Internal Report 2010-02 January 2010

Universiteit Leiden

Opleiding Informatica

Multiobjective Particle Swarm Algorithm for Building Design Optimization

Barry van Veen

BACHELORSCRIPTIE

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Contents

List of Figures

List of Tables

1 Introduction

1.1 Bachelorproject

This thesis is a report on my bachelor project at LIACS. The goal of this project is to get more acquainted with doing research and to cooperate in real research. Furthermore it is a way to use some skills learned during the bachelor.

1.2 Multi-objective optimization

This thesis is on multi-objective optimization (MOO). Optimization is about finding an optimal solution for $f(x)$ with $x \in \mathcal{X}$. MOO is optimizing with multiple objectives, we want to find an optimal solution for $f_1(x), ..., f_m(x)$ with $x \in \mathcal{X}$.

MOO problems are hard to solve because of a few reasons: first of all the search space is usually very big. In an n objective optimization problems we have n different search spaces that we all have to explore in order to find good solutions.

A second challenge is that we have to achieve more than one goal. Often these goals are conflicting, finding an optimal solution for one of the objectives may give a bad result on another objective.

From the latter follows that there is a definition problem: what is a good result? Can a result that optimizes just one objective be considered a good result? Are the results that optimize all objectives the only results that we are looking for? How do we compare different results?

1.2.1 Pareto dominance

To be able to compare different MOO results we use the Pareto dominance relation. This relation introduces the notion of Pareto dominance.

For any $y^{(1)} \in R^m$ and $y^{(2)} \in R^m$: $y^{(1)}$ dominates $y^{(2)}$ if and only if: $\forall i = 1, ..., m$: $y_i^{(1)} \leq y_i^{(2)}$ and $\exists i \in \{1, ..., m\} : y_i^{(1)} < y_i^{(2)}$.

With this notion of Pareto dominance we can say that dominating solutions are better than the solutions they dominate. Even more important is that there are non-dominated solutions. These solutions are not dominated by any other solution, but this does not mean that there is just one non-dominated solution. Two non-dominated solutions can exist at the same time because they are incomparable

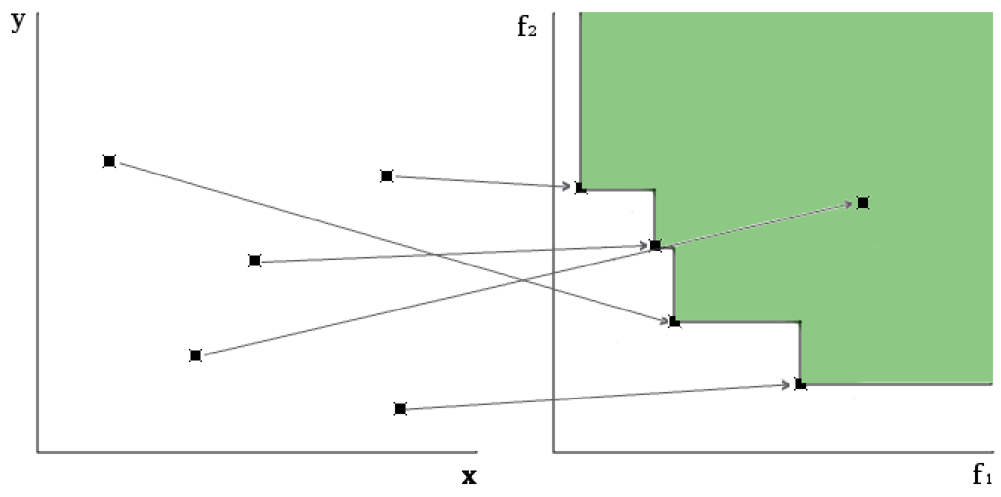


Figure 1: Illustration of the mapping of points in the search space (left) to solutions in the objective space (right). The green area is dominated by the non-dominated points.

with each other. For example, solution A is better in objective one than solution B, while solution B is better in objective two. In such a situation we cannot choose between A and B because we think both objectives are equally important. Therefore A and B are said to be incomparable.

All non-dominated points are part of the Pareto front of the problem. The input parameters that belong to these solutions are the actual solutions of the MOO problem. So, what we are actually looking for are input parameters that lead to non-dominated objective function vectors. These input parameters can afterwards be investigated by experts in the MOO problems field to choose one single solution to use in the real world. In the case of this thesis this might be a building engineer looking for a good solution in the area of building performance, which is also practical to build.

Note, that there is a possibility that different sets of input parameters produce the same objective function values.

1.3 Multi-Objective Particle Swarm Optimizer

As an extension to the work of Marijt [1] MOO on a real world problem will be studied in this thesis. Marijt researched how an evolutionary algorithm, the SMS-EMOA algorithm [2], can be used in order to optimize the thermal efficiency and comfort of building designs based on a building performance simulation tool.

Instead of using the SMS-EMOA algorithm as in Marijt [1] this thesis will investigate the Multi Objective Particle Swarm Optimizer (MOPSO) as it was proposed in [3]. The main question we want to answer is whether or not this algorithm will yield good results on the real world problem of building performance optimization. To do this we will test different parameterizations and at the end we will compare the results to those obtained with SMS-EMOA.

1.4 Vabi Building Simulator

The real-world-problem that we use for this study is the Vabi building simulator [4], the same as in [1]. The simulator is a black box from the point of view of the optimizer. It uses 77 input parameters that represent different aspects of a building. Among these are for instance the thickness and material of which the walls are made, the number of people in the building and a binary variable that decides whether or not double glass windows are used.

As an output we will read the numbers representing the 2 objectives. Objective 1 is the thermal comfort, represented by the numbers of weighted overheating hours (WOH+) and weighted underheating hours (WOH-). The second objective is the energy consumption measured in annual heating and annual cooling, both in kWh. We want to optimize (minimize) both objectives.

The Vabi building simulator was used while designing “het Bouwhuis” in Zoetermeer. “Het Bouwhuis” is the headquarters of Bouwend Nederland, an organisation of construction companies in the Netherlands. When Bouwend Nederland designed the building they chose from two alternatives for regulating the heading and cooling facilities. Option one was a regular all airconditioned system. Option two involved transparent facades on most of the buildings floors. After calculations option two was chosen, which resulted in a energy consumption that is 24 percent below the standard.

In this thesis, certain parameters of the building are optimized. These parameters were selected based on a sensitivity analysis in [5]. In [5] the effect of the parameters on the design objectives, thermal comfort and energy consumption, was compared.

The remainder of this thesis is structured as follows:
Section 2 introduces the MOPSO algorithm. Then, section 3 deals with the tests that are done to prove that the MOPSO implementation works properly. Section 4 describes the results on the Vabi building simulator. Finally, section 5 will consist of conclusions that can be drawn based on the results, together with some recommendation for future research.

Figure 2: Het Bouwhuis in Zoetermeer

2 The MOPSO

2.1 What is a MOPSO?

A particle swarm optimizer (PSO) is an algorithm based on the behavior of flocks of birds or schools of fish collectively searching for food sources. Although the MOPSO algorithm works with a population it is not an evolutionary algorithm. The population is not altered with crossover nor with mutation operators. There is no selection operator either; all individuals are kept in the population at all time. Even more important is that individuals keep a memory of their past positions and that their moves are directed.

The population of a Particle Swarm Optimizer (PSO) consists of particles that are initially placed randomly on the search space. Each particle has a direction and speed. Each iteration of the algorithm the speed and position of a particle are updated. The speed and direction of each particle are influenced by its own local optimum, a global optimum and a random factor.

2.1.1 The multi-objective variant

The multiobjective variant of the PSO, the MOPSO, is different in a few ways. Because we deal with multiple objectives the result will be a Pareto front with particles instead of a single best solution. With each iteration of the algorithm, all non-dominated points of the particles history are stored in a repository. This is the global "memory" of the particle. Another, very logical, difference is that particles are moved in n objectives instead of just one.

2.1.2 Pseudo code

Next, let us outline the pseudocode of a canonical MOPSO algorithm (cf. Coello Coello [3]).

```
1. Randomly initialize population $\overrightarrow{POP}$ of size $\mu$.
2. Initialize the speed $\overrightarrow{VEL}$ of each particle.
       $\forall i \in \{1, ..., \mu\} \quad \overrightarrow{VEL}[i] = 0$.
3. Evaluate each particle in $\overrightarrow{POP}$.
4. Store non-dominated points in the repository $\overrightarrow{REP}$.
5. Generate hypercubes (cubes in $m$ dimensions, where $m$ is the number of objectives) of the objective space explored so far. Do this by dividing the objective space explored by *numDivisions* in each dimension of it.
6. For each particle, determine in which hypercube it is positioned.
7. Initialize the memory $\overrightarrow{PBESTS}$ of each particle.
       $\forall i \in \{1, ..., \mu\} \quad \overrightarrow{PBESTS}[i] = \overrightarrow{POP}[i]$.
8. while the maximum number of loops is not exceeded, do:
     • compute the speed of each particle $i$ (in each direction) with the formula $\overrightarrow{VEL}[i] = W \times \overrightarrow{VEL}[i] + R_1 \times (\overrightarrow{PBESTS}[i] - \overrightarrow{POP}[i]) + R_2 \times (\overrightarrow{REP}[h] - \overrightarrow{POP}[i])$
           Where $W$ is an inertia weight of 0.4, $R_1$ and $R_2$ are random values between 0 and 1.
     • compute the new position of each particle with $\overrightarrow{POP}[i] = \overrightarrow{POP}[i] + \overrightarrow{VEL}[i]$.
     • Make sure each particle stays within the search space boundaries.
```

- update $\overrightarrow{REP}$, $\overrightarrow{PBESTS}$, the hypercubes and the position of each particle within these hypercubes.

The index for the global optimum, h, is randomly selected. The "fitness" of each hypercube is computed by dividing a number (in this thesis 1) by the number of particles in that hypercube. This fitness is then used to apply roulette-wheel selection. This selects a hypercube from which a particle is randomly picked. This selection should make sure that hypercubes with less particles have a higher probability to be selected. This way there is some disadvantage for highly-populated hypercubes. Particles will be attracted to the lesser populated parts of the Pareto front, which is a good thing as we want it to be well-spread.

There is one thing that is implemented in a way not proposed in [3], the size of the repository. We did not limit the amount of particles in this repository. A very large repository might be undesirable because it would make computing the Pareto front costly. In this study this is not a problem. One run of the Vabi simulator takes approximately 14 seconds. It is clear that this will be the delaying factor in our tests.

Furthermore, we do not intend to make a great number of function evaluations. Therefore we do not have to worry about too much individuals in the repository, most probably we get less then we want. It is because of these reasons that we did not put a limit on the number of individuals in the repository.

2.2 Advantages of using the MOPSO algorithm

The MOPSO algorithm is simple but it is reported to perform well [3]. The main problem in (real-world) optimization problems is the time it takes to run the algorithm. So, reducing the number of function evaluations is one of the best ways to make our solution applicable in practice. Marijt [1] reduced the amount of function evaluations by using a meta-model. This way he estimated the objective function values of his inputs and therefore needed less real evaluations.

MOPSO uses directional information and is therefore fast in approximating the Pareto front. This makes it especially suitable for this project. Still, the question is whether or not it makes good approximations using a small number of evaluations.

Another reason for using a MOPSO was that there was no MATLAB implementation at LIACS. This thesis will make such an implementation available for further research plus a function handler for the Vabi simulator. Besides that it will provide some first empirical results of this implementation on test problems.

2.3 Implementation

The MOPSO build for this thesis is based on the description in [3]. It is written in MATLAB and was built to accommodate function handling for multiple objective functions that can be easily exchanged. The function handles make sure all sorts of objective functions can be used, even external evaluation tools. The main function is called with 4 function handles. The first handle is for initializing all variables and the population. The second is for moving the particles, the third one is the objective function itself.

This setup of the function is not new, however very important. To ensure persistance of the software project it is important that the function handles are easy to use and to be reconfigured. That is why the approach with function handles was chosen. The documented source code can be found in appendix A of this thesis.

3 Evaluating the implementation

Before the MOPSO can be used to optimize with the building simulator, the proper functioning of the implemented algorithm should be validated on test problems with a known Pareto front. Therefore it was tested on the Supersphere testfunction proposed by Emmerich and Deutz [6]. Testing on this test function must prove that a good Pareto front is found and, more importantly, that it is found consistently.

Knowing how long it takes to converge is also very important. As mentioned before, the number of function evaluations must be kept to a minimum due to costly evaluations of the building simulator.

3.1 Parameters

A few parameters are essential for the MOPSO algorithm to perform well on a problem. First of all, the *number of objective function evaluations*. This parameter determines the quality of the results found by the algorithm since it determines how many computations can be done. It is debatable if this parameter is a control parameter or a given constant since we want to know how the MOPSO performes with only few objective function evaluations.

Also, the *number of divisions* we make in the objective space is an important parameter. This is used to divide the explored search space up in hypercubes, and therefore determines the size of the hypercubes. This size of the hypercubes is important when we select the repository index *h*.

The *inertia weight W* is the last parameter that we will tune in order to get the best results on the simulator. The *inertia weight* determines to what extend the old velocity of a particle is considered while calculating the new velocity. A percentage of the old velocity, depending on the *inertia weight*, can be added to the new velocity. With a low *inertia weight* the particles can adjust there "flight path" more quickly while a large *inertia weight* makes it more difficult to change course.

These are the parameters with which we will try to optimize the algorithm. There are, however, some others. There is, for example, the *number of particles* in the population. Also, the two random values used to move the particles around can be changed. Although these are all important parameters, they will not be changed, and instead use the default values introduces in section 2.1.2.

The reason for this decision is simple. The number of evaluations can be considered a static value. Taking a maximum amount of 400 should be enough to find good results. This proved to be sufficient in the tests of Marijt [1]. A second important factor is the time it takes to run a test. A test with 400 evaluations with the Vabi simulator takes approximately 23 hours. More evaluations will mean longer tests and building engineers usually do not want to wait that long for a result. We should keep an eye on the usefulness of this algorithm when applied in the real world. Therefore 400 was chosen as an upper bound for the number of evaluations.

The *number of divisions in the search space* is the second parameter to be tested. This parameter is important to ensure that the Pareto front will be spread evenly along all non-dominated solutions. Recall that the hypercubes are used to select a guiding particle from the repository to which a particle moves. A right setting of this parameter will make hypercubes of the right size. This means that whenever they are too big, all hypercubes will be full of solutions. Hypercubes that are too small are also unfavorable. In this case almost all hypercubes will contain zero or just a few particles. A good setting will make sure that some hypercubes are full of particles and some others will just contain one or two.

The *inertia weight* has it influence on the direction and speed of each particle. If the inertia weight would be 0.0 then each particle would be flying towards its own and towards a global optimum. This way there would not be any influence of previous velocities on the new one. The other extreme, a value of 1.0 would mean that all velocities are stacked upon each other. This could lead to very high velocities and uncontrolable behaviour. We will be testing a range of values between 0 and 1 to test what works best.

There are multiple reasons why the random- and *size of population* parameters have not been tested. First of all it would take a lot of tests and would exceed the scope of this bachelor thesis. A second reason is that the number of particles in the repository, also influenced by the *number of evaluations*, and the *number of divisions* strongly depend on each other. We have chosen to take those apart and keep the rest constant. Furthermore this setup worked good, there was not a real necessity to tune more parameters.

Coello Coello [3] proposed to use the MOPSO with 20-80 particles, 80-120 rounds, 30-50 divisions and an inertia weight of 0.4. These settings were tested on the superspheres problem. The results will indicate whether or not the implementation that was made works properly. Furthermore it will indicate how the parameters should be set. Once a set of good parameters is found, the algorithm can be deployed on the Vabi building simulator.

3.2 Setup of the tests

To make sure we do not make too much evaluations we keep the particles at 20 and the number of rounds, too. This will lead to 400 objective function evaluations [1], the exact number used in [1]. The expectation is that a higher *number of evaluations* will find a better Pareto front. However, the question is if the Pareto front approximation found after 400 evaluations is already good enough.

With an almost static amount of evaluations, we can focus on fine tuning the *number of divisions* and the *inertia weight*. We try to optimize the combination of these parameters so we will first optimize the first parameter and then the other, resulting in the following tests that can be found in table 1.

[1]A function evaluation here is assumed to always compute the full vector of objective function values

3.3 Results

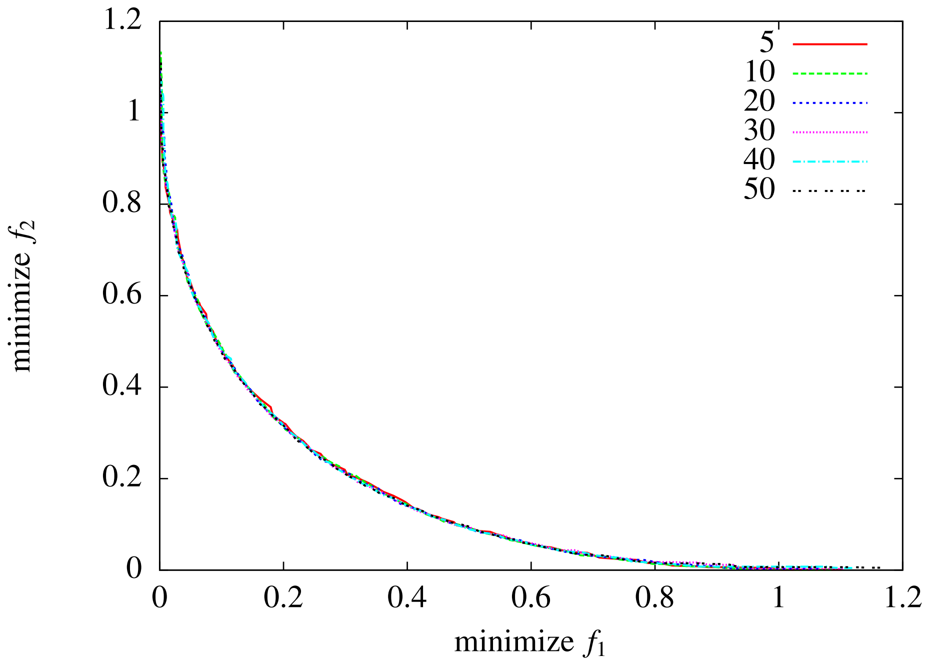
The result of the MOPSO implementation on the superspheres testfunction will be shown below. The most important measure of success will be the hypervolume measure proposed in [6]. This hypervolume measure is essentially the size of the surface that is dominated by the Pareto front. As a reference point [1.5, 1.5] was used because this point is close to the points of the Pareto front approximations but always dominated.

The next measurement is the Summary Attainment Surface (SAS) plot proposed by Fonseca and Fleming [7], plotted using the tool made by Knowles [8]. This is a way to average different Pareto fronts. It gives a good insight into what is dominated by a front and makes it easy to compare fronts from different runs.

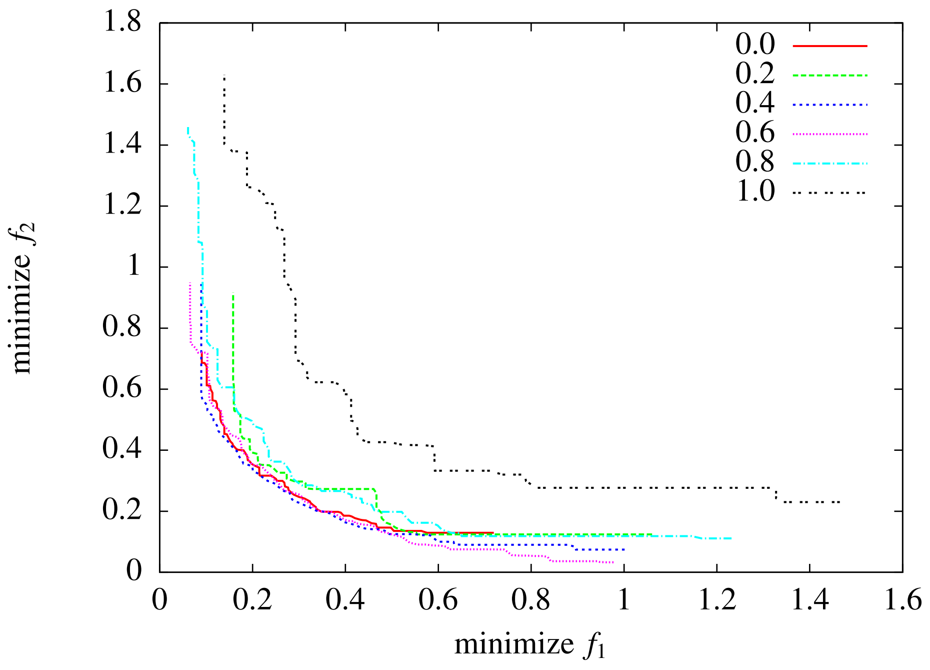
As we can see in figure 3(a) the MOPSO performes really well on the superspheres problem. So well actually that it is hard to see what setting of the *number*

20 particles	20 rounds	0.4 inertia weight	5 divisions
20 particles	20 rounds	0.4 inertia weight	10 divisions
20 particles	20 rounds	0.4 inertia weight	20 divisions
20 particles	20 rounds	0.4 inertia weight	30 divisions
20 particles	20 rounds	0.4 inertia weight	40 divisions
20 particles	20 rounds	0.0 inertia weight	best so far
20 particles	20 rounds	0.1 inertia weight	best so far
20 particles	20 rounds	0.2 inertia weight	best so far
20 particles	20 rounds	0.3 inertia weight	best so far
20 particles	20 rounds	0.4 inertia weight	best so far
20 particles	20 rounds	0.5 inertia weight	best so far
20 particles	20 rounds	0.6 inertia weight	best so far
20 particles	20 rounds	0.7 inertia weight	best so far
20 particles	20 rounds	0.8 inertia weight	best so far
20 particles	20 rounds	0.9 inertia weight	best so far
20 particles	20 rounds	1.0 inertia weight	best so far
20 particles	40 rounds	best so far	best so far

Table 1: Setup of all tests that will be done to evaluate the implementation of the MOPSO.



(a) Median attainment surface plots with different *number of divisions* on the Superspheres problem.



(b) Median attainment surface plots with different inertia weights on the Superspheres problem.

Figure 3: Different plots on the Superspheres problem.

particles	rounds	inertia weight	divisions	hypervolume
20	**20**	**0.4**	**5**	**2.1335**
20	20	0.4	10	2.0566
20	20	0.4	20	2.0518
20	20	0.4	30	2.0342
20	20	0.4	40	2.0654
20	20	0.4	50	2.0056

Table 2: Average hypervolume for each run with the superspheres function optimizing the *number of divisions*.

particles	rounds	divisions	inertia weight	hypervolume
20	20	5	0.0	1.7730
20	20	5	0.2	1.6342
20	20	5	0.4	2.0183
20	**20**	**5**	**0.6**	**2.0954**
20	20	5	0.8	1.6665
20	20	5	1.0	1.5173

Table 3: Average hypervolume for each run with the superspheres function optimizing the *inertia weight*.

of divisions works best. The hypervolume of each run, given in table 2 gives us an answer but also shows that the differences are minimal. Setting the *number of divisions* to 5 gives the best results, but all other settings also provide us with good results.

The results for the tuning of the *inertia weight* is shown in figure 3(b). A good value for the superspheres function is 0.4 or 0.6. As with the tuning of the *number of divisions* all results are very close together.

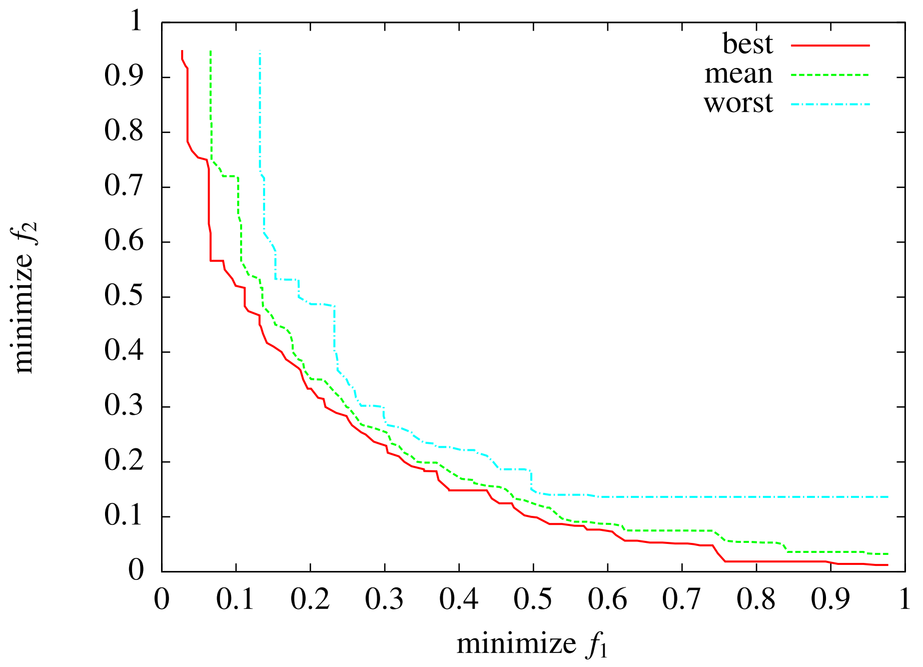
Figure 4(a) displays different SAS plots of the best run we had with the Superspheres function, with a *number of divisions* of 5 and an *inertia weight* of 0.6. What we can see are a worstcase, an average and a bestcase SAS plot. The differences between these plots are not very big which means our algorithm is consistently finding good Pareto front approximations.

The development of the hypervolume over time is plotted in figure 4(b). We can see that hypervolume initially grows very fast and than stagnates. This is what we could expect since MOPSO uses directional information to find good solutions. The convergence is fast and the established hypervolume is good.

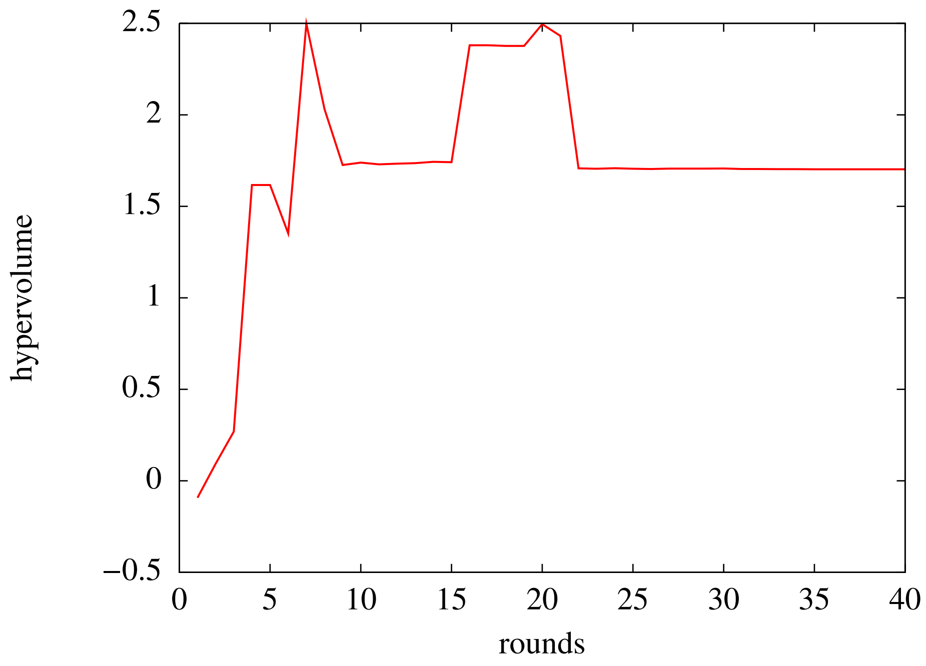
There are however a few inconsistencies in the plot where the hypervolume declines again. This can easily be explained. Since new points are selected for the Pareto front on Pareto-dominance alone, this could mean these points do not necessarily improve the hypervolume.

This result is very good. Most improvements of the hypervolume, and therefore on the Pareto front, are done within the first 400 objective function evaluations. After this the front still changes but it mostly gets denser instead of really better. This is exactly what we want because it means that even with as little as 400 evaluations a good approximation of the Pareto front is found.

As an additional test we checked if the MOPSO could converge to Pareto frontsof different shapes. To do this we had to change the Superspheres function. Changing the α parameter changes the shape of the Pareto front so that it becomes convex or straight instead of concave like it is ussualy. The test was done with only 400 function evaluations and 5 divisions.



(a) Different SAS plots for the best run on the Supersphere function, with a *number of divisions* of 5 and an *inertia weight* of 0.6.



(b) Hypervolume development for the Superspheres test function.

Figure 4: Different plots of the optimizations on the Superspheres function.

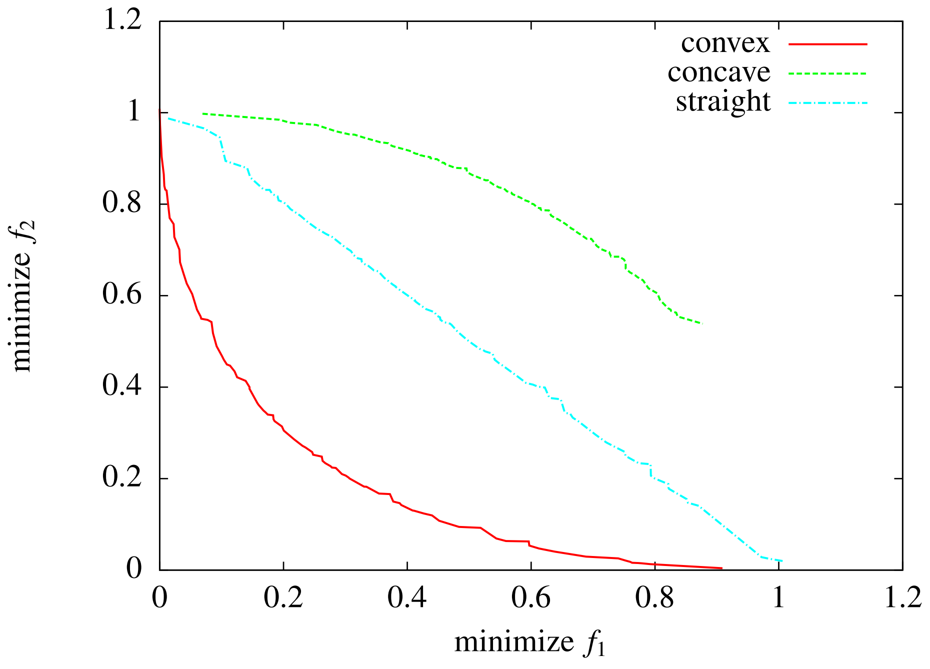


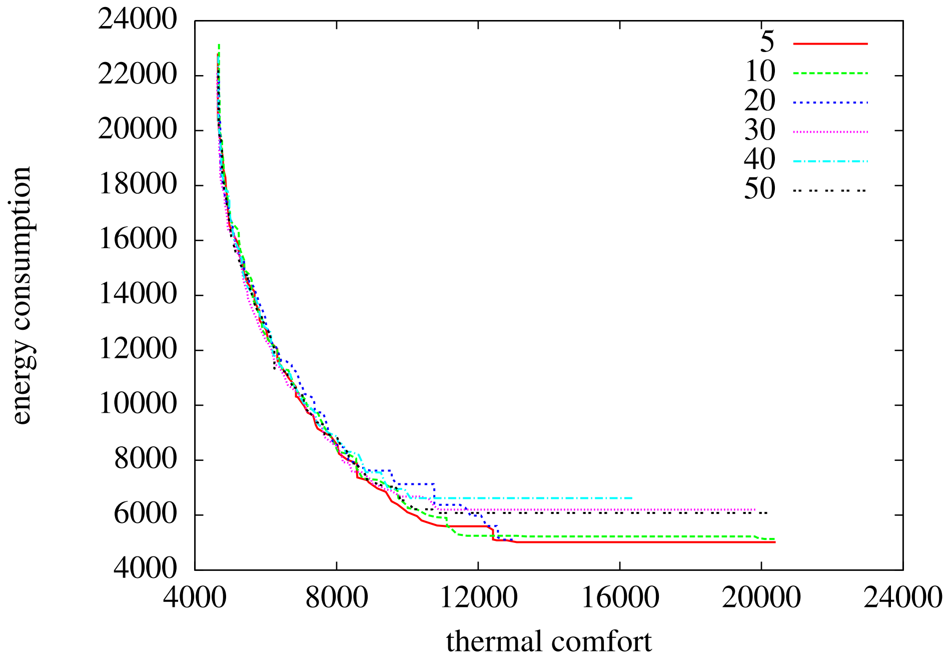
Figure 5: Results on differently shaped pareto fronts of the superspheres problem.

What we see in figure 5 is that there is a good approximation for almost all types of front. The concave front is not approximated very well, this however is understandable since this shape is most difficult to find. Overall this result is very good.

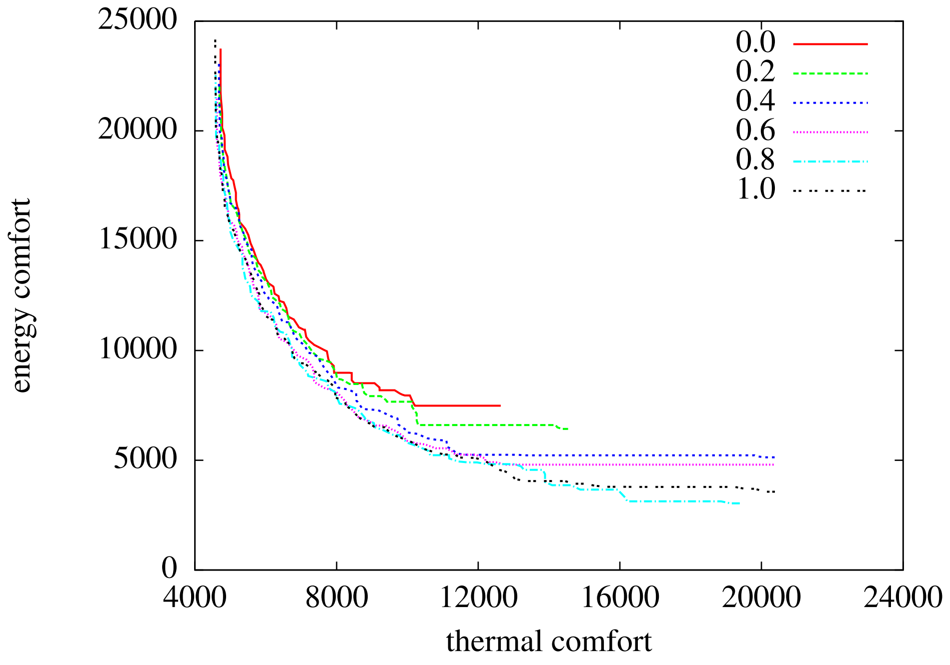
4 Tests on the Vabi building simulator

As allready said, the Vabi building simulator is our real world problem we try to optimize. And, as we also mentioned before, this simulator works like a black box. With some procedures we fill the input files that Vabi uses. Then we run the simulator using the system command of matlab. When the simulation is finished we read the output files and gather the variables that form our objective values.

We can see in figure 6(a) that the results on the Vabi building simulator are different. This figure shows the different fronts found by the MOPSO with different *numbers of divisions* over 400 function evaluations. The difference between the



(a) Median attainment surface plots with different number of divisions for the Vabi simulator.



(b) Median attainment surface plots with different inertia weights for the Vabi simulator.

Figure 6: Different plots of the optimizations on the Superspheres function.

particles	rounds	inertia weight	divisions	hypervolume
20	20	0.4	5	220.221.920
20	**20**	**0.4**	**10**	**237.740.142**
20	20	0.4	20	158.467.624
20	20	0.4	30	181.003.258
20	20	0.4	40	154.162.660
20	20	0.4	50	195.093.559

Table 4: Average hypervolume for each run with the Vabi building simulator.

particles	rounds	divisions	inertia weight	hypervolume
20	20	10	0.0	133.239.557
20	20	10	0.2	147.614.801
20	20	10	0.4	237.740.142
20	20	10	0.6	223.622.766
20	20	10	0.8	283.637.382
20	**20**	**10**	**1.0**	**286.070.400**

Table 5: Average hypervolume for each run with different inertia weights on the Vabi building simulator.

different settings are bigger than the ones on the Superspheres function. A value of 10 seems to working best allthough the difference with a value of 5 is not very big.

Table 4 shows that 5 and 10 divisions give the best results, but besides that it also shows that the hypervolume achieved with 20-50 divisions are very irregular. This could be caused by the fact that some values really work better than others. It could also be that this is caused by a few very good or very bad runs. Due to time limitations we have only been able to test each setup 9 times. More tests are needed before we can really conclude which of these values works best. Still, the hypervolumes achieved with 5 and 10 divisions are definitely better than the others.

Only 10 divisions seems to be a low value. Especially when we compare it to the suggested amounts of divisions that Coello Coello proposed: 30-50. There is however an explanation for this behaviour: the *number of divisions* and the total amount of particles in the repository are strongly correlated. Since we do not have large amounts of particles, due to the limited number of function evaluations, we get best results with a lower value. This value, in this particular case, yields the best distribution of particles over the hypercubes.

Another interesting thing we can see in figure 6(a) is that the differences between the runs are made on just a specific part of the Pareto front. At the left top of the plot all lines are very close together while the differences on the other hand of the front are biggest. This suggests there are some areas which are easier to optimize than others.

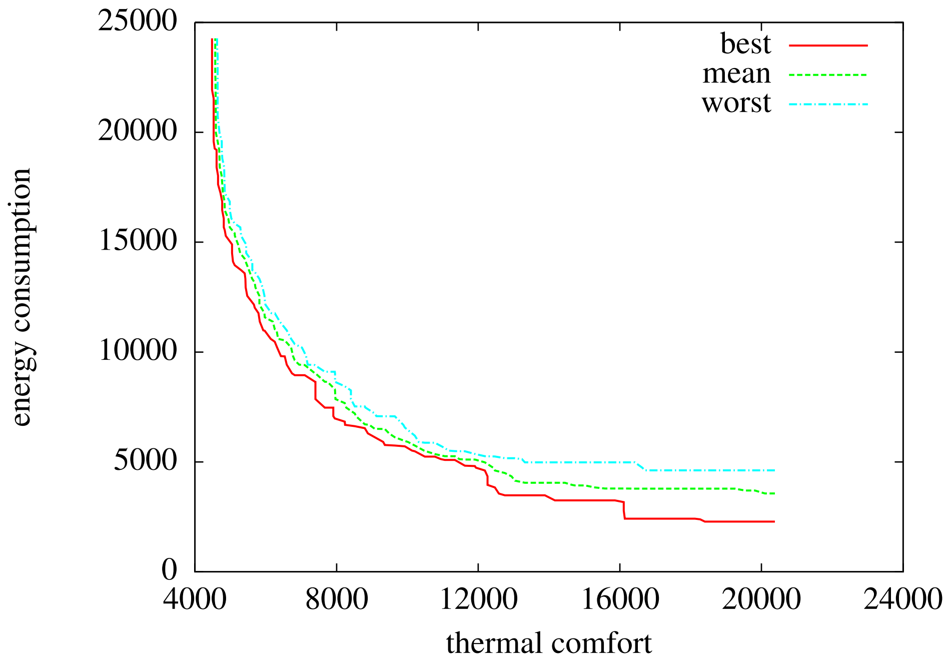
In figure 6(b) the different runs with the *inertia weight* on the Vabi simulator are plotted. As we can see the differences between these runs are bigger than the ones tuning the *number of divisions*. As we can also see in table 5 a higher value for the *inertia weight* results in a bigger hypervolume. The highest value of 1.0 yields the best results.

Just like in figure 6(a) the big differences are all concentrated on one part of the Pareto front approximation, which is the same part as they were in figure 6(a). This makes it even more likely that this part of the front is harder to optimize.

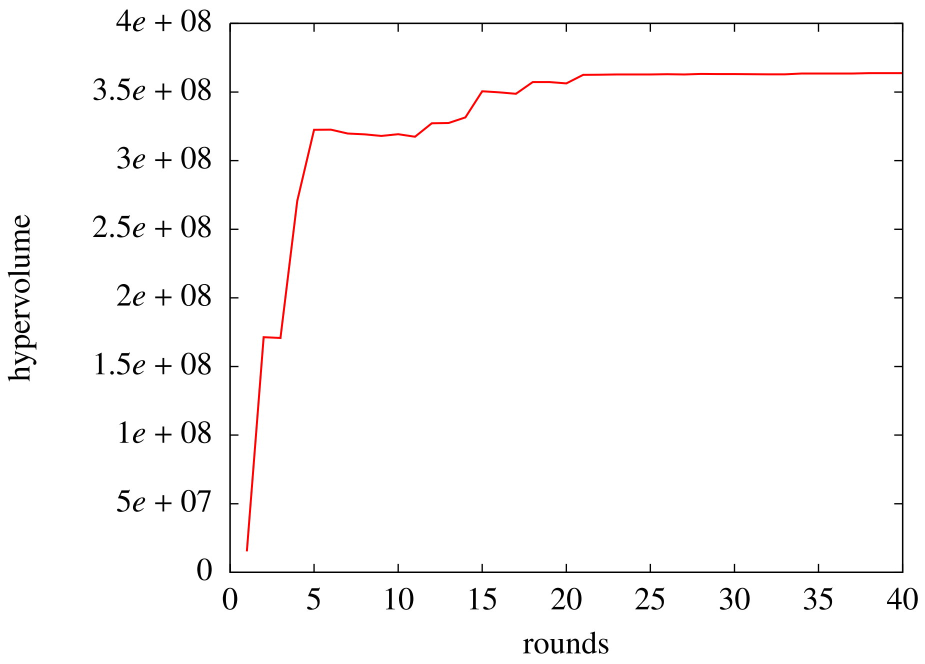
Figure 7(a) presents the SAS plots of the best run on the Vabi simulator. The overall shape of the lines looks like a Pareto front approximation. Also, the consistency of the fronts seems to be very good. Even the worstcase SAS plot stretches out over the full length of the objective space. The differences between them is namely in the points they dominate, the shape of all fronts is good.

The development of the hypervolume on the Vabi problem is presented in figure 7(b). As we can see the hypervolume grows rapidly and then somewhat stagnates. The variations in the hypervolume after the initial growth are caused by changes in the Pareto front. These variations in the hypervolume are normal and the overall look the plot is good. The largest optimizations are done within the first 20 rounds allthough this will probably not always be the case.

The most important information we can extract from figure 7(b) is that the hypervolume does not grow dramatically after 400 function evaluations, most of the progress is done beforehand. This is good because it points out that we have fast convergence. On the other hand we can see that the improvements that are achieved decrease. This means that, even though we run the algorithm longer, we do not automatically get a better result. This can either be caused by a lack of power of the algorithm, difficult local optima in the objective space or the fact that we allready found the global optimum.



(a) Summary attainment surface plots for the best run on the Vabi simulator, with a *number of divisions* of 10 and an *inertia weight* of 1.0.



(b) Hypervolume development for the Vabi simulator.

Figure 7: Different plots of the optimizations on the Superspheres function.

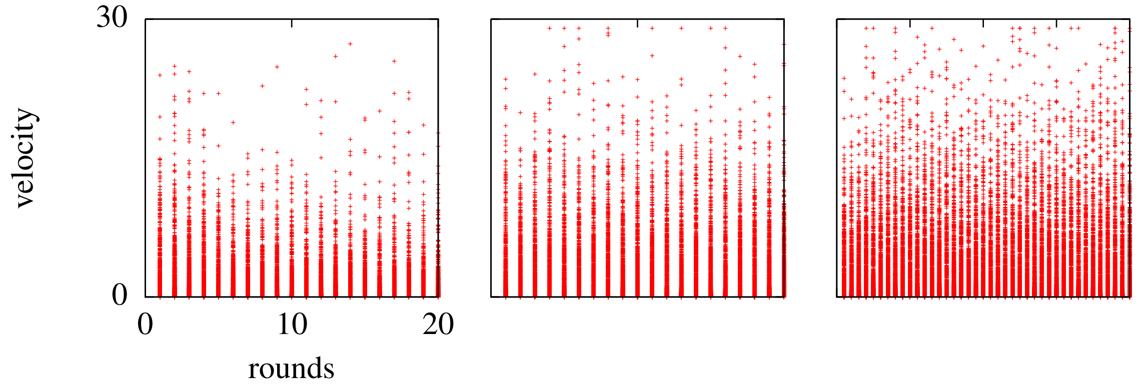


Figure 8: Velocity development for three Vabi runs.

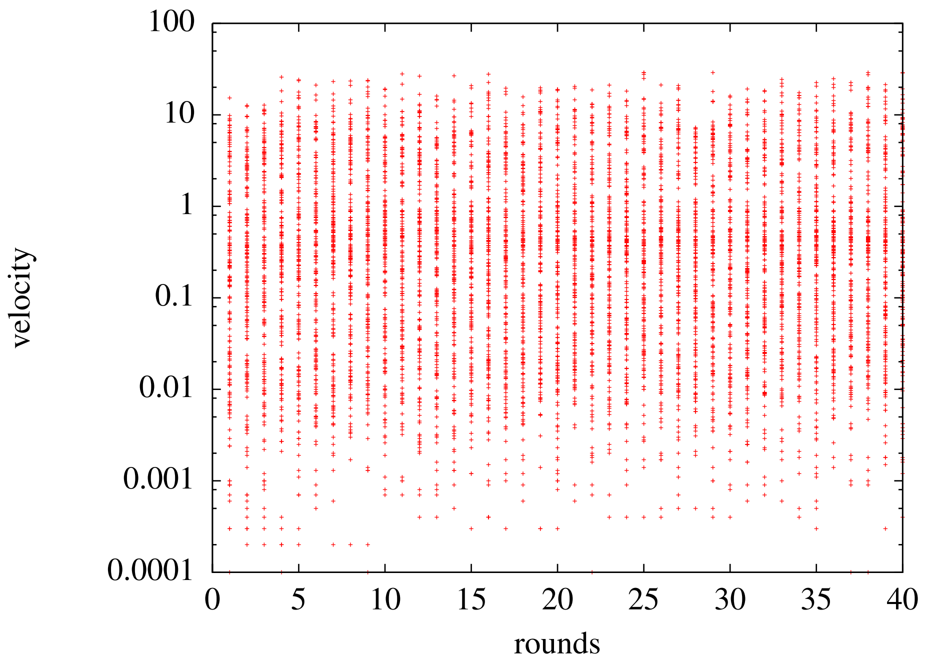


Figure 9: Velocity development for one Vabi runs on logaritmic scale.

As our last result on the Vabi simulator we have included a plot of the velocities of all particles during a run, which is shown in figure 8. The most left plot is of our worst run during the tuning of the *inertia weight*, so with an *inertia weight* of 0.0. The plot in the middle is of the best run, with an *inertia weight* of 1.0. The rightmost plot is of the long run which was also done with an *inertia weight* of

1.0. This last run lasted 40 rounds and was also used to show the hypervolume development in figure 7(b).

What we can clearly see in figure 7(b) is that a higher *inertia weight* results in a higher average velocity. The velocities are concentrated near the very small values with some occasional outliers. What is interesting is that the velocities do not seem to "grow" as we can also see in figure 9 which is plotted using a logarithmic scale. One could expect the velocities to quickly grow and get out of proportion, apparently this is not the case here. The relatively small ranges that the variables have to move make sure that a high *inertia weight* does not give this problem.

4.1 Comparison with Marijt

This research project was preceded by Marijt [1] who implemented a different algorithm to optimize the Vabi building simulator in a robust way. Even though Marijt used a robust way of optimizing the simulator, it is still interesting to see how our results compare because different optimization algorithms were used. Marijt worked with the SMS-EMOA algorithm [2] which uses a totally different approach than our MOPSO which uses directional information.

Figure 10 shows 2 different Pareto front approximations. One is the median attainment surface, also shown in figure 7(a), found in the best run using the MOPSO. The other one is the best front found by Marijt in his studie. The results look very promising and indicate that the MOPSO yields good results. There are a few differences between the 2 fronts but these can be explained.

Firstly the MOPSO achieved an overall better front. This can be expected since the robustness in Marijt's results will lead to a more conservative choice of solutions. Points that are on the MOPSO front will probably be less robust since we did not test them on robustness.

Secondly, the front achieved by the MOPSO stretches much longer than the front of Marijt. This too can be explained by the robustness in Marijt's results. The bottom side of the objective space is probably less robust since the robust front does not cover this part. This idea is strengthened by the observations we made earlier. As we noticed in figures 6(a) and 6(b) the lower part of the objective space seems to be more difficult to optimize. This could explain the fact that the robust front did not find any solutions in that part of the objective space.

Lastly, the difference in the number of solutions in the front can be explained by the differences between the algorithms. The MOPSO stores all non-dominated solutions in a repository while the SMS-EMOA algorithm is an evolutionary algorithm with a fixed population size and therefor a maximum number of solutions.

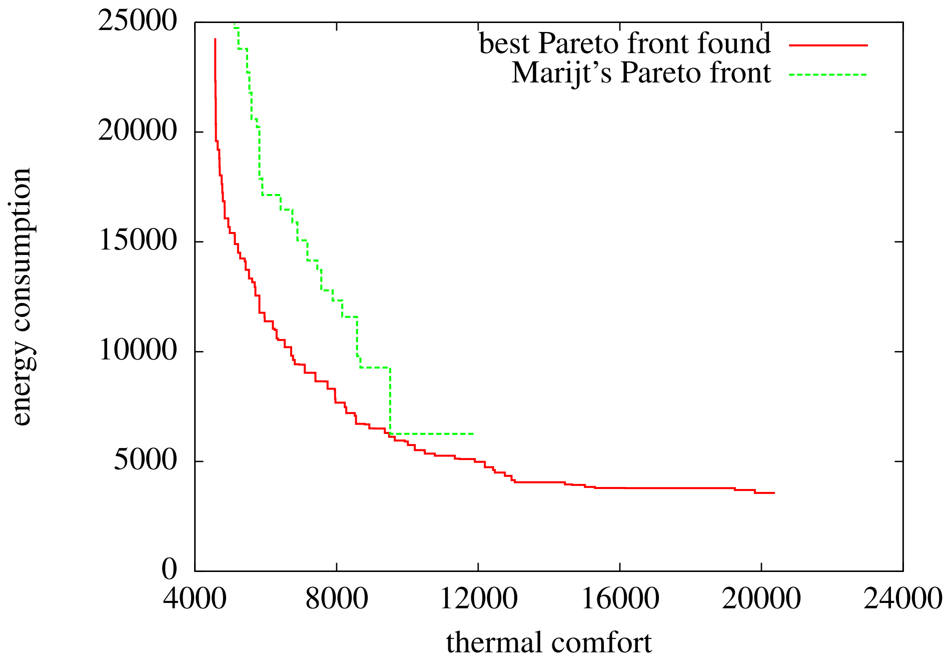


Figure 10: Comparement of the best Pareto front found with the MOPSO and one of Marijt's fronts.

5 Conclusions

5.1 MOPSO

The results presented in the last 2 chapters indicate that the implementation of the MOPSO algorithm works. Using the Superspheres function we showed that the algorithm finds optima in a fast and consistent way. It finds different Pareto front shapes, although the concave shaped Pareto front proved to be much harder to find.

We tried to find optimal values for the *number of divisions* and the *inertia weight* parameters. The optimal values are, for this particular function, 5 *divisions* and an *inertia weight* of 0.6. Especially the *inertia weight* makes an impact on the results, the importance of the number of divisions is almost none. This can however be caused by the fact that the Superspheres function is relatively easy to optimize. This makes the influence of the parameters minor, as good results are found even with suboptimal settings.

5.2 Vabi building simulator

The MOPSO algorithm performed well on the Vabi building simulator, especially considered its simple setup. Pareto front approximations seem to be found cinsistently and the results can be improved dramatically with some simple parameter tuning.

Number of divisions The *number of divisions* parameter was tuned first and the best value for the Vabi objective function was 10. This small value (compared to the value advised by Coello Coello) can be related to the small number of particles in the population. The right setting for the number of divisions not only increases the hypervolume that is achieved, more importantly it also yields Pareto front approximations that are more spread across the objective space. This is just what we expected it to do.

Inertia weight The *inertia weight* has an even bigger impact on the results. This parameter, set to the correct value of 1.0, does also influence the spread of the Pareto front. More importantly, it dramatically increases the hypervolume. The value of 1.0 would at first seem very unlikely since it makes all velocities stack upon each other, ever increasing the speed of each particle. Why this value works best can be caused by the fact that a faster particles can "travel" faster and therefor achieve better results in less evaluations. Another explanation may be that the high speed enables particles to jump over local optima and thus gain more progress.

On might expect that a high *inertia weight* would yield higher and higher velocities, which could seriously affect the functioning of the algorithm. However, the bounds of the input parameters are quite small and therefor there is not much room for developing too high velocities.

We can conclude that the MOPSO yields good results on the Vabi building simulator and that our parameter tuning really improved the results. Also, the 400 evaluations that we set as a maximum do not really form a limitation to what the algorithm achieves, which is very good news. All in all the MOPSO algorithm has very fast and fairly good convergence.

These results are very promising. It shows that algorithms that use directional information, like the MOPSO algorithm, can optimize building design problems. Up till now only evolutionary algorithms, like SMS-EMOA and NSGA-II were tested on the Vabi building simulator. This results is very good since it gives us more alternatives when searching for good optimizations.

5.3 Further research

As an extend to this thesis, more tests could be performed. Due to time limitations we have only been able to run 9 tests with each setup. This is good enough for our first results but more tests are needed to make sure our results are not based on some unpredictable inconsistencies in the results.

Furthermore, to be able to compare our results with Marijt's it would be best to test our results on robustness too. This way we could determine if our solutions are indeed less robust than Marijt's, as we think now. This would confirm our idea that Pareto front would be more like Marijt's if we would have considered robustness too.

Lastly, more algorithms could be tested on the Vabi simulator to discover what type of algorithms are capable of optimizing this problem. To be able to compare these tests it would be better to test them in the same framework. Making a non-robust SMS-EMOA or a NSGA-II implementation to run the same tests as in this thesis could be a good project.

A Source Code

A.1 Index and explanation

This appendix contains the MATLAB source code for the MOPSO algorithm that is implemented for this thesis. Each section represents a different MATLAB file with each containing one function, as is usual for MATLAB.

mopso.m

```
function mopso(fh_initialise, fh_move_particles, fh_evaluate)
```

```
global nParticles;          %nr of particles (population size)
global nMaxRounds;          %number of rounds to loop through the algorithm
global aParticles;          %current particles (population)
global aVelocity;           %speed of current particles
global aObjectiveValues;    %objective values of current particles
global aCoordinates;        %hypercube coordinates of current particles
global aRepositoryX;        %inputs of current non-dominated particles
global aRepositoryY;        %outputs of current non-dominated particles
global aPbestsX;            %personal best location of each particle
global aPbestsY;            %personal best objective values of each particle
global nCounter;            %count the current round of the algorithm,
                            %used for reading vabi results

%
%
%

%initialise different variables like nDimensions, upper- and lowerBounds
%and offcourse the population
fh_initialise();

%initialise speed
aVelocity = zeros(size(aParticles));
%evaluate particles
aObjectiveValues = fh_evaluate();

%compute non-dominated set (repositoryX and repositoryY)
[aRepositoryX, aRepositoryY] = nonDominatedSet(aParticles, aObjectiveValues); %x and y

%compute hypercube-coordinates for each particle
aCoordinates = getHypercubeCoordinates(aRepositoryY);

%initialise aPbests
aPbestsX = aParticles;
aPbestsY = aObjectiveValues;

%begin execution
for nCounter=1:nMaxRounds

    %compute speed of each particle
    %compute new position of particle
    %enforce lower and upper bound
    aParticles = fh_move_particles();

    %evaluate population
    aObjectiveValues = fh_evaluate();

    %compute non-dominated values in (aObjectiveValues + aRepository)
    %update aRepositoryX and aRepositoryY
    [aRepositoryX, aRepositoryY] = ...
    nonDominatedSet([aParticles; aRepositoryX], [aObjectiveValues; aRepositoryY]);

    %compute hypervolume-measure of current non-dominated particles
    nHypervolume = computeHypervolume(aRepositoryY); %#ok<NASGU>

    if (nCounter ~= nMaxRounds)
        %compute hypercube-coordinates for each particle
        aCoordinates = getHypercubeCoordinates(aRepositoryY);

        %update aPbestsX and aPbestsY
        for i=1:size(aParticles,1)
            if dominates(aObjectiveValues(i,:), aPbestsY(i,:))
```

```
                aPbestsY(i,:) = aObjectiveValues(i,:);
                aPbestsX(i,:) = aParticles(i,:);
            end
        end
    end

    end

    %done
    display('done.');

end
```

initialiseVabi.m

```
function initialiseVabi()

    global aParticles;          %current particles (population)
    global MU;                  %mean of all but sparam parameters
    global S;                   %standard deviation of all but sparam parameters
    global aLowerBounds;        %lower bounds for parameter 73-77
    global aUpperBounds;        %upper bounds for parameter 73-77
    global aIndexes;            %boolean array with sparam-parameters as 1
                                %and the rest as 0
    global aVariables;          %names of all parameters
    global sparam;              %names of parameters that we adjust
    global vabidir;             %path of vabi simulator
    global nParticles;          %nr of particles (population size)
    global nMaxRounds;          %number of rounds to loop through the algorithm
    global nObjectives;         %nr of objectives
    global nDivisions;          %nr of divisions used to compute hypercubes
    global aReferencePoint;     %used to compute hypervolume
    global nCounter;            %count the current round of the algorithm,
                                %used for reading vabi results

    %
    %
    %

    %fill config settings
    nParticles =     2;
    nMaxRounds =     10;
    nObjectives =    2;
    nDivisions =     10;
    aReferencePoint = [100000 100000];
    nCounter = 0;

    vabidir = 'C:\VABI_UO\VA114\progab\uogev\';

    aVariables = {'Twall1','Twall2','Twall3','Tfloor1','Tfloor2','Tfloor3','Tfloor4',...
                  'Tfloor5','Tfloor6','Troof1','Troof2','Troof3','Troof4','Troof5',...
                  'Cwall1','Cwall2','Cwall3','Cfloor1','Cfloor2','Cfloor3','Cfloor4',...
                  'Cfloor5','Cfloor6','Croof1','Croof2','Croof3','Croof4','Croof5',...
                  'Dwall1','Dwall2','Dwall3','Dfloor1','Dfloor2','Dfloor3','Dfloor4',...
                  'Dfloor5','Dfloor6','Droof1','Droof2','Droof3','Droof4','Droof5',...
                  'SHCwall1','SHCwall2','SHCwall3','SHCfloor1','SHCfloor2','SHCfloor3',...
                  'SHCfloor4','SHCfloor5','SHCfloor6','SHCroof1','SHCroof2','SHCroof3',...
                  'SHCroof4','SHCroof5','Sawall','Iewall','Oewall','Uenkel','Udouble',...
                  'infiltration','Saroof','Ieroof','Oeroof','Safloor','Iefloor',...
```

```
              'Oefloor','Saglass','Ieglass','Oeglass','glasswindow','Geometry',...
              'Height','Persons', 'Equipment','Lighting'};

% The order of the unindented variables below also hold for MU and S

% Twall1 Twall2 Twall3
% Tfloor1 Tfloor2 Tfloor3 Tfloor4 Tfloor5 Tfloor6
% Troof1 Troof2 Troof3 Troof4 Troof5
% Cwall1 Cwall2 Cwall3
% Cfloor1 Cfloor2 Cfloor3 Cfloor4 Cfloor5 Cfloor6
% Croof1 Croof2 Croof3 Croof4 Croof5
% Dwall1 Dwall2 Dwall3
% Dfloor1 Dfloor2 Dfloor3 Dfloor4 Dfloor5 Dfloor6
% Droof1 Droof2 Droof3 Droof4 Droof5
% SHCwall1 SHCwall2 SHCwall3
% SHCfloor1 SHCfloor2 SHCfloor3 SHCfloor4 SHCfloor5 SHCfloor6
% SHCroof1 SHCroof2 SHCroof3 SHCroof4 SHCroof5
% Sawall Iewall Oewall
    % XKos
% Uenkel Udouble
    % ThickEnkel ThickDouble
% infiltration
    % vermpersonen vermapparaten vermverlichting
    % glasssurface
% Saroof Ieroof Oeroof
% Safloor Iefloor Oefloor
% Saglass Ieglass Oeglass
    % glasswindow
% singledouble

% Mean of normal distributed variables.
MU = [0.005  0.127  0.2  ...
      0.8  0.28  0.1  0.0635  0.025 0.015  ...
      0.01  0.005  0.15  0.1345  0.019  ...
      50  0.04  1.41  ...
      1.41  0.84  1.13  0.025  0.15  0.06 ...
      0.96 0.5 1.13  0.04 0.056 ...
      7800 12 1900 ...
      1900 1700 2000 30 800 160 ...
      1800 1700 2000 12 380 ...
      480 840 1000 ...
      1000 800 1000 1400 2093 2500 ...
      1000 1000 1000 840 1000 ...
      0.6 0.9 0.9 ...
      5.1034 1.21 ...
      0.5 ...
      0.6 0.9 0.9 ...
      0.6 0.9 0.9 ...
      0.6 0.9 0.9];

% Standard deviation of normal distributed variables.
S = [0.0005  0.0127  0.02  ...
     0.08  0.028  0.01  0.00635  0.0025 0.0015  ...
     0.001  0.0005  0.015  0.01345  0.0019  ...
     0.75  0.0032  0.1269  ...
     0.4653  0.2772 0.1017 0.00875 0.025 0.0078 ...
     0.288 0.25 0.1017 0.0032 0.02436 ...
     25.74 1.08 28.5 ...
     332.5 297.5 30 21 25 18.4 ...
     228.6 493 30 1.08 102.6 ...
     19.2 56.28 106 ...
     107.5 86 106 378 134 0 ... % SHCfloor6 0
```

```
            195 330 106 56.28 108 ...
            0.006 0.0198 0.0198 ...
            0.255 0.0605 ...
            0.17 ...
            0.006 0.0198 0.0198 ...
            0.006 0.0198 0.0198 ...
            0.006 0.0198 0.0198];

        %define which variables that will be varied
        sparam = {'Twall3', 'Cfloor4', 'Croof2', 'infiltration', 'glasswindow','Geometry',...
                  'Height','Persons', 'Equipment','Lighting'};

        %lowerBound and upperBound for Geometry, Height, Persons, Equipment and Lighting
        aLowerBounds = [25.7661, 4.3050, 6, 6, 6];
        aUpperBounds = [26.7661, 6.3050, 25, 35, 20];

        %get boolean array which determines per attribute if it should be optimised or not
        %so, a 0 means the attribute is not changed and offcourse a 1 means that is does
        aIndexes = getindexes();

        %initialise particles
        aParticles = initPopulationVabi();

        %empty vabi output
        emptyoutput();

    end
```

getindexes.m

```
    function res = getindexes()

        global aVariables;          %names of all parameters
        global sparam;              %names of parameters that we adjust

        %
        %
        %

        %create array of zero's
        res = zeros(1,size(aVariables,2));

        %loop over variables
        for i=1:size(aVariables, 2)

            %loop over
            for j=1:size(sparam, 2),
                if strcmp(sparam(j), aVariables(i)) == 1,
                    res(i) = 1;
                    break;
                end
            end

        end

    end
```

initPopulationVabi.m

```
function res = initPopulationVabi()

    global nParticles;          %nr of particles (population size)
    global aVariables;          %names of all parameters
    global MU;                  %mean of all but sparam parameters
    global S;                   %standard deviation of all but sparam parameters
    global aIndexes;            %boolean array with sparam-parameters as 1
                                %and the rest as 0
    global aLowerBounds;        %lower bounds for parameter 73-77
    global aUpperBounds;        %upper bounds for parameter 73-77

    %
    %
    %

    %initialise empty array
    res = zeros(nParticles, size(aVariables,2));

    %produce a set of inputs, result of this function
    for i=1:nParticles
        for j=1:size(aVariables,2)

            %constant parameters, keep at mean value
            if ((j < 72) && (aIndexes(j) == 0))

                res(i,j) = MU(j);

            %adjusted variables
            elseif ((j < 72) && (aIndexes(j) == 1))

                %generate value from a normal distribution with mean A and
                %standard deviation B
                % = A + B.*randn(x,y);
                nLowerBound = MU(j) - (3*S(j));
                nUpperBound = MU(j) + (3*S(j));

                %make sure values stay within [nLowerBound, nUpperbound]
                %and always above 0
                bWithinBounds = false;
                while (~bWithinBounds)

                    res(i,j) = MU(j) + S(j) .* randn(1);
                    if ((res(i,j) >= nLowerBound) && ...
                        (res(i,j) <= nUpperBound) && (res(i,j) >= 0))
                        bWithinBounds = true;
                    end

                end

            %parameter glasswindow
            elseif (j == 72)

                %0 or 1
                res(i,j) = round(rand(1));

            %adjusted variables with own lower- and upper bounds
            elseif (j > 72)

                boundIndex = (j-72);
                res(i,j) = aLowerBounds(boundIndex) + ...
                           (aUpperBounds(boundIndex)-aLowerBounds(boundIndex)) * rand(1);

            end
        end
    end

end
```

emptyoutput.m

```
function emptyoutput()

    global vabidir;             %path of vabi simulator

    %
    %
    %

    fileNameSource = strcat(vabidir, 'va114gvu.par.N');
    fileNameTarget = strcat(vabidir, 'va114gvu.par');

    file_1 = fopen(fileNameSource,'w');

    header1 = 'Uitvoerfile va114gvu.par: gevoeligheden';
    header2 = 'IPAR IPARGEV  PARGEV         QCEVW   QCEKL   PCEVW  PCEKL IV   QLOVW...
               QLOKL PLOVW  PLOKL TLMAX TLMIN TLGEM TCMAX TCMIN OHL25 OHL28 OHC25...
               OHC28  WOH-  WOH+';
    header3 = '  -     -        -           kWh     kWh     kW     kW    -    kWh...
               kWh   kW     kW      C     C     C     C     C    h     h     h...
               h     h     h';
    fprintf(file_1,'%s\r\n%s\r\n%s\r\n',header1, header2, header3);

    fclose(file_1);
    copyfile(fileNameSource, fileNameTarget);

end
```

evaluateVabi.m

```
function res = evaluateVabi()

    global aParticles;          %current particles (population)
    global nObjectives;         %nr of objectives
    global vabidir;             %path of vabi simulator
    global nCounter;            %count the current round of the algorithm,
                                %used for reading vabi results

    %
    %
    %

    %initialise output array
    res = zeros(size(aParticles,1), nObjectives);
    %
    % Objective function
    %

    %loop over all particles
    for i=1:size(aParticles,1)

        %set some helping variables
        infiltration = aParticles(i,62);
        glasstype = aParticles(i,72);
        load = 1;

        %write some data with the changing parameters
        writedataHOE(aParticles(i,73), aParticles(i,74));
        writedataIWP(aParticles(i,75), aParticles(i,76), aParticles(i,77));

        %write an inputfile for vabi
        file_1 = fopen('temp.txt','w');
        for j=1:size(aParticles,2),
            fprintf(file_1, '%f', aParticles(i,j));
            if j<size(aParticles,2),
                fprintf(file_1,', ');
            end
        end
        fclose(file_1);

        %write other datafiles
        writedataBFY(1, load, 'temp.txt');
        writedataMVE(infiltration);
        writedataTYP(glasstype);

        %run vabi with all these files
        run = strcat(vabidir, 'exe\', 'run.bat');
        system(run); %#ok<NASGU>

        %read a single row from the vabi output file
        row = (nCounter*size(aParticles,1)) + i;
        res(i,:) = readdataGVU(row);

    end

end
```

writedataHOE.m

```
function writedataHOE(geometry, height)

    global vabidir;             %path of vabi simulator

    %
    %
    %

    fileNameSource = strcat(vabidir, 'ref\', 'VA114IN.HOE.N');
    fileNameTarget = strcat(vabidir, 'ref\', 'VA114IN.HOE');

    file_1 = fopen(fileNameSource,'w');

    header1 = 'File Va114in.hoe - coordinaten alle hoekpunten in het gebouw';
    header2 = ' Aantal hoekpuntnummers';
    header3 = '  28';
    header4 = ' Hptnr    X(in m)     Y(in m)     Z(in m)';

    fprintf(file_1,'%s\r\n%s\r\n%s\r\n%s\r\n',header1, header2, header3, header4);
    [nr,x,y,z] = readdataHOE();

    x(13:24) = geometry;
    y(18:19) = height;
    y(22:23) = height;

    for i=1:size(nr,1),
        fprintf(file_1,'%d\t%f\t%f\t%f\r\n',nr(i),x(i),y(i),z(i));
    end

    fclose(file_1);

    copyfile(fileNameSource, fileNameTarget);

end
```

writedataIWP.m

```
function writedataIWP(personen, apparaten, verlichting)

    global vabidir;             %path of vabi simulator

    %
    %
    %

    personen = round(personen * 230.3);
    apparaten = round(apparaten * 230.3);
    verlichting = round(verlichting * 230.3);

    fileNameSource = strcat(vabidir, 'ref\', 'VA114IN1.IWP.N');
    fileNameTarget = strcat(vabidir, 'ref\', 'VA114IN1.IWP');

    file_1 = fopen(fileNameSource,'w');

    header1 = '';
    header2 = ' latente of vochtdeel';
    header3 = ' VPERS VAPPA VVERL VZON RMETA RCLOW RCLOZ';
    header4 = ' 0.40 0.00 0.00  .0  1.20  0.90 0.70';
    header5 = ' convectiefactoren personen, apparatuur, verlichting, zon';
    header6 = ' CPERS CAPPA CVERL CZON';
    header7 = ' 0.50 1.00 0.80  0.1';
    header8 = ' schakelende verlichting';
    header9 = ' RLIMIN  DAGLIA  DAGLIU';
    header10 = ' 1.00  0  9999';
    header11 = ' A. vermogens van personen';
    header12 = ' IWKDAG  IUUR --->';
    header13 = ' UUR 1 t/m 24';

    fprintf(file_1,...
            '%s\r\n%s\r\n%s\r\n%s\r\n%s\r\n%s\r\n%s\r\n%s\r\n%s\r\n%s\r\n%s\r\n...
            %s\r\n%s\r\n',...
            header1, header2, header3, header4, header5, header6, header7, header8,...
            header9,header10, header11, header12, header13);
```

```
[nr h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20...
 h21 h22 h23 h24] = readdataIWP(13);
n = 75;
f = ceil(n.*rand(100,1));

h9(1:5) = personen;
h10(1:5) = personen;
h11(1:5) = personen;
h12(1:5) = personen;
h13(1:5) = personen;
h14(1:5) = personen;
h15(1:5) = personen;
h16(1:5) = personen;
h17(1:5) = personen;
h18(1:5) = personen;

for i=1:size(h9,1),
    fprintf(file_1,...
    ' %d %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f...
      %f %f\r\n',...
    nr(i), h1(i), h2(i), h3(i), h4(i), h5(i), h6(i), h7(i), h8(i), h9(i),...
    h10(i), h11(i),h12(i), h13(i), h14(i), h15(i), h16(i), h17(i), h18(i),...
    h19(i), h20(i), h21(i), h22(i),h23(i), h24(i));
end

header1 = ' B. vermogens van apparatuur';
header2 = ' IWKDAG  IUUR --->';
header3 = ' UUR 1 t/m 24';

fprintf(file_1,'%s\r\n%s\r\n%s\r\n',header1, header2, header3);
[nr h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20...
 h21 h22 h23 h24] = readdataIWP(13+ 9 + 3);

n = 75;
f = ceil(n.*rand(100,1));

h9(1:5) = apparaten;
h10(1:5) = apparaten;
h11(1:5) = apparaten;
h12(1:5) = apparaten;
h13(1:5) = apparaten;
h14(1:5) = apparaten;
h15(1:5) = apparaten;
h16(1:5) = apparaten;
h17(1:5) = apparaten;
h18(1:5) = apparaten;

for i=1:size(h9,1),
    fprintf(file_1,...
    ' %d %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f...
      %f %f\r\n',...
    nr(i), h1(i), h2(i), h3(i), h4(i), h5(i), h6(i), h7(i), h8(i), h9(i),...
    h10(i), h11(i),h12(i), h13(i), h14(i), h15(i), h16(i), h17(i), h18(i),...
    h19(i), h20(i), h21(i),h22(i), h23(i), h24(i));
end

header1 = ' C. vermogens van verlichting';
header2 = ' IWKDAG  IUUR --->';
header3 = ' UUR 1 t/m 24';

fprintf(file_1,'%s\r\n%s\r\n%s\r\n',header1, header2, header3);
[nr h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20...
```

```
     h21 h22 h23 h24] = readdataIWP(13+ 9 + 3 + 9 + 3);

    n = 75;
    f = ceil(n.*rand(100,1));

    h9(1:5) = verlichting;
    h10(1:5) = verlichting;
    h11(1:5) = verlichting;
    h12(1:5) = verlichting;
    h13(1:5) = verlichting;
    h14(1:5) = verlichting;
    h15(1:5) = verlichting;
    h16(1:5) = verlichting;
    h17(1:5) = verlichting;
    h18(1:5) = verlichting;

    for i=1:size(h9,1),
        fprintf(file_1,...
        ' %d %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f...
          %f %f\r\n',...
        nr(i), h1(i), h2(i), h3(i), h4(i), h5(i), h6(i), h7(i), h8(i), h9(i),...
        h10(i), h11(i), h12(i), h13(i), h14(i), h15(i), h16(i), h17(i), h18(i),...
        h19(i), h20(i),h21(i), h22(i), h23(i), h24(i));
    end

    fclose(file_1);

    copyfile(fileNameSource, fileNameTarget);

end
```

writedataBFY.m

```
function writedataBFY(j, load, fileName)

    global vabidir;             %path of vabi simulator

    %
    %
    %

    fileNameSource = strcat(vabidir, 'ref\', 'VA114IN.BFY.R');
    fileNameTarget = strcat(vabidir, 'ref\', 'VA114IN.BFY');

    % "database" open and connect
    fid=fopen(fileNameSource,'r');
    x=fread(fid);
    % close "database"
    fclose(fid);

    inputFile = 'samples.txt';
    if load == 1,
        inputFile = fileName;
    end

    %read symbols and translate to characters
    x1=char(x');

    % a equals number of lines
```

```
a=find(x'==10);
b=[0 a];

%write lines individually intop workspace
for i=1:length(a)
    eval(['line' num2str(i) '=x1(b(i)+1:b(i+1));'])
end

%Thickness
Twall1= csvread(inputFile,j-1,0,[j-1,0,j-1,0]);%Spalte 1
Twall2= csvread(inputFile,j-1,1,[j-1,1,j-1,1]);%Spalte 2
Twall3= csvread(inputFile,j-1,2,[j-1,2,j-1,2]);%Spalte 3

Tfloor1= csvread(inputFile,j-1,3,[j-1,3,j-1,3]);%Spalte 4
Tfloor2= csvread(inputFile,j-1,4,[j-1,4,j-1,4]);%Spalte 5
Tfloor3= csvread(inputFile,j-1,5,[j-1,5,j-1,5]);%Spalte 6
Tfloor4= csvread(inputFile,j-1,6,[j-1,6,j-1,6]);%Spalte 7
Tfloor5= csvread(inputFile,j-1,7,[j-1,7,j-1,7]);%Spalte 8
Tfloor6= csvread(inputFile,j-1,8,[j-1,8,j-1,8]);%Spalte 9

Troof1= csvread(inputFile,j-1,9,[j-1,9,j-1,9]);%Spalte 10
Troof2= csvread(inputFile,j-1,10,[j-1,10,j-1,10]);%Spalte 11
Troof3= csvread(inputFile,j-1,11,[j-1,11,j-1,11]);%Spalte 12
Troof4= csvread(inputFile,j-1,12,[j-1,12,j-1,12]);%Spalte 13
Troof5= csvread(inputFile,j-1,13,[j-1,13,j-1,13]);%Spalte 14

%Conductivity
Cwall1= csvread(inputFile,j-1,14,[j-1,14,j-1,14]);%Spalte 15
Cwall2= csvread(inputFile,j-1,15,[j-1,15,j-1,15]);%Spalte 16
Cwall3= csvread(inputFile,j-1,16,[j-1,16,j-1,16]);%Spalte 17

Cfloor1= csvread(inputFile,j-1,17,[j-1,17,j-1,17]);%Spalte 18
Cfloor2= csvread(inputFile,j-1,18,[j-1,18,j-1,18]);%Spalte 19
Cfloor3= csvread(inputFile,j-1,19,[j-1,19,j-1,19]);%Spalte 20
Cfloor4= csvread(inputFile,j-1,20,[j-1,20,j-1,20]);%Spalte 21
Cfloor5= csvread(inputFile,j-1,21,[j-1,21,j-1,21]);%Spalte 22
Cfloor6= csvread(inputFile,j-1,22,[j-1,22,j-1,22]);%Spalte 23

Croof1= csvread(inputFile,j-1,23,[j-1,23,j-1,23]);%Spalte 24
Croof2= csvread(inputFile,j-1,24,[j-1,24,j-1,24]);%Spalte 25
Croof3= csvread(inputFile,j-1,25,[j-1,25,j-1,25]);%Spalte 26
Croof4= csvread(inputFile,j-1,26,[j-1,26,j-1,26]);%Spalte 27
Croof5= csvread(inputFile,j-1,27,[j-1,27,j-1,27]);%Spalte 28

%Densitity
Dwall1= csvread(inputFile,j-1,28,[j-1,28,j-1,28]);%Spalte 29
Dwall2= csvread(inputFile,j-1,29,[j-1,29,j-1,29]);%Spalte 30
Dwall3= csvread(inputFile,j-1,30,[j-1,30,j-1,30]);%Spalte 31

Dfloor1= csvread(inputFile,j-1,31,[j-1,31,j-1,31]);%Spalte 32
Dfloor2= csvread(inputFile,j-1,32,[j-1,32,j-1,32]);%Spalte 33
Dfloor3= csvread(inputFile,j-1,33,[j-1,33,j-1,33]);%Spalte 34
Dfloor4= csvread(inputFile,j-1,34,[j-1,34,j-1,34]);%Spalte 35
Dfloor5= csvread(inputFile,j-1,35,[j-1,35,j-1,35]);%Spalte 36
Dfloor6= csvread(inputFile,j-1,36,[j-1,36,j-1,36]);%Spalte 37

Droof1= csvread(inputFile,j-1,37,[j-1,37,j-1,37]);%Spalte 38
Droof2= csvread(inputFile,j-1,38,[j-1,38,j-1,38]);%Spalte 39
Droof3= csvread(inputFile,j-1,39,[j-1,39,j-1,39]);%Spalte 40
Droof4= csvread(inputFile,j-1,40,[j-1,40,j-1,40]);%Spalte 41
Droof5= csvread(inputFile,j-1,41,[j-1,41,j-1,41]);%Spalte 42
```

```
%Specific heat capacity
SHCwall1= csvread(inputFile,j-1,42,[j-1,42,j-1,42]);%Spalte 43
SHCwall2= csvread(inputFile,j-1,43,[j-1,43,j-1,43]);%Spalte 44
SHCwall3= csvread(inputFile,j-1,44,[j-1,44,j-1,44]);%Spalte 45

SHCfloor1= csvread(inputFile,j-1,45,[j-1,45,j-1,45]);%Spalte 46
SHCfloor2= csvread(inputFile,j-1,46,[j-1,46,j-1,46]);%Spalte 47
SHCfloor3= csvread(inputFile,j-1,47,[j-1,47,j-1,47]);%Spalte 48
SHCfloor4= csvread(inputFile,j-1,48,[j-1,48,j-1,48]);%Spalte 49
SHCfloor5= csvread(inputFile,j-1,49,[j-1,49,j-1,49]);%Spalte 50
SHCfloor6= csvread(inputFile,j-1,50,[j-1,50,j-1,50]);%Spalte 51

SHCroof1= csvread(inputFile,j-1,51,[j-1,51,j-1,51]);%Spalte 52
SHCroof2= csvread(inputFile,j-1,52,[j-1,52,j-1,52]);%Spalte 53
SHCroof3= csvread(inputFile,j-1,53,[j-1,53,j-1,53]);%Spalte 54
SHCroof4= csvread(inputFile,j-1,54,[j-1,54,j-1,54]);%Spalte 55
SHCroof5= csvread(inputFile,j-1,55,[j-1,55,j-1,55]);%Spalte 56

%SA, IE, OE
SAwall= csvread(inputFile,j-1,56,[j-1,56,j-1,56]);%Spalte 57
IEwall= csvread(inputFile,j-1,57,[j-1,57,j-1,57]);%Spalte 58
OEwall= csvread(inputFile,j-1,58,[j-1,58,j-1,58]);%Spalte 59

%XKOS 59

%U-value, thickness single, double glass
US= csvread(inputFile,j-1,59,[j-1,59,j-1,59]);%Spalte 61
UD= csvread(inputFile,j-1,60,[j-1,60,j-1,60]);%Spalte 62

% ThickEnkel 62
% ThickDouble 63
% infiltration 64
% vermpersonen 65
% vermapparaten 66
% vermverlichting 67
% glasssurface 68
% singledouble 69

SAroof= csvread(inputFile,j-1,62,[j-1,62,j-1,62]);%Spalte 71
IEroof= csvread(inputFile,j-1,63,[j-1,63,j-1,63]);%Spalte 72
OEroof= csvread(inputFile,j-1,64,[j-1,64,j-1,64]);%Spalte 73

SAfloor= csvread(inputFile,j-1,65,[j-1,65,j-1,65]);%Spalte 74
IEfloor= csvread(inputFile,j-1,66,[j-1,66,j-1,66]);%Spalte 75
OEfloor= csvread(inputFile,j-1,67,[j-1,67,j-1,67]);%Spalte 76

SAglass= csvread(inputFile,j-1,68,[j-1,68,j-1,68]);%Spalte 77
IEglass= csvread(inputFile,j-1,69,[j-1,69,j-1,69]);%Spalte 78
OEglass= csvread(inputFile,j-1,70,[j-1,70,j-1,70]);%Spalte 79
glassType = csvread(inputFile,j-1,71,[j-1,71,j-1,71]);%Spalte 79

%single, double glass
line218=strrep(line218,'VAR_UD', num2str(UD,10));
line58=strrep(line58,'VAR_US', num2str(US,10));
line80=strrep(line80,'VAR_US', num2str(US,10));

if glassType == 0,
    line183=strrep(line183,'VAR', '0BR-Dubbelglas');
else
    line183=strrep(line183,'VAR', '0BR-window1');
end
```

```
%SA,IE,OE SAwall SAroof SAfloor SAglass
line6=strrep(line6,'VAR_SA', num2str(SAwall, 10));
line6=strrep(line6,'VAR_OE', num2str(OEwall, 10));
line40=strrep(line40,'VAR_SA', num2str(SAwall, 10));
line40=strrep(line40,'VAR_IE', num2str(IEwall, 10));

line103=strrep(line103,'VAR_SA', num2str(SAwall, 10));
line103=strrep(line103,'VAR_OE', num2str(OEwall, 10));
line137=strrep(line137,'VAR_SA', num2str(SAwall, 10));
line137=strrep(line137,'VAR_IE', num2str(IEwall, 10));

line144=strrep(line144,'VAR_SA', num2str(SAglass, 10));
line144=strrep(line144,'VAR_OE', num2str(OEglass, 10));
line178=strrep(line178,'VAR_SA', num2str(SAglass, 10));
line178=strrep(line178,'VAR_IE', num2str(IEglass, 10));

line241=strrep(line241,'VAR_SA', num2str(SAfloor, 10));
line241=strrep(line241,'VAR_OE', num2str(OEfloor, 10));
line275=strrep(line275,'VAR_SA', num2str(SAfloor, 10));
line275=strrep(line275,'VAR_IE', num2str(IEfloor, 10));

line282=strrep(line282,'VAR_SA', num2str(SAroof, 10));
line282=strrep(line282,'VAR_OE', num2str(OEroof, 10));
line316=strrep(line316,'VAR_SA', num2str(SAroof, 10));
line316=strrep(line316,'VAR_IE', num2str(IEroof, 10));

%wall
line10=strrep(line10,'VAR_T', num2str(Twall1, 10));
line10=strrep(line10,'VAR_C', num2str(Cwall1, 10));
line10=strrep(line10,'VAR_D', num2str(Dwall1, 10));
line10=strrep(line10,'VAR_SHC', num2str(SHCwall1, 10));
line16=strrep(line10,'VAR_T', num2str(Twall2, 10));
line16=strrep(line10,'VAR_C', num2str(Cwall2, 10));
line16=strrep(line10,'VAR_D', num2str(Dwall2, 10));
line16=strrep(line10,'VAR_SHC', num2str(SHCwall2, 10));
line19=strrep(line19,'VAR_T', num2str(Twall3, 10));
line19=strrep(line19,'VAR_C', num2str(Cwall3, 10));
line19=strrep(line19,'VAR_D', num2str(Dwall3, 10));
line19=strrep(line19,'VAR_SHC', num2str(SHCwall3, 10));

%ORW-internalpartition
line107=strrep(line107,'VAR_T', num2str(Twall1, 10));
line107=strrep(line107,'VAR_C', num2str(Cwall1, 10));
line107=strrep(line107,'VAR_D', num2str(Dwall1, 10));
line107=strrep(line107,'VAR_SHC', num2str(SHCwall1, 10));
line113=strrep(line113,'VAR_T', num2str(Twall2, 10));
line113=strrep(line113,'VAR_C', num2str(Cwall2, 10));
line113=strrep(line113,'VAR_D', num2str(Dwall2, 10));
line113=strrep(line113,'VAR_SHC', num2str(SHCwall2, 10));
line116=strrep(line116,'VAR_T', num2str(Twall3, 10));
line116=strrep(line116,'VAR_C', num2str(Cwall3, 10));
line116=strrep(line116,'VAR_D', num2str(Dwall3, 10));
line116=strrep(line116,'VAR_SHC', num2str(SHCwall3, 10));

%floor
%line245=strrep(line245,'VAR_T', num2str(Tfloor1, 10));
line245=strrep(line245,'VAR_C', num2str(Cfloor1, 10));
line245=strrep(line245,'VAR_D', num2str(Dfloor1, 10));
line245=strrep(line245,'VAR_SHC', num2str(SHCfloor1, 10));

line248=strrep(line248,'VAR_T', num2str(Tfloor1, 10));
line248=strrep(line248,'VAR_C', num2str(Cfloor1, 10));
```

```
line248=strrep(line248,'VAR_D', num2str(Dfloor1, 10));
line248=strrep(line248,'VAR_SHC', num2str(SHCfloor1, 10));

line251=strrep(line251,'VAR_T', num2str(Tfloor2, 10));
line251=strrep(line251,'VAR_C', num2str(Cfloor2, 10));
line251=strrep(line251,'VAR_D', num2str(Dfloor2, 10));
line251=strrep(line251,'VAR_SHC', num2str(SHCfloor2, 10));

line254=strrep(line254,'VAR_T', num2str(Tfloor3, 10));
line254=strrep(line254,'VAR_C', num2str(Cfloor3, 10));
line254=strrep(line254,'VAR_D', num2str(Dfloor3, 10));
line254=strrep(line254,'VAR_SHC', num2str(SHCfloor3, 10));

line257=strrep(line257,'VAR_T', num2str(Tfloor4, 10));
line257=strrep(line257,'VAR_C', num2str(Cfloor4, 10));
line257=strrep(line257,'VAR_D', num2str(Dfloor4, 10));
line257=strrep(line257,'VAR_SHC', num2str(SHCfloor4, 10));

line260=strrep(line260,'VAR_T', num2str(Tfloor5, 10));
line260=strrep(line260,'VAR_C', num2str(Cfloor5, 10));
line260=strrep(line260,'VAR_D', num2str(Dfloor5, 10));
line260=strrep(line260,'VAR_SHC', num2str(SHCfloor5, 10));

line263=strrep(line263,'VAR_T', num2str(Tfloor6, 10));
line263=strrep(line263,'VAR_C', num2str(Cfloor6, 10));
line263=strrep(line263,'VAR_D', num2str(Dfloor6, 10));
line263=strrep(line263,'VAR_SHC', num2str(SHCfloor6, 10));

%roof
line286=strrep(line286,'VAR_T', num2str(Troof1, 10));
line286=strrep(line286,'VAR_C', num2str(Croof1, 10));
line286=strrep(line286,'VAR_D', num2str(Droof1, 10));
line286=strrep(line286,'VAR_SHC', num2str(SHCroof1, 10));

line289=strrep(line289,'VAR_T', num2str(Troof2, 10));
line289=strrep(line289,'VAR_C', num2str(Croof2, 10));
line289=strrep(line289,'VAR_D', num2str(Droof2, 10));
line289=strrep(line289,'VAR_SHC', num2str(SHCroof2, 10));

line292=strrep(line292,'VAR_T', num2str(Troof3, 10));
line292=strrep(line292,'VAR_C', num2str(Croof3, 10));
line292=strrep(line292,'VAR_D', num2str(Droof3, 10));
line292=strrep(line292,'VAR_SHC', num2str(SHCroof3, 10));

line295=strrep(line295,'VAR_T', num2str(Troof4, 10));
line295=strrep(line295,'VAR_C', num2str(Croof4, 10));
line295=strrep(line295,'VAR_D', num2str(Droof4, 10));
line295=strrep(line295,'VAR_SHC', num2str(SHCroof4, 10));

line301=strrep(line301,'VAR_T', num2str(Troof5, 10));
line301=strrep(line301,'VAR_C', num2str(Croof5, 10));
line301=strrep(line301,'VAR_D', num2str(Droof5, 10));
line301=strrep(line301,'VAR_SHC', num2str(SHCroof5, 10));

% combine all lines into one line
test='';
for i=1:length(a)
    eval(['test=[test line' num2str(i) '];'])
end

% "database" open and connect
fid3=fopen(fileNameTarget,'w');
```

```
    fwrite(fid3,test');
    % close "database"
    fclose(fid3);

end
```

writedataMVE.m

```
function writedataMVE(infiltration)

    global vabidir;             %path of vabi simulator

    %
    %
    %

    fileNameSource = strcat(vabidir, 'ref\', 'VA114IN1.MVE.N');
    fileNameTarget = strcat(vabidir, 'ref\', 'VA114IN1.MVE');
    file_1 = fopen(fileNameSource,'w');

    header1 = 'gegevens van mech.vent.(m3/s)';
    header2 = '1=dagbedrijf , 2=nacht/weekendbedrijf';
    header3 = 'bedrijfsperiode, totaal, vers, afzuig, separaat';
    header4 = '1 VAR  VAR  VAR  0.000000';
    header5 = '2 0.000000  0.000000  0.000000  0.000000';
    header6 = '';
    header7 = 'nachtkoel/verw, totaal, vers, afzuig, separaat';
    header8 = '3 VAR  VAR  VAR  0.000000';
    header9 = 'er treedt onbalans op in vertrek 0';
    header10 = 'luchtstromen gaan naar(+) of komen van(-)';
    header11 = 'de omgeving of een aangrenzend vertrek';
    header12 = 'de volgende stromen betekenen :';
    header13 = 'periode,onbalans,omgeving,vertrek-a,vertrek-b,vertrek-c';
    header14 = '1 0.000000  0.000000  0.000000';
    header15 = '2 0.000000  0.000000  0.000000';
    header16 = '3 0.000000  0.000000  0.000000';

    for i=1:3,
        fprintf(file_1,'%s\r\n',eval(['header' num2str(i)]));
    end

    fprintf(file_1,'1 %.10f  %.10f  %.10f  0.000000\r\n', infiltration, infiltration,...
                infiltration);

    for i=5:7,
        fprintf(file_1,'%s\r\n',eval(['header' num2str(i)]));
    end

    fprintf(file_1,'3 %.10f  %.10f  %.10f  0.000000\r\n', infiltration, infiltration,...
                infiltration);

    for i=9:16,
        fprintf(file_1,'%s\r\n',eval(['header' num2str(i)]));
    end

    fclose(file_1);

    copyfile(fileNameSource, fileNameTarget);
```

```
end
```

writedataTYP.m

```
function writedataTYP(glass)

    global vabidir;             %path of vabi simulator

    %
    %
    %

    fileNameSource = strcat(vabidir, 'ref\', 'VA114IN1.TYP.N');
    fileNameTarget = strcat(vabidir, 'ref\', 'VA114IN1.TYP');
    file_1 = fopen(fileNameSource,'w');

    header1 = 'File VA114INx.TYP: soort vlak + bouwfysische naam van vlak';
    header2 = 'Vlak  W/R/D? Bfy-naam';
    header3 = '  -   1/2/3     -';
    header4 = '  1     1    0BW-externalwall';
    header5 = '  2     2    0BR-window1';
    header6 = '  3     2    0BR-window1';
    header7 = '  4     1    1RW-internalpartition';
    header8 = '  5     1    0BW-externalwall';
    header9 = '  6     2    0BR-window1';
    header10 = '  7     2    0BR-window1';
    header11 = '  8     1    0BW-test';
    header12 = ' 10     1    1CW-TUSSENVLOER';
    header13 = ' 11     1    0BW-roof';
    header14 = 'Naam waaronder overige informatie (TIM, plafond vent, ed.) staat';
    header15 = 'rest1';

    for i=1:11,
        fprintf(file_1,'%s\r\n',eval(['header' num2str(i)]));
    end

    if glass == 0,
        fprintf(file_1,'  9     2    0BR-Dubbelglas\r\n');
    else
        fprintf(file_1,'  9     2    0BR-window1\r\n');
    end

    for i=12:15,
        fprintf(file_1,'%s\r\n',eval(['header' num2str(i)]));
    end

    fclose(file_1);

    copyfile(fileNameSource, fileNameTarget);

end
```

readdataGVU.m

```
function ret = readdataGVU(round)
```

```
    global vabidir;             %path of vabi simulator

    %
    %
    %

    fileNameSource = strcat(vabidir, 'va114gvu.par');

    [IPAR IPARGEV PARGEV QCEVW QCEKL PCEVW PCEKL IV QLOVW  QLOKL    PLOVW  PLOKL TLMAX...
     TLMIN TLGEM TCMAX TCMIN OHL25 OHL28 OHC25 OHC28  WOH_min  WOH_plus] = ...
    textread(fileNameSource,...
             '%d %d %s %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f',...
             'headerlines',3 + round - 1);

    %output value of ret(1) to see what goes wrong
    %test = WOH_plus + WOH_min %#ok<NASGU,NOPRT>
    %test2 = abs(QCEKL) + QLOVW  %#ok<NOPRT,NASGU>

    ret(1)= WOH_plus + WOH_min;
    ret(2)= abs(QCEKL) + QLOVW;

end
```

nonDominatedSet.m

```
function [resX, resY] = nonDominatedSet(aParticles, aObjectiveValues)

    %aParticles is passed, can alse be [aParticles; aRepositoryX]
    %aObjectiveValues is passed, can alse be [aObjectiveValues; aRepositoryY]

    %
    %
    %

    %clear what will become aRepositoryX and aRepositoryY
    resX = [];
    resY = [];

    %find non-dominated particles
    for i=1:size(aObjectiveValues,1),

        %test if current particle is non-dominated
        is_dominated = false;
        for j=1:size(aObjectiveValues,1),

            if dominates(aObjectiveValues(j,:),aObjectiveValues(i,:)) == true,
                is_dominated = true;
            end

        end

        if is_dominated == false,
            %particle i is non-dominated in current population (+ repository)
            resX = [resX; aParticles(i,:)]; %#ok<AGROW>
            resY = [resY; aObjectiveValues(i,:)]; %#ok<AGROW>
        end

    end
```

```
end
```

dominates.m

```
function res = dominates(particleA, particleB)

    res = false;

    %check if A < B for all i
    for i=1:size(particleA,2)

        %check if any value of B is better than A
        if (particleB(i) < particleA(i))

            res = false;
            return

        %check if any value of A is better than B
        elseif (particleB(i) > particleA(i))

            res = true;

        end

    end

    %if we get here, this means that no value of B is better than the A

    %the value of res will tell us if there is a value of A that is
    %better than B

    %if res is false at this point than all values of A and B are the same

end
```

getHypercubeCoordinates.m

```
function res = getHypercubeCoordinates(aParticles)

    global nDivisions;          %nr of divisions used to compute hypercubes

    %
    %
    %

    %get upper- and lowerbound for objective-values discovered so far
    aMin = aParticles(1,:);
    aMax = aParticles(1,:);
    for i=1:size(aParticles,1)
        for j=1:size(aParticles,2)

            if (aParticles(i,j) < aMin(1,j))
                aMin(j) = aParticles(i,j);
            elseif (aParticles(i,j) > aMax(1,j))
                aMax(j) = aParticles(i,j);
            end

        end
    end

    %divide search space into hypercubes
    aCubeSize = zeros(1,size(aMin,2));
    for j=1:size(aMin,2)

        aCubeSize(j) = (aMax(j) - aMin(j)) / (nDivisions-1);

    end

    %compute hypercube coordinates for each particle
    %
    %   temp(x,x) = coordx, coordx, ..., coordx
    %
    temp = zeros(size(aParticles,1), size(aParticles,2));
    for i=1:size(aParticles,1)
        for j=1:size(aParticles,2)

            %assign coordinate-number of hypercube
            %add 1 to make the range go from 1-nDivisions instead of 0-(nDivisions-1)
            temp(i,j) = floor((aParticles(i,j) - aMin(j)) / aCubeSize(j)) + 1;

        end
    end

    %put particles in structs based on coordinates
    %
    %   res.coord(1) = x
    %   res.coord(2) = x
    %   ...
    %   res.particles = [x x x x]
    %
    structCounter = 1;
    for i=1:size(temp,1)

            if (i == 1)

                %first particle, start building the struct
                res(structCounter).coords = temp(i,:); %#ok<AGROW>
                res(structCounter).particles = i; %#ok<AGROW>
                structCounter = structCounter + 1;

            else

                %check if the coordinates are allready in struct
                bSameCoordinates = false;
                for k=1:size(res,2)
                    if isequal(temp(i,:), res(k).coords)

                        %coordinates are in struct
                        %add this particle to this set of coordinates
                        res(k).particles = [res(k).particles i]; %#ok<AGROW>
                        bSameCoordinates = true;

                    end
                end

                %if the coordinates weren't found in the struct
```

```
                %add this particle as the next row in the struct
                if (bSameCoordinates == false)
                   res(structCounter).coords = temp(i,:); %#ok<AGROW>
                   res(structCounter).particles = i; %#ok<AGROW>
                   structCounter = structCounter + 1;
                end

            end

        end

    end
```

moveParticlesVabi.m

```
    function res = moveParticlesVabi()

        global aParticles;
        global aVelocity;
        global aPbestsX;
        global aRepositoryX;
        global nInertiaWeight;

        global MU;
        global S;
        global aIndexes;

        global aLowerBounds;
        global aUpperBounds;

        %initialise variables
        aTempVelocity = zeros(size(aParticles,1), size(aParticles,2));
        res = zeros(size(aParticles,1), size(aParticles,2));

        %loop over particles
        for i=1:size(aParticles,1)

            %loop over attributes to compute indivual speeds
            for j=1:size(aParticles,2)

                %dont move the normal parameters
                if ((j >= 72) || (aIndexes(j) ~= 0))

                    %compute h: index for a global best particle from the repository
                    h = chooseRepositoryIndex();

                    %compute speed of particle
                    aTempVelocity(i,j) = nInertiaWeight * aVelocity(i,j) + ...
                                         rand(1) * (aPbestsX(i,j) - aParticles(i,j)) + ...
                                         rand(1) * (aRepositoryX(h,j) - aParticles(i,j));

                    %this attributes should be changed but must also stay within bounds
                    if ((j < 72) && (aIndexes(j) == 1))

                        %inputs must be withing MU + 3S
                        nLowerBound = MU(j) - (3*S(j));
                        nUpperBound = MU(j) + (3*S(j));

                        if ((aTempVelocity(i,j) + aParticles(i,j)) < 0)
                    res(i,j) = aParticles(i,j);
                    aVelocity(i,j) = 0;
                elseif ((aTempVelocity(i,j) + aParticles(i,j)) < nLowerBound)
                    res(i,j) = nLowerBound;
                    aVelocity(i,j) = nLowerBound - aParticles(i,j);
                elseif ((aTempVelocity(i,j) + aParticles(i,j)) > nUpperBound)
                    res(i,j) = nUpperBound;
                    aVelocity(i,j) = nUpperBound - aParticles(i,j);
                else
                    res(i,j) = (aTempVelocity(i,j) + aParticles(i,j));
                    aVelocity(i,j) = aTempVelocity(i,j);
                end

            %parameter glasswindow should always be 0 or 1
            elseif (j == 72)

                if ((aTempVelocity(i,j) + aParticles(i,j)) >= 0.5)
                    res(i,j) = 1;
                else
                    res(i,j) = 0;
                end
                aVelocity(i,j) = aTempVelocity(i,j);

            %these attributes should stay within there own upper- and lowerbounds
            elseif (j > 72)

                boundIndex = (j-72);

                if ((aTempVelocity(i,j) + aParticles(i,j)) < 0)
                    res(i,j) = aParticles(i,j);
                    aVelocity(i,j) = 0;
                elseif ((aTempVelocity(i,j) + aParticles(i,j)) ...
                        < aLowerBounds(boundIndex))
                    res(i,j) = aLowerBounds(boundIndex);
                    aVelocity(i,j) = aLowerBounds(boundIndex) - aParticles(i,j);
                elseif ((aTempVelocity(i,j) + aParticles(i,j)) ...
                        > aUpperBounds(boundIndex))
                    res(i,j) = aUpperBounds(boundIndex);
                    aVelocity(i,j) = aUpperBounds(boundIndex) - aParticles(i,j);
                else
                    res(i,j) = (aTempVelocity(i,j) + aParticles(i,j));
                    aVelocity(i,j) = aTempVelocity(i,j);
                end

            end

        %other attributes should always be at their mean value
        else

            %we don't adjust these parameters, just copy them
            res(i,j) = aParticles(i,j); %#ok<AGROW,NASGU>

        end

    end


end

end
```

chooseRepositoryIndex.m

```
function res = chooseRepositoryIndex()

    global aCoordinates;        %hypercube coordinates of current particles

    %
    %
    %

    %compute weight of each hypercube
    %hypercubes with more particles should get a lower weight
    aWeights = zeros(size(aCoordinates,2), 1);
    for i=1:size(aCoordinates,2)

        aWeights(i) = 1/size(aCoordinates(i).particles, 2);

    end

    %get the total weight of all hypercubes togethers
    nTotal = sum(aWeights);

    %compute the ratio for each hypercube
    aRatios = zeros(size(aCoordinates,2), 1);
    for i=1:size(aCoordinates,2)
        aRatios(i) = sum(aWeights(1:i))/nTotal;
    end

    %take a random number, see in which hypercube it lands
    nHypercube = rand(1);
    for i=1:size(aCoordinates, 2)
        if (nHypercube <= aRatios(i))
            %set the number of the hypercube as a normal integer
            nHypercube = i;
            break;
        end
    end

    %take a random particle from aCoordinates(nHypercube).particles
    nParticle = round(1 + (size(aCoordinates(nHypercube).particles, 2)-1) * rand(1));
    res = aCoordinates(nHypercube).particles(nParticle);

end
```

computeHypervolume.m

```
function res = computeHypervolume(aParticles)

    res = 0;

    %sort particles on height, heighest ones first
    [x, i] = sort(aParticles(:,2), 'descend');
    aParticles = aParticles(i,:);
    aParticles = [100000 100000; aParticles];

    %add volume that each particles contributes to the total hypervolume
    for i=2:size(aParticles,1)
        res = res + ( (2-aParticles(i,1)) * (aParticles(i-1,2)-aParticles(i,2)) );
        end
    end
```

References

[1] Robert Marijt, *Multi-objective Robust Optimization Algorithms for Improving Energy Consumption and Thermal Comfort of Buildings*, Master Thesis, Leiden Institute for Advanced Computer Science, Leiden University, June 2009.

[2] Nicola Beume, Boris Naujoks, und Michael Emmerich, *SMS-EMOA: Multiobjective selection based on dominated hypervolume. European Journal of Operational Research*, 181(3):1653-1669, 2007.

[3] Coello Coello, Carlos A. and Salazar Lechuga, Maximino, *MOPSO: A Proposal for Multiple Objective Particle Swarm Optimization, Congress on Evolutionary Computation (CEC'2002)*, IEEE Service Center, Piscataway, New Jersey, Volume 2, pp. 1051–1056, May 2002.

[4] Vabi software, *VA114 Gebouwsimulatie*, http://www.vabi.nl/.

[5] C.J. Hopfe, *Uncertainty and sensitivity analysis in building performance simulation for decision support and design optimization.*, Technische Universiteit Eindhoven, 2009.

[6] M. Emmerich, A. Deutz, *Test Problems based on Lamé Superspheres*, Evolutionary Multiobjective Optimization 2007 (EMO2007), Matsushima, Japan, 2007, LNCS 4403, Springer, pp. 922-936

[7] Carlos M. Fonseca and Peter J. Fleming, *On the Performance Assessment and Comparison of Stochastic Multiobjective Optimizers*, In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature–PPSN IV*, Lecture Notes in Computer Science, pages 584-593, Berlin, Germany, September 1996, Springer-Verlag.

[8] Knowles, J. (2005) *A summary-attainment-surface plotting method for visualizing the performance of stochastic multiobjective optimizers*, IEEE Intelligent Systems Design and Applications (ISDA V), In press.