Csaba Ráduly-Baka

Algorithmic Solutions for Combinatorial Problems in Resource Management of Manufacturing Environments

Turku Centre *for* Computer Science

TUCS Dissertations
No 139, August 2011

Algorithmic Solutions for Combinatorial Problems in Resource Management of Manufacturing Environments.

Csaba Ráduly-Baka

Department of Information Technology
Computer Science
20014 University of Turku
Finland

2011

Supervisors

Olli S. Nevalainen
Department of Computer Science
Information Technology
20014 University of Turku
Finland

Timo Knuutila
Department of Computer Science
Information Technology
20014 University of Turku
Finland

Reviewers

Erkki Mäkinen
Department of Computer Science
Information Technology
20014 University of Turku
Finland

Hans-Otto Günther
Department of Production Management
10623 University of Berlin
Germany

Opponent

Kauko Leiviskä
Faculty of Technology
90014 University of Oulu
Finland

ISBN 978-952-12-2618-2
ISSN 1239-1883

Abstract

This thesis studies the use of heuristic algorithms in a number of combinatorial problems that occur in various resource constrained environments. Such problems occur, for example, in manufacturing, where a restricted number of resources (tools, machines, feeder slots) are needed to perform some operations. Many of these problems turn out to be computationally intractable, and heuristic algorithms are used to provide efficient, yet sub-optimal solutions. The main goal of the present study is to build upon existing methods to create new heuristics that provide improved solutions for some of these problems. All of these problems occur in practice, and one of the motivations of our study was the request for improvements from industrial sources.

We approach three different resource constrained problems. The first is the *tool switching and loading* problem, and occurs especially in the assembly of printed circuit boards. This problem has to be solved when an efficient, yet small primary storage is used to access resources (tools) from a less efficient (but unlimited) secondary storage area. We study various forms of the problem and provide improved heuristics for its solution. Second, the nozzle assignment problem is concerned with selecting a suitable set of vacuum nozzles for the arms of a robotic assembly machine. It turns out that this is a specialized formulation of the *MINMAX* resource allocation formulation of the *apportionment problem* and it can be solved efficiently and optimally. We construct an exact algorithm specialized for the nozzle selection and provide a proof of its optimality. Third, the problem of feeder assignment and component tape construction occurs when electronic components are inserted and certain component types cause tape movement delays that can significantly impact the efficiency of printed circuit board assembly. Here, careful selection of component slots in the feeder improves the tape movement speed. We provide a formal proof that this problem is of the same complexity as the *turnpike problem* (a well studied geometric optimization problem), and provide a heuristic algorithm for this problem.

Acknowledgements

I would like to express my sincere thanks to all involved in supporting my studies. This research work was carried out at the Computer Science Department of the University of Turku. I have done this research as a part time activity, parallel to a full time employment. I would like to thank Turku University and Turku Centre for Computer Science for making this work possible.

I would like to thank my supervisors Professor Olli Nevalainen, PhD, and Professor Timo Knuutila, PhD, for the advice, guidance and encouragement given to complete my research papers and thesis. I'm also grateful to Kari Salonen, PhD, with whom I coauthored my first paper. I would also like to thank Mika Johnsson, PhD, who provided real word production planning problems for my research.

I also thank the reviewers of this thesis Professor Erkki Mäkinen, PhD, and Professor Dr. Hans-Otto Günther for their efforts and feedback.

I would also like to thank my family for their support and encouragement, which has been a significant factor in motivating the completion of my work.

List of original publications

P1 Salonen, K., Raduly-Baka, Cs., Nevalainen, O. A note on the tool switching problem of a flexible machine. *Computers & Industrial Engineering 50(4)*, Aug 2006, 458-465.

P2 Raduly-Baka, Cs., Knuutila T., Nevalainen, O. Minimizing the number of tool switches with tools of different sizes. In *Proceedings of the 5th International Conference on Technology and Automation 2005 (ICTA'05)*, Oct 2005.

P3 Raduly-Baka, Cs., Knuutila T., Johnsson, M., Nevalainen, O. Selecting the Nozzle Assortment for a Gantry-Type Placement Machine. *OR Spectrum*, 30, 493-513, 2008.

P4 Raduly-Baka, Cs., Knuutila T., Johnsson, M., Nevalainen, O. Construction of Component Tapes for Radial Placement Machines. *Computers & Operations Research*, Electronic edition():1-35, Nov 2009.

P5 Raduly-Baka, Cs., Knuutila T., Johnsson, M., Nevalainen, O. Organizing the operation of radial machine sequencers for multiple PCB-types. To appear in *Computers & Industrial Engineering*.

Contents

Chapter 1

Introduction

In automated manufacturing, and especially in electronics manufacturing, process efficiency has increasingly become the dominant factor for manufacturing service providers. Efficiency in this context is a broad concept that includes not just the strict time and monetary objectives, but also various aspects of quality, like material specific constraints. The objective of optimizing various aspects of manufacturing efficiency has been subject to continuous research for several decades. In present automated and intelligent manufacturing systems, computers are used to drive numerically controlled production facilities. Here, the efficiency of manufacturing can be improved with the application of exact algorithms or approximation heuristics as applied to optimization models formed for these systems.

In the second half of the past century, electronics manufacturing was an increasing source of employment in the developed countries. As the manufacturing processes were automated, the cost of employing workers has become a more important factor in the overall cost. This led to the rise of cost effective manufacturing services, provided by facilities in the developing countries like Taiwan and China. The past decade has seen the virtual removal of electronics manufacturing from developed countries (with some exception in low-volume, high-margin specialized products), and a spectacular development of outsourced manufacturing services. This reduced the cost of labour in mass production significantly, and facilitated the growth of companies that are solely focused on manufacturing services.

The separation of manufacturing from the brand owner companies shifted the value focus away from manufacturing to other aspects of product creation, like industrial design and user experience. This put a further pressure on the manufacturing margins, which in turn led to a consolidation of these services into a few large *electronics manufacturing services* (EMS) companies. These provide manufacturing services to multiple brand owner companies, initially for low-end products, but increasingly moving into higher-end

manufacturing also (observe the use of such manufacturing services by Apple to build high-end electronics products, for example).

As manufacturing concentrated into EMS companies, the efficiency objective gained a new dimension, the product variation and manufacturing flexibility. Increasingly, the same general purpose CNC machine lines are (and will be) used to manufacture and assemble different products (sometimes for competing companies), and in most cases the individual product volume have been decreasing. Also the capabilities of these CNC machines have increased, in terms of their configurability for multiple purposes. This has lead to a situation where not just a single product type determines the overall manufacturing efficiency, but it is determined by the ability to reconfigure the CNC lines and efficiently manufacture multiple product types with potentially medium or low volume.

The growth of integrated EMS provider companies has led to the commoditization of electronics manufacturing services. This has resulted in a continuous increase in the overall volume (and type) of electronics products, and a continuous growth of this market. The concentration of EMS companies (and other type of manufacturing) in Asia, and especially in China, has led to an unprecedented trade surplus and growth of the region, which has resulted in the creation of a large middle class with disposable income. This puts an upward pressure on labour costs in the region also, and it is likely that we will witness a similar increase of labour cost as has happened previously in developed countries.

In addition to increasing labour costs (and as an effect of it), the EMS companies are moving towards manufacturing higher-end products with improving quality. In the past decade, as the established consumer electronics companies competed over market shares, the focus was mostly on the volume and much less on margin. Recently there is a developing trend where exceptional quality and integration of consumer electronics is provided at premium prices, while ignoring the volume market share (see Apple). These products not just redefine the meaning of high-end, but also (since being manufactured by EMS companies) increase the ability of EMS companies to manufacture high-end products. In addition to the developments of the industrial design and user experience, a new level of product integration also becomes a differentiating factor among consumer electronics companies.

As the labour costs increase in east Asia, electronics manufacturing will evidently witness an increasing automatization of previously labour intensive tasks. With the recent shift of focus towards high-end products, the shift of low-end products towards what used to be high-end, and the increase of product variation, electronics manufacturing processes continuously will require new optimization approaches and heuristic algorithms. As the products and manufacturing requirements evolve, so do the associated objective functions of efficiency.

The integrated manufacturing systems tend to be rather complex and their efficiency can be influenced by thousands of interdependent variables. Also, there are often multiple criteria for efficiency, depending on the products and quality requirements. To address the whole manufacturing process with a single objective function has proved to be unfeasible; even relatively restricted sub-problems are known to be NP-hard and difficult to approximate. Thus, manufacturing processes are commonly separated into hierarchies of sub-problems. Individual sub-problems can either be solved optimally using exact methods, or (in most cases) they can be approached using various heuristic algorithms. Since most of these sub-problems are also known to be NP-hard, finding better and more efficient optimization heuristics has become an extensive research subject in the field. The solutions of smaller individual sub-problems then can be recombined hierarchically to approximate the solution of the joint manufacturing problem.

The focus in the present research is on finding improved solutions to a number of such sub-problems that occur in real life manufacturing environments. These types of problems exists when a limited set of resources (tool magazines, nozzle fixtures, or component feeders) are used in the manufacturing process, and the way in which these resources are allocated can have a significant impact on the overall manufacturing process. In the case of nozzle allocation of component assembly robots, these problems can be solved exactly and mathematical proofs are provided on the efficiency of such solutions. For component feeder assignment, proofs will be provided that the problem is reducible to well studied combinatorial optimization problems. As for the tool switching, significant improvements of known heuristics are reached.

Our research is based on the main hypothesis that the current algorithmic techniques are capable of finding improved solutions to these practical problems. Each of the above resource management problems will be approached with the goal of crafting heuristics that make use of the specific aspects of the problem in order to provide improved results over general purpose combinatorial approaches. Observing the results of our tool-switching algorithm for non-uniform tool sizes, we can conclude that using randomized improvement heuristics on practical problem instances of this type results in significantly better solutions than when searching for solutions with deterministic greedy steps. In contrast to that, for uniform tool sizes, our studies did not show significant differences between randomized and deterministic methods. For the feeder assignment problem we provide a formal proof that it reduces to the geometric *turnpike* problem. However, the computational complecity of the turnpike problem is presently not known, which still leaves the exact status of the feeder assignment problem open.

1.1 Flexible Manufacturing and PCB assembly

Flexible and automated manufacturing processes form an integral part of different industries, including *Printed Circuit Board* (PCB) manufacturing. The initial goal of flexible manufacturing processes has been to automate the production of consumer products, so that production lines can operate autonomously and un-interrupted (Hirvikorpi [40]). Although this is not a fully achievable goal, over the years a number of benefits of such systems have materialized. Among others, automated manufacturing systems improve product quality, reduce production times (using operational optimizations), reduce the use of human workers for repetitive tasks and reuse production equipments while manufacturing various product types (see Smed *et al.* [52]).

A *flexible manufacturing system* (FMS) consists of a set of numerically controlled production machines (integrated with material handling equipment), which are under the control of a central unit to produce various products. The machines perform operations on the product units, where each operation requires a certain set of tools. Multiple machines can be used to perform similar tasks in separate assembly lines, and an assembly line can also contain multiple machines performing specialized tasks. Among others, the problem of balancing lines and assigning tools to limited tool magazines arises.

Flexible manufacturing processes and models are also employed in electronics manufacturing. Here the goal is to assemble PCB units with optimal efficiency and quality from a given set of components. Instead of using tools (as in the metal cutting industry), electronics manufacturing machines use feeders to supply components to the assembly point, and robotic arms and vacuum nozzles to place the components onto the PCB. This presents similar optimization challenges as the conventional manufacturing industries, with some different aspects and variables. Zhou *et al.* [51] observed that PCB assembly exhibits a number of properties that make it different from conventional systems. A PCB requires numerous insertions, and these are highly repetitive tasks and there is no strict sequence to be followed (or just a partial order). Also in PCB assembly, the tooling is more constrained due to the restrictions on the widths of the component reels (Smed *et al.* [52]).

Chapter 2

Tool Switching in Flexible Manufacturing Systems

In flexible manufacturing systems, the management of tools used to process jobs (or groups of jobs) can play an important role in the overall efficiency of the production process. The *Tool Management Problem* arises in scenarios where a single machine is used to process multiple jobs of different types, and each job requires a different set of tools. The machine has a primary tool magazine of limited capacity but efficient access, and a secondary magazine of unlimited capacity, but inefficient access. The primary tool magazine does not have usually enough capacity to hold all the tools required by all the jobs, so some of the tools may need to be removed to make room for new tools when processing a new job. The tools are kept in the secondary tool magazine, and when needed brought into the primary magazine. Usually this is a time consuming step, that can be optimized by selecting a proper job sequence, and determining which tool to remove from the primary magazine to make room for new tools.

The central problem of tool management in flexible manufacturing systems is to determine how to sequence the jobs that are processed and when to load (and unload) the tools from the magazine, in order to minimize the number of tool setups [4]. This problem is especially important when changing the tools has a significant impact on the overall processing time. Originally this problem has been formulated for the metalworking industry that uses NC-forging equipment. The same problem occurs in the electronics industry [2] in the manufacturing of PCBs, when the same automated placement machine is used to manufacture different types of PCBs. Each PCB type requires a certain collection of component feeders that must be placed on the machine before the PCBs can be manufactured. Since the machine can hold only a limited number of such feeders, when the production switches to a new PCB type, some of the feeders are replaced by new

ones, specific of the new PCB type. For example in high-mix PCB shops at Hewlett-Packard the tool setup operations could take as much as 50% of the production time [7].

Next, we give a notation for the tool switching problem. Let C denote the capacity of the primary tool magazine, N the number of jobs to be processed, and T_j the set of tools used by job $j \in \{1, 2, .., N\}$. Suppose there are a total of M tools needed by all the jobs, so each T_j is a subset of $\{1, 2, .., M\}$. We assume that $M > C$, that is, the capacity of the tool magazine is not sufficient to hold all the tools required by all the jobs. We can also assume that $|T_j| \leq C$, for $j = 1, 2, ..., N$, that is, all jobs can be processed with the given tool magazine capacity. Since $M > C$, it will be necessary at some point to remove some tools already in the primary magazine, and load new tools from the secondary into the primary magazine. This is called the *tool switching step*, and its cost depends on the cost of removing and inserting these tools into the magazine. The goal in the *Tool Switching Problem* is to find a permutation of the jobs $\{1, 2, .., N\}$, and a set of tool loading steps, so that the total number of tool switching steps are minimized. If the job order is fixed, the decision on when to load the tools may still affect the efficiency of the process. This is called the *Tool Loading Problem*, and its goal is to determine when and from where to remove tools in order to make room for new tools while moving to the next job.

There are multiple variations of the tool switching problem discussed in the literature depending on the machine, tool and job types:

1. In the simplest form of the tool switching problem, the tools have a uniform size, each slot in the magazine can hold one tool and the cost of switching the tools is equal. If the sequence of jobs is fixed, the tool loading problem can be solved optimally by the means of the KTNS (Keep Tool Needed Soonest) algorithm (Tang *et al.* [8], Crama *et al.* [5]). Finding an optimal job permutation is much more difficult, even for uniform tool sizes the problem has been proved to be NP-hard (Crama *et al.* [5]). In the uniform tool size case, the locations of the tools in the magazine do not have an impact on the performance of the process.

2. The tools have a non-uniform sizes. In this case the tool magazine can be seen as a linear array of slots, and a tool may occupy one or more consequtive slots. Each tool may have a different switching cost, which does not need to be a function of the number of slots it occupies. The tool loading problem (with fixed job order) for non-uniform tool sizes is NP-hard (as shown by Crama *et al.* [5]). Finding an optimal job permutation is also NP-hard, as shown by Tzur *et al.* [21]. The KTNS-rule was extended by Tzur *et al.* [20] to handle the case of non-uniform tool size; however, the revised rule, KSTNS (Keep Smallest

Tools Needed Soonest) does not guarantee optimal solution. In this case, the placement of the tools in the tool magazines becomes relevant, and the available slots in the magazines can become fragmented. This raises the problem of *slot assignment*, which concerns about deciding where to put the tools in the magazine.

3. The primary tool magazine can be of a linear or a circular type. This has no impact on problem instances with uniform tool sizes, but it becomes relevant when non-uniform tool sizes are used. For non-uniform tool sizes, circular magazines tend to perform better.

4. The order of jobs may be fixed, which leads to the tool loading problem. For uniform tool sizes, this can be solved optimally as by the means of the KTNS-procedure. For non-uniform tool sizes the problem is NP-hard.

5. When the order of the jobs is not fixed, it may still be limited by a partial order. This is a typical real life scenario, where certain jobs must be produced before others due to constraints on component sizes, availability or other manufacturing requirements. This requirement was used by Djellab *et al.* in [9], and a randomized iterative improvement heuristic was proposed for tools of uniform size.

6. The tool loading and switching problems can be of online or offline type. In the case of an offline problem all jobs are known, and their processing order and the total number of switches can be optimized for more efficient manufacturing. In the case of online production control, only one or a limited number of jobs are known beforehand at a given moment of time.

A simplified version of the online tool switching problem is the *paging problem* that occurs in operating systems design. Since the jobs (*i.e. collections of pages needed*) are not known in advance, the problem is mainly concerned on which pages to remove when bringing new pages into the primary memory (tool loading problem). Here, the data is stored in pages of fixed size. The problem is similar to the tool management in that the pages are swapped between a primary (and more efficient) storage and a secondary (less efficient) storage.

The primary storage is the main memory of the computer, while the secondary storage is usually a slow persistent storage device like a hard disk or a flash memory. The primary memory can store a limited number of pages. The goal (of the paging problem) is to minimize the number of page swaps between the primary and secondary storage considering that the content of some pages has been changed and they need to be written

	j_1	j_2	j_3	j_4	j_5	j_6	j_7	j_8	j_9
t_1	1	0	0	1	1	1	1	0	1
t_2	0	1	1	1	0	1	0	1	1
t_3	1	1	0	1	1	0	1	0	0
t_4	1	0	1	0	1	1	0	1	1
t_5	0	1	1	1	1	1	0	0	0
t_6	1	1	1	0	0	0	1	1	1

Figure 2.1: Tool-job incidence matrix. Rows represent tools, columns represent jobs. An entry is 1 if the tool is needed by the job, 0 otherwise.

back to the secondary storage when they are removed from the primary storage. Other pages, the contents of which has not changed, can just be overwritten to make room for new pages from the secondary storage. The *paging problem* ([3]) is an on-line problem of fixed item size (the page sizes are fixed by the system), the request for a specific page is determined by the running application, and is not known in advance.

2.1 Complexity of Tool Management

In this section we briefly discuss the complexity of different variants of the tool management problem. As mentioned earlier, the tool loading problem for uniform tool sizes (the job sequence is fixed) can be solved optimally in polynomial time by the means of the KTNS-procedure.

The case of tool switching problem with uniform tool sizes (where the job order is not fixed, and the goal is to find a job order that minimizes the need for tool switching), is NP-hard [4] for both the decision and the optimization version of the problem. The decision version of the tool switching problem asks if there is a job sequence that requires exactly M setups, where the input is an $M \times N$ tool-job incidence matrix (see figure 2.1) A and a magazine capacity C, with rows representing the tools and the columns representing the jobs. This problem turns out to be the decision version of the *matrix permutation problem* ([4]) that is known to be NP-hard. The optimization version of the tool switching problem asks to find a job sequence where the number of tool switches is minimized (given the matrix A and capacity C). This is also NP-hard for any fixed $C \geq 2$, as shown by Crama *et al.* [4].

For the tool switching problem with non-uniform tool sizes it is obvious that finding a job sequence is NP-hard. This is because the uniform-sized tool switching problem is NP-hard and it is a sub-problem of the non-uniform sized tool switching problem.

The tool loading problem with non-uniform tool sized was proved to be NP-hard (Crama *et al.* [5]), when the magazine capacity C is not fixed. The proof uses a reduction from the 3-Partition problem, which is known to be NP-complete (see Garey and Johnson [6]).

The 3-Partition problem is defined as follows: Given as set S of $3 \cdot n$ elements, a positive integer B, and integral weights $w_k \geq 0$ for all elements $k \in S$ such that $\frac{B}{4} < w_k < \frac{B}{2}$ and such that $\sum_{k \in S} w_k = n \cdot B$, the problem is to partition S into n disjoint triples $T_1, ..., T_n$ such that, for $1 \leq i \leq n$, $\sum_{k \in T_i} w_k = B$.

In the case of uniform tool sizes the physical location of the tools in the magazine becomes important, since a tool may occupy multiple slots, and fragmentation of the magazine space can occur. When the value of C is fixed, the tool loading problem of non-uniform tool sizes can be solved in polynomial time, by computing the shortest path on a network of $O(|T|^C C!)$ nodes, for details see [5].

2.2 Heuristics

The complexity of tool switching problems has motivated the research of optimization heuristics. These approaches typically fall into one of the following three categories [4]:

- *Construction strategies*, that construct a single job sequence from an input, using some of the specific properties of the tool switching problems.
- *Improvement strategies*, these algorithms start from an initial job sequence and iteratively improve it.
- *Composite strategies* are obtained by combining the above two, and they have been successfully employed in numerous recent studies.

An initial classification of heuristic approaches for the tool switching problem with uniform tool sizes was given by Crama *et al.* in [4]. They propose six basic strategies, which are briefly outlined next.

2.2.1 Traveling salesman heuristics

The idea to solve the tool switching problem by traveling salesman (TSP) heuristics (see Gutin *et al.* [1]), was introduced by Tang and Denardo [8]. In this approach a tool switching problem instance is transformed into a directed graph $G = (V, E, lb)$, where V is the set of jobs, E is the set of all pairs of jobs, and the length of a directed edge (i, j) is given by $lb(i, j)$ which is an underestimate (lower bound) of the cost of switching from job i to j.

Let T_i and T_j define the sets of tools for jobs i and j, and C the capacity of the tool magazine, then lb can be computed as follows:

$$lb(i,j) = max(|T_i \cup T_j| - C, 0).$$

The above edge length gives the actual cost of tool switching between jobs i and j when both jobs require exactly C tools. If they have less than C tools, then $lb(i,j)$ is a lower bound of the actual tool switching cost. The job sequence is given by a traveling salesman path in graph G.

2.2.2 Block minimization heuristics

Block minimization heuristics are an alternative way to transform the tool switching problem to the traveling salesman problem. A directed graph $D = (V^*, E, ub)$, where V^* is the set of jobs plus a 0 node, and the edge costs are given by $ub(i,j) = |T_i \backslash T_j|$. In this case ub is an upper bound on the cost of switching tools between jobs i and j. Crama *et al.* in [4] proposed two block minimization heuristics to construct a *TS* (traveling salesman) path in G: nearest neighbor heuristic and farthest insertion heuristic.

2.2.3 Greedy heuristics

Greedy heuristics build a job sequence iteratively using greedy decision rules. Crama *et al.* [4] gave an algorithm, where an initial job sequence (containing an arbitrarily selected single job) is extended to a complete sequence by iteratively adding a job so that the tool switching cost given by the *KTNS* method is minimal.

Djellab *et al.* [9] also introduced a greedy heuristic (called *Best Insertion* heuristics) as the initial construction step for their composite strategy heuristic. In this case a job is inserted into the sequence in a position that minimizes the *KTNS* cost.

2.2.4 Interval heuristics

The goal in interval heuristics is to transform the tool/job incidence matrix into an *interval matrix*. An interval matrix is a matrix of 1s and 0s, where each row of the matrix has at most one continuous block of 1s (see [10]). If the tool/job incidence matrix A is also an interval matrix, then the job sequence requires only one setup per tool, *i.e.* it is an optimal sequence.

Therefore, if the matrix A can be transformed into an interval matrix, by permuting its columns (NP-hard problem as shown in [11] and [12]), the same permutation, applied to the job order, gives an optimal job ordering.

2.2.5 2-opt strategies

2-opt strategies are simple improvement strategies that are widely used for several combinatorial hard optimization problems. Given a sequence of jobs the idea is to, improve the solution by identifying two jobs, which, if switched, result in a better job sequence in terms of the number of tool switches. Such a 2-opt heuristic has been proposed for the tool switching problem by Brad [2].

2.2.6 Load-and-optimize strategy

Load-and-optimize strategy is a combination of the interval heuristics and the traveling salesman heuristics (see Crama *et al.* [4]). Let σ be a job sequence and P be a column permutation of matrix A according to σ. The aim is to create a new matrix T, that has exactly C 1s in each column by applying the KTNS-procedure on P. This in turn means that the tool switching instance defined by matrix T is a traveling salesman problem (since each column has exactly C tools). The second step of the load-and-optimize strategy is to apply a traveling salesman heuristic to the instance T.

2.3 Hybrid heuristics for the tool switching problem

Various heuristic strategies classified by Crama *et al.* [4] use some more general combinatorial problem when solving the tool switching problem. In general, these solutions forego the nature of the tool switching problem (hence being more generic). Because of this, a number of specialized heuristics have been proposed in the literature that address the job sequencing problem more directly.

2.3.1 Improvements on the TSP model

The tool switching problem can be transformed into a TSP instance using a distance function as discussed earlier (Crama *et al.* in [4]). The main problem with this approach is that this transformation uses either an underestimate, or an overestimate of the actual cost of moving from one job to another. The cost estimate is correct only when each job uses the full magazine (*i.e.* exactly C tools), which is not a realistic assumption. A more realistic cost estimate should consider the existing content of the tool magazine, that depends on the already processed jobs. This means that the cost of moving from job i to job j depends on what jobs have already been processed, since that determines the current content of the magazine.

The cost estimation problem was addressed by Hertz *et al.* [13]. They introduced two new distance metrics as an improved estimate of the cost of moving between two jobs. They also introduced an iterative improvement algorithm called *GENIUS*, which uses an earlier TSP optimization technique called *GENI* (Gendreau *et al.* [14]). The GENIUS heuristic is in fact a general TSP heuristic that works for symmetric and asymmetric instances as well.

In the present work we further improve the GENIUS heuristic in Salonen *et al.* ([15]), by an iterative heuristic called *Grouping with Minimum Setup Algorithm (GMSA3)* that uses GENIUS as a sub-routine. The input of GSMA3 is M, N, C and a_{ij} $(i = 1...M; j = 1...N)$, where M is the number of tools, N is the number of parts (or jobs), C is the capacity of the primary tool magazine and a_{ij} are entries in the tool-job incidence matrix. The algorithm calculates an approximation for the job ordering (Π^*) of the given set of N jobs, and a cost of the tool switches (S^*).

Input parameter n_{merge} controls the number of merge operations. The MSAGenius heuristic is a multi-start algorithm, that repeats GENIUS (of Hertz *et al.* [13]) N times (where N is the number of jobs) with a different job as a starting point, and selecting the best result from this iteration.

The proposed GMSA3 algorithm uses the following steps:

1. Solve the tool switching problem using the MSAGenius procedure (see below). Let $k = 0$.
2. While $k < n_{merge}$ perform steps 3, 4 and 5.
3. Merge two jobs i and j (by creating a super-job that uses tools from both jobs), in such a way that the magazine capacity is not exceeded, using the following steps:
   - Calculate *Jaccard's similarity coefficient* $s_{ij} = \frac{|T_i \cap T_j|}{|T_i \cup T_j|}$, where T_i and T_j denote the tools for job pairs (i, j).
   - Determine the pair (i^*, j^*) with the highest similarity coefficient from all pairs i, j where the merge is possible (*i.e.* the merged job can be processed with the magazine).
   - Merge jobs i^* and j^*, by creating a super-job containing tools $E_{i^*} \cup E_{j^*}$.
   - Let $N \leftarrow N - 1$.
4. For the remaining parts, solve the tool switching problems using GENIUS with results Π and S.
5. Let $k \leftarrow k + 1$.
6. If $S < S^*$ update $\Pi^* \leftarrow \Pi$ and $S^* \leftarrow S$.

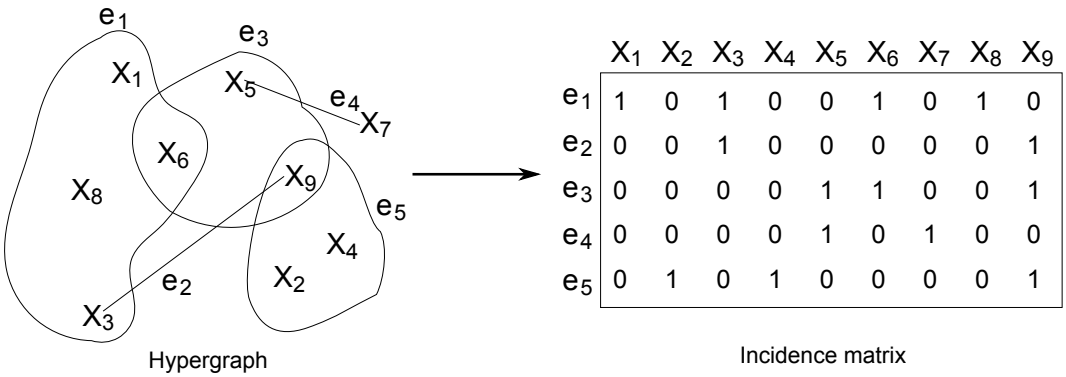

Figure 2.2: Hypergraph representation of a tool switching problem instance with the corresponding tool-job incidence matrix (edges connecting two vertices are represented by line segments, edges connected three or more vertices are represented as sets).

2.3.2 Hypergraph based model

Most notably, for uniform tool sizes, Djellab *et al.* [9] proposed a hybrid heuristic that constructs a job sequence and then iteratively improves it using randomized choices. Here, they use a hypergraph representation of the tool switching problem. A hypergraph $\varrho = (X, E)$ consists of a finite set of vertices $X = x_1, ..., x_n$ and a set E multi-edges, where each multi-edge e_i connects a set of vertices (see figure 2.2 for example). The vertices in X represent jobs, and the edges in E represent tools. An edge contains the set of jobs that use a particular tool. The hypergraph can also be represented as an edge-vertex incidence matrix, which is exactly the same as tool/job incidence matrix used by Crama *et al.* in [4].

One important benefit of the Djellab heuristic is that its input includes a partial order for the job sequence. This is a very useful feature in practical scenarios, where certain jobs must be processed before others. The heuristics also considers non-uniform switching costs for tools, but the tools still are of identical size, so that no fragmentation of the magazine occurs. The problem input is then defined as a hypergraph $\psi = (X, E, C, <)$, where $C = (c_1, ..., c_m)$ gives the costs associated to each edge $e_i \in E$, and $<$ is a partial order on the set of vertices X representing the jobs.

An *injective projection* of $(X, <)$ is a permutation $\pi : X \rightarrow I_n = 1, ..., n$, where $\forall x, y \in X, x < y \Rightarrow \pi(x) < \pi(y)$. The injective projection $\pi(e_i)$ of the vertices in an edge e_i may be continuous, or it may contain one or more gaps. If the projection is continuous, the tool associated by e_i does not need to be switched, while the number of gaps in $\pi(e_i)$ represents the number tool switches for tool i. The number of gaps in a projection of an edge e is denoted by $t_\pi(e)$, where $t_\pi(e) \geq 0$ (see figure **??** for an example). The goal

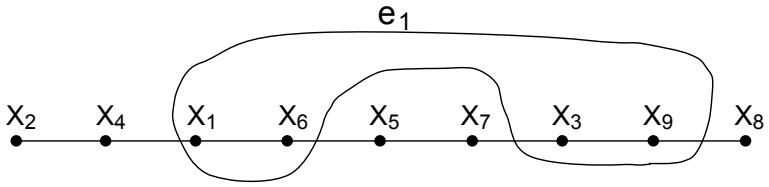

Figure 2.3: An injective projection of edge e_1 from the earlier hypergraph example. The projection results in one gap.

of the problem formulated in ([9]) is to find a projection π that minimizes the weighted sum of the gaps of edge projections: $T_\pi(E) = \sum_{e \in E} C(e) t_\pi(e)$, weight $C(e)$ expresses the cost of edge e. This is basically an alternate formulation of the matrix permutation problem.

The heuristic proposed in [9] consists of two procedures. The first procedure (called *Best Insertion*) iteratively inserts the vertices into a projection that satisfies the partial order given in the input. The order in which the vertices are considered is an input to the procedure. The insertion locations are determined by a greedy choice that minimizes a local cost function. In the second procedure (called *Iterative best Insertion*), the first procedure is called repeatedly using random orderings of vertices, while the partial order is completed with incompatible pairs (x, y) (*i.e.* pairs that have no defined order in the partial ordered set). The procedure stops when the partial order becomes complete (see Djellab *et al.* [9] for the details of the steps in these procedures).

2.4 Non-uniform tool sizes

The various heuristics described earlier have been used to solve tool switching problems, when the tool sizes are uniform. In those cases, the tools can be placed anywhere in the magazine, and the tool loading problem can be solved optimally for a given job sequence by the means of the KTNS procedure.

When the tool sizes are non-uniform, each tool may occupy one or more slots in the magazine, depending on the tool type. This makes the tool loading problem much more difficult due to the fragmentation of the magazine space. The decision version of the problem is now NP-complete (as shown by Crama *et al.* [5]). The job-sequencing of course remains NP-hard for non-uniform tool sizes also, since it is a super-problem of the uniform tool size problem, which is also NP-hard.

The non-uniform tool size problem was studied by Tzur *et al.* in [20] and by Matzliach *et al.* in [21] and [22]. Tzur *et al.* [20] proposed a hybrid algorithm called *Aladdin*, that uses the GENIUS procedure for produc-

ing tool sequences. They also introduced the KSTNS (Keep Smallest Tool Needed Soonest) procedure as a heuristics for the tool loading problem. Instead of creating a job sequence, the Aladdin procedure uses GENIUS algorithm to create a tool sequence, where the tool distances are given as $d(i,j) = N - |J_i \cap J_j|$, where, N is the number of jobs (as defined earlier), i and j are tools and J_i (J_j) is the set of jobs using i (j). For more details see [20].

In the present study *[P2]* we extend the heuristics introduced by Djellab *et al.* [9] to the case of non-uniform tool sizes. This extension is not at all trivial since one has to account for the magazine fragmentation. For the placement of tools in the magazine, a new heuristics called *Iterative Multi-Tool Manager (IMTM)* will be proposed. The algorithm is fast so that it can be used when evaluating job sequences in the *Iterative Best-Insertion* algorithm of Djellab *et al.* [9].

The IMTM algorithm adds tools to the magazine for each job by considering the magazine content from the previous job. If there is sufficient room to add a new tool in the magazine, the IMTM procedure selects the smallest free space where the tool fits, in this way leaving larger space fragments for later tools. If tools need to be removed, *tool removal cost* R is defined for each job j and tool t

$$R(j,t) = \begin{cases} c_t/dist(j,t), & \text{if } dist(j,t) > 0, \\ \alpha \cdot c_t, & \text{if } dist(j,t) = 0, \end{cases}$$

where α is an arbitrary constant greater than 2 to give the highest removal cost for tools used by the current job, and c_t the cost is associated with the switching of tool t (not necessarily equal to the tool size). The value of *dist* is defined as:

$$dist(j,t) = \begin{cases} 0, & \text{if } A(j,t) = 1, \\ m, & \text{if } A(j+m,t) = 1 \text{ and } A(j+i,t) = 0 \\ & \quad \text{for all } i \in [0,m), \text{ and } m > 0, \\ N, & \text{if } A(j+i,t) = 0 \text{ for all } i \in [0, N-j), \end{cases}$$

where A is the job-tool incidence matrix. The distance (*dist*) is 0 if the tool is used by job j, m if the next use of tool t in matrix A occurs m jobs after job i, and it is N (the number of jobs) if tool t is used no more in A after job j. A tool with the smallest removal cost R is selected for removal.

In the present study the IMTM procedure is used as a tool loading heuristic in combination with the IBI heuristics of Djellab *et al.* [9]. This algorithm (called *DG+*) can be used to solve the joint tool-loading and job-sequencing problem for multiple tool sizes. Our comparative experiments show that the algorithm performs very well in comparison with the Aladdin heuristics of Tzur *et al.* [20].

Chapter 3

Surface Mount Placement Machines

Surface mount device (SMD) placement machines are used in PCB manufacturing to place components onto a bare PCB. Ayob *et al.* [23] provide a detailed classification of SMD placement machines in their work. They group these machines into five categories: *dual-delivery*, *multi-station*, *turret-type*, *multi-head* and *sequential pick-and-place*. In these machines, the component placement operation begins by loading the PCB into the machine and identifying the so called "fiducial marks" on the PCB to properly position the PCB (Ayob *et al.* [23]). Then the electronic components are placed on the PCB in an order specific to the machine type and the optimization software for the control of the machine operations. In the final step, the PCB is moved out of the machines, and the components are soldered to the PCB.

In typical PCB assembly operations the SMD placement machines act as performance bottlenecks when considering the manufacturing efficiency. These machines are rather expensive, and the optimization of their operation can have a significant impact on the efficiency of the whole assembly process. Due to the differences in the operation principles of various SMD placement machine types, the optimization of a PCB assembly line requires modeling specific to each particular machine type (see Crama *et al.* [27]).

The SMD placement machines use component feeders (one or more, depending on the machine type) as an intermediate storage for efficient access of components required by the assembled PCB. The optimization of the feeder setup can have a significant impact on the efficiency of the production process. The actual component placements are done by one or more placement heads (again, depending on machine type), that position the components from the feeder into the PCB. These operation are influenced by multiple factors. The topology of the PCB determines the *traveling route* of the robotic arm from the feeder to the location on the PCB where the

component is placed. Component size and location may result in further constraints on the order in which components are placed.

Some SMD machine types allow the movement and rotation of the PCB table to assist a cartesian/gantry robotic arm in placing the component. Machines with multiple heads impose further constraints on the component route and placement order, to avoid collisions of the robotic arms. Due to these (and numerous other) characteristics of the various SMD placement machines, a unified realistic mathematical model becomes complicated to state and solve. Therefore, various aspects of the operation principles of these machines are addressed separately by mathematical models and tailored optimization heuristics have been proposed, resulting in better approximations of sub-problems of the manufacturing processes (see Leipälä and Nevalainen [25], Crama *et al.* [28] and [29], Gavish *et al.* [30], for example).

We next review the five categories of the SMD machines as classified by Ayob *et al.* in [23].

3.1 Dual-delivery machines

Dual-delivery machines have two feeder units and two component placement heads mounted on the sides of the PCB holding table. The placement operations alternate between the two heads to avoid collision; while one head picks a component from its associated feeder, the other head can move and place a component on the PCB. The optimization model for these types of machines must account for the constraint that the two heads cannot operate on the same area simultaneously. The control and optimization of these types of machines has been studied by Ahmadi *et al.* [31], [32], [33], Wilhelm *et al.* [35] and Choudhury [36].

3.2 Multi-station placement machines

Multi-station placement machines have several identical machine modules that are capable of working concurrently. The PCB is fixed to a pallet and moved through these modules by a conveyor (Csaszar *et al.* [37]). The modules work autonomously in parallel and each of them performs a subtask on a PCB. Configurable versions of this machine type have recently gained popularity.

Since these stations work concurrently but share a common conveyor system, the efficiency of the assembly process significantly depends on the synchronization of the conveyor steps. The overall assembly time of the machine is the sum of the maximum times of the stations in each step (see Csaszar *et al.* [37] and Grunow *et al.* [38]).

3.3 Rotary turret machines

Turret-type machines have a rotating turret with multiple placement heads (holding nozzles). The heads move between a fixed placement and pickup location. The rotating turret is at a fixed position, while the PCB holding board moves in X-Y directions and a feeder unit moves along the x-direction, so that the required components are below the placement head when they are picked up. These machine types are very fast (also called "chip-shooters"), but large mass forces cause mechanical stress and the size of the machines tends to be large.

Typically these machine types have rotating placement heads equipped with 12 to 24 nozzles, which can be changed on-the-fly (Ayob *et al.* [23]). The PCB holding table moves simultaneously (with the rotating turret) to position the next placement location under the placement head (Crama *et al.* [28]). As the turret rotates a nozzle holding a component from the pickup to the placement position, visual inspection of the component is made for orientation and diagnostics, and miss-oriented or invalid (depending on quality criteria) components are ejected. After the component has been placed onto the PCB, the rotating turret moves the head towards the pickup location, and the nozzles (in the placement head) are set up and reoriented for the next pickup operation. Parallel to this operation, the PCB table moves in X-Y direction to position the next placement point under the turret contact point.

In practice, the performance of the rotary turret machines strongly depends on the performance of PCB table movements. As such, in this case the optimization of the manufacturing process depends on the topology of the PCB. For example, components can be fed to the nozzles of the turret head in an order that reduces the total PCB movement time. This is done by minimizing a TSP instance on the placement points using the Chebychev distance (*i.e.* $max(|\Delta x|, |\Delta y|)$) that determines the PCB movement time (Francis *et al.* [39]). One must at the same time consider also the time needed for turret rotations and feeder movements.

3.4 Multi-head placement machines

Multi-head placement machines differ from the rotary turret machines in their component transportation mechanism. In multi-head placement machines, components are transported from the feeder to the placement location (on a non-moving PCB), by a gantry head. The head can be moved in X-Y directions by the aid of two step motors running on two orthogonal bars (gantries). This flexible machine design can handle various component packages (Bentzen *et al.* [41]).

The placement head is equipped with a set of nozzles suitable for the components (nozzle changes are required if the existing nozzles are not suitable). The head moves to the pickup location (a feeder unit), where a set of components is picked up by the nozzles. The head moves then on the X-Y plane of the PCB to place these components. The components are inserted on the PCB by moving the nozzle down in the Z-direction. The set of components that are picked up by the placement head determines a sub-tour of the PCB locations.

Just as in the case of turret machines, the performance of multi-head machines is strongly determined by the time it takes for the placement head to travel in the X-Y directions, and place the components (down-up movement) on the PCB. Thus, the operation of these machines can also be improved by modeling its control as a variation of the traveling salesman problem. For example, the order in which components are placed in a sub-tour determines the length of the tour (thus, the time taken by the robotic arm to move along the set of locations). In addition, grouping the components into sets will also have an impact on the quality of the sub-tours of those sets (see van Laarhoven *et al.* [42], Du and Lu [43], Sun and Lee [44], and Park and Kim [45]).

One aspect of the efficiency of these types of machines is the selection of nozzle assortments in the gantry head. Specific component types can be carried and placed onto the PCB by specific nozzle types, only. This results in the problem of selecting the right assortment of the nozzles to the placement head of the machine, since changing nozzles on-the-fly can be a cost-intensive operation. At each pick-and-place step, the machine head moves from the PCB area to the component feeder area, picks up components from the feeder, moves the head over the PCB, and places the components onto the PCB. The number of such steps depends on how many components can be carried at a step.

A trivial solution for this problem is to choose at least one nozzle for each component type, which gives a minimum necessary setup. Nevertheless, this may result in a large number pick-and-place steps to be performed. This motivates the search for a method that selects a nozzle assortment where the number of pick-and-place steps is minimized.

The overall efficiency of the process also depends on the topology of the PCB, that is, the locations of where the components are placed. For example, in one pick-and-place step it is worth selecting components that are located close to each other, thus reducing the placement head movements over the PCB.

3.5 Sequential pick-and-place machines

Sequential pick-and-place machines are similar to the multi-head machines, but they have only one nozzle in the head, thus a sub-tour can handle one component, only. In these machines, the head moves in the X-Y plane of the PCB, and after every placement it must move back to the feeder unit to pick a new component. Alternatively, both PCB table and the head can be stationary or movable (depending on machine type), see *e.g.* Ball and Magazine [46], T. Leipala and Nevalainen [47, 48], Fu and Su [49] and Ayob and Kendall [50] for literature.

Machines of *radial type* (which we address in our fourth and fifth papers) are a variation of pick-and place machines, and are used in the manufacturing of robust electronics devices. In these machines the components are transported to the assembly point by a component tape, and a robotic arm places them onto the PCB. The component tape is constructed online by a separate *feeder unit* (also called *sequencer*), that contains a set of slots storing component reels of various types. The component tape moves under the feeder, and components are *inserted* onto the tape. Depending on the component type, the insertion step may delay the tape movement due to the operation principle of the sequencer. The feeder is capable of placing multiple components at the same time, thus reducing the total delay caused by specific (wide) components. The order in which the components are placed to the PCB is fixed, requiring the feeder to insert the components in the same order.

Chapter 4

Summary of publications

This section gives a brief description of the five original publications included in this thesis. The first two publications deal with the use of hybrid techniques on various types of tool switching problems, both uniform and non-uniform instances. The third publication focuses on the problem of selecting nozzles for a gantry machine, and provides an optimal solution for the problem. The fourth and fifth publications study the problem of creating feeder setups for radial placement machines.

4.1 A note on the tool switching problem of a flexible machine

In this publication [P1], we study hybrid methods for the tool switching problem which arises in the metal-working industry, where numerically controlled flexible machines are used to manufacture parts. A form of this problem also occurs in the electronics industry, where the objective is to determine a sequence of PCB jobs so that the number of feeder setups is minimized. We propose a new hybrid method and compare it against existing heuristics in the literature. The new heuristics is based on existing methods (GENIUS by Hertz *et al.* [13]) by invoking repeated searches on different starting points. The scope of this study is to find useful methods for real production environments, by considering both the solution quality and execution time.

4.2 Minimizing the number of tool switches with tools of different sizes

We introduce a new heuristic algorithm for the combined problem of job ordering and tool loading with non-uniform tool sizes, and evaluate it against

existing heuristics in the literature. The new heuristics extends an existing hybrid method for uniform tool sizes introduced by Djellab *et al.* [9], and introduces a new storage management method to solve the magazine fragmentation problem. The combined heuristic is evaluated against the results of the *Aladdin* heuristics introduced by Tzur *et al.* in [20]. The experiments show that by employing the randomized iterative improvement steps of Djellab *et al.* [9] in combination with our new storage management method, significant improvements can be achieved on the quality of the results.

4.3 Selecting the nozzle assortment for a gantry-type placement machine

The third publication [P3] studies the problem of selecting nozzle assortments for gantry-type placement machines. These types of machines are extensively used in PCB manufacturing. They have a number of attributes that make them popular; including great flexibility, accuracy and moderate price. There are various configurations of gantry-type placement machines; here we focus on the single-arm, multi-head type, which can pick up multiple components and place them on the PCB within one pick-and-place phase.

The focus in this paper is on the sub-problem of nozzle assortment selection, regardless of PCB-component topology. We show that the problem can be solved efficiently and optimally by using the *MINMAX* resource allocation formulation of the *apportionment problem* (see Ibaraki *et al.* [54]). We also provide a simplified proof of the optimality for the specific problem instance of optimal nozzle assortment selection.

4.4 Construction of component tapes for radial placement machines

The fourth publication [P4] studies the problem of component tape construction using a component feeder unit. The component tapes are employed by machines of *radial type*, a variation of pick-and-place machines.

The objective of our study is to create a feeder assignment, that can produce a given component sequence and minimize the tape delay, caused by the insertion of specific component types. These component types demand an extra time factor caused by their large dimensions. The tape delay can be reduced by creating a feeder assignment, where two or more large component types can be inserted at once.

We show that the tape construction problem is closely related to the *turnpike problem* (see Redstone *et al.* [55]), a well known and extensively

studied geometric optimization problem. Whether the turnpike problem is NP-hard or polynomially solvable is presently not known. This motivated the search for an optimization heuristics to construct a better than trivial feeder assignment.

The paper provides a formal proof of the reduction of the turnpike problem to the feeder assignment problem. An integer programming formulation of the problem is also given along with a heuristic algorithm that can be used to construct a feeder assignment. The heuristic algorithm is evaluated and the results are compared against a naive feeder construction method. Results show that in cases where the component tape contains repetitive patterns of component sequences, a significant reduction of tape movement delay is obtained by the heuristic algorithm.

4.5 Organizing the operation of radial machine sequencers for multiple PCB-types

The fifth publication [P5] extends the results of the fourth article by studying radial placement machine setups where a single feeder assignment is used to process multiple PCB types. Each PCB type is manufactured in a different lot size, thus contributing with different amounts to the overall process delay. The objective of this study is to construct a feeder assignment that minimizes the tape insertion delay, considering that the same feeder assignment is used to manufacture multiple PCB types with different lot sizes. Obviously, the problem is an extension of the single-PCB problem instance studied in the previous paper. The article provides a modified integer programming formulation for the multi-PCB feeder assignment problem. Also, a non-trivial extension of the single-PCB heuristic algorithm to the multi-PCB problem is proposed. The extended heuristic is evaluated on various data sets against a naive heuristics. Similarly to the previous paper, the results show significant improvement when using the extended heuristic on component tapes that contain repetitive patterns.

Bibliography

[1] G. Gutin, A.P. Punnen, The Traveling Salesman Problem Problem and Its Variations, Kluwer Academicm Publishers, 2002.

[2] Bard, J.F., A Heuristic for Minimizing the Number of Tool Switches on a Flexible Machine, IIE Transactions, Vol.20, No. 4, pp. 382-391 (1988).

[3] Belady, L.A., A study of replacament algorithms for a virtual-storage computer. IBM Systems Journal, 5, 78-101 (1966).

[4] Y. Crama, A.W.J. Kolen, A.G. Oerlemans and F.C.R. Spieksma, Minimizing the number of tool switches on a flexible machine, International Journal of Flexible Manufacturing Systems 6 (1994) 33-54.

[5] Y. Crama, L.S. Moonen, F.C.R. Spieksma and E. Talloen, The tool switching problem revisited, European Journal of Operational Research 182 (2007) 952-957

[6] Garey, M.R., Johnson D.S. (1979). Computers and intractability. A guide to the theory of NP-completeness. W.H. Freeman and Company, New York.

[7] Jain, S., Johnson, M.E., Safai, F. (1996). Implementing setup optimization on the shop floor. Operations Research, 43(6), 843-851.

[8] Christopher S. Tang , Eric V. Denardo, Models arising from a flexible manufacturing machine, Part I: minimization of the number of tool switches, Operations Research, v.36 n.5, p.767-777, Sept/Oct 1988

[9] Djellab, H., Djellab, K., Gourgand, M. (2000). A new heuristic based on a hypergraph representation for the tool switching problem. International Journal of Production Economics, 64, 165-176.

[10] Fulkerson, D.R. and Gross, D.A., "Incidence Matrices and Interval Graphs," Pacific Journal of Mathematics, Vol. 15, No. 3, pp. 835-855 (1965).

[11] Möhring, R.H., "Graph Problems Related to Gate Matrix Layout and PLA Folding;' in Computational Graph Theory, G. Tinhofer *et al.* (eds.), Springer-Vedag, Wien, pp. 17-51 (1990).

[12] Kashiwabara, T. and Fujisawa, T., "NP-Completeness of the Problem of Finding a Minimum-Clique-Number Interval Graph Containing a Given Graph as a Subgraph," in Proceedings of the 1979 International Symposium on Circuits and Systems, pp. 657-660 (1979)

[13] A. Hertz, G. Laporte, M. Mittaz, K.E. Stecke, 1998, "Heuristics for Minimizing Tool Switches Over Time on a Flexible Machine", IIE Transactions 30/8, 689-694.

[14] Gendreau, M., Hertz, A. and Laporte, G. (1992) New insertion and post-optimization procedures for the traveling salesman problem. Operations Research, 40, 1086-1094.

[15] K. Salonen, Cs. Ráduly-Baka, O. Nevalainen. A note on the tool switching problem of a flexible machine. Computers & Industrial Engineering 50(4), Aug 2006, 458-465.

[16] Cs. Ráduly-Baka, T. Knuutila, O. Nevalainen. Minimizing the number of tool switches with tools of different sizes. In Proceedings of the 5th International Conference on Technology and Automation 2005 (ICTA'05), Oct 2005.

[17] Cs. Ráduly-Baka, T. Knuutila, M. Johnsson, O. Nevalainen. Selecting the Nozzle Assortment for a Gantry-Type Placement Machine. OR Spectrum, 30, 493-513, 2008.

[18] Cs. Ráduly-Baka, T. Knuutila, M. Johnsson, O. Nevalainen. Construction of Component Tapes for Radial Placement Machines. Computers & Operations Research, Electronic edition():1-35, Nov 2009.

[19] Cs. Ráduly-Baka, T. Knuutila, M. Johnsson, O. Nevalainen. Organizing the operation of radial machine sequencers for multiple PCB-types. To appear in Computers & Industrial Engineering.

[20] M. Tzur and A. Altman, Minimization of tool switches for a flexible manufacturing machine with slot assignment of different tool sizes, IIE Transactions (2004) 36, pp. 95-100.

[21] B. Matzliach and M. Tzur, Storage management of items in two levels of availability, European Journal of Operational Research (2000) 121 pp. 363-379.

[22] Matzliach, B., and Tzur, M. (1998). The online tool switching problem with non-uniform tool size. International Journal of Production Research, 36(12), 3407-3420.

[23] Masri Ayob and Graham Kendall. A Survey of Surface Maount Device Placement Machine Optimisation: Machine Classification, European Journal of Operational Research (2005), Vol. 164, pp.609-626.

[24] Duman, E., Or, I., 2004. Precedence constrained TSP arising in printed circuit board assembly. International Journal of Production Research 42 (1), 6778.

[25] Leipälä, T., Nevalainen, O., 1989. Optimization of the movements of a component placement machine. European Journal of Operational Research 38, 167177.

[26] Shih, W., Srihari, K., Adriance, J., 1996. Expert system based placement sequence identification for surface mount PCB assembly. International Journal of Advanced Manufacturing Technology 11, 413424.

[27] Crama, Y., Klundert, J. van de, Spieksma, F.C.R., 2002. Production planning problems in printed circuit board assembly. Discrete Applied Mathematics 123, 339361.

[28] Crama, Y., Flippo, O.E., van de Klundert, J.J., Spieksma, F.C.R., 1996. The component retrieval problem in printed circuit board assembly. International Journal of Flexible Manufacturing Systems 8, 287312.

[29] Crama, Y., Flippo, O.E., Klundert, J.J.V.D., Spieksma, F.C.R., 1997. The assembly of printed circuit boards: A case with multiple machines and multiple board types. European Journal of Operational Research 98, 457472.

[30] Gavish, B., Seidmann, A., 1988. Printed circuit boards assembly automation-formulations and algorithms. In: Mital, A. (Ed.), Recent Developments in Production Research. Elsevier Science, Amsterdam, pp. 624635.

[31] Ahmadi, J., Grotzinger, S., Johnson, D., 1988. Component allocation and partitioning for a dual delivery placement machine. Operations Research 36, 176191.

[32] Ahmadi, J., Grotzinger, S., Johnson, D., 1991. Emulating concurrency in a circuit card assembly system. International Journal of Flexible Manufacturing Systems 3 (1), 4570.

[33] Ahmadi, J., Ahmadi, R., Matsuo, H., Tirupati, D., 1995. Component fixture positioning for printed circuit board assembly with concurrent operations. Operations Research 43, 444457.

[34] W. E.Wilhelm *et al.* A model to optimize placement operations on dualhead placement machines. Discrete Optimization (2007), Vol. 4, Issue 2, pp. 232-256.

[35] W. E.Wilhelm *et al.* A model to optimize placement operations on dualhead placement machines. Discrete Optimization (2007), Vol. 4, Issue 2, pp. 232-256.

[36] N.D. Choudhury *et al.* Process planning for circuit card assembly on a series of dual head placement machines. European Journal of Operational Research 192 (2007) 626-639.

[37] P. Csaszar *et al.* Optimization of a high-speed placement machine using tabu search algorithms. Annals of Operations Research 96 (2000) 125-147.

[38] M. Grunow *et al.* Component allocation for printed circuit board assembly using modular placement machines. Int. Journal of Production Research (2003), Vol. 41, No. 6, 1311-1331.

[39] Francis, R.L., McGinnis, L.F., White, J.A., 1992. Facility Layout and Location: An Analytical Approach. Prentice-Hall, Englewood Cliffs, NJ.

[40] Mika Hirvikorpi, 2005. On the Tactical Level Production Planning in Flexible Manufacturing Systems. TUCS Dissertations No. 66, November 2005.

[41] Bentzen, B., 2000. SMD placement, in the SMT in FOCUS. (September 25, 2002).

[42] P.J.M van Laarhoven and W.H.M. Zijm. Production Preparation and Numerical Control in PCB Assembly, The International Journal of Flexible Manufacturing Systems, 5 (1993): 187-207.

[43] Xuan Du and Zongbin Li. New model and Hybrid Genetic Algorithm for Component Placement of Multihead Gantry Mount Machine. Int. Conference on Industrial Engineering and Engineering Management, 2008, pp. 790-794.

[44] Dong-Seok Sun and Tae-Eog Lee. A Branch-and-price algorithm for placement routing for a multi-head beam-type component placement tool. OR Spectrum, 2008, Vol 30, pp 515-534.

[45] Tae-Hyoung Park and Nam Kim. A Dynamic Programming Approach to PCB Assembly Optimization for Surface Mounters. International Journal of Control, Automation and Systems, 2007, Vol. 5, No. 2, pp.192-199.

[46] M.O. Ball and M.J. Magazine. Sequencing of insertions in printed circuit board assembly. Operations Research (1998), Vol 36, No. 2.

[47] T. Leipälä *et al.* Job grouping in surface mounted component printing. Robotics and Computer-Integrated Manufacturing (1999) v.15 pp. 39-49.

[48] T. Leipälä *et al.* Optimization of the movements of a component installation automaton, European Journal of Operational Research (1989), Vol. 38, pp. 167-177.

[49] Hsin-Pin Fu and Chao-Ton Su. A comparison of search techniques for minimizing assembly time in printed wiring assembly. Int. Journal of Production Economics 63 (2000), 83-90.

[50] M. Ayob and G. Kendall. A triple objective function with a Chebychev dynamic pick-and-place point specification approach to optimise the surface mount placement machine. European Journal if Operational Research (2005), Vol. 164, Issue 3, pp. 609-626.

[51] M. Zhou and M. C. Leu. Petri net modeling of a flexible assembly station for printed circuit boards. In Proceedings of the 1991 IEEE International Conference on Robotics and Automation, pages 25305, Sacramento, CA, Apr. 1991.

[52] J. Smed, M. Johnsson, T. Johtela, O. Nevalainen, Techniques and Applications of Production Planning in Electronics Manufacturing Systems, Technical Report TUCS-TR-320, 1999.

[53] Mika Johnsson, Operational and tactical level optimization in printed circuit board assembly, TUCS Dissertations, No. 16, 1999. Technical Report TUCS-TR-320, 1999.

[54] Ibaraki, T. and N. Katoh (1988), Resource Allocation Problems: Algorithmic Approaches, The MIT Press, Cambridge, Massachusetts, ISBN 0-262-09027-9.

[55] Joshua Redstone, Walter L. Ruzzo. Algorithms for a Simple Point Placement Problem. Algorithms and Complexity, 4th Italian Conference, CIAC 2000, Rome, Italy, March 2000, pp. 32-43.

[56] Paul Lemke, Steven S. Skiena, and Warren D. Smith. Reconstructing sets from interpoint distances. Tech. Rept. DIMACS-2002-37, 2002.

[57] Z. Zhang. An exponential example for partial digest mapping algorithm, J. Computational Biology 1,3 (1994) 235-239.

Chapter 5

Publications

Publication I

Salonen, K., Raduly-Baka, Cs., Nevalainen, O. A note on the tool switching problem of a flexible machine. *Computers & Industrial Engineering 50(4)*, Aug 2006, 458-465.

Available online at www.sciencedirect.com
ScienceDirect
Computers & Industrial Engineering 50 (2006) 458–465
computers & industrial engineering
www.elsevier.com/locate/dsw

A note on the tool switching problem of a flexible machine

Kari Salonen *, Csaba Raduly-Baka, Olli S. Nevalainen

Department of Information Technology and Turku Centre for Computer Science (TUCS), University of Turku, FIN-20014 Turku, Finland

Available online 8 August 2006

Abstract

The problem of minimizing the total number of tool switches for a numerically controlled flexible machine is considered. A set of parts is to be processed with the machine. Each part needs a set of tools which should reside in the magazine of the machine at the moment of processing. Because of the limited capacity of the magazine, tools must be switched and the objective is to minimize the amount of this work. We propose an algorithm which tries to avoid sticking to a local minimum by repeated searches from different initial starting points which are created by repeated construction of super parts from parts with similar tools. The proposed algorithm and a number of efficient heuristics presented in the literature are empirically tested by both random test problems and real production data. The new algorithm performs well when considering the tradeoff between solution quality and running time.

Keywords: Tool switching; Flexible manufacturing; Heuristics; Minimum setup

1. Introduction

We consider a production situation where a set of parts is processed by a single numerically controlled flexible machine which is capable of using several different tools at the same run. The tools are stored in a tool magazine from which they are picked up when needed. Each part is processed by a non-empty subset of all possible tools. All the tools needed by a part should be in the magazine at the moment of processing. It is supposed that the capacity of the magazine is large enough to hold the tools of each particular part and possibly some extra tools but the union of the tools needed by all parts is too large to fit in the magazine at the same time.

It is supposed that each tool occupies one slot of the magazine, tools can be stored in whichever slot, the lifetime of each tool is infinite and the change cost is the same (constant) for all tools. The processing time for different parts includes the setup time for inserting the proper tools and the actual manufacturing time for making the necessary operations with the tools on the part. This later time component is supposed to be constant, i.e., it does not depend on the position of the tools in the magazine or on the processing sequence of the parts. On the contrary to that, the setup time depends linearly on the number of tools inserted.

* Corresponding author. Fax: +358 2 333 8600.
E-mail address: kari.salonen@it.utu.fi (K. Salonen).

0360-8352/$ - see front matter
doi:10.1016/j.cie.2004.11.002

The objective in the *tool switching problem* (TSwP) is to determine a sequence of the parts for which the total time of tool switches is minimal. The above discussion naturally implies that the solution to the problem gives (with the assumptions made) a time optimal processing sequence of the parts. It is *NP*-hard to solve the problem optimally (Tang & Denardo, 1988) but numerous efficient heuristics have been proposed to it (Al-Fawzan & Al-Sultan, 2002; Bard, 1988; Barnea & Sipper, 1993; Crama, Koleen, Oerlemans, & Spieksma, 1994; Djellab, Djellab, & Gourgand, 2000; Follonier, 1994; Günther, Gronalt, & Zeller, 1998; Hertz, Laporte, Mittaz, & Stecke, 1998; Shirazi & Frizelle, 2001).

While the tool switching problem has been originally stated for metalworking industry (see Crama et al., 1994) our interest is on a quite different context. In electronics industry the principal task is the insertion and fixation of electronic components on *printed circuit boards* (PCB types) by the means of automated high speed component placement machines. In this context, the magazine (called feeder) is capable of holding a set of electronic components (typically stored in component tape reels) which are placed on the PCB. Now the problem of finding a sequence of the assembly jobs (i.e., PCBs) giving the *minimum setup* work for changing the component reels in the feeder is identical to the tool switching problem. It is a common practice in PCB assembly industry to form batches of identical PCB types and handle each of them at a time. We therefore suppose this kind of manufacturing and speak shortly of a PCB when the question is actually of a batch of PCBs of the same type (or more exactly with the same set of components to be assembled).

The tool switching problem of PCB assembly has been studied by several authors, see e.g., Barnea and Sipper (1993), Dillon, Jones, Hinde, and Hunt (1998), Rajkumar and Narendran (1997), and Günther et al. (1998). There are, however, some details which cause difficulties in the mapping between the TSwP and the PCB assembly case:

1. The width of the components (i.e., component tape reels) may vary making the change operations more difficult, see Günther et al. (1998), Jain, Johnson, and Safai (1996), Hirvikorpi, Salonen, Knuutila, and Nevalainen (2006).
2. The component-slot mapping may have a significant impact on the component placement time so that the original assumption of constant processing time is not valid any more. This difference should be considered with care especially for large PCB batches.
3. The number of feeder setup instances may be the dominating time factor so that one should include the number of setup instances and the number of feeder changes in the objective function, see Crama et al. (1994), Matzliach and Tzur (1998), Matzliach and Tzur (2000), Smed, Salonen, Johnsson, Johtela, and Nevalainen (2003).
4. Duplication of some frequently used components may be advantageous for the overall processing time.
5. The use of two different setup strategies at the same time, permanent setup and changing setup, may improve the operation control of the shop floor, see Smed, Johnsson, Puranen, Leipälä, and Nevalainen (1999).
6. The feeder unit may consist of several subunits (feeder banks) the borders of which may not be used (Smed et al., 1999).

Our interest to the tool switching problem originates from the previous experience on the third point mentioned above. On the request of our industrial partner we were originally developing a practical software system for grouping PCB assembly jobs to minimal number of groups such that a feeder change was unnecessary inside each group (*job grouping problem*) (Smed et al., 1999). We then observed that a more realistic model must weight the number of tool switches and reel change occasions by factors which depend on the average unit time for these operations. It is therefore possible to select these factors freely in our heuristics so that in an extreme case it acts as a pure tool switching algorithm. This algorithm works straightforwardly by first determining a good (or even optimal) grouping of the PCBs and then sequencing them. To our surprise the method, now and then, outperformed the original very efficient tool switching algorithm by Hertz et al. (1998). The observation was unexpected because a tool switching algorithm is searching for a solution from the original problem space spanned by all the permutations of the PCBs and the grouping of PCBs decreases the number of possible solution candidates by generating artificial super-PCBs from each group which are then sequenced in a normal way. A natural reason for this strange result lies in the suboptimality of the sequencing method

used in tool switching algorithms. We therefore reconsidered the sequencing method in hope to reach even better results for the TSwP by first grouping the PCBs, see Salonen, Smed, Johnsson, and Nevalainen (2006) for a preliminary draft of the idea.

In the present paper, we take a still more general approach. Instead of one shot processing, where the grouping and sequencing is performed only once, we now iterate the grouping – sequencing steps. In order to avoid identical groupings we use the hierarchical job grouping technique of Leon and Peters (1998) stepwise. This algorithm iterates the merge of two groups as long as the result remains feasible. Our idea is to sequence the groups after each merge step in order to search for the solution using a broader front in the solution space.

The rest of the paper is organized as follows. In Section 2, we add a preprocessing phase to a tool switching algorithm (GENIUS) by Hertz et al. (1998). A comparison of the proposed variant of the heuristics and some existing tool switching algorithms is given for the test data of Crama et al. (1994) and Smed et al. (2003) in Section 3. Concluding remarks are made in Section 4.

2. Tool switching heuristics

2.1. Utilization of the result of job grouping

We propose a joint heuristics – *Grouping with Minimum Setup Algorithm* (GMSA3), for the tool switching problem by using the iterative group minimization technique of Salonen et al. (2006) and the setup minimization algorithm GENIUS of Hertz et al. (1998).

The TSwP can be defined as follows. Suppose that a set of N parts, M different tools with infinite life times and a tool magazine with a capacity of C slots are given. Each tool occupies a single magazine slot. The need for using tool i ($1 \leqslant i \leqslant M$) when processing part j ($1 \leqslant j \leqslant N$) is expressed by the tool-part binary matrix $\{a_{ij}\}$, where $a_{ij} = 1$ if and only if tool i is needed for part j. Thus, part j needs a given subset of tools in its processing. A constant cost is caused by the insertion of a tool or switching of two tools (including removal of a tool from the magazine and insertion of a new tool). The task in TSwP is to form such a permutation Π^* of the parts and a change program for the tools in the magazine that all parts can be processed (i.e., the necessary tools are in the magazine when starting the processing of a new part) and the total number of tool changes (insertions and switches) S^* will be minimal.

Heuristic (GMSA3) accepts M,N,C and a_{ij} ($i = 1\ldots M; j = 1\ldots N$) as its input and determines the best possible approximation of Π^* and S^* as the output. The tool change program is determined for each fixed permutation of the parts implicitly by the Keep Tool Needed Soonest (KTNS)-rule, which has been shown to be optimal for tools of constant widths and constant change costs Crama et al. (1994).

GMSA3 works as follows:

1. Solve the tool switching problem giving Π^* and S^* by using MSAGenius, see Section 2.2. Let $k = 0$.
2. While $k < n_{\text{merge}}$ perform Steps 3 to 4.
3. Merge two parts i and j (by forming a super part) in such a way that the feeder capacity does not exceed:
   (a) Calculate *Jaccard's similarity coefficient* $s_{ij} = \frac{|E_i \cap E_j|}{|E_i \cup E_j|}$ (sets E_i and E_j denote the tools of the parts i and j for each part pair (i,j)).
   (b) Let (i^*, j^*) be the part pair with the highest similarity coefficient $\max(s_{ij})$ so that the merge is feasible (i.e., the feeder capacity is not exceeded).
   (c) Merge the parts i^* and j^* so that the super part contains the tools $|E_i \cup E_j|$.
   (d) Let $N \leftarrow N - 1$.
4. Solve the tool switching problem by GENIUS for the remaining parts giving Π and S (see Section 2.2). Let $k \leftarrow k + 1$. If $S < S^*$ then update $\Pi^* \leftarrow \Pi$ and $S^* \leftarrow S$.

The parameter n_{merge} determines the maximal number of merge operations. By setting $n_{\text{merge}} \leftarrow 0$ the merge operations are totally omitted and we get heuristic MSAGenius.

The GENIUS algorithms of Hertz et al. (1998) transform the TSwP at step 4 to a special TSP *(travelling salesman problem)* by treating the parts as cities to be visited and defining the function

$d_2(i,j) = |E_i \cup E_j| - |E_i \cap E_j|$ as the distance between two cities (parts) i and j. It is easy to see that $d_2(i,j)$ is actually an upper bound for the switching cost when proceeding from part i to j. The use of this distance function is beneficial because it is symmetric and therefore facilitates the solution of the TSP corresponding to the actual TSwP. The heuristic has worked well in the tests by Hertz et al. (1998). To keep the resent paper self contained we next briefly recall the ideas of GENIUS.

GENIUS is in fact a general TSP heuristic (and not restricted to TSwP) working for both symmetric and asymmetric problems. It consists of two parts, a construction heuristic, called GENI, and an improvement heuristic US which is performed as a post processing phase of the first heuristic (see Gendrau, Hertz, & Laporte, 1992).

GENI starts with three randomly selected vertices and inserts iteratively a new vertice k into the current tour so that the tour is reconstructed according to a local optimization procedure in a special neighbourhood of p closest vertices. Reorganization of the existing TSP path may cause reversion of some parts of the path and change of several edges. The algorithm thus unites the insertion of new vertice and local tour optimization. The insertions are iterated until all parts have been processed. The time complexity of GENI is $O(np^4 + n^2)$, where n is the number of different parts and p is a parameter giving the neighborhood size of vertices to be inserted.

US removes by the reverse GENI operation a vertice and then reinserts it again. The removing and reinsertion operations are repeated until no improvement is observed. As noted above, GENIUS solves a specially defined TSP when it is applied in the context of TSwP. The algorithm thus produces a final sequence of parts. On the basis of the distance function d_2, the consecutive parts on the path are similar with respect to the tools used by them. The tool switching decisions are finally made by the KTNS-rule after the final path has been constructed by the means of US. It is clear from the operation principle of US that, the asymptotic analysis of GENIUS cannot be done deterministically in terms of n and p.

MSAGenius is the base heuristic of GMSA3. It is multistart algorithm which repeats GENIUS (adapted to the TSwP) N times by taking each vertice (here part) as a starting point. The heuristic then improves the solution with a 2-opt heuristic (with d_2) and uses the KTNS-method when finally assigning tools to the magazine for the fixed sequence found in this way. The neighborhood size is set to $p = 3$ in order to make a tradeoff between time and quality.

2.2. Other heuristics

In addition to the GMSA3 we will report results for several other efficient heuristics solving the TSwP. These algorithms have been chosen on their competiveness as judged from previous experiments with them.

- GENIUS, as described above. The neighborhood size p is 6.
- GENIUS* differs from the GENIUS in that the possibilities in insertions are evaluated by KTNS at each iteration (instead of d_2). This has been shown to improve the results significantly.
- MSAGenius as described above.
- MSANI (Smed et al., 2003) is just like MSAGenius, but it uses NI (*nearest insertion*) as the TSP construction algorithm (see Lawler, Lenstra, Rinnooy Kan, & Shmoys, 1985 & Lenstra & Aarts, 1997) instead of GENIUS. NI takes a subtour of k nodes at iteration k and chooses the closest node r which is not in the tour. After that, r is inserted between arc (i,j) in the subtour so that the cost $c_{ir} + c_{rj} - c_{ij}$ is minimized. The time complexity of the algorithm is $O(n^2)$.
- MSAFI (Smed et al., 2003) uses FI (*farthest insertion*) as the TSP construction algorithm (see Lawler et al., 1985 & Lenstra & Aarts, 1997). The algorithm works like MSANI but it chooses the farthest node instead of the nearest.
- MAPE1, the matrix permutation method introduced by Djellab et al. (2000) with parameter value $A = 1$, where A is the number repeating the procedure. The method relies on the nice observation (Crama et al., 1994) that the tool switching problem can be stated as a matrix reorganization problem where the objective is to minimize the number of 0-blocks in the reordered tool-part matrix. The columns of the matrix state the sequence of the parts and a block of 0-elements in a row means that the parts in the corresponding subsequent columns do not need the tool given by the row index. As mentioned above, it is supposed in TSwP

that all parts can be processed, i.e., the tool magazine can hold the tools of each particular part. On the other hand, all the tools do not fit the magazine simultaneously. This means that one should search for a permutation of the columns which minimizes the total number of 0-blocks in the matrix.
The algorithm has two advantages. First, it includes a natural way of including the different change costs to the ordering method. Second, it is possible to give a set of precedence constraints to some of the job pairs in a simply way if necessary. The method was selected to the comparison on the basis of its radically different method of formulating the problem and its high quality results with known test problems of Crama et al. (1994). The original code of the matrix permutation method was not available to us and we therefore implemented it on the basis of pseudo code in Djellab et al. (2000). The complexity of MAPEx heuristic is $O(mn^4)$, where m is the number of components and n is the number of parts.

- MAPE50, as above but with $A = 50$.

In addition to these algorithms, there are several other heuristics proposed in the literature for the TSwP. Among these one should mention tabu search algorithms introduced by Follonier (1994), and Al-Fawzan and Al-Sultan (2002). Because their implementations were not available to us we were forced to omit them from comparisons.

3. Computational experiments and results

3.1. Test problems

We used two sets of test problems:

- *Test problems 1:* Crama et al. (1994) generated 160 instants of random test problems of different sizes (M,N,C). There are 10 instances of each size and the number of different tools varies randomly between *min* and *max* for each repetition, see Table 1. The tools of the parts are randomly selected so that duplicates are omitted.
  As noted by Crama et al. (1994) the instances (10,10,4), (20,15,6), (40,30,15), and (60, 40, 20) are "dense" in the sense that the small capacity does not leave many choices to the sequencing. The problems (10,10,7), (20,15,12), (40,30,25), and (60,40,30) represent the other end of the scale being "sparse". Note that Hertz et al. (1998) used the same method for generating their test problems, but using different random numbers.
- *Test problems 2:* The test problems of Smed et al. (2003) include three different data sets, where the number of different parts are 20, 30, 40, respectively. Each data set has 100 different problem instances, from real-world production data in PCB assembly. Each PCB type contains from 5 to 80 different component types (see Fig. 1).

The two data sets differ with respect to the magazine (or feeder size) and number of tools (or components) per part (or PCB).

Table 1
Parameter settings of different problem instances (M,N,C)

(M,N)	C_1	C_2	C_3	C_4	*min*	*max*
(10,10)	4	5	6	7	2	4
(20,15)	6	8	10	12	2	6
(40,30)	15	17	20	25	5	15
(60,40)	20	22	25	30	7	20

M is the total number of different tools, N is the number of different parts and C is the capacity of the tool magazine. *min* and *max* determine a range of different tools in each part of the problem instance.

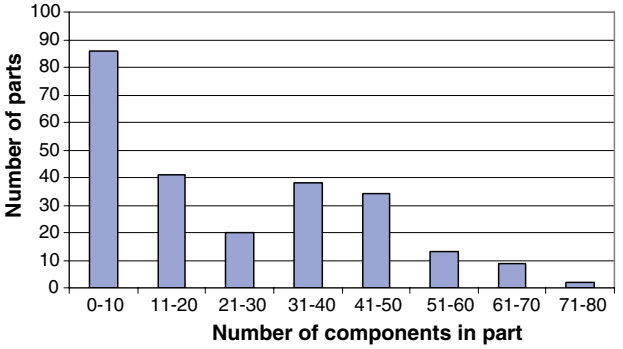

Fig. 1. Distribution of the number of components in 246 PCB assembly jobs by Smed et al. (2003).

Table 2
Average number of tool switches for randomly generated test problems by Hertz et al. (1998) (test problems 1)

Algorithm	Number of different parts; capacity of tool magazine																
	10;4	10;5	10;6	10;7	15;6	15;8	15;10	15;12	30;15	30;17	30;20	30;25	40;20	40;22	40;25	40; 30	Sum
GENIUS*[a]	12,6	**10,8**	**10,1**	**10,0**	27,2	22,1	**19,8**	**19,2**	103,3	87,5	71,1	54,3	205,6	182,1	154,6	121,9	1112,2
GMSA3	12,7	10,9	**10,1**	**10,0**	27,8	22,7	20,5	**19,2**	108,4	93,3	76,1	58,4	216,7	192,0	164,0	131,7	1174,5
MSAGenius	12,9	11,0	10,2	**10,0**	28,5	23,4	20,9	19,5	109,9	93,7	77,8	59,7	217,5	192,9	165,1	133,1	1186,1
GENIUS	13,4	11,5	10,4	**10,0**	29,5	24,0	21,1	19,7	111,7	97,3	79,8	61,7	218,9	196,5	169,9	137,4	1212,8
MAPE1	13,8	12,2	10,6	10,1	29,9	24,5	21,2	19,5	115,8	99,4	80,3	60,5	228,1	202,1	171,9	137,9	1237,8
MAPE50	13,4	11,8	10,4	**10,0**	28,8	23,2	20,5	19,3	112,2	96,4	77,5	58,0	223,0	198,1	166,9	132,6	1202,1
MSANI	13,0	11,0	10,2	**10,0**	28,2	23,3	20,6	19,3	111,3	95,5	78,2	60,1	218,1	194,5	167,0	133,6	1193,9
MSAFI	12,8	11,1	10,3	**10,0**	28,4	23,1	20,7	19,3	110,0	94,1	77,6	59,6	214,1	190,6	164,5	132,6	1178,8
Hertz best of all[a]	**12,5**	**10,8**	**10,1**	**10,0**	**26,9**	**22,0**	**19,8**	**19,2**	**102,0**	**85,9**	**69,4**	**53,6**	**203,2**	**179,0**	**152,5**	**120,9**	**1097,8**

Best solutions are indicated in bold font.
[a] Results given by Michel Mittaz, see Hertz et al. (1998).

3.2. Computational results

Table 2 gives a summary of test runs with the test problems 1. When comparing the different versions of GENIUS, GENIUS* gives clearly better solutions on an average. The case $(N,C) = (10,7)$ makes an exception where the solutions are the same. A remarkable drawback of GENIUS* is its very large running time (see Hertz et al., 1998). The running time of MSAGenius is longer than that of GENIUS, because GENIUS is iterated N times in MSAGenius with different starting nodes. On the other hand, the neighborhood size was only 3 in MSAGenius instead of 6 in GENIUS. MSAGenius improved the solution in comparison to GENIUS significantly, see for example case $(N,C) = (30,17)$ in Table 2 where the averages are 93.7 and 97.3, respectively. The increase of the neighborhood size p from 3 to 6 had no significant impact on the solution quality, whereas the running time increased significantly. Interestingly enough, GMSA3 gave on an average even better results than MSAGenius, although the running times of GMSA3 are still far from the times of GENIUS*. MSAFI found better solutions than MSAGenius, but not so good as GMSA3, see Tables 2 and 3. Comparing the results of GMSA3 and MAPE50 for real production data (Table 3), we notice that they give the same overall solution quality, but the running times of MAPE50 were on an average about five times longer than for GMSA3. In addition to this, MAPE50 was implemented with C++ and GMSA3 with Java: in performance tests, Java was 1.4 times slower than C++. Our MAPE1 and MAPE50 implementations did not give exactly the same results as reported in the research by Djellab et al. (2000). Differences are not systematic and they are small. We therefore think that the differences are due to different pseudo random generators used in the codes.

Table 4 shows on what iteration round the best results of GMSA3 were found. When the best result was found at the first iteration, GMSA3 couldn't improve the result of MSAGenius at all. For example, test set 2

Table 3
Average number of tool switches for real production data (test problems 2)

PCBs; capacity	MAPE1		MAPE50		MSAGenius		GMSA3		MSANI		MSAFI	
	Cost	Time/s	Cost	Time/s	Cost	Time/s	Cost	Time/s	Cost	Time/s	Cost	Time/s
20; 80	213,3	0,2	**210,3**	7,9	212,2	0,3	211,3	1,5	212,5	0,2	211,9	0,1
20; 120	**207,7**	0,2	**207,7**	9,0	**207,7**	0,3	**207,7**	1,5	**207,7**	0,2	**207,7**	0,1
30; 80	255,5	0,6	**249,9**	27,5	251,8	1,0	250,3	5,7	254,6	0,6	252,2	0,6
30; 120	**236,6**	0,6	**236,6**	31,2	**236,6**	1,0	**236,6**	5,7	236,7	0,6	**236,6**	0,6
40; 80	305,3	1,3	296,1	65,9	299,1	2,4	**295,6**	15,0	303,6	1,9	298,9	1,8
40; 120	264,2	1,5	**263,2**	77,0	263,6	2,4	263,3	15,1	263,8	1,9	263,6	1,8
Sum	1482,6	4,4	**1463,8**	218,5	1471,0	7,4	1464,8	44,5	1478,9	5,4	1470,9	5,0

The computation times (in seconds) on Dell machine (800 MHz, Intel Celeron). Best solutions are indicated in bold font.

Table 4
The number of times that GMSA3's best result is found at the given iteration round

PCBs; capacity	The number of times best result of GMSA3 found at the iteration round							
	1	2	3	4	5	6	7	8
20; 80	77	12	7	1	2	1	0	0
20; 120	100	0	0	0	0	0	0	0
30; 80	51	17	8	5	10	3	5	1
30; 120	100	0	0	0	0	0	0	0
40; 80	28	15	13	11	8	11	3	11
40; 120	92	3	3	0	0	1	0	1

The total number of runs was 600 (test problems 2).

includes 100 runs of 30 PCBs with capacity 80 (row, 30; 80) where GMSA3 got at the first iteration round its best result 51 times and gave 49 times better result than MSAGenius. When increasing the capacity to 120 (row, "30; 120"), the problems became much easier to solve. Therefore all the best results of GMSA3 originated from the first iteration round and there were no differences between the results of GMSA3 and MSA-

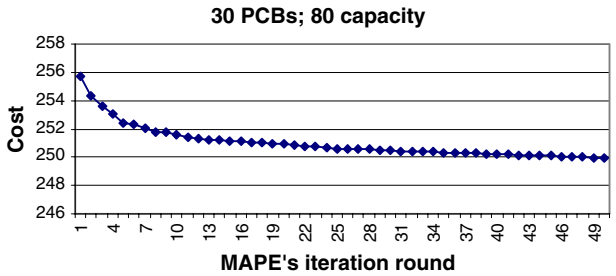

Fig. 2. Average cost (number of tool switches) of MAPEx for the case of 30 PCBs and capacity of 80 magazine slots (test problems 2) as a function of the number of iterations.

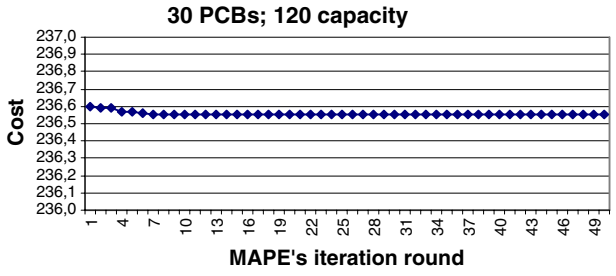

Fig. 3. Average cost (number of tool switches) of MAPEx for the case of 30 PCBs and capacity of 120 magazine slots (test problems 2) as a function of the number of iteration rounds.

Genius (236.6), see Table 3. Figs. 2 and 3 show, how the average results of MAPEx progress at the iterations 1 to 50 for the cases 30 PCBs (capacities 80 and 120). The first iteration gives results for MAPE1 and the last for MAPE50. Like in GMSA3, using several iterations in the cases with low capacity (80), MAPEx can improve its results (Fig. 2), but the iterations are waste of time for higher capacities (120).

4. Concluding remarks

GMSA3 relies on the simple idea of merging parts prior to the application of a sequencing heuristics. It turns out that this often gives better solutions than the sole application of the sequencing heuristics. The algorithm is fast even for more complex cases of real production data from PCB assembly and it is therefore a good candidate for the practical heuristics of the tool switching problem. Our results, however, show that the new method can not beat a high quality tool switching algorithm (GENIUS*) which on the other hand suffers from the unpractically large running time.

It was supposed here that all tools are of the same width and their location in the magazine is unrestricted. The case of unequal tool widths has been considered by few researches, only. More work in this field should be done because the component reels in PCB assembly are commonly of different widths and their optimal change strategy is complicated to solve.

References

Al-Fawzan, M. A., & Al-Sultan, K. S. (2002). A tabu search based algorithm for minimizing the number of tool switches on a flexible machine. *Computers & Industrial Engineering, 44*, 35–47.

Bard, J. F. (1988). Heuristic for minimizing the number of tool switches on a flexible machine. *IIE Transactions, 20*(4), 382–391.

Barnea, A., & Sipper, D. (1993). Set-up reduction in PCB automated assembly. *Computer Integrated Manufacturing Systems, 6*(1), 18–26.

Crama, Y., Koleen, A. W. J., Oerlemans, A. G., & Spieksma, F. C. R. (1994). Minimizing the number of tool switches on a flexible machine. *International Journal of Flexible Manufacturing Systems, 6*, 33–54.

Dillon, S., Jones, R., Hinde, C. J., & Hunt, I. (1998). PCB assembly line setup optimization using component commonality matrices. *Journal of Electronics Manufacturing, 8*(2), 77–87.

Djellab, H., Djellab, K., & Gourgand, M. (2000). A new heuristic based on a hypergraph representation for the tool switching problem. *International Journal of Production Economics, 64*, 165–176.

Follonier, J.-P. (1994). Minimization of the number of tool switches on a flexible manufacturing machine. *Belgian Journal of Operations Research, 34*(1), 55–72.

Günther, H. O., Gronalt, M., & Zeller, R. (1998). Job sequencing and component set-up on a surface mount placement machine. *Production Planning & Control, 9*(2), 201–211.

Gendrau, M., Hertz, A., & Laporte, G. (1992). New insertion and postoptimization procedures for the travelling salesman problem. *Operations Research, 40*(6), 1086–1094.

Hertz, A., Laporte, G., Mittaz, M., & Stecke, K. E. (1998). Heuristics for minimizing tool switches when scheduling part types on a flexible machine. *IIE Transactions, 30*, 689–694.

Hirvikorpi, M., Salonen, K., Knuutila, T., & Nevalainen, O. S. (2006). The general two level storage management problem – a reconsideration of the KTNS-rule. *European Journal of Operational Research, 171*, 189–207.

Jain, S., Johnson, M. E., & Safai, F. (1996). Implementing setup optimization on the shop floor. *Operations Research, 43*(6), 843–851.

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., & Shmoys, D. B. (1985). *The traveling salesman problem: A guided tour of combinatorial optimization*. New York, NY: John Wiley & Sons.

Lenstra, J. K., & Aarts, E. M. (Eds.). (1997). *Local search in combinatorial optimization*. New York, NY: John Wiley & Sons.

Leon, V. J., & Peters, A. (1998). A comparison of setup strategies for printed circuit board assembly. *Computers & Industrial Engineering, 34*(1), 219–234.

Matzliach, B., & Tzur, M. (1998). The online tool switching problem with non-uniform tool size. *International Journal of Production Research, 36*(12), 3407–3420.

Matzliach, B., & Tzur, M. (2000). Storage management of items in two levels of availability. *European Journal of Operational Research, 121*, 363–379.

Rajkumar, K., & Narendran, T. T. (1997). A heuristic for sequencing PCBs with due-dates. *International Journal of Operations & Production Management, 17*(5), 446–467.

Salonen, K., Smed, J., Johnsson, M., & Nevalainen, O. (2006). Grouping and sequencing PCB assembly jobs with minimum feeder setups. *Robotics and Computer Integrated Manufacturing, 22*, 297–305.

Shirazi, R., & Frizelle, G. D. M. (2001). Minimizing the number of tool switches on a flexible machine: an empirical study. *International Journal of Production Research, 39*(15), 3547–3560.

Smed, J., Johnsson, M., Puranen, M., Leipälä, T., & Nevalainen, O. (1999). Job grouping in surface mounted component printing. *Robotics and Computer-Integrated Manufacturing, 15*(1), 39–49.

Publication II

Raduly-Baka, Cs., Knuutila T., Nevalainen, O. Minimizing the number of tool switches with tools of different sizes. In *Proceedings of the 5th International Conference on Technology and Automation 2005 (ICTA'05)*, Oct 2005.

Minimising the number of tool switches with tools of different sizes.

Csaba Ráduly-Baka* Timo Knuutila†‡
Olli S. Nevalainen†

2005

Abstract

In this paper we address the combined problem of job-ordering and tool placement, where each tool can occupy more than one slot of the primary storage magazine. The capacity of the magazine is limited so that all the tools neccessary in the production cannot fit into the magazine at the same time, and the cost of magazine reorganization depends linearly on the number of tool moves. Our task is to find the order of processing the jobs and the positions to put the tools in the magazine, so that the total cost of switching tools from one job to the next is minimized. We introduce a new heuristic for the problem. The algorithm hybridizes an efficient tool switching algorithm based on the matrix permutation problem and a novel two level storage management algorithm. We compare the proposed solution method to previous approaches from the literature. Our comparisons indicate that the new algorithm procudes results with costs almost a third of the costs produced by algorithms previosly known in this field.

Keywords: tool management; tool switching; two level storage management; flexible manufacturing; heuristics; combinatorial optimization

*Elcoteq Design Center Oy, Joensuunkatu 13, FIN-24100 Salo, Finland
†Department of Information Technology and TUCS, University of Turku, FIN-20014 Turku, Finland
‡Corresponding author: knuutila@it.utu.fi

1 Introduction

The problem of increasing the production efficiency in flexible manufacturing systems by computational means has become an important aspect of manufacturing with the arrival of computer numerically controlled (CNC) machinery. While the kind of machinery is an excellent means of increasing the degree of automatization in the production, it still leaves numerous questions open with respect to its optimal usage. Some of these deal with the tool management which has clear effect on the operating cost and the productivity of the manufacturing process.

In particular, we consider the case where a CNC machine processes a number of jobs with a number of changeable tools. It is imposed that each job requires some subset of tools for its processing and that the machine has a *primary* tool magazine with a lower capacity than the number of all different tools [2]. When proceeding to the next job, the tools used by the next job must placed in the magazine, if not already there. Because not all the tools fit into the tool magazine some of the tools in the magazine may have to be removed to make room for the tools of the new job.

Flexible manufacturing systems are used in various industries to achieve efficiency in low volume high-mix product manufacturing. One example is, in the electronics industry, the assembly of printed circuit boards (PCB). Each PCB contains a number of components, and different PCB types are assembled by the system [5]. One can, in this context, interpret the component types (stored in component reels, sticks, etc.) as "tools" and the feeder unit as a "magazine".

Various aspects of the tool management have been discussed in literature. One aspect is the ordering of the jobs so that the number of required tool switches is minimized. This problem is also known as the *tool switching problem*. The problem is NP-hard even in the simplified case where all the tools are of the same size, and a number of heuristic solutions have been proposed in the literature [1]. The *tool-loading problem* considers a case where the job sequence is given, and the goal is to find a placement and removal order of tools, that the total cost of required tool changes is minimized. This problem can be solved optimally in the case of equal tool sizes using the KTNS (Keep Tool Needed Soonest) algorithm, as showed by Tang and Denardo [4].

For different tool sizes, the tool-loading problem becomes NP-hard as showed by Matzliach and Tzur in [6]. This problem is addressed in the literature also as the *Dynamic Storage Management Problem*, or *DSMP*. Because of different tool sizes, the storage area will become fragmented after processing a number of jobs, and the question arises how to place the tools to

minimize the fragmentation. The online version of this problem also appears in operating systems, where the system memory is shared among various tasks [3]. The offline version has been discussed in the context of flexible manufacturing systems [7].

The *DSMP* has been addressed in literature as the two level storage management problem in [6] and [7]. In this problem there are two magazines holding tools, one with a limited capacity internal to the machine, the other with an unlimited capacity external to the machine. The primary (internal) magazine can hold only a limited number of tools, and if a required tool is not in the primary storage area the machine must pick it up from the secondary (external) storage area. This may also involve the removal of some tools from the primary storage area, see Matzliach and Tzur in [6] and Hirvikorpi *et al.* in [7].

The combined problem of job-ordering, tool placing and tool-loading problems has been studied previously only for equal tool sizes [9]. The problem is easy in the sense that the tool-loading and placing can be solved optimally for it. In real life manufacturing systems it is likely that the tool sizes are unequal, see [8] for a number of heuristics for solving the problem.

In this paper we consider a new approach to the job-ordering, tool loading and placement problem with unequal tool sizes. Our approach to the problem is similar to that described by Tzur and Altman in [8], except that we consider a different combination of job ordering and storage management algorithms. Our storage management algorithm is based on the heuristic by Hirvikorpi *et al.* in [7], with small variations required by the nature of this problem. For the job-ordering we modify the heuristic introduced by Djellab *et al.* in [9]. This heuristic has been used very successfully to optimize instances of job-ordering problem with equal tool sizes [11]. The original Djellab heuristic is based on the KTNS tool-loading heuristic which we replace with our own tool-loading and placement heuristic.

The rest of this paper is organized as follows. In Section 2 we formally specify the problem and its assumptions. In section 3 the Djellab tool switching heuristic is briefly described, a detailed description can be found in [9]. Our use of the Djellab method is neutral in the sense that it does not take any position on the (equal or unequal) sizes of the tools. In Section 4, a modified version of the storage management heuristic of Hirvikorpi *et al.* [7] is described. In Section 5 we give the combined heuristics for the tool switching problem with different tool sizes. Section 6 summarizes the results of numerical tests with the new algorithm, here we observe that the new algorithm outperforms existing solution by a factor of 3, for large problem instances. Section 7 contains concluding remarks on the subject.

2 Problem description

The problem of *Job ordering, tool Loading and tool Placement* with unequal tool sizes (*JLP* for further abbreviation) has the following parameters:

- N - the number of jobs to be processed.
- M - the number of different tools available.
- C - the capacity of the tool magazine (number of slots) used as the primary storage area.
- T - set of tools, where each tool $t \in T$ has a transfer cost $c_t \in \mathbb{R}^+$ which is the cost to move, remove or insert the tool into the magazine, and a size $s_t \in \mathbb{N}$ which is the number of slots occupied by the tool.
- A - a job-tool incidence matrix. The size of A is $N \times M$ and the element a_{ij} of A is 1 if the ith job requires the jth tool, and 0 otherwise.

The goal is to find an ordering of the jobs (job scheduling) and the placement and loading order of the tools, so that the total cost of switching the tools between jobs is minimized. This problem statement omits the lifetime limitations of the tools. We assume that the tools do not wear and therefore can reside in the magazine as long as they are needed.

The *JLP* problem is NP-hard as shown by Matzliach and Tzur in [6], even in the case where the order of the jobs has been fixed. The best known heuristic for this problem is the *Aladdin* heuristic proposed by Tzur and Altman in [8]. The *Aladdin* algorithm combines the job-ordering and the tool-loading problems into a single heuristic.

We propose an algorithm, in which a job-ordering algorithm is used to sequence the jobs and a two level storage management algorithm is used to minimize the tool switching cost of each job sequence. Tzur and Altman experimented in [8] with various two-stage heuristics, where the job ordering and tool switching stages were separated. Their tests indicated that two-stage approaches gave weaker results than a unified one-stage solution, namely the *Aladdin* algorithm. We will see that, by choosing the heuristics carefully, one can create an efficient two-stage approach. We also note that the choice of storage management heuristic is crucial: most of the job-ordering heuristics presented in [8] can be improved simply by a different choice of the tool storage management algorithm.

3 The Djellab-Gourgand tool switching heuristic for equal tool sizes

The novel heuristic algorithm proposed by Djellab and Gourgand in [9] uses a hypergraph representation of the tool switching problem. The algorithm solves a weighted version of the matrix permutation problem [11]. As presented by Djellab *et al.* in [9], the algorithm gives excellent results for the tool switching problem when the tool sizes are equal.

A major strength of the heuristic by Djellab *et al.* [9] is that it is able to consider also an initial partial order of the jobs. The fact that certain jobs can be processed only after other jobs is commonly ignored in the tool switching literature.

The Djellab-Gourgand heuristic consists of two major parts:

- Given a priority order in which the jobs are considered for insertion, create a job sequence so that the number of tool switches using the KTNS policy is minimized. This procedure is called the *Best Insertion* (*BI*) heuristic.

- In the second part, iteratively try to improve the results by generating random priority orders and using the BI heuristic with these priority orders. This procedure is called the *Iterative Best Insertion* (*IBI*) heuristic.

In our implementation we modified the Djellab-Gourgand heuristic to allow different tool sizes. This is necessary due to the fact that the KTNS policy which is used in the original algorithm assumes equal tool sizes in order to be optimal.

The reader is refered to [9] for a detailed description of the original Djellab-Gourgand heuristic. Basically we apply here the same algorithm with the difference that we do not use the KTNS policy for tool placement, but the *IMTM* storage management algorithm proposed in section 4.

4 The storage management heuristic

The problem of storage management with different tool sizes and fixed job ordering has been proved to be NP-hard by Matzliach and Tzur in [6]. A number of algorithms have been proposed to solve this problem. In the case of moveable tool positions Matzliach and Tzur in [6] proposed an algorithm. The problem becomes even more complicated in the case when the tools cannot be moved unless involving the tool switching cost to each move. This

problem has been recently addressed by Hirvikorpi *et al.* in [7] and they give heuristics to solve the problem.

We suppose in the storage management problem (with different tool sizes and magazine reorganization costs) that there are N jobs to be processed in a predefined order, and the jobs use in total M different tools. The task is to determine a tool switching and placement strategy, which minimizes the total cost of tool changes. Tool t consumes s_t slots of the magazine, and the cost of removing the tool t into the magazine is c_t. The $N \times M$ binary matrix A tells whether the tool t is used ($A(j,t) = 1$) or not ($A(j,t) = 0$) in a job j. The capacity of the tool magazine is C slots.

In the present paper we implement a heuristic based on the *SMMT-2* heuristic introduced by Hirvikorpi *et al.* in [7]. We name this algorithm *Iterative Multi-Tool Manager* or *IMTM* algorithm. In our implementation we have improved for efficiency reasons the method *SMMT-2* uses for choosing tools for removal. This was necessary because storage management decisions are made repeatedly in the *IBI* procedure (of the Djellab-Gourgand heuristic) a great number of times and they thus become a bottleneck of the joint solution algorithm of the *JLP* problem. As to the results produced by the new heuristic and that of [7], according to our evaluations the solutions are of similar quality, but the new heuristic runs about 190 times faster.

4.1 Removal cost matrix

When cleaning the magazine for new tools, the decisions on which tools to remove are central to the *IMTM* algorithm. In order to save computation time, we precompute a matrix R of size $N \times M$, containing the removal cost for each tool of each job. Whenever the algorithm has to decide which tool to remove at a certain job, it uses the removal cost matrix to choose the tool.

The elements of R are calculated in a way that resembles how the KTNS rule is used to choose the tools to be removed, but now the removal costs are taken into consideration. For each job j and tool t let $dist(j,t)$ be the distance to the next job using the tool. If job j uses this tool, the distance is 0, otherwise it is the number of jobs until the tool is used again. Formally,

$$dist(j,t) = \begin{cases} 0 & \text{if } A(j,t) = 1 \\ m & \text{if } A(j+m,t) = 1 \text{ and } A(j+i,t) = 0 \text{ for all } i \in [0,m), \text{ and } m > 0 \\ N & \text{if } A(j+i,t) = 0 \text{ for all } i \in [0, N-j) \end{cases}$$

If the tool is not used in any of the forthcoming jobs, the distance is the number of jobs N, so that no other (used) tool can have this large distance. The tool removal costs (R) are then given by the following formula which

weights the tool removal costs by their distance to the next use. In this way we reuse the principle of the KTNS algorithm for tools of different sizes.

$$R(j,t) = \begin{cases} c_t/dist(j,t) & \text{if } dist(j,t) > 0 \\ \alpha \cdot c_t & \text{if } dist(j,t) = 0 \end{cases}$$

Here α is an arbitrary constant greater than 2, to give the highest removal cost for tools used by the current job (with distance 0). The storage management algorithm uses the R matrix when searching for the block of consequtive magazine slots with the lowest removal cost, to make room for new tools. Naturally R is only one (heuristic) measure for favourable actions to be taken when solving the storage management problem.

4.2 Outline of the *IMTM* algorithm

The *IMTM* algorithm builds a content of the tool magazine for each job by considering the magazine from the previous job, and inserting the tools needed by the job but not already in the magazine.

The tools are always inserted one by one, into the smallest available space they can fit. If the magazine does not contain enough empty space, the algorithm removes from the magazine some tools, which are not needed by the current job, to make room for the new tool. The choice of the tools to be removed is made according to the lowest removal cost value defined by R.

If no room can be made by removing unused tools, the algorithm removes from the magazine a tool which is required by the current job, and restarts, trying to insert the new tools (including the removed one) needed by the current job. This last step will cause a rearrangement of the tools in the magazine.

The algorithm tries to build a magazine with limited capacity for the current job using the tool magazine setting of the previous job. The algorithm iterates through all jobs and applies a magazine building procedure for each job. The magazine is considered to be empty before the first job.

4.3 Inserting in a free area

The storage management algorithm first looks for available empty spaces in the magazine for the new tool. This is done by the *InsertFreeArea* procedure. Given the size of the tool to be inserted, the *InsertFreeArea* searches the magazine to the shortest sequence of free slots, where the sequence contains at least as many slots as required by the tool size.

If such an area is not found, the storage management proceeds with the next step by trying to insert the new tool over some removable tools.

4.4 Inserting in a removable area

In order to insert a new tool over some removable tools, the algorithm must choose which tool can be removed. This choice is made according to the removal cost values stored in the R matrix.

The *InsertRemovableArea* procedure implements the insertion of a new tool into slots containing removable tools. The algorithm differentiates between tools which are used by the current job, and tools which are not and can thus be removed.

The algorithm separates the magazine space into removable and non-removable areas. Given a tool with size s_t and a removable area containing enough slots for the tool, the algorithm searches for a sequence of s_t slots with a lowest total removal cost. This is done by checking the removal cost by the aid of matrix R for each possible insertion location of the new tool.

The *InsertRemovableArea* procedure relies on the *FindMinimumCostPlace* procedure to find the lowest cost insertion slots in a removable area. The *GetToolRemovalCost* procedure is used to calculate the removal cost of tools stored in a number of consecutive slots. These algorithms operate on the *magazine* variable, defined as an array of slots.

```
GetToolRemovalCost(magazine, job, pos, len): cost
// computes the cost of making len slots free starting at pos
// - job is the job for which the cost is computed
// - pos is the first slot to be cleaned
// - len is the number of slots to be cleaned
   cost = sum of R[job, t] for all t stored in magazine[pos...pos + len - 1]
   return cost

FindMinimumCostPlace(magazine, job, start, end, ts, cost): pos
// finds the lowest cost insertion slot between slots start and end
// - job is the currently processed job
// - start is the start of area where insertion is considered
// - end is the last slot where insertion can occur
// - ts is the tool size
// - the cost of making free space is returned in cost

   Execute the GetToolRemovalCost procedure for all starting positions and
   return the position for which the GetToolRemovalCost procedure give the
   lowest cost.

InsertRemovableArea(magazine, job, tool, ts)
// inserts the tool into slots containing removable tools
// - job is the currently processed job
// - ts is the tool size
   pos = 0 // index of the slot where insertion can occur
   cost = 0 // current cost of insertion at pos
   for j = 1 to C
      if tool at magazine[j] is not used
```

```
        next = index of the first used tool after j in the magazine, or
               the number of slots in the magazine + 1.
        i = FindMinimumCostPlace(magazine, job, start, next - 1, ts, c)
        // c is the cost of inserting the tool at position i into the magazine
        if i > 0 and c < cost then // suitable for insertion
          set pos to i and cost to c
        end if
      end if
   end for
   if pos > 0 then // there is a slot where we can insert
      remove tools from magazine[pos...pos + ts - 1]
      store tool at magazine[pos]
   end if
   return pos > 0
```

The *InsertRemovableArea* algorithm can thus insert tools anywhere in a removable area, even into the middle of it. This means that the magazine can become fragmented after a number of tools have been placed over removable tools. This problem is somewhat handled by considering the tools in their decreasing order of size. Nevertheless, the *InsertRemovableArea* algorithm might fail to insert the tool, and it then returns to a higher level in the storage management algorithm to reconsider the tool insertion into a fragmented magazine space.

4.5 Inserting multiple tools

Let us now consider a higher level of hierarchy of the *IMTM* algorithm. As stated in the outline of the storage management algorithm, a number of new tools are inserted into the magazine for each job. This is done by the *InsertMultiTool* procedure. The insertions are done tool by tool. The algorithm first tries to insert using the *InsertFreeArea* procedure, and if that fails then with the *InsertRemovableArea* procedure.

```
InsertMultiTool(magazine, job, T)
// - job is the currently processed job.
// - T is the set of new tools to be inserted
   for i = 1 to |T|
      tool = t[i]
      size = sizeof(tool) // gets the size of the tool
      pos = InsertFreeArea(magazine, tool, size)
      if pos = 0 then
         pos = InsertRemovableArea(magazine, job, tool, ts)
      end if
      if pos = 0 then
         return FALSE
      end if
   end for
   return TRUE
```

If the *InsertMultiTool* algorithm fails for the current job (no place for new tools), the *IMTM* storage management algorithm tries to rearrange the used tools using the *IteratedInsertMultiTool* procedure.

The *IteratedInsertMultiTool* procedure considers the tools used by the current job, and inherited from the previous magazine in their increasing order of removal costs. At each step removes a tool from the magazine and inserts it in the set of new tools T (the ones not in the magazine but needed by the current job) which will be passed to the *InsertMultiTool* again.

```
IteratedInsertMultiTool(magazine, job, T)
// - job is the currently processed job.
// - T is the set of new tools to be inserted
   prevMagazine = magazine
   success = InsertMultiTool(magazine, job, T)
   if not success then
      MT = set of tools from the magazine needed by the current job
      sort MT by increasing order of removal cost from R
      for j = 1 to |MT|
         tool = MT[j]
         remove tool from prevMagazine
         add tool to T // set of new tools
         magazine = prevMagazine
         success = InsertMultiTool(magazine, job, T)
         if success then
            return success
         end if
      end for
      if not success then
         clear magazine
         insert T into magazine // inserts T as one block at the start
                                   of the magazine
      end if
   end if
   return success
```

After moving the tool from the magazine into the set of new tools, the *IteratedInsertMultiTool* procedure employs again the *InsertMultiTool* procedure to insert the set of T tools into the magazine. This loop is iterated on until all the tools have been inserted, or all the needed tools have been removed from the magazine, and they are all considered as new tools.

It is easy to see that the *IteratedInsertMultiTool* procedure cannot fail. If there is no possibility to insert the new tools into the magazine, all the tools needed by the current job will end up in T. These are either inserted into the lowest removal cost places (as the last step of the for loop) or the magazine is emptied and the tools are inserted as one block.

The *IMTM* algorithm employs the *IteratedInsertMultiTool* procedure to insert the new tools needed for the current job into the magazine.

4.6 Generating the magazine set-up for each job

At the highest level of the algorithm hierarchy, the storage management algorithm iterates over the set of N jobs, and builds a magazine set-up for each job using the *InsertJobTools* algorithm. We finally obtain as the output

an array of magazine set-ups. This is used to compute the total cost of tool switching for a particular sequence S of the jobs.

```
InsertJobTools(magazine[], T, N)
// - magazine is an array of magazine set-ups for each job.
// - T is the set of new tools to be inserted
// - N is the number of jobs.
   for j = 1 to N
      Tj = tools needed by job j.
      magazine[j] = magazine[j - 1] // copy the content of the
                                       previous magazine, where
                                       magazine[j] is a magazine
                                       set-up for one job.
      IteratedInsertMultiTool(magazine[j], j, Tj)
   end for
```

4.7 Comparison to the *SMMT-2*

Altough the storage management algorithm described above is based on the ideas introduced by Hirvikorpi *et al.* in [7], there are several fundamental differences between this algorithm and the *SMMT-2* algorithm of Hirvikorpi. Both algorithms produce results of similar quality, while the new algorithm does that about 200 times faster in similar conditions on the same data set. This allows us to use it as a storage management algorithm embedded into the Djellab-Gourgand job ordering heuristics.

The main differences between the *IMTM* algorithm and the *SMMT-2* algorithm presented by Hirvikorpi *et al.* in [7] are the following.

The *SMMT-2* algorithm tries to insert all the tools for a job in a single block. This might succeed in some cases, but for the upcoming jobs it might cause additional fragmentation.

The *SMMT-2* algorithm considers tools to be inserted in a random order. In the *IMTM* algorithm we consider tools in the decreasing order of their size, placing first tools with larger size.

However, the main difference lies on how tools for removal are chosen. The *SMMT-2* chooses the lowest removal cost tool to be removed, and then tries to place a new tool in its place. By removing the tools with the lowest removal cost, one does not always produce enough space for a new tool, and then other tools have to be removed. This causes problems especially when, for example, the two lowest cost removable tools are not in adjacent slots, so their removal does not free up relevant space. This means that we do not always remove the tool with the lowest removal cost.

When the *SMMT-2* algorithm runs out of possibilities of placing the new tools, it will try to place them as a single continuous block, relying on a less efficient algorithm. In the *IMTM* algorithm we remove a tool which is in the

magazine from the previous job, and then try the same algorithm to insert the tools required for the current job and not in the magazine.

In *SMMT-2* the tool removal cost is calculated in every step, which adds up to the complexity of the algorithm. In the *IMTM* algorithm we use a precomputed matrix of tool removal costs.

5 The combined heuristics

The *Best Insertion* (*BI*) routine of the job ordering algorithm by Djellab *et al.* in [9] does not utilize any knowledge of tool sizes. It is practically a gap minimization algorithm, which takes the job/tool incidence matrix, and searches for a job permutation, for which the number of horizontal gaps is minimized. These gaps represent tools which are removed from the magazine to make room for other tools. We use the *Best Insertion* routine as such in the *IMTM* algorithm.

The *Iterative Best Insertion* (*IBI*) algorithm [9] relies on the KTNS algorithm to evaluate the cost of job permutations found by the *Best Insertion* algorithm. In our implementation, for ordering jobs with unequal tool sizes, we replace the KTNS algorithm with the *IMTM* algorithm. We name the new algorithms with the replaced storage management *Best Insertion** and *Iterative Best Insertion** respectively. In this way the *Iterative Best Insertion** algorithm will return a job sequence which has the lowest cost according to the *IMTM* algorithm, from all sequences found by *Best Insertion**. Because the *IMTM* algorithm is used at each step in the iteration of the *Iterative Best Insertion** algorithm, the speed of the *IMTM* algorithm is crucial. Our evaluation showed that the *IMTM* algorithm is fast enough so that the Djellab-Gourgand heuristic remains usable in what comes to the time consumption.

Besides the replacement of the KTNS algorithm by the *IMTM* algorithm, all the other aspects of the job ordering remain as described by Djellab *et al.* in [9]. We name the new job ordering algorithm using tools with unequal sizes *DG+*.

6 Computational results

We compared the *DG+* algorithm presented in this paper to the Aladdin algorithm introduced by Tzur and Altman in [8]. We used the original implementation of the Aladdin algorithm, which was kindly provided to us by Dr. M. Tzur.

The Aladdin algorithm evaluation in [8] counts the number of tool switches

Table 1: Problem instances

N	M	Min	Max	C
10	10	2	4	12, 15, 20, 25
15	20	2	6	18, 25, 30, 35
30	40	5	15	45, 50, 55, 60
40	60	7	20	60, 65, 70, 75

instead of the cost of these switches. Indeed, in some of the manufacturing environments, the impact on the manufacturing cost is the number of switches (steps to refill the magazine), and not proportional to the size of the tools. In our comparison we used this switch counting method both for Aladdin (as it was originally) and the *DG+* algorithm.

The original Aladdin evaluation in [8] used random tool size sets for each instance. There were 10 instances for each job/tool number configuration, and there were a total of 4 job/tool configurations as follows: (10, 10), (15, 20), (30, 40) and (40, 60). Each job/tool configuration was tested with 4 different magazine capacities. We fixed the tool size distributions in our evaluation for both the Aladdin and for the *DG+* algorithm. Practically this means that we gave the same set of tools as an input for both the Aladin algorithm and the *DG+* algorithm. Table 1 summarizes the problem instances used in our test. The problem types are characterized by the following parameters:

- *N* - The number of jobs to be processed.
- *M* - The number of tools used to process these jobs.
- *Min* - The minimum number of tools used by a job.
- *Max* - The maximum number of tools used by a job.
- *C* - The capacity of the tool magazine.

It is important to note, that the comparison does not take account the initial tool setup. This means, that practically only the tool removals are counted when the number switches are evaluated.

In our test we used various tool size distributions, of 3 different tool sizes, occupying 1, 2 or 3 slots. The tool size frequency is indicated in each table. For example (1/3, 1/3, 1/3) means that each tool size has been used in equal proportion. The tables (2, 3, 4, 5) containing the test results are organized

Table 2: Results for (1/3, 1/3, 1/3) tool size frequencies

Instance		Switching cost		Running time	
Type	C	Aladdin	DG+	Aladdin	DG+
(10, 10, 2, 4)	12	13.100	4.300	0.150	0.030
(10, 10, 2, 4)	15	7.600	2.300	0.200	0.020
(10, 10, 2, 4)	20	0.600	0.700	0.290	0.020
(10, 10, 2, 4)	25	0.000	0.000	0.350	0.030
(15, 20, 2, 6)	18	43.100	12.300	0.721	0.110
(15, 20, 2, 6)	25	29.100	6.000	0.701	0.110
(15, 20, 2, 6)	30	12.500	3.200	0.711	0.110
(15, 20, 2, 6)	35	1.800	1.200	0.751	0.120
(30, 40, 5, 15)	45	243.200	62.600	1.892	1.832
(30, 40, 5, 15)	50	226.800	48.100	1.852	1.793
(30, 40, 5, 15)	55	201.400	35.800	1.892	1.763
(30, 40, 5, 15)	60	168.700	25.100	1.912	1.642
(40, 60, 7, 20)	60	472.100	137.900	4.736	5.999
(40, 60, 7, 20)	65	451.500	116.200	4.506	5.958
(40, 60, 7, 20)	70	437.700	99.600	4.536	6.029
(40, 60, 7, 20)	75	421.500	83.800	4.426	5.918

as follows. The first column contains the type of the problem instance. The second column contains the capacity of the magazine. The third and fourth columns contain the tool switching costs computed by Aladin and *DG+* respectively. The fifth and sixth column contain the running times for Aladin and *DG+* respectively.

Tables 2, 3, 4 and 5 present the comparison of results between the Aladdin algorithm and the *DG+* algorithm. There are 3 different tool sizes, tools occupying 1, 2 and 3 magazine slots. The frequency array specifies the number of tools with a given size. The running time of the algorithms was evaluated in milliseconds.

It is observed that DG+ outperformed Aladdin in most of the cases. In small problem instances the results are similar (in one case better for Aladdin). The DG+ algorithm performed excellently in cases of large instances.

7 Conclusions

We introduced a new heuristic approach for the combined job scheduling, tool loading and tool placement problem, with unequal tool sizes. The main idea behind this algorithm is to use an efficient storage management algorithm

Table 3: Results for (1/5, 1/5, 3/5) tool size frequencies

Instance		Switching cost		Running time	
Type	C	Aladdin	DG+	Aladdin	DG+
(10, 10, 2, 4)	12	15.400	5.500	0.550	0.051
(10, 10, 2, 4)	15	9.700	3.700	0.450	0.050
(10, 10, 2, 4)	20	2.100	1.800	0.500	0.040
(10, 10, 2, 4)	25	0.000	0.000	0.490	0.030
(15, 20, 2, 6)	18	48.700	17.400	0.751	0.130
(15, 20, 2, 6)	25	37.700	10.300	0.871	0.130
(15, 20, 2, 6)	30	27.100	7.200	0.901	0.121
(15, 20, 2, 6)	35	17.900	4.200	0.851	0.130
(30, 40, 5, 15)	45	265.100	84.800	1.952	2.063
(30, 40, 5, 15)	50	249.900	71.300	1.942	2.013
(30, 40, 5, 15)	55	232.100	58.700	1.912	2.053
(30, 40, 5, 15)	60	207.900	45.900	1.952	2.003
(40, 60, 7, 20)	60	516.200	184.600	4.706	6.709
(40, 60, 7, 20)	65	491.400	163.800	4.636	6.970
(40, 60, 7, 20)	70	474.300	144.100	4.626	7.031
(40, 60, 7, 20)	75	457.600	125.400	4.606	7.020

Table 4: Results for (1/5, 3/5, 1/5) tool size frequencies

Instance		Switching cost		Running time	
Type	C	Aladdin	DG+	Aladdin	DG+
(10, 10, 2, 4)	12	10.900	4.100	0.460	0.030
(10, 10, 2, 4)	15	5.100	2.000	0.450	0.030
(10, 10, 2, 4)	20	0.000	0.000	0.460	0.040
(10, 10, 2, 4)	25	0.000	0.000	0.480	0.030
(15, 20, 2, 6)	18	44.100	14.200	0.731	0.130
(15, 20, 2, 6)	25	27.500	7.200	0.711	0.120
(15, 20, 2, 6)	30	13.200	3.700	0.741	0.111
(15, 20, 2, 6)	35	2.900	1.300	0.731	0.130
(30, 40, 5, 15)	45	247.700	62.300	1.872	1.923
(30, 40, 5, 15)	50	217.700	47.600	1.802	1.942
(30, 40, 5, 15)	55	185.900	35.900	1.822	1.833
(30, 40, 5, 15)	60	148.100	26.300	1.882	1.753
(40, 60, 7, 20)	60	485.500	140.000	4.376	6.419
(40, 60, 7, 20)	65	464.800	120.300	4.276	6.399
(40, 60, 7, 20)	70	439.300	104.100	4.236	6.389
(40, 60, 7, 20)	75	405.500	87.700	4.236	6.390

Table 5: Results for (3/5, 1/5, 1/5) tool size frequencies

Instance		Switching cost		Running time	
Type	C	Aladdin	DG+	Aladdin	DG+
(10, 10, 2, 4)	12	6.200	1.700	0.450	0.030
(10, 10, 2, 4)	15	0.600	0.600	0.450	0.020
(10, 10, 2, 4)	20	1.600	0.000	0.470	0.020
(10, 10, 2, 4)	25	0.000	0.000	0.470	0.030
(15, 20, 2, 6)	18	36.400	8.000	0.741	0.120
(15, 20, 2, 6)	25	10.700	2.800	0.721	0.110
(15, 20, 2, 6)	30	0.500	0.500	0.741	0.100
(15, 20, 2, 6)	35	0.000	0.000	0.751	0.121
(30, 40, 5, 15)	45	206.200	31.200	1.862	1.642
(30, 40, 5, 15)	50	150.400	17.800	1.832	1.542
(30, 40, 5, 15)	55	72.400	8.500	1.932	1.412
(30, 40, 5, 15)	60	11.900	2.900	2.072	1.312
(40, 60, 7, 20)	60	433.100	83.400	4.105	5.548
(40, 60, 7, 20)	65	409.700	66.100	4.055	5.258
(40, 60, 7, 20)	70	385.600	50.500	3.995	5.137
(40, 60, 7, 20)	75	313.500	36.600	4.075	4.927

with a previously known job ordering algorithm [9].

The new storage management algorithm is based on the ideas introduced in [7], but here we applied a different policy on how tools are removed, and how the fragmented magazine is rearranged. The combination of these algorithms has not been addressed previously in the literature.

We compared the results of the new algorithm with the results of the Aladdin algorithm introduced in [8]. We found that combining high performance heuristics from both job scheduling and tool placement problems resulted in good quality and time performance. Our results indicate that this algorithm makes a remarkable improvement over previously known approaches. Further consideration of tool wearing could be added to the problem statement of *JLP* problem.

A further study could investigate the combination of other job scheduling algorithms with the *IMTM* algorithm and evaluate the result provided by the combination.

8 Acknowledgement

We are grateful to Dr. Michal Tzur for providing us with the implementation of the Aladdin algorithm.

References

[1] Y. Crama and J. van de Klundert, The approximability of tool management problems, Technical Report, Maastricht Economic Research School on Technology and Organizations (1996).

[2] Y. Crama and J. van de Klundert, The worst-case performance of approximation algorithms for tool management problems. Naval Research Logistics, 46, pp. 445-462 (1999).

[3] A. S. Tanenbaum, Modern operating systems, Prentice Hall Ic., 2nd edition, 200-201, 2001.

[4] C.S. Tang and E.V. Denardo, Models arising from a flexible manufacturing machine, part I: Minimization of the number of tool switches, Operations Research, Vol. 36, No. 5, pp. 767-777 (1988).

[5] M. Johnsson, Operational and tactical level optimization in printed circuit board assembly, PhD thesis, University of Turku (1999).

[6] B. Matzliach and M. Tzur, Storage management of items in two levels of availability, European Journal of Operational Research (2000) 121 pp. 363-379.

[7] Mika Hirvikorpi, Kari Salonen, Timo Knuutila, Olli Nevalainen, General Two Level Storage Management Problem - A reconsideration of the KTNS-Rule, European Journal of Operational Research (in print).

[8] M. Tzur and A. Altman, Minimization of tool switches for a flexible manufacturing machine with slot assignment of different tool sizes, IIE Transactions (2004) 36, pp. 95-100.

[9] H. Djellab, K. Djellab, M. Gourgand, A new heuristic based on the hypergraph representation for the tool switching problem. International Journal of Production Economics, (2000) 64, pp. 165-176.

[10] L.T. Kou, Polynomial complete consequtive information retrieval problems, SIAM Journal of Computing 6 (1) (1997) pp. 67-75.

[11] K. Salonen, Cs. Raduly-Baka, O. Nevalainen, A note on the tool switching problem of a flexible machine, Special Issue of Computers and Industrial Engineering, Selected papers from 32nd ICC&IE in Limerick, (to appear).

Publication III

Raduly-Baka, Cs., Knuutila T., Johnsson, M., Nevalainen, O. Selecting the Nozzle Assortment for a Gantry-Type Placement Machine. *OR Spectrum*, 30, 493-513, 2008.

Selecting the Nozzle Assortment for a Gantry-Type Placement Machine.

Csaba Ráduly-Baka[*] Timo Knuutila[†‡] Mika Johnsson[§]
Olli S. Nevalainen[†]

December 17, 2010

Abstract

Electronics manufacturing systems employ increasingly multi-head gantry machines, where several vacuum nozzles are used simultaneously in pick-and-place operations to insert components on bare PCBs. Their use includes several options that have an impact on the overall manufacturing speed of the machine. In the present paper we address the problem of selecting the nozzles for this kind of a gantry machine, which is an important subproblem of the larger scheduling problem of multi-head gantry machines. Nozzles come in different types, and different types of components may require different types of nozzles in their placing. We address first a case where a single PCB type is manufactured and the only limitation on the number of nozzles is given by the capacity of the placement head. Then we discuss the case where there is a budget limitation on the total cost of the nozzles we can buy. We show that both of these problems can be solved optimally by the means of efficient greedy algorithms. We also discuss the case of selecting nozzles when manufacturing multiple different PCB types.

Keywords: printed circuit boards, optimization, flexible manufacturing, heuristics, combinatorial optimization, multi-head placement machines.

[*]Elcoteq Design Center Oy, Joensuunkatu 13, FIN-24100 Salo, Finland
[†]Department of Information Technology and TUCS, University of Turku, FIN-20014 Turku, Finland
[‡]Corresponding author: knuutila@it.utu.fi
[§]Valor Computerized Systems (Finland) Oy, Ruukinkatu 2, 20450 Turku, Finland

1 Introduction

Multi-head placement machines of gantry-type are used increasingly in PCB (printed circuit board) manufacturing to place surface mounted components on bare PCBs. There are many reasons for their popularity; these include the great flexibility, accuracy and moderate price. Several different types of gantry machines are used in various manufacturing environments. One way to make a distinction between these types is to classify them to single arm and multiple arm machines. Our focus here is on the single-arm, multi-head placement machines, which can pick multiple components and place them on a PCB within one pickup-placement phase.

A multi-head placement machine has a pickup - placement arm, which can hold multiple nozzles of various types. Each nozzle can hold at a time one component of a certain type, which will be placed on the PCB. The number of nozzles in the arm is limited by the capacity of the arm. The machine is configured to manufacture a batch of some PCB types in an automated way. The components required for manufacturing the assembly are in a feeder adjacent to the machine. The placement arm is programmed to work in cycles of three operations: to pick up a number of components from the feeders, move along the (x, y)-plane parallel to the PCB, and place these components on their proper locations on the PCB. The movements of the arm are accomplished by two fast and accurate step-motors. The type and amount of the components to be placed are specific to the PCB type. Each component type can be picked and placed by a certain nozzle type, although a single nozzle type may support the placement of multiple component types. Furthermore, a given component may be handled by more than one nozzle type, of which one type is the "primary type" giving the most accurate placement.

There are various aspects of the PCB manufacturing process, which can be subject to optimization, like the ordering of jobs, grouping the jobs for minimizing feeder changes, or deciding the topology of the arm movements [3, 7]. Another aspect in the organization of PCB assembly processes is the cost of purchasing the necessary equipment. For example, in some situations one may have sufficient funds to provide the machine with a rich set of nozzles to reach an optimal nozzle-to-arm allocation within a fixed arm capacity. In such cases, duplicating some of the frequently used nozzles often reduces the time spent in arm movements. On the other hand, in real life manufacturing, one is often tied to a given budget when buying the nozzles. Each nozzle type has a price of it's own, and one has to keep the total purchase price within the limits of the budget when filling the arm with nozzles. Given this constraint, the problem of filling the arm with nozzles so that the number of

pick-up steps is minimized, becomes more complicated.

A multi-head surface mounting placement machine can be used to manufacture multiple PCB types, but the machine must usually be reconfigured before starting the manufacturing of a PCB batch of a new type. This operation may involve the change of some nozzles in the arm. This does not create additional problems in the case of an unlimited nozzle budget, since one can just choose an optimal nozzle configuration for each PCB type. In the case of a limited budget the problem becomes more complicated. Different PCB types may use the same nozzle (assuming no wearing of nozzles) to achieve better nozzle usage budget. But it is still possible that the available budget is less than the cost of all possible nozzles of an optimal solution of the problem with unlimited budget. In this case, we have to give up certain multiple copies of nozzles and thereby sacrifice the best possible efficiency of pick-up steps for PCBs of certain type, in order to stay within the budget.

Another important assumption deals with the availability of the nozzles. It is supposed here that the arm is capable of holding all the nozzles needed by a particular PCB. This restriction is valid for some placement machines, while there are other machines with a separate tool magazine from which nozzles may be changed automatically, if necessary.

In this paper we focus on the problem of optimizing the number of pick-up steps executed by the arm in two different scenarios. A "pick-up step" stands for the process of fetching the components of the next placement step to the empty nozzles of the arm. One can here use at least one and at most all of the nozzles of the arm. In the scenario of unlimited budget, we want to find a nozzle selection for the arm so that the number of pick-up steps is minimized for a single PCB type. This problem is called the *Optimal Nozzle Selection* problem, abbreviated as *ONS*, and it is closely related to the *MINIMAX* resource allocation formulation of the *apportionment problem* [4]. The apportionment problem is the problem of allocating seats in a proportional representation system (or federal system). Each region or unit receives seats according to its population size. A fair (or optimal) seat allocation is when each region receives a number of seats in proportion to its population. Since this is practically not achievable due to rounding, various fairness and optimality criterias can be defined. When the apportionment problem is formulated as the *MINIMAX* resource allocation problem, the objective function to minimize is similar as for the *ONS*.

In the *MINIMAX* problem, the goal is to minimize $max_{1\leq j\leq n} f_j(x_j)$, subject to $\sum_{j=1}^{n} x_j = N$. The algorithm provided in [4] is more general than needed for *ONS*, and its complexity (when applied to ONS problem) depends both on the size of the arm and on the number of nozzle types. We give a fast solution algorithm whose complexity does not depend on the size of the

arm, and prove its optimality for the *ONS* problem. This optimal solution is obtained by the means of a greedy algorithm, which performs greedy selections in 3 separate phases. The new algorithm is specialized on solving *ONS*, and would not work in general for *MINIMAX* resource allocation instances.

In the second scenario, a cost for each nozzle type and a maximum limit on the total nozzle budget are given. This problem variant is called the *Budget Constrained Nozzle Selection* problem, abbreviated as *BNS*. We start by considering initially a single PCB type, propose an algorithm similar to the one used for the *ONS* problem, and show that we can produce optimal results also in this case. We then proceed to the case of multiple PCB types, where it turns out that a direct generalization of *BNS* does not guarantee an optimal solution. Heuristic solution algorithms, where the objective function consists of the sum of the objective functions for each separate PCB type, is introduced for this more general case.

One should notice that the order of component placements is not fixed either in *ONS* or *BNS*. From the point of view of modelling the manufacturing costs this means that no costs are connected to the feeder-to-PCB, placement-to-placement, PCB-to-feeder, or feeder-to-feeder movements of the arm. This, of course, gives an underestimate of real costs, but the solution of the joint model is extremely hard, as is nowadays generally recognized [7].

Our aim is to approach the question of purchasing a beneficial set of nozzles on a coarse level. As another extreme of nozzle-related optimization problems there is a case where the placement order of the components is fixed [8]. For example, solution of an optimal nozzle selection of the arm can then be used at higher level of optimization when searching for an optimal component placement sequence.

The paper proceeds as follows. Section 2 gives a formal definition of the *ONS* and *BNS* problems. Optimal algorithms for these two problems are given in Sections 3 and 4. Section 5 introduces two heuristic methods for multiple PCB types with a budget constrained scenario. The proposed heuristics are evaluated and compared against optimal solutions in Section 6. Concluding remarks and some aspects for further studies are discussed in Section 7.

2 Problem descriptions

Let us assume that each component type can be manipulated with one nozzle type, only. We also assume that the arm capacity is large enough to hold at least one nozzle of each nozzle type required by the components of any

single PCB type[1]. The problem to solve in *ONS* is *which nozzles should be duplicated in order to minimize the number of pick-up phases when producing PCBs of a given type.* We start by defining some notation and giving a mathematical formulation for the *ONS* and the *BNS* problems.

2.1 Optimal nozzle selection problem

An instance of the *ONS* problem is described with the following concepts:

- n - number of different nozzle types required to manufacture the PCB type.
- R - the capacity of the arm (*i.e.* the number of places to hold nozzles in the arm).
- T - the set of nozzle types, where $T = \{\tau_1, \tau_2, ..., \tau_n\}$, and τ_i indicates the ith nozzle type.
- $\overline{p} = p_1 p_2 \cdots p_n$ - numbers of components on the PCB requiring nozzle type τ_i.
- $\overline{a} = a_1 a_2 \cdots a_n$ - a solution to *ONS*; *i.e.* the number of nozzles of types $\tau_1, \tau_2, ..., \tau_n$ put to the arm. These are the output variables of the *ONS*.

We assume that $p_i > 0$ for all $i = 1, 2, ..., n$, which implies that $a_i \geq 1$ for all i, because each component has to be placed.

When considering a particular nozzle type τ_i with a_i copies in the arm, the number of pick-up steps required to place the p_i components is

$$\left\lceil \frac{p_i}{a_i} \right\rceil .$$

The maximum of these expressions for all indices i gives the required number of pick-up steps of manufacturing a single PCB (note that we ignore here the ordering of placements). Let us denote the cost of placing components with nozzle allocation $\overline{a}$ by:

$$cost(\overline{a}) = \max_{i=1..n} \left\{ \left\lceil \frac{p_i}{a_i} \right\rceil \right\} . \tag{1}$$

[1]This renders manufacturing of a batch of PCBs of the same type without changes of the arm contents possible. Observe that the more general case with interleaved nozzle changes is also common in practice. Depending on the machine type, the cost of changes may be relatively high or moderate.

The task in *ONS* is to determine $\overline{a}$ so that $cost(\overline{a})$ is minimized with respect to the constraints (2) and (3):

$$\sum_{i=1}^{n} a_i \leq R, \tag{2}$$

$$a_i \geq 1, \text{for all } i. \tag{3}$$

2.2 Budget constrained nozzle selection problem *BNS*

In the case of the *BNS* problem we add the following concepts to the ones presented for the *ONS*:

- $\overline{c} = c_1 c_2 \cdots c_n$, where c_i is the cost of purchasing one nozzle of type τ_i.
- B - total budget for buying the nozzles.

The goal in the BNS problem is to calculate the values of a_i for all nozzle types τ_i, so that the total cost of these nozzles does not exceed B, and that the number of pick-up steps $cost(\overline{a})$ is minimized. For this, we add the following constraint to the model:

$$\sum_{i=1}^{n} c_i \cdot a_i \leq B. \tag{4}$$

It should be clear that an optimal solution given for the *ONS* may be too expensive in terms of limit (4) for the *BNS* problem. The *BNS* is an extension of the *ONS* problem, and in that sense is an extension to the *MINIMAX* resource allocation problem also. Basically the *BNS* is a budget-constraint extension of the *MINIMAX* of [4] and as such it is not solved in [4].

2.3 Linear programming approximation to ONS and BNS

It should be noted that if we drop the ceiling operator from (1), the resulting problem can be written as a linear program with the objective function:

$$A' = \max_{i=1\ldots n} \left\{ \frac{p_i}{a_i} \right\} \tag{5}$$

and constraints (2), (3) (and (4) in the BNS case). Equation (5) can be also written as to minimize y, where $y \geq p_i/a_i$ for all $i \in [1, n]$. By substituting

y with $1/z$ we get the problem of maximizing z, where $z \leq a_i/p_i$ for all $i \in [1, n]$. Together with (2), (3) (and (4)) this forms a linear optimization problem.

2.4 Nozzle selection for multiple PCB types

When selecting nozzles for multiple PCB types, the *ONS* problem remains the same as defined earlier, except that there will be separate instances of the problem for each PCB type. These instances are independent of each other and they can be solved optimally for each PCB type.

Let Φ denote the set of different PCB types to be manufactured. Each PCB type has its numbers of component types and amounts of components for these types. We use the following notations for multi-type *ONS* and *BNS*:

- R - capacity of the arm.
- Φ - set of m different PCB types, $\Phi = \{\phi_1, \phi_2, ..., \phi_m\}$.
- n - total number of different nozzle types required to manufacture the PCB types in Φ.
- T - set of nozzle types used by all PCB types, where $T = \{\tau_1, \tau_2, ..., \tau_n\}$. We assume that each τ_i is needed in the production of at least one PCB type.
- p_{ki} - the number of those components inserted to the PCB type ϕ_k that require τ_i. This value is 0 if no such components exist.
- q_k - total amount of components inserted to the PCB type ϕ_k, $q_k = \sum_{i=1}^{n} p_{ki}$.
- B - the budget constraint for buying the nozzles.
- $\overline{c} = c_1 c_2 \cdots c_n$ - the purchasing costs c_i of a nozzle of type τ_i.

The output variables and objectives are described below.

- a_{ki} - the number of nozzles of type τ_i allocated to the manufacturing of PCB type ϕ_k. If τ_i is not required in the production of ϕ_k, then a_{ki} will be 0.
- $cost(\overline{a}_k)$ - number of pick-up steps required to place all components onto all PCBs of type ϕ_k. This is the multiply of PCB count of type ϕ_k, and the number of steps to build one PCB of type ϕ_k. The values $cost(\overline{a}_k)$ can be calculated once $\overline{a}_k$ is known.

- $cost(\overline{a})$ - number of pick-up steps required to manufacture all PCB types, $cost(\overline{a}) = \sum_{k=1}^{m} cost(\overline{a}_k)$.

Our goal is to minimize $cost(\overline{a})$.

One should observe that in this notation it is allowed that a PCB type ϕ_k has no components requiring nozzles of type τ_i. This may cause overflow in calculations, because we have the expression p_{ki}/a_{ki} to calculate the number of pick-up steps, required to place all components using nozzles of type τ_i. We therefore ignore the unnecessary nozzles in capacity calculations.

In the present heuristic we consider the batch sizes of various PCB types by multiplying the number of components needed for one PCB by the batch size. We omit the cost for changing the nozzles to the arm when changing the PCB type.[2]

The different PCB types are manufactured in separate sessions. The machine has to be reconfigured between these sessions, and the arm contents are updated with nozzles best suited for the upcoming PCB type. In the case of the multi-type *ONS* problem, we may use the same or different nozzles (if nozzle wearing is a problem) for various PCB types, and the total cost of the nozzles is not supposed to be an issue.

When the nozzle costs are limited by a budget (in the multi-type problem), some nozzles may have to be reused between various PCB types. In this case, one may still have a situation where the budget is large enough, to allow optimal selection of nozzles for each sub-problem (each PCB type). Otherwise, we have to reduce or redistribute the amounts of various nozzle types. Furthermore, reducing the amount of nozzles of one type, may still not reduce the optimally required amount (in the *ONS* sense) of that nozzle type for a certain PCB type ϕ_k.

3 Optimal solution of the *ONS* problem

In this section we describe a solution algorithm for the *ONS* problem, and prove the optimality of the solution. We also provide a complexity analysis of the algorithm.

The *ONS* problem is similar to the *apportionment problem* described in [4]. The apportionment problem has many variations in terms of the objective function, and it can be formulated as a resource allocation problem with the same objective function as the *ONS* problem. This variant of the

[2]If the nozzle change costs depend on the number of changed nozzles, we face an additional problem to be solved, the job sequencing problem, which we want to avoid at this phase of our work.

apportionment problem is also known as the *MINIMAX resource allocation problem*, which is a generalization of our *ONS* problem. Especially, the *ONS* problem can be solved optimally in time $O(max\{n, n\log(R/n)\})$ using the algorithm provided for *MINIMAX* in [4]. The algorithmic solution proposed here is tailored to the *ONS* problem, and it solves it in $O(n \cdot \log n)$ time, where sorting the nozzle types is a dominating factor. This is an improvement over the general method whenever $R > n^2$.

3.1 Preliminaries

Before going into the details of our *Optimal Nozzle Selection Algorithm* (*ONSA*), we formulate some mathematical relations, regarding to any optimal solution $\overline{a}$ of the *ONS*. This will help us to understand the operations of the *ONSA*, and facilitates its proof of optimality. We will denote with P the number of all placements on a PCB, that is

$$P = \sum_{i=1}^{n} p_i.$$

Note 1. *Let* $\overline{b}$ *be any solution for* ONS. *Then*

$$cost(\overline{b}) \geq \left\lceil \frac{P}{R} \right\rceil.$$

Proof. We always need at least $\lceil P/R \rceil$ pickups to pick P components with a hand of capacity R. This happens *e.g.* when all components are of the same type. □

In the *ONSA* algorithm, we will rely on a simple observation, concerning the placement of component types with a low amount of components, requiring only one nozzle.

Lemma 1. *Let* $\overline{a}$ *be an optimal solution for* ONS *and let* $i \in 1, 2, ..., n$ *be such that* $p_i \leq P/R$. *Then there exists a solution* $\overline{b}$ *for* ONS *such that*

1. $b_i = 1$ *and*
2. $cost(\overline{b}) = cost(\overline{a})$

Proof. Define $\overline{b}$ as

$$\overline{b} = \begin{cases} b_j = a_j & \text{for } j \neq i \text{ and} \\ b_i = 1. \end{cases}$$

Since $a_i \geq 1$ (by assumption) we know that

$$\sum_{i=1}^{n} b_i \leq \sum_{i=1}^{n} a_i \leq R.$$

Hence, $\overline{b}$ must be a feasible solution.

Let

$$k = \arg\max_{j=1\ldots n} \left\{ \left\lceil \frac{p_j}{a_j} \right\rceil \right\},$$

that is, τ_k determines the cost of $\overline{a}$ (note that there may be several such k). If $k \neq i$, then $b_k = a_k$ and consequently $cost(\overline{b}) = cost(\overline{a})$. If $k = i$ we first note that $p_i/b_i \geq p_i/a_i$ (since $b_i = 1$ and $a_i \geq 1$) and henceforth

$$i = \arg\max_{j=1\ldots n} \left\{ \left\lceil \frac{p_j}{b_j} \right\rceil \right\}$$

as well, and

$$cost(\overline{b}) = \left\lceil \frac{p_i}{b_i} \right\rceil.$$

Since $b_i = 1$ we have that $p_i/b_i = p_i$. The initial assumption was that $p_i \leq P/R$, and consequently

$$cost(\overline{b}) = p_i \leq \frac{P}{R}.$$

From *Note 1* we know that $P/R \leq cost(\overline{a})$, which means that $cost(\overline{b}) \leq cost(\overline{a})$, too. Equality must hold, since $\overline{a}$ is an optimal solution. □

Corollary 1. *Let* $S_1 = \{\tau_i \mid \tau_i \in T \text{ and } p_i \leq P/R\}$ *(*i.e. *all such* τ_i *that fulfill the assumption made in* Lemma 1*), and let* $\overline{a}$ *be an optimal solution of* ONS*. Then there exists a solution* $\overline{b}$ *such that*

1. $b_i = 1$ *for all* $i \in S_1$ *and*
2. $cost(\overline{b}) = cost(\overline{a})$.

Proof. The proof is obtained by apply *Lemma 1* sequentially for all $\tau \in S_1$. □

The next proposition gives an upper bound for the number of nozzles of a certain type in an optimal solution.

Proposition 1. ONS *has an optimal solution that contains at most*

$$\left\lfloor \frac{R}{P} \cdot p_i \right\rfloor + 1$$

nozzles of any nozzle type τ_i.

Proof. Let $\tau : i \in T$ and define

$$b_i = \left\lfloor \frac{R}{P} \cdot p_i \right\rfloor .$$

This implies that

$$b_i + 1 > \frac{R}{P} \cdot p_i. \tag{6}$$

With $b_i + 1$ nozzles of type τ_i and p_i components requiring τ_i, the number of pickup steps to place all of them is clearly

$$\left\lceil \frac{p_i}{b_i + 1} \right\rceil .$$

Inequality (6) implies that

$$\left\lceil \frac{p_i}{b_i + 1} \right\rceil < \left\lceil \frac{p_i}{\frac{R}{P} \cdot p_i} \right\rceil = \left\lceil \frac{P}{R} \right\rceil \leq cost(\overline{a}). \tag{7}$$

Since $\lceil p_i/(b_i + 1) \rceil \leq cost(\overline{a})$, using more than $b_i + 1$ nozzles for τ_i will not improve the solution. □

3.2 The *ONSA* algorithm

We describe here algorithm *ONSA* for the *ONS* problem. The result of the algorithm is an array $\overline{b}$ of nozzle type occurrences in the arm, which we show that to be optimal.

3.2.1 Phase 1

The algorithm partitions nozzles into two sets S_1 and S_2, where

$$S_1 = \{\tau_i \mid \tau_i \in T \text{ and } p_i \leq P/R\}$$

and

$$S_2 = T \backslash S_1.$$

Cor. 1 tells that it is sufficient to assign one nozzle to the members of S_1. We denote with $R' = R - |S_1|$ the capacity and with $P' = P - \sum_{i \in S_1} p_i$ the number of placements remaining after moving certain nozzle types into S_1.

If S_2 is empty, the algorithm stops here. Note that, in such case we have a trivial problem instance where assigning one nozzle of each type leads to the optimal solution (see Cor. 1). In this case $p_i = P/n$ for all τ_i, and $n = R$.

3.2.2 Phase 2

The algorithm removes some nozzle types from S_2 and places them into a new set S_3. S_3 will contain those nozzle types that already have the maximum reasonable nozzle amounts. For doing this, the algorithm first assigns $b_i = \lfloor (R'/P') \cdot p_i \rfloor$ nozzles for all nozzle types in S_2. Let us denote the remainingarm capacity with R'', where $R'' = R' - \sum_{\tau_i \in S_2} b_i$. If $R'' > 0$, the algorithm distributes the remaining R'' free places to nozzle types having the highest numbers of pick-up steps. The algorithm searches in S_2 for the maximum p_i/b_i, adds one (nozzle) to b_i and moves τ_i from S_2 into S_3. This process is repeated R'' times.

After the above steps, the algorithm ends up with S_2 containing nozzles with $b_i = \lfloor (R'/P') \cdot p_i \rfloor$, and S_3 with $b_i = \lfloor (R'/P') \cdot p_i \rfloor + 1$. It was shown in Prop. 1 that there must be an optimal solution that does not contain more nozzles than $\lfloor (R'/P') \cdot p_i \rfloor + 1$ for any τ_i.

At the end of this second phase, it holds for the intermediate result $\overline{b}$ that $p_i < cost(\overline{b})$ for all $\tau_i \in S_1$. This is because Cor. 1 shows that $p_i < cost(\overline{a})$, and obviously $cost(\overline{a}) \leq cost(\overline{b})$. Also, we have $p_i/b_i < cost(\overline{b})$ for all $\tau_i \in S_3$, which follows from Prop. 1 and from the way in which numbers b_i were calculated. So, after the first two phases, $cost(\overline{b}) \leq \max_{\tau_i \in S_2} \{p_i/b_i\}$.

3.2.3 Phase 3

The rest of the algorithm is about still improving the allocation of nozzles in S_2. This is done by searching for the maximum p_i/b_i among the $\tau_i \in S_2$ and trying to decrease it by increasing b_i.

Let $i \neq j$ be two such indices that $p_j/(b_j - 1) < p_i/b_i$ and $b_j > 1$. If there is more than one such a pair, we choose the one where p_i/b_i is maximal and $p_j/(b_j - 1)$ is minimal. If such an i and j are found, the algorithm decreases b_j by one and increases b_i by one. It also moves τ_i from S_2 into S_3, because decreasing b_i to its original setting would give a greater $cost(\overline{b})$ than the one given by the new S_2. The algorithm repeats this step while there pairs with the mentioned conditions can be found. It is clear that this process will terminate after at most $|S_2| - 1$ steps.

After executing the third phase, the remaining content of S_2 will define the number of pick-up steps, that is, $cost(\overline{b}) = \max_{\tau_i \in S_2}(p_i/b_i)$. We show later, that this is also an optimal solution for the *ONS* problem. Appendix 1 contains the pseudocode of the *ONSA* algorithm.

The time complexity of the *ONSA* algorithm, is determined by the maximum complexity of the three phases in the implementation. The first phase runs in $O(n)$ time. The second phase has two steps, the first of which runs in $O(n)$ time, while the second one takes at most $O(n^2)$ time since $R'' < n$. The inequality holds, since

$$R'' = R' - \sum_{\tau_i \in S_2} b_i = R' - \sum_{\tau_i \in S_2} \left\lfloor \frac{R'}{P'} \cdot p_i \right\rfloor \leq R' - \sum_{\tau_i \in S_2} \left(\frac{R'}{P'} \cdot p_i - 1 \right) = |S_2| \leq n.$$

This phase can be optimized to $O(n \cdot \log n)$ by sorting the nozzle types in a decreasing order of p_i/b_i.

The third phase runs in $O(n)$ time assuming that the access to nozzle types is done in a decreasing order of values p_i/b_i. Set S_3 can not contain more than n nozzles, and each step can be executed in constant time assuming that the maxima p_i/b_i and minima $p_j/(b_j - 1)$ are determined using sorted arrays. Hence, the *ONSA* algorithm finds a solution of the *ONS* problem in time $O(n \cdot \log n)$.

3.3 The proof of optimality of *ONSA*

The next proposition shows the optimality of *ONSA*.

Proposition 2. *The result of the* ONSA *algorithm is optimal.*

Proof. As stated in Cor. 1, the assignment of one nozzle to each $\tau_i \in S_1$ must be part of an optimal assignment. If S_2 is empty, all nozzle types are in S_1, and so the solution is optimal (see Cor. 1). It remains to prove that the optimal solution is obtained after the rearrangement of S_2 has been done in the third phase of the algorithm. Let $\overline{a}$ be an optimal solution to the *ONS* problem. Let $\overline{b}$ be the intermediary state in phase 3 (modified by the algorithm), and let $\overline{b'}$ be the final state of $\overline{b}$ when the algorithm stops, that is, the solution provided by *ONSA*. We show that as phase 3 of *ONSA* completes, the $\overline{b'}$ (the final state of $\overline{b}$) whill contain an optimal solution and $cost(\overline{b'}) = cost(\overline{a})$, by assuming that $cost(\overline{b'}) > cost(\overline{a})$ and showing that this leads to a contradiction.

For all $\tau_i \in S_3$ we assigned b_i nozzles, where

$$b_i = \left\lfloor \frac{R}{P} \cdot p_i \right\rfloor + 1.$$

By replacing b_i in (7) of Prop. 1 with $(b_i - 1)$ we get

$$\frac{p_i}{b_i} < cost(\overline{a}).$$

In phase 2 of the algorithm we placed iteratively into S_3 those nozzle types for which $p_i/(b_i - 1)$ was maximal (we also increased their number of copies by one, which is where the denominator $b_i - 1$ originates from). By increasing their nozzle amount we either reduced or maintained the current value of $cost(\overline{b})$ (that is, the intermediary state in phase 3). So we get

$$\frac{p_i}{b_i} < cost(\overline{a}) < cost(\overline{b}) \leq \frac{p_i}{b_i - 1}$$

for all $\tau_i \in S_3$. This means that $\overline{a}$ contains the same amount of nozzles as $\overline{b}$ does for all $\tau_i \in S_3$. It follows from this observation that the claimed difference $cost(\overline{a}) < cost(\overline{b})$ must originate from S_2.

We know from the earlier discussion that there exists some optimal solution with the same amount of nozzle allocations as S_1 and S_3. Hence, the nozzle types in S_2 have the same total number of nozzles both in $\overline{b}$ and in $\overline{a}$. Assumption $cost(\overline{a}) < cost(\overline{b})$ means that the nozzles in S_2 can be allocated in a better way than the allocation found by *ONSA*.

Let τ_j be the "bottleneck nozzle" of $\overline{b}$ in S_2, *i.e.* $cost(\overline{b}) = \lceil p_j/b_j \rceil$. Assuming that $cost(\overline{a}) < cost(\overline{b})$, it must hold that $a_j > b_j$. If a_j is to be greater than b_j, there must be some other nozzle $\tau_k \in S_2$ such that $a_k < b_k$. This implies that

$$\frac{p_k}{b_k - 1} \leq \frac{p_k}{a_k} \leq cost(\overline{a}).$$

However, we also know that $cost(\overline{a}) < cost(\overline{b}) = p_j/b_j$, which means that

$$\frac{p_k}{b_k - 1} < \frac{p_j}{b_j}.$$

But this is not possible, because such situations were eliminated from S_2 in the third phase of *ONSA*. In this phase, nozzle types τ_i were moved into S_3 for all pairs i, j where $p_j/(b_j - 1) < p_i/b_i$. Thus, the assumption $cost(\overline{a}) < cost(\overline{b})$ would lead to a contradiction, and we can conclude that $cost(\overline{a}) = cost(\overline{b})$. □

4 Optimal solution of the *BNS* problem

Depending on the budget, the solution computed by *ONSA* may not be valid for the *BNS* problem. In order to account for the additional constraint (4)

in *BNS*, we use a modified version of the *ONS* algorithm. It is shown that this new algorithm, named *BNSA*, computes an optimal solution for the *BNS* problem.

The *BNSA* algorithm starts by assigning 1 nozzle of each type in an originally empty array $\overline{b}$. It then looks for the bottleneck type (τ_i for which p_i/b_i is maximum) and increases the number of copies for this nozzle type by one if the budget constraint (4) and the arm capacity constraint (2) still hold. The algorithm repeats this step (searching for a bottleneck type and increasing it's amount) until either of the limits is reached, after which it stops and returns $\overline{b}$. We will show that this algorithm finds an optimal solution for the *BNS* problem. The pseudocode for *BNSA* is given in Appendix 2.

The running time of *BNSA* is $O(R \cdot n)$, which is pseudo-polynomial since the size of the input is $O(n \log(R) + \log(B))$. Although we could employ the *BNSA* algorithm to solve the *ONS* problem, too (*BNSA* is much easier to implement and analyze than *ONSA*), the running time of *ONSA* is much smaller ($O(n \log n)$). We expect this to be important in practice, since the capacity R may be much larger than the number of nozzle types n.

4.1 The proof of optimality of *BNSA*

Let us first look closer to the method for increasing the nozzle amounts. At each iteration, the algorithm greedily selects τ_i for which p_i/b_i is maximal. This is justified, since such τ_i must be one of the types determining the current cost of $\overline{b}$. The algorithm then increases b_i, but only if all other constraints ((2) and (4)) remain valid. Here we observe that for any τ_j, if $b_j > 1$ then $p_j/(b_j - 1) \geq cost(\overline{a})$, because b_j can be greater than one only if it has been increased by the algorithm, which can happen only if τ_j has been the bottleneck nozzle in some earlier iteration.

Proposition 3. *The result of the* BNSA *algorithm is optimal.*

Proof. Let $\overline{b}$ be the nozzle assignment produced by *BNSA* and let $\overline{a}$ be an optimal solution. We need to show that $cost(\overline{b}) = cost(\overline{a})$, and we again assume the contrary, *i.e.* $cost(\overline{b}) > cost(\overline{a})$.

Since $cost(\overline{a}) < cost(\overline{b})$, there is some $\tau_i \in T$ such that $a_i > b_i$ and $\lceil p_i/b_i \rceil = cost(\overline{b})$. This means that we would need a larger b_i in order to produce a better result. But we also know that when stopping at the bottleneck nozzle type, *BNSA* has reached at least one of the constraints (2) and (4). This means that there must be some τ_j such that $b_j > a_j$, because otherwise $\overline{a}$ could not fulfill these constraints either.

The property $b_j > a_j$ implies that $b_j > 1$, which means that τ_j must have been selected by *BNSA* as a bottleneck nozzle type, since b_j has been

increased. This means that $p_j/(b_j - 1) \geq cost(\overline{b})$, and with $b_j > a_j$ (and consequently $b_j - 1 \geq a_j$) it results that $p_j/a_j \geq cost(\overline{b})$, which leads to a contradiction under our assumption $cost(\overline{a}) < cost(\overline{b})$, since $p_j/a_j \leq cost(\overline{a})$ (by the definition of the cost function). □

5 Nozzle selection for multiple PCB types

As mentioned earlier, the *ONS* problem for multiple PCB types remains essentially the same as in the case of a single PCB type. We can just solve the problem for each PCB type separately and combine the results to obtain the optimal solution for *ONS* problem with multiple PCB types.

The *BNS* problem is more complicated. In this case we may reuse nozzles between consecutive PCB type sessions (assuming unlimited nozzle lifetimes). This most probably reduces the total cost of nozzles, but it may still be larger than the available budget. For this problem we outline two simple greedy heuristics (*MBNSA1* and *MBNSA2*) similar to the ones given for the single PCB *BNS* problem. These heuristics do not give optimal solutions, which was revealed in our testing by comparing their solutions against a brute-force method. We do not yet know the complexity class of this problem, which property remains a subject of further study.

5.1 MBNSA1

The *MBNSA1* algorithm employs *ONSA* to solve optimally the *ONS* problem for each PCB type. It then considers the resulting nozzle selection, and iteratively reduces the number of nozzle copies until the given budget is satisfied.

At each step of this iteration, the algorithm searches for a victim nozzle type, such that dropping out one copy of that type causes minimal increase in the total number of pick-up steps. This step is repeated until the desired budget constrain is reached. *Appendix 3* contains the pseudocode of *MBNSA1*.

5.2 MBNSA2

The *MBNSA2* algorithm considers initially only one nozzle for each type and then iteratively increases this amount in such a way that the total nozzle cost remains in the budget. At each step it chooses the type for which the benefit of an extra copy is locally maximal, when counting over all PCB types. Thus,

a) case 1	p_1	p_2	p_3	p_4	p_5
PCB 1	200	200	100	100	0
PCB 2	0	200	200	100	100
PCB 3	100	0	100	200	200
Costs	1	1	1	1	1

b) case 2	p_1	p_2	p_3	p_4	p_5
PCB 1	200	200	100	100	0
PCB 2	0	200	200	100	100
PCB 3	100	0	100	200	200
Costs	1	1	2	2	3

c) case 3	p_1	p_2	p_3	p_4	p_5
PCB 1	1000	100	100	100	0
PCB 2	0	100	100	1000	100
PCB 3	100	0	100	100	1000
Costs	1	1	1	1	1

d) case 4	p_1	p_2	p_3	p_4	p_5
PCB 1	1000	100	100	100	0
PCB 2	0	100	100	1000	100
PCB 3	100	0	100	100	1000
Costs	1	1	2	2	3

e) case 5	p_1	p_2	p_3	p_4	p_5
PCB 1	400	200	200	100	100
PCB 2	0	100	200	100	400
PCB 3	100	0	100	400	200
Costs	1	2	1	2	1

f) case 6	p_1	p_2	p_3	p_4	p_5
PCB 1	400	200	200	100	100
PCB 2	0	100	200	100	400
PCB 3	100	0	100	400	200
Costs	1	2	3	4	5

Table 1: The problem instances used to evaluate the multi-model heuristics. The table shows the number of components and the cost for each nozzle type.

the algorithm generalizes the greedy decisions of *BNSA*. *Appendix 4* contains the pseudocode of *MBNSA2*.

6 Experiments

This section summarizes the results of various experiments on the cost/performance tradeoff given by the *MBNSA1* and *MBNSA2* algorithms. The *ONSA* and *BNSA* algorithms solve their problems optimally, so we do no present results of those algorithms here. Their complexity is known and their implementation is fast (execution takes less than a second) on any practical instance. We analyze the number of pick-up steps resulting from a given nozzle budget B, which we vary to see it's effect on the number of pick-up steps. These results are compared with the optimal solution provided by a brute force algorithm.

We used problem instances with 5 nozzle types (n) and 3 PCB types (m) in our testing. The capacity of the arm (R) was 10 in each of the problem instances. We study the effect of cost (c_i) and component and PCB variation on the efficiency of production (the total number of component pick-up steps for all PCB types).

The heuristics did not always find the optimal solution, as can be seen in *Fig. 1.* Nevertheless, the solutions are fairly close to the optimal ones in most cases. Similarly to the *BNS* problem (for which we have an exact solu-

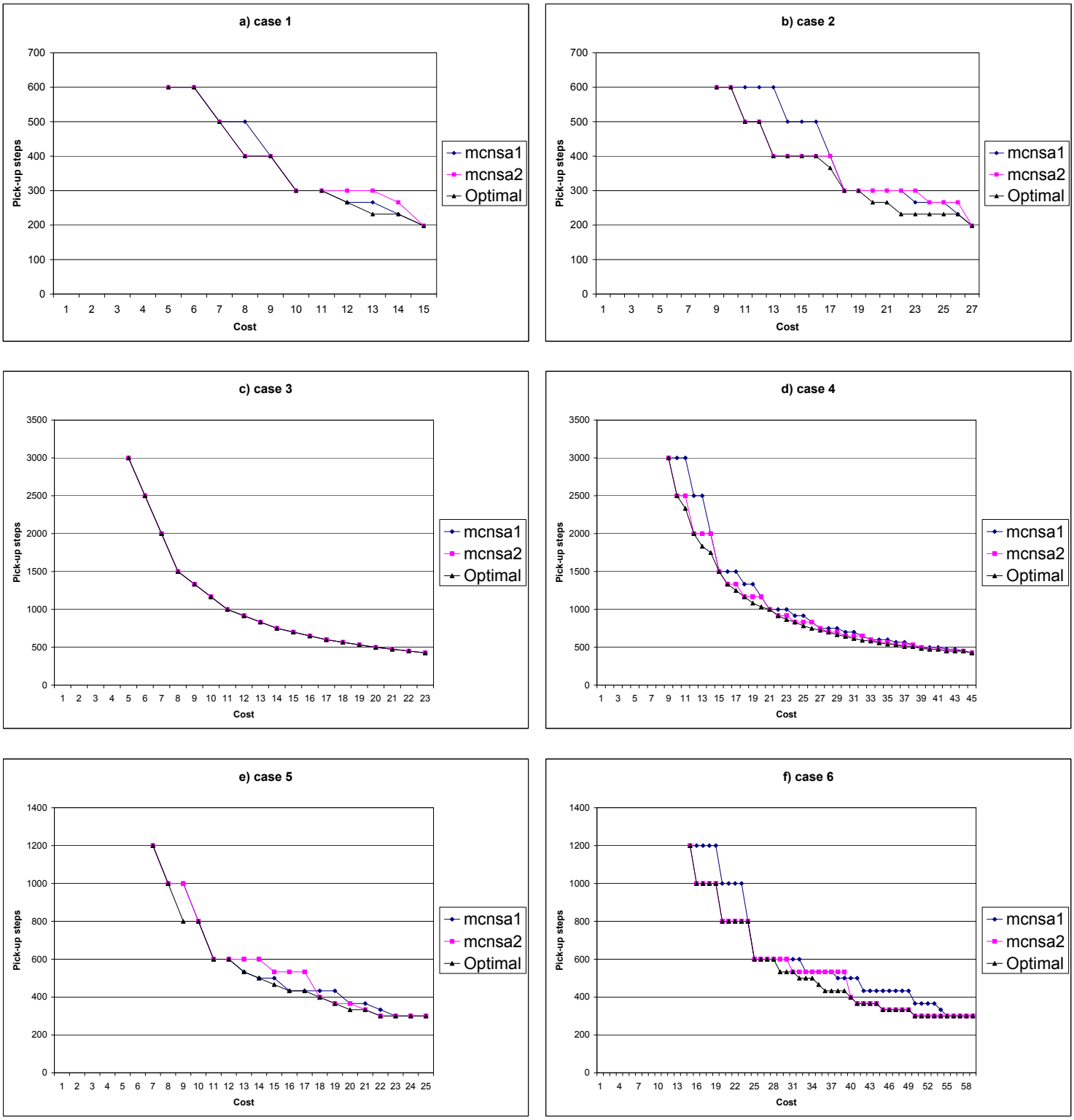

Figure 1: Multi-model problem with constrained budget (*MBNS*). The graphs show the number of pick-up steps as a function of the budget (B) for buying the nozzles. The problem parameters have been stated in *Table 1*. Notify that the scales of the axes are different between panels.

tion), the *MBNS* problem also exhibits a cost-performance variation where an initial budget increase gives a great increase in performance.

7 Conclusion

We introduced an algorithm for the *optimal nozzle selection* (*ONS*) problem and showed that the problem can be solved optimally in polynomial time. The complexity of the proposed algorithm is $O(n \log n)$, where n is the number of nozzle types. The *ONS* can also be solved by using more general purpose algorithms discussed in [4], but those algorithms are more complicated to implement and their complexity depends on both the number of nozzle types and the arm capacity. Note also that for some inputs the sorting can be carried out in linear time if bucket sort is applicable. This applicability depends on the distribution of numbers p_i/b_i, which we haven't yet analyzed.

A similar algorithm was introduced for the more complicated case of *budget constrained nozzle selection* (*BNS*) problem. The algorithm runs in pseudo-polynomial time $O(n \log(R) + \log(B))$. We also proposed two heuristics for the problem of selecting nozzles when producing multiple PCB types (*MBNS*). This last problem is not solved optimaly by these heuristics, nevertheless the results were very close to optimal in our tests.

We assumed a simple mapping between nozzle and component types, meaning that a certain component type can be placed with a single specific nozzle type, only. A further study of nozzle selection could investigate the possibility of having secondary nozzle types for each component type. These problems can also be extended to several machines operating in parallel, and to machines organized in production lines, where the nozzles must be distributed between these machines.

Acknowledgments

We thank the anonymous referees for their exceptionally useful comments, in particular concerning the apportionment problem.

References

[1] Y. Crama and J. van den Klundert. The approximability of tool management problems, Technical Report, Maastricht Economic Research School on Technology and Organizations (1996).

[2] Y. Crama and J. van de Klundert. The worst-case performance of approximation algorithms for tool management problems. Naval Research Logistics, 46, pp. 445-462 (1999).

[3] Y. Crama, J. van de Klundert and F.C.R. Spieksma. Production planning problems in printed circuit board assembly. *Discrete Applied Mathematics*, 123:339-361, 2002.

[4] Ibaraki, T. and N. Katoh (1988), Resource Allocation Problems: Algorithmic Approaches, The MIT Press, Cambridge, Massachusetts, ISBN 0-262-09027-9.

[5] M. Ayob and G. Kendall. Real-time scheduling for multi head placement machine. In *5th IEEE Int. Symposium on Assembly and Task Planning* (ISATP'03), pages 18-133, Besancom, France, July 2003.

[6] E.K. Burke, P.I. Cowling, and R. Keuthen. The printed circuit board assembly problem: Heuristic approaches for multi-head placement machinery. In *Proceedings of the Conference on Artificial Intelligence (ICAI2000), Volume III*, pages 1456-62, Las Vegas, NV, June 2001, CSREA Press.

[7] Mika Johnsson, Operational and Tactical Level Optimization in Printed Circuit Board Assembly. TUCS Dissertations No 16. Turku Centre for Computer Science (1999).

[8] Timo Knuutila, Sami Pyöttiälä, and Olli S. Nevalainen. Minimizing the number of pickups on a multi-head placement machine. *The Journal of the Operational Research Society*, 58:115–121, 2007.

Appendix 1

Pseudo-code for the *ONSA* algorithm.

```
onsa_phase1(n; R; p; PS; S1; S2): array of integers (a)
// this is the implementation of the first phase of ONSA.
// the values of R and PS are changed by this algorithm.
// n is number of nozzle types.
// PS is sum of all component amounts given by p[].
  FOR i = 1 TO n DO
    IF R * p[i] < PS THEN
      // nozzle types with one nozzle only.
      a[i] = 1;
      R = R - 1;
      PS = PS - p[i];
      S1 = S1 + i; // add i to S1;
    ELSE
      // nozzle types which might have more than one copy.
      S2 = S2 + i; // add i to S2;
    END IF
  END FOR
  RETURN a
END onsa_phase1

onsa_phase2(n; R; p; PS; S2; S3; a)
// this is the implementation of the second phase of ONSA.
// this algorithm changes a according to phase 2.
// n is number of nozzle types.
// PS is sum of all component amounts given by p[].
  Q = R;
  FOR i IN S2 DO
    // set initial nozzle amount.
    a[i] = R * p[i] / PS;
    Q = Q - a[i];
  END FOR
  // here we sort the content of S2 in decreasing order of p[i]/a[i]
  // so later access is speed up.
  SX = SORT S2 by decreasing (p[i]/a[i]);
  WHILE Q > 0 DO
    // distribute remaining capacity.
    // take the top element of SX (a sorted S2)
    i = TOP(SX);
    a[i] = a[i] + 1;
    // place nozzle type into S3.
    SX = SX - i; // remove the top element from SX;
    S2 = S2 - i; // also remove from S2 to keep it up to date.
    S3 = S3 + i; // add i to S3;
    Q = Q - 1; // remove from the remaining places.
  END WHILE
END onsa_phase2

onsa_phase3(n; R; p; S2; S3; a)
// this is the implementation of the third phase of ONSA.
// this algorithm changes a
// n is number of nozzle types.
  WHILE S2 <> 0 DO // not empty
    i = MAX(p[i]/a[i], i IN S2);
    j = MIN(p[j]/(a[j] - 1), j IN S2);
    // find i and j as described earlier.
    IF p[j]/(a[j] - 1) < p[i]/a[i] THEN
      // if such i and j is found, redistribute 1 nozzle place.
```

```
      a[j] = a[j] - 1;
      a[i] = a[i] + 1;
      S2 = S2 - i; // remove i from S2.
      S3 = S3 + i; // add nozzle i to S3.
    ELSE
      RETURN // no more steps to do.
    END IF
  END WHILE
END onsa_phase3

ONSA(n; R; p): array of integers (a)
// a is an array of integers containing
// the amount of nozzles of each nozzle type.
  S1 = 0;
  S2 = 0;
  S3 = 0; // empty S1, S2 and S3.
  PS = SUM(p[i]); // total amount of components;
  a = onsa_phase1(n; R; p; PS; S1; S2);
  onsa_phase2(n; R; p; PS; S2; S3; a);
  onsa_phase3(n; R; p; S2; S3; a);
  RETURN a
END ONSA
```

Appendix 2

Pseudo-code for the *BNSA* algorithm.

7.1 BNSA

```
BNSA(n; R; p; c; B): array of integers (a)
// this algorithm calculates the nozzle allocation with limited budget.
// a is an array of integers containing the
// amount of nozzles of each nozzle type.
  // set initial nozzle amounts to 1
  FOR i = 1 TO n DO
    a[i] = 1;
  END FOR
  // calculate initial cost.
  cc = SUM(c[i] * a[i]);
  // remaining capacity:
  cap = cap - n;
  WHILE cap > 0 AND cc < B DO
    // find the nozzle requiring the largest step.
    i = MAX(p[i] / a[i], i >= 0);
    IF cc + c[i] <= B THEN
      a[i] = a[i] + 1;
      cc = cc + c[i];
    ELSE
      RETURN a
    END IF
    // nozzle inserted so reduce remaining capacity.
    cap = cap - 1;
  END WHILE
  RETURN a
END BNSA
```

Appendix 3

Pseudo-code for the MBNS1 algorithm.

```
MBNSA1(n; m; R; p; c; B): array of arrays of integers (a)
// a is an array of integers containing
// the amount of nozzles of each nozzle type.
// This algorithm starts from the solution of the problem
// without budget constrain and reduces it by leaving out nozzles
// iteratively until a feasible solution is found.
// calculate optimal pick-up step for each PCB type separately.
  FOR k = 1 TO m DO
    a[k] = ONSA(n; R; p[k]); // optimal sub-problem.
  END FOR
  // calculate the maximum nozzle amount of each nozzle type to array qt.
  // nozzles are reused between PCB types.
  FOR i = 1 TO n DO
    qt[i] = MAX(a[k][i], k <= m);
  END FOR
   // calculate the pick-up steps for each PCB type in A.
  FOR k = 1 TO m DO
    A[k] = MAX(p[k][i] / a[k][i], i <= n);
  END FOR
  // the cost of buying all these nozzles.
  cc = SUM(c[i] * qt[i]);
  WHILE cc > B DO
    // An is the next number of pick-up steps.
    An = 0;
    // l is the PCB type for which nozzles are reduced.
    l = 0;
    FOR k = 1 TO m DO
       // pick-up steps for this PCB if nozzles are reduced.
      At = MIN(p[k][i] / (a[k][i] - 1); qt[i] > 1 and a[k][i] = qt[i]);
      // if a smaller pick-up step increase was found.
      IF (k == 1 OR At < An) AND A[k] < At THEN
        An = At;
        l = k;
      END IF
    END FOR
    // if the next number of pick-up steps is greater than the current one.
    IF An > A[l] THEN
      // reduce number of copies of the nozzles for the PCB type given by l.
      FOR i = 1 TO n DO
        WHILE a[l][i] >= 2 AND p[l][i] / (a[l][i] - 1) <= An DO
          a[l][i] = a[l][i] - 1;
        END WHILE
      END FOR
      // calculate new nozzle count for each nozzle type.
      FOR i = 1 TO n DO
        qt[i] = MAX(a[k][i], k <= m);
      END FOR
      // calculate new number of pick-up steps for each PCB type.
      FOR k = 1 TO m DO
        A[k] = MAX(p[k][i] / a[k][i], i <= n);
      END FOR
      // calculate the new nozzle cost,
      cc = SUM(c[i] * a[i]);
    ELSE
       // no nozzle reduction is possible, just exit
      FOR k = 1 TO m DO
        FOR i = 1 TO n DO
```

```
          a[k][i] = 1;
        END FOR
      END FOR
      cc = B;
    END IF
  END WHILE
  RETURN a
END MBNSA1
```

Appendix 4

Pseudo-code for the MBNS2 algorithm.

```
// this auxiliary routine counts the number of pick-up steps
// caused by a particular nozzle type.
MBNSA2_NOZZLESTEP(n; m; R; p; c; a; nozzle): number of pick-up steps
  sum = 0
  FOR k = 1 TO m DO
    i = MAX(p[i] / a[i], i >= 0);
    IF i = nozzle THEN
       sum = sum + p[i] / a[i];
    END IF
  END FOR
  RETURN sum
END MBNSA2_NOZZLESTEP

// this auxiliary routine searches for the
// nozzle type causing the largest pick-up step.
MBNSA2_NEXTNOZZLE(n; m; R; p; c; a): nozzle type
  max = 0
  nozzle = -1
  FOR i = 1 TO n DO
    step = MBNSA2_NOZZLESTEP(n; m; R; p; c; a; i)
    IF max < step THEN
       max = step
       nozzle = i
    END IF
  END FOR
  RETURN nozzle
END MBNSA2_NEXTNOZZLE

MBNSA2(n; m; R; p; c; B): array of arrays of integers (a)
// Calculate optimal set of nozzles step for each PCB type separately.
// This algorithm starts from assigning one nozzle of each
// type and then iteraitvely increases the nozzle amounts
// until the budget constrain is reached.
  FOR k = 1 TO m DO
    FOR i = 1 TO n DO
      IF p[k][i] > 0 THEN
         a[k][i] = 1
      ELSE
         a[k][i] = 0
      END IF
    END FOR
  END FOR
  // calculate the maximum nozzle number of each nozzle type in qt.
  // nozzles are reused between PCB types.
  FOR i = 1 TO n DO
    qt[i] = MAX(a[k][i], k <= m);
```

```
  END FOR
   // calculate the pick-up steps for each PCB type in A.
  FOR k = 1 TO m DO
    A[k] = MAX(p[k][i] / a[k][i], i <= n);
  END FOR
  // the cost of buying all these nozzles.
  cc = SUM(c[i] * qt[i]);
  WHILE cc < B DO
    // find nozzle causing the most pick-up steps
    i = MBNSA2_NEXTNOZZLE(n; m; R; p; c; a)
    // if such a nozzle is found try to add a copy of
    // that nozzle to the arm settings of various PCB types
    IF i >= 0 AND cc + c[i] <= B THEN
       FOR k = 1 TO m DO
         ns = SUM(a[k][i])
         // there is room for this PCB type and
         // the PCB type can use this nozzle
         IF ns < cap AND p[k][i] > 0 THEN
           // calculate which nozzle is determining
           // the pick-up steps for this PCB type
           j = MAX(p[j] / a[j], j >= 0);
           // if this is the nozzle type requiring
           // the most pick-up steps in this PCB type,
           // then increase the nozzle amounts.
           IF j = i THEN
             a[k][i] = a[k][i] + 1;
           END IF
         END IF
       END FOR
    ELSE
      RETURN a
    END IF
  END WHILE
  RETURN a
END MBNSA2
```

Publication IV

Raduly-Baka, Cs., Knuutila T., Johnsson, M., Nevalainen, O. Construction of Component Tapes for Radial Placement Machines. *Computers & Operations Research*, Electronic edition():1-35, Nov 2009.

Construction of component tapes for radial placement machines.

Csaba Ráduly-Baka [*] Timo Knuutila[†‡] Mika Johnsson[§]
Olli S. Nevalainen[†]

November 30, 2009

Abstract

With a great variation of products, PCB assembling machines must be reconfigured often, but at the same time the efficiency of the assembly process should be kept high. In this paper we consider PCB assembly machines of the *radial type*, which are used for manufacturing robust electronics devices. In this machine type the components are brought to the assembly point by the means of a single component tape, and a robotic arm places them onto a bare PCB one at a time. The component tape is constructed on-line by a separate feeder unit (sequencer) composed of a set of slots storing component reels of various types. While the insertion of components to the tape does not normally delay their placements on the PCB, certain (broad) components delay the processing due to the operation principle of the sequencer, thus increasing the manufacturing time. We study the problem of assigning components to the sequencer in such a way that tape construction delay is minimized, give an integer programming formulation of the problem, and present an optimization algorithm to reduce the component insertion time caused by slow components. The results of this optimization algorithm show considerable improvement against a simple feeder assignment, in case of tape instances containg repeating sequences of components.

Keywords: printed circuit boards, optimization, flexible manufacturing, heuristics, combinatorial optimization.

[*]Elcoteq Design Center Oy, Joensuunkatu 13, FIN-24100 Salo, Finland
[†]Department of Information Technology and TUCS, University of Turku, FIN-20014 Turku, Finland
[‡]Corresponding author: knuutila@it.utu.fi
[§]Valor Computerized Systems (Finland) Oy, Ruukinkatu 2, 20450 Turku, Finland

1 Introduction

In electronics manufacturing, flexibility has become a key attribute of the production process. Manufacturing efficiently specific products with specific machines has been researched extensively in literature. Along with efficiency, more attention has been paid to the flexibility of production in a situation, where the type of products changes frequently and a time consuming machine setup is required when changing the product batch ([1, 2, 3, 5, 6]). The flexibility and efficiency of the machine configuration has a notable impact on the joint manufacturing costs of the products ([7, 8, 9, 32]).

In this paper we study the control of a specific electronics manufacturing machine used in PCB assembly. In so called *radial machines*, a single input tape is used to bring the electronic components from a component feeder to the assembly point, where a robotic head places them on the PCB. This is a traditional machine type which originates from the era of through-hole fixation technique from 1980s. It is however, still actual (RAD5 and RAD8 of Universal Instruments, for example) in cases where the product should be more robust and the power consumption is relatively high, like in amplifiers, TV-sets or control units.

1.1 Radial placement machines

In this machine model the topology of the PCB and other restrictions specific to the PCB type and the robotic *placement head* determine the order in which components (i.e. various types) are arranged on the *component input tape* (called simply the *tape*) of the placement machine and the components arrive in the same order to the assembly point. The components are placed onto the tape by a *sequencer* (called also feeder unit), which comprises a linear array of *feeder slots*. The different components are stored in *component (tape) reels*, which are assigned stationary to the slots of the sequencer. Each individual reel contains a supply of identical components. Duplicate component reels of a given component type may be installed to the sequencer for efficiency reasons.

The tape (used by the placement head) moves (in steps of constant length and time) under the sequencer, and when proper tape locations are aligned with feeder slots containing the required component types, the feeder places those components onto the tape. Due to the physical dimensions of the component reels, the space between two adjacent feeder slots is twice the space between two adjacent tape locations (see figure 1). Because of this, only every second tape location can be inserted with components at a given moment. These placement operations from the sequencer to the tape should

respect the final ordering of performing the actual component placements to PCB. As components are placed into the tape, some component types (so-called broad components, or double pitch components) cause the tape movement to be delayed by a specific amount of time.

The design of the actual placement machine is rather simple. There is a stationary pick-up and place head, which takes a component from the input tape and places it to the PCB. The PCB holding table is movable in the (x, y)-plane and it can be rotated in steps of $\pi/2$ to get the proper final orientation for the inserted component (which is in itself always introduced in the same fixed direction). The operations of the placement machine and the sequencer are synchronized so that one has to wait for the slower operation to be completed.

The machine operates in cycles where each cycle consists of a pick-up and placement operation. Here, the placement head picks up an element from the component tape and simultaneously the PCB holding table is moved so that the proper insertion point of the new component is beneath the placement head and the PCB table is eventually rotated if the component orientation demands it. After this the component will be placed on the PCB.

There are numerous aspects where this kind of manufacturing setup can be optimized. The order of the components on the tape must be determined for each PCB type properly. The capabilities of the palecement head should be taken into account: certain component orders are not allowed due to the dimensions of the head. Then, assuming a given component order, another aspect for optimization is the content of the sequencer, namely how to assign components into the slots of the sequencer, in order to allow the placement of multiple broad components to the tape in the same time. A third aspect of optimization is the actual feeder-to-tape placement process. In this step, multiple feeder slots may be assigned to the same component type, and one should decide at which point to insert these components from the sequencer to the tape of the robotic head.

In these types of machines the components are inserted on the tape either without extra delay, or with a fixed delay of the tape movement, depending on the type (width) of the component. One can generalize this problem into an n-pitch problem by defining an arbitrary number of delay times. Such a problem would, of course, be at least as complex as the practical (2-pitch) feeder assignment problem (see chapter 4). In the present paper we focus on the 2-pitch case, which is present in the control of radial machine sequencers. We will also show that the heuristic algorithm introduced by this paper can be easily adapted to the n-pitch case.

1.2 Previous research

There are slightly differing classifications of component placement machines, see for example Moyer and Gupta [20], Tirpak et al. [21], or Ayob and Kendall [19].

In their recent survey M. Ayob and G. Kendall [19] (see also [22]) classified the surface mount device (SMD) placement machines to five categories. These include dual-delivery, multi-station, turret-type, multi-head and pick-and-place machines. Research on the control of these machine types has been very active durring the last twenty years. While the operation principles of different machine types differ at critical points, techniques used for optimizing the control of placement machines are similar in most cases. We therefore briefly characterize the different machine types and mention few references (older and recent ones) for research on them.

Dual-delivery machines have a head and a feeder unit on both sides of the machine, see Ahmadi et al. [24], Wilhelm et al. [37] and Choudhury [25] for control optimization of this machine type.

Multi-station placement machines contain several identical machine modules connected to each other with a conveyor. The modules work in parallel and each of them perform a subtask for the PCB, see Grunow et al. [27] and Csaszar et al. [26]. Reconfigurable versions of this machine type have recently gained their popularity.

Rotary turrent machines have a revolver head with multiple nozzles. The head is at a fixed position and the PCB holding board and the linearly organized feeder unit are moveable, see e.g. Crama et al. [28] and Ho et al. [29]. While the machine type is extremely fast (called also "chip shooter"), the large mass forces cause mechanical stress and the physical dimensions of the machines tend to be rather large.

Multi-head placement machines differ from the rotary turret machines in the sense that their placement head is movable in two or one dimension. In a common design of the machine type PCB and feeders are stationary and head moves along two gantries (so-called gantry machines), the head may be of revolver type or a linear array of spindles, see Van Laarhoven and Zijm [30], Du and Lu [31], Sun and Lee [33], and Park and Kim [34]. The machine is nowadays popular due to its flexibility and accuracy.

Sequential pick-and-place machines have only one nozzle in their placement head, and the PCB table and feeder unit are either stationary or movable, see e.g. Ball and Magazine [38], T. Leipälä and Nevalainen [17, 18], Fu and Su [35] and Ayob and Kendall [36] for literature.

Radial placement machines belong to the class of sequencial pick-and-place machines.

Ball and Magazine [38] model the placement sequencing problem of pick-and-place machines as a directed postman problem, whereas Leipälä and Nevalainen [17] treat it as a TSP-problem and quadratic assignment problem. Kumar and Li [39] used linear programming for feeder assignment and placement sequencing. They improve the initial machine control heuristically using a number of efficient heuristics (local search with 2- and 3-changes).

Recently, modern metaheuristics, among others evolutiuonary algorithms, tabu search, simulated annealing, swarm optimization have been succesfully used to solve several different machine control problems. An advantage of these methods is their ability to consider the feeder setup and placement sequencing problems jointly. Then, the objective function, which is most commonly the total manufacturing time per PCB, is expressed as a function depending on feeder assignment and placement sequence. The value of the objective is then calculated by a function which abstracts the operation of the machine. The level of the abstraction strongly influences the quality of the solution and also the need for computing time.

As an example of this kind of approach, Fu and Su [35] applied a genetic algorithm, simulated annealing and tabu search to solve the joint control problem of a pick-and-place machine. The current trend with PCB component placement problems, and with hard combinatorial problems in general, is the hybridization of a metaheuristics with problem specific local search. As an other trend, there is the increase of the number of research articles using exact solution methods. While the theoretical computational complexity puts limits to our ability to find exact solutions to large problem instances, there are instances of realistic size, which have been solved optimally by recent efficient problem solvers; c.f. for example traveling salesman problems for job grouping problems (Knuutila et al. [23]).

1.3 Feeder construction

In this research we *focus* on the *assignment of component reels into the feeder slots* and the *feeder-to-tape placement process* of the components performed by the sequencer[1].

While we concentrate on the sequencer optimization, we note that problems occuring in this task are closely related to optimization of nozzle change operations and component pickup operations of multi-head machines.

In the machine type studied here, the component placement time is determined by the width of the component, also called the component *pitch*.

[1]The problem was originally proposed by a developer of PCB control programs (Valor Finland), who has noted that sequencer operations form a bottleneck in the operation of RAD5-machines.

Components of a narrow pitch are inserted into the tape as an interation of two-step process:

1. advance component tape and unwind component feeder reel in parallel.
2. place component to tape and pick-up component to placement head in parallel.

Thus narrow components are inserted at the same speed as the tape moves (in optimal case). It is also possible to place multiple narrow pitch components simultaneously, but that will not have any positive effect on the tape movement delay, because all components have to pass the placement head.

Double pitch components also occupy one feeder slot each, but their placement demands an extra processing step from the sequencer. Durring this step the component tape is stopped and the placement head can not thus pick up a new component [2]. The extra time of the double pitch components does not, however, depend on how many such components are brought from the sequencer to the tape in parallel. Placing one double pitch component causes the same amount of delay as a simultaneous placement of, say, three double pitch components. If multiple feeder slots align with correct tape locations at the same time (of the tape stop position), so that the slots contain the components which need to be placed just into these tape locations, placements to these locations can be done at the same time. We can thus decrease the potential delay caused by the sequencer by creating a feeder slot assignment where such alignment scenarios are frequent. It is therefore natural to take as the optimization objective the minimization of the number of the extra tape stoppings.

In this research it is assumed that the *component ordering on the tape has been already established by some other decision process.* The assumption is, at least partly, motivated by the precedence constraints of the placements originating from physical dimensions of the components and the placement head. Because of their dimensions and location, certain component orders cannot be managed by the machine, since the placement head may accidentally touch and damage already placed components.

We introduce an algorithm to address the feeder assignment problem of assigning double pitch component types to the feeder slots. The algorithm selects frequently occurring patterns of component types on the component

[2]The reason for extra stop is that double pitch components are oriented on the component feeder tapes the long side aligned with the tape direction and the movements of the sequencer are synchronized in steps of "one pitch movements".

input tape and tries to reproduce these patterns in the sequencer. The algorithm takes into account the fact that the feeder slot size is such that at a given moment only every second tape location is aligned with a feeder slot. A natural assumption is that the size of the sequencer (the total number of feeder slots) is much smaller than the length of the tape to be constructed. An other motivation for fixed component ordering is a hierarchical view of the machine control; for each particular component placement ordering we have to optimize the sequencer operations.

Having the feeder slots assigned with the necessary component types, the sequencer inserts the neccessary components onto the tape as the tape is moved stopwise under the feeder. Because multiple slots may contain the same component type, it must be determined at which feeder-to-tape alignment these multiply occurring components are inserted. We show that this problem is basically the *Minimum Set Cover* problem [14], and a well known approximation algorithm used for the *Minimum Set Cover* problem can be used in tape construction also.

We test the feeder assignment algorithm on practical data including single pitch and double pitch components. The results are compared to those of a brute force algorithm and a simpler (naive) heuristics. In addition to the above practical contributions we give an integer formulation to the feeder assignment problem and discuss its relation to the turnpike problem [10]. We show that the later problem is reducible (in P-time) to the feeder assignment problem.

2 The feeder assignment and component tape construction problem

In this section we formulate the *Feeder Assignment And Component Tape Construction* (FACTC) problem. The following assumptions are made:

1. The order of the components on the tape of the placement machine is fixed.

2. A sequencer (feeder unit) is used to insert the components onto the tape, in the specified order.

3. One feeder slot of the sequencer can contain components of a single type, only. Some of the slots may be unused.

4. Component types can be of single or double pitch.

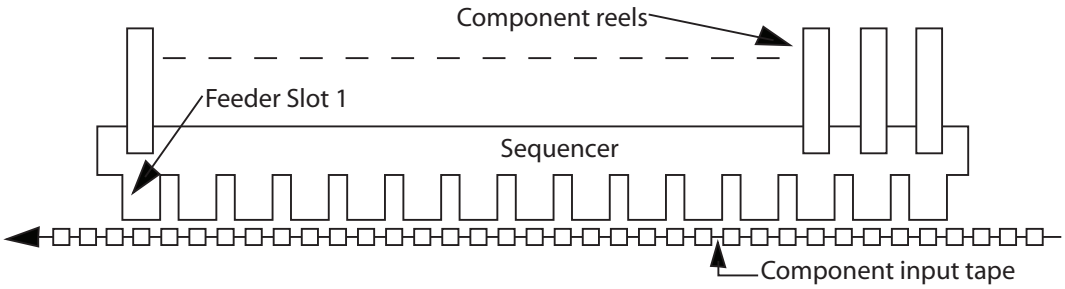

Figure 1: Component input tape moving leftwards under the sequencer. Every second input tape location is under a feeder slot. Component reels are connected to the feeder slots.

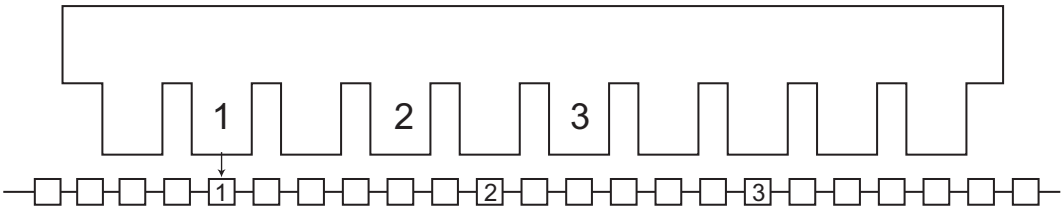

Figure 2: When feeder slots of double pitch components are not properly aligned, each component insertion causes a separate delay. (The numbers on the tape and the sequencer indicate the component types.)

5. The insertion of a double pitch component onto the tape blocks the tape movement and causes a delay of a constant time (c) in the operation of the component assembly robot.

6. The startup and shutdown time of the component assembly and tape construction process are considered fixed.

7. The distance between two neighboring feeder slots is twice the distance between two neighboring tape locations, so at any given moment, the sequencer can insert a component onto every second tape location only.

Figure 1 illustrates how the tape moves under the sequencer, and how the feeder slots are aligned with tape locations at a given moment. The tape moves stepwise leftwards, and at each step the sequencer can insert into the tape locations that are under each slot. Since there is only one component type in a slot, only those slots can make an insertion, which contain the component type marked for the tape location aligned with the slot.

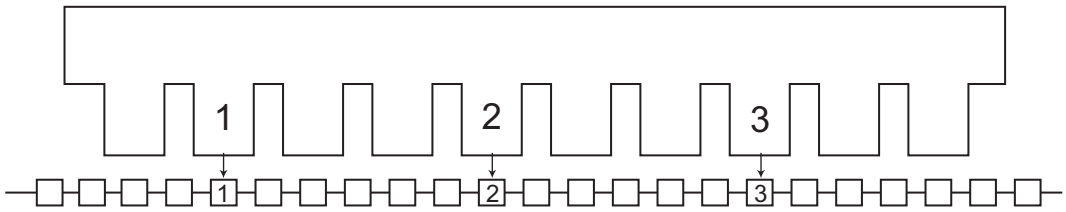

Figure 3: If feeder slots and tape positions are properly aligned, three double pitch components can be inserted together causing a single delay.

The feeder assignment has an effect on the delay of the tape movement caused by the insertions of the double pitch components. Figure 2 illustrates a feeder configuration with three different double pitch components. In this example, the tape movement is delayed by $3c$, since each component is inserted separately. This delay can be reduced by inserting several double pitch components in parallel. This can be done by placing the double pitch components in such feeder slots, that they align simultaneously to their intended tape positions, see figure 3. In this case the 3 components are inserted in one step, and the delay caused by these insertions is reduced from $3c$ to c.

The feeder assignment may be disadvantageous for a given tape configuration, since there may be a weak match between the patterns of double pitch components of the tape and sequencer. For example, the sequencer may contain an allocation which allows parallel placement of certain components, but does not contain the possibility of performing another more frequent parallel placement which appears periodically in the tape.

2.1 Notation

In the feeder assignment problem we use the following notations:

l - the total number of components to be inserted in the tape;

n - the number of different component types;

$N = \{1, ..., n\}$ - the set of component types;

$\overline{t} = \{t_1, t_2, ..., t_l\}$ - the components on the tape from left to right, where $t_i \in N$ is some component type;

$\overline{w} = \{w_1, w_2, ..., w_n\}$ - the pitches of the component types, where w_i is either 0 indicating a narrow component, or 1 indicating a double pitch component;

s - the number of slots in the feeder;

$\overline{f} = \{f_1, f_2, ..., f_s\}$ - a feeder assignment, where $f_i \in N \cup 0$ specifies the component type in slot i. If $f_i = 0$, no component type is assigned to that slot.

p - the number of double pitch component types, $p \leq n$. The value of p can be calculated by summing up values of 1 in $\overline{w}$

m - the number of slots in the feeder used for double pitch components, $p \leq m \leq (s - n + p)$. The value of m is an input parameter to our problem.

$L = \{i_1, i_2, ..., i_h\}$ - where $i_j \in 1, ..., l$ give the locations of the double pitch components on the tape (i.e. for $i \in 1, ..., l$: $i \in L$ if and only if $w_{t_i} = 1$), and h is the number of double pitch components to be placed into the tape.

The feeder must accommodate at least one instance of each component type (as a minimum requirement). It is typical that the size of the feeder is larger than the number of component types and it is possible to assign multiple slots with the same component type. Denote by $\overline{f}$ an assignment giving the minimal delay in tape construction. As mentioned before, it is supposed that the position of single pitch components in the feeder does not cause any tape construction delay. Nevertheless, we may still want to allow more feeder slots for single pitch components than the minimum necessary $n - p$ slots [3]. The input parameter m specifies the number of slots used for wide components.

2.2 Integer programming formulation

We next present an integer programming formulation for the unified problem of feeder assignment and tape construction (FACTC). In this formulation one can (as observed in the beginning of Section 2) omit the further consideration of the $n - p$ single pitch components and only look for the double pitch components. Single pitch components are supposed to occupy (in some order) the feeder slots remaining free after allocating the double pitch components.

We must account for the practical aspect of the feeder, where at any given moment; the sequencer can insert a component onto every second tape location only. This can be modeled by using a larger feeder of size $s' = 2 \cdot s - 1$ and forbidding component assignment to every second feeder slot.

[3] Some frequently used components may be fed by two reels for practical reason.

We may assume, without the loss of generality that types $1, ..., p$ are the 2-pitch components.

Let

$$g_{ij} = \begin{cases} 1 & \text{if } i \text{ is odd and feeder slot } i \text{ contains 2-pitch component type } j \\ 0 & \text{otherwise} \end{cases}$$

where we limit the use of feeder slots to slots of odd locations, so that proper spacing is maintained.

A legal feeder assignment is defined by the following constraints.

For each double pitch component type, there must be at least one slot assigned with that component type:

$$(i) \textit{ for all } j \in 1, ..., p \qquad \sum_{i=1}^{s'} g_{ij} \geq 1$$

The number of slots used for double pitch components is m:

$$(ii) \qquad \sum_{i=1}^{s'} \sum_{j=1}^{p} g_{ij} = m$$

A slot can contain at most one double pitch component:

$$(iii) \qquad \sum_{j=1}^{p} g_{ij} \leq 1 \text{ for all } i \in 1..s'$$

2.2.1 The tape construction step

Constants (i)-(iii) define a *legal feeder assignment* for the double pitch components. In order to determine an *optimal feeder assignment*, we must also consider the tape construction steps used to insert the components from the feeder into the moving tape. Insertion of a component from feeder to tape can occur when a feeder location with an assigned component type is aligned with a tape location assigned for the same component type. There are at most $l + s'$ feeder-to-tape alignments, that is, there are $l + s'$ steps from the initial to the final position of the tape (see figure 5). Observe, that at the initial and final positions, an insertion to the tape does not occur, and insertion of a double pitch component is feasible only at a subset of these positions.

At each feeder-to-tape alignment, a given tape location may or may not be inserted with a component, depending on whether the aligned slot contains

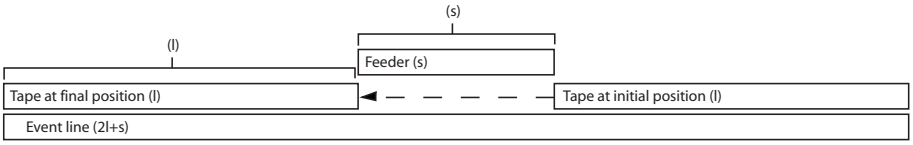

Figure 4: The tape moves under the feeder from an initial to a final position, defining a feeder/tape alignment event line.

a proper component type, and whether insertion at that alignment leads to an optimal solution (i.e. minimal delay).

Let $t \in [k..s' + k - 1]$ denote a feeder-to-tape alignment. The so-called *feeder-to-tape alignment event line* visualizes the orientation of the sequencer in relation to the tape (see figure 5). The event line includes $2l + s'$ feeder-to-tape alignment positions which are indexed from left to right as $1, 2, ..., 2l+s'$. Let s'_{kt} denote the position of the kth tape element on the event line at moment t. Then we have $s'_{kt} = l + s' + k - t$.

Let a_k denote the component type at location k on the tape. For alignment t, the insertion of a component to location k of the tape from feeder slot i (numbered from left to right as $1, 2, ..., s'$) is possible if:

(a) the feeder slot i contains the required component type:

$$g_{ia_k} = 1 \tag{iv}$$

and

(b) the tape location k is aligned with the feeder slot i:

$$S_{kt} = l + i. \tag{v}$$

Formula (v) gives us $l+s'+k-t = l+i$ from which it follows $i = s'+k-t$ (when $t = 1$, the first tape location ($k = 1$) is aligned with the last feeder slot ($i = s'$) and the tape moves leftwards (see figure 5), for $t = s'$, the first tape location ($k = 1$) is aligned with the first feeder slot $i = 1$). Of course k could be out of bounds ($k \in 1..l$) meaning that there is no tape under feeder slot i.

Let q_{kt} denote the decision variable whether the component at tape location k is inserted at alignment t:

$$q_{kt} = \begin{cases} 1 & \text{a 2-pitch component is inserted to tape location } k \text{ at alignment } t, \\ 0 & \text{otherwise.} \end{cases}$$

Then for all $k \in 1..l$:

$$(vi) \qquad \sum_{t=1}^{l+s'} q_{kt} = 1$$

meaning that a tape location is inserted with a component exactly at one alignment. Since the insertion can occur only if both *(iv)* and *(v)* hold at moment t, we have :

$$(vii) \qquad q_{kt} \leq g_{ia_k}, \text{ where } i = s' + k - t.$$

Thus, the values of q_{kt} give both the feeder assignment and the component tape construction sequence.

2.2.2 Objective function

We want to minimize the number of extra stops (delays) of the tape caused by insertions of the double pitch components. For any given tape alignment t, $(1 - q_{kt})$ is 0 if a double pitch component is inserted to the kth location of the tape, and 1 if insertion of the double pitch component does not occur. For an alignment t the product

$$(viii) \qquad \prod_{k \in L} (1 - q_{kt})$$

is 1 only when no insertion of a double pitch component occurred at that alignment. At any given alignment t, the tape either moves on, or stops so that component insertion can occur. Therefore minimizing the number of stops means maximizing the number of alignments where no double pitch components are inserted. The total number of alignments with no double pitch insertions is the sum of the above product:

$$(ix) \qquad \sum_{t=1}^{l+s'} \prod_{k \in L} (1 - q_{kt})$$

The objective function to minimize in the combined problem of feeder assignment and tape construction is then:

$$(x) \qquad l + s' - \sum_{t=1}^{l+s'} \prod_{k \in L} (1 - q_{kt})$$

that is, because there are a fixed $(l + s')$ alignments where component insertions can occur, so maximizing *(ix)* is equivalent to minimizing *(x)*. (Note that one could easily modify the above formulation to minimize the total number of times components are moved from feeder slots to the tape (i.e. maximization of parallelism). Then, q_{kt}-settings should be solved for all k-indexes by extending the product in (x) for indexes $k \in [1, l]$.)

By introducing a new variable z_t, the above objective function of the *FACTC* problem can be linearized to the form:

$$(xi) \qquad \sum_{t=1}^{l+s'} z_t$$

with the following constraint on z_t:

$$(xii) \qquad z_t \leq (1 - q_{kt}) \text{ for all } k \in L, t \in [1..l + s']$$

3 Brute force search

The feeder assignment and component tape construction problem can be solved by the means of a brute force search algorithm as follows.

3.1 Brute force search for feeder assignments

The brute force algorithm for the *feeder assignment problem* (called *Brute Force for Feeder Assignment* or *BFFA*) builds all possible feeder configurations, evaluates the tape construction delay for each of them, and selects the one resulting in the smallest number of tape movement delays. The complexity of the brute force approach depends on the feeder size s and the number of component types n. As mentioned, the positions of the single pitch component types are those remaining free after assigning the double pitch components.

The *BFFA* algorithm enumerates all possible ways to store the p double pitch components in m slots. There are $\binom{s}{m}$ possible slot subsets of size m in the feeder, and each of them can be populated in p^m different ways with p double pitch component types. Some of these ways will be unfeasible, since a valid feeder assignment must contain components of all types. In order to reduce the number of such assignments, we separate the m slots into two groups, one having p slots and containing one double pitch component of each type, an the other having $m - p$ slots containing any combination of double pitch components. The first group generates $p!$ possible configurations and the second group p^{m-p} possible configurations. Each configuration of these

two groups is then combined into m slots. Because there are $\binom{m}{p}$ possible ways to combine these two groups, the number of slot assignments in of the brute force algorithm is:

$$p! \cdot p^{m-p} \cdot \binom{m}{p} \cdot \binom{s}{m}.$$

At each step, the brute force algorithm calls to the component tape construction algorithm to evaluate the feeder configuration, and stores the best feeder configuration found.

In a typical PCB, the number of different component types can be large (for example $n = 60$ as a small case), and more than half on these can be of double pitch type (i.e. $p = 30$). The size of the sequencer is also large enough to accommodate multiple copies of components (i.e. $s = 120$). In order to run the brute force algorithm with $m = 60$ (maximally two double pitch components of each type), the number of possibilities to be checked becomes too large for practical computation (approximately 10^{128}). This motivates the search of a heuristic, which may not provide the optimal result but runs in practical time.

3.2 Brute force search for tape construction

The *component tape construction* should be done in optimal way for each feeder assignment. This problem can also be addressed with a brute force algorithm (lets call it *Brute Force for Tape Construction* or *BFTC*). The tape moves leftwards under the feeder (see figure 5), and at a specific alignment the feeder inserts a set of (one or more) double pitch components onto the tape. The position where the insertion is performed may influence the forthcoming choices and the total number of double pitch insertion operations (see figure 6, where it is beneficial to postpone the insertion of the leftmost component of type "1", to the alignment shown).

As mentioned above, the brute force algorithm tries out every possible feeder-to-tape alignment and for every alignment all the possible subsequent alignments. This leads often to m^r different possibilities, where

$$r = \sum_{i=1}^{l} w_{t_i}$$

is the number of double pitch component types inserted into the tape (w_{t_i} is the pitch of a component at the tape location i, as defined earlier), which are difficult to evaluate exhaustively in practice. The algorithm could be improved by keeping track of the optimal choices made for every position on the tape.

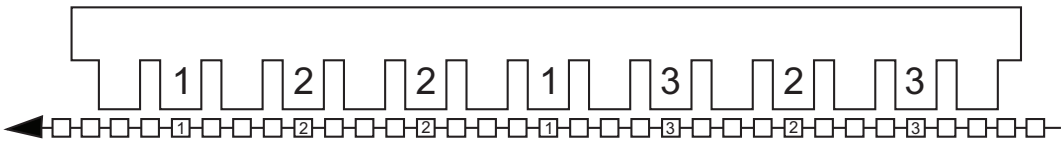

Figure 5: Parallel insertion of 7 double pitch components.

4 Time complexity

In the following, we relate the decision version of *FACTC* problem to the well known *turnpike decision problem* (*TP*). Altough, the *turnpike* problem is not similar to our *FACTC* problem, we will show later that for any *turnpike* instance it is possible to construct a *FACTC* problem instance, which if solved optimally, will give a turnpike solution.

In the *turnpike problem* ([10, 11, 12]) a multiset D of $\binom{n}{2}$ point distances is given, and the goal is to reconstruct the actual locations of n points on a line so that the distances between all pairs of the points match exactly the values in the multiset. The turnpike problem does not label the distances with pairs of points, so we do not know which distance resulted from which pair of points.

The turnpike problem has been extensively studied by literature mainly because of its use in DNA sequencing, where it is also known as the *Partial Digest Problem*. In DNA sequencing, a large amount of short sequences must be assembled into the complete DNA sequence. The complexity of the turnpike problem is currently not known, although there are instances where the number of solutions is exponential [13].

However, we can characterize the *TP* and *FACTC* decision problems by proofing the following lemma [4].

Lemma 1. *The turnpike problem is reducible in polynomial to the combined feeder assignment and component tape construction problem: TP* $\leq$ *FACTC.*

Proof. Consider an arbitrary *TP* instance T, where $q = \binom{n}{2}$ and $D = \{d_i | i \in 1..q\}$ is a multiset of distances of n points. For convenience, it is assumed, that all point distances are integer values, which does not reduce the complexity of *TP* (see [10], [11]). A multiset D of integer numbers is a valid point distance set, if there is a set of points on a line whose inter-point distances match exactly the multiset. On the other hand, multiset D can be

[4] We define the decision version of *FACTC* to decide whether there is a solution of the *FACTC* optimization problem such that the number of double pitch placement stops is at most a given constant R.

degenerate, meaning that it is not possible to construct a set of points on a line which result in D distances. Such a case is, for example, where all values in D are equal.

We want to construct a $FACTC$ decision problem F, which has a "YES" instance if and only if T is a "YES" instance of TP problem. Therefore, let $M = max_{i \in [1..q]} d_i$ be the largest distance in D and let $S = M + 2$ denote the number of feeder slots required by two components at distance M. We construct a $FACTC$ problem instance using a single double pitch component of type c, and a feeder with size S. Let the input tape consist of q pairs of double pitch components spaced at distances given by d_i, $i = 1..q$, and the space between these pairs is S, so the input tape will be:

$$\overline{t} = cd_1cScd_2cScd_3cS...cd_qc$$

We then define $FACTC$ instance F with component tape $\overline{t}$, feeder size $S = M + 2$, and constant $R = q$. Then, double pitch components that are at distance S (or larger) cannot be inserted in parallel because the feeder size is S. For that reason, the feeder can insert at most 2 double pitch components at distance d_i where $d_i \in D$ (like cd_ic) in parallel. This means that the component tape construction with the optimal feeder assignment requires at least q steps, since there are q sub-sequences cd_ic in $\overline{t}$.

The resulting optimal feeder assignment can be considered as a line segment, and the slots assigned with double-pitch component c are points on this line segment.

1. Suppose that D is a degenerate multiset of distances. Then the points on the line segment corresponding to the feeder assignment will not define the distances in D, which can be verified in polynomial time.

2. On the other hand if D was a valid point distance set, there exists a point set which reconstructs D, which means that there is a feeder assignment, equivalent to that point set, which can insert in parallel 2 double pitch components at d_i distance for any i. This means that this hypothetical feeder assignment constructs $\overline{t}$ in q steps, which is the minimum amount of steps to construct $\overline{t}$.

Lets assume, that there is a algorithm called $OFACTC$, which solves the $FACTC$ decision problem. This means that all sequences cd_ic are inserted in parallel (but spacing between c's prohibits placement of more c-components). Therefore, this feeder assignment will also be a valid point set on a segment obeying d_i-distances.

The above means that using a hypothetical optimal feeder assignment algorithm, it is possible to determine in polynomial time for an arbitrary

multiset D whether it is a valid point set distance or not, and if it is, it is possible (in polynomial time) to reconstruct the point set.

□

It is not known whether the turnpike problem is NP-complete or not. The exact complexity class of *FACTC* remains therefore still open. All the current solutions for the general case of the turnpike problem are pseudo-polynomial algorithms. In case of the feeder assignment problem it is still possible that a pseudo-polynomial algorithm exists but for practical needs, the above lemma motivates the search for a heuristic algorithm.

4.1 Component tape construction

As described earlier, having a feeder assigned with component types, the second part of the process is to determine the steps where to stop the tape to allow the insertion of one or (preferably) multiple double-pitch components. The number of such stops should be minimized.

Earlier we denoted with L the set of h tape locations where double-pitch components are to be placed. Let P_t denote the set of tape locations which can be placed at alignment t where $t \in 1..l + s$ as denoted earlier. There will be a number of t-indices for which $P_t = \oslash$, and those can be ignored. Our goal in the *component tape construction* is to find a minimum number of sets P_t such that their union is L. This basically is the *Minimum Set Cover* problem, which is known to be NP-hard [15].

Actually, our problem is a sub-problem of the Minimum Set Cover problem, where the elements of set L are on a line, and the size of subsets P_t is limited by the feeder size. A formal proof of whether the Component Tape Construction is as hard as the (being a sub-problem of) Minimum Set Cover problem is not in the scope of this paper. This relationship motivates the use of a greedy heuristic for component tape construction, which is commonly used for the Minimum Set Cover problem [14].

5 A simple method for feeder assignment

A simple method for constructing a feeder assignment is to assign m double pitch components to random feeder slot locations. The final feeder must contain all component types present in the tape (in order to be able to print the tape). For each double-pitch component type, the number of components in the feeder of that type should be proportional of the number of components of that type in the tape. Let b_i denote the number of double-pitch components of type i in the tape. The maximum number of components of type

i in the feeder is then $a_i = \left\lceil \frac{m*b_i}{h} \right\rceil$, where h is the total number of double-pitch components in the tape, as defined in the notation. The *simpleassign* algorithm is described by the following steps:

1. Count the number of components to be placed into the final tape for each double-pitch component type.
2. Proportionally to the above counts, an a_i number is assigned to each double-pitch component type so that the sum of these numbers does not exceed m (limit of double-pitch component types in the feeder).
3. For each double-pitch component type, a_i components are assigned to random slot locations in the feeder.
4. The remaining feeder slots (m is always less than the actual feeder size), can be used for narrow component types.

6 An improved heuristics for feeder assignment

In this section we describe a heuristic algorithm for the feeder assignment problem, that improves on the algorithm of Section 5 by considering short, frequently occuring sequences of double pitch components from the tape. The design of this heuristics is motivated by the relationship of $FACTC$ to the TP problem shown earlier. In some turnpike algorithms, the distances of set D are placed on an open line as line segments, trying out various possibilities using backtracking from the longest distance of D. Our approach is similar in that we want to reserve feeder slots to double pitch components, so that the distances between these slots in the feeder reproduce as many similar distances on the tape as possible.

Having a feeder with m double pitch component types and p double pitch components on the tape, we observe that in an ideal scenario, if we could insert with all m slots at the same time onto the tape, the number of insertion steps would be $\left\lceil \frac{p}{m} \right\rceil$. This would be possible only if the tape would contain the same repeating pattern of double pitch components, as the pattern of m double pitch components in the sequencer. For example in the particular scenario of figure 6, the feeder is able to insert three double pitch components at once and then let the tape move and align again for three parallel insertions. This would be an optimal feeder allocation and placement tactics.

In practice the input is not as systematic, and may not allow such a perfect feeder assignment. On the other hand practical scenarios may still

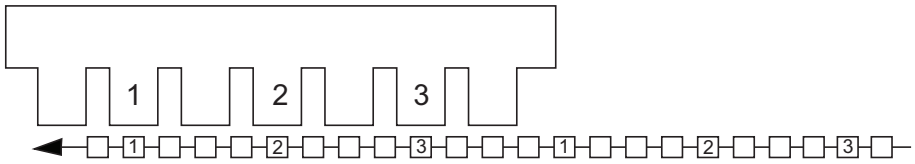

Figure 6: An optimal scenario of component insertions for double pitch components. The tape contains the same repeating pattern of components in the sequencer.

exhibit some pattern repetitions. These occur inside a placement task of a single PCB and also because the same tape pattern is used to assemble multiple PCBs of the same type. A useful method must therefore be able to handle non-systematic tape content and at the same time recognize and exploit repeating patterns if they occur.

The length of the sequencer (number of slots) s limits the maximum length of the pattern which could be relevant for recognition and (if possible) can be inserted at once. Our heuristic algorithm looks for patterns up to maximum length s in the tape and creates a slot assignment in the feeder in order to reproduce the most often occurring patterns.

6.1 Constructing the pattern set

In order simplify the representation for the algorithm, we transform the initial tape representation into an alternative form. This form is a series if double pitch components and spaces between them:

$$\overline{u} = (e_0, c_1, e_1, c_2, e_2, ..., c_d, e_d),$$

where $c_i \in [1, n]$ is a double pitch component type. The integer value $e_i \geq 0$ specifies the number of intervening tape locations reserved for single pitch components.

The length of this array is $2 \cdot d + 1$. We can ignore in this notation the initial space e_0 occurring before the first double pitch component, and the last space (e_d). Let $\overline{u}(i, j)$ denote a subsequence of $\overline{u}$ between indices $i < j$ so that $\overline{u}(i, j) = (c_i, e_i, c_{i+1}, e_{i+1}, ..., c_j)$. Such patterns, inserted in one step, can speed up the tape construction process. Determining the number of occurrences of a pattern in a sequence (or string) has been extensively discussed in [4], as the *Exact Matching Problem*. We skip the details of these algorithms, most of the exact matching algorithms from [4] can be used here.

If the sequencer contains the component pattern $\overline{u}(i,j)$, it is not necessary, that all these components are inserted at once. The tape may contain matching subsequences of $\overline{u}(i,j)$ at various locations, and the sequencer will be able to match those also, if $\overline{u}(i,j)$ is part of the sequencer contents. For the current approach, we do not consider more complicated super-patterns (i.e. patterns with holes like $\{c_i, e_i, c_{i+3}, e_{i+3}, ...\}$), since that would make the number of such patterns exponentially large. It is left for further study, what the effect of considering those super-patterns would be on the final tape delay. The number of such super-patterns in the tape of length l can be as much as $2^{m+1} \cdot l$, where m the maximum length of such pattern. When considering only continuous patterns, their number is not larger than $m \cdot l$. Nevertheless, parts of $\overline{u}(i,j)$ are considered separately by the pattern enumeration process, and placed in the feeder at such distances which minimize the final delay. (If we would consider instead super-patterns of $\overline{u}(i,j)$, we would fix these distances into locally suitable values, i.e. suitable for these super-pattern only.)

One of the initial conditions for the feeder assignment is that the sequencer inserts components simultaneously into either even or odd tape positions. This also means that in a particular $\overline{u}(i,j)$-pattern there is an odd number of tape locations between any two tape location where parallel insertion can occur (values of e are odd).

If pattern $\overline{u}(i,j)$ contains even e-values, it is separated into two patterns, one where the double pitch components on the tape are at odd locations and the other where they are in even locations. Let's denote them with $\overline{u}_1(i,j)$ and $\overline{u}_2(i,j)$. Both of these patterns will contain spaces of odd e-values.

Another limitation is given by the feeder size. We denote the size of the space occupied by a pattern in the feeder by $S(\overline{u}_x(i,j))$ where x stands for 1 or 2:

$$S(\overline{u}_x(i,j)) = j - i + 1 + \sum_{k=i}^{j-1} (\frac{e_k}{2}), \tag{1}$$

where $(j - i + 1)$ gives the number of double pitch components aligned with their feeder slots, and the sum gives the spaces in the slots in-between. We limit our set of patterns by the condition

$$S(\overline{u}_x(i,j)) \leq s. \tag{2}$$

A simple analysis shows that the worst case time performance of the pattern set construction step is $O(l^2 m^3)$ (assuming a naive algorithm). There are at most $l \cdot m$ patterns to consider, and it takes at most $l \cdot m^2$ steps to

check for each pattern whether it is already in the pattern set. The pattern construction step can be improved by using suffix trees (see [4]).

6.2 Building a feeder assignment from the pattern set

Let $\Psi = \{\psi_1, \psi_2, ..., \psi_r\}$ denote a set of r patterns, where each ψ_k satisfies condition (2). For each pattern $\psi_k \in \Psi$ we denote with v_k the number of occurrences of ψ_k in $\overline{u}$.

The patterns in Ψ are sorted in decreasing order by v_k. This may result in short but frequently occurring patterns preceding longer/less frequent patterns. If the number of occurrences of two patterns are the same, the longer pattern (equation (1)) precedes the shorter one [5].

The elements of Ψ are then considered in the sorted order for placing into the feeder. If a pattern is successfully inserted into the feeder (there is room for it and improves the final tape delay), it is removed from Ψ and all remaining patterns which now can be found in the new feeder setup are also removed from Ψ.

Placing a pattern into the feeder means finding a slot position to which the first double pitch component of the pattern can be assigned and the remaining elements of the pattern can be assigned to slots at distances given in the pattern. In this step, a double pitch component can be assigned to a slot if it is either free or it already contains the same type of double pitch component.

In order to a pattern to be insertable into the feeder at a given position, the resulting feeder must satisfy two conditions: the number of double pitch components must not be greater than m, and the number of components of each double-pitch component type must not be greater than a_i. The value of a_i is calculated in a similar way for each component type i as in the *simpleassign* algorithm described earlier.

If, by placing the pattern, the feeder would contain more than m double pitch components, or more than a_i components of type i, the pattern is ignored and removed from Ψ and the algorithm proceeds with the next pattern in Ψ. When there is no more space to place new patterns into the feeder, the algorithm stops and returns the resulting feeder.

The detailed implementation of the feeder assignment and tape construction heuristics would be several pages long. Therefore, the ideas are outlined by the following pseudo routines:

[5] We also evaluated the scenario where the sorting is done based on the product of pattern occurrences and pattern length, which more accurately represents the actual component count associated with a pattern. No differences in the results between the two models (occurrence only or occurrence times length) were observed.

```
// assigns a sequence of double pitch components to
// the sequencer at a random location.
01  procedure addfeeder:
02  input f, r, B,
03  // B - array of remaining patterns
04  // f - feeder to be constructed
05  // r - pattern to be added
06    X[s] = an array where X[i] is 1 if slot i is empty and
07       has not yet been tried out this algorith,
08    i = a random slot so that X[i] = 1
09    // cf is a copy of the feeder, which is locally modified.
10    cf = f;
11    add r to cf at position i;
12    X[i] = 0;
13    if cf is a valid feeder
14       f = cf
15       remove r from B
16    else
17       repeat from line 08
18    end
19  end addfeeder.
20

// builds a feeder assignment from a component tape input
21  procedure assignfeeder:
22  input values l, n, t, w, s, p, m
23  // t - component tape
24  // l - tape length
25  // w - gives the component pitches for each component type
26  // s - number of slots in the feeder
27  // p - number of double pitch component types
28  // m - maximum number of double pitch components in the feeder
29  // n - number of different component types (length of w)
30     // u - compact representation of the tape.
31     let u = compact(t, w, l);
32     <B, v> = unique patterns of u
33       // patterns according to conditions described earlier in text
34       // each pattern in B has a associated count of occurences in v
35     sort B by v;
36     for all p in B
37        addfeeder(f, p, t);
```

```
38        if f has m double pitch components
39          stop
40  output f
41  end assignfeeder.
```

Routine *compact* transforms the initial tape input $\overline{t}$ into a sequence $\overline{u}$ of double pitch components with spaces. Adding a pattern reference ψ_k to the feeder means positioning the double pitch components from the pattern into the feeder at the distances specified by the e_i- counts of the pattern. The contents of the feeder is valid if the number of double-pitch components in it is not more than m, and if it does not contain more than a_i components of type i.

The output of *assignfeeder* is the assignment of double pitch component types to sequencer slots. Observe, that the algorithm might not assign all double pitch components to slots, because some double pitch components do not participate in frequently occurring patterns. These unassigned components (including also the single pitch components) can be placed into remaining empty slots of the feeder at arbitrary locations. This can be done by reserving sufficient empty feeder slots to accommodate these remaining components, and limiting the number of slots which can be filled to m when adding frequently occurring patterns to the feeder. (In the pseudocode outline given above we omitted this rather straightforward step.)

As mentioned before, there are at most $l \cdot m$ patterns to consider. The number of patterns can still be limited by limiting the minimum and maximum length of component sequences considered when enumerating the patterns. In our experiments we considered component sequences of lengths between 2 and 10. This reduces the number of patterns to $O(l)$. Since the pattern enumeration step looks up each pattern in the pattern set, the actual pattern set construction includes in the worst case $O(l^2)$ steps. For each pattern there are at most s feeder slot locations where the pattern can be inserted into the feeder. At each location it takes s steps to verify that the feeder is a valid assignment. This results in a $O(l^2 s^2)$ time complexity of the feeder assigmment algorithm. Since the feeder size is usually fixed in a specific manufacturing setup, the value of s can be considered constant.

6.3 Extension to the n-pitch case

The above heuristics can be easily modified to support the n-pitch generalization of the feeder assignment problem. In the n-pitch generalization each component type can have a different delay time. In this case the narrow-pitch components are designated to be those with 0 delay time.

The feeder assignment heuristics works in a similar way for the n-pitch case as for the 2-pitch case, except that in the n-pitch case the patterns contain component types with non-zero delay time. Since the delay times of components are not considered in the *assignfeeder* algorithm, that remains unmodified in the case of n-pitch components. The actual delay times are considered only in the tape construction step (of section 6).

Further details of the n-pitch problem, and designing a better heuristic algorithm for this extension are not considered here.

6.4 Special cases

In general, the feeder assignment problem is at least as hard to solve optimally as it is to find a solution for the turnpike problem. It is thus obvious that the *assignfeeder* does not provide an optimal solution for all problem instances. If there are repetitive patterns in the tape, the feeder assignment heuristics will recognize and exploit them. Whether this leads to an optimal solution depends on how regular these patterns are.

One can look for regularities in the tape to determine if an optimal solution can be found in polynomial time. For example, if at any given moment, only 1 component fits under the feeder, then it is trivial to insert optimally. But if there are at least 2 (2-pitch) components under the feeder at arbitrary distance, then we have already reached the turnpike complexity. Furthermore, if there is a sufficiently small set of different distances between 2-pitch component pairs on the tape, one can make an exhaustive search to build an optimal solution.

7 Heuristics for component tape construction

Having the feeder slots assigned with various component types, the sequencer starts to insert the components from the feeder onto the tape. As discussed earlier, the objective is to minimize the number of double pitch placement occasions. This step can be achieved by the use of a greedy heuristics for the Minimum Set Cover problem. The algorithm consists of two steps. First a *Matrix Reduction Algorithm* ([14]) is applied to extract a number of subsets which are part of an optimal solution. Then, in the second step (if the set L is not covered by the subsets extracted in the first step), the subsets are considered in decresing order of their size, until the L is covered. More details on this algorithm can be found in [14], [15] and [16].

8 Experiments

We summarize the results of the experiments with the feeder assignment and tape construction algorithms in this section. The algorithms were tested with 3 different types of input tapes. The tape size was fixed to 1000 locations in these tests [6] and the feeder size was set to 120. We also studied the effect of the number of slots for double pitch components (m) on the total delay.

8.1 Delays

Case 1. The tape content was generated by repeating a single pattern of component types (various different patterns of length between 30 and 60 were used). The resulting costs (delay of tape construction) were averaged over all tapes (using different tape instances for each m-value). The size of the extra feeder space for the double pitch components was varied between 0 and 80. The number of component types in this test was $n = 30$ of which $p = 18$ were double pitch, see figure 7 for a summary of results.

It is observed that, as the value of m increases, the tape delay rapidly decreases to a low stationary level. The algorithm effectively recognizes the repeating patterns in the tapes and is able to create a feeder assignment which utilizes these patterns. For the naive algorithm the increase in the number of extra available slots also produced similar improvement on the delay, but the overall quality of the result is worse than for *assignfeeder*.

Case 2. Several random patterns were randomly repeated to generate tape contents. This results in semi-random tapes where the random patterns are mixed in the tape. Similar attributes were used as in the first test (feeder size 120, $m \in [0, 80]$, 18 double pitch components).

Figure 8 shows a similar reduction in delay as in case 1, except that it is less accentuated due to the increased randomness of the tape.

Case 3. The tapes were generated as pseudo-random sequences of component types. No pattern repetition was included in these tapes. Other data in the generation of test cases were the same as for case 1.

Figure 9 reveals that, in this case, increasing the value of m does not provide such a good decrease in the tape construction delay. This can be attributed to the fact that the random content of the tapes does not support the existence of any repeating patterns. It is also observed that when increasing the feeder size, due to the randomness of the tape, any arbitrary random solution (apportioned according to component counts) is as good as counting patterns, since in this case most pattern counts are only 1.

[6] This is a relatively large number, but it helps to observe differences between algorithms and to see whether the running times become unpractically large.

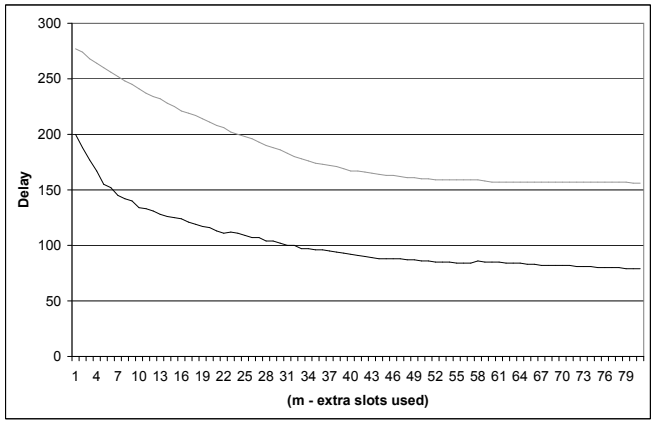

Figure 7: Test case 1 (repetition of a random pattern) using the feeder assignment heuristics. The delay of the sequencer operations is shown as a function of the size of extra slots (m) for duplicating the double pitch components. Delays are expressed as the number of extra operation cycles of the sequencer. The feeder size is $s = 120$. Upper curve: results of *simpleassign*. Lower curve: results of *assignfeeder*. Results are averages of 100 test cases (for each m-value).

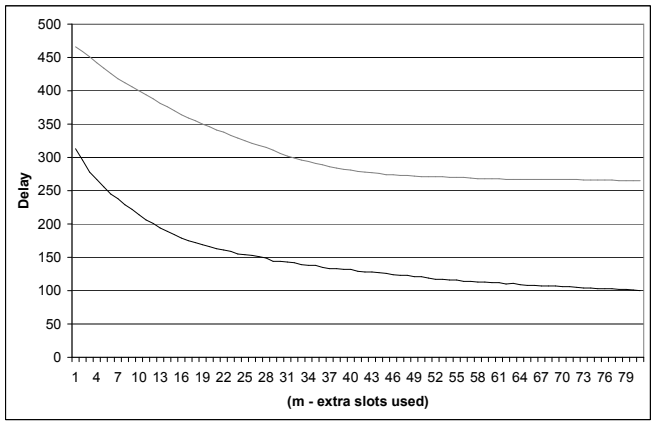

Figure 8: Test case 2. Delay of the sequencer as a function of m for a set of semi-random tapes with feeder size 120. Upper curve: results of *simpleassign*. Lower curve: results of *assignfeeder*. The results are averages for 100 repetitions with semirandom component tapes.

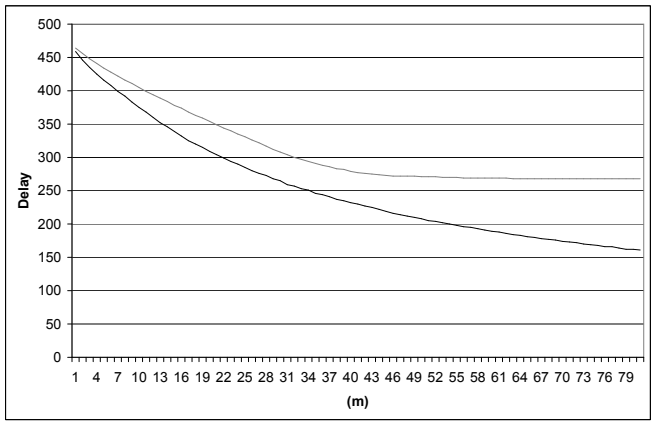

Figure 9: Test case 3. Delay of the sequencer as a function of m for a set of random tapes with feeder size 120. Upper curve: results of *simpleassign*. Lower curve: results of *assignfeeder*. The results are averages for 100 repetitions with semirandom component tapes.

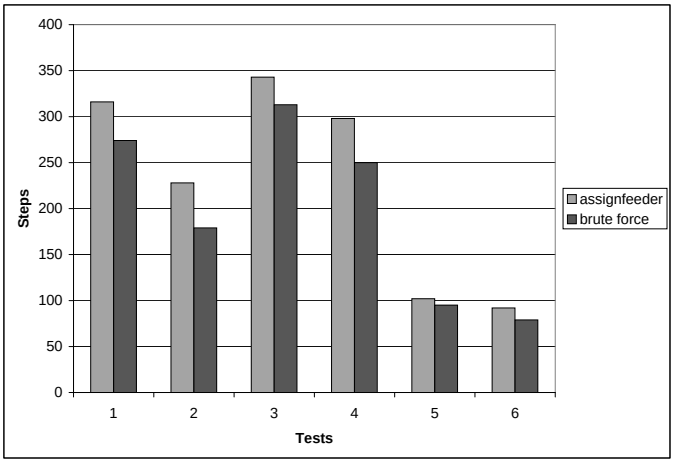

Figure 10: Results of feeder assignment heuristics *assignfeeder*) compared to the results of brute force method (*BFFA* with *BFTC*) for similar test cases as previously discussed, but shorter tape lengths (100 components) and feeder unit (10 slots).

8.2 Comparison with brute force methods

It is evident that there are cases where our heuristics will not produce optimal results. We therefore implemented brute force algorithms (*BFFA* with *BFTC*) for the joint problem of feeder assignment and tape construction, and compared the results of the heuristics to the brute force solutions. Due to the very high time complexity of the brute force methods, the practical problem instances presented in section 7.1. were too large to execute, and we had to content ourselves to small problem instances. These problems are similar to the ones described earlier, except that the tape lengths were set to 100 and the feeder size to 10. The problem instances were generated in a similar way as the ones described earlier (case 1, 2 and 3).

Figure 10 shows that while the feeder assignment heuristics *assignfeeder*) does not always provide the optimal solution, its results were close to optimal and the algorithm runs in a practical time. For the instances tested here, the running time of our heuristic algorithm was less than a second for all data instances, while the brute force algorithm took about 30 minutes to execute for the same (small) instances. The feeder assignments produced by the heuristics caused on average 15% more delay than those produced by the

brute force method.

We also developed an ILOG model based on the integer formulation presented in Section 2, and tested the model using version 10.0.0 of CPLEX. In our evaluation of the model using CPLEX, we were able to get results in useful time for trivial problems with the following properties:

There are less than or equal to m 2-pitch components in the tape.

The distance between the 1st and last 2-pitch components is less than or equal to the size of the feeder.

These are problem instances where all the 2-pitch components can be inserted in one step, as they are clustered at a short (feeder size length) region of the tape. In these simple cases both our algorithm and CPLEX finds the optimal solution.

Another simple case, where the optimal solution is known, is a tape with $l = 21$ components [123456789101234567891 01] and a feeder with $s = 40$ slots, where $m = 20$ are used for 2-pitch components. All components in the tape are 2-pitch. In this case the optimal solution is a feeder containing [13579135792468102468 10], which inserts components in 3 steps optimally. For this type of problem instance, our heuristics did found a similar feeder configuration ($[1, 3, 5, 7, 9, 1, 3, 5, 7, 9, 0, 0, 0, 0, 0, 0, 0, 2, 4, 6, 8, 10, 2, 4, 6, 8, 10]$), where the two significant patterns are randomly positioned in the feeder, and the resulting number of steps was also 3.

However, in practical instances the 2-pitch components are spread out on the tape at arbitrary locations, and the length of the tape is several times the feeder size. When we used CPLEX to solve our test instances presented earlier (with tape length 1000), we did not get any answer after a 30 minute run. We also tried with smaller problem instances with CPLEX, by using the first 100 locations of the tapes, without success.

9 Conclusions

This study considered the *FACTC* problem of assigning electronic components to feeder slots and constructing the input tape of a radial placement machine. The task of the sequencer of the placement machine is to construct the input tape for the actual operation of component placements. In this machine, both the assignments of components to feeder slots and the way of picking up components from the feeder slots, determine the efficiency of the feeder operations. We introduced a sub-optimal but fast algorithm *assignfeeder* to solve the feeder assignment problem and compared it against

(optimal) brute force component assignment algorithm on small problem instances. On large problem instances the brute force algorithm was all too slow to be practical.

We showed that the component-to-feeder assignment has a strong relationship with the turnpike problem, which has been extensively researched before and so far its complexity is not clarified. While our heuristics provided good practical results, for complex problem instances the results were sub-optimal.

In this paper it was assumed that the order of component placements is given as an input, and the goal is to optimize the feeder content and tape construction procedure. Another problem to be solved with these machines is the ordering of components on the tape. These two problems can be seen as sub-problems of the *component tape ordering* problem which should consider details like the precedence constraints of component placements determined by PCB topology, orientation of the components on the board and the machine speed factors.

It turned out that the order of tape reels in sequencer has a significant effect on the processing time of the machine. The utilization of group picks is here a decisive factor. A similar problem, i.e. maximization of the number of group picks, occurs also in the control of multi-head placement machines. These one can take advantage of the possibility of picking up several components at the same head-to-feeder location. An other similar situation occurs when changing nozzles between head and nozzle magazine. The considerations of the present paper might be of use when solving these hard problems.

References

[1] Y. Crama and J. van den Klundert. The approximability of tool management problems, Technical Report, Maastricht Economic Research School on Technology and Organizations (1996).

[2] Y. Crama and J. van de Klundert. The worst-case performance of approximation algorithms for tool management problems. Naval Research Logistics, 46, pp. 445-462 (1999).

[3] Y. Crama, J. van de Klundert and F.C.R. Spieksma. Production planning problems in printed circuit board assembly. *Discrete Applied Mathematics*, 123:339-361, 2002.

[4] Dan Gusfield. Algorithms on Strings, Trees, and Sequences. Cambridge University Press, 1997. ISBN 0521585198.

[5] M. Ayob and G. Kendall. Real-time scheduling for multi head placement machine. In *5th IEEE Int. Symposium on Assembly and Task Planning* (ISATP'03), pages 18-133, Besancom, France, July 2003.

[6] E.K. Burke, P.I. Cowling, and R. Keuthen. The printed circuit board assembly problem: Heuristic approaches for multi-head placement machinery. In *Proceedings of the Conference on Artificial Intelligence (ICAI2000), Volume III*, pages 1456-62, Las Vegas, NV, June 2001, CSREA Press.

[7] Mika Johnsson. Operational and Tactical Level Optimization in Printed Circuit Board Assembly. TUCS Dissertations No 16. Turku Centre for Computer Science (1999).

[8] Timo Knuutila, Sami Pyöttiälä, Olli S. Nevalainen. Minimizing the number of pickups on a multi-head placement machine. TUCS Technical Report 649. Turku Centre for Computer Science (2004). Journal of the Operational Research Society (2007) 58, 115-121.

[9] Timo Knuutila, Sami Pyöttiälä, Olli S. Nevalainen. Improving the Pickups of Components on a Gantry-Type Placement Machine. TUCS Technical Report 692. Turku Centre for Computer Science (2005). 5th International Conference on Technology and Automation, ITCA 2005, 227-231.

[10] Joshua Redstone, Walter L. Ruzzo. Algorithms for a Simple Point Placement Problem. Algorithms and Complexity, 4th Italian Conference, CIAC 2000, Rome, Italy, March 2000, pp. 32-43.

[11] William R. Newell, Richard Mott, S. Beck, and Hans Lehrach. Construction of genetic maps using distance geometry. Genomics, 30:5970, 1995.

[12] Paul Lemke, Steven S. Skiena, and Warren D. Smith. Reconstructing sets from interpoint distances. Tech. Rept. DIMACS-2002-37, 2002.

[13] Z. Zhang. An exponential example for partial digest mapping algorithm, J. Computational Biology 1,3 (1994) 235-239.

[14] V. Chvtal. A greedy heuristic for the set-covering problem, Math. of OR 4:3 (1979), 233-235.

[15] U. Feige. A threshold of ln n for approximating set cover, Proc. 28th Annual Symp. on Theory of Computing (1996), 314-318.

[16] D. S. Johnson. Approximation Algorithms for Combinatorial Problems, J. Computer System. Sci. 9 (1974) 256-278.

[17] T. Leipälä et al. Job grouping in surface mounted component printing. Robotics and Computer-Integrated Manufacturing (1999) v.15 pp. 39-49.

[18] T. Leipälä et al. Optimization of the movements of a component installation automaton, European Journal of Operational Research (1989), Vol. 38, pp. 167-177.

[19] Masri Ayob and Graham Kendall. A Survey of Surface Maount Device Placement Machine Optimisation: Machine Classification, European Journal of Operational Research (2005), Vol. 164, pp.609-626.

[20] L. K. Moyer and S. M. Gupta. Development of the surface mount assembly process through an angular board orientation, Int. Journal of Production Research (1998), Vol. 36, No 7, 1857-1881.

[21] Thomas M. Tirpak et.al. A Generic Classification and Object-Oriented Simulation Toolkit for SMT Assembly Equipment. IEEE Transactions on Systems (2002), Manufacturing and Cybernetics, Part A, Vol. 32, No 1.

[22] Masri Ayob et al. Optimisation of Surface Mount Placement Machines. Proceedings of the IEEE ICIT'02 Bangkok, 11-14 Dec. 2002 498-503.

[23] T. Knuutila et al. Three Perspectives of Solving the Job Grouping Problem, Int. Journal of Production Research, Vol. 39, Issue 18, 2001, pp 4261-4280.

[24] J. Ahmadi and R. Ahmadi. Component Fixture Positioning/Sequencing for Printed Circuit Board Assembly width Concurrent Operations. Operations Research, Vol. 43, No 3, (May-Jun, 1995).

[25] N.D. Choudhury et al. Process planning for circuit card assembly on a series of dual head placement machines. European Journal of Operational Research 192 (2007) 626-639.

[26] P. Csaszar et al. Optimization of a high-speed placement machine using tabu search algorithms. Annals of Operations Research 96 (2000) 125-147.

[27] M. Grunow et al. Component allocation for printed circuit board assembly using modular placement machines. Int. Journal of Production Research (2003), Vol. 41, No. 6, 1311-1331.

[28] Y. Crama et al. The Component Retrieval Problem in Printed Circuit Board Assembly. The International Journal of Flexible Manufacturing Systems, 8 (1996): 287-312.

[29] William Ho and Ping Ji. Component scheduling for chip shooter machines: a hybrid genetic algorithm approach. Computers and Operations Research, Vol. 30, Issue 14, 2003, pp 2175-2189.

[30] P.J.M van Laarhoven and W.H.M. Zijm. Production Preparation and Numerical Control in PCB Assembly, The International Journal of Flexible Manufacturing Systems, 5 (1993): 187-207.

[31] Xuan Du and Zongbin Li. New model and Hybrid Genetic Algorithm for Component Placement of Multihead Gantry Mount Machine. Int. Conference on Industrial Engineering and Engineering Management, 2008, pp. 790-794.

[32] C. Ráduly-Baka, T. Knuutila, M. Johnsson, O.S. Nevalainen. Selecting the nozzle assortment for a Gantry-type placement machine. OR Spectrum, 2008, Vol 30, pp 493-513.

[33] Dong-Seok Sun and Tae-Eog Lee. A Branch-and-price algorithm for placement routing for a multi-head beam-type component placement tool. OR Spectrum, 2008, Vol 30, pp 515-534.

[34] Tae-Hyoung Park and Nam Kim. A Dynamic Programming Approach to PCB Assembly Optimization for Surface Mounters. International Journal of Control, Automation and Systems, 2007, Vol. 5, No. 2, pp.192-199.

[35] Hsin-Pin Fu and Chao-Ton Su. A comparison of search techniques for minimizing assembly time in printed wiring assembly. Int. Journal of Production Economics 63 (2000), 83-90.

[36] M. Ayob and G. Kendall. A triple objective function with a Chebychev dynamic pick-and-place point specification approach to optimise the surface mount placement machine. European Journal if Operational Research (2005), Vol. 164, Issue 3, pp. 609-626.

[37] W. E. Wilhelm et al. A model to optimize placement operations on dual-head placement machines. Discrete Optimization (2007), Vol. 4, Issue 2, pp. 232-256.

[38] M.O. Ball and M.J. Magazine. Sequencing of insertions in printed circuit board assembly. Operations Research (1998), Vol 36, No. 2.

[39] R. Kumar and H. Li. Integer programming approach to printed circuit board assembly time optimization. IEEE Transactions on Components, Packaging, and Manufacturing Technology, 1995, Part B, Vol 18, Issue 4, pp 720-727.

Publication V

Raduly-Baka, Cs., Knuutila T., Johnsson, M., Nevalainen, O. Organizing the operation of radial machine sequencers for multiple PCB-types. To appear in *Computers & Industrial Engineering*.

Organizing the operation of radial machine sequencers for multiple PCB-types.

Csaba Ráduly-Baka [*] Timo Knuutila[†‡] Mika Johnsson[§]
Olli S. Nevalainen[†]

May 7, 2010

Abstract

With a great variation of products, and small product lot sizes, PCB assembling machines must be reconfigured frequently, and their configuration must account for multiple product types. The tradeoff between reconfiguring between product types, or using a single (albeit locally less efficient) configuration for all product types, depends on product lot sizes, and of course, on the cost of machine reconfiguration. In this paper we consider PCB assembly machines of the *radial type*, which are used in manufacturing robust electronics devices. In this machine type, the components are brought to the assembly point by the means of a single component tape. The component tape is constructed on-line by a separate feeder unit (which is the sequencer), composed of a set of slots storing component reels of various types. Insertion of certain component types (slow components) causes a delay in the movement of the component tape. We study the problem of assigning component reels to the sequencer in such a way that the tape construction delay is minimized for multiple PCB types. We assume that all the necessary components fit in the sequencer and therefore, machine reconfiguration between PCB types can be avoided. We also give an integer programming formulation for the problem, and present a heuristic optimization algorithm to reduce the component insertion time caused by slow components.

[*]Nokia Oy, Salo, Finland
[†]Department of Information Technology and TUCS, University of Turku, FIN-20014 Turku, Finland
[‡]Corresponding author: knuutila@it.utu.fi
[§]Valor Computerized Systems (Finland) Oy, Ruukinkatu 2, 20450 Turku, Finland

Keywords: printed circuit boards, production planning, radial machines, component assembly, flexible manufacturing, heuristics, combinatorial optimization.

1 Introduction

In high-mix low-volume electronics manifacturing environments, flexibility has a significant impact on the efficientcy of the PCB production. In addition to the actual manufacturing of a single product, setting up the manufacturing environment for multiple product types has increasingly become a key aspect of the optimization process ([1, 2, 3, 5, 6]). The flexibility of the machine configuration, and the efficiency to manufacture multiple PCB types with a given machine configuration (*minimizing costs of the machine configuration*) has a notable impact on the joint manufacturing cost of the products ([7, 8, 9, 16, 18, 19, 20]).

In the present paper we study the configuration and operation control of a specific electronics manufacturing machine (called *radial placement machine*), in a problem setting where multiple different PCB types are manufactured. In radial placement machines, a single input tape is used to bring the components from a component feeder to the PCB assembly point, where a robotic head places them one by one on the PCB ([21, 22, 23, 24]).

In this machine model the topology of the PCB and other constraints specific to the PCB type, component sizes, and the robotic placement head, put a number of restrictions to the order in which components of various types are arranged on the *component tape* of the placement machine. Due to this property we consider the case where the order of components in the tape is predefined. The components are placed onto the *sequencer* (also called *feeder unit*), which comprises a linear array of feeder slots.

In some common radial machine types (RAD5 and RAD8 of Universal Instruments, for example), due to the physical dimensions of the component reels, the space between two adjacent feeder slots is twice the space between two adjacent tape locations (see figure 1). This means that only every second tape location can be inserted with a component at any given feeder-to-tape alignment. The machine operates in cycles, where each cycle consists of a pick-up and placement operation.

In these machine types, the insertion of certain component types causes an extra delay on the tape movement. This depends on the type (or pitch) of the component. So-called *narrow-pitch* components cause no delay in the tape movement, and they can be inserted in a single tape movement step. Other components (of *double-pitch*) require a fixed delay in the tape

movement, so that the component is properly inserted into the tape. This problem can be generalized into an n-pitch insertion problem, where each component type can be associated with a tape delay of n units. The total of delays of the tape movements caused by these component insertions can be reduced, by inserting multiple components onto the tape at the same time.

There are numerous aspects of these kind of machines which can be subject of optimization. In the present paper we focus on the optimization of the construction of component tapes for multiple PCB types in the presence of double-pitch components. The objective is to minimize the extra time caused by these components.

1.1 Previous research

The delay of tape movements was minimized for a single PCB type in [17]. We extend this problem setting for multiple PCB types and assume that all the PCB types can be manufactured by a single machine. This, in fact, is not a big restriction as we can consider a subset of the PCB types that fulfills the capacity constraint. The optimization task includes two subtasks. First, an assignment of the component reels to the sequencer is determined, and then the component insertions are done in an optimized way.

Each PCB type is defined by a set of components of specific types and a component ordering on the tape. Since the number of 2-pitch components and their location on the tape are given, the tape movement delay is reduced by creating a feeder assignment that places multiple 2-pitch components in a single step, thus reducing the number of such (delay causing) steps.

The multiple PCB-type problem becomes relevant in manufacturing environments, where relatively small lot sizes (a few hundreds) of PCBs are manufactured of each type. Reconfiguring the feeder (i.e. assigning new component types to feeder slots) between these PCB types can block the machine operation for a considerable amount of time (several hours). Therefore, creating a single feeder assignment, that is suitable for sequential manufacturing of multiple PCB types, and accounts for the component ordering characteristics of each of these PCB types, can reduce the total operating time of a machine. In this case, the manufacturing process doesn't need to be blocked between two PCB types, while the total tape delay (of all PCB types) is also reduced.

We extend the heuristic solution presented in [17] to multiple PCB types, and evaluate our method against a simple feeder assigment. We show that the multiple PCB-type feeder assignment problem is computationally as hard as the turnpike problem ([10, 11, 12]). The tape construction problem from a given feeder assignment is the same for multiple PCB-types as for a single

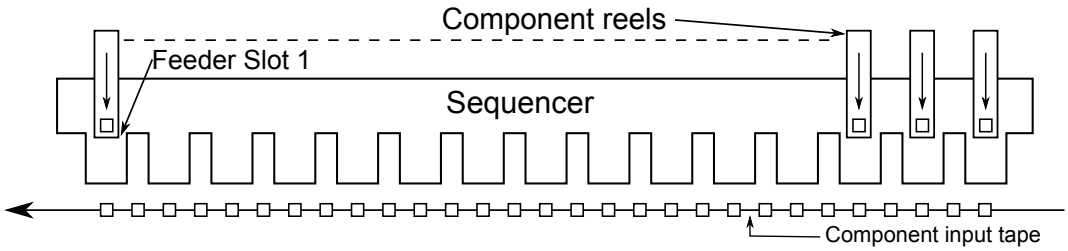

Figure 1: Component input tape moving leftwards under the sequencer. Component reels are connected to the feeder slots.

PCB type. We show that the tape construction step basically is a *Minimum Set Cover* problem ([13, 14]), and we use a minimum set cover heuristic to solve this step.

2 The feeder assignment and component tape construction problem for multiple PCB types

In this section we formulate the *Multiple PCB Feeder Assignment and Component Tape Construction* (MFACTC) problem. Figure 1 illustrates how the component tape moves under the sequencer unit, and how the feeder slots are aligned with tape locations at a given moment. The tape moves stepwise leftwards, and at each step the sequencer can insert into one or several tape locations that are under each slot.

To sum up, the following assumptions are made:

1. All PCB types are manufactured with a single freeder setup.
2. The lot sizes for different PCB types are known.
3. The order of the components on the tape of the placement machine are predetermined for each PCB type.
4. A sequencer (feeder unit) is used to insert the components onto the tape, in the specified order.
5. One feeder slot of the sequencer contains components of a single type, only. Some of the feeder slots may be unused (i.e. empty).
6. Component types can be of single or double-pitch.

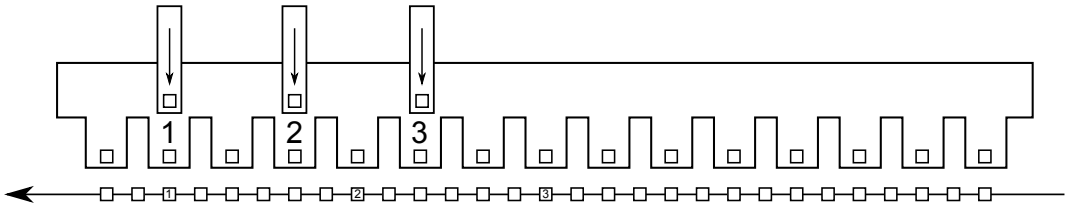

Figure 2: Unfavourable case. When feeder slots of double-pitch components are not properly aligned, each double-pitch component insertion causes delay. The numbers on the tape and the sequencer indicate the component types.

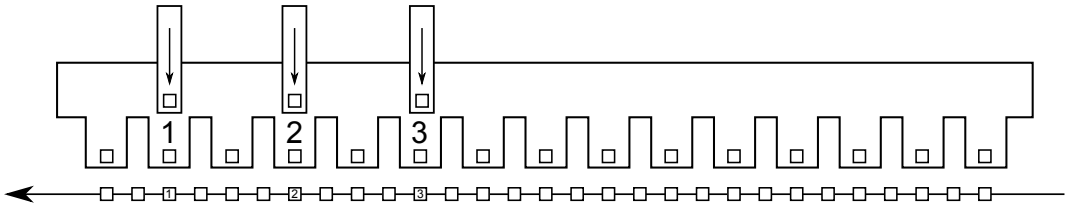

Figure 3: Favourable case. When the feeder slots and the tape positions are properly aligned, all three double-pitch components can be inserted in one step causing a single delay (c) of the tape movement.

7. The insertion of a double-pitch component onto the tape blocks the tape movement and thus causes a delay of a constant time (c) in the operation of the component assembly robot. Thus, it is assumed that the component tape movements form a decisive time factor of the assembly process.

8. The startup and shutdown time of the component assembly and tape construction process are considered fixed.

9. The distance between two neighboring feeder slots is twice the distance between two neighboring tape locations, so at any given moment, the sequencer can insert a component onto every second tape location only.

10. One may duplicate some component reels in the sequencer.

The tape movement delay, caused by double-pitch component insertion stops, can be reduced by creating a feeder assignment where such stops are minimized. For example, figure 2 illustrates a feeder assignment where the tape movement is delayed by $3c$ since there is a stop for each of the three

marked components on the tape. This delay can be reduced if all these components were inserted in one step. This can be done if the needed component reels are located in such feeder slots, that they align with the tape locations where they are inserted (see figure 3).

Since there are multiple PCB types to be manufactured with a single feeder assignment, and there can be different lot sizes of each PCB type, the feeder assignment may be disadvantageous for a given PCB type of small lot size, in order to improve the performace for some larger lot sizes.

2.1 Notations

In the *MFACTC* problem we use the following notations:

s the number of the feeder slots;

n the number of different component types used in all PCB types;

$N = \{1, ..., n\}$, the set of component types used in all PCB types;

$\overline{y} = \{y_1, y_2, ..., y_n\}$, the pitches of the component types, where y_i is either 0 indicating a narrow component, or 1 indicating a double-pitch component;

x the number of different PCB types;

$\overline{w} = \{w_1, w_2, ..., w_x\}$, the lot sizes the PCB types;

l_k, the total number of components to be inserted in the tape for PCB type k;

$\overline{t}_k = t_{k1}, t_{k1}, ..., t_{kl_k}$ the component sequence on the tape from left to right for PCB type k, where $t_{kj} \in N$;

$\overline{f} = \{f_1, f_2, ..., f_s\}$, a feeder assignment, where $f_i \in N \cup \{0\}$ specifies the component type in slot i. If $f_i = 0$, no component reel is assigned to that slot.

p, the number of double-pitch component types, $p \leq n$. The value of p can be calculated by summing up values of $\overline{y}$;

m, the number of slots in the feeder used for double-pitch components, $p \leq m \leq (s - n + p)$. The value of m is an input parameter to our problem;

The goal in *MFACTC* is to find a feeder assignment $\overline{f}^*$ that is valid (i.e. all PCBs can be manufactured) and optimal (tape movement delay due to insertions of the double-pitch component is minimized).

Since all PCB types are manufactured with a common feeder setting, the sequencer must accommodate at least one instance of each component type. If the union of the component types required by the various PCB types is larger than the feeder size, the PCB manufacturing must be partitioned into multiple groups, so that each group (containing a subset of PCBs) can be processed with the sequencer. The feeders of the sequencer must then be reconfigured while advancing from a group to another. Our focus here is to consider one group of PCBs, only. If neccessary, construction of groups can be done by well-known job grouping methods ([7]). When the size of the feeder unit is larger than the number of required component types, multiple slots can be used for a single component type and thus potentially improving the tape insertion performance.

Since insertion of single-pitch components does not cause any tape movement delays, it is not a factor in our optimization goal determining slots for these components. As a side note, it may still be a good idea to store two copies of some (heavily used) 1-pitch components. The space for 1-pitch components is controlled indirectly by the input parameter m, which limits the number of 2-pitch components in the feeder.The remaining slots $n - m$ can be used by single-pitch components.

2.2 Integer programming formulation

In an earlier study of the feeder assignment problem *FACTC*, an integer programming formulation was given for the single PCB type case. There the lot size of PCBs has no effect on the optimality of the feeder assignment. In the multiple PCB type problem of this study, lot sizes have a significant impact on the quality of a feeder assignment, since the component positions in the feeder must be balanced according to the number of PCBs of each type.

One simple way to extend the integer programming formulation of [17], is to create a *virtual* tape where each $\overline{t}_k$ is repeated w_k times and then using the single PCB model to solve this problem. The issue with this approach is that it creates extremely large problem instances because there can be hundreds of PCBs in a single lot of a PCB type.

An other way to consider the different lot sizes is to include them in the constraints of the integer program formulation. In order to do this, we create a virtual tape containing exactly one copy of the component sequence of each PCB type. We also include in the tape a space at least as large as the feeder,

(containing an arbitrary sequence of single-pitch components that have no impact on the tape delay) between two consecutive component sequences of different PCB types. This extra sequence allows that there is a long enough gap between components of two different PCB types so that the sequencer does not have to consider mixed insertions of two types.

Assumption 9 (the sequencer can insert a component onto every second tape location only), can be modeled by using a virtual feeder of size $s' = 2 \cdot s - 1$ and forbidding component assignment to every second feeder slot. Let's define the length of the virtual tape used in these calculations as $l' = \sum_{k=1}^{x}(l_k + s')$.

We also introduce weights w'_t associated with feeder-to-tape alignments, where the weights are equal to the lot sizes of the PCB types ($t = 1, 2, ..., l'$), where l' is the virtual tape length. w'_t takes into account how often the feeder-to-tape alignment occurs when considering the manufacturing of a PCB lot. Note here that alignment t is connected to a particular PCB type as described by the blocks of the virtual tape.

We may assume without loss of generality that types $1, ..., p$ are the double-pitch components.

Let

$$g_{ij} = \begin{cases} 1 & \text{if } i \text{ is odd and slot } i \text{ contains double-pitch component of type } j \\ 0 & \text{otherwise.} \end{cases}$$

The use of feeder slots is limited to odd locations, so that proper spacing is maintained in the real feeder unit.

A legal feeder assignment is defined by the following three constraints as adapted from [17]:

For each double-pitch component type j, there must be at least one slot assigned with that component type:

$$(i) \qquad \sum_{i=1}^{s'} g_{ij} \geq 1, \text{ for all } j \in \{1..p\}$$

The total number of slots used for double-pitch components is m:

$$(ii) \qquad \sum_{i=1}^{s'} \sum_{j=1}^{p} g_{ij} = m.$$

A slot can contain at most one double-pitch component:

$$(iii) \qquad \sum_{j=1}^{p} g_{ij} \leq 1, \text{ for all } i \in \{1..s'\}.$$

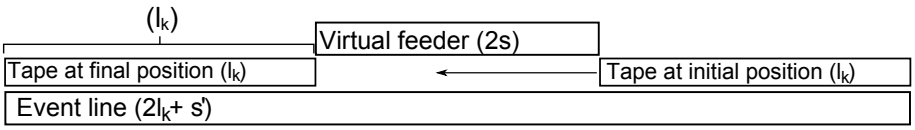

Figure 4: The virtual tape moves under the virtual feeder from an initial to a final position, defining a feeder/tape alignment event line.

2.2.1 The tape construction step

A *legal feeder assignment* for the double-pitch components is defined by constraints (i)-(iii). To calculate an *optimal feeder assignment* one must also account for the tape construction step, where the components are inserted from the feeder slots into the moving component tape. Insertion of a component can occur if the component types of the feeder slot and the component tape position under the slot match. In the following we extend the integer programming model introduced in [17] to the multiple PCB type problem.

There are at most $l'+s'$ feeder-to-tape alignments, that is, there are $l'+s'$ steps from the initial to the final position of the tape (see figure 4). Observe, that the insertion of double-pitch components is feasible only at a subset of these alignments, for example, at the initial and final positions no insertion occurs.

Let $t \in [k..s'+k-1]$ denote a feeder-to-tape alignment on a virtual tape of length l'. The so-called *feeder-to-tape alignment event line* visualizes the orientation of the sequencer in relation to the tape (see figure 4). There are $2l'+s'$ feeder-to-tape alignment positions on the event line, which are indexed from left to right as $1, 2, ..., 2l'+s'$. Let s'_{kt} denote the position of the kth tape element on the event line at moment t. Then we have $s'_{kt} = l'+s'+k-t$.

Let a_k denote the component type at location k on the tape. For alignment t, the insertion of a component to location k of the tape from feeder slot i (numbered from left to right as $1, 2, ..., s'$) is possible if:

(a) the required component type a_k is in feeder slot i:

$$g_{ia_k} = 1. \tag{iv}$$

and

(b) feeder slot i and virtual tape location k are aligned:

$$S_{kt} = l' + i. \tag{v}$$

Equation (v) gives us $l'+s'+k-t = l'+i$, which means that $i = s'+k-t$. For example, when $t = 1$, the first tape location ($k = 1$) is aligned with the

last feeder slot ($i = s'$) and the tape moves leftwards (see figure 4), for $t = s'$, the first tape location ($k = 1$) is aligned with the first feeder slot $i = 1$.

Let q_{kt} denote the decision variable whether the component at tape location k is inserted at alignment t:

$$q_{kt} = \begin{cases} 1 & \text{a 2-pitch component is inserted to tape location } k \text{ at alignment } t, \\ 0 & \text{otherwise.} \end{cases}$$

A component can be inserted at only one alignment to a virtual tape location. Therefore, for all $k \in \{1..l'\}$:

$$(vi) \qquad \sum_{t=1}^{l'+s'} q_{kt} = 1$$

Component insertion can occur only if both *(iv)* and *(v)* hold at moment t:

$$(vii) \qquad q_{kt} \leq g_{ia_k}, \text{ where } i = s' + k - t.$$

Thus, the values of q_{kt} give both the feeder assignment and the component tape construction sequence.

2.2.2 Objective function

The goal in the feeder assignment and tape construction problem is to minimize the tape movement delay caused by double-pitch component insertions. For a given feeder-to-tape alignment t, the expression $(1-q_{kt})$ is 0 if a double-pitch component is inserted to the kth location of the tape, and 1 if insertion of a double-pitch component does not occur.

Let $L = \{i_1, i_2, ..., i_h\}$, where $i_j \in 1, ..., l'$, denote the set of locations of the double-pitch components on the virtual tape. Here h denotes the total number of double-pitch components required for all PCB types (when considing one instance of each PCB type).

Like in [17], for a feeder-to-tape alignment t, the product:

$$(viii) \qquad \prod_{k \in L} (1 - q_{kt}) = 1$$

only when no insertion of a double-pitch component occurred at that alignment.

At any given alignment t, the tape either moves on, or stops so that component insertion can occur. Therefore, minimizing the number of stops means

maximizing the number of alignments where no double-pitch components are inserted.

At this point, we must also consider the lot sizes of different PCB types. Since the virtual tape considers one instance of each PCB type, one has to weight the double-pitch stops by lot sizes. Hence, the real impact of a double-pitch insertion stop at alignment t is:

$$(ix) \qquad w'_t \cdot (1 - \prod_{k \in L}(1 - q_{kt}))$$

which is either w'_t, meaning a delay because of one or more double-pitch component insertion, or 0, indicating no delay at the alignment t. That is, the cost of a stop depends directly on the lot size of a particular PCB type.

The sum of insertion penalties at each alignment is then:

$$(xi) \qquad \sum_{t=1}^{l'+s'} w'_t \cdot (1 - \prod_{k \in L}(1 - q_{kt})).$$

By introducing a new variable z_t, the above objective of the *MFACTC* problem can be linearized to the form:

$$(xii) \qquad min\{\sum_{t=1}^{l'+s'} w'_t \cdot (1 - z_t)\}$$

with the constraints:

$$(xiii) \qquad z_t \leq (1 - q_{kt}) \text{ for all } k \in L, t \in [1..l' + s'].$$

2.3 Time complexity

It was shown in [17] that the *turnpike problem* can be polynomially reduced to the decision version of the single PCB problem (*FACT*). In the turnpike problem ([10, 11, 12]) a multiset D of $\binom{n}{2}$ point distances is given, and the goal is to reconstruct the actual locations of n points on a line, so that the distances between all pairs of the points match exactly the values in the multiset. The turnpike problem does not label the distances with pairs of points, so we do not know which distance resulted from which pair of points.

It is obvious, that any single PCB type problem instance can be solved using an algorithm, that solves the *MFACTC* problem, by creating an *MFACTC* instance with a single PCB type. This means that the *FACTC* problem is polynomial reducible to the *MFACTC* problem and we get the following result.

Note 1. *The turnpike problem is reducible in polynomial time to the decision version of the combined feeder assignment and component tape construction for multiple PCB types problem: TP* $\leq_p$ *MFACTC.*

Although the turnpike problem has been extensively studied in existing literature ([10, 11, 12]), at this time it is not known whether it can be solved in polynomial time. The current solutions for the general case of the turnpike problem are pseudo-polynomial algorithms. This motivates the search for a heuristic algorithm for the feeder assignment and tape construction for multiple PCB types problem.

2.4 Component tape construction

After assigning component types to feeder slots, the next step is the actual component tape construction. In the component tape construction, an originally empty tape moves under the feeder, and we have to determine when to stop the tape movement to allow the fixture of the double-pitch components. The goal in this step is to minimize such stops.

Let P_t denote the set of tape locations where double-pitch components can be inserted at alignment $t \in \{1..l + s\}$. Of course, there are values of t for which P_t is empty, and those can be ignored. In the *Component Tape Construction* step, the goal is to find a minimum number of P_t sets so that their union is L (the union of all tape locations where double-pitch components are inserted).

The above problem is basically the *Minimum Set Cover* problem, where a minimum number of subsets must be selected from a given number of subsets that cover a target set (that is the union of all initial subsets). This problem is known to be NP-hard [14].

Actually, the *Component Tape Construction* problem is a special case of the Minimum Set Cover problem, where the elements of set L are on a line, and the size of subsets P_t is limited by the feeder size. This motivates the use of a greedy heuristic algorithm for component tape construction, which provides good results for the Minimum Set Cover problem [13].

3 Feeder assignment

In the following we propose two feeder assignment methods to assign double-pitch components to feeder slots in the multiple PCB type manufacturing case. We first propose a simple (somewhat trivial) method that considers frequencies of single components and then a more elaborate method that considers frequencies of component sequences.

3.1 A simple method

The simple feeder assignment method assigns m double-pitch components to random feeder slot locations in such a way that all neccessary component types are stored in the sequencer. This naive method works as a benchmark in our evaluations of the feeder assignment algorithms.

The number of double-pitch components is determined by apportioning the total number of required components of each type (for all PCB types) to the m feeder slots. Let b_i denote the number of placements of the double-pitch component of type i when summing up all placements in the lots $\overline{w}$ of all PCB types. Then the number of components of type i placed into the feeder is $a_i = \left\lceil \frac{m*b_i}{h} \right\rceil$, where $h = \sum b_i$.

The *simpleassign* algorithm can be summarized by the following steps:

1. For each double-pitch component type i, calculate b_i. Calculate h, and as a basis of these values calculate a_i $(i = 1..p)$.

2. For $i = 1..n$, assign a_i component reels of double-pitch type i to randomly selected (different) locations of the feeder unit.

3. The remaining $n - m$ feeder slots are used for the narrow component types; their assignment is arbitrary.

3.2 Repeated patterns of component subsequences

The simple assignment algorithm makes no attempt to recognize repeated subsequences of component types. When large and varying number of lots of different PCB types are manufactured, component tapes may contain repeated component type sequences, which, if inserted in one step, can reduce the tape movement delay significantly. Altough the component tape layout of one PCB type does not fit into the feeder (a common assumption in practical scenarios), multiple PCB types may share short double-pitch component subsequences, that fit into the feeder and thus, using these sequences as a basis for the feeder assignment may improve the insertion time.

This idea was used in [17] for the single PCB type problem. It was shown that by assembling a set of such sequences in the component feeder, the tape insertion can be considerably improved in cases where the component ordering exhibits some amount of regularity.

The previous heuristics is extended here to multiple PCB types by using the lot sizes as weights when determining the frequencies of the double-pitch component sequence occurences. In order to keep our representation self

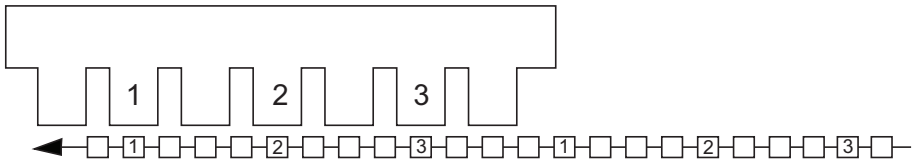

Figure 5: Ideally, a tape contains a repeating sequence of double-pitch components, that can be inserted in one step significantly improves the sequencer performance.

contained, we recall the basics of the *FACTC* heuristics while describing the new *MFACTC* heuristics.

The following heuristic looks from the tapes of different PCB types for component subsequences of length at most s (step 1), and creates a feeder slot assignment that reproduces the most frequently occuring subsequences when the lot sizes are taken into account (step 2).

Step 1. As mentioned before, the number of delayed component insertions depends only on the assignment of double-pitch component reels. The tape can therefore be coded as a sequence of double-pitch components and sequences of feeder slots for 1-pitch components:

$$\overline{u} = (e_0, c_1, e_1, c_2, e_2, ..., c_d, e_d),$$

where $c_i \in [1, n]$ is a double-pitch component type and the integer value $e_i \geq 0$ gives the number of intervening tape locations of narrow-pitch components. The length of this sequence is $2 \cdot d + 1$, where d is the number of double pitch component occurences in the combined tape of all PCB types.

Let $\overline{u}(i, j)$ denote a subsequence $\overline{u}$, between locations $i < j$. We determine a set of frequently occuring subsequences that can be inserted in the feeder in order to allow multiple double-pitch component insertions. Determining the number of occurrences of a sequence in an other sequence (or string) has been extensively discussed in [4], as the *Exact Matching Problem.*

The space occupied by a subsequence $\overline{u}(i, j)$ in the feeder is:

$$S(\overline{u}_x(i, j)) = j - i + 1 + \sum_{k=i}^{j-1} (\frac{e_k}{2}), \tag{1}$$

where $(j - i + 1)$ gives the number of double-pitch components aligned with their feeder slots, and the sum gives the spaces in the slots in-between. The sequence length cannot be larger than the feeder size:

$$S(\overline{u}_x(i, j)) \leq s. \tag{2}$$

Let $C_k(\overline{u}(i,j))$ denote the number of occurences of subsequence $\overline{u}(i,j)$ in PCB type k. Each feasible $\overline{u}(i,j)$ subsequence is assigned with a frequency value $F(\overline{u}(i,j)) = \sum_{k=1}^{x} w_k \cdot C_k(\overline{u}(i,j))$ that is, its number of occurences in the tape multiplied with the lot size of the PCB type. $F(\overline{u}(i,j))$ values are used to determine the order in which the $\overline{u}(i,j)$ sequences are considered for insertion of component reels in the feeder unit.

The worst case time performance of the subsequence set construction algorithm is $O(l'^2 \cdot m^3)$ (assuming a naive algorithm), where $l' = \sum_{k=1}^{x}(l_k + s')$ and m is number of slots reserved for double-pitch components in the feeder. There are at most $l \cdot m$ subsequences to consider, and each subsequence is checked whether it is already in the subsequence set in at most $l' \cdot m^2$ steps. The subsequence construction step can be improved by using suffix trees (see [4]).

Step 2. Let $\Psi = \{\psi_1, \psi_2, ..., \psi_r\}$ denote a set of r subsequences of double-pitch components, created using the method outlined above. Each subsequence ψ_k has an associated weight $v_k = F(\psi_k)$. The feeder assignment algorithm used in [17], does not depend on whether the Psi originated from a single or multiple PCB types, so we can reuse it without modification.

4 Heuristics for component tape construction

Having a feeder assignment of component types, the sequencer can proceed with the insertion of the components onto the tape for the various PCB types. One must then determine at which feeder-to-tape alignments to insert double-pitch components, so that the maximum amount of double-pitch components are inserted at one step.

The above problem can be solved by transforming it to the *Minimum Set Cover* problem, and using any of the well known heuristics ([13], [14] and [15]) to solve that problem. The component tape construction problem is considered separately for each PCB type.

The component tape construction problem for a PCB type can be transformed to the minimun set cover problem, by considering the tape as a set of locations. For each feeder-to-tape alignment of the given PCB type, a subset of location is defined by the tape locations that can be inserted at that alignment. The resulting set of subsets, and the initial set of all tape locations, defines the input for the minimum set cover problem. Using an algorithm for the minium set cover problem, we can select the minimum number of subsets (i.e. feeder-to-tape alignments), that are necessary to construct the tape.

5 Experiments

We evaluated the above subsequence based heuristics for feeder assignment, and compared the results with the simple heuristics. Both algorithms used the same tape construction technique based on a solution for the Minimum Set Cover problem. In the following we summarize the results of these experiments.

The algorithms were tested on 3 different types of input sets. All PCB types used tapes of 400 locations. Each problem set contained multiple number of PCB types (a random number in the [5..16] range). The feeder size was set to 120, and the number of usable slots for double-pitch components (m) varied between 0 and 100. Each test set contained 100 different problem instances (of multiple PCB types). The results are average tape movement delays per PCB type, over all 100 problem instances. These tapes used 30 different component types, of which $15 - 20$ were of double pitch.

The problem sets were also evaluated using the single PCB-type algorithm, where the feeder content was optimized separately for each PCB type. In this case we separated the multiple PCB-type instances into individual single PCB type instances for which a separate feeder was constructed. The results of these tests serve as a kind of lower bound and indicate the loss when solving a joint feeder assignment for all PCB types.

In the first test set, each tape was generated by repeating one randomly selected component sequence (different tapes contained different sequences). This is the easiest problem type, because these sequences are well identified by the feeder assignment heuristics and can be used in the feeder construction. Since a single PCB-type consists only of the repetion of a single component sequence, creating separate feeders for each PCB-type brings significant improvement on the performance. On the other hand the multiple PCB-type case has to accomodate different component sequences in the same feeder, resulting in significant performance deterioration.

In the second test set, tapes were generated by repeating multiple randomly generated component sequences in each tape. The same set of random component sequences was used by all the tapes. This problem instance gives similar results to the first one. Frequently repeated component sequences are recognized by the algorithm, and used in the feeder construction. This test set reproduces realistic scenarios, where the individual PCB-types contain some form of sequence repetition. In this case optimizing separate feeder assignment for each PCB-type did not result in significant improvement over the joint multiple PCB-type feeder assignment.

In the third test set, all tapes were generated by randomly selecting component types (independently for each tape). This is the most inefficient

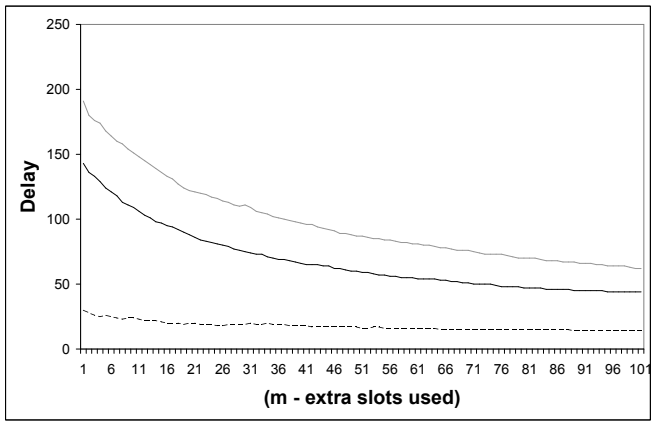

Figure 6: Test case 1 (repetition of a random sequence) using the feeder assignment heuristics. The delay of the sequencer operations is shown as a function of the size of extra slots (m) for duplicating the double-pitch components. Results are averages of 100 test cases (for each m-value in $[0 - 100]$). Bold line: subsequence heuristics; thin line: simple heuristics. Dotted line: separate feeder assignment for each PCB type.

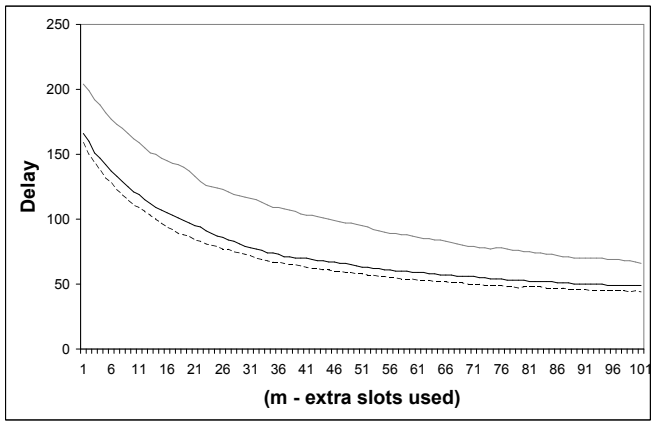

Figure 7: Test case 2 (multiple random sequences shared by tapes) using the feeder assignment heuristics. The delay of the sequencer operations is shown as a function of the size of extra slots (m) for duplicating the double-pitch components. Results are averages of 100 test cases (for each m-value in $[0 - 100]$). Bold line: subsequence heuristics; thin line: simple heuristics. Dotted line: separate feeder assignment for each PCB type.

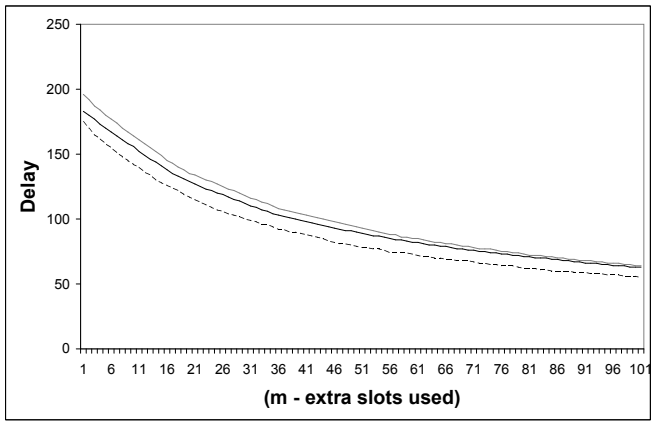

Figure 8: Test case 3 (independent randomly generated tape) using the feeder assignment heuristics. Results are averages of 100 test cases (for each m-value in $[0 - 100]$). Bold line: subsequence heuristics; thin line: simple heuristics. Dotted line: separate feeder assignment for each PCB type.

scenario, where there are no frequently occuring sequences shared between multiple tapes. In this case the two feeder assignment heuristics produced similar results, indicating that for fully random problem instances a random feeder selection is as good as trying to discover and reuse component sequences.

6 Conclusions

The multiple PCB type version of the feeder assignment and tape construction problem was considered in the present study. This problem is an extension of the single PCB-type problem discussed in [17]. The multiple PCB type scenario is a valid practical problem setting, because reconfiguring the feeders of a radial placement machine between different PCB types, can hold up the production for several hours. The multiple PCB-type problem setting allows efficient manufacturing of multiple PCB types with a single feeder assignment.

Just like in the single PCB-type problem, the solution of the multiple PCB-type problem is broken down into two steps. The *Minimum Set Cover*

formulation is used to solve where to stop the tape movement for double-pitch insertions to reduce tape movement delay.

It was assumed here that the order of component placements is given as an input. A further extension of the problem would be to consider, a partial ordering of the components, and determine a final order and feeder assignment with the goal of reducing tape movement delay, caused by the double-pitch insertions. The initial partial order of the components can be determined by the PCB topology and machine limitations. It is obvious that the extra degree of freedom (component ordering) would result in better final performance of the tape construction process.

Our experiments showed that using a single feeder setup for multiple PCBs may reduce the efficiency of production significantly in some case (an obvious conclusion since a feeder assignment considering only one PCB-type can be more efficient). Yet considering the feeder setup cost, optimizing for multiple PCB types, especially in cases where PCB types may share short sequences of similar component types, will improve manufacturing efficiency.

A similar problem, i.e. maximization of the number of group picks, occurs also in the control of multi-head placement machines. These can take advantage of the possibility of picking up several components at the same head-to-feeder location. An other similar situation occurs when changing nozzles between head and nozzle magazine. The considerations of the present paper might be of use when solving these hard problems.

The proposed feeder assignment heuristics, can be easily adapted to support the n-pitch generalization of the feeder assignment problem. In the n-pitch scenario, each component type is associated with a specific delay time. In this case the narrow-pitch components are designated to be those with 0 delay time.

In this case, the component type sequences are collected in the similar way as described earlier. The sequence weights (determining in which order the sequences are processed) are also calculated in a similar way as described before, except that in the n-pitch case, one can multiply the resulting weight of a sequence with the sum of delay values of component types in the sequence. In this way, component sequences causing longer delays (and having high frequency) are considered with higher priority for feeder construction.

References

[1] Y. Crama and J. van den Klundert. The approximability of tool management problems, Technical Report, Maastricht Economic Research School on Technology and Organizations (1996).

[2] Y. Crama and J. van de Klundert. The worst-case performance of approximation algorithms for tool management problems. Naval Research Logistics, 46, pp. 445-462 (1999).

[3] Y. Crama, J. van de Klundert and F.C.R. Spieksma. Production planning problems in printed circuit board assembly. *Discrete Applied Mathematics*, 123:339-361, 2002.

[4] Dan Gusfield. Algorithms on Strings, Trees, and Sequences. Cambridge University Press. ISBN 0521585198.

[5] M. Ayob and G. Kendall. Real-time scheduling for multi head placement machine. In *5th IEEE Int. Symposium on Assembly and Task Planning* (ISATP'03), pages 18-133, Besancom, France, July 2003.

[6] E.K. Burke, P.I. Cowling, and R. Keuthen. The printed circuit board assembly problem: Heuristic approaches for multi-head placement machinery. In *Proceedings of the Conference on Artificial Intelligence (ICAI2000), Volume III*, pages 1456-62, Las Vegas, NV, June 2001, CSREA Press.

[7] Mika Johnsson. Operational and Tactical Level Optimization in Printed Circuit Board Assembly. TUCS Dissertations No 16. Turku Centre for Computer Science (1999).

[8] Timo Knuutila, Sami Pyöttiälä, Olli S. Nevalainen. Minimizing the number of pickups on a multi-head placement machine. Journal of the Operational Research Society (2007) 58, 115-121.

[9] Timo Knuutila, Sami Pyöttiälä, Olli S. Nevalainen. Improving the Pickups of Components on a Gantry-Type Placement Machine. 5th International Conference on Technology and Automation, ITCA 2005, 227-231.

[10] Joshua Redstone, Walter L. Ruzzo. Algorithms for a Simple Point Placement Problem. Algorithms and Complexity, 4th Italian Conference, CIAC 2000, Rome, Italy, March 2000, pp. 32-43.

[11] William R. Newell, Richard Mott, S. Beck, and Hans Lehrach. Construction of genetic maps using distance geometry. Genomics, 30:5970, 1995.

[12] Paul Lemke, Steven S. Skiena, and Warren D. Smith. Reconstructing sets from interpoint distances. Tech. Rept. DIMACS-2002-37, 2002.

[13] V. Chvtal. A greedy heuristic for the set-covering problem, Math. of OR 4:3 (1979), 233-235.

[14] U. Feige. A threshold of ln n for approximating set cover, Proc. 28th Annual Symp. on Theory of Computing (1996), 314-318.

[15] D. S. Johnson. Approximation Algorithms for Combinatorial Problems, J. Computer System. Sci. 9 (1974) 256-278.

[16] C. Ráduly-Baka, T. Knuutila, M. Johnsson, O.S. Nevalainen. Selecting the nozzle assortment for a Gantry-type placement machine. OR Spectrum (2008).

[17] T. Knuutila, M. Johnsson, Olli S. Nevalainen, C. Ráduly-Baka, Construction of component tapes for radial placement machines. Computers & Operations Research (2010), Vol. 37, pp. 1488-1499.

[18] T. Leipälä et al. Job grouping in surface mounted component printing. Robotics and Computer-Integrated Manufacturing (1999) v.15 pp. 39-49.

[19] T. Leipälä et al. Optimization of the movements of a component installation automaton, European Journal of Operational Research (1989), Vol. 38, pp. 167-177.

[20] T. Knuutila et al. Three Perspectives of Solving the Job Grouping Problem, Int. Journal of Production Research (1999).

[21] Masri Ayob and Graham Kendall. A Survey of Surface Maount Device Placement Machine Optimisation: Machine Classification, European Journal of Operational Research (2005), Vol. 164, pp.609-626.

[22] L. K. Moyer and S. M. Gupta. Development of the surface mount assembly process through an angular board orientation, Int. Journal of Production Research (1998), Vol. 36, No 7, 1857-1881.

[23] Thomas M. Tirpak et.al. A Generic Classification and Object-Oriented Simulation Toolkit for SMT Assembly Equipment. IEEE Transactions on Systems (2002), Manufacturing and Cybernetics, Part A, Vol. 32, No 1.

[24] Masri Ayob et al. Optimisation of Surface Mount Placement Machines. Proceedings of the IEEE ICIT'02 Bangkok, 11-14 Dec. 2002 498-503.

Turku Centre for Computer Science
TUCS Dissertations

107. **Dorina Marghescu**, Evaluating Multidimensional Visualization Techniques in Data Mining Tasks
108. **Tero Säntt**i, 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 Communication 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

Turku Centre *for* Computer Science

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www. tucs.fi

University of Turku
Faculty of Mathematics and Natural Sciences
- Department of Information Technology
- Department of Mathematics

Turku School of Economics
- Institute of Information Systems Science

Åbo Akademi University
Division for Natural Sciences and Technology
- Department of Information Technologies

ISBN 978-952-12-2618-2
ISSN 1239-1883

Csaba Ráduly-Baka

Algorithmic Solutions for Combinatorial Problems in Resource Management of Manufacturing Environments