A Performance Test of a Power Index Computer Program

Acta politica aboensia, B 3
Department of Political Science and Contemporary History
FI-20014 UNIVERSITY OF TURKU
ISBN 978-951-29-4922-9
ISSN 1798-3134
ISSN-L 1798-3134
Uniprint Turku 2012

Abstract

There is a considerable literature studying voting power and power indices, however a study exploring the practical aspects of voting power computation seems missing. This study examines a power index program termination time and runtime memory usage in large voting bodies up to 190 voters. In a comparison an up-to-date computer surprisingly performs overwhelmingly better compared to a slightly older model. The simulations reveal the greater speed of the up-to-date computer being due to more advanced processor architecture together with a more efficient data bus and memory. The applied all-in-one program is found rather slow due to simultaneous processing of many indices. The runtime memory usage is found modest in the simulations. The literature suggests that the time and storage complexity of the applied algorithm could be reduced.

Introduction

The tools of voting power analysis, the power indices, are used to measure voting power of the members of a voting body. The only variables, which are taken into account, are the resources of the voters (votes) and the vote threshold (quota). Thinking of constitutional design the motivation is to analyse *a priori* what the voters can do with their votes in a voting body instead of just comparing formal vote amounts. Here power is understood as being on a pivotal position in a vote and thus having the ability to control the vote outcome. This information cannot be revealed by comparing the voters' vote amounts. Another possible motivation for the use of power indices is normative. For example, in an existing voting body it is possible to analyse and argue the fairness of some vote distribution. The computation of the power indices is, however, very tedious and can reliably be carried out only with the aid of a computer. In what follows I shall focus on certain aspects of computerized voting power analysis.

The scholarly literature on voting power and power indices is vast.[1] The publications can roughly be divided into two main branches, from which the first apply the power indices and hence analyse voting bodies. For example the European Union enlargements have motivated numerous analyses with respect to the fairness of various Member States' vote distributions in the Council of Ministers (see e.g. Hosli 1993; Widgrén 1994, 1995; Felsenthal and Machover 2000; Leech 2002a). Other popular objects of similar analyses have been the U.S. presidential Electoral College, the European Parliament as well as national parliaments. The second branch of literature is more mathematically oriented and is interested in the properties of power indices (see e.g. Felsenthal and Machover 1998; Pajala 2003; Widgrén and Napel 2008). On the background of these literature branches there are few studies, which discuss the computer aided computation of voting power. These studies have emerged from the need to be able to study voting bodies with more than just few voters. The articles present, develop and analyse various methods and algorithms for power index computation (Brams and Affuso 1976; Leech 2002a; Lambert 1982; Lucas 1978; Mann and Shapley 1960, 1962; Matsui and Matsui 1998, 2000; Uno 2003). Although the studies include theoretical discussion on the complexity of the computation in the

[1] Homo Oeconomicus alone has published several special issues on power indices (see Widgrén 2000; Holler and Owen 2000; Holler and Owen 2002; Gambarelli and Holler 2005; Gambarelli 2007 and the references therein).

algorithm level, present outlines of computer programs and even give some hints in practical terms, to the best of my knowledge no empirical studies exist on the subject.

In all practical computation two aspects are of primary interest: the program termination time and the runtime memory usage. These measures indicate the limits of the computer internal components i.e. the hardware. The two related questions are: First, is the processor capable of processing the computational task? And second, do we have enough memory resources for the process? Accordingly, this technical report will examine and discuss termination times and runtime memory usage of a power index program. The limit of how large voting bodies can be analysed with a computer has risen over the decades. The first known computer programs together with the related articles appeared in the early 1960s. In those days the mainframe computers were able to analyse at least a 50 member body. While the two seminal articles by Mann and Shapley (1960, 1962) report program termination times, they provide no information on the memory usage. Later studies appear to provide even less information on the practical aspects of computation. Nowadays the processors and other computer hardware are developing rapidly and the processing capability of a modern desktop computer is more or less astronomical compared to the mainframes of the 1960s.

This article has its roots in the need to know how large voting body a World Wide Web based public calculator service could analyse within a reasonable time. For online solutions the computation time cannot be more than few seconds. The first respective analyses were carried out in 2003. In addition to the search of the limits of feasible online voting games it was tested whether the computer hardware and the program could cope with very large voting bodies up to 190 voters. This refers roughly to the size of the IMF board of governors, which is the largest public organization having been analysed in the literature. Roughly two or three (Intel) processor generations later an identical test sequence was carried out again in 2009 and 2010, however now using an up-to-date workstation. This time the comparison was done in order to evaluate whether it was worth considering updating the old server hardware dating from 2003.[2] Our main research questions are: First, to what extend do the execution times differ between the

[2] In technical terms the old CPU is a single core Intel Pentium 4 generation CPU running at the clock frequency of 2.4 GHz. The new CPU is an Intel Core 2 Duo generation CPU running at 2.66 GHz. The new CPU has two cores, however only one core at the time is used in the program execution.

old and new workstations? Second, what are the main causes for the possible differences? And third, how substantial are the memory requirements regarding the computation processes?

In a nutshell, the test sequences begin with an input of a 50 voter voting game [638; 1, 2,..., 50], which is fed to the program and the execution time is measured. For each subsequent measurement round 10 more voters are added to the game ending up with 190 voters. Also a second test sequence is carried out, however this time multiplying the voters' votes by 10 and thus giving the processor (CPU) a much harder load. The quota is always kept at the simple majority.[3] Regarding the memory requirements the literature points out that the applied calculation algorithm is modest on processor time consumption, however the trade-off is that the memory requirements can be substantial. Simulations carried out in 2010 will, for the first time, provide an insight to the memory consumption aspect of the computation.

The comparison between the hardware used in 2003 and 2009/2010 appeared to be very interesting. What came as a surprise were that seemingly only small advances in computer technology within just few years substantially affected the program termination times. The new computer was in certain voting games well over ten times faster compared to the old hardware. Moreover, the comparison of the computation times between the computers using a hard input appeared to be counterintuitive at the first sight. Regarding small voting bodies the new hardware was gaining relative speed over the old hardware, however regarding large voting bodies the relative speed surprisingly diminishes. The memory requirements appeared to be rather modest considering the amount of physical memory available in modern computers. It became apparent that the applied program could be enhanced in several ways.

Subsequent to the introduction I shall briefly discuss voting power measurement together with the relevant literature. This is followed by a brief discussion on the applied power index computation algorithms and the key aspects of computer hardware. Subsequently I report the computation time and memory usage analyses and discuss the

3 Memory requirement and computation speed grow in the opposite directions: a higher quota speeds up the computation, however affects the memory usage to increase (and vice versa). Simple majority is thus the hardest possible load regarding program termination time.

effects of the computer hardware. Finally, I review and discuss several possibilities to enhance the applied program.

Voting power and the software side of computation

In a nutshell the power index values are based on the analysis of either voter combinations or voter permutations of a voting body. Why computers are needed is because there are always 2^n voter combinations or $n!$ voter permutations to analyse, n being the number of voters. Regarding each combination or permutation we need the information whether any voter(s) are in a pivotal position so that if a voter i changes (swings) her vote from *yes* to *no* the remaining of the voter combination is no longer winning without her i.e. does not meet the vote threshold. The pivots and swings are the raw material for power index calculation. For example, the voter permutation based Shapley-Shubik index value for voter i is her relative amount of all voters all pivots (Shapley and Shubik 1954). The voter combination based standardized Banzhaf index value, in turn, is the relative amount of all voters all swings for voter i (Banzhaf 1965). Dividing voters swings with the constant of 2^{n-1} constitutes the third of the most applied power indices – the Penrose (a.k.a. Penrose-Banzhaf, or absolute Banzhaf) index (Penrose 1946). For more discussion and details of the power indices the reader is advised to refer to the in depth studies of e.g. Felsenthal and Machover (1998) or Straffin (1994).

Turning to the computerized voting power analysis Penrose (1946) or few years later Shapley and Shubik (1954) carried out their analyses with pen and paper. The two seminal papers regarding computerized power index analysis were published by Mann and Shapley (1960; 1962) in the beginning of the 1960s. Applying the Shapley-Shubik index Mann and Shapley analysed the U.S. Presidential Electoral College, which at the time consisted of the 50 U.S. States. It is remarkable that the basic idea of the computation algorithm introduced by Mann and Shapley in their 1962 paper is the same applied here. In fact, the Mann and Shapley algorithm was virtually forgotten for decades and has been studied in more detail during the last ten years. Later in the 1960s John Banzhaf (1965) introduced the (standardized) Banzhaf index. Most probably he was able to use a computer at least in some of his analyses. Banzhaf did not apply the Mann and Shapley (1962) method of computation for the first such application in conjunction with the Banzhaf index appeared later in Brams and Affuso (1976). To the best of my knowledge Lee Papayanapoulos was consulting Banzhaf, at least in some point. Papayanapoulos has also later studied the computation of the Banzhaf index (see e.g. Papayanapoulos 1981).

There are several known algorithms for power index computation. These can roughly be divided into two categories: First, the so-called approximation algorithms compute index values, which are, as the name suggests, approximations (Leech 2003; Owen 1972, 1975). Second, there are algorithms, which area able to yield exact index values. Both categories include several techniques (see the survey by Leech (2002b)). The challenge to the computer is the complexity of the computation, which brings about heavy demands for computer resources in one way or another. With respect to exact index value computation the algorithms are either very processor intensive, or alternatively, very storage intensive. A workaround to this have been the approximation algorithms.

Setting aside the approximation methods, among the exact computation algorithms the simplest way to analyse a voting body is called direct enumeration. The iterative, though not very effective, algorithm is a rather natural way of thinking of how to compute power index values. It is a very straightforward algorithm which rests on the idea that every voter combination is actually created in the computer memory and then analysed for voter swings or pivots. The algorithm is very processor intensive, however the storage requirements remain very modest. In fact, the random access memory (RAM) is only used to store few rather short vectors, which are updated only when needed. The number of possible voter combinations double with every additional voter, and unfortunately the same applies to the program execution time. The practical computation time limit (assuming no voters have the same amount of votes and applying the simple majority quota) in a modern PC is around 30 voters.[4] Accordingly, a 31 voter voting game [233; 1, 2, ..., 31] takes over an hour to terminate, so a 32 member body would require 2-3 hours to terminate and so on. In the mid-1970s the computational limit was less than 20 voters (Brams and Affuso 1976).

4 Algorithm researchers usually set the limit of reasonable program execution time to one hour (Uno 2003).

The computation algorithm we apply here is based on the mathematical properties of the so-called generating functions. Among the known algorithms it is the most effective. The seminal paper was published by Mann and Shapley in1962. Previously Mann and Shapley (1960) were only able to come up with approximations of voting power regarding the member states of the U.S. Presidential Electoral College. The computation of the exact values became possible after David Cantor proposed the use of generating functions to Mann and Shapley (Mann and Shapley 1962). The algorithm and its mathematical and other properties in conjunction with various power indices have been later studied at least by Brams and Affuso (1976), Lucas (1978), Lambert (1988), Leech (2002), Matsui and Matsui (2000) and Uno (2003) (see also Alonso-Meijide et al. 2005; 2009). In terms of a computer program a generating functions based algorithm, after certain optimization, is rather short in code length. Basically, the core of the program consists of few recursive loops (instead of iterative loops) which are able to reduce the time complexity of the algorithm into polynomial time (instead of exponential). In comparison to the above the voting game [233; 1, 2, ..., 31] will now terminate within few fractions of a second, which is feasible for online use. The challenge and problem in direct enumeration algorithm is that the number of coalitions double every time a new player is added to the game. However, provided that the weights of the voters and the quota are integer numbers there is a workaround using a different approach. The generating functions enable a quick and easy way to find out the number of coalitions of each size. The numbers (of coalitions of each size) are certain coefficients of the generating function. To discover the swings for the Banzhaf and Coleman indices it actually suffices to analyse coalitions having the total weight equal to the quota or less. Once this information is obtained it is possible to find out the swings by knowing the weights of the voters. The same idea with a different generating function can be used to find out the pivots for the Shapley-Shubik index. For the Deegan-Packel index the computation is harder as it is based on minimal winning coalitions thus requiring processing of certain additional information. Basically, the generating functions (recursively) create a sequence of sub-problems, i.e. matrices of the generating function coefficients (or factorials), which include part of the information for the swing and pivot analyses. The program will store the unsolved sub-problems into a stack until the root of the recursion is reached and all the sub-problems can be solved. Recursion is a loop structure which a computer is able to execute extremely fast, however downside is that the coefficient matrices can become very large and require a lot of RAM memory for the sub-problem stack. During the computation process some of the integers can also become very large. As is shown below, there is variation in how fast the algorithm can perform in conjunction with various power indices. More details of the algorithm together with computer program outlines are discussed e.g. in Brams and Affuso (1976), Matsui and Matsui (2000), Lambert (1988) and Leech (2002).

The hardware side of the computation

In order to compare and explain possible differences in the performance of the two workstations the most important components the computers need to be briefly discussed. Basically, a computer consists of three main parts: First, the most important part of a PC is its processor, which is able to do arithmetic and logical operations. Second, for any data to be processed the computer needs (various kinds of) user programmable memory. Third, the computer needs a certain infrastructure, the motherboard, to enable the processor and memory to work together. Table 1 shows the key components of the old and new hardware and also the most important memory types and speeds together with the speed of the data bus.

Table 1. Key hardware components in two PC workstations

Motherboard (Asus)	CPU (Intel)	CPU clock frequency	CPU L1 cache	CPU L2 cache	Data bus speed	RAM speed
P4S533-MX	Pentium 4	2.4 GHz	12 Kb	512 Kb	400 MHz	333 MHz
P5K-VM	Core 2 Duo	2.66 GHz	2x32Kb*	6 Mb*	1333 MHz	667 MHz

*The cache is shared among both CPU cores.

The new CPU in table 1 was introduced roughly 2-3 Intel processor generations after the old one. The Pentium 4 (single core) CPU was introduced in the year 2000 while the newer Core 2 Duo type (dual core) processor was released in 2006. The clock frequencies at which the processors run are nearly equal, however the new CPU has two cores (i.e. two processors in one chip) running at 2.6 GHz instead of the single 2.4 GHz core of the old CPU. There has been certain development in the CPU architecture; most notably the L2 cache memory size is ten times larger and the L1 cache size is five times larger in the new CPU. The L2 cache is shared among the cores of the new CPU. If two processors are running at the same clock frequency, the caches will have the effect that the one with the larger L2 cache is the more efficient. Regarding the internal CPU architecture the instruction set of the new CPU is more advanced.

All internal and external data traffic flow through the motherboard. This printed circuit board hosts all hardware components and peripherals which communicate via the motherboard using the data bus. The data bus can be thought as a collection of wires joining every computer component together. Compared to the old PC the data bus is three times faster in the new PC.

The fastest type of memory is integrated into the CPU as the cache memory. The processor uses the cache memory for the most frequently used data and can access the data extremely fast without the data bus. The idea using cache memory is that first the CPU searches the L1 level cache. If L1 does not include the desired information the CPU accesses the L2 cache and if it does not contain the desired information the third option, the main memory, is accessed. The main memory, however, produces a considerably higher memory operations latency compared to the cache. Table 1 shows the main memory speed having doubled in the new PC. I could not find details on the cache memory speed(s), however, if they function close to the CPU clock frequency the cache of the new PC is slightly faster compared to the old PC in this respect. Other computer hardware components do not have much of an effect in the computation speed in our case.

Computation results

The applied power index program is written in C programming language. The sources were compiled into executable files in various ways in both workstations in order to maximize the comparability of the results. The program is based on the algorithms presented in Matsui and Matsui (2000) for computing the Shapley-Shubik, Banzhaf and Deegan-Packel indices. The applied program has additional routines in order to compute the Holler, Zipke and Coleman indices to initiate and prevent action. Basically, the values of the additional indices are deducted from the information provided by the original three indices. The program will also output the number of winning and losing coalitions, the number of minimal winning coalitions and the Coleman power of the collectivity to act index value.[5] It is important to notice that the computational complexity of the indices vary (within polynomial time). Matsui and Matsui (1998; 2000) and Uno (2003) provide us with the following: The Banzhaf index is the easiest to compute and in terms of complexity the values for all players can be computed in O(n^2q) time. The complexity of the Shapley-Shubik index is in the order of O(n^3q) and for the Deegan-Packel index O(n^4q), respectively. Hence, the program would be faster (and more simple) should only e.g. the Banzhaf index values be computed. As this is not possible in the current implementation the termination times below are bounded by the Deegan-Packel index. That is, the program can perform only as fast as the slowest index (Deegan-Packel) allows. For comparison the time complexity of the direct enumeration algorithm is exponential in the order of O(2^n). The magnitude of the space complexity in the current program is in the order of O(n^2).[6] The input and the programs are equal, so they should use an equal amount of memory in both computers. There are, however, very marginal differences between the two PCs in this respect due to program compilation.

For comparable results the new and the old hardware have the same Linux operating system (OS) installed (SuSE 10.3). The program sources are compiled separately in both machines ensuring the best performance. Figure 1 show the program termination times for games with 50-190 voters using a simple majority quota. Accordingly, the 50 voter game is [638; 1, 2,..., 50] and the largest 190 voter game is [9073; 1, 2,..., 190]. As the mathematical complexity analysis suggests, the measured termination times appear to follow the complexity (growth rate) of the Deegan-Packel index. Up to 100 voters the programs terminate within few fractions of a second in both computers, however in larger games the speed of the new machine is superior to the old one. Regarding the 190 voter game the program terminates just under three minutes in the new PC (167 seconds), while the old machine requires around 29 minutes to terminate. The new machine is over ten times faster regarding games larger than 140 voters, so the difference is considerable. When 10 more voters are added to the game the termination time grows around 40-50 % in each round. If we take a look at the relation between the

[5] For the index definitions see Deegan and Packel (1979), Holler (1982), Nevison et al. (1978) and Coleman (1971).

[6] It is not feasible to compare the effect of input size magnitude, which would require program execution with e.g. input size of 10, 100, 1000 and possibly even 10 000 voters.

termination times in figure 1 we see that up to 160 voters the new machine is gaining a speed difference up to 12 times. In games larger than 160 voters the relative speed of the new machine, however, begins to diminish. Figure 1 also shows the limitation of the all-in-one program. A generating functions based online program *ipgenf* by Dennis Leech computes only the Banzhaf and the Coleman indices. Giving *ipgenf* the above 190 voter game it will terminate within one second.

Figure 1. Power index program termination times and the relation between the termination times in two computers

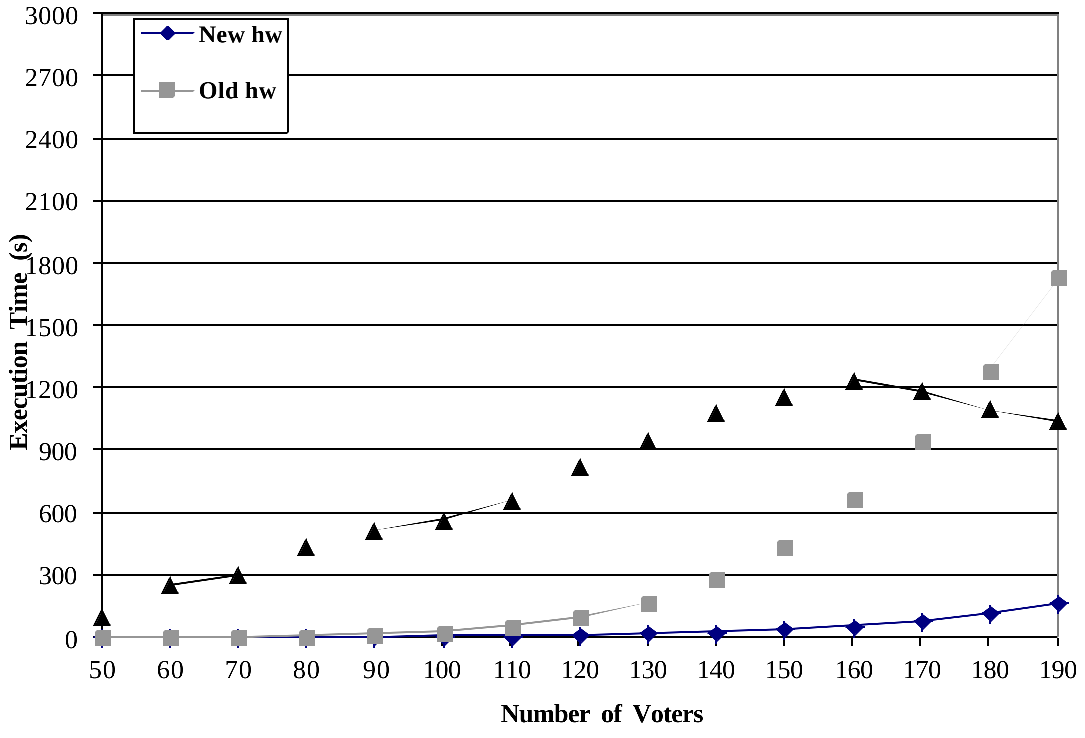
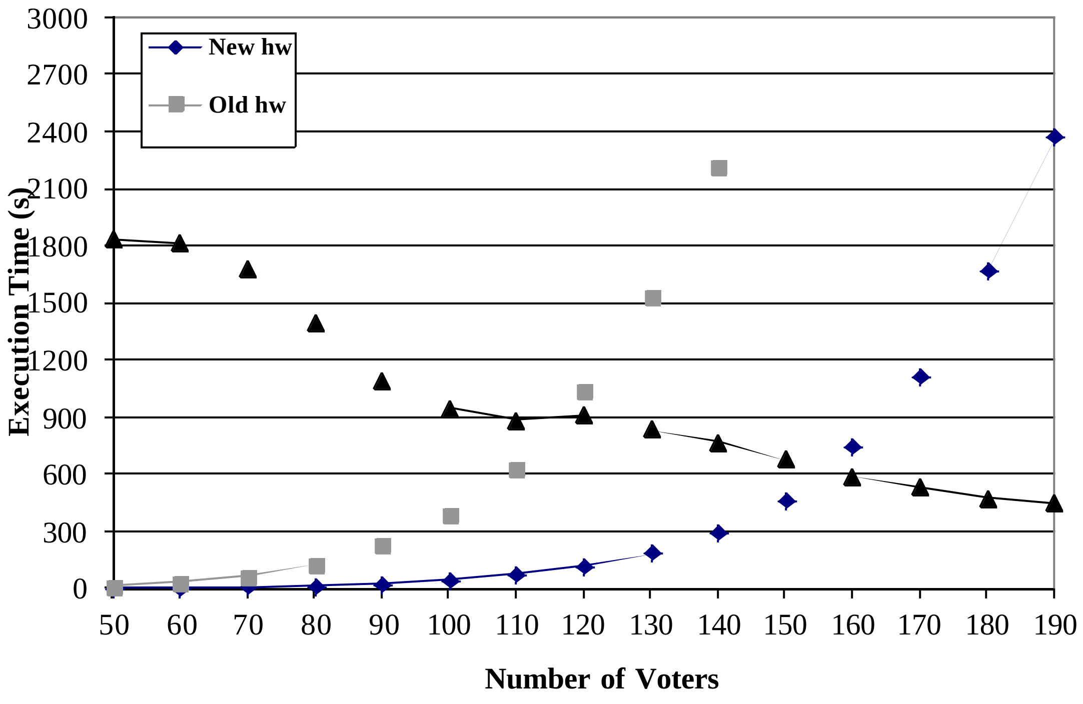


Figure 2 shows the program termination times in games with 50-190 voters where the previous vote amounts have been multiplied by ten (e.g. [6376; 10, 20,...,500]). As expected, the termination times are now considerably higher and again follow the suggested growth rate of the Deegan-Packel index. The CPUs now have to deal with larger numbers and also more memory is used compared to figure 1. This is due to the fact that the coefficient matrices become larger. Now the termination times are in the order of thousands of seconds instead of hundreds of seconds. The new hardware is now able to compute the 190 voter game in just under seven minutes while the old hardware will need approximately half an hour (nearly 10 600 seconds) to terminate. When 10 additional voters are added to the game this increases the termination times around 30-50 % in each round. Surprisingly the relation between the program termination times seems to be the opposite compared to figure 1: In a 60 voter game the new computer is roughly 18 times faster while in a 160 voter game the new computer is only six times faster and in a 190 voter game only four times faster. The difference to *ipgenf* is even more striking here as *ipgenf* will still terminate within few seconds if the 190 voter game is given to the program.

Figure 2. Power index program termination times and the relation between the termination times in two computers



Now, why is the new PC so much faster to the old one? For the answer we shall consider several variables and explanations. Accordingly, additional simulations were carried out in order to control for differences in certain hardware and software aspects.

Let us *first* consider the programs and the input i.e. the software side of the computation: The input files and the program source files are identical. However, the compiled and executable version of the program regarding the new computer might be more effective as the compiler optimizes the program for each CPU. The instruction set in the new CPU is more advanced and if the compiler is able to utilize the advantage the executable program version for the new CPU might be (substantially) more efficient. To measure and control for the impact of the instruction set, the obvious way is to compile the sources in the old hardware for the old CPU and use the obtained executable in the new hardware. As both machines are using the same OS version the differences should mainly reflect the impact of the instruction set. Performing the same sequences of computations as in figs. 1 and 2 in the new hardware, the computation takes around 35 % longer time in average. Hence, the instruction set has a clear effect, however it is only a partial explanation to the overwhelming performance of the new PC.

It was also possible to compare differences between Linux OS generations (SuSE 9.0 vs. SuSE 10.3). It could be suspected that the OS generations are not likely to affect the termination times substantially, as the programs are executed in a command shell and not necessarily requiring a graphical user interface. Indeed, performing the computation sequences in figs. 1 and 2 shows that the new OS version is faster, however the difference is only around 10 % in average. The difference is probably due to a slightly improved and more efficient Linux kernel version.

Second, among the hardware components, the CPU clock frequencies (2.4 GHz and 2.66GHz) refer to the number of operations the CPUs are able to perform within a certain period of time. The respective values are nearly equal so they cannot explain the greater speed of the new computer. The fact that the new computer architecture is a 64-bit *vis-á-vis* a 32-bit architecture in the old PC contributes mainly to the capability of the new computer to handle larger numbers with total accuracy.

Third, according to table 1 the data bus in the new computer is three times faster and also the main RAM memory modules are twice as fast. This obviously affects the computation times as the data bus sets the internal speed limit of the computer. A stream of bits can not move from the CPU to the main memory (or back) faster than the data bus allows, even if the CPU would have the capacity of performing faster. The same effect applies to the main memory modules; they can become bottlenecks in the computation process as the modules are able to exchange data only as fast as their clock frequencies allow. It is difficult to estimate precisely how much the more advanced data bus and main memory affect the computation even when the CPU clock frequencies are virtually equal. In the light of the above results, however, we know that the new computer is at least three times faster regarding any of the given inputs.[7] Hence, it is likely that the new computer can perform from three to four times faster in any similar computational task.

Finally, we consider a *fourth* possible cause, as the above factors are not able to explain the peculiar relation between the computation times but only partially. Table 1 shows the amount of the level 2 cache memory being ten times larger in the new PC. Indeed, the speed and especially the size of the L2 cache play the key role in the computation speed difference between the hardware setups. Figure 1 shows the overwhelming speed of the new computer up to 160 voters. In larger games the relative speed in new computer is not increasing any more, but rather the difference seems to diminish. This phenomenon is due to the L2 caches. The sequences of coefficient tables, which the program creates, fit considerably better in the L2 cache of the new PC. Recall that the cache memory is available to the CPU without the main memory latency. The cache in the new computer is only so large and when more voters are added to the game the coefficient table will grow in size and eventually will have to reside more and more in the main memory. The cache saturation point seems to be around 160 voters as in larger games the relative speed of the new computer begins to diminish. Using the same logic we can also explain the mirror phenomenon in the computation time relation in figure 2. Regarding games with 50-60 voters the coefficient matrices still fit mostly in the cache of the new PC. In larger games the main memory is increasingly used and the relative speed of the new computer rapidly diminishes. Eventually in a 190 voter game the new computer is only four times faster as the coefficient tables now reside for the most parts in the main memory. From this we can deduct that the same phenomenon would appear

7 The 50 voter game in figure 1 is disregarded due to measurement difficulties. The program takes only few milliseconds to terminate in both workstations and the automatic OS processes probably create fluctuation in the computation times.

also in figure 1 if games larger than 190 voters were analysed. As stated above, the new computer would probably perform from three to four times faster due to the other more advanced hardware components even if the voting game would include several hundred voters.

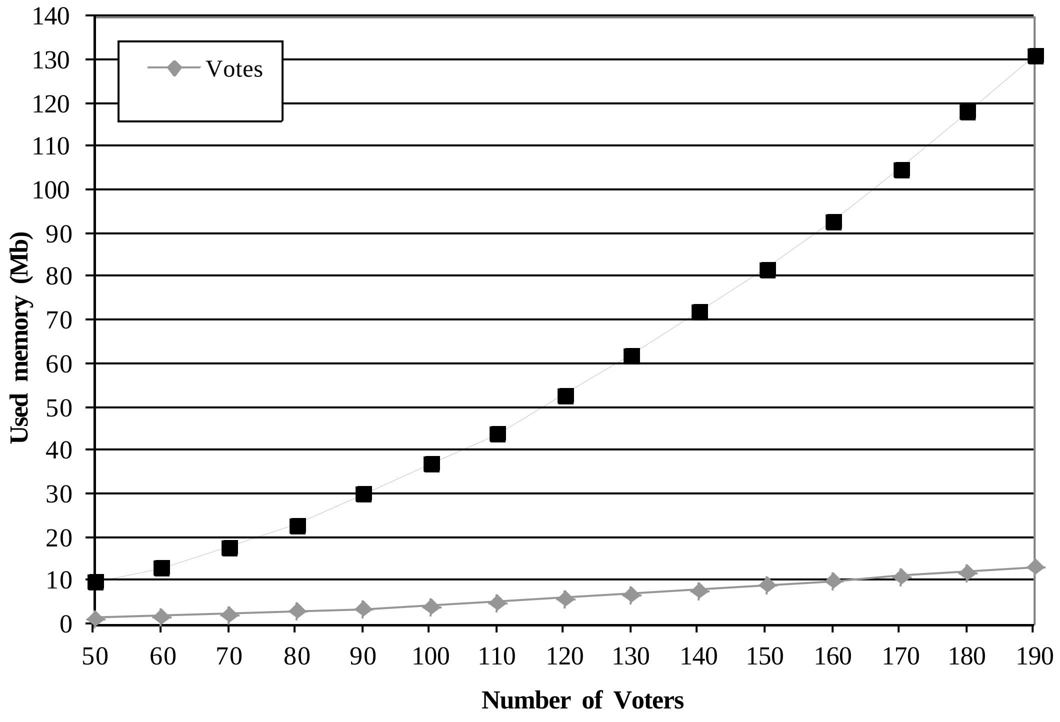
Now we shall focus on the runtime memory consumption of the computation process. This analysis is needed to confirm the above L2 cache effect. As stated above (and after certain testing), the memory consumption is virtually equal regarding both computers due to the same algorithm. The memory consumption is measured with the system utility TOP, which is run parallel with the power index program. In Linux (or UNIX) systems the TOP program is able to list several details of running or sleeping processes (tasks). One of these details is a task's currently used share of available physical memory, which is comprised of the L2 cache and the main memory. The measurement TOP is able to provide is a bit rough, however the accuracy is sufficient for our purposes.

In figure 3 we can see the process memory consumption of two sets of voting games. Referring to the grey line and games in figure 1, the observed RAM memory consumption is (i.e. the process will reserve) 1.4Mb for the 50 voter game. Growing almost linearly the 190 voter game requires approximately 13Mb of memory. The storage complexity is less demanding compared to the time complexity and so the almost linear growth rate in figure 3 does not apply for the program execution times. Multiplying the voters' votes by 10 substantially increases the required memory space: Following the black line in figure 3 the game [6376; 10, 20,..., 500] will use around 13Mb of memory. The largest 190 voter game consumes around 131Mb while the growth rate is again almost linear, however much steeper compared to the previous (recall the above difference to program execution times). Keeping in mind that modern desktop computers have several gigabytes of RAM memory available, we can conclude that in the above games the memory consumption is quite modest.[8] Rather, in our case the bottleneck seems to be the program execution time. Another result is that the

[8] The memory requirements *can* be substantial in certain cases, especially, if the total number of votes is very large. As an example, consider the 24 IMF executive directors (elected by predefined groups of countries of the IMF governing board) having a total of 2 214 607 votes. If the simple majority quota is applied, the computation process will terminate within 90 seconds, however the memory requirement is over 300Mb.

saturation point (6Mb) of the L2 cache memory of the new PC is around 120-130 voters. Apparently the use of main memory is still so marginal that up to the 160 voter game in figure 1 the new computer is still gaining relative speed. Notably the memory requirement is roughly equal between the 190 voter game in figure 1 and the 50 voter game in figure 2. The memory consumption grows in a steeper manner regarding figure 2 causing the relative speed difference to reduce between the PCs.

Figure 3. Power index program runtime memory usage under two voting scenarios



Discussion and development possibilities

With respect to the research questions we have demonstrated that the voting power of members of a voting body having the size of the U.S. Electoral College can nowadays be computed in just few fractions of a second. This result is irrespective whether the computation is carried out using an up-to-date hardware or a computer being half a decade older. A large voting body having the size of the IMF board of governors could also be computed with both modern computers, however now the program will terminate considerably faster in the new workstation. For comparison Mann and Shapley (1962, 7) used an IBM 7090 model mainframe to measure voting power in the U.S. Electoral College and their FORTRAN program took around 70 seconds to terminate.[9] The IBM 7090 generation mainframe had roughly the size of a volleyball field and typically was worth $2 900 000 at the time. The inspection of the applied main hardware components show that the more advanced architecture of the new CPU (larger cache memory and a more advanced instruction set) is the key to the speed of the new hardware. In every computational scenario the new PC will perform at least 3-4 times faster to the old one. The memory requirements in the above voting game examples were found to be rather modest as the most demanding computational task used about 132Mb RAM memory, while the available amount of RAM memory in standard desktops is several gigabytes. However, it has to be noted that when the total amount of voters and especially votes is very large, the memory requirements can be substantial.

The applied program is an all-in-one implementation. The main strength of the program is that it will output results regarding many indices. The main weakness of the program is that it cannot perform faster as the most complex index allows; in our case the Deegan-Packel index. The termination times would be substantially lower if only the least complex indices (the Banzhaf indices) are computed. An obvious enhancement to the program would be to allow the user to choose which indices are to be computed. Currently everything is executed automatically. The computation algorithms could also be improved. According to Uno (2003) the complexity of the algorithms could be significantly reduced: The Banzhaf indices for all voters could be computed in only

[9] The three Electoral College variants Mann and Shapley analysed were comprised of only 18 or 19 different size voters as many states had the same amount of votes. This reduces memory requirements as well as execution time compared to our 50 voter example.

O(nq) time and the Shapley-Shubik and the Deegan-Packel indices in O(n^2q) time. Currently the complexities are in the order of O(n^2q), O(n^3q) and O(n^4q), respectively. The program execution on the hardware could possibly be enhanced, if the computational task could be parallel with respect to the multiple cores of modern CPUs. There might still be certain slowing bottlenecks, from which the CPU cache size and the speed of the data bus and main memory are the most important. The computation could also be enhanced with respect to space complexity. The applied algorithms seem to do no unnecessary operations, however the data structure regarding certain variables could be changed from a matrix into a linked list (Uno 2003). The coefficient matrices can grow substantial in size, however it is known that the matrices are sparse as majority of the cell values are zeros. The more there are voters and especially the larger the vote total is, the more zeros appear in the coefficient matrices. The idea behind the linked list is that when the first zero is encountered in a matrix row, this zero is replaced with a link pointing to the beginning of the next row. This is possible as the mathematical properties stemming from the generating functions suggest that the first zero in a row is only followed by more zeros. The idea of a linked list in combination with power index computation has been suggested at least by Uno (2003) and Lindner (2004). In terms of space complexity it would be interesting to compare the original storage method to a modified linked list algorithm. Of equal interest would also be to compare the program execution times under both storage scenarios: A linked list is a slower data structure compared a two-dimensional vector, however a linked list would require far fewer memory operations. In any case, a linked list data structure would at least potentially enable the analysis voting bodies in which the total amount of votes is in the order of 10^6 or possibly even 10^7.

References

Alonso-Meijide, J., Bilbao, J., Casas-Mendéz, B. and Férnandez, J. 2009. Weighted multiple majority games with unions: Generating functions and applications to the European Union. European Journal of Operational Research 198, 530-44.

Alonso-Meijide, J. and Bowles, C. 2005. Generating Functions for Coalitional Power Indices: An Application to the IMF. Annals of Operations Research 137, 21-44.

Banzhaf, J. 1965. Weighted Voting Does not Work: A Mathematical Analysis. Rutgers Law Review 35, 317-43.

Brams, S. and Affuso, P. 1976. Power and Size: A New Paradox. Theory and Decision 7, 29-56.

Coleman, J. 1971. Control of Collectivities and the Power of a Collectivity to Act. In Lieberman (ed.), *Social Choice*. New York: Gordon & Breach, 269-300.

Deegan, J. and Packel, E. 1979. A New Index of Power for Simple n-Person Games. *International Journal of Game Theory* 7:2, 113-23.

Felsenthal, D. S. and Machover, M. 2000. Enlargement of the EU and Weighted Voting in its Council of Ministers. VPP 01/00 Centre for Philosophy of Natural and Social Science.

Felsenthal, D. S. and Machover, M. 1998. *The Measurement of Voting Power: Theory and Practice, Problems and Paradoxes*. Cheltenham: Edward Elgar.

Gambarelli, G. (ed.). 2007. Power Measures IV. *Homo Oeconomicus* 24:3-4, special issue.

Gambarelli, G. and Holler, M. (eds.). 2005. Power Measures III. *Homo Oeconomicus* 22:4, special issue.

Holler, M. 1982. Forming Coalitions and Measuring Voting Power. *Political studies* 30, 262-71.

Holler, M. and Owen, G. (eds.). 2000. Power Measures I. *Homo Oeconomicus* 17:1-2, special issue.

Holler, M. and Owen, G. (eds.). 2002. Power Measures II. *Homo Oeconomicus* 19:3, special issue.

Hosli, M. 1993. Admission of European Free Trade Association States to the European Community: Effects on Voting Power in the European Community Council of Ministers. *International Organization* 47, 629-43.

Lambert, J. 1988. Voting Games, Power Indices and Presidential Elections. *UMAP Journal* 9, 216-77.

Leech, D. 2002a. Designing the voting system for the Council of the European Union. *Public Choice* 113, 437–64.

Leech, D. 2002b. Computation of Power Indices. Warwick Economic Research Papers, Number 644. URL: http://www.warwick.ac.uk/~ecrac/twerp644.pdf.

Leech, D. 2003. Computing Power Indices for Large Voting Games. *Management Science* 49, 831-38.

Lindner, I. 2004. Power Measures in Large Weighted Voting Games: Asymptotic Properties and Numerical Methods. University of Hamburg.

Lucas, W. 1978. Measuring Power in Weighted Voting Systems. In Brams, S., Lucas, W. and Straffin, P. (eds.), *Modules in Applied Mathematics 2: Political and related Models*,183-238.

Mann, I. and Shapley, L. 1960. Values of Large Games, IV: Evaluating the Electoral

College by Monte Carlo Techniques. *Memorandun RM-2651, The RAND Corporation*, Santa Monica.

Mann, I. and Shapley, L. 1962. Values of Large Games, VI: Evaluating the Electoral College Exactly. *Memorandun RM-3158-PR, The RAND Corporation*, Santa Monica.

Matsui, T. and Matsui, Y. 2000. A Survey of Algorithms for Calculating Power Indices of Weighted Majority Games. *Journal of Operations Research Society of Japan* 41, 71-86.

Matsui, Y. and Matsui, T. 1998. NP-completeness for Calculating Power Indices of Weighted Majority Games. *Technical Report METR 98-01*. Dept. of Mathematics, Faculty of Science, University of Tokyo.

Nevison, C., Zicht, B. and Schoepke, S. 1978. A Naive Approach to the Banzhaf Index of Power. *Behavioral Science* 23, 130-131.

Owen, G. 1972. Multilinear Extensions of Games. *Management Science* 18, 64-79.

Owen, G. 1975. Multilinear Extensions and the Banzhaf Value. *Naval Research Logistics Quarterly* 22, 741-50.

Pajala, A. 2003. *Expexted Power and Success in Coalitions and Space*. Annales Universitatis Turkuensis, ser. B, no. 253.

Papayanopoulos, L. 1981. Computerized Weighted Voting Reapportionment. AFIPS Joint Computer Conferences. Proceedings of the National Computer Conference, May 4-7, USA, 623-29.

Penrose, L. 1946. 'The Elementary Statistics of Majority Voting'. *Journal of the Royal Statistical Society* 109, 53-7.

Shapley, L. 1953. A Value for n-Person Games. *Annals of Mathematics Study* 28, 307-

17.

Shapley, L. and Shubik, M. 1954. 'A Method of Evaluating the Distribution of Power in a Committee System'. *American Political Science Review* 48, 787-92.

Uno, T. 2003. Efficient Computation of Power Indices for Weighted Majority Games. *NII Technical Report NII-2003-006E*. National Institute of Informatics, Japan.

Widgrén, M. 1994. Voting Power in the EU and the Consequences of Two Different Enlargements. *European Economic Review* 38, 1153-70.

Widgrén, M. 1995. Probabilistic Voting Power in the EU Council: The Cases of Trade Policy and Social Regulation. *Scandinavian Journal of Economics* 97, 345-56.

Widgrén, M. (ed.), 2000. Symposium: Power Measures. *Homo Oeconomicus* 16:4.

Widgrén, M. and Napel, S. 2008. Shapley-Shubik vs. Strategic Power: Live from the UN Security Council. In Braham, M. and Steffen, F. (eds.), *Power, Freedom and Voting*, Springer, 99-118.