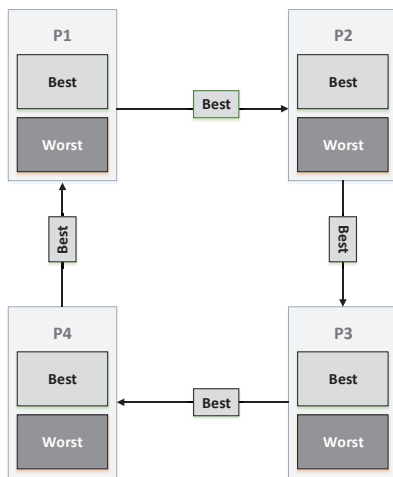


Fig. 2. Multi-population migration operation

Figure 2 shows the architecture of the multi-population structure with a ring topology. Figure 3 presents the Multi-

Population ICA pseudo code. In our implementation, all parameters in different processors are equal, and all ICA computations run independently in different processors. On the other hand, the migration operations run synchronously as explained above.

In the multi-population ICA, the pressure of selection increases by growing the number of countries, which helps to obtain more accurate results in a shortest time and converge to the results faster than the sequential ICA. Therefore it is beneficial to increase the number of countries.

```

Processor  $P_i$ :
1-Create independent initial countries.
2-Run ICA algorithm independently.
3-If now is time of migration do
  a) Wait until all processors arrive to this point.
  b) Send the best country to processor  $(P_{i+1}) \bmod (\#processors)$ .
  c) Receive a country from  $(P_{i-1}) \bmod (\#processors)$  and replace the worse country with the received one.
4-If termination condition is reached, then terminate algorithm.
5-Show the best country.
6-End.
  
```

Fig. 3. Multi-population ICA pseudo code

V. EXPERIMENT AND RESULTS

In this section, five commonly explored problems have been utilized to demonstrate the performance of the PICA. The obtained results have been compared with the other well-known methods that have used the same problems. The parallel ICA has been implemented on both share memory and message passing models. The message passing interface (MPI) has been utilized to parallelize our algorithm and MPICH2 to run the algorithm.

In the multi-population ICA, processors have been connected in a ring topology with different processors on the different tests. The proposed algorithm has been tested on an Intel core i3-330M, processors 2.13 GHz (64-bit) and memory 4 GB. The best results of benchmarks have been obtained by 30 independent runs. The used parameters for solving the problems have been illustrated in Table 1.

A. Benchmarks

Test 1: 10-dimension Rastrigin Function

$$f(x) = \sum_{i=1}^{10} [x_i^2 - 10 \cos(2\pi x_i) + 10n] \quad |x_i| \leq 5.2 \quad (3)$$

The answer of this test with $f(0, 0, 0, \dots, 0)$ is 0. This test has been solved by Mo et al. [2] and ICA [24] with 1000 iterations and 300 population sizes. PICA has been applied to optimize it with the same parameters.

TABLE I. USED PARAMETERS IN PICA FOR TESTS AND CASES

Parameters	Test 1	Test 2	Case 1	Case 2	Case 3
Total population size	300	300	250	250	300
Number of empires	10	10	10	10	10
Number of iteration	1000	1000	45	50	1000
Revolution rate	0.04	0.04	0.04	0.04	0.04
Ξ	0.02	0.02	0.02	0.02	0.02
θ	0.5	0.5	0.5	0.5	0.5
β	2	2	2	2	2

The results of Mo et al. [2], ICA [24] and our proposed algorithm have been presented in Tables 2-4. Figures 4 and 5 indicate the convergence history of ICA and PICA, respectively. The stability chart of PICA has been indicated in Figure 6. PICA has reached to the optimized answer before 50 iterations.

Test 2: This example has been used as a benchmark in [3] and [24].

$$\min f(x) = \sum_{i=1}^D \left[\sin(x_i) + \sin\left(\frac{2x_i}{3}\right) \right] \quad (4)$$

The answer of the test 2 is 1.21598D and the variables of the function are in (3, 13). The comparison results of ICA [24] and PICA with 1000 iterations and 300 population sizes have been given in Table 5. PICA has solved Test 2 quicker than prior methods before 200 iterations (see Figures 7-9).

B. Case study

In this section, three commonly explored systems of nonlinear equations have been used to demonstrate the performance of the proposed method, and the obtained results have been compared with the other known methods.

Case 1: This example has been given in [3], [11], [24], and [25]:

$$\begin{cases} x_1 - 3x_1x_2^2 - 1 = 0 \\ 3x_1^2x_2 - x_2^3 + 1 = 0 \end{cases} \quad (5)$$

The solutions in [3] and [11] have been obtained with 120 iterations with an unknown number of population sizes. The parameters of the ICA method [24] have been set to 50 iterations with 250 countries. The obtained solutions by PICA are better and more accurate than the previous works (see Table 6). Figures 10-12 indicate the convergence history of the Case 1. Figure 13 shows the stability chart of this case.

Case 2: (Problem 2 in [8], Test Problem 14.1.4 in [13], and Case study in [24] and [25])

$$\begin{aligned} f_1(x_1, x_2) &= 0.5 \sin(x_1x_1) - 0.25x_2/\pi - 0.5x_1 = 0 \\ f_2(x_1, x_2) &= (1 - 0.25/\pi)((\exp(2x_1) - e) + ex_2/\pi - 2ex_1 = 0 \end{aligned} \quad (6)$$

The results of Case 2 in [8], [13], [24], and [25] with the 50 iterations and the 250 population sizes were compared with

PICA in Table 8. The obtained solutions of PICA have outperformed the mentioned methods with 250 countries and 35 iterations. The speed up of the proposed algorithm gets better than the other literatures (see figures 14-16).

Case 3: (Problem 6 in [8] and Test Problem 14.1.6 in [13])

Case 3 has been solved by the filled function method in [8] and has been proposed as a problem in [13] and [24].

$$4.731 \times 10^{-3}x_1x_3 - 0.357x_2x_3 - 0.1238x_1 + x_7 - 1.637 \times 10^{-3}x_2 - 0.9338x_4 - 0.3 = 0 \quad (7)$$

$$0.2338x_1x_3 + 0.7623x_2x_3 + 0.2638x_1 - x_7 - 0.07745x_2 - 0.6734x_4 - 0.6022 = 0$$

$$x_6x_8 + 0.3578x_1 + 4.731 \times 10^{-3}x_2 = 0$$

$$-0.7623x_1 + 0.2238x_2^{0.3461} = 0$$

$$x_1^2 + x_2^2 - 1 = 0$$

$$x_3^2 + x_4^2 - 1 = 0$$

$$x_5^2 + x_6^2 - 1 = 0$$

$$x_7^2 + x_8^2 - 1 = 0$$

$$-1 \leq x_i \leq 1, \quad i = 1, 2, \dots, 8.$$

The number of iterations for this problem in [8], [13] and [24] is 1000 and the population size is 300. Our results with the same iterations and countries have been compared in Table 9. The convergence history of ICA [24] and PICA have been shown in Figures 17 and 18, respectively. Figure 19 shows the stability chart of PICA for the Case 3. The statistical results of tests and cases have been illustrated in Table 7. The comparison statistical results of the serial ICA and the parallel ICA have been given in Table 10.

VI. DISCUSSION

In this paper a parallel implementation of ICA based on the multi-population method has been utilized to solve the systems of nonlinear equations. There are different kinds of the PICA implementation such as the master-slave, the multi-population, and the hybrid methods that each one has different advantages. For example, the Master-Slave method can be utilized when we simply intend to increase the speed of our algorithm, but Multi-Population method must be used when we intend to increase both speed and accuracy. Multi-Population method increases the number of the initial population and therefore, the pressure of selection grows that it causes to find more accurate results.

In our implementation, the ring connection topology has been considered to connect the processors. In our algorithm, the migration operation causes that each processor has the ability to send its countries to next processors and receives some countries from previous processors. With the aforementioned ability each processor shares the best results with other processors which reduces the number of iterations considerably.

In this paper, PICA has been compared with other methods through some well-known benchmarks and case studies. PICA has obtained more accurate results with the lower number of iterations.

The most important result of PICA is about super linear performance (where the efficiency value of the algorithm is more than one). Our implementation achieved the super linear performance that means it is an outstanding method and is the best way to solve non-linear problems.

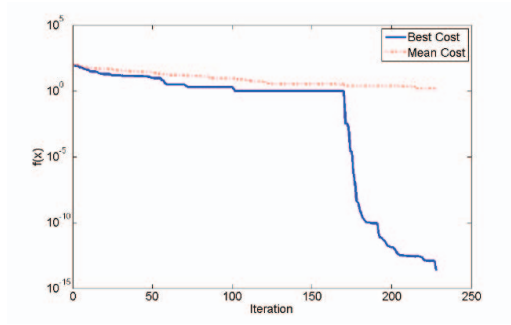


Fig. 4. The convergence history of Rastrigin Function (from [24])

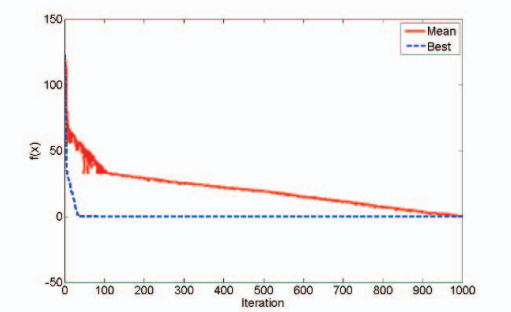


Fig. 5. The convergence history of Rastrigin with PICA (test 1)

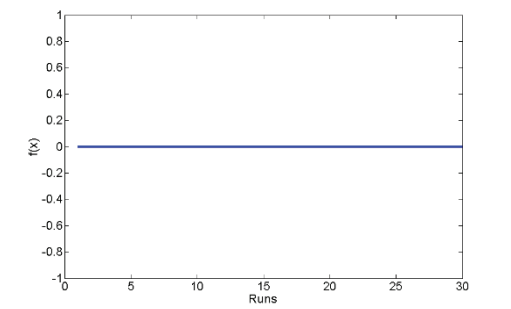


Fig. 6. The stability chart of Rastrigin with PICA (test 1)

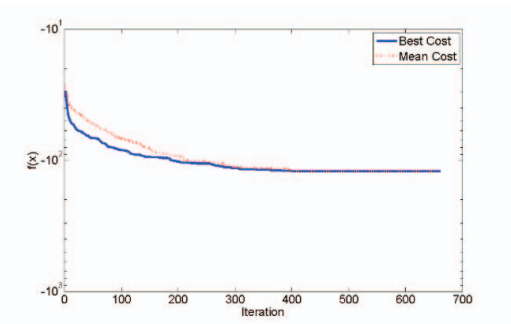


Fig. 7. The convergence history of test 2 with $D=100$ (from [24])

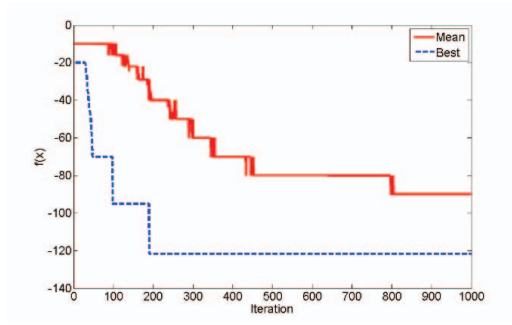


Fig. 8. The convergence history of PICA for test 2 with $D=100$

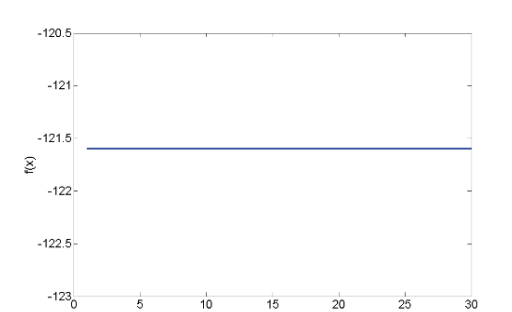


Fig. 9. The Stability chart of test 2 with $D=100$

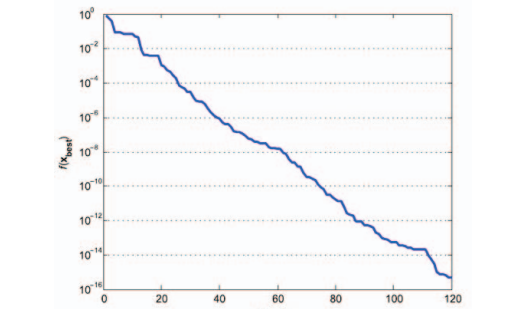


Fig. 10. The convergence history of case 1 (from [3])

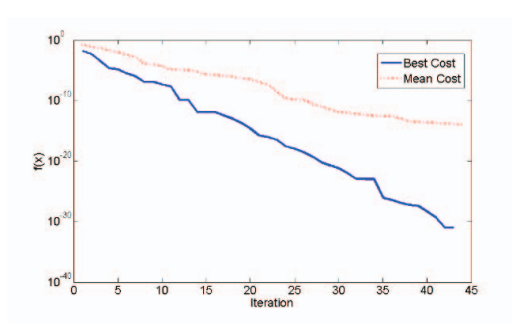


Fig. 11. The convergence history of case 1 (form [24])

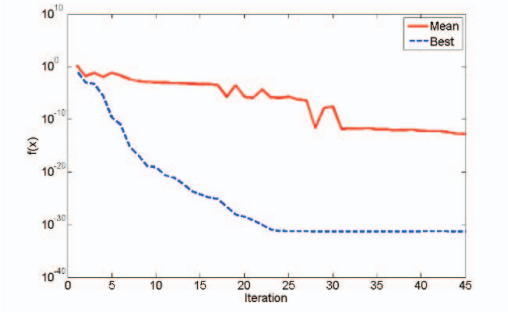


Fig. 12. The convergence history of case 1 with PICA

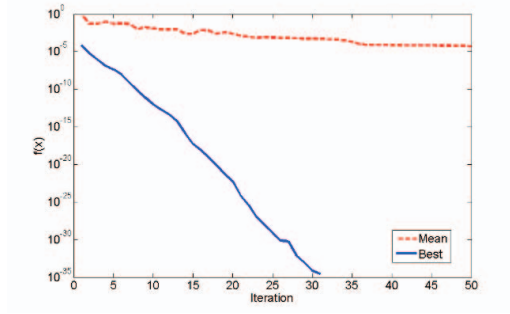


Fig. 16. The convergence history of case 2 with PICA

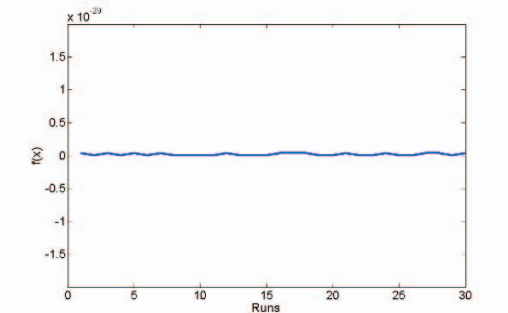


Fig. 13. The stability chart of case 1 with PICA

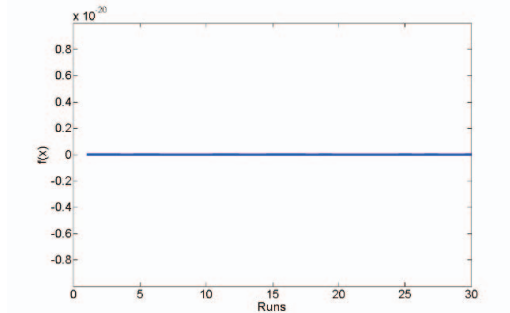


Fig. 17. The stability chart of case 2 with PICA

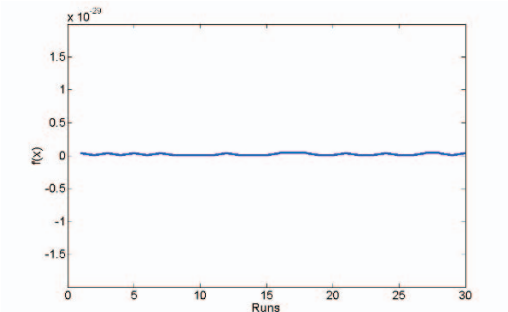


Fig. 14. The stability chart of case 1 with PICA

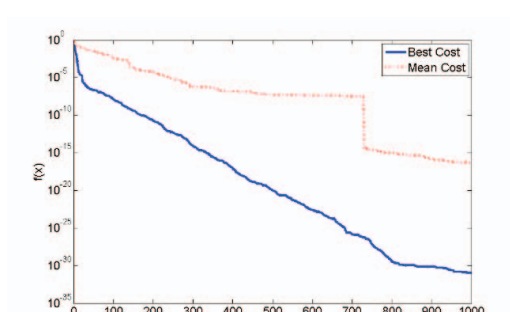


Fig. 18. The convergence history of case 3 (from [24])

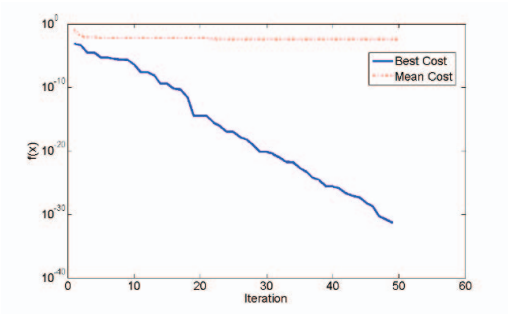


Fig. 15. The convergence history of case 2 (from [24])

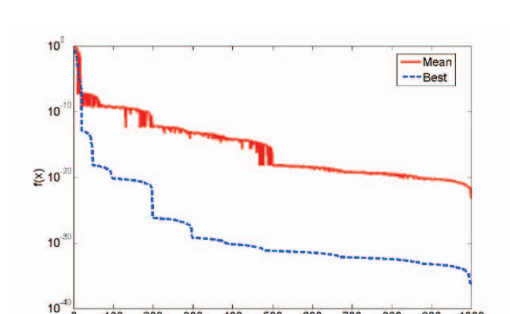


Fig. 19. The convergence history of case 3 with PICA

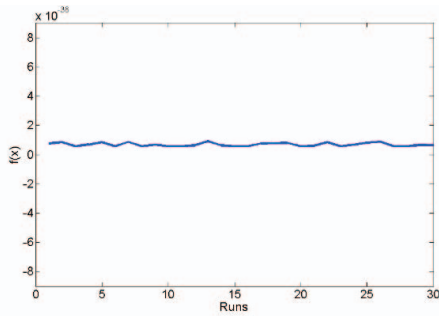


Fig. 20. The stability chart of case 3 with PICA

TABLE II. RESULTS OF TEST 1 WITH MO ET AL. (FROM [2])

Variables	Initial iteration	After 200 iterations	After 400 iterations	After 600 iterations	After 800 iterations	After 1000 iteration
x_1	0.1431	-0.0001	-0.0007	0.0001	-0.0000	-0.0000
x_2	2.1983	-0.0001	0.0000	0.0001	0.0001	0.0001
x_3	1.9401	0.0000	0.0000	0.0001	-0.0000	0.0001
x_4	-1.7080	-0.0002	-0.0001	0.0000	-0.0000	-0.0000
x_5	0.2261	-0.9950	-0.9962	-0.9948	0.0001	0.0001
x_6	0.9392	0.9950	0.9941	0.9949	0.9950	0.9949
x_7	-0.1129	0.9949	0.9949	0.0001	-0.0001	0.0000
x_8	-0.1516	0.9950	0.9949	0.9949	0.9949	-0.0000
x_9	-2.1893	-0.0001	0.0000	0.0001	-0.0000	-0.0000
x_{10}	4.9798	0.9950	0.0000	0.0001	-0.0000	-0.0000

TABLE III. RESULTS OF TEST 1 WITH ICA (FROM [24])

Variables	Initial iteration	After 200 iterations	After 400 iterations	After 600 iterations	After 800 iterations	After 1000 iteration
x_1	-2.883527	-0.1084e-007	-0.1404e-008	-0.1404e-008	-0.1404e-008	-0.1404e-008
x_2	-2.111072	0.1710e-007	0.0275e-008	0.0275e-008	0.0275e-008	0.0275e-008
x_3	-0.869045	-0.0048e-007	-0.0656e-008	-0.0656e-008	-0.0656e-008	-0.0656e-008
x_4	1.985114	0.6947e-007	0.0855e-008	0.0855e-008	0.0855e-008	0.0855e-008
x_5	-1.156667	0.0328e-007	-0.1015e-008	-0.1015e-008	-0.1015e-008	-0.1015e-008
x_6	3.083374	0.0356e-007	0.0899e-008	0.0899e-008	0.0899e-008	0.0899e-008
x_7	3.093877	-0.0948e-007	0.0349e-008	0.0349e-008	0.0349e-008	0.0349e-008
x_8	-2.020172	0.1528e-007	0.1610e-008	0.1610e-008	0.1610e-008	0.1610e-008
x_9	2.832951	-0.2304e-007	-0.0180e-008	-0.0180e-008	-0.0180e-008	-0.0180e-008
x_{10}	-2.208695	-0.2454e-007	0.0147e-008	0.0147e-008	0.0147e-008	0.0147e-008
$f(x)$	83.041615	1.3287e-012	0	0	0	0

TABLE IV. RESULTS OF TEST 1 WITH PICA (PRESENT STUDY)

Variables	Initial iteration	After 200 iterations	After 400 iterations	After 600 iterations	After 800 iterations	After 1000 iteration
x_1	1.928441	-4.716237e-011	-4.716237e-011	-4.716237e-011	-4.716237e-011	-4.716237e-011
x_2	-2.248101	2.157231e-011	2.157231e-011	2.157231e-011	2.157231e-011	2.157231e-011
x_3	1.341671	-1.001342e-011	-1.001342e-011	-1.001342e-011	-1.001342e-011	-1.001342e-011
x_4	0.728811	1.713127e-011	1.713127e-011	1.713127e-011	1.713127e-011	1.713127e-011
x_5	1.728128	-7.887191e-011	-7.887191e-011	-7.887191e-011	-7.887191e-011	-7.887191e-011
x_6	-2.839121	-2.837190e-012	-2.837190e-012	-2.837190e-012	-2.837190e-012	-2.837190e-012
x_7	1.871831	1.238291e-011	1.238291e-011	1.238291e-011	1.238291e-011	1.238291e-011
x_8	1.934281	-6.348271e-011	-6.348271e-011	-6.348271e-011	-6.348271e-011	-6.348271e-011
x_9	1.409124	8.119381e-011	8.119381e-011	8.119381e-011	8.119381e-011	8.119381e-011
x_{10}	0.365281	1.981381e-011	1.981381e-011	1.981381e-011	1.981381e-011	1.981381e-011
$f(x)$	1.241803e+002	0	0	0	0	0

TABLE V. COMPARISON RESULTS OF TEST 2 WITH D=100

$f(x)$	Initial iteration	After 100 iteration	After 200 iteration	After 300 iteration	After 400 iteration	After 500 iteration	After 600 iteration	After 700 iteration	After 800 iteration	After 900 iteration
ICA [24]	29.786871	-78.6467	-103.3897	-113.0125	-120.5298	-121.5923	-121.5979	-121.5982	-121.5982	-121.5982
PICA	-20.0000	-95.0017	-121.5982	-121.5982	-121.5982	-121.5982	-121.5982	-121.5982	-121.5982	-121.5982

TABLE VI. COMPARISON RESULTS OF PICA FOR CASE 1 WITH [3], [11], [24] AND [25]

Methods	x_1	x_2	$f(x)$
PPSO [3] and Gyrhan [11]	-0.29051455550725	1.08421508149135	4.686326815078573e-029
PPSO [3] and Gyrhan [11]	-0.793700525984100	-0.793700525984100	1.577721810442024e-030
COA [25]	1.08421508149135	-0.29051455550725	4.686326815078573e-029
COA [25]	-0.29051455550725	1.08421508149135	4.686326815078573e-029
ICA [24]	1.084215081491351	-0.290514555507251	3.562200025138631e-030
ICA [24]	-0.793700525984100	-0.793700525984100	1.577721810442024e-030
ICA [24]	-0.290514555507251	1.084215081491351	3.562200025138631e-030
PICA (present study)	1.0842150814913511	-0.2905145555072514	4.9303806576313238e-032
PICA (present study)	-0.79370052598409995582	-0.79370052598409995582	3.9443045261050590e-031
PICA (present study)	-0.2905145555072514	1.0842150814913511	4.9303806576313238e-032

TABLE VII. STATISTICAL RESULTS

Problem	N	Mean	Std. Deviation	Std. Error Mean	Worst	Best
Test 1	30	0.0	0.0	0.0	0.0	0.0
Test 2	30	-1.2159819999999999e+002	7.226896453227138e-014	1.319444736062194e-014	-1.215982000000000e+002	-1.215982000000000e+002
Case 1	30	1.9885860000000001e-031	1.739458967563944e-031	3.175803407964731e-032	3.944300000000000e-031	4.930400000000000e-032
Case 2	30	1.046312443884771e-026	1.511543708264576e-026	2.759688618905053e-027	4.4145000000000001e-026	0.0
Case 3	30	6.8980499999999997e-037	1.150107106181705e-037	2.099798685342363e-038	9.039099999999999e-037	5.800000000000000e-037

TABLE VIII. COMPARISON RESULTS OF CASE 2

Methods	X	Variable values	f	Functions values	F(x)
The best in [8]	x_1	0.500432850000000	f_1	-0.000238520000000	7.693745216994211e-008
	x_2	3.141863170000000	f_2	0.000141590000000	
The best in [13]	x_1	0.299450000000000	f_1	6.139739265609290e-007	1.014347133848949e-012
	x_2	2.836930000000000	f_2	-7.983627943186633e-007	
	x_1	0.500000000000000	f_1	2.111655261760603e-007	
	x_2	3.141590000000000	f_2	-2.296034435467220e-006	
The best in COA [25]	x_1	0.299300000000000	f_1	-7.128922385554737e-005	5.792081721117691e-009
	x_2	2.836600000000000	f_2	2.664447941302939e-005	
The best in ICA [24]	x_1	0.299448692495720	f_1	1.305289210051797e-012	5.631272867601562e-024
	x_2	2.836927770471037	f_2	2.284838984678572e-013	
	x_1	0.500000000000000	f_1	0	
	x_2	3.141592653589794	f_2	0	
The best of PICA	x_1	0.2994486924902598	f_1	-1.387778780781446e-016	6.856310602018560e-032
	x_2	2.8369277704589400	f_2	2.220446049250313e-016	
	x_1	0.500000000000000	f_1	0	
	x_2	3.141592653589794	f_2	0	

TABLE IX. COMPARISON RESULTS OF CASE 3

Methods	x	Variables values	f	Functions values
The best in [8]	x_1	0.67154465	f_1	-0.00000375
	x_2	0.74097111	f_2	0.00001537
	x_3	0.95189459	f_3	0.00000899
	x_4	-0.30643725	f_4	0.00001084
	x_5	0.96381470	f_5	0.00001039
	x_6	-0.26657405	f_6	0.00000709
	x_7	0.40463693	f_7	0.00000049
	x_8	0.91447470	f_8	-0.00000498
The best in [13]	x_1	0.1644	f_1	-8.8531e-005
	x_2	-0.9864	f_2	3.5894e-005
	x_3	-0.9471	f_3	6.6216e-006
	x_4	-0.3210	f_4	2.1560e-005
	x_5	-0.9982	f_5	1.2320e-005
	x_6	-0.0594	f_6	3.9410e-005
	x_7	0.4110	f_7	-6.8400e-005
	x_8	0.9116	f_8	-6.4440e-005
The best of ICA [24]	x_1	0.164431665854327	f_1	2.775557561562891e-016
	x_2	-0.986388476850967	f_2	-1.110223024625157e-016
	x_3	0.718452601027603	f_3	-1.110223024625157e-016
	x_4	0.718452601027603	f_4	1.734723475976807e-018
	x_5	0.997964383970433	f_5	0
	x_6	0.063773727557003	f_6	0
	x_7	-0.527809105283546	f_7	0
	x_8	-0.849363025083964	f_8	0
The best of PICA	x_1	0.164431665854327405	f_1	5.368529659036217811e-019
	x_2	-0.986388476850967110	f_2	2.548307417523678423e-019
	x_3	0.718452601027603350	f_3	-3.378192205891815512e-019
	x_4	-0.695575919707310931	f_4	3.389211820587187123e-019
	x_5	0.997964383970432520	f_5	0
	x_6	0.063773727557002571	f_6	0
	x_7	-0.527809105283546241	f_7	0
	x_8	-0.849363025083964123	f_8	0

TABLE X. THE COMPARISON STATISTICAL RESULTS OF SERIAL ICA [24] AND PICA

Problem	Speed Up	Efficiency	Serial ICA time	PICA time	#processors	Super linear performance
Test 1	4.02	2.01	3.38	0.84	2	Yes
Test 2	7.22	3.61	11.82	1.63	2	Yes
Case 1	2.82	1.41	0.0341	0.01	2	Yes
Case 2	5.1	2.55	2.1	0.41	2	Yes
Case 3	6.24	3.12	6.78	1.08	2	Yes

VII. CONCLUSION AND FUTURE WORKS

In this paper, the parallel imperialist competitive algorithm based on the MPI instructions (Multi-Population) was utilized

to solve the systems of nonlinear equations. The PICA was compared with the serial ICA and some of the other proposed methods. According to the obtained results, the PICA is suitable for solving different kinds of complex problems, and it is faster and more efficient than the other methods. The figures indicated that the answers of our algorithm are stable and the convergence of the PICA to the best solution is faster than the other methods, with the lower number of iterations and better run time. As a result, we claim that the proposed PICA is a faster and more accurate method, which can be employed to solve and improve the complex problems. At the end, our future works will consist of using the proposed parallel algorithm to solve some of the more practical optimization problems, like constrained engineering optimization

REFERENCES

- [1] Y.Z. Luo, G.J. Tang, L.N. Zhou, Hybrid approach for solving systems of nonlinear equations using chaos optimization and quasi-Newton method, Appl. Soft. Comput. 8 (2008) 1068-1073. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68-73.
- [2] Y. Mo, H. Liu, Q. Wang, Conjugate direction particle swarm optimization solving systems of nonlinear equations, Comput. Math. Appl. 57 (2009) 1877-1882.
- [3] M. Jaberipour, E. Khorram, B. Karimi, Particle swarm algorithm for solving systems of nonlinear equations, Comput. Math. Appl. 62 (2011) 566-576.
- [4] E. Cantu-Paz, A Survey of Parallel Genetic Algorithms, Department of Computer Science and Illinois Genetic Algorithms Laboratory University of Illinois at Urbana-Champaign, 1997.
- [5] H. Liu, P. Li, and Y. Wen, Parallel Ant Colony Optimization Algorithm, World Congress on Intelligent Control and Automation, China, June, 2006.
- [6] R. Parpinelli, C. Benitez, and S. Lopes, Parallel Approaches for the Artificial Bee Colony Algorithm, Handbook of Swarm Intelligence, 8(2010) 329-345.
- [7] C. Grosan, A. Abraham, A New Approach for Solving Nonlinear Equations Systems, IEEE Trans. Syst. Man Cybern. A 38 (3) (2008) Senior Member, IEEE.
- [8] C. Wang, R. Luo, k. Wu, B. Han, A new filled function method for an unconstrained nonlinear equation, Comput. Appl. Math. 235 (2011) 1689-1699.
- [9] L. Vanneschi, D. Codecasa, and G. Mauri, A Comparative Study of Four Parallel and Distributed PSO Methods, New Generat. Comput. 29(2011) 129-161.
- [10] J. Dugalakis, and K. Margaritis, A Parallel Memetic Algorithm for Solving Optimization Problems, 4th Metaheuristics International Conference, Parallel Distributed Processing Laboratory, Greece, 2001.

- [11] Gyurhan H. Nedzhibov, A family of multi-point iterative methods for solving systems of nonlinear equations, *J. Comput. Appl. Math.* 222(2008) 244250.
- [12] E. C. G. Wille, E. Y. H. S. Lopes, Discrete Capacity Assignment in IP networks using Particle Swarm Optimization, *Appl. Math. Comput.*, 217 (2011) 5338-5346.
- [13] C. A. Floudas, P. M. Pardalos, C. S. Adjiman, W. R. Esposito, Z. H. Gumus, S. T. Harding, J. L. Klepeis, C. A. Meyer, C. A. Schweiger, *Handbook of Test Problems in Local and Global Optimization*, Kluwer Academic Publishers, Dordrecht, the Netherlands, 1999.
- [14] A. Mousa, W. Wahed, R. Allah, A Hybrid Ant Colony Optimization Approach Based Local Search Scheme for Multi Objective Design Optimizations, *Electr. Pow. Syst. Res.* 81 (2011) 1014-1023.
- [15] Ibrahim M. M. El-Emary and Mona M. Abd El-Kareem, Toward Using Genetic Algorithm for Solving Nonlinear Equation Systems, *World Appl. Sci. J.* 5 (2008) 282-289.
- [16] M. Abdollahi, A. Isazadeh, D. Abdollahi, Solving systems of nonlinear equations using imperialist competitive algorithm, *The 8th International Industrial Engineering Conference*, 8 (2012) 1-6.
- [17] Nikos E. Mastorakis, Solving Non-linear Equations via Genetic Algorithms, *Proceedings of the 6th WSEAS Int. Conf. on Evolutionary Computing*, Lisbon, Portugal, June 16-18 (2005) 24-28.
- [18] G. Li, Zh. Zeng, A neural-network algorithm for solving nonlinear equation systems, *IEEE International Conference on Computational Intelligence and Security*, CIS08 (2008) 20-23.
- [19] G. Huan-Tong, S. Yi-Jie, S. Qing-Xi, W. Ting-Ting, Research of Ranking Method in Evolution Strategy for Solving Nonlinear System of Equations, *IEEE International Conference on Information Science and Engineering*, ICISE09 (2009) 348-351.
- [20] A. Ouyang, Y. Zhou, Q. Luo, Hybrid Particle Swarm Optimization Algorithm for Solving Systems of Nonlinear Equations, *IEEE International Conference on Granular Computing*, GRC09 (2009) 460-465.
- [21] J. Wu, Zh. Cui, J. Liu, Using Hybrid Social Emotional Optimization Algorithm with Metropolis Rule to Solve Nonlinear equations, *IEEE International Conference on Cognitive Informatics & Cognitive Computing*, ICCI*CC'11 (2011) 405-411.
- [22] N. Henderson, W. F. Sacco, G. Mendes Platt, Finding more than one root of nonlinear equations via a polarization technique: An application to double retrograde vaporization, *Chem. Eng. Res. Des.* 88 (2010) 551-561.
- [23] E. Pourjafari, H. Mojallali, Solving nonlinear equations systems with a new approach based on invasive weed optimization algorithm and clustering, *Swarm Evol. Comput.* 4 (2012) 3343.
- [24] M. Abdollahi, A. Isazadeh, D. Abdollahi, Imperialist competitive algorithm for solving systems of nonlinear equations, *Comput. Math. Appl.* 65 (2013) 1894-1908.
- [25] M. Abdollahi, Sh. Lotfi, D. Abdollahi, Solving systems of nonlinear equations using cuckoo optimization algorithm, *The 3rd International conference on The Contemporary Issues in Computer Sciences and Information Technology (CICIS)*, 3 (2012) 191-194.
- [26] E. Atashpaz-Gargari, C. Lucas, Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition, in: *IEEE Congress on Evolutionary Computation*, 2007, pp. 4661-4667.
- [27] A. Majd, Sh. Lotfi and G. Sahebi, "Review on Parallel Evolutionary Computing and Introduce Three General Framework to Parallelize All EC Algorithms," *5th Conference on Information and Knowledge Technology (IKT)*, 2013.
- [28] A. Majd, Sh. Lotfi, G. Sahebi, M. Daneshlab and J. Plosila, "PICA: Multi-Population Implementation of Parallel Imperialist Competitive Algorithms," *24th Euromicro International Conferences on Parallel, Distributed and Network-Based Processing*, PDP 2016.
- [29] M. Abdollahi, A. Bouyer, D. Abdollahi, Improved cuckoo optimization algorithm for solving systems of nonlinear equations, *J. Supercomput.* 72 (2016) 1246-1269.

Paper IV

Finding Near-Optimal Task Scheduling for Distributed Real-Time Environments

M. Salimi, **A. Majd**, M. Loni, C. Seceleanu, T. Seceleanu, M. Sirjani, M. Daneshtalab, and E. Troubitsyna

Multi-objective Optimization of Real-Time Task Scheduling Problem for Distributed Environments

Maghsood Salimi
msalimi@khayam.ut.ac.ir
Tehran University
Tehran, Iran

Amin Majd
amajd@abo.fi
Department of Information
Technology, Åbo Akademi University
Turku, Finland

Mohammad Loni
mohammad.loni@mdh.se
School of Innovation, Design and
Engineering, Mälardalen University
Västerås, Sweden

Tiberiu Seceleanu
tiberiu.seceleanu@mdh.se
School of Innovation, Design and
Engineering, Mälardalen University
Västerås, Sweden

Cristina Seceleanu
cristina.seceleanu@mdh.se
School of Innovation, Design and
Engineering, Mälardalen University
Västerås, Sweden

Marjan Sirjani
marjan.sirjani@mdh.se
School of Innovation, Design and
Engineering, Mälardalen University
Västerås, Sweden

Masoud Daneshtalab
masoud.daneshtalab@mdh.se
School of Innovation, Design and
Engineering, Mälardalen University
Västerås, Sweden

Elena Troubitsyna
elena.troubitsyna@abo.fi
KTH Royal Institute of Technology
Stockholm, Sweden

ABSTRACT

Real-world applications are composed of multiple tasks which usually have intricate data dependencies. To exploit distributed processing platforms, task allocation and scheduling, that is assigning tasks to processing units and ordering inter-processing unit data transfers, plays a vital role. However, optimally scheduling tasks on processing units and finding an optimized network topology is an NP-complete problem. The problem becomes more complicated when the tasks have real-time deadlines for termination. Exploring the whole search space in order to find the optimal solution is not feasible in a reasonable amount of time, therefore meta-heuristics are often used to find a near-optimal solution.

We propose here a multi-population evolutionary approach for near-optimal scheduling optimization, that guarantees end-to-end deadlines of tasks in distributed processing environments. We analyze two different exploration scenarios including single and multi-objective exploration. The main goal of the single objective exploration algorithm is to achieve the minimal number of processing units for all the tasks, whereas a multi-objective optimization tries to optimize two conflicting objectives simultaneously considering the total number of processing units and end-to-end finishing time for all the jobs. The potential of the proposed approach is demonstrated by experiments based on a use case for mapping a number of jobs covering industrial automation systems, where each of the jobs consists of a number of tasks in a distributed environment.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Scheduling*; **Sensor**; • **Computer Science** → **Optimization**.

KEYWORDS

Distributed Task Scheduling, Real-Time Processing, Evolutionary Computing, Multi-Objective Optimization

ACM Reference Format:

Maghsood Salimi, Amin Majd, Mohammad Loni, Tiberiu Seceleanu, Cristina Seceleanu, Marjan Sirjani, Masoud Daneshtalab, and Elena Troubitsyna. 2019. Multi-objective Optimization of Real-Time Task Scheduling Problem for Distributed Environments. In *Proceedings of ECBS '19*. ACM, September 02 - 03, 2019, Bucharest, Romania, 9 pages. <https://doi.org/10.1145/3352700.3352713>

1 INTRODUCTION

Industrial applications often require guaranteeing real-time execution, fault tolerant implementations and providing reliable functionality. In general, it is impossible for a single processing unit to satisfy all these needs. However, a distributed processing environment provides a variety of computational capabilities, which can be utilized to perform an application that has diverse execution requirements. An application job can be decomposed into tasks. Tasks may have data dependencies and it is possible that each task needs a certain computational throughput. For distributing tasks, the following decisions should be made respectively: ① task allocation, i.e. assigning tasks to processing units, and ② tasks scheduling, i.e. defining task execution order and the order of data transfers among processing units. The general goal of task allocation and scheduling is to minimize the end-to-end cost of computation, i.e. minimizing overall response time of the application, minimizing the number of processing units, or both.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

ECBS '19, September 02 - 03, 2019, Bucharest, Romania

© 2019 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-7636-5.

<https://doi.org/10.1145/3352700.3352713>

Performance of such parallel systems can be optimized by employing an efficient task allocation and scheduling approach, however, the allocation and scheduling problem is an NP-complete problem [1]. Using exhaustive approaches for finding optimal solution is time-consuming and is impossible in practice. Many heuristic task scheduling strategies have been proposed [2, 3] to find a near-optimal solution in a reasonable amount of time. Evolutionary Computing (EC) is a set of methods proposed to solve the allocation and scheduling problem. Genetic Algorithm (GA) is a popular EC method which can better locate a near-optimal solution than other similar approaches in most cases [4–7]. Although GA is a powerful solution, defining a proper fitness function is always challenging and requiring expertise especially when the size of design space is huge. Plus, GA is relatively slow and may be trapped in local optima [8].

To overcome aforementioned challenges, a Multi-Population Genetic Algorithm (MPGA) [9] is leveraged in this research for task allocation and scheduling over a collection of nonuniform processing units. MPGA is a static scheduling strategy, where the execution times of tasks and the data transfer times between tasks are known. MPGA is the parallel version of GA that provides better convergence rate and more speedup compared to single population GA [8]. In addition, MPGA highly reduces the probability of falling into local optima trap. Two different MPGA strategies have been considered to solve the allocation and scheduling problem including: ① *Single objective optimization* and ② *Multi-objective optimization*. While the single objective optimization minimizes the number of processing units, the multi-objective optimization considers the second conflicting metric, jobs end-to-end finishing time, to find solutions satisfying multiple user needs.

Contribution. In a nutshell our main contributions are:

- In this paper, we solved task allocation and scheduling problem in a distributed environment. To attain this purpose, we leveraged a MPGA optimization method with different optimization scenarios.
- Defining novel fitness functions to efficiently explore the design space in both single and multi-objective optimization scenarios.
- The evaluation results based on an industrially inspired use case show the impact of the proposed fitness function while converging to better solutions.

Paper Organization. The paper is organized as follows. Section II defines the allocation and scheduling problem and our use case. Section III explains the MPGA and the specifications of fitness functions for both single objective and multi-objective optimization scenarios. Section IV presents the experimental results and demonstrates the efficiency and convergence of the proposed algorithm. Some related work reviewed in Section V. We end with concluding remarks and future work in Section VI.

2 PROBLEM DEFINITION

We start here by describing a generic distributed process control system. In such systems, a series of computing devices operate on data collected from sensors placed close to a physical process, and update control signals to other devices - actuators - able to control the evolution of the process. A process may be exemplified by a

simple tank-filling operation or by more complex systems, such as ore separation, water purification, etc. The process parameters (such as liquid levels, temperatures, etc.) are usually required to be maintained within a certain range of values, even when the environment is disturbed. Whenever new values are presented via sensors to the processing devices, certain procedures hosted within these devices are launched, and potential new values are sent to the process-responsible actuators. In large systems, there are potentially thousands or more such procedures, installed in tens to hundreds of *control devices*.

The main problem that we raise here is how to allocate the number of processing operations on an as small as possible set of processing devices, such that planned operations are not affected with respect to their timing and duration, and the processing devices are operating within their nominal characteristics.

In order to cover most of the aspects of interest when solving this, in the following we employ a synthetic example of a system as use case, with elements presented in Fig. 1. Here, we have a control system composed of 8 jobs, their characteristics and further decomposition being detailed below.

2.1 System Model Elements

The system we consider is composed of a number of complex control processes, referred from now on as *jobs*. The system reads data from a set of input elements - the *sensors* S_1, \dots, S_{11} and processes the data on a number of available *processing units* (P_1, \dots, P_{24}). The processed data is sent further in the system to other elements - the *actuators* A_1, \dots, A_{11} - notice that having a similar number of sensors and actuators is a coincidence of no relevance in the analysis to come. A job refers to the data trip from sensors to actuators.

A job can be further described as a collection of *tasks*, acting mostly sequentially, but not excluding parallel processing - especially if tasks belong to different jobs. To illustrate a more critical situation, we assume that all the considered tasks are non-interruptible. A task is a unitary, and with the assumed non-interruptible characteristic, an *atomic* system element, to be executed on one of the available processing units. Each task has input either a sensor, or the output of a precedent task. At the output of a task stays either an actuator, or the input of a follower task. Multiple inputs to a task are possible (see task T13 in Job8 on actuator A10), in which case all of them must be present for the task to start its operation. Differently, if a task has more than one output (see task T21 in Job7), all of them are presented at the same moment.

Fig. 1 describes additional information pertaining to task and job execution, as well as some characteristics of interest for the processing units. Thus, a task is also defined with a potential maximal load that it presents to the processing unit, and with a maximal execution time. For instance, task T1 in Fig. 1 produces a maximal load of 10, and it executes in maximum 10tu ("time units": μ -seconds to seconds, for instance. However, an actual specification of these units is not of interest in our work here). At the same time, the available processing unit P1 can withhold a maximum load of 100, and possess 5 connection interfaces.

In their turn, the jobs have an execution time (the sum of the execution times of the composing tasks), and a frequency: how

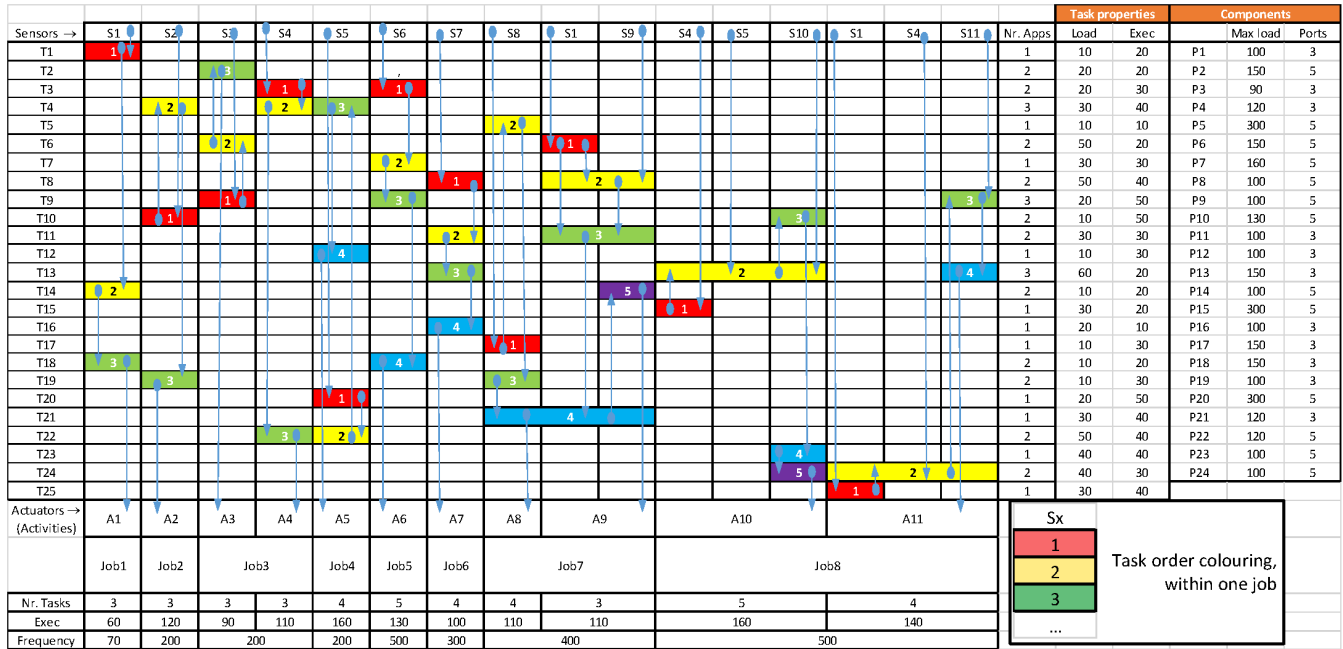


Figure 1: Representing The use case including jobs, intra-task dependencies, tasks load complexity, real-time deadlines and processing unit specifications.

often data is read from sensors, and it has to be sent to the actuators. For instance, Job1 - a sequence of T1, T14 and T18 - has a maximal execution time of 60tu, and it is recurring every 70tu. A more complex situation is presented by jobs 3, 7 and 8, where two actuators are related to each job. The times for processing the data corresponding to each actuator may be different, but what holds them together is the execution frequency (200tu, 400tu and 500tu, per job, respectively). Fig. 2 shows the dependency graph between tasks of a job example from the use case (see Section II.A).

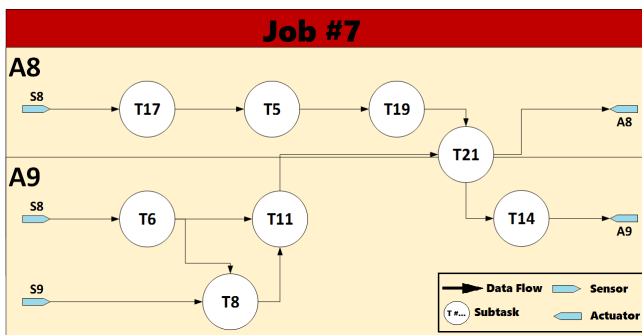


Figure 2: Dataflow of Job #7.

2.2 Problem Assumption

To only focus on the allocation and scheduling problem, we made the following additional assumptions. First of all, we assumed that each task is written in a machine-independent language. Moreover,

it is assumed that we know all the data dependencies among tasks before execution (as described by Fig. 1 and partially by Fig. 2). The distributed processing platform is nonuniform, consisting of multiple homogeneous processing units with various processing potential.

If a data conditional is based on input data, it is assumed to be contained inside a task. A loop that uses an input data item to determine one or both of its bounds is also assumed to be contained inside a task. When two communicating tasks are mapped onto the same processing units we assume that the communication delay is zero. However, when they are mapped onto different processors a finite communication delay is assumed and modeled by 1tu.

Moreover, we do not (yet) consider here aspects related to reliability, fault tolerance, safety, etc. These aspects may (such as in the case of fault tolerance) require a duplication of allocation and synchronization of data across duplicated locations. These additional objectives are subject of further work analysis.

3 MPGA DESCRIPTION

GA is an iterative population-based exploration solution mimicking the process of natural selection and evolution where the characteristics of the process can be utilized in solving optimization problems. All GA-based methods have an initial population where selection, crossover, mutation operators are applied to initial population for producing improved population. The operations will be repeated until satisfying user criteria (reaching suitable results) or stopping after a predefined number of iterations. The following subsections explain the basic components of GA.

Step 1. Generating Initial Population. The initial population includes random solutions in the design space, where each solution

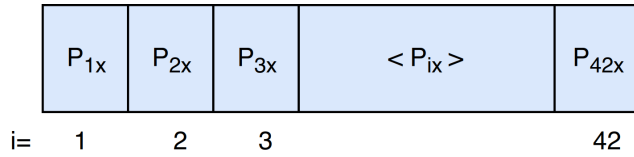


Figure 3: Representing a valid allocation and scheduling by GA/MPGA chromosome type.

represented by chromosome is a schedule for all the jobs. The size of initial population depends on the size of design space. To check the validity of solutions in the initial population, each solution is examined by using the objective function represented in Equation (1). Invalid solutions will be removed from the population.

Step 2. Fitness Evaluation. Objective function (fitness function) is a metric for comparing different scheduling that satisfy problem constraints. Equation (1) and Equation (2) represent the fitness functions for single-objective and multi-objective optimization, respectively.

$$Fitness_1 = \#Processors + (\gamma \times (\alpha + \beta + \theta)) \quad (1)$$

$$Fitness_2 = \frac{\#Processors}{\gamma} + \frac{Run - time}{BiggestDeadline} + 3 \times (\alpha + \beta + \theta) \quad (2)$$

where α is the total extra loads of the all assigned tasks that exceed the load of processing units, β is the total extra deadline of all assigned tasks that exceed the real-time deadlines, θ is the total extra ports of all job assignments that exceed the total number ports per processing units, and γ is equal to 23, the total number of processing units. *BiggestDeadline* is the maximum possible time for finishing the slowest job. The scale of extra load (α), extra deadline (β), and extra ports (γ) could be very different, thus all the α , β , and γ parameters should be normalized. However, we did not normalized them since the range of these parameters are deterministic and the fitness functions are customized for the studied use case.

In (1), minimizing the number of processors (*#Processor*) is the exploration objective. Whereas in (2), minimizing both the end-to-end finishing time of all the jobs (*Run - time*) and *#Processor* are the exploration objectives.

Step 3. Selection. Obviously the schedules with better fitness function are selected as the next generation and the others will be removed from population set. The goal is to find a solution in design space with lowest fitness function in both Equations (1) and (2).

Step 4. Crossover Operator. Is the most important operator of GA. GA randomly selects two genomes from the population set based on a certain crossover rate. Then two genome strings exchange parts of their corresponding chromosomes to create two new genomes. In our use case, the chosen scheduling are exchanged with the other scheduling for producing two new schedules with most likely better schedules. Fig. 3 illustrates the representation of the all jobs scheduling by a genome type. Each genome consists of 42 portions since the use case has 42 different tasks. All possible assignments

to processing units for each task is $\gamma=23$. This representation also indicate the task scheduling by prioritizing the assigned tasks to the same processor. Such that the processor operates on the tasks from left to right i.e, if Task #4 (T4), Task #7 (T7)and Task #11 (T11) are assigned to Processor #2 (P2), the processor first runs T4, then T7 and T11 respectively.

Step 5. Mutation Operator. The main goal of mutation operator is to increase genetic diversity. Mutation alters one gene value (assigned processor to task) in a chromosome string from its initial state. The solution may be better or even worst solution by using mutation. Mutation forces GA to get rid of local optima. For doing mutation, we need to randomly select one gene in chromosome and modify its assigned value to a new valid number.

After each cycle of selection, crossover and mutation, the newly generated set of solutions (schedules) is called as new generation. All the generations are evaluated based on the fitness function to determine if they represent a good enough solution to satisfy the fitness function. This determines if the GA can stop searching, or if otherwise, for the GA to continue searching until the predefined stopping criteria is met. The stopping criteria could be the number of generations, or evolution time, or fitness threshold, or fitness convergence, or population convergence. In our case, the number of generations was set as the stopping criteria. The schedule obtained after the stopping criteria will be the optimal or near optimal schedule.

3.1 MPGA Algorithm

Although EC methods can improve the quality of results, using them have some difficulties. First of all, an evolutionary algorithm may not converge towards the optimal solutions or even to near-optimal solutions in the case of very huge exploration space. One possible solution is to increase the initial population size, but leading to increase the execution time of evolutionary algorithms. Parallelizing these algorithms can remarkably diminish their execution time and improve the quality of results. In the parallelized GA, multiple processors work together where each one runs a simple GA and has an independent populations.

Step 6. After a predefined number of iterations, all processors share their best chromosomes among each other (**migration operation**).

Step 6 above is specific to the parallel procedure, which, including the previous 5 steps is called MPGA. Sharing the best individuals aids the MPGA to get avoid of local optima. This procedure comes to be utilized in the algorithm that we propose in further.

Fig. 4 represents the behavior of the MPGA and the flowchart of consequent operations is shown in Fig. 5. The pseudo-code of MPGA is presented in Algorithm 1. The inputs of proposed meta-heuristic optimization approach include: ① the specification of processing units including maximum processing potential, the total number of input/output ports, and ② the specifications of jobs and tasks including load complexity, run-time deadlines, and task dependencies.

4 EVALUATIONS

This section presents the results of experiments that have been fulfilled to evaluate the impact of the proposed MPGA on the use

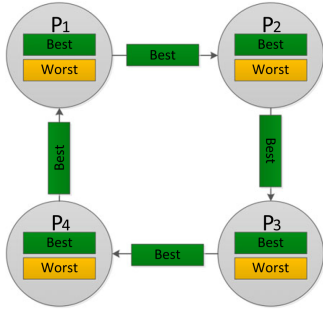


Figure 4: Multi-population migration operation.

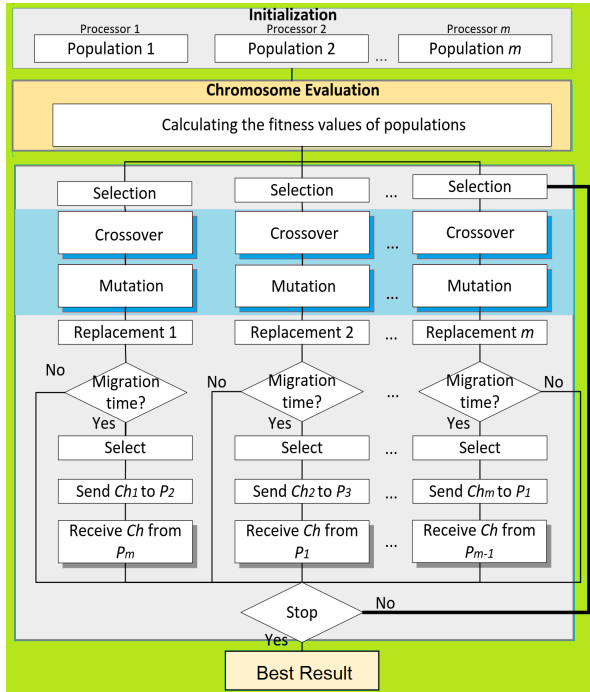


Figure 5: Flowchart of MPGA.

case. The evaluations have been done based on two different optimization scenarios including single objective and multi-objective optimization. *It is necessary to mention that the single objective optimization has been solved with simple GA, while we leveraged MPGA to solve the multi-objective optimization.*

4.1 Implementation Details

MPGA is implemented in C++ and MPI library has been utilized for parallelization. Ring topology is used for connections between processors for running MPGA. For the implementations, an Intel Core i7-4770 CPU 3.40 GHz with 16.0 GB RAM running on 64-bit Windows 10 has been used. Seven cores have been leveraged in the parallel implementation. The specification of MPGA parameters is shown in Table 1.

Algorithm 1: Pseudo Code of MPGA

Input: • Processor P_i ; $1 \leq i \leq \#$ Processors

- Distributed processing units Specifications
- Jobs and related tasks Specifications
- N : Population Size
- T : Maximum Number of Iterations

Output: A Set of Near-Optimal Solutions

Function MPGA(N, T):

```

(Step 1):  $U_{i,0} = \text{Random\_Population}(N)$ ; //Creating
initial random population and assign to each  $P_i$ 
(Step 2):  $\text{Fitness\_Function}(U_{i,0})$ ; //Evaluating the
objectives of each solution in the all populations
 $t = 1$ ;
while  $t \leq T$  | SatisfyingUserNeeds do
(Step 3):  $U'_{i,t} = \text{Select}(U_{i,t})$ ; //Select some
chromosomes from the  $U_{i,t}$  randomly.
(Step 4):  $U''_{i,t} = \text{Crossover}(U'_{i,t})$ 
(Step 5):  $Y_{i,t+1} = \text{Mutation}(U''_{i,t})$ 
(Step 2):  $\text{Fitness\_Function}(Y_{i,t+1})$ ;
(Step 6): if #Iterations%MigrationRate == 0 then
    Select the best chromosome from  $Y_{i,t+1}$ 
    Send the best chromosome to  $P_{i+1}$ 
    Receive the best chromosome from  $P_{i-1}$ 
 $t = t + 1$ ;
return  $Y_{i,t+1}$ 

```

Table 1: MPGA Algorithm Parameters.

Parameter	Value
N: Initial Population Size (Each Processor)	100
# Populations	15
Maximum # Iterations	750
Crossover Rate	{0.1, 0.5, 0.9} per each 5 populations
Mutation	One-Point Mutation
Migration Rate	3
Migration Gap	25
Mutation Rate	1 - Crossover Rate

Table 2: Experimental Results Compared to [11].

Exploration Approach	End-to-end finishing time	# Processing units
Our Single Objective Equation(1) (Solved by simple GA)	250	7
Our Multi-Objective Equation(2) (Solved by MPGA)	Solution①: 210	7
	Solution②: 250	8
	Solution③: 160	9
Single Objective [11]	210	9
Multi-Objective [11]	180	11

4.2 Experimental Results Convergence

One of the main limitations of evolutionary algorithms is decreasing the convergence speed by increasing the number of iterations leading to make non-convergent results in low iterations for difficult problems. Fig. 6 and Fig. 7 represent the convergence of fitness

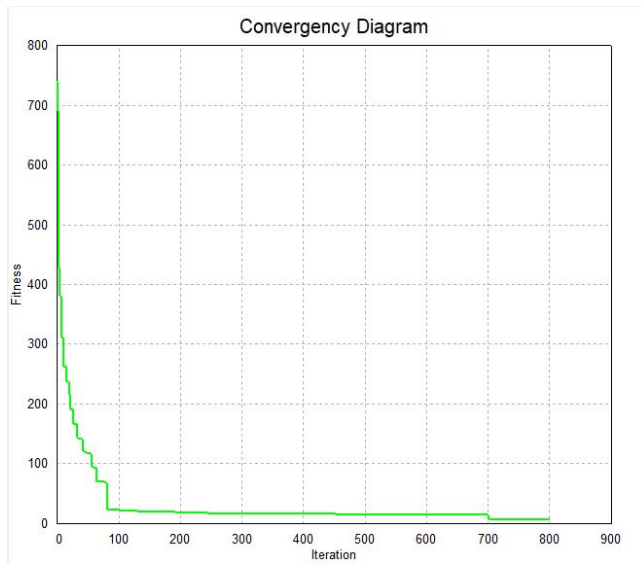


Figure 6: Convergence diagram of the fitness function for the single objective optimization ((1)).

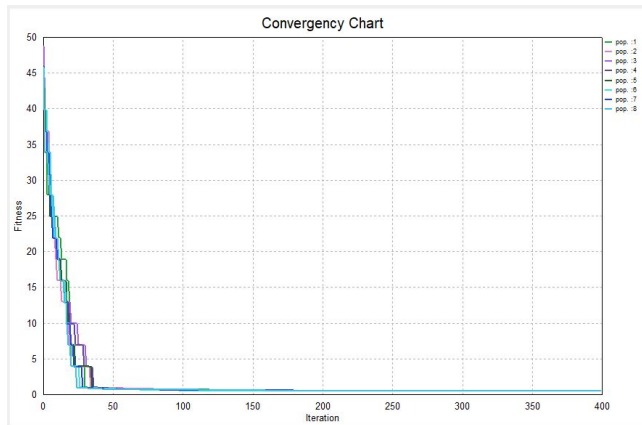


Figure 7: MPGA convergence diagram of the fitness function for the eight different populations ((2)).

functions for both single and multi-objective optimization, respectively. It can be easily observed from the convergence figures that both strategies are highly convergent toward the improved results by contentious reduction in fitness functions as the system cost (see Equation (1) and Equation (2)).

a) Single Objective Optimization. Fig. 8 illustrates the variation trend of total number of utilized processing units over the number of iterations. As mentioned before, the aim of single objective optimization is to decrease the number of processing units used in jobs scheduling. Fig. 8 shows considerable improvement in finding scheduling with less required processing units. According to the results of Table 2, we need 22 processing units for scheduling in the first iteration, while by proceeding the exploration algorithm, we found a solution with only seven required processing units.

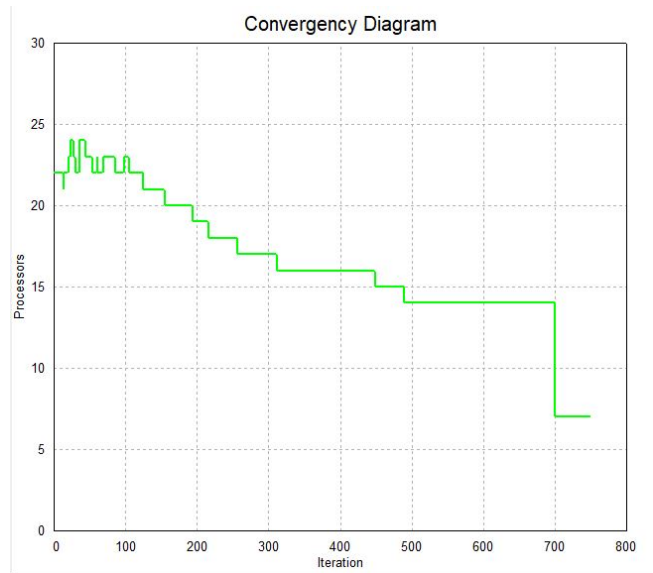


Figure 8: Convergence diagram of the variations # processing units in single objective optimization solved by simple GA.

Although there exist some breaks in continuous improvement, the overall trend moves toward improvement.

b) Multi-Objective Optimization. As mentioned before, the total number of processing units and end-to-end finishing time (represented as *Run - time* in (2)) for all the jobs are the two main objectives of MPGA. Fig. 9 and Fig. 10 illustrate the convergence figures of required processing units for scheduling and end-to-end finishing time for all the scheduled jobs, respectively. We can conclude from the figures that both the objectives are approaching toward optimized results. Although there are some failures or stops in achieving better results in each iteration, the overall Progression of MPGA always approaches toward superior outcomes (Fig. 11).

Table 2 shows three different solutions on the Pareto frontier of the last Population. We have a variety of options based on the user needs. Solution① is a schedule with minimized number of processing units (7 processing units) while takes more time, 210tu, for running. On the other hand, Solution③ provide the minimum elapsed end-to-end finishing time (160tu), while needs 9 processing units for running.

4.3 Comparison between MPGA and simple GA

For evaluating the impact of multi-population optimization on the allocation and scheduling problem, the results of single objective optimization has been achieved by leveraging single population GA (simple GA). On the other hand, the results of multi-objective optimization has been achieved by using MPGA. We compared MPGA and simple GA schemes in terms of exploration time and quality of results in the following sections.

a) Exploration Time and Speedup. Fig. 8 and Fig. 9 represent the convergence of processing units for single population GA and

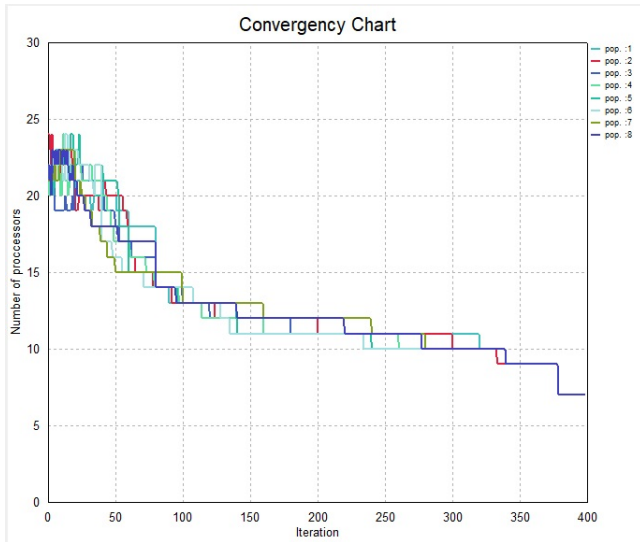


Figure 9: Convergence diagram of the # processing units in the multi-objective optimization by using MPGA approach.

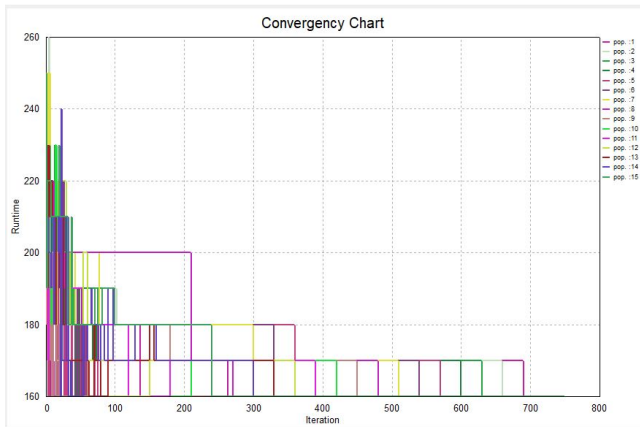


Figure 10: Convergence diagram of the end-to-end runtime in the multi-objective optimization by using MPGA approach.

MPGA, respectively. MPGA achieve the best result after 400 iterations, while single population GA needs 750 iterations for finding the best solution. Obviously converging to the best result needs in less number of iterations by using MPGA which is the best proof to show the benefits of applying the MPGA, especially when the design space is large.

b) Quality of Results. According to the results of Table 2, MPGA found a solution with 7 required processing units and 210tu for the end-to-end finishing time, while single population GA found a solution with the same required processing units but takes 250tu for the end-to-end finishing time. MPGA provides more quality of results compared to single population GA even when single population GA tries to optimize only one objective.

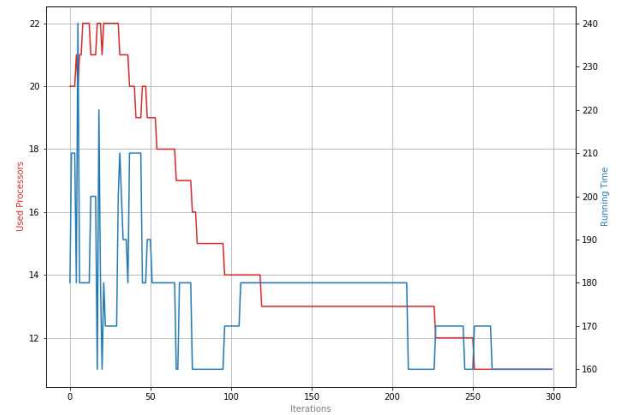


Figure 11: Improvement proceeding of exploration objectives including the number of processing units and end-to-end use case run-time.

4.4 Comparison Between MPGA and Morady et al. [11]

We compare the proposed MPGA solution with a similar evolutionary approach [11]. However, we customize the fitness functions for the studied use case. Table 2 represents the evaluation results after applying [11] on the industrial use case. As seen in Table 2, our method in single objective scenario found a schedule with 7 required processing unit, while [11] proposed a solution with 9 required processing units. In addition, in multi-objective scenario, [11] needs 180tu to finish all the jobs, while compared to Solution③, our proposed method needs 160tu. Therefore, we can conclude our customized MPGA overcomes a similar EC method represented in [11].

4.5 Allocation and Scheduling Results

We have considered here the use case described in section 2, and illustrated entirely by Fig. 1. After applying MPGA to the use case tasks, the near-optimal scheduling result is shown in Fig. 12, a valid scheduling for all jobs and their related tasks with minimum number of processing units (Solution①).

5 RELATED WORK

Here, we first explore more traditional list scheduling heuristics that have considered communication costs.

The basic idea is to make an ordered list of nodes by assigning them orders, and then to repeatedly execute the following two steps until a valid schedule is obtained: ① Select from the list the node with the highest order for scheduling. ② Select a processor to accommodate this node. In realistic cases, scheduling needs to exploit parallelism by identifying the task graph structure and take into consideration task granularity, arbitrary computation, and communication costs.

In [10], the modified critical path algorithm (MCP) is proposed, based on the latest possible start time of a node. A node's latest possible start time is determined via the as-late-as-possible (ALAP)

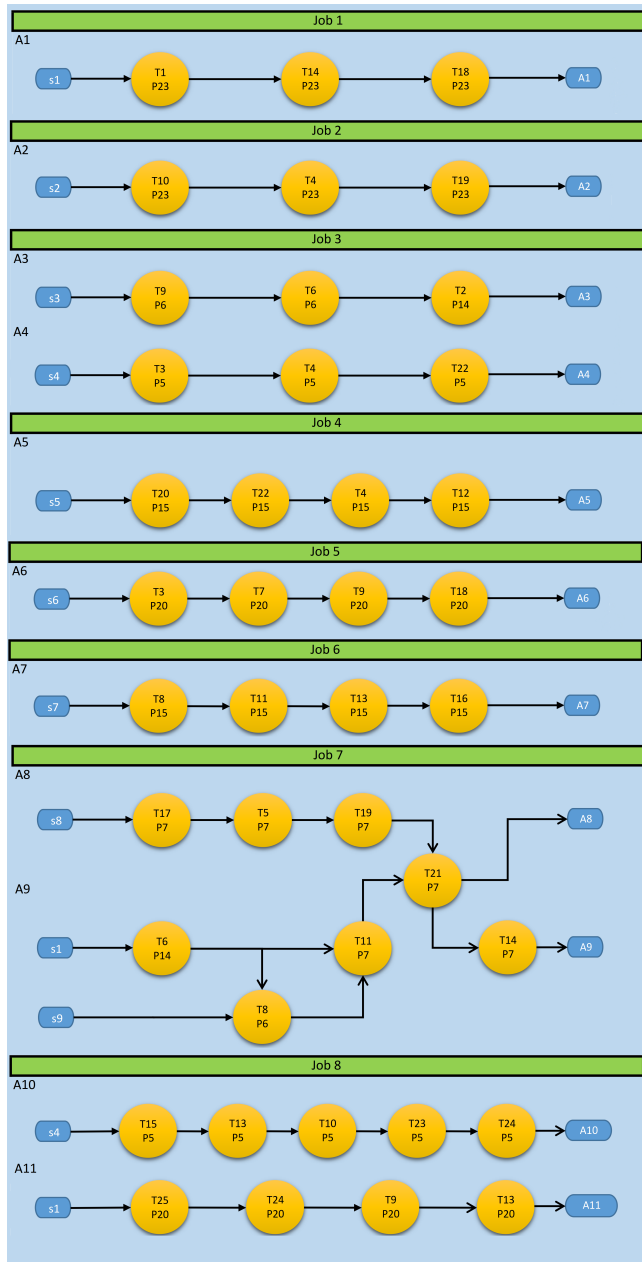


Figure 12: The best solution for multi-objective optimization with minimum number of processing units (solution ①).

binding by traversing the task graph upward from the exit nodes to the entry nodes while pulling the node’s start times downwards as much as possible. The latest possible start time of the node itself is followed by a decreasing order of the latest possible start times of its successor nodes. Furthermore, in [10], the dominant sequence clustering algorithm (DSC) is presented. It is based on the dominant sequence, which is essentially the critical path of the partially scheduled task. CP (the critical path of task graph) node is

a ready node. If so, DSC schedules it to a processor allowing the minimum start time. Such a minimum start time may be achieved by rescheduling some of the node’s predecessors to the same processor. If the highest CP node is not a ready node, DSC does not select it for scheduling. Instead, it chooses the highest node which lies on a path reaching the CP for scheduling. Moreover, also in [10], the mobility directed algorithm (MD) is presented. MD selects a node at each step based on relative mobility which is defined as the difference between a node’s earliest start time and latest start time. Similar to the ALAP binding, the earliest possible start time is assigned to each node via the as-soon-as-possible (ASAP) binding. This is performed by traversing the task graph downward from the entry nodes to the exit nodes while pulling the nodes upward as much as possible. Moreover, relative mobility is obtained by dividing the mobility with the node’s computation cost. Basically, a node with zero mobility is a node on the CP. At each step, MD schedules the node with the smallest mobility to the first processor having a large enough time to accommodate the node without considering the minimization of the node’s start time. After a node has been scheduled, the relative mobility values of the remaining nodes are updated.

In [11], a MPGA is presented which outperforms deterministic and non-deterministic methods described in [12, 13]. In [14], a new encoding mechanism with a multi-functional chromosome is presented, using a priority representation that is called priority-based multi-chromosome (PMC). PMC can efficiently represent a task schedule and assign tasks to processors. It is another meta-heuristic method that uses a GA to achieve near-optimal scheduling of tasks.

Research on static mapping methods includes the work of Lei et al., who proposed a genetic mapping algorithm to optimize application execution time [15]. In their work, graphs represent applications and the target architecture is a NoC. Wu, et al. also investigated genetic mapping algorithms [16]. By combining dynamic voltage scaling techniques with mapping, they achieved 51% savings in energy consumption. Murali et al. explored mappings for more than one application in NoC design, using the tabu search (TS) algorithm [17]. Manolache, et al. investigated task mapping in NoCs, trying to guarantee packet latency [18]. For this purpose, both the task-mapping algorithm (TS) and the routing algorithm are defined at design time. Hu et al. presented a branch-and-bound algorithm to map a set of IP cores (IPs) onto a NoC with bandwidth reservation [19]. Their results show energy savings of 51.7% in the communication architecture. In [20] presented a task scheduling scheme on heterogeneous computing systems using a multiple priority queues genetic algorithm (MPQGA). Their experimental results for large-sized problems for a large set of randomly generated graphs as well as graphs of real-world problems with various characteristics showed that the proposed MPQGA algorithm outperformed two non-evolutionary heuristics and a random search method.

6 CONCLUSIONS AND FUTURE WORK

Leveraging a distributed environment for task scheduling can enhance reliably and provide a specification compliant processing scheme. The inherent difficulties in distributing application jobs

and scheduling them among processing units may lead applications to expose low performance, or the system may require extra (unnecessary) resource costs. Here, a parallel Multi-Population Genetic Algorithm is developed to overcome complexity barriers, towards optimizing both operation time and resource numbers, while preserving application requirements. For the evaluations, a synthetic use case has been studied, grouping many aspects of actual industrial systems. The final results offer a better resource efficiency (requiring less number of processing unit) while guaranteeing real-time execution. In addition, MPGA provides better efficiency compared to other similar evolutionary approaches. We expect more complex problems to appear when we need to deal with duplication of tasks and synchronization activities, related to reliability and fault tolerance aspects, in future research actions.

ACKNOWLEDGMENTS

This work is supported by the Knowledge Foundation (KKS) through the DPAC project.

REFERENCES

- [1] Freund, R. F. Optimal selection theory for superconcurrency. Proc. Supercomputing '89. IEEE Computer Society, Reno, NV, 1989, pp. 699-703.
- [2] Adam TL, Chandy KM, Dickson JR. A comparison of list schedules for parallel processing systems. Communications of the ACM 1974;17(12):685-690.
- [3] Wu MY, Gajski DD. Hypertool: a programming aid for message-passing systems. IEEE Transactions on Parallel and Distributed Systems 1990;1(3):330-343.
- [4] Hou ESH, Ansari N, Hong R. A genetic algorithm for multiprocessor scheduling. IEEE Transactions on Parallel and Distributed Systems 1994;5(2):113-120.
- [5] Hwang RK, Gen M. Multiprocessor scheduling using genetic algorithm with priority-based coding. Proceedings of IEEE conference on electronics, information and systems; 2004.
- [6] Wu AS, Yu H, Jin S, Lin K-C, Schiavone G. An incremental genetic algorithm approach to multiprocessor scheduling. IEEE Transactions on Parallel and Distributed Systems 2004;15(9):824-834.
- [7] Majd, Amin, et al. "NOMeS: Near-optimal meta-heuristic scheduling for MPSoCs." Computer Architecture and Digital Systems (CADS), 2017 19th International Symposium on. IEEE, 2017.
- [8] Majd, Amin, Golnaz Sahebi, Masoud Daneshmand, Juha Plosila, Shahriar Lotfi, and Hannu Tenhunen. "Parallel imperialist competitive algorithms." Concurrency and Computation: Practice and Experience 30, no. 7 (2018): e4393.
- [9] Y. Chen, Y. Zhong, "Automatic Path-oriented Test Data Generation Using a Multi-population Genetic Algorithm," Proc. Fourth International Conference on Natural Computation, pp. 566-570, Oct 2008.
- [10] Adam TL, Chandy KM, Dickson JR. A comparison of list schedules for parallel processing systems. Communications of the ACM 1974;17(12):685-690.
- [11] R. Morady and D. Dal, A Multi-Population Based Parallel Genetic Algorithm for Multiprocessor Task Scheduling with Communication Costs, 2016 IEEE Symposium on Computers and Communication (ISCC).
- [12] Wu MY, Gajski DD. Hypertool: a programming aid for message-passing systems. IEEE Transactions on Parallel and Distributed Systems 1990;1(3):330-343.
- [13] Hou ESH, Ansari N, Hong R. A genetic algorithm for multiprocessor scheduling. IEEE Transactions on Parallel and Distributed Systems 1994;5(2):113-120.
- [14] R. Hwang, M. Gen and H. Katayama, "A comparison of multiprocessor task scheduling algorithms with communication costs," Computers & Operations Research, Vol. 35, pp. 976-993, ELSEVIER, 2008.
- [15] T. Lei and S. Kumar, "Two-Step Genetic Algorithm for Mapping Task Graphs to a Network on Chip Architecture," Proc. Euromicro Symp. Digital System Design (DSD 03), IEEE Press, 2003, pp. 180-187.
- [16] D. Wu, B. Al-Hashimi, and P. Eles, "Scheduling and Mapping of Conditional Task Graphs for the Synthesis of Low Power Embedded Systems," Proc. Design, Automation and Test in Europe (DATE 03), IEEE CS Press, 2003, pp. 90-95.
- [17] S. Murali and G. De Micheli, "Bandwidth-Constrained Mapping of Cores onto NoC Architectures," Proc. Design, Automation and Test in Europe (DATE 04), IEEE CS Press, 2004, pp. 896-901.
- [18] S. Manolache, P. Eles, and Z. Peng, "Fault and Energy-Aware Communication Mapping with Guaranteed Latency for Applications Implemented on NoC," Proc. 42nd Annual Design Automation Conf. (DAC 05), ACM Press, 2005, pp. 266-269.
- [19] J. Hu and R. Marculescu, "Energy- and Performance-Aware Mapping for Regular NoC Architectures," IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol. 24, no. 4, 2005, pp. 551-562.
- [20] Y. Xu, K. Li, J. Hu and K. li, "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," Information Sciences, Vol.270, pp. 255-287, Elsevier, 2014.

Paper V

Placement of Smart Mobile Access Points in Wireless Sensor Networks and Cyber-Physical Systems using Fog Computing

A. Majd, G. Sahebi, M. Daneshtalab, J. Plosila and H. Tenhunen

Placement of Smart Mobile Access Points in Wireless Sensor Networks and Cyber-Physical Systems using Fog Computing

Amin Majd
Department of Information
Technology
University of Turku
Turku, Finland
Amin.Majd@utu.fi

Juha Plosila
Department of Information Technology
University of Turku
Turku, Finland
juplos@utu.fi

Golnaz Sahebi
Department of Information
Technology
University of Turku
Turku, Finland
Golnaz.Sahebi@utu.fi

Masoud Daneshtalab
Royal Institute of Technology
(KTH)
Stockholm, Sweden,
masdan@kth.se

Hannu Tenhunen
Royal Institute of Technology
Stockholm, Sweden
University of Turku, Finland
Hannu@kth.se

Abstract — Increasingly sophisticated, complex, and energy-efficient cyber-physical systems and wireless sensor networks are emerging, facilitated by recent advances in computing and sensor technologies. Integration of cyber-physical systems and wireless sensor networks with other contemporary technologies, such as unmanned aerial vehicles and fog or edge computing, enable creation of completely new smart solutions. We present the concept of a Smart Mobile Access Point (SMAP), which is a key building block for a smart network, and propose an efficient placement approach for such SMAPs. SMAPs predict the behavior of the network, based on information collected from the network, and select the best approach to support the network at any given time. When needed, they autonomously change their positions to obtain a better configuration from the network performance perspective. Therefore, placement of SMAPs is an important issue in such a system. Initial placement of SMAPs is an NP problem, and evolutionary algorithms provide an efficient means to solve it. Specifically, we present a parallel implementation of the imperialistic competitive algorithm and an efficient evaluation or fitness function to solve the initial placement of SMAPs in the fog computing context.

Keywords- smart mobile access point; fog computing; wireless sensor networks; cyber-physical systems; multi-objective optimization; evolutionary computing; parallel approaches; ICA; parallel programming; multi-population; placement.

I. INTRODUCTION

Wireless sensor networks (WSN) and cyber-physical systems (CPS) are two current important fields of technology that are tightly intertwined [1], [2]. There are different kinds of real WSN applications implemented for cyber-physical systems. The combination of wireless sensor networks and the other new technologies, such as unmanned aerial vehicles (UAV) and mobile robots, has created a new revolution in this area. WSNs with mobile nodes can obtain better performance by using mobile access points embedded in UAVs or mobile robots [25], [26]. In this paper, we aim at improving the resource

utilization and power-performance (energy efficiency [27], [3]) by selecting the positions for SMAPs of such a CPS using Smart Mobile Access Points (SMAPs).

SMAPs can make a cluster for computing together and make decisions for improving quality of the network. SMAPs can change their positions based on their decisions. Therefore, placement of SMAPs is a critical problem in this concept.

Placement is a multi-objective optimization problem. Different kinds of approaches, such as static or dynamic methods, can be used to solve this problem [6]. For initial placement, which is a static placement problem, we can utilize an Evolutionary Computing (EC) method [7], [10] such as Particle Swarm Optimization (PSO) [9] and the Imperialist Competitive Algorithm (ICA) [8]. Of these, we introduce a multi-population version of ICA (PICA) [11] with an efficient fitness function for solving the initial placement problem in SMAP. This is based on the fog computing context in order to improve the speed and accuracy of our approach.

In Section 2, SMAPs such as new concept is presented. In section 3, a review of related works is presented. In Section 4, a parallel multi-population implementation of ICA based on fog computing to solve the placement of SMAPs is introduced. In Section 5, PICA is compared with ICA, random method, and mathematical method.

II. SMART MOBILE ACCESS POINTS

SMAPs are mobile access points that enable creation of a smart sensor network. The task of SMAPs is to predict the behaviour of the network and select the best approach to support the network at any given time. SMAPs receive and send signals (e.g. battery levels, the number and IDs of sensors that have been covered, and help requests) from and to the other network nodes, collect the received information, predict the next operations based on this data, and run these operations. An SMAP sometimes utilizes previous knowledge and operations, or creates new operations by learning or

evolving through machine learning techniques. The most important operations of SMAPs are the following: 1) Finding new (optimal) positions for access points to obtain a better coverage of sensors; 2) Making decisions about moving the access points to new areas [17]; 3) Participating in distributed communication and computation tasks (fog computing [18]) other than the processing carried out as an inherent part of the decision making in 1) and 2).

Decision-making on moving an SMAP to a new position can lead to different approaches, such as creation of a new network configuration, supporting e.g. fault tolerance by replacing a faulty access point with a functioning one, or covering a missed area of sensors that is not currently supported or covered by any other access point. This scenario is illustrated in Figure 1. The network has missed the red static access point (in Figure 1 (a)), then SMAPs move to new positions (P_1 and P_2) to support the connectivity of the network (in Figure 1 (b)). SMAPs make a cluster connection for essential computing together by fog connection that is illustrated in Figure 1.

SMAPs improve quality and flexibility of WSNs. Their advantages can be summarized as follows:

- SMAPs enhance the system by improving coverage in large scale networks.
- SMAPs enable dynamic reconfiguration of WSNs.
- SMAPs facilitate solving hotspot problems in the network.
- SMAPs can together make a grid network to run real-time computing tasks in parallel, facilitating near-sensor processing and fog computing.

- SMAPs can be used to replace faulty access points.

It is clear that placement of SMAPs has a significant effect on the performance of the network. This motivates us to concentrate on the placement problem in this paper. For simplicity, we will focus especially on the initial placement to demonstrate efficiency of our approach.

SMAPs should have an initial placement and initial clustering [18] (a cluster is a set of sensors that has been covered by an access point). Also, they should be able to dynamically move to new positions, because the behaviour of the network can change at run-time. This reconfiguration process is a reaction to the status of the network, and SMAPs can update their positions for each new configuration.

Such a placement and clustering problem can be solved by utilizing either cloud computing or fog computing [19], [20]. Our approach is based on the latter.

Each cluster center, access point, or area always has a distinct weight. This weight is equal to the probability of requesting for reconfiguration at the access point in question. Weights can be computed based on different effective parameters, such as the network traffic at these access points, the lifetime of their batteries, or properties of the covered area. This phase can be handled using a reinforcement learning approach [28], where certain features defined by users (e.g. battery life time) along with some essential characteristics of the network (e.g. connectivity) are utilized. The model for SMAPs is illustrated in Figure 2.

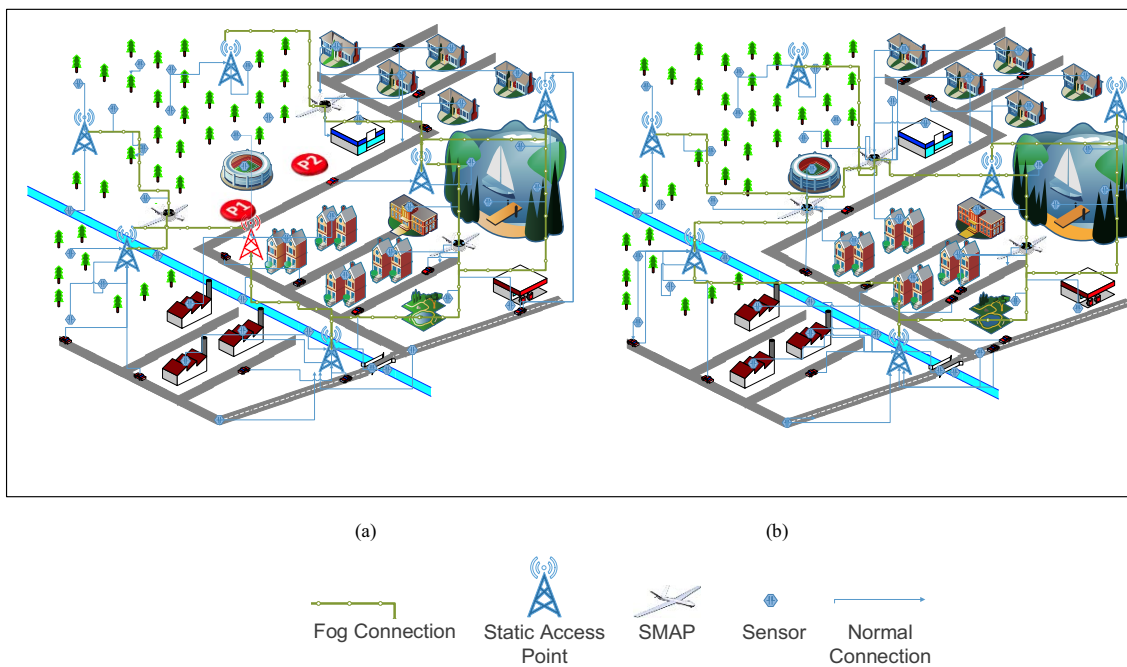


Fig. 1. SMAP model that using Fog network.

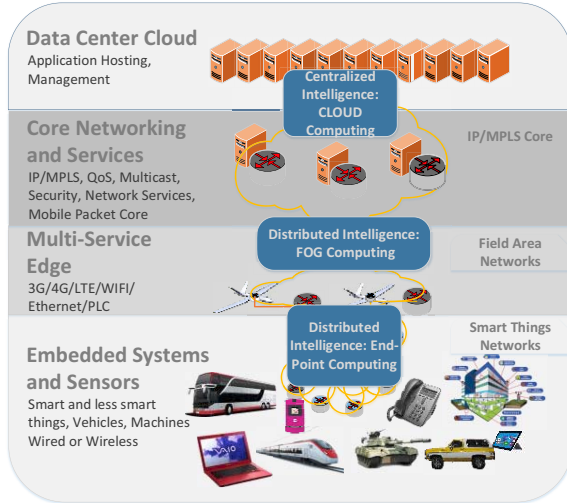


Fig. 2. SMAP model that using Fog network.

III. BACKGROUND

In this section, brief reviews on sensor and access point placement methods and on Evolutionary Algorithms (EA), especially Imperialistic Competitive Algorithms (ICA), are presented.

A. Different Methods for Placement

In a dynamic WSN, some nodes can be repositioned to new places. For example, sensors could be moved to new locations to get more useful information, or they could be replaced with different types of sensors to sense other physical phenomena or to get more accurate readings

Real-time replacement of nodes during network operation is in general very complex. Indeed, there is a considerable difference between static initial placement and dynamic run-time placement. In dynamic placement, there are more complex parameters such as the environment base, mobile targets, and lifetime of sensors. Furthermore, replacement of a node requires very careful handling since it can potentially cause a disruption in data connectivity and delivery. In some cases, sensors can also move to new positions to obtain better readings or to sense other information. In this kind of scenarios, dynamic placement is very challenging. Two major factors contribute to this complexity. The first one is the ability of nodes to move and reposition to an infinite number of locations. The second factor is the obligation of the network to maintain full connectivity and to cover all sensors, i.e., to reliably collect data from all static and dynamic sensors in the network.

There are two main types of placement in WSNs: placement of sensors and placement of access points. In this paper, we only focus on placement of access points.

Placement of access points is a multi-objective optimization problem in which most of the set goals should be satisfied. There are different goals, but two of

them are more critical. The first key objective is to guarantee continuous network connectivity. This means in practice that each sensor node should have a connection to at least one access point at any given time. The second key objective is about providing reliable communication in the case of access point failures. In other words, all other nodes in the network should maintain their communication if an access point runs out of power. Using extra (redundant) access points is the best solution to obtain this goal. With this approach, the number of possible access points for sensors is increased.

In the real world, there are heterogeneous sensor networks with three different kinds of nodes: sensors, access points, and gateways. Sensors can only communicate with access points. Access points are utilized for routing and can communicate with all other types of nodes. Also, access points have more memory and computing capacity than sensors. Access point nodes can only communicate with access points. The following three main conditions must be respected to obtain efficient WSNs;

- Full connectivity: All access points and sensors must have at least one connection (direct or indirect) to a gateway.
- Configurable redundancy robustness [20]: It is necessary to have more than one different route between each sensor and gateways.
- Placement constraints: In practice, an access point cannot be placed in every possible location, i.e., there can be some obstacle to this placement.

Finding an optimal placement to improve connectivity, reliability, and energy consumption is an NP-hard problem [23]. There are different heuristic methods to solve this problem, which are presented in [24], [19].

Solving the problem of missing access points is essential for real large-scale WSNs. Also, it is evident that using ordinary methods to implement more redundancy (i.e. adding fixed/stationary access points) is very expensive. SMAPs can be used to solve this problem at a lower cost and with more flexibility.

B. Evolutionary Algorithms

The multi-objective placement problem is an NP-hard problem. Evolutionary Algorithms (EAs) are the best choice to solve it. There are different kinds of EAs that have been utilized for discrete and continuous problems. For example, Genetic Algorithms (GAs) [15] are very popular to solve discrete problems. Also, Particle Swarm Optimization (PSO) [16], [17] and Bee Colony Optimization (BCO) [14] have been used for continuous problems. The Imperialistic Competitive Algorithm (ICA), in turn, is an efficient EA that has been utilized for continuous problems [8], [12], [13].

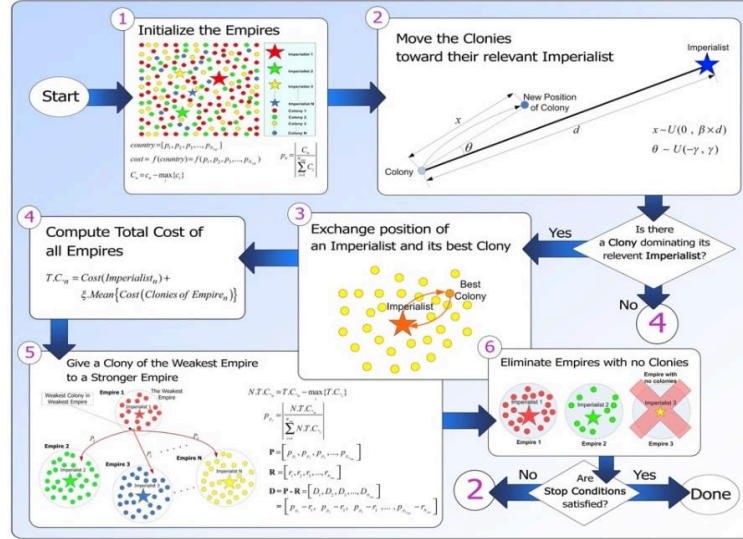


Fig. 3. Imperialist Competitive Algorithm [8].

C. Imperialistic Competitive Algorithm

ICA is an optimization method, an efficient evolutionary algorithm, based on imperialistic competition. In this algorithm, all countries are divided into two categories: colonies and imperialist states. The main part of this algorithm is imperialistic competition, which is expected to cause the colonies to converge to the global minimum. ICA is a suitable method for optimization [12], [13], but there are similar problems in ICA as in other evolutionary algorithms. For example, in the case of a large search space, a considerable initial population is required to obtain a more accurate and reliable result. However, we cannot realize this requirement with a single processor. The computing capacity would not be sufficient. When we face a complex problem that needs complex computations, the run time will increase, and therefore we need to utilize a new method to improve speed and efficiency. A parallel computing method for ICA, called PICA, which we have previously proposed [11], is employed here to improve performance. The method is a multi-population implementation of ICA.

IV. PROPOSED APPROACH

In this section, a parallel implementation of ICA (multi-population PICA) is applied to develop an efficient method for solving the initial placement problem in SMAP based systems. For this, we have divided our approach into two layers: an architectural layer and an algorithmic layer.

A. Architectural Layer

In the proposed distributed system, a set of SMAPs form a computing network. Each SMAP consists of two node types; application nodes (AN) and a manager node (MN). The nodes are organized in a federated manner as shown in Figure 5, that is, each autonomous subnetwork of ANs is managed by a single manager node, and a

group of subnetworks form a larger network. The larger network is controlled by a single main manager node elected from the subnetwork managers with the support of a cloud service. Since all manager nodes are connected to the cloud, the communication between manager nodes can take through the cloud. Optionally, near managers can communicate directly without the need to pass through the cloud.

Following the node types, there are different levels of uptime. An application node is in a sleep state most of the time and initiates communication whenever it wakes up. Application nodes run algorithmic layer tasks. They receive a manager advertisement, which contains the address of the manager. Manager nodes manage communication between their application nodes and other SMAPs. For ease of discussion, Figure 5 presents five subnetworks each having a single manager (blue nodes in Figure 5) and communicating using an appropriate protocol (such as 6LoWPAN or Bluetooth Low Energy). The local manager is known as the home manager for the application nodes in its subnetwork.

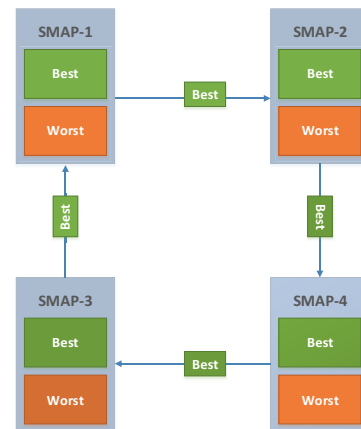


Fig. 4. Multi-population PICA based on Fog computing by SMAPs

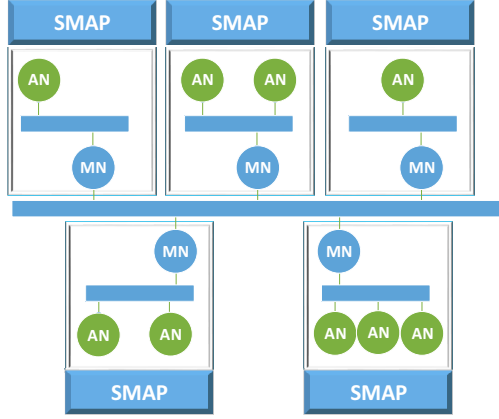


Fig. 5. Federated architecture of SMAPs

B. Algorithmic Layer

To have the best placement of SMAPs, in our work, all access points are divided into two main groups that are SMAPs and normal access points. Normal access points are static access point and only transfer data and maintain connectivity in the network. SMAPs can work like normal access points, but they also can perform other tasks, like supporting other access points when needed. In this paper, however, to simplify the problem, normal access points have been selected as target points for SMAPs, as we are considering the static initial placement only. Also, each normal access point has an independent weight. The weight of each access point indicates the probability of needing support from SMAPs (i.e. the need for network reconfiguration). This probability is a value that can be derived from different parameters such as communication traffic and battery lifetime. In our approach, the rate of communication traffic is selected for this purpose.

1. Modeling of the problem

We assign a set, denoted by A , for normal access points that have specific coordinates (X_i, Y_i) . This set consists of the coordinates of all (normal) access points in the system (static access point). Also, each access point has a certain weight, denoted by W_i , and the set of all weights is denoted by W . W_i can be a value that represents connectivity, reliability, energy or any other important parameter in the network. The objective of our work is to discover the best locations for SMAPs in their search space so that SMAPs are placed closer to points that have higher values of W_i . These distances, which are proportional, should be dependent on all W_i . The set which consists of the points or coordinates where SMAPs are to be placed is denoted by P .

In order to solve the placement problem, we consider a set S of n_s points $S_i \in \mathbb{R}^2$. Each S_i can represent either an SMAP, a point whose position is not known, or a normal access point whose position is known. Thus, S

can be partitioned into the normal access point set A , and the SMAP set P , with cardinalities $|A| = n_A$ and $|P| = n_P$.

$$A = \{(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)\}$$

$$W = \{W_1, W_2, \dots, W_n\}$$

$$P = \{(X_1, Y_1), (X_2, Y_2), \dots, (X_m, Y_m)\}$$

Given a pair of points $(S_i, S_j) \in S \times S$, or (i, j) for short, their Euclidean distance $\|S_i - S_j\|_2$ is denoted by d_{ij} .

The whole distance set can be represented by a Euclidean distance matrix (EDM in short) $EDM \in \mathbb{R}^{n_A \times n_P}$. For m points, the set of all possible Euclidean distance matrices, denoted as EDM^m , is a rich algebraic structure: the EDM^m is a convex cone contained in the subspace of all symmetric hollow matrices, that is closely related with the semi definite matrix cone.

To solve this problem, i.e., to find an optimal solution for EDM , we use the multi-population PICA which is described next-

2. Multi-Population PICA

In [11], we have utilized a multi-population model to implement the PICA by applying a selective local search strategy. In this implementation, several processors are connected in a ring topology on the message passing architecture, and they are illustrated in Figure 4. This approach can run on both share-memory and message passing architectures; although, shared-memory methods actually do not have any specific topology.

As the first step, independent countries are initiated in each processor, and then ICA is independently run in each processor. After some decades (the time is different on different runs) the best country migrates from the processor P_i to P_{i+1} in the ring and replaces the worse country in P_{i+1} . The migration takes place synchronously (simultaneously) in all processors in the ring. There can be different migration strategies leading to different results. In the case of a sparse connection topology, such as a ring which we have utilized, it is better to use low migration rates and as short migration distances as possible. In a fully connected topology, in turn, the best results are obtained when the migration rates are high. The pseudo-code for PICA is presented in Figure 6. A system-wide migration operation is executed at each designated migration time point. During the migration, the processors do not execute any other tasks. The algorithm terminates either after a fixed number of iterations or when some other termination condition is reached.

PICA is a highly efficient parallel method which improves the speed, stability, and accuracy of results [11]. This motivates us to apply it to the SMAP placement problem. In order to appropriately adapt the method, each SMAP is considered a processor, and a ring topology is assumed. A ring is a simple structure, facilitating efficient short-distance migrations.

Processor P_i :

1. Create independent initial countries.
2. Run ICA algorithm independently.
3. If now is the time of migration do
 - 3.1 Wait until all processors arrive to this point.
 - 3.2 Send the best country to processor $(P_{i+1}) \bmod$ (number of processors).
 - 3.3 Receive a country from $(P_{i-1}) \bmod$ (number of processors) and replace the worst country with the received one.
4. If the termination condition is reached then terminate the algorithm
Else go to 2.
5. Show the best country.
6. End.

Fig. 6. Pseudo-code of Multi-Population ICA.

2.1. Creation of countries

First, a matrix, in which each cell is a data structure of the form (X_i, Y_i) , is created. Each row of the matrix is a country, which is a set of points, and also, it can be the best solution. These points are a possible placement of the SMAPs in the proposed search space. Values of X_i and Y_i are real values ($X_i, Y_i \in R$) and are randomly generated. Each column corresponds to the location of each placement of a SMAP. Hence, the number of rows is equal to the size of the initial population (N_{pop}) and the number of columns is equal to the number of SMAPs.

2.2. Evaluation function

In all problems that are solved by EAs, the essential task is to find a suitable fitness function to evaluate the whole population. In our approach, the evaluation function receives a country as its input and returns a real value as its output. The output is computed based on the distance between the SMAPs and all normal access points and on the weight of each normal access point. A country with the lowest value of the evaluation or fitness function is the best country (solution). In other words, this problem is a minimization optimization problem.

One common fitness function for solving the placement of a single SMAP, which has been utilized in most related work, is illustrated in Equation 1.

$$\text{Fitness Function} = \sum_{i=1}^m \sum_{j=1}^n \sqrt{(X_i - X_j)^2 + (Y_i - Y_j)^2} \quad (1)$$

Indeed, this function is suitable for selecting the best place for a single SMAP, but when the number of SMAPs is increased, the placement problem becomes more complex, and the function presented in Equation 1 does not provide correct results anymore and cannot therefore be applied. Here we propose a new and efficient fitness function DM, presented in Equation 2, which provides accurate results for any number of SMAPs.

$$\text{DM} = \sum_{i=1}^{NA} \left((D_{NN_i}) + \left(\frac{D^2 - NN_i}{4} \right) \right) / \text{Distance} \quad (2)$$

$$D = \lfloor \sqrt{n} \rfloor$$

Here n equals the number of normal access points and NA equals the number of SMAPs. The D_{NN_i} equals the Euclidean distance between SMAP number i and D

nearest normal access points. The *Distance* variable equals the sum of distances between all SMAPs that is presented in Equation 3.

$$\text{Distance} = \sum_{i=1}^m \sum_{j=1, j \neq i}^m \sqrt{(X_i - X_j)^2 + (Y_i - Y_j)^2} \quad (3)$$

2.3. Selecting imperialists and making colonies

In the next step, the most powerful countries (N_{imp}) are selected (the number of N_{imp} is equal to five percent of the number of all contraries). In addition, each of them becomes an imperialist and every imperialist selects some other countries based on its power. After this, each country in colonies starts moving towards its relevant imperialist; this operation corresponds to the exploitation operation in any other EAs and is illustrated in Figure 2.

Then some countries are randomly selected and their positions are randomly changed. This operation explores the search space and is well-known in all EAs. The most significant advantage of this operation is that it prevents the algorithm from dropping in local optimums.

2.4. Exchanging the position of an imperialist and a colony

In every iteration, if the situation of a colony improves and it becomes better than its imperialist, their positions are exchanged. This operation is presented in Figure 2.

2.5. Imperialist Competition

In this step, imperialists start a competition to obtain new countries from other imperialists. In this phase, power of all imperialists and their colonies are computed to determine the probability of winning the computation. Finally, the weakest imperialist is detected, and its colonies are granted to the other imperialists. The receiver imperialist is selected by a tournament operation based on the probability of winning. The competition operation is shown in Figure 2. All the above steps are repeated until the next time of migration arrives.

2.6. Migration Operator

At a migration time point, processors stop their current tasks and discover their best imperialists and the worsts colonies. Then the processors synchronously send their best imperialists to the next processors and receive the best imperialists from the previous processors, and replace their worst colonies with the received ones.

Finally, the algorithm stops after some certain number of iterations, and the best imperialist is determined. This represents the best solution for the SAMP placement. This operation is presented in Figure 3.

V. EXPERIMENTAL RESULTS

The proposed work has been implemented on Intel® Core™ i5, CPU @ 2.9GHz, RAM 8 GB at Department of Information Technology, University of Turku, Finland. VC++ 2015 has been utilized for the

implementation and MPI instructions for parallelization. Furthermore, MPICH2.3.2 has been applied for parallel execution of the algorithms. The proposed PICA based approach has been tested on four processors in all tests for four benchmarks. Well-known 2-D benchmarks have been utilized. The obtained results have been compared with the serial ICA, mathematical, and random placement methods.

A. SINGLE SMAP

In the first three benchmarks, the best place is in the coordinate (0, 0). The aim in these cases is to find the best place for a single SMAP on their search spaces. In the fourth benchmark, the goal is to place more than one SMAP to different positions. All the methods have been run on these benchmarks, 40 times for each benchmark, and the results have been compared with each other.

The first benchmark contains eight normal access points (static access points) that have a circular arrangement. The radius of this circle is equal to 500 meters. The second benchmark has eight normal access points that are concurrent based on the center point. The coordinate of the center point is (0, 0). The third benchmark has 50 simultaneous access points that are randomly created. In all these benchmarks, the weights of the normal access points are equal. The results of PICA and serial ICA have been obtained in 100 iterations with 500 countries as an initial population.

Figure 7 shows the placements that have been discovered by PICA. It clearly indicates that the results are very close to the optimal point, and the best results are obtained 35 times in the series of 40 runs. The results of the random placement and serial ICA placement methods are illustrated in Figures 8 and 9, respectively. It can be clearly seen that the results of PICA are significantly better than the results of the random and serial ICA methods. Figures 10, 11 and 12 indicate the results of the PICA, random, and serial ICA placement methods, respectively, for the second benchmark. Finally, for the third benchmark, Figures 13, 14 and 15 present results of all three placement methods. The statistical results of the three methods and a mathematical approach for all the benchmarks are illustrated in Table 1. These results demonstrate that PICA is more accurate and efficient than the other methods. They also indicate that the proposed PICA method is successful on both simple and complex benchmarks. Figures 16, 17 and 18 present the stability diagrams of all the methods for the first three benchmarks. The migration operation in PICA is the primary factor for the accuracy of results in our approach. Also, this factor can be a strong motivation for using PICA.

In Table 1, there are three important columns described as follows: The best distance shows the minimum distance between the results of each method and the best position (center point) of each benchmark.

The worst distance presents the maximum distance between the results of each method and the best position of each benchmark. The correct placement count column shows that how many times each method finds the best position for each benchmark in the set of 40 consecutive test runs.

B. MULTIPLE SMAPS

Unlike in the three benchmarks considered above, in the fourth benchmark the algorithms should choose the best locations for several SMAPs instead of just for one. In this benchmark, there are 16 normal access points and 5 SMAPs. The SMAPs should have different positions. Such complex placement problems are relevant for real-world wireless sensor networks and cyber-physical systems. In a single SMAP placement discussed above, a mathematical method could find the best place, but when dealing with several SMAPs, mathematical methods are not applicable in practice, as they cannot determine the best positions for the SMAPs in a finite time.

In this paper, the PICA and serial ICA methods are run on all the four benchmarks for 40 times with 100 iterations in each round. The number of initial countries is 500. The results of these tests for the fourth benchmark are indicated in Figures 19 and 20. In this placement problem, the DM function (Equation 2) is utilized as the fitness function for both PICA and ICA to obtain the best results. The overall results show that the proposed fitness function is very efficient, enabling PICA and ICA to find the best positions also in the complex placement cases. The results also demonstrate that PICA performs much better than ICA.

Finally, the runtimes of all methods in all benchmarks are illustrated in Table 2. The results for PICA in this table is based on four cores. The results show that PICA in all benchmarks finds the best place in a short time, but mathematical methods cannot determine the best positions for the SMAPs in the fourth benchmark.

TABLE I. STATISTICAL RESULTS

	Benchmarks	The Best Distance	The Worst Distance	Correct Placement Count
PICA	3	0	2.8635	35
Random		13.2549	78.1004	0
Serial ICA		0	11.5873	7
Mathematical		0	0	40
PICA	2	0	61.4191	36
Serial ICA		0	127.3185	4
Random		52.9906	1.0572e+03	0
Mathematical		0	0	40
PICA	1	0	475.0343	21
Serial ICA		0	4.6043e+03	2
Random		846.77	1.0447e+04	0
Mathematical		0	0	40

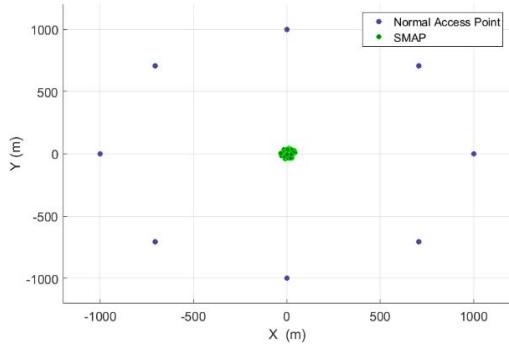


Fig. 7. The placement of PICA on benchmark 1.

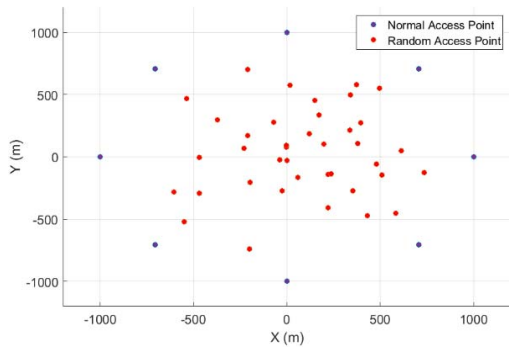


Fig. 8. The placement of random method on benchmark 1.

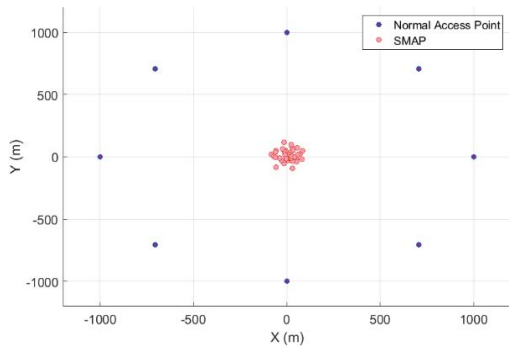


Fig. 9. The placement of ICA on benchmark 1.

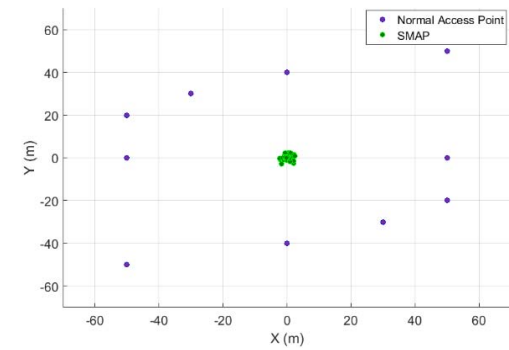


Fig. 10. The placement of PICA on benchmark 2.

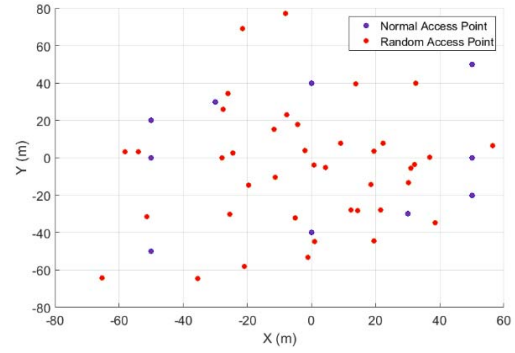


Fig. 11. The placement of Random method on benchmark 2.

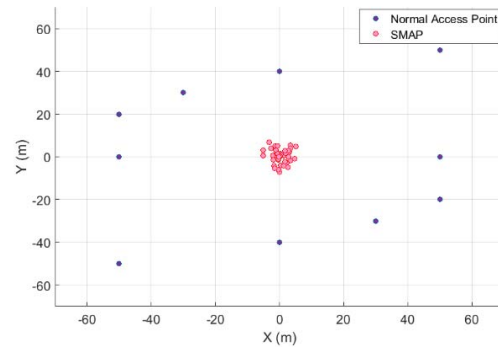


Fig. 12. The placement of ICA on benchmark 2.

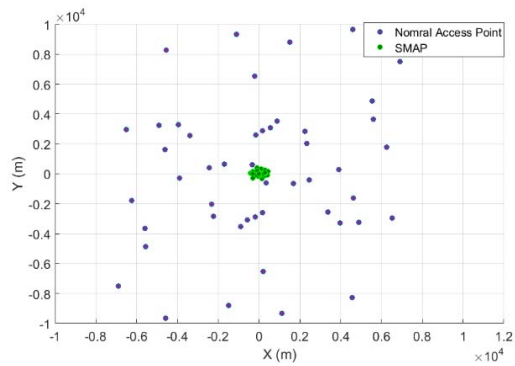


Fig. 13. The placement of PICA on benchmark 3.

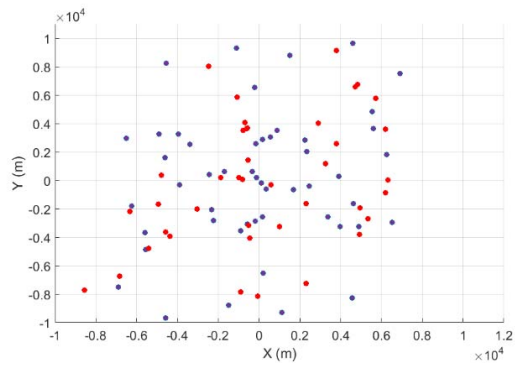


Fig. 14. The placement of Random method on benchmark 3.

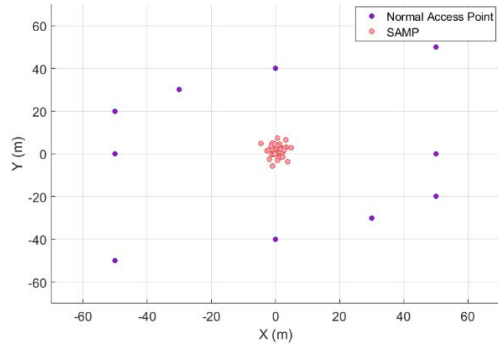


Fig. 15. The placement of ICA on benchmark 3.

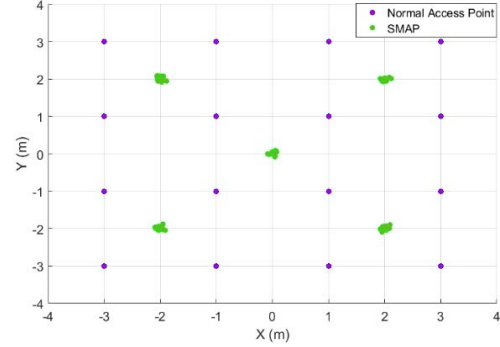


Fig. 19. The placement of PICA on benchmark 4.

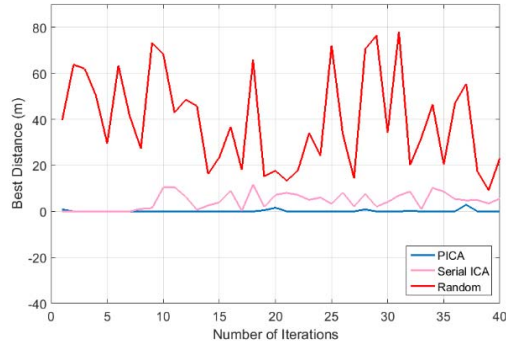


Fig. 16. The stability diagram of all methods on benchmark 1.

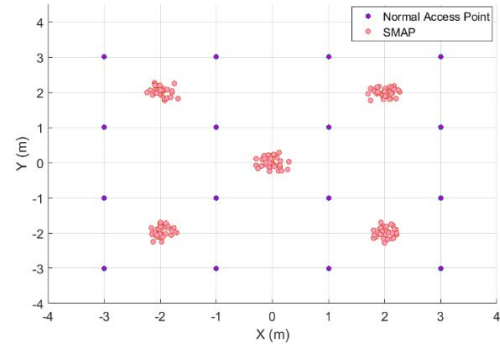


Fig. 20. The placement of ICA on benchmark 4.

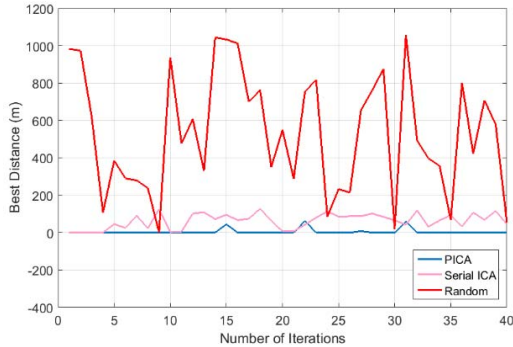


Fig. 17. The stability diagram of all methods on benchmark 2.

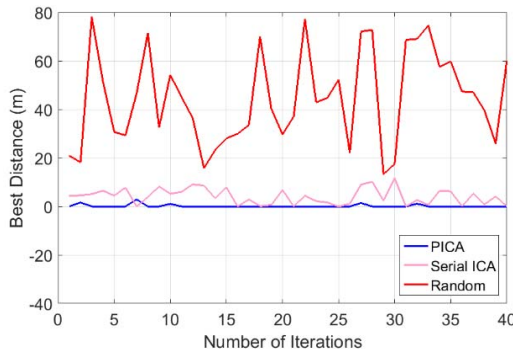


Fig. 18. The stability diagram of all methods on benchmark 3.

TABLE II. RUN TIME TABLE FOR ALL BENCHMARKS.

Benchmarks	Run Time (Sec)			
	1	2	3	4
Methods				
PICA	11.26	10.67	16.02	40.81
ICA	43.23	41.71	63.34	159.97
Random	3.01	3.00	3.61	3.40
Mathematical	4.21	4.20	4.65	----

VI. CONCLUSION

In this paper, the concept of a smart mobile access point (SMAP) was introduced, utilization of SMAPs in wireless sensor networks and cyber-physical systems was briefly discussed. The main focus of this study was on the placement of SMAPs, especially on finding the optimal initial placement. For this, we applied PICA as a heuristic method for solving the placement problem and proposed a new efficient fitness function tailored especially for the complex placement of SMAPs. It was demonstrated that this new fitness function improved the results of PICA in placement of several SMAPs. In our approach, SMAPs were utilized as a fog computing platform to run our optimization algorithm in a distributed manner. The proposed PICA method was tested on four different benchmarks and was compared with serial ICA, random placement and mathematical methods. The obtained results demonstrated that PICA is

an outstanding method and capable of solving efficiently complex placement problems that are highly relevant for real-world wireless sensor networks and cyber-physical systems.

As a future research topic, we will focus on dynamic runtime placement of SMAPs, which is a significantly more challenging problem than the initial placement problem discussed in this paper.

REFERENCES

- [1] Fang-Jing Wua , Yu-Fen Kao b , Yu-Chee Tseng a, “From wireless sensor networks towards cyber physical systems,” *Pervasive and Mobile Computing*, Elsevier, Vol. 7, pp. 397–413, 2011.
- [2] Priyanka Pandit1 , Swarupa Kamble2, “A Survey on Knowledge Extraction from WSN,” *International Journal of Science and Research (IJSR)*, Vol. 6, pp. 384-387, 2016.
- [3] Roselin,J and Latha, P, “Energy Efficient Coverage Using Artificial Bee Colony Optimization in Wireless Sensor Networks,” *International Journal of Scientific and Industrial Research*, Vol. 75, pp. 19-27, 2016.
- [4] M. Bhuiyan, J. Wu, G. Wang and J. Cao, “Sensing and Decision-Making in Cyber-Physical Systems: The Case of Structural Event Monitoring,” *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS*, 2016.
- [5] F. Bonomi, R. Milito, J. Zhu and S. Addepalli, “Fog Computing and Its Role in the Internet of Things,” *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13-16, 2011
- [6] Y. J. Cha, A. Raich, “Optimal placement of active control devices and sensors in frame structures using multi-objective genetic algorithms,” *Structural Control and Health Monitoring*, Vol. 20, pp. 16–44, 2013
- [7] A. Majd, Sh. Lotfi and G. Sahebi, “Review on Parallel Evolutionary Computing and Introduce Three General Framework to Parallelize All EC Algorithms,” *The 5th Conference on Information and Knowledge Technology*, IEEE, pp. 61-66, 2013.
- [8] E. A. Gargari, and C. Lucas, “Imperialist Competitive Algorithm: An Algorithm for Optimization Inspired by Imperialistic Competition,” *Congress on Evolutionary Computation*, IEEE, 2007.
- [9] H. Narasimhan, “Parallel Artificial Bee Colony (PABC) Algorithm,” *World Congress on Nature & Biologically Inspired Computing (NaBIC)*, IEEE, 2009.
- [10] A.Majd and G.Sahebi, “A Survey on Parallel Evolutionary Computing and Introduce Four General Frameworks to Parallelize all EC Algorithms and Create New Operation for Migration,” *Journal of Information and Computing Science*, vol. 9, pp.97-105,2014.
- [11] A. Majd, Sh. Lotfi, G. Sahebi, M. Daneshtalab and J. Plosila, “PICA: Multi-Population Implementation of Parallel Imperialist Competitive Algorithms,” *24th Euromicro International Conferences on Parallel, Distributed and Network-Based Processing, PDP 2016*.
- [12] M. Abdechiri, K. Faez and H. Bahrami, “Adaptive Imperialist Competitive Algorithm (AICA),” *9th IEEE International Conference on Cognitive Informatics (ICCI)*, 2010.
- [13] H. Bahrami, M. Abdechiri and M. Meybodi, “Imperialist Competitive Algorithm with Adaptive Colonies Movement,” *I.J. Intelligent Systems and Applications*, vol. 2, pp. 49-57, 2012.
- [14] H. Narasimhan, “Parallel Artificial Bee Colony (PABC) Algorithm,” *World Congress on Nature & Biologically Inspired Computing (NaBIC)*, IEEE, 2009.
- [15] E. Alba, F. Luna, A. J. Nebro and J. M. Troya, “Parallel Heterogeneous Genetic Algorithms for Continuous Optimization,” *Parallel Computing*, vol. 30, pp. 699–719, ELSEVIER, 2004.
- [16] Y. Zhou and Y. Tan, “Particle Swarm Optimization with Triggered Mutation and its Implementation Based on GPU,” *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, ACM, pp. 1-8, 2011
- [17] A. Basturk, R. Akay and A. Kalinli, “Comparison of fine-grained and coarse-grained parallel models in particle swarm optimization algorithm” *2nd World Conference on Information Technology (WCIT)*, 2011.
- [18] J. Zhu, D. S. Chan, M. S. Prabhu, P. Natarajan, H. Hu and F. Bonomi, “Improving Web Sites Performance Using Edge Servers in Fog Computing Architecture,” *IEEE Seventh International Symposium on Service-Oriented System Engineering*, pp. 320-323, 2013.
- [19] Y. T. Hou, Yi Shi, and Ha. D. Sherali, “On Energy Provisioning and Relay Node Placement for Wireless Sensor Networks,” *In IEEE Trans. on Wireless Comm.*, (4)5:2579–2590, Sep. 2005.
- [20] ed. J. Wu. Auerhach, “*Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks*” Publications, 2006.
- [21] G. Karypis, E. H. Han, and V. K. Chameleon, “Hierarchical Clustering Using Dynamic Modeling,” *IEEE Computer*, Aug 1999.
- [22] M. M’edard, S. G. Finn, R. A. Barry, and R. G. Gallager, “Redundant Trees for Preplanned Recovery in Arbitrary Vertex-Redundant or Edge-Redundant Graphs,” *IEE/ACM Trans on Networking*, 7(5):641–652, Oct 1999.
- [23] J. Suomela, “Computational Complexity of Relay Placement in Sensor Networks” *In SOFSEM 2006, LNCS 3831*, pp. 521–529. Springer-Verlag, 2006.
- [24] S. Toumpis and G. A. Gupta, “Optimal Placement of Nodes in Large Sensor Networks Under a General Physical Layer Model,” *In Proc of IEEE SECON*, September 2005.
- [25] N. Heo and P. K. Varshney, “Energy-Efficient Deployment of Intelligent Mobile Sensor Networks,” *IEEE Transactions on Systems, Man, Cybernetics, Part A*, Vol. 35, No. 1, pp. 78-92, January 2005.
- [26] A. Kansal et al., “Controlled Mobility for Sustainable Wireless Sensor Networks,” *in the Proceedings of IEEE Sensor and Ad Hoc Communications and Networks (SECON’04)*, Santa Clara, CA, October 2004.
- [27] G. Wang, G. Cao, and T. La Porta, “Movement-Assisted Sensor Deployment,” *in the Proceedings of the 23rd International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM’04)*, Hong Kong, March 2004.

Paper VI

Hierarchal Placement of Smart Mobile Access Points in Wireless Sensor Networks using Fog Computing

A. Majd, G. Sahebi, M. Daneshtalab, J. Plosila and H. Tenhunen

Hierarchal Placement of Smart Mobile Access Points in Wireless Sensor Networks using Fog Computing

Amin Majd*, Golnaz Sahebi*, Masoud Daneshlab†, Juha Plosila* and Hannu Tenhunen*

* University of Turku, Finland

Email: {amin.majd, golnaz.sahebi, juplos, hannu.tenhunen@utu.fi}

†Mälardalen University and Royal Institute of Technology, Sweden

Email: masdan@kth.se

Abstract — Recent advances in computing and sensor technologies have facilitated the emergence of increasingly sophisticated and complex cyber-physical systems and wireless sensor networks. Moreover, integration of cyber-physical systems and wireless sensor networks with other contemporary technologies, such as unmanned aerial vehicles (i.e. drones) and fog computing, enables the creation of completely new smart solutions. By building upon the concept of a Smart Mobile Access Point (SMAP), which is a key element for a smart network, we propose a novel hierarchical placement strategy for SMAPs to improve scalability of SMAP based monitoring systems. SMAPs predict communication behavior based on information collected from the network, and select the best approach to support the network at any given time. In order to improve the network performance, they can autonomously change their positions. Therefore, placement of SMAPs has an important role in such systems. Initial placement of SMAPs is an NP problem. We solve it using a parallel implementation of the genetic algorithm with an efficient evaluation phase. The adopted hierarchical placement approach is scalable; it enables construction of arbitrarily large SMAP based systems.

Keywords- smart mobile access point; fog computing; wireless sensor networks; cyber-physical systems; multi-objective optimization; evolutionary computing; parallel approaches; genetic algorithms; parallel programming; multi-population; placement.

I. INTRODUCTION

The combination of Wireless Sensor Networks (WSN) and other new technologies, such as unmanned aerial vehicles (UAV) and mobile robots in general, has created a new revolution in the field of Cyber Physical Systems (CPS). Performance of a WSN can be significantly improved by using unmanned mobile nodes (either aerial or ground vehicles) as access points and even as sensor or monitoring nodes [10], [11].

The concept of a Smart Mobile Access Point (SMAP) provides a novel way to use UAVs or mobile robots to build an intelligent dynamic network. Such a network can consist of several clusters of SMAPs. By optimizing the positions of the SMAPs within each cluster, the quality of the network can be improved [14]. As SMAPs can adaptively change their positions, placement of SMAPs becomes a crucial issue in this approach. Adopting a hierarchal model for SMAP placement helps in expanding the network without an additional network reconfiguration cost. In the hierarchal model, there are two or more layers where an upper layer manages behavior and positioning of its respective lower layer. This hierarchal structure allows expansion of the network by adding new layers when needed.

Placement is a multi-objective optimization problem which can be solved using static or dynamic methods [1]. For a static problem, such as initial placement of SMAPs, Evolutionary Computing (EC) methods are the most well-known approaches [2], [3]; and a Genetic Algorithm (GA) is a powerful EC method [2]. There are several improvements for GAs. Among them, in this paper, a multi-population version of a GA (MPGA) [5] with an efficient fitness function for solving the initial placement problem in a hierarchical SMAP based system is proposed. The proposed method is based on the distributed fog computing model to enhance speed, accuracy, and scalability of the approach. The aim of this work is to improve resource utilization and energy-efficiency by the adopted hierarchical system architecture and optimal positioning of SMAPs.

The rest of the paper is organized as follows. Section II provides the background of this work. Section III reviews literature study in the area. Section IV proposes a parallel multi-population implementation of a GA, based on fog computing, to solve the initial hierarchical placement of SMAPs. In Section V, the proposed MPGA is evaluated by comparing with a traditional GA [17] and a mathematical method [19]. Finally, Section VI concludes the paper.

II. BACKGROUND

In this section, brief reviews on Smart Mobile Access Points (SMAP), access point placement methods and Evolutionary Algorithms (EA) are presented.

Smart Mobile Access Points

SMAPs are mobile access points that enable the creation of a smart sensor network [14]. The SMAPs predict the behaviour of the network. Also, they select and make the best strategy to support the network at any given time. A SMAP has authority to utilize previous knowledge and operations or creates new operations by evolving or learning through machine learning techniques [14]. A SMAP carries out the following three key operations: 1) Finding new (optimal) positions for access points to obtain a better coverage of sensors; 2) Making decisions about moving the access points to new areas [5]; 3) Participating in distributed communication and computation tasks (fog computing [6]) other than the processing carried out as an inherent part of the decision making in 1) and 2), illustrated in Figure 1.

Finding an optimal placement to improve energy consumption, reliability, and connectivity is an NP-hard problem [8]. Metaheuristic methods are the best approaches to solve such a problem. Different metaheuristic methods are presented for example in [9], [7].

III. RELATED WORK

Finding optimal placement to improve connectivity, reliability and energy consumption is an NP-hard problem [16]. There are different heuristic methods to solve it that presented in [15], [14]. In this section, router placement methods are divided into three main group: a) non-redundant placement, b) redundant and c) non-trivial redundant router placement. Y. Thomas et al. [14] have developed a heuristic algorithm to increase the network lifetime by iteratively moving an RN to a better location. In [18], its authors have presented several trajectory control algorithms with different assumptions on locating capabilities to achieve the objectives of reducing hop count and reducing overhead. W. Youssef et al. [17] have employed genetic algorithms for selecting the best spot for placing each gateway so that sensors' data can be delivered to a gateway with the least latency (GAHO). A. Krause et al. [19] have presented a data-driven approach (pSPIEL) that addresses the three central aspects of this problem: measuring the predictive quality of a set of sensor locations, predicting the communication cost involved with these placements, and designing an algorithm with provable quality guarantees that optimizes the NP-hard tradeoff. These methods have been utilized in the different problem, but there are several gaps that the proposed work covers them such as hierarchal architecture to solve the scalability problem and weighted sensitive to increase the focus of placement method to the more critical area for covering. The proposed method has several critical properties that motivate us to utilize it in these kinds of placement problems. These properties are the hierarchal architecture, the weighted sensitive fitness function, and the multi-population implementation to increase the reliability of results.

IV. PROPOSED APPROACH

In this section, a parallel implementation of a multi-population GA (MPGA) is proposed to establish an efficient method for solving the initial placement problem in hierarchal SMAP-based systems.

A. Algorithmic Layer

To have the best placement of SMAPs on different layers, all access points are categorized in our work into two main groups that are normal access points and SMAPs. Normal access points are static nodes and only transfer data and maintain connectivity in the network [14]. SMAPs can emulate all functionalities of normal access points, but they also can perform other tasks, like dynamically supporting other access points when needed. In this paper, however, to simplify the problem, normal static access points have been selected as target points for SMAPs on the lowest layer. Also, all SMAPs on each layer are divided into several clusters, and each cluster represents target points for other SMAPs on a higher level, as we are considering the static initial placement only. Moreover, each normal access point has an individual and independent weight which indicates the probability of needing support from SMAPs (i.e. the need for network reconfiguration) [14]. This probability is equal to a value that can be derived from different

parameters such as communication traffic and battery lifetime. In our approach, the rate of communication traffic is selected for this purpose.

B. Multi-Population Genetic Algorithm

SMAPs work on a discrete space; therefore, a Genetic Algorithm (GA) is a suitable choice to solve the placement problem [2], [4], [14]. Furthermore, a multi-population GA is more efficient than a sequential GA for these problems since the multi-population method increases the selection pressure and improves the diversity. Thus, we have utilized a parallel multi-population genetic algorithm to achieve the best results for selecting the best positions according to Section II.

In this work, there are some important aspects concerning the multi-population strategy that have been efficiently matched with the problem of hierarchical placement of SMAPs and enhanced to obtain more accurate solutions. The key benefits of multi-population GAs, regarding their efficient utilization in this problem, are: 1) Increasing the diversity of initial population, 2) Increasing the selection pressure, and 3) Migration operator [2], [14].

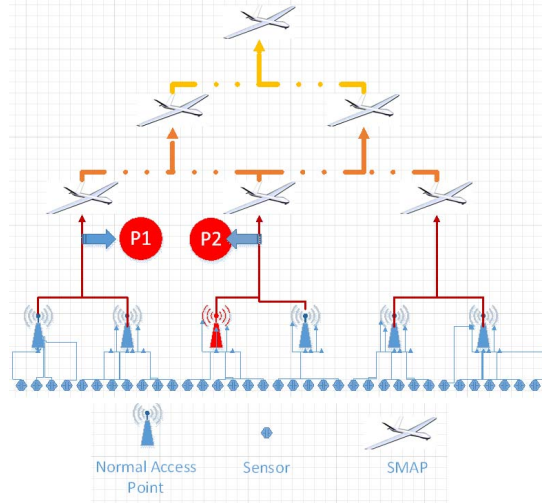


Fig. 1. SMAP model the using Fog network

The multi-population strategy leads to a better diversity of population which improves the search space of the best positions. This diversity also enhances the selection pressure to obtain the most accurate placement. Furthermore, the migration operator enables processors to exchange their best genetic material, thereby improving their genetic populations.

The proposed multi-population method uses a ring topology because of its simplicity and efficiency in near-neighbor communication because communication cost is the critical parameter, and the proposed work obtains the lowest connections to communication in the ring topology. Our approach can run on both the shared memory and the message passing architectures. The algorithm obtains a result which satisfies all conditions of an efficient

placement. The main phases of our algorithm are shown as follows:

1) Initialization

First, a matrix, in which each cell is a data structure of the form (X_i, Y_i) , is created. Each row of the matrix is a chromosome, which is a set of points, and also, it can be the best solution. All of these points are a possible placement of the SMAPs in the proposed search space on different layers. Values of X_i and Y_i are real values ($X_i, Y_i \in R$) and are randomly generated. Let us assume that the number of SMAPs on the first layer (the lowest layer) is equal to m^1 , on the second layer is equal to m^2 , and, correspondingly, on the l^{th} layer (the highest layer) is equal to m^l , where $m^1 \leq m^2 \leq \dots \leq m^l$. Each column in this matrix corresponds to the location of each placement of a SMAP. Hence, the number of rows is equal to the size of the initial population (N_{pop}) and the number of columns is equal to the number of all SMAPs on all layers. A sample chromosome for two layers of SMAPs is illustrated in Figure 2.

X_1^1, Y_1^1	X_2^1, Y_2^1	...	$X_{m^1}^1, Y_{m^1}^1$	X_1^2, Y_1^2	X_2^2, Y_2^2	...	$X_{m^2}^2, Y_{m^2}^2$
----------------	----------------	-----	------------------------	----------------	----------------	-----	------------------------

Fig. 2. Structure of the chromosome for two layers SMAPs

2) Chromosome Evaluation

In all problems that are solved by GAs, finding a suitable fitness function to evaluate all chromosomes is the essential task. In our approach, the evaluation function receives a chromosome as its input and returns a real value as its output. The output is computed based on the distance between the SMAPs and all normal access points. A chromosome for which the fitness function evaluates to the lowest value is the best chromosome (solution). In other words, this problem is a minimization optimization problem.

There are some simple equations that are suitable for selecting the best place for a single SMAP, but when the number of SMAPs is increased, and SMAPs should be placed in different layers, the placement problem becomes more complex, and the simple functions do not provide correct results anymore and cannot therefore be applied. We propose a new and efficient fitness function DM2 in Equation 4. It is a scalable fitness function, providing accurate results for any number of SMAPs on different hierarchy levels. This equation is compatible with the scalability problem. To construct our fitness function, we first define:

$$\theta = \frac{\sum_{i=1}^{m^1} \left(D_{NN_i} \times W_{D_{NN_i}} + \left(\frac{D_{NN_i}^2}{4} \times W_{D^2_{NN_i}} \right) \right)}{Distance^1} \quad (1)$$

$$D = \lfloor \sqrt{n} \rfloor$$

where n is the number of normal access points, and D_{NN_i} is the Euclidean distance between the SMAP number i and D nearest normal access points on the first layer. The $W_{D_{NN_i}}$ is the total weights of all normal access points in D_{NN_i} . $Distance^t$ is the sum of distances between all SMAPs in the t^{th} layer and is defined in Equation 2. $W_{D_{SM_j}}$ is the total weight of all SMAPs on the j^{th} layer. So Θ is the fitness value for the first layer.

Equation 3 is used for the layers other than the first layer: γ^l is the fitness value for l^{th} layer. By combining Equations 1 and 3, we then obtain the fitness function DM2 presented in Equation 4 that is total fitness value of all layers.

$$Distance^t = \sum_{i=1}^{m^t} \sum_{\substack{j=1 \\ i \neq j}}^{m^t} \sqrt{(X_i - X_j)^2 + (Y_i - Y_j)^2} \quad (2)$$

$$\gamma^l = \sum_{j=1}^{m^l} \left(\left(D_{SM_j} \times W_{D_{SM_j}} \right) + \left(\frac{D_{SM_j}^2}{4} \times W_{D^2_{SM_j}} \right) \right) / Distance^l \quad (3)$$

$$DM2 = \theta + \sum_{l=1}^l \gamma^l \quad (4)$$

3) Selection operator

The basic idea of the selection operator is that it gives preference to better chromosomes and allows them to pass on their genes to the next generation. The proposed algorithm adopts the tournament method with three members to select the best chromosome. First, three chromosomes are randomly selected, and then the best one is selected for the next generation in every cycle.

4) Crossover operator

A GA has two main operators: crossover and mutation. Since a GA is a semi-random optimization method, its operators do not occur with 100% certainty. In other words, they happen with less than 100% probability. The crossover operator selects genes from chromosomes, that are the parents, and creates a new offspring. This means that the operator exploits the search space to find more accurate solutions. The crossover chromosomes are chosen randomly from the population according to a probability called a crossover rate (P_c). The crossover rate determines the frequency with which the crossover operator is applied [12].

5) Mutation operator

The mutation operator is an exploration operation and occurs with a probability called a mutation rate (P_m) [28]. If this operator happens on a chromosome, it randomly changes the new offspring according to the mutation rate. In other words, this operator explores the search space to discover a new search area and prevents all solutions in a population from falling into a local optimum. The mutation rate is a measure for determining when mutations occur over time [13].

6) Replacement operator

The current generation of chromosomes is replaced by the recently generated offspring based on a particular replacement approach. In our algorithm, the steady-state strategy is utilized for the replacement operator. The operation compares each chromosome of the current population with the last generation. If a chromosome in the current generation is better than its corresponding chromosome in the last generation, the new chromosome replaces the old one.

7) Migration operator

During the migration process, some of the best chromosomes, in each processor, are chosen and sent to the

next processor in the ring at each migration time point. Concurrently, each processor receives the chromosomes sent by the previous processor and replaces its worst chromosomes with the received ones.

8) Stopping strategy

Finally, the algorithm stops after some certain number of iterations, and the best chromosome is determined. This represents the best solution for the SMAP placement.

V. EXPERIMENTAL RESULTS

The proposed work has been implemented on Intel® Core™ i5, CPU @ 2.9GHz, RAM 8 GB. VC++ 2015 has been utilized for the implementation and MPI instructions for parallelization. Furthermore, MPICH2.3.2 has been applied for parallel execution of the algorithms. The proposed MPGA based approach has been tested on four processors in all tests for two case studies. Normal access points have different weights. The results have been compared with the GAHO (GAHO) [17] and a mathematical method (pSPIEL) [19].

In the considered two case studies, presented as follows, the algorithms should select the best locations for several SMAPs in two layers. In a single SMAP placement, a pSPIEL can find the best place, but when dealing with several SMAPs, pSPIEL are not applicable in practice, as they cannot determine the best positions for the SMAPs in a finite time.

In this paper, the MPGA and GAHO methods are run on all the two case studies for 30 times with 100 iterations in each round. The size of the initial population is 500. The results for the two case studies are shown in Figures 3-6; and Tables 1. In these placement problems, the DM2 function (Equation 4) is utilized as the fitness function for both MPGA and GA to obtain the best results. The overall results indicate that the proposed fitness function is very efficient, enabling MPGA and GA to find the best positions also in the complex placement cases. The results also demonstrate that MPGA performs much better than GA.

The first case study has 32 normal access points that are concurrent based on the center point. They are divided into two subsets, i.e., A and B, as shown below. The weight of each normal access points in A and B is equal to 2 and 1, respectively. The best places on the first layer are in the coordinates (-10, 0), (0, -10), (10, 0), and (0, 10); and the best places on the second layer are in the coordinates (5, 0) and (-5, 0). The results of MPGA and GAHO have been obtained in 100 iterations with 500 chromosomes as an initial population.

$$A = \left\{ \begin{array}{l} (-13,3), (-10,3), (-7,3), (-7,0), (-7,-3), (-10,-3), (-13,-3), \\ (-13,0), (13,3), (13,0), (13,-3), (10,3), (10,-3), (7,3), (7,0), (7,-3) \end{array} \right\}$$

$$B = \left\{ \begin{array}{l} (-3,-7), (-3,-10), (-3,-13), (0,-7), (0,-13), (3,-7), (3,-10), \\ (3,-13), (-3,7), (-3,10), (-3,13), (0,7), (0,13), (3,7), (3,10), \\ (3,13) \end{array} \right\}$$

The second case study has sixty normal access points, residing on two hyperbolic curves (a horizontal curve and a vertical one). The mathematical definitions of these curves are presented in Equations 5 and 6. The weights of the normal access points in Equation 5 (the horizontal curve) and Equation 6 (the vertical curve) are equal to 2 and 1,

respectively. According to the theory of hyperbolic curves, parameters a, b, and c (in Equations 5, 6, and 7) are very effective. In addition, the points (0, c) and (0, -c) in the horizontal hyperbolic curve, and (c, 0) and (-c, 0) in the vertical hyperbolic curve, have the minimum distances to any points on these hyperbolic curves. These points are the best places for placement of SMAPs on the first layer.

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1, \quad a = 3, b = 4 \quad (5)$$

$$\frac{y^2}{b^2} - \frac{x^2}{a^2} = 1, \quad a = 3, b = 4 \quad (6)$$

$$b = \sqrt{c^2 - a^2} \quad (7)$$

Figure 3 shows the placements that have been discovered by MPGA in the first case study. It clearly indicates that the results are very close to the optimal points, and the best results are obtained 35 times in the series of 40 runs. The corresponding results of the GAHO placement method are illustrated in Figure 4. It can be clearly seen that the results of MPGA are 47% in the first case study and 38% in the second case study more reliable than the GAHO method. Figures 5 and 6 indicate the results of the MPGA and GAHO placement methods, respectively, for the second case study. The statistical results of both considered methods and pSPIEL, for all two case studies, are illustrated in Table 1. These results demonstrate that MPGA is more accurate and efficient than the other methods. They also indicate that the proposed MPGA method is successful in both simple and complex case studies. The migration operation in MPGA is the primary factor for the accuracy of the results in our approach. Therefore, the accuracy is a strong motivation for using the MPGA method.

TABLE I. STATISTICAL RESULTS

	Case	The Best Distance	Correct Placement Count	The Worst Distance
PGA	1	0	26	11.814
GAHO		0	7	34.899
pSPIEL		-	0	-
PGA	2	0	28	9.465
GAHO		0	13	18.927
pSPIEL		-	0	-

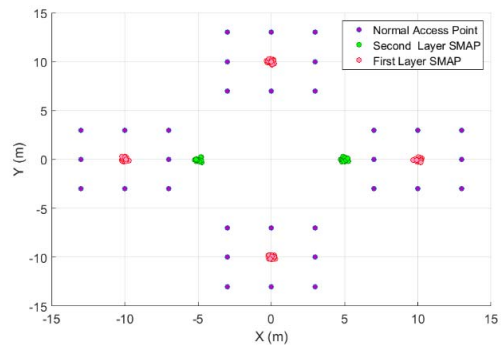


Fig. 3. Placement with MPGA in the case study 1.

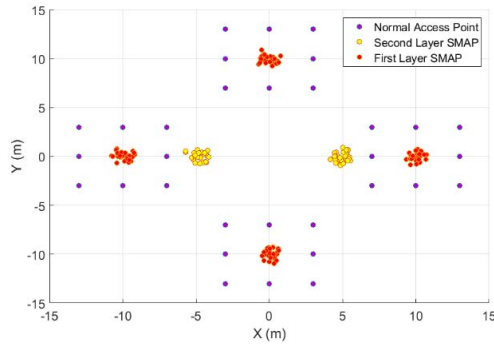


Fig. 4. Placement with GAHO in the case study 1.

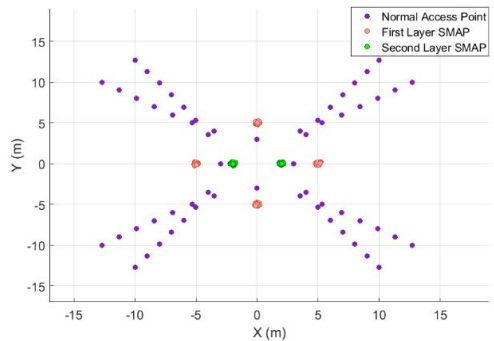


Fig. 5. Placement with MPGA in the case study 2.

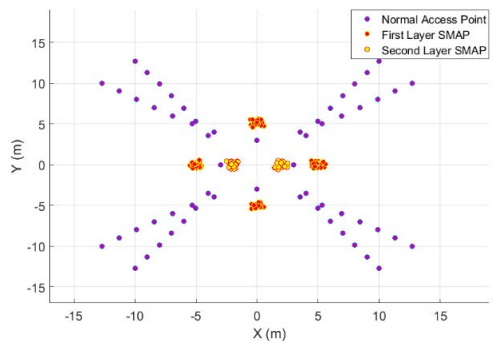


Fig. 6. Placement with GAHO in the case study 2.

REFERENCES

- [1] Y. J. Cha, A. Raich, "Optimal placement of active control devices and sensors in frame structures using multi-objective genetic algorithms," *Structural Control and Health Monitoring*, Vol. 20, pp. 16-44, 2013
- [2] A. Majd, Sh. Lotfi and G. Sahebi, "Review on Parallel Evolutionary Computing and Introduce Three General Framework to Parallelize All EC Algorithms," *The 5th Conference on Information and Knowledge Technology*, IEEE, pp. 61-66, 2013.
- [3] A. Majd and G. Sahebi, "A Survey on Parallel Evolutionary Computing and Introduce Four General Frameworks to Parallelize all EC Algorithms and Create New Operation for Migration," *Journal of Information and Computing Science*, vol. 9, pp.97-105, 2014.
- [4] A. Majd, Sh. Lotfi, G. Sahebi, M. Daneshmand and J. Plosila, "PICA: Multi-Population Implementation of Parallel Imperialist Competitive Algorithms," *24th Euromicro International Conferences on Parallel, Distributed and Network-Based Processing*, PDP 2016.

- [5] A. Basturk, R. Akay and A. Kalinli, "Comparison of fine-grained and coarse-grained parallel models in particle swarm optimization algorithm" *2nd World Conference on Information Technology (WCIT)*, 2011.
- [6] J. Zhu, D. S. Chan, M. S. Prabhu, P. Natarajan, H. Hu and F. Bonomi, "Improving Web Sites Performance Using Edge Servers in Fog Computing Architecture," *IEEE Seventh International Symposium on Service-Oriented System Engineering*, pp. 320-323, 2013.
- [7] Y. T. Hou, Yi Shi, and Ha. D. Sherali, "On Energy Provisioning and Relay Node Placement for Wireless Sensor Networks," *In IEEE Trans. on Wireless Comm.*, (4)5:2579-2590, Sep. 2005.
- [8] J. Suomela, "Computational Complexity of Relay Placement in Sensor Networks" *In SOFSEM 2006*, LNCS 3831, pp. 521-529. Springer-Verlag, 2006.
- [9] S. Toumpis and G. A. Gupta, "Optimal Placement of Nodes in Large Sensor Networks Under a General Physical Layer Model," *In Proc of IEEE SECON*, September 2005.
- [10] N. Heo and P. K. Varshney, "Energy-Efficient Deployment of Intelligent Mobile Sensor Networks," *IEEE Transactions on Systems, Man, Cybernetics, Part A*, Vol. 35, No. 1, pp. 78-92, January 2005.
- [11] A. Kansal et al., "Controlled Mobility for Sustainable Wireless Sensor Networks," in the *Proceedings of IEEE Sensor and Ad Hoc Communications and Networks (SECON'04)*, Santa Clara, CA, October 2004.
- [12] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning." *Machine Learning*, 1988.
- [13] T. Back, "Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms", Oxford University Press Oxford, 1996.
- [14] A. Majd, G. Sahebi, M. Daneshmand, J. Plosila and H. Tenhunen, "Placement of Smart Mobile Access Points in Wireless Sensor Networks and Cyber-Physical Systems using Fog Computing" *The 16th IEEE International Conference on Scalable Computing and Communications (Scalcom 2016)*, 2016.
- [15] S. Toumpis and G. A. Gupta. *Optimal Placement of Nodes in Large Sensor Networks Under a General Physical Layer Model*. *In Proc of IEEE SECON*, September 2005
- [16] J. Suomela. *Computational Complexity of Relay Placement in Sensor Networks*. *In SOFSEM 2006*, LNCS 3831, pp. 521-529. Springer-Verlag, 2006.
- [17] W. Youssef and M. Younis, "Intelligent Gateways Placement for Reduced Data Latency in Wireless Sensor Networks" *In Proceedings of the 32nd IEEE International Conference on Communications (ICC 2007)*, Glasgow, Scotland, UK, June 2007
- [18] Ch. Shen, O. Koc, Ch. Jaikao and Z. Huang, "Trajectory control of mobile access points in MANET" *the IEEE GLOBECOM*, 2005.
- [19] A. Karause, C. Guestrin, A. Gupta and J. Kleinberg "Near-optimal Sensor Placements: Maximizing Information while Minimizing Communication Cost," *the 5th international conference on Information processing in sensor networks*, pp. 2-10, 2006.

Paper VII

Optimizing Scheduling for Heterogeneous Computing Systems using Combinatorial Meta-heuristic Solution

A. Majd, G. Sahebi, M. Daneshtalab, E. Troubitsyna

Optimizing Scheduling for Heterogeneous Computing Systems using Combinatorial Meta-heuristic Solution

Amin Majd

Department of Information Technology
Åbo Akademi University
Turku, Finland
amin.majd@abo.fi

Masoud Daneshtalab

Mälardalen University,
Västerås, Sweden,
Masoud.daneshtalab@mdh.se

Golnaz Sahebi

Department of Information Technology
University of Turku
Turku, Finland
golnaz.sahebi@utu.fi

Elena Troubitsyna

Department of Information Technology
Åbo Akademi University
Turku, Finland
elena.troubitsyna@abo.fi

Abstract — Today, based on fast development especially in Network-on-Chip (NoC)-based many-core systems, the task scheduling problem plays a critical role in high-performance computing. It is an NP-hard problem. The complexity increases further when the scheduling problem is applied to heterogeneous platforms. Exploring the whole search space in order to find the optimal solution is not time efficient, thus metaheuristics are mostly used to find a near-optimal solution in a reasonable amount of time. We propose a compound method to select the best near-optimal task schedule in the heterogeneous platform in order to minimize the execution time. For this, we combine a new parallel meta-heuristic method with a greedy scheme. We introduce a novel metaheuristic method for near-optimal scheduling that can provide performance guarantees for multiple applications implemented on a shared platform. Applications are modeled as directed acyclic task graphs (DAG) for execution on a heterogeneous NoC-based many-core platform with given communication costs. We introduce an order-based encoding especially for pipelined operation that improves (decreases) execution time by more than 46%. Moreover, we present a novel multi-population method inspired by both genetic and imperialist competitive algorithms specialized for the scheduling problem, improving the convergence policy and selection pressure. The potential of the approach is demonstrated by experiments using a Sobel filter, SUSAN filter, RASTA-PLP, and JPEG encoder as real-world case studies.

Keywords- *parallel imperialist competitive algorithm (PICA); multi-population technique; evolutionary computing (EC); task graph scheduling; heterogeneous platform.*

I. INTRODUCTION

Contemporary multiprocessor system-on-chip (MPSoC) based parallel processing, in a vast variety of applications, is the result of many breakthroughs over the last two decades. The development of embedded MPSoCs has led to their use in many applications like health monitoring, video and audio processing, and autonomous vehicles to mention just a few. The data and program code for these applications can be distributed among the available multiple cores, and thus maximal benefits from these parallel systems can be obtained by employing efficient task partitioning and scheduling strategies. A homogeneous MPSoC contains a set of identical

processing elements (PEs), typically programmable processors. The usage of homogeneous MPSoCs simplifies task migration, while heterogeneous MPSoCs support a wider variety of applications, because they integrate different PEs [26].

In this work, we consider the deterministic model for the task scheduling problem, where the execution time of tasks and the data communication time between tasks are assigned. The directed acyclic task graph (DAG) that represents the precedence relations of the tasks is well known as an NP-complete problem. Task mapping, which consists of finding a placement for a set of tasks that meets a specific requirement such as energy consumption savings is an important issue. Static mapping defines task placement at design time; dynamic mapping defines task placement at runtime [27]. Many heuristic methods for the task scheduling dilemma have been proposed [1]-[7] because the precedence constraints between tasks can be non-uniform therefore rendering the requirement for a uniformity solution. We assume that a NoC is a heterogeneous multiprocessor system and non-preemptive (each processor completes the current task before the new one starts its execution).

Recently, evolutionary approaches have been developed to solve this problem. For example, a genetic algorithm (GA) based approach can better locate a near optimal solution than a list schedule in most cases [8], [9], and [10]. For multiprocessor scheduling problem, several GA methods have been applied, but none of them have considered the communication cost. On the other hand, a number of parallel GA methods have been proposed to solve the task graph scheduling problem in [13]. However, in all of them, there is a fundamental problem that derives from the convergence idea of a GA and our paper resolve this problem. In this paper, we tackle this problem by a combination of a GA and the Imperialist Competitive Algorithm (ICA) [14]. Moreover, we explicitly consider the communication delays between processors. When two communicating tasks are mapped onto the same processor we assume that the communication delay is zero, and if they are mapped on different processors, there will be a communication cost. For this problem, we propose an extension of the priority-order-based coding method as the priority-order-based country (OBC) for the pipelined execution of the tasks. This coding shows the priority and

selected processor to run each task. The priority-based encoding [11] is the knowledge of how to handle the problem of producing the encoding that can treat the precedence constraints efficiently. To obtain the best result, we present a new multi-population method, a combination of ICA and GA. This combination helps our approach to match the scheduling problem and addresses some problems of GA, such as the convergence policy and selection pressure, by keeping all countries (possible solutions) in all iterations and removing the general selection operation such as roulette wheel and tournament methods. In this paper, we propose a novel multi-population implementation of the ICGA, a hybrid approach using both ICA and GA, for solving the task scheduling problem. The goal is to improve the execution time and reliability (proximity of the achieved result and the best result) of the task scheduling.

The rest of the paper is organized as follows. Section 2 discusses the related work. In Section 3, definitions and important parameters in the task scheduling problem are presented. In Section 4, the proposed parallel algorithm for the scheduling problem (MICGA) is presented. Section 5 presents the experimental results and demonstrates the efficiency of the proposed algorithm. We end with concluding remarks in Section 6.

II. RELATED WORKS

The basic idea is to make an ordered list of nodes by assigning them priorities, and then to repeatedly execute the following two steps until a valid schedule is obtained: (1) Select from the list the node with the highest priority for scheduling. (2) Select a processor to accommodate this node.

In realistic cases, scheduling needs to exploit the parallelism by identifying the task graph structure and take into consideration task granularity, arbitrary computation, and communication costs.

The modified critical path algorithm (MCP) [29] is proposed based on the latest possible start time of a node. A node's latest possible start time is determined via the as-late-as-possible (ALAP) binding by traversing the task graph upward from the exit nodes to the entry nodes while pulling the nodes start times downwards as much as possible. The latest possible start time of the node itself is followed by a decreasing order of the latest possible start time of its successor nodes. In addition, the dominant sequence clustering algorithm (DSC) [29] is based on the dominant sequence, which is essentially the critical path of the partially scheduled task graph at each step. At each step, DSC checks whether the highest CP (the critical path of task graph) node is a ready node. If so, DSC schedules it to a processor allowing the minimum start time. Such a minimum start time may be achieved by rescheduling some of the node's predecessors to the same processor. If the highest CP node is not a ready node, DSC does not select it for scheduling. Instead, it chooses the highest node which lies on a path reaching the CP for scheduling. The mobility directed algorithm (MD) [29] selects a node at each step based on relative mobility which is defined as the difference between a node's earliest start time and latest start time. Similar to the ALAP binding, the earliest possible start time is assigned to each node via the as-soon-as-possible (ASAP) binding. This is performed by traversing the task graph downward from the entry nodes to the exit nodes while pulling the nodes upward as much as possible. Moreover, relative mobility is obtained by dividing the mobility with the node's computation cost. Basically, a node with zero mobility is a node on the CP. At each step, MD schedules the node with the smallest mobility

to the first processor having a large enough time to accommodate the node without considering the minimization of the node's start time. After a node is scheduled, all the relative mobility is updated.

In [13], a multi-population implementation of GA (MPGA) is presented which outperforms deterministic and nondeterministic methods [2]-[8]. In [29], a new encoding mechanism with a multi-functional chromosome is presented and it uses the priority representation that called priority-based multi-chromosome (PMC). PMC can efficiently represent a task schedule and assign tasks to processors. PMC is another metaheuristic method that is presented in [29] that uses a priority model for encoding the chromosomes and use a GA to achieve the results.

Research on static mapping method includes the work of Lei et al., who proposed a genetic mapping algorithm to optimize application execution time [20]. In their work, graphs represent applications and the target architecture is a NoC. Wu, et al. also investigated genetic mapping algorithms [21]. By combining dynamic voltage scaling techniques with mapping, they achieved 51% savings in energy consumption. Murali et al. explored mappings for more than one application in NoC design, using the tabu search (TS) algorithm [22]. Manolache, et al. investigated task mapping in NoCs, trying to guarantee packet latency [23]. For this purpose, both the task-mapping algorithm (TS) and the routing algorithm are defined at design time. Hu et al. presented a branch-and-bound algorithm to map a set of IP cores (IPs) onto a NoC with bandwidth reservation [24]. Their results show energy savings of 51.7% in the communication architecture. Marcon et al. investigated how to map modules into a NoC, targeting low energy consumption [25]. They compared several algorithms, using a model that characterizes applications by their inter-task communication volume. Y. Xu, K. Li, J. Hu and K. li in [26] presented a task scheduling scheme on heterogeneous computing systems using a multiple priority queues genetic algorithm (MPQGA). Their experimental results for large-sized problems for a large set of randomly generated graphs as well as graphs of real-world problems with various characteristics showed that the proposed MPQGA algorithm outperformed two non-evolutionary heuristics and a random search method in terms of schedule quality.

III. BACKGROUND

In the following we present the considered system model.

A. Hardware platform model

A heterogeneous MPSoC is a set of different PEs interacting through a communication network. For our work, a NoC-based heterogeneous MPSoC model is selected, in which each PE manages execution of one task at a time. PEs can support either software or hardware task execution. Software tasks execute in instruction set processors (ISPs), and hardware tasks in reconfigurable logic (RL) or dedicated IPs [27]. Among the available PEs, one of the processors is selected as a manager processor (MP). The MP is responsible for resource control, configuration control, and four task manipulation operations, namely: binding, scheduling, migration, and mapping [27]. The MP runs each application's initial task. When a running task needs to communicate with another task not yet mapped, the latter is loaded into a PE from the task memory. A DAG, in which vertices represent software or hardware tasks and edges define communicating PE pairs, models each application. Each edge of a DAG defines a master-slave communication channel. The edge

source vertex is the master of the communication, and the edge destination vertex is the slave.

B. Application Model

We assume a weighted DAG. In such a DAG, each node represents a task that has a finite execution time, and each edge illustrates the communication cost between two connected tasks. Also, the DAG can define the priority of all tasks.

We assume that each application TG_a is modeled as a weighted DAG. In this DAG, each node represents a task that has a finite execution time, and each edge illustrates the communication cost between two connected tasks. The edges also represent the precedence relations. Let TG_a be a weighted DAG defined as a 4-tuple $TG_a = (V_a, E_a, W_a, C_a)$, where $V_a = \{T_1^a, T_2^a, \dots, T_n^a\}$ is the set of vertices (tasks), $E_a = \{e_{ij}^a = (T_i^a, T_j^a)\}$ is the set of communication (dependency) edges, W_a is the set of task weights (execution times), C_a is the set of edge weights (communication times), $w_i^a \in W_a$ is the execution time of task $T_i^a \in V_a$, $c_{i,j}^a \in C_a$ is the communication cost (delay) on the edge of $e_{i,j}^a \in E_a$, that is assumed to be zero if T_i^a and T_j^a are bound to the same processor. We assume a simple linear model between the execution times and PE frequencies, e.g., if PE_1 runs two times faster than PE_2 , then task T_i^a also runs two times faster on PE_1 than PE_2 .

The execution time of an application TG_a is defined as the completion time of its last task. When constructing a schedule for execution of the applications, it is of great importance to consider both task execution times and the communication times. Since considering the communication times significantly increases the number of schedules to be explored, most of the previous work only consider the task execution times. However, when running tasks with precedence relation on different processing elements, the communication overhead is not negligible and it has to be considered in the model. To this end, we explicitly model the communication delays.

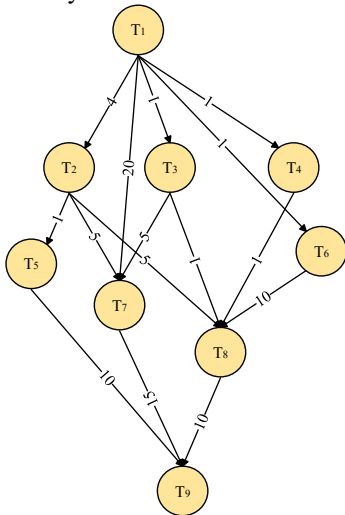


Figure 1. DAG Task Graph

C. Problem Statement

Our task scheduling method, MICGA, takes a set of task graphs $\{TG_1, \dots, TG_N\}$ and a set of processing elements $\{PE_1, \dots, PE_m\}$ as its inputs. The output of MICGA is the task bindings, i.e. task to processor mappings and static schedules,

i.e. the order of task executions. The overall goal of the scheduling is to minimize the execution times (also known as the schedule length, latency or makespan) of all applications while preserving the precedence constraints. Let f_a represent the execution time of application TG_a . We present the following mathematical formulation as our optimization problem:

$$\begin{aligned} \min & f_1, \dots, f_N, \\ \text{s.t.} & t_i^a - w_i^a - e_{ji}^a \geq t_j^a, a \in [1, \dots, N], \forall e_{ij}^a \in E_a, \end{aligned}$$

where t_i^a is the completion time of T_i^a . Note that the above formulation is a multi-objective optimization problem as we want to minimize the execution times of all applications. The optimization is subject to the precedence relations imposed by the application graphs.

IV. PROPOSED METHOD

In this work, we combine the GA and ICA methods and customize the resulting hybrid method for the scheduling problem in heterogeneous systems. The operations of GA are suitable for scheduling but its convergence strategy to find more reliable and accurate results is not efficient [15], [16], [17], because the selection operation in GA misses some chromosomes that have a probability to become potential genes for obtaining the best results. In ICA (where we use countries as the population), all the countries will be available in all iterations and they may move to other places in the search space. However, in GA, the selection operation highly depends on random functions which can easily converge to a local optimum. To solve this, we use an efficient convergence strategy in order to improve the reliability of results.

All the scheduled tasks in a DAG should satisfy the precedence relations, and therefore we use the order based coding mechanism suited for multiprocessor scheduling. It should fulfill the following rules:

1. All the predecessors of a task must have completed their execution before initiating the task execution.
2. In a DAG, all the tasks must be executed at least once. This representation eliminates the need to consider the precedence relations between the computational tasks. The precedence relation is encoded in the rule 1.

A. Order-based country (OBC)

There are several approaches [8], [10], and [12][12] that have used GA for the multiprocessor scheduling problem, but they suffer from inefficient coding methods. So, how to encode a solution of the problem into a chromosome is a key issue here. This has been investigated from two different angles which are: mapping characters from the genotype space to the phenotype space when countries are encoded into a solution [29]; and metamorphosis properties when countries are manipulated by genetic operators [14].

There are two main challenges involved with the encoding problem: 1) there is a need for storing a huge number of chromosomes for each schedule; 2) in the case of a large number of tasks, the exploration and exploitation operators will have difficulties in working on the precedence relations among tasks.

To overcome these challenges, we introduce an extension of OBC that strings a present task priority of task nodes with the corresponding processor simultaneously. For example, Figure 3 shows an OBC that represents nine tasks along with three processors with mentioned DAG in Figure 1.

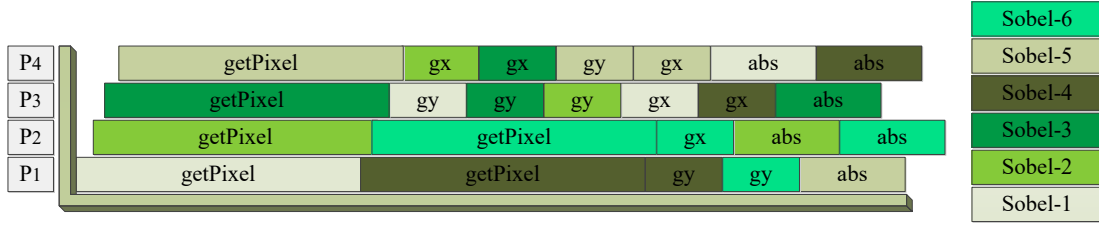


Figure 2. A proposed schedule of six Sobel application on four processors ($pp=6$)

B. Priority-based encoding/decoding

Finding an encoding which can treat the precedence constraint efficiently is a critical step. The proposed encoding method is based on the task priority, and our work focuses on the pipelined execution model of applications. In the pipeline model, input data is divided into several parts, and these parts are then one by one used as inputs to an application. We make a pack of the parts and find the best schedule for this pack of data. Each part is an application such as Sobel Application (Figure 7), and the number of parts (applications) in each pack is pp . A proposed schedule of six Sobel applications ($pp=6$) on four processors is illustrated on Figure 2. The following procedure explains the generation of the initial OBC:

First, we input the order-based task numbers $X_j[i]$ ($1 \leq X_j[i] \leq \# \text{Task}$, $1 \leq j \leq pp$ and $1 \leq i \leq \# \text{Task}$) randomly independent from the processor number in the X rows. Then, the algorithm fills the Y rows with random numbers $Y_j[i]$ ($1 \leq Y_j[i] \leq \# \text{Processor}$ and $1 \leq j \leq pp$).

C. ICGA Operation

Now, a fast review of ICGA is presented in this subsection then in the next sections all operations with more detail are presented. In the first ICGA generates k countries ($\{C_1, \dots, C_k\}$), i.e., k is the the number of countries, (see Figure 4.1), then ICGA sorts the countries, so that:

if ($i \leq j$) then ($f(C_i) \leq f(C_j)$), where f is the fitness function.

Then, the best countries, whose fitness values are the lowest (because this problem is a minimizing problem) ($\{C_1, C_2, \dots, C_{em}\}$ where $em = \# \text{number of empires}$) will be selected to become imperialists, and the remaining countries ($\{C_{em+1}, \dots, C_k\}$) form the colonies of these imperialists. These colonies start moving toward their respective imperialists, after all colonies have been divided among the imperialists [14].

The next step computes the power of each imperialist and the imperialistic competitive step follows. The weakest imperialist loses its weakest colony and the selected imperialist obtains this colony. These steps are repeated until reaching a termination condition. There can be different types of termination conditions. For example, ICGA could be set to stop when we have one imperialist with all colonies as its members.

Similarly to other ECs, when solving a large problem with a far-reaching search area, we need a large initial population to obtain a more accurate and reliable result. We can also prevent convergence to local optima by using a multi-population method, where each processor runs ICGA independently. If one processor converges to a local optimum, all the other processors can continue their work on other parts of the search space.

D. Multi-Population ICGA (MICGA)

To implement MICGA, we use a selective local search strategy which runs on MICGA, with several processors connected together based on a ring topology and using a message passing method for communication between the processors. In each processor, we first initiate independent

countries and run the ICGA independently. Occasionally, the best country is copied from a processor P_i to P_{i+1} , replacing the worse country in P_{i+1} . This operation takes place synchronously (simultaneously) in all processors. The number of countries in each processor remains the same even when countries migrate to other processors. In Figure 5, the pseudo code for the multi-population ICGA is presented.

In the multi-population ICGA, we increase the number of all countries along with the selection pressure [16], [17] which helps in obtaining more accurate results in a short time period. Also, the convergence to results is much faster than in the sequential ICGA. In EC methods, when the diversity of countries are high enough, the rate of convergence rises. The following steps run until MICGA obtains the best results.

Priority	1	2	3	4	5	6	7	8	9	
X1	1	3	2	7	4	5	6	8	9	
Y1	1	2	2	1	2	1	3	2	1	
X2	1	2	3	4	6	7	5	8	9	
Y2	2	2	3	1	3	2	1	3	1	
P1-Chain	1	7	5	9	4	5	9			
	X1				X2					
P2-Chain	3	2	4	8	1	2	7			
	X1				X2					
P3-Chain	6	3	6	8						
	X1	X2								

Figure 3. Chain of tasks based on the presented country

1) Make Countries.

As previously mentioned, each processor generates k countries (such as Figure 3), having its own world ($World_\beta = \{C_1, C_2, \dots, C_k\}$ and ($World_\beta = World_\gamma$ or $World_\beta \neq World_\gamma$)), where all worlds are independent.

2) Evaluation of Countries.

In all problems that are solved by meta-heuristic methods such as GAs and ICA, the essential task is to find a suitable fitness function to evaluate the whole population. In the proposed approach, the evaluation function receives a country as its input and returns a real value as its output. The output is computed based on the execution time of the application task graph by each country. A country with the lowest fitness function value is the best country (solution). In other words, this is a minimization optimization problem. The evaluation operation is composed of several steps that are introduced below. Note that these steps run in all countries. The Evaluation operation in ICGA run three step that presented as follow:

a) Making Tasks Chains on each Processor.

It is clear that each country represents an arbitrary schedule of tasks. For each processor, the algorithm makes a chain of tasks based on Equation 1, the chain of tasks define that which tasks should run on the same processor, then computes the total execution time of each chain based on Equation 2. Chains of tasks are presented in Figure 3.

$$chain_{P_i} = (\cup_{j=1}^{pp} \cup_{i=1}^{n} X_i[j] \mid Y_i[j] = P_i) \quad (1)$$

$$CCost = \sum_{i=1}^{|chain_{p_i}|} p_j \quad (2)$$

b) Assigning Probabilities to Processors.

Since this algorithm is proposed for heterogeneous systems, the greedy algorithm (Knapsack's algorithm [28]) is utilized to compute the mapping probability of chains to processors. Based on the Knapsack idea, each chain with a higher computation time has a higher probability to be assigned to faster processors. Also, the algorithm computes the probability of chains based on their total computation time and then runs a Roulette wheel algorithm [12] to assign chains to processors. In this phase, there is a competition among chains to obtain faster processors.

$PC = \{PC_1, PC_2, \dots, PC_m\}$ is the set of the processor's chains, $TPC = \{TPC_1, TPC_2, \dots, TPC_m\}$ is the corresponding set of the chain execution times, $PPC = \{PPC_1, PPC_2, \dots, PPC_m\}$ is the corresponding set of the chain probabilities, and

$$(TPC_i \leq TPC_j) \rightarrow PPC_i \leq PPC_j \quad (3)$$

c) Compute Fitness Values.

In this step, the fitness value of each country, which has been generated randomly based on OBC, will be computed according to its power metric corresponding to the execution time of the country's scheduling.

3) Initialize the Empire.

The algorithm selects the best set of countries as the imperialists, and divides the remaining countries as colonies among them. Each imperialist with its colonies form an empire.

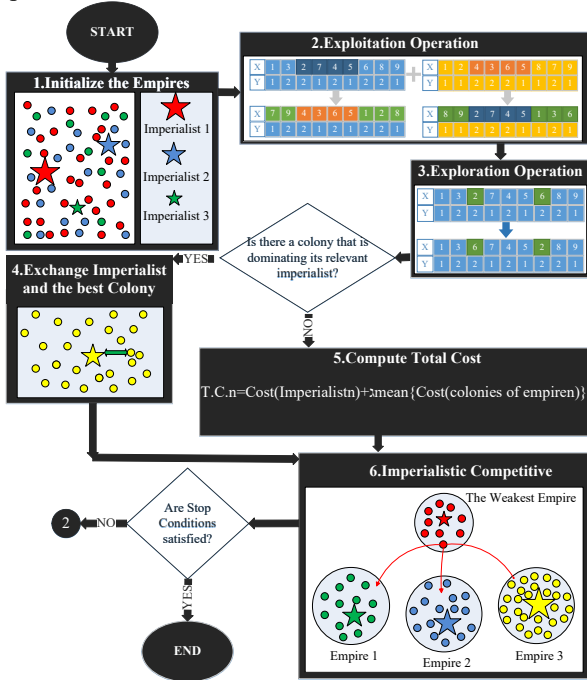


Figure 4. The ICGA flowchart

4) Exploitation and Exploration Operation.

In the exploitation phase, each colony can randomly pair with its imperialist or another colony in its empire, but cannot pair with any colonies from other empires. We use the order-based crossover (OX) function [14] to replace the task numbers between the paired colonies. As shown in Figure 4.2 (exploitation operation), OX selects two indices (e.g. 3 and 6 in their OBCs) of paired colonies and exchanges all the tasks between two indexes of paired colonies. Then the remaining tasks will be exchanged sequentially. Afterwards, the exploration operation selects several colonies randomly in

order to exchange two tasks of each colony. The indices of two tasks will be generated randomly.

5) Exchange Position and Compute the Total Cost.

After the exploitation and exploration, if a colony obtains a better fitness function value than its imperialist, the algorithm exchanges the colony's and its imperialist's positions. After the exchange operation, the algorithm computes the total cost of the empire based on Equation 4.

$$\begin{aligned} \text{Total Cost}_n &= \text{Cost}(\text{Imperialist}_n) \\ &+ \xi \cdot \text{Mean}\{\text{Cost}(\text{Colonies of Empire}_n)\} \end{aligned} \quad (4)$$

6) Imperialist Competition.

In this phase, all the imperialists try to achieve one (randomly selected) colony from the weakest imperialist. The algorithm searches among the empires and selects the weakest empire based on their total cost. The taken colony will be dedicated to the strongest empire.

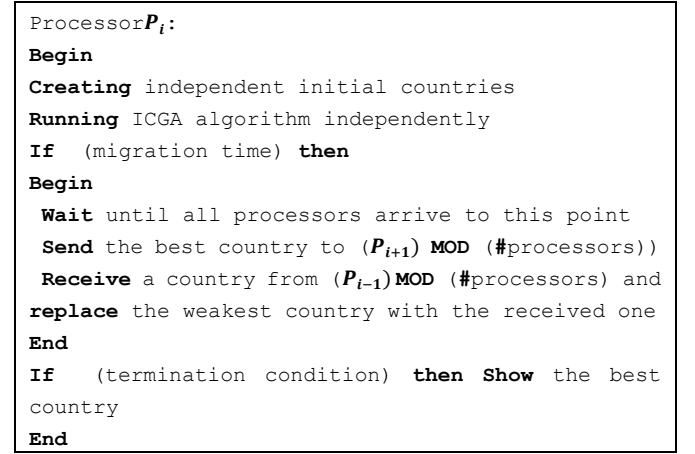


Figure 5. The Pseudocode of the Multi-Population ICGA

7) Migration Operation.

The migration operation is the key aspect of our method, where several processors are connected together using a ring topology and employing message passing protocol for communication. The ring topology has been selected because of its low communication cost and simplicity. Each processor is first initialized with a set of independent countries (the number of countries in each processor is the same) and running ICGA independently in each processor. Occasionally, the best country (colony) migrates from a processor P_i to the next processor P_{i+1} and replaces the worst country in P_{i+1} . Since we utilize the ring topology to connect processors together, and because the migration takes place in all processors synchronously, the number of countries in each processor remains the same through the migration operation. The migration strategy can affect the result as well. Generally, it is better to establish a balance between the migration rate and data communication. The chosen ring topology is utilized to reduce the migration rate and to decrease the distance of the migrations.

In MICGA, the pressure of selection increases when the number of countries gets higher, which helps to obtain more accurate results in a short time and converge to the results faster than the sequential ICGA. Therefore, it is highly beneficial to increase the number of countries.

V. EXPERIMENTAL RESULTS

In this section, four commonly explored real applications have been utilized to demonstrate the performance of the MICGA. The obtained results have been compared with the

other well-known methods that have been used for the same problems. The parallel ICGA has been implemented using both shared memory and message passing models. The message passing interface (MPI) has been utilized to parallelize and MPICH2 to run our algorithm.

In the multi-population ICGA, processors have been connected in a ring topology with different parameters. The proposed algorithm has been tested on an Intel Core i5-4570S, processor clocked at 2.90 GHz (64-bit) and with 24GB of memory. The best results of benchmarks have been obtained by 15 independent runs. The used parameters for solving the problems have been illustrated in TABLE I.

The proposed approach is tested on real applications such as Sobel filter, SUSAN filter, RASTA-PLP and JPEG encoder [18], [19] that are illustrated in Figure 7. Also, MICGA is compared with MPGA [13], PMC [29] and MPQGA [26] in the same conditions (e.g. the same platform and initial population) on all benchmarks. These methods are re-implemented accurately, and they have been tested by their results to obtain the best results.

TABLE I. THE PARAMETERS OF MICGA

Parameters	Values
Number of Countries	100
Number of Empire	5
Terminate Condition	20 Iterations
Number of Processors	5
Exploitation Rate	0.8
Exploration Rate	0.3
Migration Rate	1 Chromosome

Figure 8, and show the execution time of all methods on the benchmarks on six processors. MICGA improves (decreases) execution time by more than 46%, 8%, 15%, and 18% for the Sobel, SUSAN, RASTA-PLP, and JPEG encoder applications, respectively. Figure 6 shows the execution time values of the MICGA with different values of pp (the number of applications in each pack is pp) on the benchmarks for 100 applications on six processors. It can be seen that the adopted packing method improves the execution time.

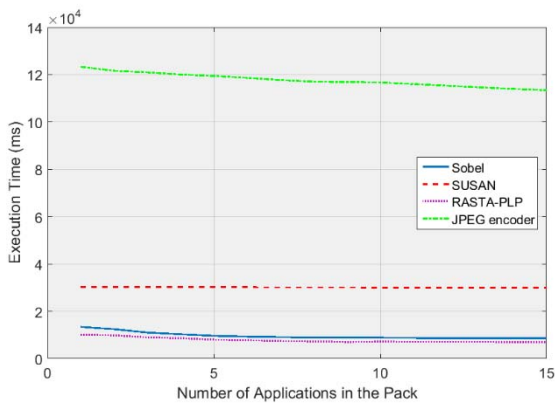


Figure 6. The execution time of MICGA on all benchmarks with different number of pp on six processors

Figure 10, Figure 11, Figure 12, and Figure 13 show the execution time of all methods on the benchmarks with different processors. The best and the worst possible case of execution time for 100 application on 60 processors on Sobel are 7006 and 14112, Susan are 11846 and 20236, RASTA-PLP are 5412 and 11082; and JPEG encoder are 79608 and 147362. Also, MICGA improves (decreases) execution time by more than 62.74%, 32.86%, 51.51%, and 37.13 % for the Sobel, SUSAN, RASTA-PLP, and JPEG encoder applications, respectively. These results show that MICGA has better effect in more complex applications.

TABLE II. THE STATISTICAL RESULTS WITH 6 PROCESSORS

		MICGA	MPGA	PMC	MPQGA
Sobel (ms)	Mean	7458.4	1217.0	1314.1	1415.5
	Best	7006	11465	12065	13510
	Worst	8456	13510	14865	14962
	STD	665.7	874.4	1144.7	714.7
	Median	7006	12065	13510	13510
SUSAN (ms)	Mean	12491	15478	17610	17740
	Best	12103	14860	17224	17465
	Worst	13246	17224	18245	18245
	STD	4982	833.7	437.5	351.1
	Median	12103	14860	17465	17465
RASTA-PLP (ms)	Mean	12510	15409	17384	17617
	Best	12103	14860	17224	17465
	Worst	13442	15986	17584	18012
	STD	537.0	479.2	161.5	195.66
	Median	12103	15670	17465	17584
JPEG-encoder (ms)	Mean	87597	112491	115573	115146
	Best	83440	108600	111006	111006
	Worst	108600	119688	119688	122865
	STD	8631.5	4473.1	3290.6	4765.6
	Median	83440	108600	115820	111006

TABLE III. THE SPEEDUP AND EFFICIENCY OF NOMES ON 6 PROCESSORS

	Serial MICGA Time (s)	MICGA Time (s)	Speedup	Efficiency
Sobel	45.32	11	4.12	0.824
SUSAN	75.6	18	4.20	0.840
RASTA-PLP	89.67	21	4.27	0.854
JPEG encoder	133.12	32	4.16	0.832

The stability and reliability of experimental results are other important factors for selecting the best method. It is a fact that heuristic and metaheuristic methods cannot achieve the best results in all runs, so the stability diagram is the best criteria to show the reliability of results. We run each real application 15 times to demonstrate how many times each method can find the best results. The stability diagrams of all mentioned methods are illustrated in Figure 14. The statistical results have been listed in TABLE II. TABLE II. demonstrates that our algorithm is more accurate with fewer errors than the existing other methods. Also, TABLE II. shows that MICGA has obtained the optimal result for Sobel. In TABLE III. , two critical parameters, speedup, and efficiency are also compared to both serial and parallel approaches. The speedup values show that the parallel method how many times is faster than serial method, and the efficiency values achieve from the portion of the speedup and the number of processors. The table shows that the efficiency of the proposed parallel implementation is outstanding, and it clearly outperforms the existing serial methods.

$$Speedup = ExecutionTime_{serial} / ExecutionTime_{parallel} \quad (5)$$

$$Efficiency = Speedup / \#Processors \quad (6)$$

Finally, based on the quality and reliability of MICGA results, and the efficiency of this method, as well as its usability to real applications, MICGA is truly an outstanding candidate for task graph scheduling.

VI. CONCLUSION

In this paper, we first introduced the new evolutionary computing method, named ICGA, a combination of the imperialist competitive algorithm (ICA) and the genetic algorithm (GA). Then, we presented a multi-population implementation of ICGA (MICGA) in order to improve the performance (execution time) and reliability (proximity of archived the best result) for task graph scheduling on heterogeneous platforms using the pipelined execution model. Experimental results revealed that MICGA is the best candidate for solving the task graph scheduling problem. It is faster and more reliable than the other state-of-the-art methods.

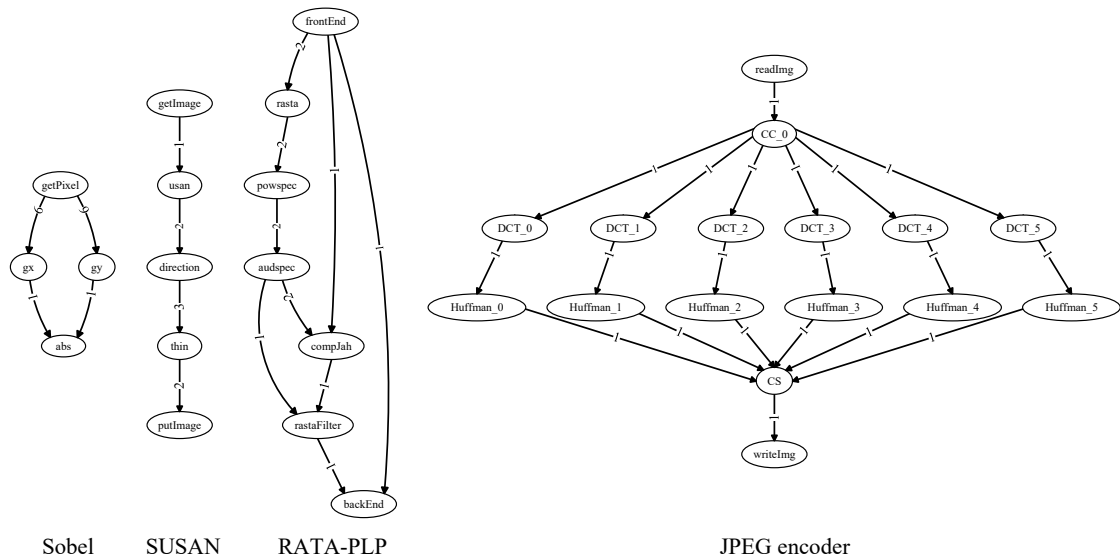


Figure 7. The benchmarks

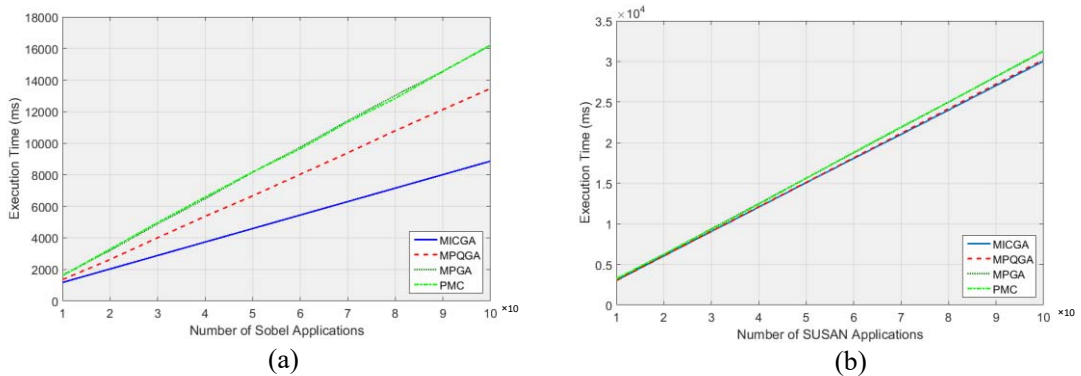


Figure 8. The execution time of all methods on different number of (a) Sobel and (b) Susan application on six processors

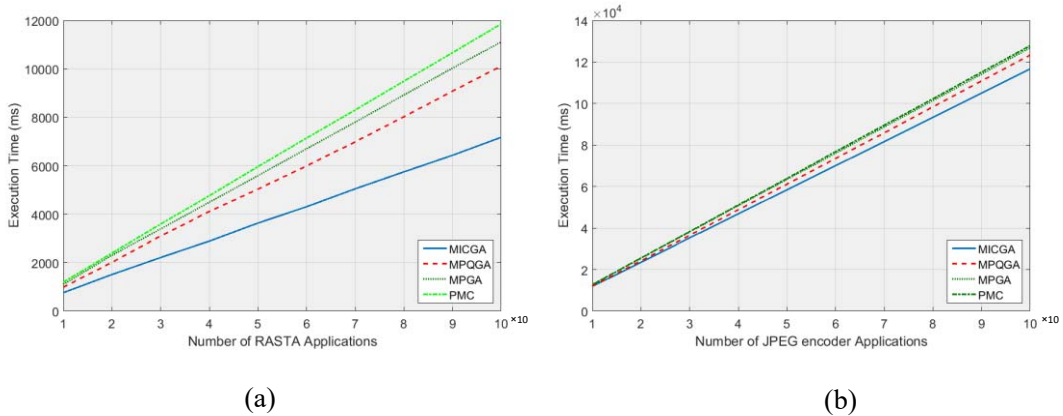


Figure 9. The execution time of all methods on different number of (a) Rasta and (b) JPEG encoder application on six processors

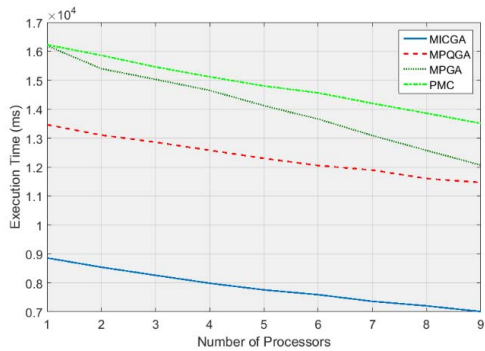


Figure 10. The execution time of all methods on 100 of Sobel application (pp=100)

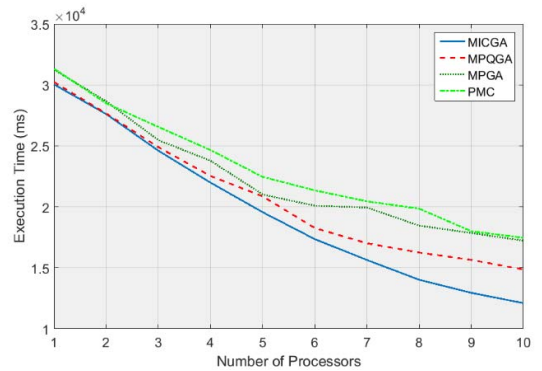


Figure 11. The execution time of all methods on 100 of SUSAN application (pp=100)

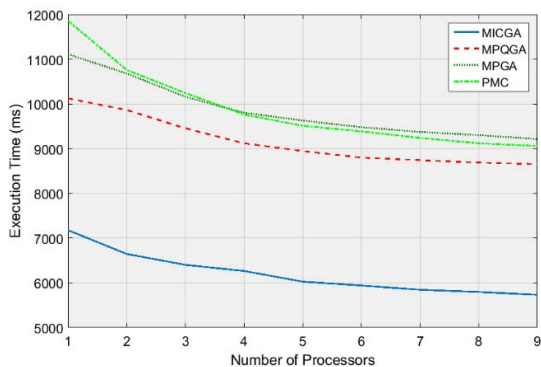


Figure 12. The execution time of all methods on 100 of RASTA-PLP application ($pp=100$)

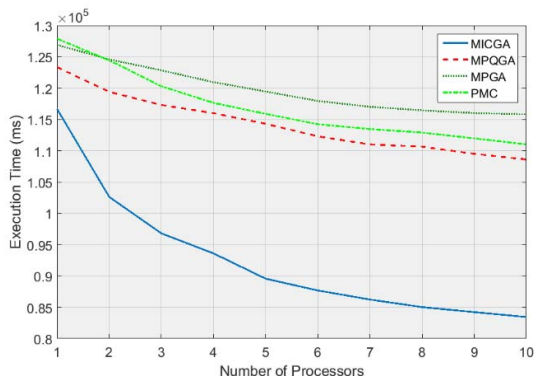


Figure 13. The execution time of all methods on 100 of JPEG-encoder application ($pp=100$)

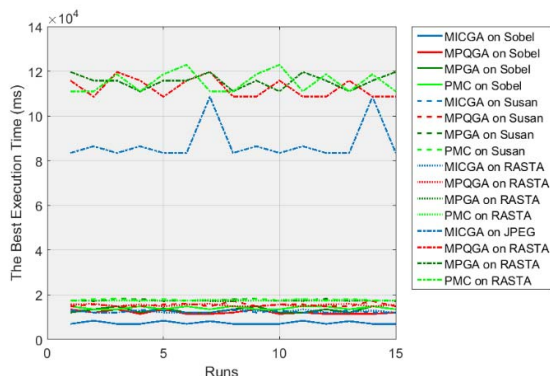


Figure 14. The stability diagram of MICGA, MPGA, PMC and MPQGA on Sobel, SUSAN, RASTA-PLP and JPEG encoder with 4 processors

REFERENCES

- [1] Adam TL, Chandy KM, Dicksoni JR. A comparison of list schedules for parallel processing systems. Communications of the ACM 1974;17(12):685–90.
- [2] Wu MY, Gajski DD. Hypertool: a programming aid for message-passing systems. IEEE Transactions on Parallel and Distributed Systems 1990;1(3):330–43.
- [3] Yang T, Gerasoulis A. DSC: scheduling parallel tasks on an unbounded number of processors. IEEE Transactions on Parallel and Distributed Systems 1994;5(9).
- [4] Correa RC, Ferreira A, Rebreyend P. Scheduling multiprocessor tasks with genetic algorithms. IEEE Transactions on Parallel and Distributed Systems 1999;10(8):825–37.
- [5] Thanalapati T, Dandamudi S. An efficient adaptive scheduling scheme for distributed memory multicomputers. IEEE Transactions on Parallel and Distributed Systems 2001;12(7):758–68.
- [6] Nissanke N, Leulseged A, Chillara S. Probabilistic performance analysis in multiprocessor scheduling. Journal of Computing and Control Engineering 2002;13(4):171–9.
- [7] Corbalan J, Martorell X, Labarta J. Performance-driven processor allocation. IEEE Transactions on Parallel and Distributed Systems 2005;16(7):599–611.
- [8] Hou ESH, Ansari N, Hong R. A genetic algorithm for multiprocessor scheduling. IEEE Transactions on Parallel and Distributed Systems 1994;5(2):113–20.
- [9] Hwang RK, Gen M. Multiprocessor scheduling using genetic algorithm with priority-based coding. Proceedings of IEEE conference on electronics, information and systems; 2004.
- [10] Wu AS, Yu H, Jin S, Lin K-C, Schiavone G. An incremental genetic algorithm approach to multiprocessor scheduling. IEEE Transactions on Parallel and Distributed Systems 2004;15(9):824–34.
- [11] Gen M, Cheng R. Genetic algorithm and engineering optimization. NewYork:Wiley; 2000.
- [12] Tsujimura Y, Gen M. Genetic algorithms for solving multiprocessor scheduling problems. In: Simulated evolution and learning. Heidelberg: Springer; 1995. p. 106–115.
- [13] R. Moradi and D. Dal, A Multi-Population Based Parallel Genetic Algorithm for Multiprocessor Task Scheduling with Communication Costs, 2016 IEEE Symposium on Computers and Communication (ISCC).
- [14] E. Atashpaz-Gargari, C. Lucas, Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition, in:IEEE Congress on Evolutionary Computation, 2007, pp. 46614667.
- [15] A. Majd, Sh. Lotfi and G. Sahebi, "Review on Parallel Evolutionary Computing and Introduce Three General Framework to Parallelize All EC Algorithms," 5th Conference on Information and Knowledge Technology (IKT), 2013.
- [16] A. Majd, Sh. Lotfi, G. Sahebi, M. Daneshdab and J. Plosila, "PICA: Multi-Population Implementation of Parallel Imperialist Competitive Algorithms," 24th Euromicro International Conferences on Parallel, Distributed and Network-Based Processing, PDP 2016.
- [17] A. Majd, M. Abdollahi, G. Sahebi, D. Abdollahi, M. Daneshdab, J. Plosila and H. Tenhunen, Multi-Population Parallel Imperialist Competitive Algorithm for Solving Systems of Nonlinear Equations, The 2016 International Conference on High Performance Computing & Simulation (HPCS 2016).
- [18] K. Rosvall and I. Sander, A constraint-based design space exploration framework for real-time applications on MPSoCs, Proceedings of the conference on Design, Automation & Test in Europe (DATE '14), 2014.
- [19] N. Khalilzad, K. Rosvall and I. Sander, A Modular Design Space Exploration Framework for Multiprocessor Real-Time Systems, Forum on specification and Design Languages (FDL'16), 2016.
- [20] T. Lei and S. Kumar, "A Two-Step Genetic Algorithm for Mapping Task Graphs to a Network on Chip Architecture," Proc. Euromicro Symp. Digital System Design (DSD 03), IEEE Press, 2003, pp. 180-187.
- [21] D. Wu, B. Al-Hashimi, and P. Eles, "Scheduling and Mapping of Conditional Task Graphs for the Synthesis of Low Power Embedded Systems," Proc. Design, Automation and Test in Europe (DATE 03), IEEE CS Press, 2003, pp. 90-95.
- [22] S. Murali and G. De Micheli, "Bandwidth-Constrained Mapping of Cores onto NoC Architectures," Proc. Design, Automation and Test in Europe (DATE 04), IEEE CS Press, 2004, pp. 896-901.
- [23] S. Manolache, P. Eles, and Z. Peng, "Fault and Energy-Aware Communication Mapping with Guaranteed Latency for Applications Implemented on NoC," Proc. 42nd Annual Design Automation Conf. (DAC 05), ACM Press, 2005, pp. 266-269.
- [24] J. Hu and R. Marculescu, "Energy- and Performance- Aware Mapping for Regular NoC Architectures," IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol. 24, no. 4, 2005, pp. 551-562.
- [25] C. Marcon et al., "Comparison of NoC Mapping Algorithms Targeting Low Energy Consumption," IET Computers & Digital Techniques, vol. 2, no. 6, 2008, pp. 471-482.
- [26] Y. Xu, K. Li, J. Hu and K. li, "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," Information Sciences, Vol.270, pp. 255-287,Elsevier,2014.
- [27] E. L. s. Carvalho, N. L. V. Calazans and F. G. Moraes, "Dynamic Task Mapping for MPSoCs," IEEE Design and Test of Computers, 2010.
- [28] H. Ishibuchi, N. Akedo and Y. Nojima, "Behavior of Multiobjective Evolutionary Algorithms on Many-Objective Knapsack Problems," IEEE Transactions on Evolutionary Computation, Vol. 19, pp. 264 – 283,2015
- [29] R. Hwang, M. Gen and H. Katayama, " A comparison of multiprocessor task scheduling algorithms with communication costs" Computers & Operations Research, Vol. 35, pp. 976 – 993, ELSEVIER, 2008.

Paper VIII

NOMeS: Near-Optimal Metaheuristic Scheduling for MPSoCs

A. Majd, M. Daneshtalab, J. Plosila, N. Khalilzad, G. Sahebi, and E. Troubitsyna

NOMeS: Near-Optimal Metaheuristic Scheduling for MPSoCs

Amin Majd¹, Masoud Daneshtalab², Juha Plosila⁴, Nima Khalilzad³, Golnaz Sahebi⁴, and Elena Troubitsyna¹

¹Åbo Akademi University, Finland,

²Mälardalen University, Sweden

³KTH Royal Institute of Technology

⁴University of Turku

{amajd, etroubit}@abo.fi, masoudd@mdh.se, nima.khalilzad@kth.se, {golsah, juplos}@utu.fi

Abstract — The task scheduling problem for Multiprocessor System-on-Chips (MPSoC), which plays a vital role in performance, is an NP-hard problem. Exploring the whole search space in order to find the optimal solution is not time efficient, thus metaheuristics are mostly used to find a near-optimal solution in a reasonable amount of time. We propose a novel metaheuristic method for near-optimal scheduling that can provide performance guarantees for multiple applications implemented on a shared platform. Applications are represented as directed acyclic task graphs (DAG) and are executed on an MPSoC platform with given communication costs. We introduce a novel multi-population method inspired by both genetic and imperialist competitive algorithms. It is specialized for the scheduling problem with the goal to improve the convergence policy and selection pressure. The potential of the approach is demonstrated by experiments using a Sobel filter, a SUSAN filter, RASTA-PLP and JPEG encoder as real-world case studies.

Keywords— parallel imperialist competitive algorithm (PICA); multi-population technique; evolutionary computing (EC), task graph scheduling, multi-objective optimization;

I. INTRODUCTION

Parallel processing in contemporary Multiprocessor System-on-Chips, or MPSoCs, in a vast variety of applications, is the result of many breakthroughs over the last two decades. The development of embedded MPSoCs has led to their use in many application domains such as health monitoring, video and audio processing, and autonomous vehicles to mention just a few. The data and the processing tasks of these applications are distributed on the available multiple cores. Performance of such parallel systems can be optimized by employing efficient task partitioning and scheduling strategies.

In this work, we consider a deterministic model for the task scheduling problem, where the execution times of tasks and the data communication times between tasks are known. The scheduling problem based on a directed acyclic task graph (DAG) that represents the precedence relations of the tasks is known to be an NP-hard problem. Many heuristic methods for task scheduling have been proposed [1], and [2]. In general, precedence constraints between tasks can be non-uniform, but we assume here, for simplicity, that the MPSoC platform is uniform (a homogeneous multiprocessor system) and non-preemptive (each processor completes the current task before starting the execution of the next task).

Recently, evolutionary approaches have been developed to solve the scheduling problem. For example, a genetic algorithm (GA) based approach can better locate a near optimal solution than a list schedule in most cases [3-5]. Only a few of these approaches have considered a task graph with a communication cost. A number of parallel GA based methods have been proposed for solving the task graph scheduling problem in [8]. However, in all of them, there is a fundamental problem that stems from the relatively inefficient and unreliable convergence strategy of a GA, originating from the selection operation of a GA. In this paper, we tackle this problem by proposing a combination of a GA and the Imperialist Competitive Algorithm

(ICA) [9]. Moreover, we explicitly consider the communication delays between processors. When two communicating tasks are mapped onto the same processor we assume that the communication delay is zero. However, when they are mapped onto different processors a finite communication delay is assumed and modeled.

We propose the concept of an order-based country (OBC) as an extension of the order-based coding method. This coding specifies the order of tasks and the selected processor to run each task. The order-based encoding [6] provides means to efficiently deal with precedence constraints. To improve the outcome of the optimization process, we present a new multi-population method, called NoMeS, which takes advantage of both ICA and GA. This combination enhances the convergence policy and selection pressure, by keeping all countries in all iterations and avoiding the use of a general selection operation such as roulette wheel and tournament schemes. To the best of our knowledge, this is the first effort using a combined multi-population method to reach an optimal scheduling in MPSoCs.

The paper is organized as follows. In Section II, definitions and important parameters in the task scheduling problem are presented. Section III discusses the related work. In Section IV, the proposed parallel algorithm for the scheduling problem (NoMeS) is presented. Section V presents the experimental results and demonstrates the efficiency of the proposed algorithm. We end with concluding remarks in Section VI.

II. BACKGROUND

We assume that a set of N applications consists of n tasks running on a multiprocessor with m processors. In the following, we present the details of the assumed system model.

1) Hardware platform model:

We assume a homogeneous MPSoC is a set of processing elements (PEs) $\{PE_1, \dots, PE_m\}$ interacting through a communication network. For our work, a Network-on-Chip (NoC) based homogeneous MPSoC model is selected, in which each PE manages execution of one task at a time. PEs can support either software or hardware task execution. Software tasks execute in instruction set processors (ISPs), and hardware tasks in reconfigurable logic (RL) or dedicated IP blocks [19]. Among the available PEs, one of the processors is selected as a manager processor (MP). The MP is responsible for resource control, configuration control, and task manipulation operations, namely: binding, scheduling, and migration [19]. The MP runs each application's initial task. When a running task needs to communicate with another task not yet mapped, the latter is loaded into a PE from the task memory.

2) Application model:

We assume that each application TG_a is modeled as a weighted DAG. In this DAG, each node represents a task that has a finite execution time, and each edge illustrates the communication cost between two connected tasks. The edges also represent the precedence relations. Let TG_a be a weighted DAG defined as a 4-tuple $TG_a = (V_a, E_a, W_a, C_a)$, where $V_a =$

$\{T_1^a, T_2^a, \dots, T_n^a\}$ is the set of vertices (tasks), $E_a = \{e_{ij}^a = (T_i^a, T_j^a)\}$ is the set of communication (dependency) edges, W_a is the set of task weights (execution times), C_a is the set of edge weights (communication times), $w_i^a \in W_a$ is the execution time of task $T_i^a \in V_a$, $c_{i,j}^a \in C_a$ is the communication cost (delay) on the edge of $e_{i,j}^a \in E_a$, that is assumed to be zero if T_i^a and T_j^a are bound to the same processor. We assume a simple linear model between the execution times and PE frequencies, e.g., if PE_1 runs two times faster than PE_2 , then task T_i^a also runs two times faster on PE_1 than on PE_2 .

The execution time of an application TG_a is defined as the difference between the completion time of its last task and the start time of its first task. When constructing a schedule for execution of the applications, it is of great importance to consider both task execution times and the communication times. Since considering the communication times significantly increases the number of schedules to be explored, most of the previous work only consider the task execution times. However, when running tasks with precedence relations on different processing elements, the communication overhead is not negligible and it has to be considered in the model. To this end, we explicitly model the communication delays.

3) Problem statement:

Our task scheduling method, NoMeS, takes a set of task graphs $\{TG_1, \dots, TG_N\}$ and a set of processing elements $\{PE_1, \dots, PE_m\}$ as its inputs. The output of NoMeS is a set of task bindings, i.e. task to processor mappings and static schedules, i.e. the order and timing of task executions for each application task graph. The overall goal of the scheduling is to minimize the execution times (also known as the schedule length, latency or makespan) of all applications while preserving the precedence constraints. Let f_a represent the execution time of an application TG_a . We present the following mathematical formulation as our optimization problem:

$$\begin{aligned} & \min f_1, \dots, f_N, \\ & \text{s. t. } t_i^a - w_i^a - e_{ji}^a \geq t_j^a, \quad a \in [1, \dots, N], \forall e_{ij}^a \in E_a, \end{aligned}$$

where t_i^a is the completion time of T_i^a . Note that the above formulation is a multi-objective optimization problem as we want to minimize the execution times of all applications. The optimization is subject to the precedence relations imposed by the application graphs.

III. RELATED WORK

Here, we first explore more traditional list scheduling heuristics that have considered communication costs.

The basic idea is to make an ordered list of nodes by assigning them orders, and then to repeatedly execute the following two steps until a valid schedule is obtained: (1) Select from the list the node with the highest order for scheduling. (2) Select a processor to accommodate this node.

In realistic cases, scheduling needs to exploit parallelism by identifying the task graph structure and take into consideration task granularity, arbitrary computation, and communication costs.

In [1], the modified critical path algorithm (MCP) is proposed, based on the latest possible start time of a node. A node's latest possible start time is determined via the as-late-as-possible (ALAP) binding by traversing the task graph upward from the exit nodes to the entry nodes while pulling the nodes' start times downwards as much as possible. The latest possible start time of the node itself is followed by a decreasing order of the latest possible start times of its successor nodes. Furthermore, in [1], the dominant sequence clustering algorithm (DSC) is presented. It is based on the dominant sequence, which is essentially the critical path of the partially scheduled task

graph at each step. At each step, DSC checks whether the highest CP (the critical path of task graph) node is a ready node. If so, DSC schedules it to a processor allowing the minimum start time. Such a minimum start time may be achieved by rescheduling some of the node's predecessors to the same processor. If the highest CP node is not a ready node, DSC does not select it for scheduling. Instead, it chooses the highest node which lies on a path reaching the CP for scheduling. Moreover, also in [1], the mobility directed algorithm (MD) is presented. MD selects a node at each step based on relative mobility which is defined as the difference between a node's earliest start time and latest start time. Similar to the ALAP binding, the earliest possible start time is assigned to each node via the as-soon-as-possible (ASAP) binding. This is performed by traversing the task graph downward from the entry nodes to the exit nodes while pulling the nodes upward as much as possible. Moreover, relative mobility is obtained by dividing the mobility with the node's computation cost. Basically, a node with zero mobility is a node on the CP. At each step, MD schedules the node with the smallest mobility to the first processor having a large enough time to accommodate the node without considering the minimization of the node's start time. After a node has been scheduled, the relative mobility values of the remaining nodes are updated.

In [8], a multi-population implementation of the GA method (MPGA) is presented which outperforms deterministic and nondeterministic methods described in [2-3]. In [20], a new encoding mechanism with a multi-functional chromosome is presented, using a priority representation that is called priority-based multi-chromosome (PMC). PMC can efficiently represent a task schedule and assign tasks to processors. It is another metaheuristic method that uses a GA to achieve near-optimal scheduling of tasks.

Research on static mapping methods includes the work of Lei et al., who proposed a genetic mapping algorithm to optimize application execution time [12]. In their work, graphs represent applications and the target architecture is a NoC. Wu, et al. also investigated genetic mapping algorithms [13]. By combining dynamic voltage scaling techniques with mapping, they achieved 51% savings in energy consumption. Murali et al. explored mappings for more than one application in NoC design, using the tabu search (TS) algorithm [14]. Manolache, et al. investigated task mapping in NoCs, trying to guarantee packet latency [15]. For this purpose, both the task-mapping algorithm (TS) and the routing algorithm are defined at design time. Hu et al. presented a branch-and-bound algorithm to map a set of IP cores (IPs) onto a NoC with bandwidth reservation [16]. Their results show energy savings of 51.7% in the communication architecture. Marcon et al. investigated how to map modules into a NoC, targeting low energy consumption [17]. They compared several algorithms, using a model that characterizes applications by their inter-task communication volume. Xu et al. In [18] presented a task scheduling scheme on heterogeneous computing systems using a multiple priority queues genetic algorithm (MPQGA). Their experimental results for large-sized problems for a large set of randomly generated graphs as well as graphs of real-world problems with various characteristics showed that the proposed MPQGA algorithm outperformed two non-evolutionary heuristics and a random search method in terms of schedule quality.

IV. NOMES

In this work, we combine the GA and ICA methods and customize this combined approach for the scheduling problem. The GA operations are basically suitable for scheduling but a GA's convergence strategy to find more reliable and accurate results is not efficient [10], because the selection operation in a GA misses some chromosomes that have a probability to

become potential genes for obtaining the best results. In the ICA, where we use countries as the population, all the countries will be available in all iterations; they may only move to other places in the search space. However, the selection operation highly depends on random functions which can easily converge to a local optimum. Therefore, we will use an efficient convergence strategy (the ICA approach) in order to improve the reliability of results.

All the scheduled tasks in a DAG should satisfy the precedence relations so that we use an order based coding mechanism suited for multiprocessor scheduling, which should fulfill the following rules:

1. All the predecessors of a task must have completed their execution before initiating the execution of the task.
2. In a DAG, all the tasks must be executed at least once. Such a representation eliminates the need to consider the precedence relations between the computational tasks.

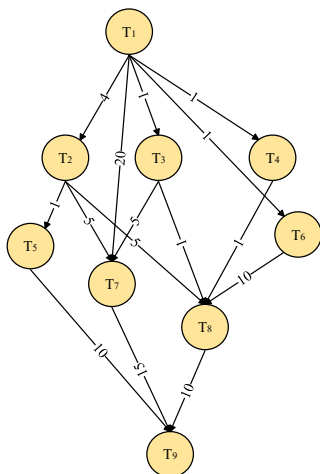


Figure 1. DAG Task graph

A. Order-based country (OBC)

There are several approaches [3, 5, 7] using GA for the multiprocessor scheduling problem, but they are suffering from inefficient coding methods. How to encode a solution of the problem into a chromosome is a key issue here. It has been investigated from two different angles which are: 1) mapping characters from the genotype space to the phenotype space when countries are decoded into solutions [1]; and 2) metamorphosis properties when countries are manipulated by genetic operators [9].

The two main challenges with respect to the encoding problem are the following: 1) there is a need for storing a huge number of chromosomes for each schedule; and 2) in the case of a large number of tasks, the exploration and exploitation operations will become very complex due to the precedence relations among tasks.

To overcome these difficulties we introduce the concept of an order based country (OBC) that strings the present task order of task nodes with the corresponding processors simultaneously. For example, Figure 2 shows an OBC that represents nine tasks along with two processors for the DAG shown in Figure 1. For example, the third column shows that the third scheduling is for the task number 2 which should be run on the processor 2.

B. Order-based encoding/decoding

Establishing an encoding which can treat the task precedence constraints efficiently is a critical step. The proposed encoding method is based on the task order and is therefore a viable solution. The following procedure explains the generation of the initial OBC:

First, we input the order-based task numbers ($1 \leq X[i] \leq \# \text{Task}$) randomly in the second row, independently of the

processor number. Then, the algorithm fills in the third row with random numbers $Y[i]$ ($1 \leq Y[i] \leq \#p$).

Priority	1	2	3	4	5	6	7	8	9
X (Task Number)	1	3	2	7	4	5	6	8	9
Y (Processor Index)	1	2	2	1	2	1	2	2	1

Figure 2. An order based country (OBC)

C. NoMeS Operation:

In the initial population, the algorithm generates several countries such as the one presented in Figure 4.1. The major part of this algorithm is imperialistic, and it causes the colonies to converge to the global minimum. After the initial step, algorithm sorts the countries. Then, the best countries (whose fitness values are better than those of the others) will be selected to become imperialists, and the remaining countries form the colonies of these imperialists. These colonies start moving toward their relevant imperialists after all colonies have been divided among the imperialists [9]. The next step computes the power of each imperialist and the imperialistic competition step follows. The weakest imperialist loses its weakest colony, and the selected imperialist obtains this colony. The above steps are then repeated until reaching a termination condition. The termination condition can be defined in different ways. For example, the NoMeS could be set to stop when one imperialist has all colonies as its members.

Like with other ECs, when solving a problem with a large search space, we need a large initial population to obtain a more accurate and reliable result. We can also prevent convergence to local optima by using a multi-population method, where each processor runs serial NoMeS independently. If one processor converges to a local optimum, all the other processors can continue their work on other parts of the search space.

D. Multi-population NoMeS

To implement the NoMeS algorithm, we use a selective local search strategy which runs NoMeS on several processors connected in a ring topology, using the message passing method for communication. In each processor, we first initiate independent countries and run the serial NoMeS independently. Regularly, the best country is migrated from processor P_i to P_{i+1} replacing the worse country in P_{i+1} . The number of countries in each processor remains equal even when countries migrate to other processors, because all migrations between processors are done synchronously.

In the multi-population NoMeS, we increase the number of countries to get a higher selection pressure [10]; this facilitates obtaining more accurate results in a short time and much faster convergence to results, compared with the sequential NoMeS algorithm. Indeed, in EC methods, when the diversity of population is high enough, the rate of convergence improves.

1) Initialize the Empires

In this step, the fitness value of each country (OBC), which has been generated randomly, is computed based on its power metric which corresponds to the execution time of the country's schedule. Then the algorithm selects the best set of countries as the imperialists, and divides the remaining countries as colonies among them. Each imperialist with its colonies constitute an empire.

2) Exploitation and Exploration Operation

In the exploitation phase, each colony can randomly pair with its imperialist or another colony in its empire, but cannot pair with any colonies from other empires. We use the order-based crossover (OCX) function [9] to replace the task numbers between the paired colonies. As shown in Figure 3 (exploitation operation), OCX selects two indices (e.g. 3 and 6 in their OBCs)

of paired colonies and exchanges all the tasks between two indexes of paired colonies. Then the remaining tasks are exchanged sequentially. Afterwards, the exploration operation selects several colonies randomly in order to exchange two tasks of each colony. The indices of the two tasks are randomly determined.

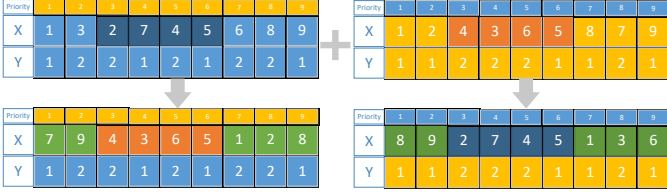


Figure 3. The exploitation operation

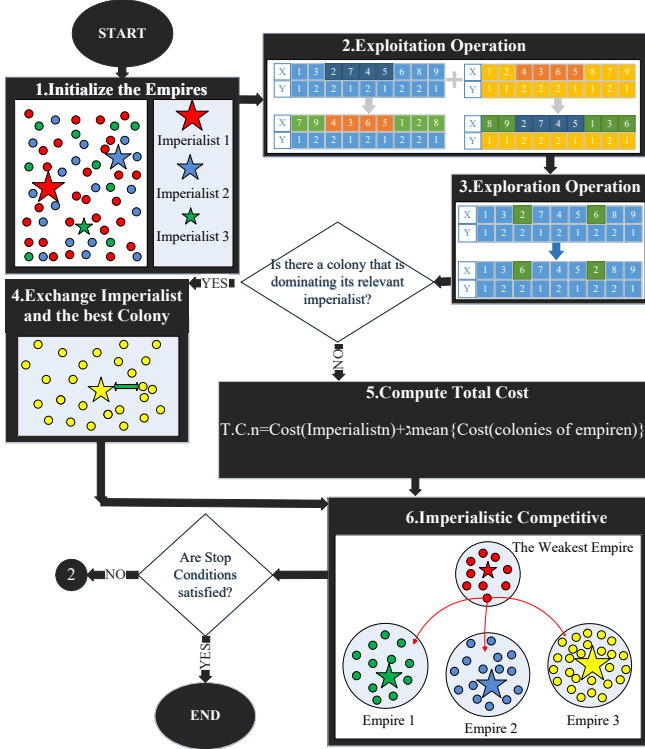


Figure 4. The NoMeS flowchart

3) Exchange Position and Compute the Total Cost

After the exploitation and exploration, if a colony obtains a better fitness value than its imperialist, the algorithm exchanges the positions of this colony and the imperialist, i.e. the colony becomes an imperialist and the imperialist becomes a colony. After the exchange operation, the algorithm computes the total cost (power) of the empire based on Equation 1.

$$\text{Total Cost}_n = \text{Cost}(\text{Imperialist}_n) + \xi \cdot \text{Mean}\{\text{Cost}(\text{Colonies of Empire}_n)\} \quad (1)$$

4) Imperialistic Competition

In this phase, all the imperialists try to achieve one (randomly selected) colony from the weakest imperialist. The algorithm searches among the empires and selects the weakest empire based on their total cost. The taken colony will be given to the strongest empire.

5) Migration Operation

The migration operation is the key aspect of our method. We consider a system, where several processors are connected in a ring topology using message passing protocol for communication. The ring topology has been selected because of its low communication cost and simplicity. Each processor is first initialized with a set of independent countries (the number of countries in each processor is the same) and running serial NoMeS independently on each processor. Now and then, in some time intervals, the best country (colony) migrates from processor P_i to the next processor P_{i+1} and replaces the worst

country in P_{i+1} . Since we utilize the ring topology to connect processors together, and because the migration takes place in all processors synchronously, the numbers of countries in any two processors are equal at any given time. The migration strategy can affect the result as well. Generally, it is better to establish a balance between the migration rate and data communication between processors in multi-population implementations. The chosen simple ring topology is utilized to reduce the migration rate and to decrease the distance of the migrations.

V. EXPERIMENTAL RESULTS

In this section, four well-known real applications have been employed to demonstrate the performance of the proposed NoMeS algorithm. The obtained results have been compared with those of the other optimization methods that have used the same applications. The parallel NoMeS has been implemented based on both shared memory and message passing platforms. The message passing interface (MPI) has been utilized to parallelize our algorithm and MPICH2 to run the algorithm.

In the multi-population NoMeS model, five processors (the processors that are used to run the NoMeS optimization algorithm) have been connected in a ring topology, considering different numbers of processors for executing the selected four applications on a NoC based MPSoC platform. The proposed algorithm has been tested on an Intel Core i5-4570S desktop computer clocked at 2.90 GHz (64-bit) with 24GB of memory. The results of the considered benchmark applications have been obtained by 20 independent runs. The involved parameters for solving the problems are specified in TABLE I.

The four real applications, used to test the proposed approach, are: Sobel filter, SUSAN filter, RASTA-PLP and JPEG encoder [11]. They are illustrated in Figure 9. The benchmarks Figure 9. Also, NoMeS is compared with MPGA [8], PMC [20] and MPQGA [18] in the same conditions (e.g. the same platform and initial population) on all benchmarks.

Figure 5, Figure 6, Figure 7 and Figure 8 show the execution times of the benchmarks for all the considered optimization methods with different numbers of processors (the processors that run the scheduled tasks on a NoC based MPSoC platform). NoMeS improves (decreases) execution time by more than 29.5%, 68.1%, 47.4%, and 10.1% for the Sobel, SUSAN, RASTA-PLP, and JPEG encoder applications, respectively. These results show that NoMeS is more effective in more complex applications. TABLE II. shows the execution times of the benchmarks for all the methods. The execution time of each predicted scheduling clearly demonstrates that NoMeS is more fit and efficient than the other methods for the DAG scheduling problem.

The stability and reliability of the experimental results are the other important factors for selecting the best method. It is a fact that heuristic and metaheuristic methods cannot achieve the best results in all runs, so the stability diagram is the best criterion to show the reliability of the results. We run each optimization method for each benchmark application 20 times to demonstrate how many times each method can find the best results. The stability diagrams of the considered methods are illustrated in Figure 10. The statistical results are listed in TABLE III. The table demonstrates that our algorithm is more accurate with fewer errors than the existing other methods. Also, this table shows that NoMeS has obtained the best results for Susan, RASTA-PLP and JPEG encoder. In TABLE IV. , two critical parameters, namely speedup and efficiency (Equations 2 and 3), are also compared, considering both serial and parallel (multi-population) NoMeS approaches. The speedup values show how many times faster the parallel method is compared with the serial method, and the efficiency values indicate the average speedup per processor. The table shows that the

efficiency of the proposed parallel implementation is outstanding, and it clearly outperforms the serial version of the NoMeS method.

$$Speedup = Execution\ Time_{Serial} / Execution\ Time_{Parallel} \quad (2)$$

$$Efficiency = Speedup / \#Processors \quad (3)$$

The convergence diagrams of the NoMeS method on all benchmarks are presented in Figure 11. Based on the quality and reliability of the results, the efficiency of this method, as well as its usability for real applications, NoMeS is truly a prominent candidate for task graph scheduling.

VI. CONCLUSION

In this paper, we first introduced a new evolutionary computing method, called NoMeS, a combination of the imperialist competitive algorithm (ICA) and a genetic algorithm (GA). Then, we presented a multi-population implementation of NoMeS, in order to improve its performance (execution time) and reliability (proximity of achieved results), for task graph scheduling which is a multi-objective optimization problem. Experimental results revealed that NoMeS is the best candidate for solving the task graph scheduling problem, being more reliable and providing significantly better solutions than the other considered state-of-the-art methods.

TABLE I. THE PARAMETERS OF NOMES

Parameters	Values
Number of Countries	100
Number of Empires	5
Termination Condition	20 Iterations
Number of Processors	5
Exploitation Rate	0.8
Exploration Rate	0.3
Migration Rate	1 Chromosome

TABLE II. EXECUTION TIMES OF THE BENCHMARKS FOR THE CONSIDERED OPTIMIZATION METHODS

	#p	NoMeS (ns)	MPGA (ns)	PMC (ns)	MPQGA (ns)
Sobel	2	10520	11440	11440	10946
SUSAN	2	41540	49120	49120	47218
RASTA	2	20240	27040	28320	24126
JPEG	2	109400	120200	117280	116320
JPEG	4	108680	120200	117280	114806
JPEG	6	95240	108680	108680	106354
JPEG	8	95240	104630	105624	104658
JPEG	10	95240	101460	101460	99468

TABLE III. THE STATISTICAL RESULTS WITH 2 PROCESSORS

		NoMeS	MPGA	PMC	MPQGA
Sobel (Cycle)	Mean	530.6	577.8	583	571
	Best	526	572	572	556
	Worst	572	612	612	598
	STD	14.1585	12.4799	15.728	17.281
	Median	526	572	572	572
SUSAN (Cycle)	Mean	2134.9	2484.3	2484.9	2372.8
	Best	2077	2456	2456	2246
	Worst	2478	2532	2532	2532
	STD	141.5953	33.29	29.4974	131.502
	Median	2077	2478	2478	2362
RASTA-PLP (Cycle)	Mean	1012	1414.8	1470.2	1343.4
	Best	1012	1352	1416	1246
	Worst	1012	1612	1612	1564
	STD	0	82.8960	66.6488	128.5565
	Median	1012	1384	1464	1246
JPEG-encoder (Cycle)	Mean	5475.8	6233.3	6153	5948.8
	Best	5370	6010	5864	5812
	Worst	5840	6580	6580	6328
	STD	203.6293	259.4247	294.7416	186.3369
	Median	5370	6010	6010	5812

TABLE IV. THE SPEEDUP AND EFFICIENCY OF NOMES ON 5 PROCESSORS

	Serial NoMeS Time (s)	NoMeS Time (s)	Speedup	Efficiency
Sobel	45.32	11	4.12	0.824
SUSAN	75.6	18	4.20	0.840
RASTA-PLP	89.67	21	4.27	0.854
JPEG encoder	133.12	32	4.16	0.832

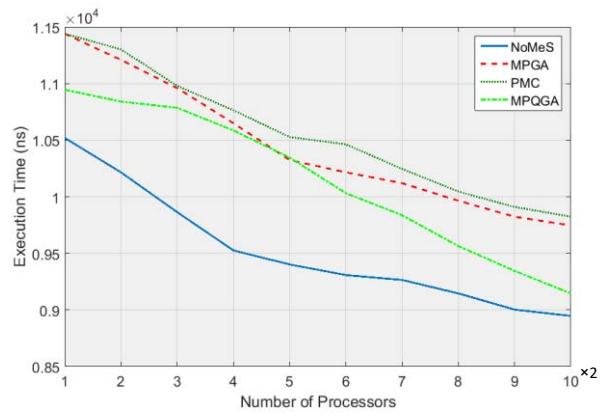


Figure 5. The Execution Time of Sobel filter with different numbers of processors

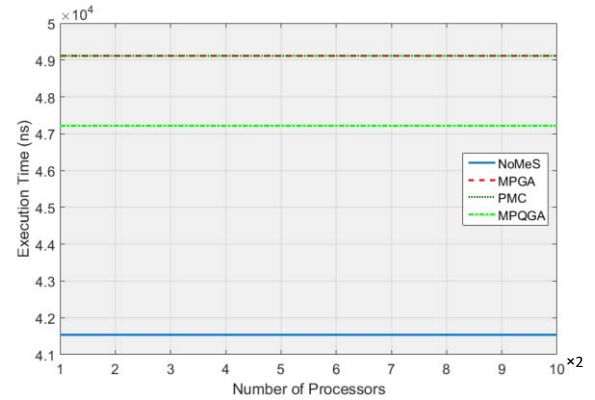


Figure 6. The Execution Time of Susan filter with different numbers of processors

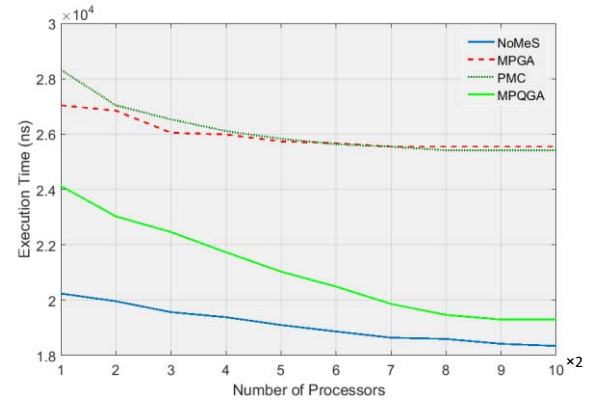


Figure 7. The Execution Time of RASTA-PLP filter with different numbers of processors

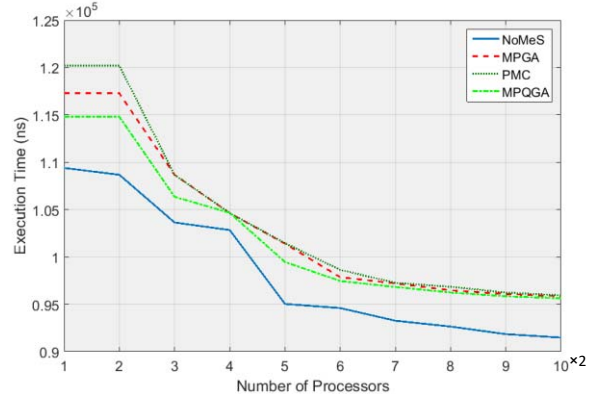


Figure 8. The Execution Time of JPEG-encoder with different numbers of processors

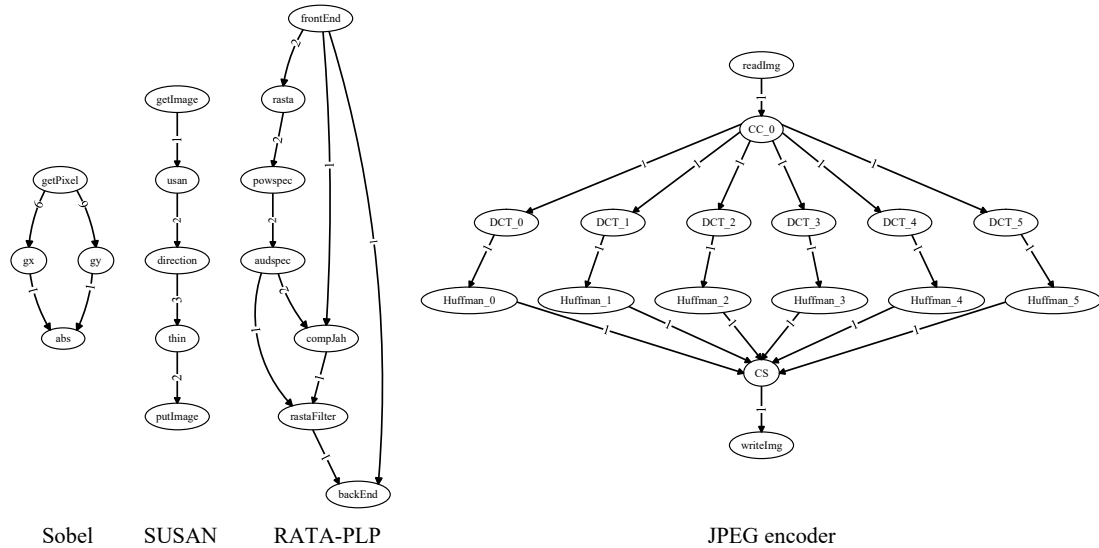


Figure 9. The benchmarks

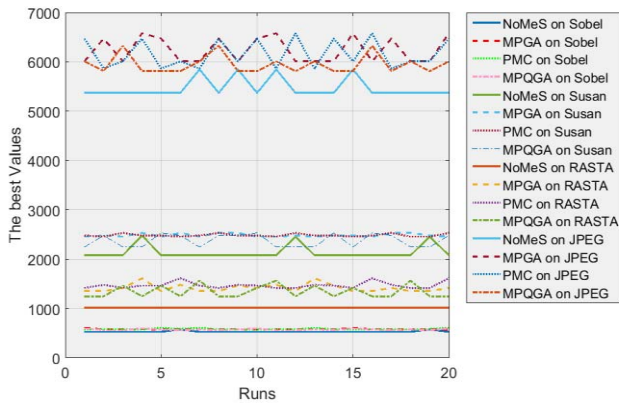


Figure 10. The stability diagram of NoMeS, MPGA, PMC and MPQGA on Sobel, SUSAN, RASTA-PLP and JPEG encoder with 2 processors.

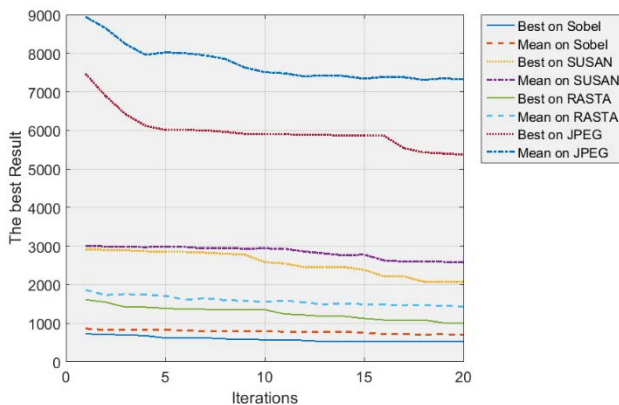


Figure 11. The convergence diagram of NoMeS on Sobel, SUSAN, RASTA-PLP and JPEG encoder with 2 processors.

REFERENCES

- [1] Adam TL, Chandy KM, Dicksoni JR. A comparison of list schedules for parallel processing systems. Communications of the ACM 1974;17(12):685–90.
- [2] Wu MY, Gajski DD. Hypertool: a programming aid for message-passing systems. IEEE Transactions on Parallel and Distributed Systems 1990;1(3):330–43.
- [3] Hou ESH, Ansari N, Hong R. A genetic algorithm for multiprocessor scheduling. IEEE Transactions on Parallel and Distributed Systems 1994;5(2):113–20.
- [4] Hwang RK, Gen M. Multiprocessor scheduling using genetic algorithm with priority-based coding. Proceedings of IEEJ conference on electronics, information and systems; 2004.
- [5] Wu AS, Yu H, Jin S, Lin K-C, Schiavone G. An incremental genetic algorithm approach to multiprocessor scheduling. IEEE Transactions on Parallel and Distributed Systems 2004;15(9):824–34.
- [6] Gen M, Cheng R. Genetic algorithm and engineering optimization. NewYork:Wiley; 2000.
- [7] Tsujimura Y, Gen M. Genetic algorithms for solving multiprocessor scheduling problems. In: Simulated evolution and learning. Heidelberg: Springer; 1995. p. 106–115.
- [8] R. Moradi and D. Dal, A Multi-Population Based Parallel Genetic Algorithm for Multiprocessor Task Scheduling with Communication Costs, 2016 IEEE Symposium on Computers and Communication (ISCC).
- [9] E. Atashpaz-Gargari, C. Lucas, Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition, in: IEEE Congress on Evolutionary Computation, 2007, pp. 4661-4667.
- [10] A. Majd, M. Abdollahi, G. Sahebi, D. Abdollahi, M. Daneshalab, J. Plosila and H. Tenhunen, Multi-Population Parallel Imperialist Competitive Algorithm for Solving Systems of Nonlinear Equations, The 2016 International Conference on High Performance Computing & Simulation (HPCS 2016).
- [11] N. Khalilzad, K. Rosvall and I. Sander, A Modular Design Space Exploration Framework for Multiprocessor Real-Time Systems, Forum on specification and Design Languages (FDL'16), 2016.
- [12] T. Lei and S. Kumar, "A Two-Step Genetic Algorithm for Mapping Task Graphs to a Network on Chip Architecture," Proc. Euromicro Symp. Digital System Design (DSD 03), IEEE Press, 2003, pp. 180-187.
- [13] D. Wu, B. Al-Hashimi, and P. Eles, "Scheduling and Mapping of Conditional Task Graphs for the Synthesis of Low Power Embedded Systems," Proc. Design, Automation and Test in Europe (DATE 03), IEEE CS Press, 2003, pp. 90-95.
- [14] S. Murali and G. De Micheli, "Bandwidth-Constrained Mapping of Cores onto NoC Architectures," Proc. Design, Automation and Test in Europe (DATE 04), IEEE CS Press, 2004, pp. 896-901.
- [15] S. Manolache, P. Eles, and Z. Peng, "Fault and Energy-Aware Communication Mapping with Guaranteed Latency for Applications Implemented on NoC," Proc. 42nd Annual Design Automation Conf. (DAC 05), ACM Press, 2005, pp. 266-269.
- [16] J. Hu and R. Marculescu, "Energy- and Performance- Aware Mapping for Regular NoC Architectures," IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol. 24, no. 4, 2005, pp. 551-562.
- [17] C. Marcon et al., "Comparison of NoC Mapping Algorithms Targeting Low Energy Consumption," IET Computers & Digital Techniques, vol. 2, no. 6, 2008, pp. 471-482.
- [18] Y. Xu, K. Li, J. Hu and K. li, "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," Information Sciences, Vol.270, pp. 255-287, Elsevier, 2014.
- [19] E. L. s. Carvalho, N. L. V. Calazans and F. G. Moraes, "Dynamic Task Mapping for MPSoCs," IEEE Design and Test of Computers, 2010.
- [20] R. Hwang, M. Gen and H. Katayama, "A comparison of multiprocessor task scheduling algorithms with communication costs" Computers & Operations Research, Vol. 35, pp. 976 – 993, ELSEVIER, 2008.

Turku Centre for Computer Science

TUCS Dissertations

1. **Marjo Lipponen**, On Primitive Solutions of the Post Correspondence Problem
2. **Timo Käkölä**, Dual Information Systems in Hyperknowledge Organizations
3. **Ville Leppänen**, Studies on the Realization of PRAM
4. **Cunsheng Ding**, Cryptographic Counter Generators
5. **Sami Viitanen**, Some New Global Optimization Algorithms
6. **Tapio Salakoski**, Representative Classification of Protein Structures
7. **Thomas Långbacka**, An Interactive Environment Supporting the Development of Formally Correct Programs
8. **Thomas Finne**, A Decision Support System for Improving Information Security
9. **Valeria Mihalache**, Cooperation, Communication, Control. Investigations on Grammar Systems.
10. **Marina Waldén**, Formal Reasoning About Distributed Algorithms
11. **Tero Laihonen**, Estimates on the Covering Radius When the Dual Distance is Known
12. **Lucian Ilie**, Decision Problems on Orders of Words
13. **Jukkapekka Hekanaho**, An Evolutionary Approach to Concept Learning
14. **Jouni Järvinen**, Knowledge Representation and Rough Sets
15. **Tomi Pasanen**, In-Place Algorithms for Sorting Problems
16. **Mika Johnsson**, Operational and Tactical Level Optimization in Printed Circuit Board Assembly
17. **Mats Aspñäs**, Multiprocessor Architecture and Programming: The Hathi-2 System
18. **Anna Mikhajlova**, Ensuring Correctness of Object and Component Systems
19. **Vesa Torvinen**, Construction and Evaluation of the Labour Game Method
20. **Jorma Boberg**, Cluster Analysis. A Mathematical Approach with Applications to Protein Structures
21. **Leonid Mikhajlov**, Software Reuse Mechanisms and Techniques: Safety Versus Flexibility
22. **Timo Kaukoranta**, Iterative and Hierarchical Methods for Codebook Generation in Vector Quantization
23. **Gábor Magyar**, On Solution Approaches for Some Industrially Motivated Combinatorial Optimization Problems
24. **Linas Laibinis**, Mechanised Formal Reasoning About Modular Programs
25. **Shuhua Liu**, Improving Executive Support in Strategic Scanning with Software Agent Systems
26. **Jaakko Järvi**, New Techniques in Generic Programming – C++ is more Intentional than Intended
27. **Jan-Christian Lehtinen**, Reproducing Kernel Splines in the Analysis of Medical Data
28. **Martin Büchi**, Safe Language Mechanisms for Modularization and Concurrency
29. **Elena Troubitsyna**, Stepwise Development of Dependable Systems
30. **Janne Näppi**, Computer-Assisted Diagnosis of Breast Calcifications
31. **Jianming Liang**, Dynamic Chest Images Analysis
32. **Tiberiu Seceleanu**, Systematic Design of Synchronous Digital Circuits
33. **Tero Aittokallio**, Characterization and Modelling of the Cardiorespiratory System in Sleep-Disordered Breathing
34. **Ivan Porres**, Modeling and Analyzing Software Behavior in UML
35. **Mauno Rönkkö**, Stepwise Development of Hybrid Systems
36. **Jouni Smed**, Production Planning in Printed Circuit Board Assembly
37. **Vesa Halava**, The Post Correspondence Problem for Market Morphisms
38. **Ion Petre**, Commutation Problems on Sets of Words and Formal Power Series
39. **Vladimir Kvassov**, Information Technology and the Productivity of Managerial Work
40. **Frank Tétard**, Managers, Fragmentation of Working Time, and Information Systems

41. **Jan Manuch**, Defect Theorems and Infinite Words
42. **Kalle Ranto**, Z_4 -Goethals Codes, Decoding and Designs
43. **Arto Lepistö**, On Relations Between Local and Global Periodicity
44. **Mika Hirvensalo**, Studies on Boolean Functions Related to Quantum Computing
45. **Pentti Virtanen**, Measuring and Improving Component-Based Software Development
46. **Adekunle Okunoye**, Knowledge Management and Global Diversity – A Framework to Support Organisations in Developing Countries
47. **Antonina Kloptchenko**, Text Mining Based on the Prototype Matching Method
48. **Juha Kivijärvi**, Optimization Methods for Clustering
49. **Rimvydas Rukšėnas**, Formal Development of Concurrent Components
50. **Dirk Nowotka**, Periodicity and Unbordered Factors of Words
51. **Attila Gyenesei**, Discovering Frequent Fuzzy Patterns in Relations of Quantitative Attributes
52. **Petteri Kaitovaara**, Packaging of IT Services – Conceptual and Empirical Studies
53. **Petri Rosendahl**, Niho Type Cross-Correlation Functions and Related Equations
54. **Péter Majlender**, A Normative Approach to Possibility Theory and Soft Decision Support
55. **Seppo Virtanen**, A Framework for Rapid Design and Evaluation of Protocol Processors
56. **Tomas Eklund**, The Self-Organizing Map in Financial Benchmarking
57. **Mikael Collan**, Giga-Investments: Modelling the Valuation of Very Large Industrial Real Investments
58. **Dag Björklund**, A Kernel Language for Unified Code Synthesis
59. **Shengnan Han**, Understanding User Adoption of Mobile Technology: Focusing on Physicians in Finland
60. **Irina Georgescu**, Rational Choice and Revealed Preference: A Fuzzy Approach
61. **Ping Yan**, Limit Cycles for Generalized Liénard-Type and Lotka-Volterra Systems
62. **Joonas Lehtinen**, Coding of Wavelet-Transformed Images
63. **Tommi Meskanen**, On the NTRU Cryptosystem
64. **Saeed Salehi**, Varieties of Tree Languages
65. **Jukka Arvo**, Efficient Algorithms for Hardware-Accelerated Shadow Computation
66. **Mika Hirvikorpi**, On the Tactical Level Production Planning in Flexible Manufacturing Systems
67. **Adrian Costea**, Computational Intelligence Methods for Quantitative Data Mining
68. **Cristina Seceleanu**, A Methodology for Constructing Correct Reactive Systems
69. **Luigia Petre**, Modeling with Action Systems
70. **Lu Yan**, Systematic Design of Ubiquitous Systems
71. **Mehran Gomari**, On the Generalization Ability of Bayesian Neural Networks
72. **Ville Harkke**, Knowledge Freedom for Medical Professionals – An Evaluation Study of a Mobile Information System for Physicians in Finland
73. **Marius Cosmin Codrea**, Pattern Analysis of Chlorophyll Fluorescence Signals
74. **Aiying Rong**, Cogeneration Planning Under the Deregulated Power Market and Emissions Trading Scheme
75. **Chihab BenMoussa**, Supporting the Sales Force through Mobile Information and Communication Technologies: Focusing on the Pharmaceutical Sales Force
76. **Jussi Salmi**, Improving Data Analysis in Proteomics
77. **Orieta Celiku**, Mechanized Reasoning for Dually-Nondeterministic and Probabilistic Programs
78. **Kaj-Mikael Björk**, Supply Chain Efficiency with Some Forest Industry Improvements
79. **Viorel Preoteasa**, Program Variables – The Core of Mechanical Reasoning about Imperative Programs
80. **Jonne Poikonen**, Absolute Value Extraction and Order Statistic Filtering for a Mixed-Mode Array Image Processor
81. **Luka Milovanov**, Agile Software Development in an Academic Environment
82. **Francisco Augusto Alcaraz Garcia**, Real Options, Default Risk and Soft Applications
83. **Kai K. Kimppa**, Problems with the Justification of Intellectual Property Rights in Relation to Software and Other Digitally Distributable Media
84. **Dragoş Truşcan**, Model Driven Development of Programmable Architectures
85. **Eugen Czeizler**, The Inverse Neighborhood Problem and Applications of Welch Sets in Automata Theory

86. **Sanna Ranto**, Identifying and Locating-Dominating Codes in Binary Hamming Spaces
87. **Tuomas Hakkarainen**, On the Computation of the Class Numbers of Real Abelian Fields
88. **Elena Czeizler**, Intricacies of Word Equations
89. **Marcus Alanen**, A Metamodeling Framework for Software Engineering
90. **Filip Ginter**, Towards Information Extraction in the Biomedical Domain: Methods and Resources
91. **Jarkko Paavola**, Signature Ensembles and Receiver Structures for Oversaturated Synchronous DS-CDMA Systems
92. **Arho Virkki**, The Human Respiratory System: Modelling, Analysis and Control
93. **Olli Luoma**, Efficient Methods for Storing and Querying XML Data with Relational Databases
94. **Dubravka Ilić**, Formal Reasoning about Dependability in Model-Driven Development
95. **Kim Solin**, Abstract Algebra of Program Refinement
96. **Tomi Westerlund**, Time Aware Modelling and Analysis of Systems-on-Chip
97. **Kalle Saari**, On the Frequency and Periodicity of Infinite Words
98. **Tomi Kärki**, Similarity Relations on Words: Relational Codes and Periods
99. **Markus M. Mäkelä**, Essays on Software Product Development: A Strategic Management Viewpoint
100. **Roope Vehkalahti**, Class Field Theoretic Methods in the Design of Lattice Signal Constellations
101. **Anne-Maria Ernvall-Hytönen**, On Short Exponential Sums Involving Fourier Coefficients of Holomorphic Cusp Forms
102. **Chang Li**, Parallelism and Complexity in Gene Assembly
103. **Tapio Pahikkala**, New Kernel Functions and Learning Methods for Text and Data Mining
104. **Denis Shestakov**, Search Interfaces on the Web: Querying and Characterizing
105. **Sampo Pyysalo**, A Dependency Parsing Approach to Biomedical Text Mining
106. **Anna Sell**, Mobile Digital Calendars in Knowledge Work
107. **Dorina Marghescu**, Evaluating Multidimensional Visualization Techniques in Data Mining Tasks
108. **Tero Säntti**, A Co-Processor Approach for Efficient Java Execution in Embedded Systems
109. **Kari Salonen**, Setup Optimization in High-Mix Surface Mount PCB Assembly
110. **Pontus Boström**, Formal Design and Verification of Systems Using Domain-Specific Languages
111. **Camilla J. Hollanti**, Order-Theoretic Methods for Space-Time Coding: Symmetric and Asymmetric Designs
112. **Heidi Himmanen**, On Transmission System Design for Wireless Broadcasting
113. **Sébastien Lafond**, Simulation of Embedded Systems for Energy Consumption Estimation
114. **Evgeni Tsivtsivadze**, Learning Preferences with Kernel-Based Methods
115. **Petri Salmela**, On Commutation and Conjugacy of Rational Languages and the Fixed Point Method
116. **Siamak Taati**, Conservation Laws in Cellular Automata
117. **Vladimir Rogojin**, Gene Assembly in Stichotrichous Ciliates: Elementary Operations, Parallelism and Computation
118. **Alexey Dudkov**, Chip and Signature Interleaving in DS CDMA Systems
119. **Janne Savela**, Role of Selected Spectral Attributes in the Perception of Synthetic Vowels
120. **Kristian Nybom**, Low-Density Parity-Check Codes for Wireless Datacast Networks
121. **Johanna Tuominen**, Formal Power Analysis of Systems-on-Chip
122. **Teijo Lehtonen**, On Fault Tolerance Methods for Networks-on-Chip
123. **Eeva Suvitie**, On Inner Products Involving Holomorphic Cusp Forms and Maass Forms
124. **Linda Mannila**, Teaching Mathematics and Programming – New Approaches with Empirical Evaluation
125. **Hanna Suominen**, Machine Learning and Clinical Text: Supporting Health Information Flow
126. **Tuomo Saarni**, Segmental Durations of Speech
127. **Johannes Eriksson**, Tool-Supported Invariant-Based Programming

128. **Tero Jokela**, Design and Analysis of Forward Error Control Coding and Signaling for Guaranteeing QoS in Wireless Broadcast Systems
129. **Ville Lukkarila**, On Undecidable Dynamical Properties of Reversible One-Dimensional Cellular Automata
130. **Qaisar Ahmad Malik**, Combining Model-Based Testing and Stepwise Formal Development
131. **Mikko-Jussi Laakso**, Promoting Programming Learning: Engagement, Automatic Assessment with Immediate Feedback in Visualizations
132. **Riikka Vuokko**, A Practice Perspective on Organizational Implementation of Information Technology
133. **Jeanette Heidenberg**, Towards Increased Productivity and Quality in Software Development Using Agile, Lean and Collaborative Approaches
134. **Yong Liu**, Solving the Puzzle of Mobile Learning Adoption
135. **Stina Ojala**, Towards an Integrative Information Society: Studies on Individuality in Speech and Sign
136. **Matteo Brunelli**, Some Advances in Mathematical Models for Preference Relations
137. **Ville Junnila**, On Identifying and Locating-Dominating Codes
138. **Andrzej Mizera**, Methods for Construction and Analysis of Computational Models in Systems Biology. Applications to the Modelling of the Heat Shock Response and the Self-Assembly of Intermediate Filaments.
139. **Csaba Ráduly-Baka**, Algorithmic Solutions for Combinatorial Problems in Resource Management of Manufacturing Environments
140. **Jari Kyngäs**, Solving Challenging Real-World Scheduling Problems
141. **Arho Suominen**, Notes on Emerging Technologies
142. **József Mezei**, A Quantitative View on Fuzzy Numbers
143. **Marta Olszewska**, On the Impact of Rigorous Approaches on the Quality of Development
144. **Antti Airola**, Kernel-Based Ranking: Methods for Learning and Performance Estimation
145. **Aleksi Saarela**, Word Equations and Related Topics: Independence, Decidability and Characterizations
146. **Lasse Bergroth**, Kahden merkkijonon pisimmän yhteisen alijonon ongelma ja sen ratkaiseminen
147. **Thomas Canhao Xu**, Hardware/Software Co-Design for Multicore Architectures
148. **Tuomas Mäkilä**, Software Development Process Modeling – Developers Perspective to Contemporary Modeling Techniques
149. **Shahrokh Nikou**, Opening the Black-Box of IT Artifacts: Looking into Mobile Service Characteristics and Individual Perception
150. **Alessandro Buoni**, Fraud Detection in the Banking Sector: A Multi-Agent Approach
151. **Mats Neovius**, Trustworthy Context Dependency in Ubiquitous Systems
152. **Fredrik Degerlund**, Scheduling of Guarded Command Based Models
153. **Amir-Mohammad Rahmani-Sane**, Exploration and Design of Power-Efficient Networked Many-Core Systems
154. **Ville Rantala**, On Dynamic Monitoring Methods for Networks-on-Chip
155. **Mikko Pelto**, On Identifying and Locating-Dominating Codes in the Infinite King Grid
156. **Anton Tarasyuk**, Formal Development and Quantitative Verification of Dependable Systems
157. **Muhammad Mohsin Saleemi**, Towards Combining Interactive Mobile TV and Smart Spaces: Architectures, Tools and Application Development
158. **Tommi J. M. Lehtinen**, Numbers and Languages
159. **Peter Sarlin**, Mapping Financial Stability
160. **Alexander Wei Yin**, On Energy Efficient Computing Platforms
161. **Mikołaj Olszewski**, Scaling Up Stepwise Feature Introduction to Construction of Large Software Systems
162. **Maryam Kamali**, Reusable Formal Architectures for Networked Systems
163. **Zhiyuan Yao**, Visual Customer Segmentation and Behavior Analysis – A SOM-Based Approach
164. **Timo Jolivet**, Combinatorics of Pisot Substitutions
165. **Rajeev Kumar Kanth**, Analysis and Life Cycle Assessment of Printed Antennas for Sustainable Wireless Systems
166. **Khalid Latif**, Design Space Exploration for MPSoC Architectures

167. **Bo Yang**, Towards Optimal Application Mapping for Energy-Efficient Many-Core Platforms
168. **Ali Hanzala Khan**, Consistency of UML Based Designs Using Ontology Reasoners
169. **Sonja Leskinen**, m-Equine: IS Support for the Horse Industry
170. **Fareed Ahmed Jokhio**, Video Transcoding in a Distributed Cloud Computing Environment
171. **Moazzam Fareed Niazi**, A Model-Based Development and Verification Framework for Distributed System-on-Chip Architecture
172. **Mari Huova**, Combinatorics on Words: New Aspects on Avoidability, Defect Effect, Equations and Palindromes
173. **Ville Timonen**, Scalable Algorithms for Height Field Illumination
174. **Henri Korvela**, Virtual Communities – A Virtual Treasure Trove for End-User Developers
175. **Kameswar Rao Vaddina**, Thermal-Aware Networked Many-Core Systems
176. **Janne Lahtiranta**, New and Emerging Challenges of the ICT-Mediated Health and Well-Being Services
177. **Irum Rauf**, Design and Validation of Stateful Composite RESTful Web Services
178. **Jari Björne**, Biomedical Event Extraction with Machine Learning
179. **Katri Haverinen**, Natural Language Processing Resources for Finnish: Corpus Development in the General and Clinical Domains
180. **Ville Salo**, Subshifts with Simple Cellular Automata
181. **Johan Ersfolk**, Scheduling Dynamic Dataflow Graphs
182. **Hongyan Liu**, On Advancing Business Intelligence in the Electricity Retail Market
183. **Adnan Ashraf**, Cost-Efficient Virtual Machine Management: Provisioning, Admission Control, and Consolidation
184. **Muhammad Nazrul Islam**, Design and Evaluation of Web Interface Signs to Improve Web Usability: A Semiotic Framework
185. **Johannes Tuikkala**, Algorithmic Techniques in Gene Expression Processing: From Imputation to Visualization
186. **Natalia Díaz Rodríguez**, Semantic and Fuzzy Modelling for Human Behaviour Recognition in Smart Spaces. A Case Study on Ambient Assisted Living
187. **Mikko Pänkäälä**, Potential and Challenges of Analog Reconfigurable Computation in Modern and Future CMOS
188. **Sami Hyrynsalmi**, Letters from the War of Ecosystems – An Analysis of Independent Software Vendors in Mobile Application Marketplaces
189. **Seppo Pulkkinen**, Efficient Optimization Algorithms for Nonlinear Data Analysis
190. **Sami Pyötiälä**, Optimization and Measuring Techniques for Collect-and-Place Machines in Printed Circuit Board Industry
191. **Syed Mohammad Asad Hassan Jafri**, Virtual Runtime Application Partitions for Resource Management in Massively Parallel Architectures
192. **Toni Ernvall**, On Distributed Storage Codes
193. **Yuliya Prokhorova**, Rigorous Development of Safety-Critical Systems
194. **Olli Lahdenoja**, Local Binary Patterns in Focal-Plane Processing – Analysis and Applications
195. **Annika H. Holmbom**, Visual Analytics for Behavioral and Niche Market Segmentation
196. **Sergey Ostroumov**, Agent-Based Management System for Many-Core Platforms: Rigorous Design and Efficient Implementation
197. **Espen Suenson**, How Computer Programmers Work – Understanding Software Development in Practise
198. **Tuomas Poikela**, Readout Architectures for Hybrid Pixel Detector Readout Chips
199. **Bogdan Iancu**, Quantitative Refinement of Reaction-Based Biomodels
200. **Ilkka Törmä**, Structural and Computational Existence Results for Multidimensional Subshifts
201. **Sebastian Okser**, Scalable Feature Selection Applications for Genome-Wide Association Studies of Complex Diseases
202. **Fredrik Abbors**, Model-Based Testing of Software Systems: Functionality and Performance
203. **Inna Pereverzeva**, Formal Development of Resilient Distributed Systems
204. **Mikhail Barash**, Defining Contexts in Context-Free Grammars
205. **Sepinoud Azimi**, Computational Models for and from Biology: Simple Gene Assembly and Reaction Systems
206. **Petter Sandvik**, Formal Modelling for Digital Media Distribution

207. **Jongyun Moon**, Hydrogen Sensor Application of Anodic Titanium Oxide Nanostructures
208. **Simon Holmbacka**, Energy Aware Software for Many-Core Systems
209. **Charalampos Zinoviadis**, Hierarchy and Expansiveness in Two-Dimensional Subshifts of Finite Type
210. **Mika Murtojärvi**, Efficient Algorithms for Coastal Geographic Problems
211. **Sami Mäkelä**, Cohesion Metrics for Improving Software Quality
212. **Eyal Eshet**, Examining Human-Centered Design Practice in the Mobile Apps Era
213. **Jetro Vesti**, Rich Words and Balanced Words
214. **Jarkko Peltomäki**, Privileged Words and Sturmian Words
215. **Fahimeh Farahnakian**, Energy and Performance Management of Virtual Machines: Provisioning, Placement and Consolidation
216. **Diana-Elena Gratie**, Refinement of Biomodels Using Petri Nets
217. **Harri Merisaari**, Algorithmic Analysis Techniques for Molecular Imaging
218. **Stefan Grönroos**, Efficient and Low-Cost Software Defined Radio on Commodity Hardware
219. **Noora Nieminen**, Garbling Schemes and Applications
220. **Ville Taajamaa**, O-CDIO: Engineering Education Framework with Embedded Design Thinking Methods
221. **Johannes Holvitie**, Technical Debt in Software Development – Examining Premises and Overcoming Implementation for Efficient Management
222. **Tewodros Deneke**, Proactive Management of Video Transcoding Services
223. **Kashif Javed**, Model-Driven Development and Verification of Fault Tolerant Systems
224. **Pekka Naula**, Sparse Predictive Modeling – A Cost-Effective Perspective
225. **Antti Hakkala**, On Security and Privacy for Networked Information Society – Observations and Solutions for Security Engineering and Trust Building in Advanced Societal Processes
226. **Anne-Maarit Majanoja**, Selective Outsourcing in Global IT Services – Operational Level Challenges and Opportunities
227. **Samuel Rönqvist**, Knowledge-Lean Text Mining
228. **Mohammad-Hashem Hahgbayan**, Energy-Efficient and Reliable Computing in Dark Silicon Era
229. **Charmi Panchal**, Qualitative Methods for Modeling Biochemical Systems and Datasets: The Logicome and the Reaction Systems Approaches
230. **Erkki Kaila**, Utilizing Educational Technology in Computer Science and Programming Courses: Theory and Practice
231. **Fredrik Robertsén**, The Lattice Boltzmann Method, a Petaflop and Beyond
232. **Jonne Pohjankukka**, Machine Learning Approaches for Natural Resource Data
233. **Paavo Nevalainen**, Geometric Data Understanding: Deriving Case-Specific Features
234. **Michal Szabados**, An Algebraic Approach to Nivat’s Conjecture
235. **Tuan Nguyen Gia**, Design for Energy-Efficient and Reliable Fog-Assisted Healthcare IoT Systems
236. **Anil Kanduri**, Adaptive Knobs for Resource Efficient Computing
237. **Veronika Suni**, Computational Methods and Tools for Protein Phosphorylation Analysis
238. **Behailu Negash**, Interoperating Networked Embedded Systems to Compose the Web of Things
239. **Kalle Rindell**, Development of Secure Software: Rationale, Standards and Practices
240. **Jurka Rahikkala**, On Top Management Support for Software Cost Estimation
241. **Markus A. Whiteland**, On the k-Abelian Equivalence Relation of Finite Words
242. **Mojgan Kamali**, Formal Analysis of Network Routing Protocols
243. **Jesús Carabaño Bravo**, A Compiler Approach to Map Algebra for Raster Spatial Modeling
244. **Amin Majd**, Distributed and Lightweight Meta-heuristic Optimization Method for Complex Problems

TURKU CENTRE *for* COMPUTER SCIENCE

<http://www.tucs.fi>

tucs@abo.fi



University of Turku

Faculty of Science and Engineering

- Department of Future Technologies
- Department of Mathematics and Statistics

Turku School of Economics

- Institute of Information Systems Science



Åbo Akademi University

Faculty of Science and Engineering

- Computer Engineering
- Computer Science

Faculty of Social Sciences, Business and Economics

- Information Systems

ISBN 978-952-12-3864-2

ISSN 1239-1883

Amin Majid

Amin Majid

Amin Majid

Distributed and Lightweight Meta-heuristic Optimization Method for Complex Problems

Distributed and Lightweight Meta-heuristic Optimization Method for Complex Problems

Distributed and Lightweight Meta-heuristic Optimization Method for Complex Problems