# A new Parallel Genetic Algorithm for Energy Management

Peter Palensky
Institute of Computer Technology
Vienna University of Technology
Gusshausstrasse 27-29/384
A-1040 Vienna, Austria

## 1 Introduction

Energy management in this work is seen as a problem that consists of a number of sub-problems. Individually solving the sub-problems does not necessarily mean to solve the overall problem at the same time. It is the particular property of such a problem that minimums (or maximums) of the individual sub-problems do not sum up to a minimum (or maximum) of the overall problem. This multi-parameter problem will be solved by a new type of parallel genetic algorithm (PGA).
The problem of energy management is abstracted to a number of resource consuming entities (or devices) that have a certain future behavior like a load chart of electrical energy. Each entity has a number of such possible future load charts (or device states) that are represented by a number.

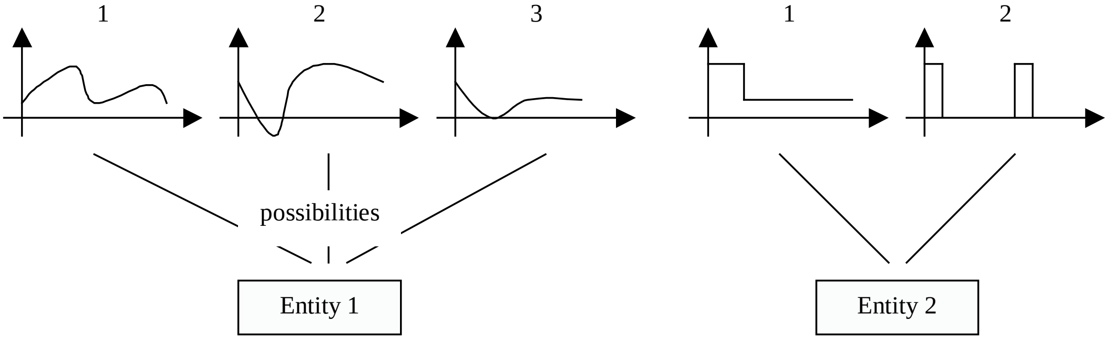


**Figure 1: Entities with different possible future behavior**

Figure 1 for instance shows two devices. The first one has three possible future device states and the second only two. The devices offer these possibilities to the optimizing algorithm by using their device specific and local knowledge. It might be that a device has only one possible state, but sometimes devices can chose between a couple of alternatives. Typical examples are "programmed" devices that execute some sequence or control algorithm like refrigerators, washing machines, air conditioning and the like. These devices offer a finite number of different short-term future device states that do not affect the original purpose of the device (keeping the food cooled).

Suppose now that we have a number of AN such devices whose device states sum up to a system state. The system state is a load chart again that is (in our example) for instance the arithmetical sum of the individual device state load charts.

This means that each entity can chose out of some number of possibilities and contributes to the system state – either as a problem or a solution to the problem. We well from now on not distinguish between a device state that "solves" or "is" a problem, the mathematical background is always the same (consuming, storing or producing a resource in a timely manner). Sub-problems are summed up to an overall problem (Figure 2) that can be evaluated by a so-called utility function.

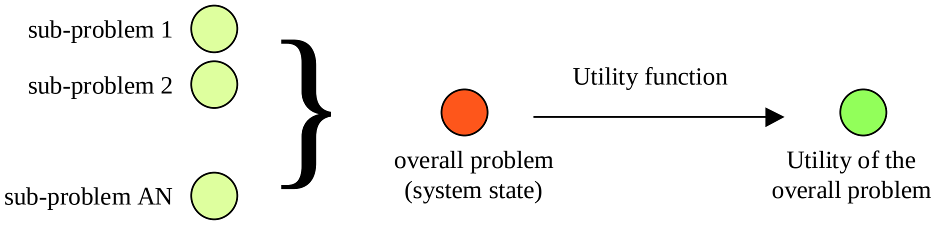


**Figure 2: sub-problems and utility function**

This utility function is not applicable to a sub-problem in a meaningful way but only to the overall problem. It outputs a value (for example between 0 and 1) that is higher for "good" system states. This means for instance that power-peaks or consumption during high-tariff-time are "punished" while distribution of load over time is honored.

Having this utility function the only task is to find the right combination of device states that maximize the utility value. This is a combinatorial multi-parameter optimization problem.

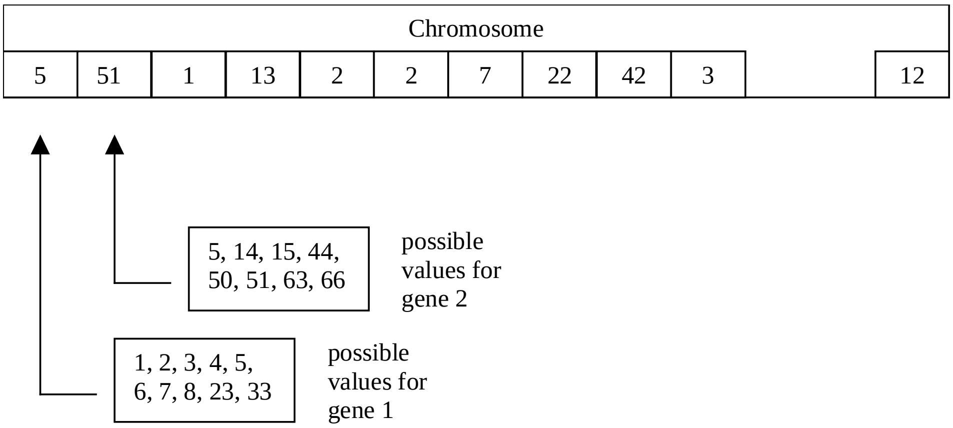
## 2 Representing the problem as a chromosome

Solving a multi-parameter problem by means of genetic algorithms is not new [Paechter98]. The individual parameters are represented as assemble to a chromosome. We have a number of AN genes (or devices with a certain device state, or sub-problems) and therefore the chromosome has a length of AN genes. One such a chromosome represents a system state (Figure 3).

| Chromosome | | | | |
|---|---|---|---|---|
| gene for sub-problem 1 | gene for sub-problem 2 | gene for sub-problem 3 | | gene for sub-problem AN |

**Figure 3: a chromosome that represents a system state**

Each gene can have a sub-state chosen out of a number (lets call it DS) of possible sub-states at a given point of time where the optimization starts. These sub-states have certain properties that take influence on the utility of the overall solution (or the utility of the chromosome). If for instance all genes have DS=100 possible sub-states the system state has 100^AN possible states i.e. there are 100^AN possible different contents for the chromosome. The goal now is to find a chromosome content that fulfills the requirements of the utility function. If the utility function transforms a chromosome into a value between 0 and 1 the goal could be to find a chromosome that results in a utility value of greater than 0,8.
Genes will be represented as integer numbers (that represent the possibilities in Figure 1) and the chromosome as a tuple of AN such numbers.



The possible values for the genes contain not only their numerical representation but also their properties for the utility function (the individual load charts). Having a number of such chromosomes and a given utility function it is possible to search for the best of them. But with increasing numbers of devices and possibilities the number of possible chromosomes gets very large and it becomes impractical to search in a traditional way. Genetic algorithms help to search such large “search spaces”.

## 3 A parallel genetic algorithm based on agent technology

Our problem is now a search space. We represent the elements of this space as possible values of a chromosome. This work is about a new variant of a parallel genetic algorithm that searches the space. Each gene will be represented by one software agent.
In this document, software agents shall be seen as autonomous software programs that can interact with other agents to solve for instance a common goal [Franklin97]. These agents can be run on different platforms or on one and the same as long as they can communicate with each other.

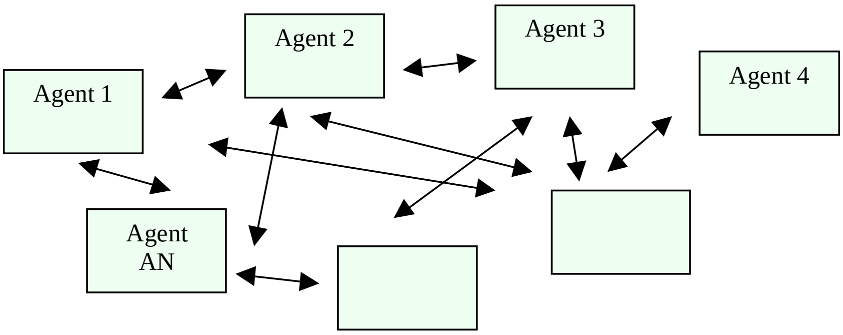


**Figure 4: Software Agents communicate with each other**

Agent communication happens peer-to-peer and in an asynchronous manner [Petrie97]. Agent communication is offered by some sort of network operating system. Every agent represents, as already mentioned, one gene of the chromosome – means one sub-problem. A PGA works with a number of chromosomes in parallel – the population. The members of such a population are called individuals. Each individual represents a full system state and can be evaluated with the utility function.
This means the population is distributed in a new way. Usually the individuals are distributed, in groups of the like, on different programs [Whitley94]. Here the chromosomes of the individuals are distributed – each agent takes care of one and the same gene position of all individuals as shown in Figure 5.

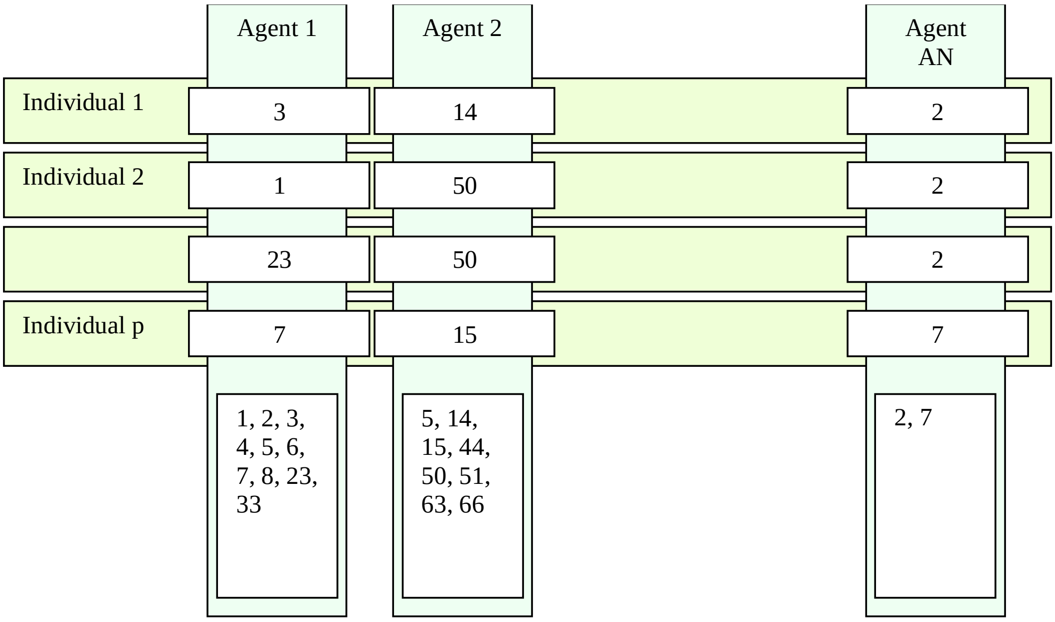


**Figure 5: Agents with distributed chromosomes and local knowledge**

Agent 1 stores all genes number 1 of all individuals. It also has the "local knowledge" that belongs to gene number 1 – the possible values for this gene. One individual has its genes spread over all agents. To evaluate such a individual it is necessary to "ask" each agent for the gene and its properties, to collect all AN genes of one individual and to subsequently apply the utility function to the resulting chromosome. The knowledge about the population is therefore highly distributed.

The advantage is the object-oriented modeling of the problem. Each new problem that "joins" the overall problem is tackled with a new agent that contains all information of the new member. This does not necessarily increase the population or the memory requirements of the individual agents. Having all these agents, it is near to hand to use them in a way of parallel computing power which is another advantage.

The agents are thought to be hosted on "intelligent devices" that are networked with some sort of home- or building automation network [Palensky99] or on small industrial PCs that server consumers in the range of a house or a small business facility.

The algorithm works as follows:

1. One agent (Agent A) decides (or is told by whomever) that the system must be optimized immediately.
2. It invites all participating agents by means of a broadcast message, assigns each individual to an agent and asks the agents to evaluate "their" individual(s).
3. All agents evaluate their assigned individual(s) in parallel and report the result to agent A.
4. Agent A sorts the individuals by their utility (fitness).

5. It calculates the rules for creating a new generation out of the PGA parameters and the fitness-list and tells them to the other agents.
6. All agents perform selection, crossover and mutation of their genes in parallel.
7. Agent A decides if another generation is necessary. If yes, it tells some other agent to do this next generation.

Point 6 is a special one that distinguishes this algorithm from ordinary parallel genetic algorithms. Let us suppose a population of 7 with a chromosome-length of 5. The following table gives the population already sorted by their fitness (Individual 5 has the largest value of fitness and is therefore at number 1). It shows the number of the individuals, the content of their 5 genes and their fitness.

| *Ind.* | *G1* | *G2* | *G3* | *G4* | *G5* | *Fitness* |
|---|---|---|---|---|---|---|
| *5* | 2 | 4 | 6 | 12 | 76 | 0.5432 |
| *3* | 23 | 45 | 67 | 44 | 2 | 0.4432 |
| *1* | 1 | 54 | 7 | 4 | 2 | 0.4104 |
| *4* | 2 | 4 | 67 | 8 | 43 | 0.3769 |
| *2* | 23 | 12 | 12 | 34 | 33 | 0.2700 |
| *7* | 28 | 23 | 42 | 713 | 21 | 0.1624 |
| *6* | 85 | 23 | 123 | 122 | 254 | 0.1023 |

If agent A decides that the first 3 are the surviving parents (survival rate of 3/7=43%) that will generate the next generation of a “generational replacement GA” (also called “basic GA”) the last 4 ones will be replaced by the children. Agent A broadcasts the information about the mating procedure that looks for instance like this:

“((5+3=4) (3+1=2) (5+3=7) (5+1=6))”

This string means that individual 4 will be generated out of the genes of individual 5 and individual 3 and so on. This string is sent out to all agents via a broadcast message. Each agent can only manipulate its own set of genes. So each agent will have to delete genes number 4, 2, 7 and 6. Afterwards it takes a 50% chance if gene 4 is filled with the value of gene 5 or with the value of gene 3. This happens with all genes of all children in a parallel way on all agents. Due to the random factor in choosing the parent the outcome for the offspring it is unpredictable.

The same can be said for the mutation phase. If the overall mutation rate is 30% each gene has a chance to be mutated of 30%/chromosome_size/population_size. Mutating means that a random value of the set of possible genes is taken and written to the gene that has to be mutated. Agent A sends out this “gene mutation rate” and all agents perform the mutation phase in parallel.

So the genetic algorithm has two special aspects:

- it performs unpredictable distributed multi-point crossover
- if performs distributed mutation

All the agents have the same software program. Simply one starts the procedure by calculating one generation. The whole algorithm can terminate if no further success can be achieved for a certain number of generations, if a fixed number of generations were calculated or when the utility value satisfies the goal.

## 4 The Implementation

The requirements for the chosen software framework for the reference implementation are:

- It should offer a general-purpose programming language to implement the agent software.
- It should offer parallel or pseudo parallel execution of multiple agents on one hardware platform.
- It should offer peer-to-peer and broadcast agent communication between agents on one hardware platform as well as between agents on different hardware platforms, preferably based on TCP/IP network communication.
- An agent communication language like KQML [Finin94] or FIPA [Steiner98] should be supported.
- It should offer the standard functionality of a simulation environment (time base, graphical representation and analysis of the results, and the like).
- It should be free.

The framework of my choice was JATLite [Leon00]. It offers a KQML parser and a template for Java-Agents. Communication is done via a message router that is provided by JATLite.

The agents were implemented as a multi-threaded JAVA console application that can host a random number of agents. Agents within one of these applications communicate in the same manner like with agents in other applications or even machines. All communication is done via TCP-socket based KQML messages.

The simulation does the following:

1. It gets a (random) system state and all possibilities for the genes
2. It optimizes the system state for a predefined number of generations using a given utility function

Two aspects of the algorithms are logged into output files:

1. The number of exchanged packets
2. The number of performed computations

The exchanged packets of interests (the really large and therefore expensive ones) are records about the system state. They have to describe the future load chart in some table. This table is of fixed length to make the packet size constant and counts as one communication unit (1 packet).

The computations are counted as numbers of executing one utility function calculation. All other calculations are not counted. If 10 agents are performing 30 utility computations it costs 3 units, because they do the calculations in a parallel way.

# 5 Results

The PGA performs well and produces generations of increasing utility. The interesting questions are what parameters take influence on the performance of the algorithm.

The algorithms were always started with the following input data:

- DS the number of possible device states (gene possibilities)
- P population size
- AN number of agents
- S survival rate
- M mutation rate
- N number of generations to be computed

These parameters are given as a string of the form DSxxPxxANxxSxxMxxNxx. The costs for the optimization are the exchanged packets (packet) and the computation cost (c). The following charts show some results of the test runs.

The algorithms tend to converge to a maximum value that is better the larger the population is. There is no way to determine what the best value (absolute maximum) is, all algorithms find just some good value (local maximum), that might fulfil the requirements of the utility function. Figure 6 shows the achieved utility values with increasing population size.

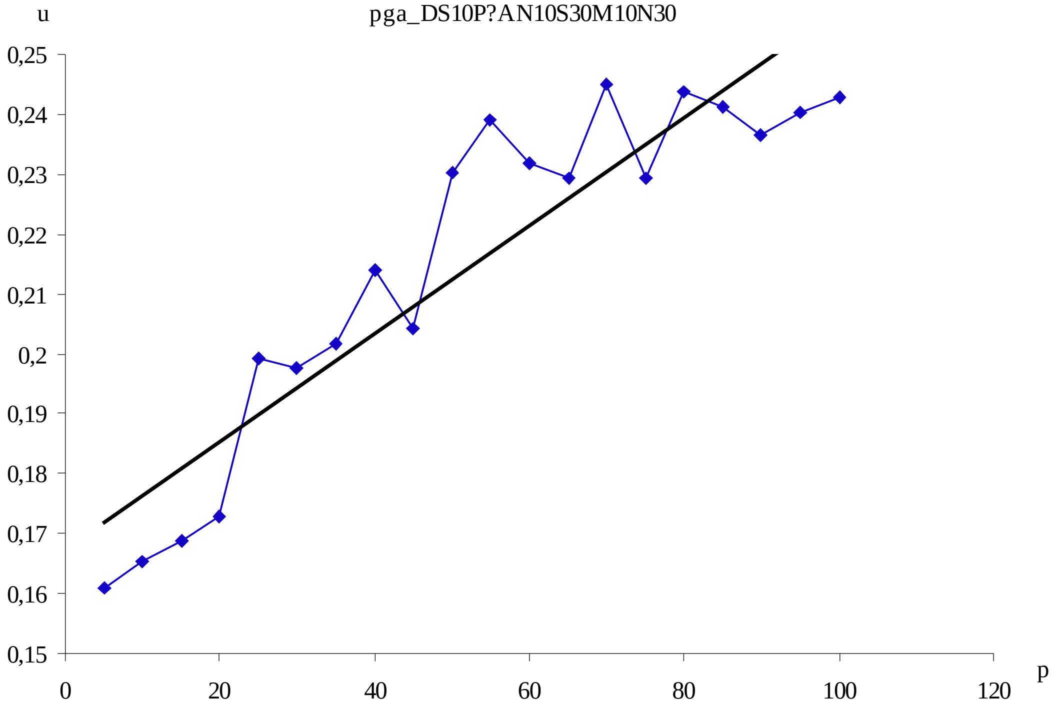


**Figure 6: the effect of the population size**

The thick line shows the trend curve. We see that the size of the population has an obvious influence on the quality of the result. The drawback is that a larger population causes more network traffic between the agent which can slow down the algorithm significantly.

One interesting aspect is the beginning of the sequence. It would be advantageous to find the optimum between caused packet load and optimization speed. The following image shows that a population size of 30 or 40 finds a better solution in a shorter time than a system with p=50 (although p=50 finds a better one finally).

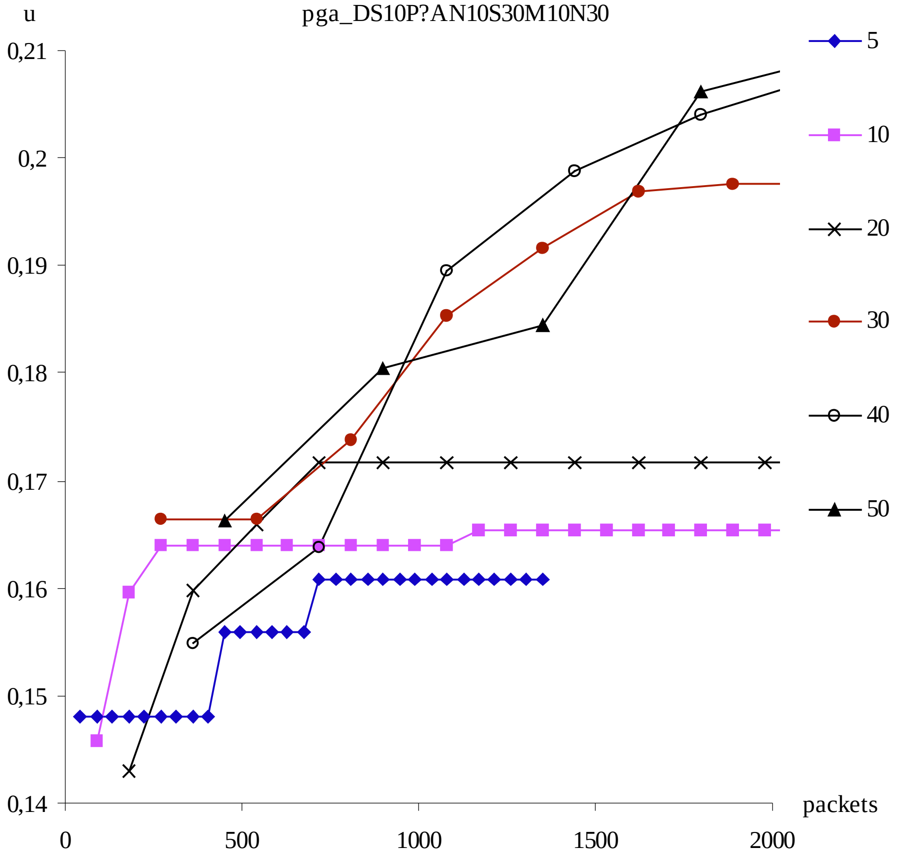


**Figure 7: the first steps of the optimization**

Small populations seem to have absolutely no chance. Small means small compared to the size of possible genes (DS). In the above image DS=10.

The seed (the random initial population) has also an influence on the result. Sometimes the seed is very bad and the algorithm can not extract and mate a good solution out of it. Increasing the mutation rate helps in this case.

Beside the mutation rate the survival rate is important and not easy to chose. The following and last chart shows that a mutation rate of 45% seems to be the ideal value.

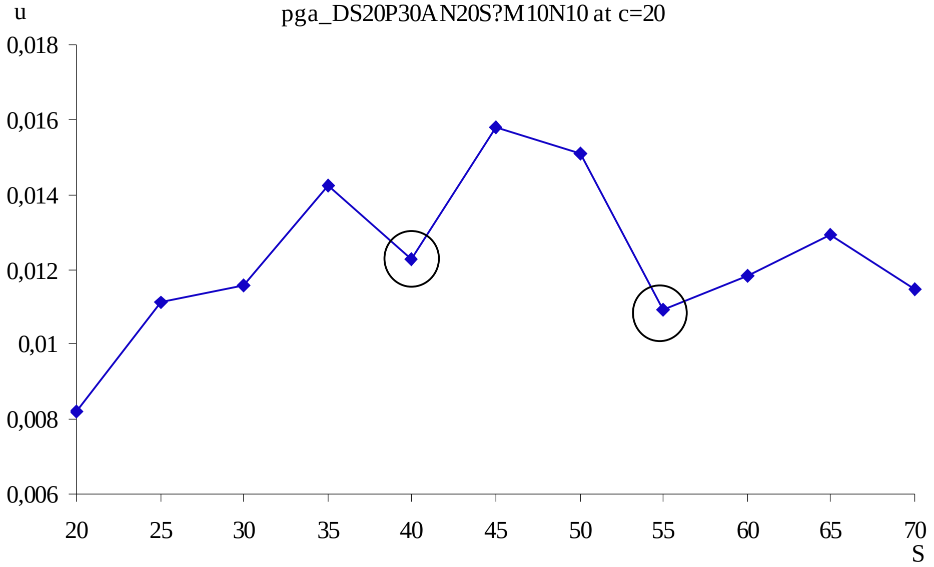


**Figure 8: impact of the survival rate**

The two exceptions at S=40% and S=55% are caused by a bad seed that could not be solved with a mutation rate of 10%. Higher mutation rates help to overcome a bad seed of the PGA.

## 6 Conclusion

The way to represent sub-problems by a single agent helps to structure the software and to increase the scalability of the algorithm. Scalability was one goal of this work. Adding one entity (or problem) to the system should not require reprogramming or upgrading the existing members of the agent society. With this PGA a new agent does not necessarily cause additional memory requirements for the other agents. The chromosome gets longer, but the new agent takes care of it.

The problem of energy management in this context is especially suited for this kind of distributed algorithm. Every sub-problem (means energy consuming, -storing or producing device) that adds to the overall problem does not only increase the complexity of the problem but also the computational power of the system. Supposing that every device is equipped with a field area network node or some other networked controller it helps to cope with the added complexity.

Generally it seemed that the algorithm is pretty slow, which might have two reasons:

- the JAVA programming language is of poor computational performance
- the socket-based, routed and redirected communication slows down the algorithm

One possible improvement would be not to use the JATLite message router but to address the agents directly (which is by the way also possible with JATLite). But therefore each agent has to have a list of the addresses of the other agents (because of the missing broadcast functionality). This would decrease the scalability of the software. Another possibility to overcome this problem is to use specialized, broadcast-able protocols where agents can join multi-agent-system by simply listening to some broadcast port.

Generally this method is a pretty simple one of modeling such complex systems. Other scheduling or optimization techniques require deep insight into the structure of the problem [Rana96]. Here the devices that join the optimization system provide a simplified and easy-to-find set of their possible contributions to the problem.

Further research will be done on how the devices can give more and more accurate predictions about their possible future states. Historical data and heuristic insight will be used to generate a “good” set of possible genes. The current tests are run with random genes and not with real-world values. Having devices modeled, it is possible to make estimations or statements about their future behavior.

Another aspect for future research is to minimize communication. Some networks (Internet via telephone lines for instance) cause costs not only by their throughput but also by their schedule of use. Costs can depend on the time of the day or on the number of “being on-line”. If the agent cooperation can be based on the exchange of message that does not happen that often it would help to decrease costs in this sense as well.

This algorithm has a number of possible improvements but it already seems that the approach of modeling the problem of scheduling energy consumption via genetic algorithms is very promising. Deeper insight into the problem and its structure will help to increase its performance.

## Literature

[Finin94] Tim Finin, Richard Fritzon, Don McKay and Robin McEntire: “KQML as an Agent Communication Language”, in: Proceedings of the third International Conference on Information and Knowledge Management (CIKM’94), ACM Press, 1994

[Franklin97] S. Franklin and A. Graesser: “Is it an agent, or just a program? A taxonomy for autonomous agents”, in Jörg P. Müller, Michael J. Wooldridge and Nicholas R. Jennings (editors): “Lecture Notes in Artificial Intelligence 1193, Intelligent Agents III, Proceeding of ECAI’96 Workshop”, ISBN 3-540-62507-0, Springer, 1997

[Leon00] H. Jeon, C. Petrie, M. R. Cutkosky: „JATLite: A Java Agent Infrastructure with Message Routing“, IEEE Internet Computing, Mar./Apr. 2000

[Paechter98] Ben Paechter, R.C. Rankin, Andrew Cumming and Terence C. Fogarty: ”Timetabling the Classes of an Entire University with Evolutionary Algorithms”, in Agoston E. Eiben, Thomas Beck, Marc Schoenauer and Hans-Paul Schwefel (eds.): “Proceedings of the 5th international conference on Parallel problem solving from nature - PPSN V", Amsterdam, September 1998 , Springer, Berlin, 1998

[Palensky99] Peter Palensky and Mikhail Gordeev: "Demand Side Management by using Distributed Artificial Intelligence and Fieldbus Technology", in Proceedings of Intelligent and Responsive Buildings Conference, Brugge, 1999

[Petrie97] Charles J. Petrie: "What is an Agent?", in Jörg P. Müller, Michael J. Wooldridge and Nicholas R. Jennings (editors): "Lecture Notes in Artificial Intelligence 1193, Intelligent Agents III, Proceeding of ECAI'96 Workshop", ISBN 3-540-62507-0, Springer, 1997

[Rana96] Soraya Rana, Adele E. Howe, L. Darrell Whitley and Keith Mathias: "Comparing Heuristic Search Methods and Genetic Algorithms for Warehouse Scheduling", in Proceedings of the Third Artificial Intelligence Planning Systems Conference (AIPS-96), 1996.

[Steiner98] Donald Steiner: "Die FIPA-Initiative für Agenten-Standardisierung", in: "it+ti", Vol. 4/98, ISSN 0944-2774, Munich 1998

[Whitley94] Darrell Whitley: "A Genetic Algorithm Tutorial", in Statistics and Computing Volume 4, 1994