UPPSALA
UNIVERSITET

Elaboration of the optimization model for data centers

Project in computational science
February 11, 2019

Supervisor:
Kateryna Mishchenko
ABB AB, Corporate Research

Authors:
Chiedza Chadehumbe
Naresh Kumar Nagarajan
Josefine Sjöberg

Abstract

When it comes to building and maintenance, the cooling of data centers is a challenge. This paper is an elaboration of a previous thesis which models a data center trying to optimize the cooling using Model Predictive Control, MPC. The elaboration consists of improving the model, evaluating the performance of the local solver used to solve the optimization problem and evaluating the performance of three global solvers. The local solver used in the previous thesis is *fmincon* and the global solvers evaluated in this project are GlobalSearch, MultiStart and PatternSearch. Comparing all the solvers, *fmincon* is found to be the best solver for the problem when regarding both time efficiency and constraint violation.

Contents

1 Introduction

1.1 Background

A data center can be defined as a facility that contains servers to store, manage and process data. The increasing demand for data centers due to the increasing usage of computing resources, cloud computing and handling of big amounts of data has led to data centers becoming an interesting research field. One of the main challenges when building and maintaining data centers is the need of an efficient cooling system for the servers. The servers perform energy intense computing resulting in heated server temperatures. The cooling of data centers is one of the directions of research at ABB Corporate Research Center. In 2017, the master thesis [1] was done by Erik Berglund at ABB which modeled a small sized data center trying to optimize the cooling system.

The model which outperforms the other controllers tested in the master thesis is the one where Model Predictive Control (MPC) algorithm is implemented. The objective of MPC is to minimize the energy usage of the Computer Room Air Handler (CRAH) units, used to cool down the servers, during a given prediction horizon. The minimization needs to satisfy lower and upper bound constraints on the CRAH outlet airflow and temperature as well as the server temperature. The optimization model is simulated in MATLAB.

1.2 Goal

The goal of our project is to elaborate the MPC optimization function developed at the ABB Corporate Research Center. This is done by improving the general model, evaluating the performance of the local solver, *fmincon*, and evaluating the performance of the three global solvers: MultiStart, GlobalSearch and PatternSearch and comparing the solvers to find the best one for the problem.

2 Literature review

2.1 Outline of MPC

MPC does not specify a specific control strategy but a broad range of control methods. The ideas of MPC can be expressed as the three following characteristics [2]:

- Explicit use of a model to predict the process output of the system at discrete points of time (prediction horizon).
- Calculation of a control sequence which minimizes some objective function.
- Application of the first control signal from the calculated control sequence as input to the system, followed by recalculations and then displacement of the prediction horizon one step into the future.

2.2 Optimization

Mathematical optimization techniques are used to optimize a function by finding a set of parameters $x = (x_0, x_1, x_2, ...)$ that minimize or maximize the given function. The function to be optimized, called the objective function $F(x)$, can be subjected to constraints and boundary conditions that limit the amount of acceptable parameter values. The optimization problem in this project is defined by:

$$\begin{aligned} minimize\ F(x) \\ Subject\ to\ G(x) \geq 0 \\ H(x) = 0 \\ x_l < x < x_u, \end{aligned} \tag{1}$$

where $F(x)$ is the objective function, $G(x)$ is a nonlinear inequality constraint, $H(x)$ is a nonlinear equality constraint and x_l and x_u are lower and upper bounds. The constraints and boundaries are limitations of the optimization problem that a solution must satisfy. Mathematically this is formulated as $x \in \Omega \subset R^n$, where Ω is the feasible region and in this case the bounds are $x_l \leq x \leq x_u$.

2.3 Local optimization

A local optimization algorithm searches for a minimum in a search region. The solver stops when it finds a minimum, or some other stopping criteria is met.

2.3.1 *fmincon*

One of the local optimization solvers provided in MATLAB, is *fmincon*, a gradient based solver designed to solve constrained nonlinear multi-variable problems [3]. The solver needs an initial guess as starting point. It is well known that an initial guess close to a minimum speeds up the computations. From the initial guess the solver iterates in steps over the objective function, performing several function evaluations per iteration until a stopping criterion is met.

The solver has five optimization algorithms as options: interior-point, trust region reflective, SQP, SQP legacy and active set [4]. In this project three were tested, interior-point, SQP and active-set, since trust region reflective requires a predefined gradient and SQP legacy is similar to regular SQP but is slower and consumes more memory.

Interior-point is the default algorithm for *fmincon* and it approaches the optimal solution from the interior of the feasible region. The good characteristics of this method are that it has low memory usage and that it can solve large problems fast. The drawback is that it is less accurate than other algorithms. Sequential Quadratic Programming, SQP, iterates forward by solving quadratic subproblems. The algorithm creates these subproblems by replacing the objective function with a quadratic approximation and by linearizing the

constraint functions. It is suitable for both small and large problems and for problems with significant nonlinearities. Active set is similar to SQP as it also solves quadratic subproblems. The difference between the two is that active set can take larger steps and can also move in the infeasible region for intermediate steps.

The solver *fmincon* has several other options that the user can customize other than the choice of algorithm [3]. The options can impose different stopping criteria, what *fmincon* can produce as output or use in the optimization. There are four stopping criteria besides finding a local minimum: the step size, the number of function evaluations or iterations and the tolerance in the first-order optimality measure. The first-order optimality condition is a measure of how close the solution is to a minimum and should be zero at a minimum. The options *StepTolerance*, *MaxFunctionEvaluations*, *MaxIterations* and *OptimalityTolerance* allows the user to customize these stopping criteria to regulate accuracy and real time. The *ConstraintTolerance* option is a more direct way of adjusting the accuracy of the solver as it sets the tolerance of constraint violations which affects the size of the feasible region. By default, *fmincon* uses forward finite differences to calculate the gradient in each step. *FiniteDifferenceStepSize* is an option for adjusting the difference in step length between two steps for finite differences. A larger value results in *fmincon* taking larger steps which can be useful if the objective function does not change much per iteration. The user can enable parallel processing for *fmincon* setting the option *UseParallel* to *true*. This enables the solver to estimate the gradient in parallel.

2.4 Global Optimization

Global optimization is a process of searching for a global solution to a problem that contains multiple minima or maxima. A global minimum is a point where the objective function value is less than or equal to all other values of the feasible points. In theory, we want to find all global solutions assuming the solution set are finite. In practice, the purpose of global optimization is to find a feasible parameter set of the global optimum. The global solvers used in this project are MultiStart, GlobalSearch and PatternSearch.

2.4.1 GlobalSearch

GlobalSearch is a multi-start heuristic algorithm which calls the local solver *fmincon* from multiple starting points attempting to find a global minimum. It uses a scatter search mechanism for generating trial points. The number of generated trial points is by default 1000 but can be modified by the user for larger evaluations. GlobalSearch then analyzes the trial points and runs the best start point candidates rejecting the points that are unlikely to improve the result [5]. GlobalSearch does not enable parallel processing.

2.4.2 MultiStart

MultiStart is a metaheuristic algorithm which allows various choices of local solvers such as *fminunc, lsqcurvefit, lsqnonlin* apart from *fmincon*. It provides a filtering option based on feasibility and runs the chosen local solver from starting points which can be customized by the user. The amount of start points is given by the user as k, and MultiStart generates k-*1* start points and takes the first start point from the problem structure to add up to the total points [5]. MultiStart then starts the local solver in these k points. The solver enables parallel processesing by running the start points in parallel.

2.4.3 PatternSearch

PatternSearch is a global solver which is not gradient based. It works by searching in a pattern around the current point starting with the initial guess. The solver calculates the objective function value at the mesh points given by the pattern vectors. When solving problems with nonlinear constraints PatternSearch uses a formulation of the optimization problem called the Augmented Lagrangian Pattern Search (ALPS) [6]. ALPS creates a subproblem using Lagrangian multipliers and penalty parameters to add the nonlinear constraints to the objective function. Each subproblem is solved during one iteration. If an iteration gives a minimum which satisfies the constraints it updates the Lagrangian multipliers and reduces the penalty parameters, but if a minimum is not found the penalty parameters is increased. It stops when the optimal solution is found or when another stopping criterion is met. PatternSearch supports parallel processing by computing the objective function and constraints in parallel.

2.5 Parallel processing

Parallel computing is simultaneous use of multiple computer cores to solve a computational problem. Figure 1 illustrates an application broken down into multiple processors to solve simultaneously. Each task is executed on a different core which is managed by a job scheduler in MATLAB. The job scheduler contains a parallel pool which consists of workers on a cluster or a desktop [7]. The parallel pool in MATLAB is initiated by the function *parpool*. The workers in the parallel pool can interact and communicate with each other until the session expires.

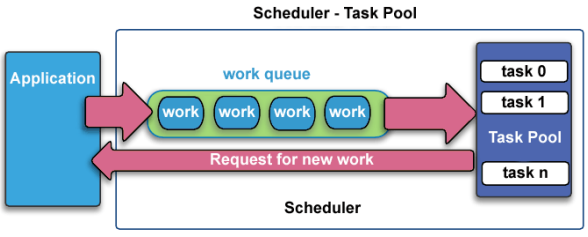

Figure 1: Parallel processing work flow

During optimization, traditional gradient-based search is often time consuming. In a single iteration the optimization solver estimates the gradient of the function which it uses to determine the search direction and magnitude of the step to next point. In the parallel version of gradient-based search, see Figure 2, the computation is distributed among the workers and several function evaluations occur simultaneously [8].

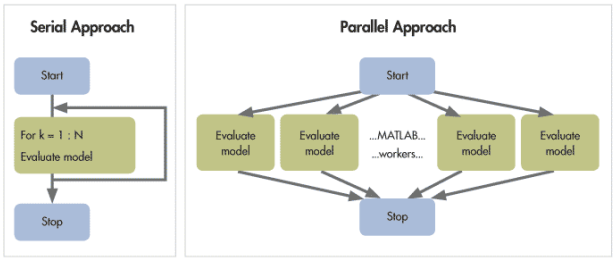

Figure 2: Serial and parallel approaches to gradient estimation [10].

3 Method

The method we follow to reach the goal of the project can be simplified into the three following steps:

- Study and improvement of the general MATLAB model.
- Evaluate the performance of the original *fmincon* optimization.
- Evaluate the performance of the three global solvers:

  i GlobalSearch

  ii MultiStart

  iii PatternSearch

  and comparing all solvers to find the best suited for the problem.

3.1 Improvement of the model

Theoretical study of the general model is necessary before any improvement of its implementation can be done. We read the master thesis [1] to obtain an understanding of the implemented optimization model. Following the theoretical study, we evaluate the performance of the original model by running the MATLAB tool Profiler trying to locate bottlenecks in the code where improvements can be made. We received a directory of MATLAB scripts where

the relevant ones for this project are titled *MPC script reduced constraints*, the MPC simulation script, and *MPC4 all racks* which is the function called to run the MPC optimization algorithm. The latter is the script where improvements can be made.

3.2 Evaluation of the performance of *fmincon*

We continue by studying the problem definition, trying to find the reason for the observed poor performance. This is done by investigating the behavior of the objective function, to draw conclusions on its possible flatness and of multi-modality. We also investigate the specifics of the *fmincon* routine.

3.3 Evaluation of the performance of global solvers

To research the possibility of finding a global solution we evaluate three global solvers: GlobalSearch, MultiStart and PatternSearch. Parallel versions of MultiStart and PatternSearch are investigated for speedup. The performance of all the solvers are analyzed and compared.

3.4 Problem set up

If no other information is given, the parameters are set according to Table 1 in the scripts *MPC script reduced constraints* and *MPC4 all racks*.

Parameter	**Description**	**Value**
N	Number of time steps in the prediction horizon	50
Δt	Length of a time step	36.9037s
TolCon	Tolerance on the constraint violation for *fmincon* and PatternSearch	10^{-1}

Table 1: Parameter values used in tests.

The tests are not run with steady state as initial state. When initial guesses 1 to 6 is mentioned it is the values displayed in Table 2 for the CRAH outlet airflow and temperature.

Initial guess	**Airflow** (m^3/s)	**Temperature** $(^\circ C)$
1	1.3	27
2	1.476	25.2
3	1.652	23.4
4	1.828	21.6
5	2.004	19.8
6	2.18	18

Table 2: The six initial guesses tested.

To measure the real time the MATLAB functions *tic* and *toc* are used and for the CPU time the function *cputime*.

3.5 Testing environment

The tests and resulting figures are produced on Uppsala University's student servers where the computer specifications are:

- CPU model: Intel Xeon(R) CPUE5520, 2.27GHz, 16-cores
- Scientific Linux, release 6.10 (Carbon)
- 12 workers to be used for parallel computing in MATLAB.

The MATLAB version used is R2018a and the necessary toolboxes for this project are the Optimization-, Global Optimization- and Parallel Computing-toolboxes.

4 Results

4.1 Improvement of the model

Three different *fmincon* algorithms were investigated in [1]: SQP, interior-point and active set, and SQP was found to have the properties best suited for this problem set up. This conclusion was draw from a different set of data, so to validate the SQP algorithm's advantage, tests were done with the provided data set. The difference in performance for the three algorithms for 50 time steps with initial guess 1 is illustrated in Table 3. The constraint violation refers to the average total constraint violation per server and the total energy consumption is the energy used by the CRAH units over the time period of the simulation. The real time is the time it takes to run the simulation.

	Real time (s)	Constraint violation	Total energy consumption
SQP	2207	$7.02596705 \cdot 10^{-4}$	$2.21026747 \cdot 10^{-7}$
Interior-point	498	$2.11615269 \cdot 10^{-2}$	$2.20753344 \cdot 10^{-7}$
Active set	5681	$2.16150753 \cdot 10^{-4}$	$2.21175404 \cdot 10^{-7}$

Table 3: Performance of the algorithms for 50 time steps.

In [1], it is stated that the original model is too slow and the real time needs to decrease by a factor of five. Using Profiler, five time-consuming matrix builds were located and in particular the function call *spdiags* used to construct these matrices. This function builds sparse diagonal matrices and using the functions *sparse* and *diag* instead saves time. The function *diag* makes dense diagonal matrices and *sparse* makes sparse matrices so together they produce sparse diagonal matrices like *spdiags*.

Profiler also showed that the function *Dynamics*, within *MPC4 all racks*, is called more times than the main function itself. *MPC4 all racks* is called once per time step and *Dynamics* is called over ten thousand times. Therefore, by moving code from *Dynamics* into *MPC4 all racks*, time was saved. The result from these changes for ten time steps is illustrated

in Table 4, where the first method is the use of *sparse* and *diag* and the second method is moving the initialization of a part of the build out of *Dynamics*.

Method	Real time (s)	CPU time (s)
Original	525	2876
First	437	3188
Second	414	3047

Table 4: Real time and CPU time for the three different methods.

4.2 Evaluation of the performance of *fmincon*

To investigate the behavior of the objective function, the convergence history for different time steps was investigated. In Figure 3, a convergence plot for one time step with initial guess 1 is shown, and in 4 for initial guess 6. The convergence history for other time steps was similar in behaviour to Figure 3 and Figure 4.

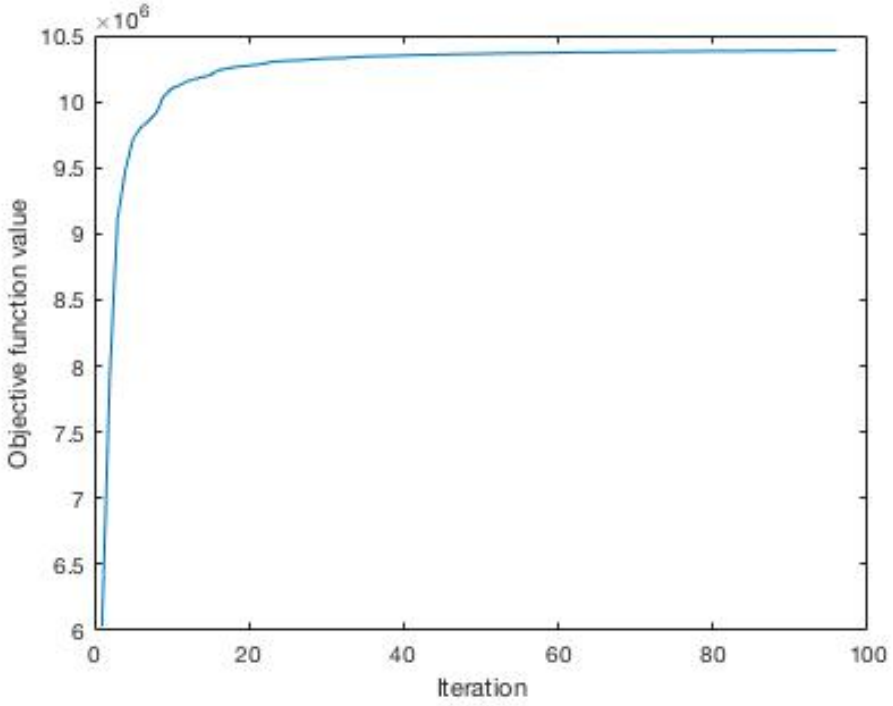

Figure 3: Convergence plot for one time step with initial guess 1.

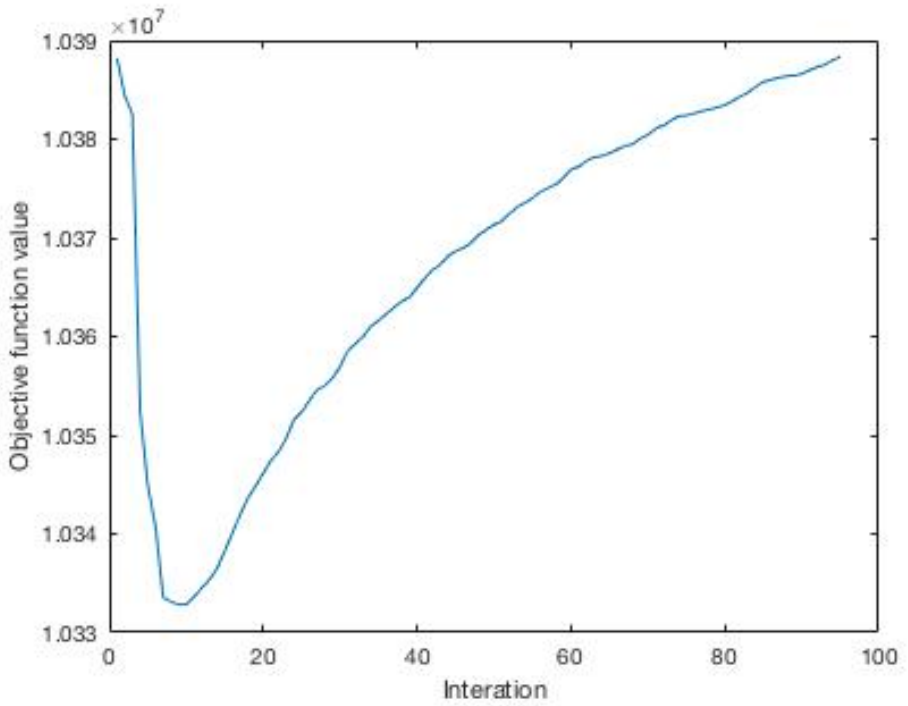

Figure 4: Convergence plot for one time with initial guess 6.

Changing the *FiniteDifferenceStepSize* option affects the difference between iteration steps. A larger value forces *fmincon* to take larger steps. The result for different *FiniteDifferenceStepSize* values for 25 time steps and with initial guess 1 is shown in Table 5. The default value is $\sqrt{\epsilon}$ where ϵ is the machine epsilon, the floating-point relative accuracy, and for MATLAB ϵ is $2.220446049250313 \cdot 10^{16}$. The most common reason for stopping for the values 10^{-3} and above was reaching the *MaxFunctionEvaluations* and for 10^{-2} it was the size of the current step.

FinDiffStepSize	Real time (s)	Constraint violation	Total energy consumption
$\sqrt{\epsilon}$	1088	$1.10398435 \cdot 10^{-3}$	$1.20547581 \cdot 10^{-7}$
10^{-7}	1044	$9.27606345 \cdot 10^{-4}$	$1.20615299 \cdot 10^{-7}$
10^{-6}	1140	$1.02209863 \cdot 10^{-3}$	$1.20722526 \cdot 10^{-7}$
10^{-5}	1114	$8.33399909 \cdot 10^{-4}$	$1.20552667 \cdot 10^{-7}$
10^{-4}	1171	$1.05156777 \cdot 10^{-3}$	$1.20626226 \cdot 10^{-7}$
10^{-3}	1727	$1.81792500 \cdot 10^{-3}$	$1.21078589 \cdot 10^{-7}$
10^{-2}	281	$3.45114694 \cdot 10^{-3}$	$1.20438112 \cdot 10^{-7}$

Table 5: Result and real time for different *FiniteDifferenceStepSize* values.

The *MaxFunctionEvaluations* was the most reached stopping criterion for *fmincon* in the original code from [1]. The default value for the *MaxFunctionEvaluations* is $2 \cdot 10^4$. It was

raised to investigate how it would affect the output and to see if a local minimum would be found. The result is seen in Table 6. For $2 \cdot 10^4$ maximum function evaluations, the solver stopped due to reaching *MaxFunctionEvaluations* for 24 of the time steps. For $3 \cdot 10^4$ evaluations it was one time step and for $4 \cdot 10^4$ evaluations it was down to zero.

MaxFunEvals	Real time (s)	Constraint violation	Total energy consumption
$2 \cdot 10^4$	1088	$1.10398435 \cdot 10^{-3}$	$1.20547581 \cdot 10^{-7}$
$3 \cdot 10^4$	1744	$1.06747104 \cdot 10^{-3}$	$1.21088150 \cdot 10^{-7}$
$4 \cdot 10^4$	2029	$1.66846973 \cdot 10^{-3}$	$1.20707163 \cdot 10^{-7}$

Table 6: Result and real time for three different maximum number of function evaluations.

When *fmincon* no longer stopped due to *MaxFunctionEvaluations* the next stopping criterion reached, the *StepTolerance*, was adjusted to see if a local minimum would be found. The result is seen in Table 7. For all tested values of the step size tolerance, the solver stopped due to the *StepTolerance* for all time steps.

TolX	Real time (s)	Constraint violation	Total energy consumption
10^{-6}	2029	$1.66846973 \cdot 10^{-3}$	$1.20707163 \cdot 10^{-7}$
10^{-8}	2276	$9.69050921 \cdot 10^{-4}$	$1.20978212 \cdot 10^{-7}$
10^{-10}	1752	$1.04887562 \cdot 10^{-3}$	$1.20989531 \cdot 10^{-7}$
10^{-12}	2232	$1.07935747 \cdot 10^{-3}$	$1.21098763 \cdot 10^{-7}$
10^{-14}	2768	$9.71074238 \cdot 10^{-4}$	$1.21173107 \cdot 10^{-7}$

Table 7: Result and real time for different step size tolerances.

To search a larger area of the objective function, different initial guesses for 25 time steps was tested and the result is seen in Table 8. No local minima were found for either of the initial guesses or time steps.

Initial guess	Real time (s)	Constraint violation	Total energy consumption
1	1088	$1.10398435 \cdot 10^{-3}$	$1.20547581 \cdot 10^{-7}$
2	1021	$8.66601159 \cdot 10^{-4}$	$1.20514169 \cdot 10^{-7}$
3	1030	$9.04615969 \cdot 10^{-4}$	$1.20648104 \cdot 10^{-7}$
4	1100	$8.82375355 \cdot 10^{-4}$	$1.20597904 \cdot 10^{-7}$
5	1092	$8.32923997 \cdot 10^{-4}$	$1.20660065 \cdot 10^{-7}$
6	1138	$8.51739682 \cdot 10^{-4}$	$1.20498406 \cdot 10^{-7}$

Table 8: Result for different initial guesses for 25 time steps

The tolerance on the constraint violation, *TolCon*, was set to 10^{-1} in the original code from [1]. Tests were done for different *TolCon* values with initial guess 1 and for 25 time steps to investigate its impact on accuracy and real time. The result is shown in Table 9.

TolCon	Real time (s)	Constraint violation	Total energy consumption	Infeasible points
1	227	$7.83196527 \cdot 10^{-2}$	$1.15468067 \cdot 10^{-7}$	0
10^{-1}	1088	$1.10398435 \cdot 10^{-3}$	$1.20547581 \cdot 10^{-7}$	0
10^{-2}	3107	$7.48453895 \cdot 10^{-4}$	$1.20643617 \cdot 10^{-7}$	7
10^{-3}	3556	$7.86600190 \cdot 10^{-4}$	$1.21049692 \cdot 10^{-7}$	13
10^{-4}	2353	$6.99329127 \cdot 10^{-4}$	$1.24358417 \cdot 10^{-7}$	10
10^{-5}	4135	$7.52098966 \cdot 10^{-4}$	$1.20736190 \cdot 10^{-7}$	10
10^{-6}	3740	$6.18640660 \cdot 10^{-4}$	$1.21008064 \cdot 10^{-7}$	11

Table 9: Table showing how TolCon influences the result and real time of the model.

In Table 10 a comparison between *fmincon* running non-parallel versus parallel is shown. The test was done over 25 time steps with initial guess 1.

	Real time (s)
Single core	1057
Parallel	1099

Table 10: Real time for running *fmincon* non-parallel and parallel.

4.3 Evaluation of the performance of global solvers

The results for tests with the implemented global solvers for the six initial guesses in Table 2, can be seen in Table 11. All tests were for one time step and the number of starting points for MultiStart was 50.

Initial	MultiStart		GlobalSearch		PatternSearch	
guess	CPU time (s)	fval	CPU time (s)	fval	CPU time (s)	fval
1	85.4	$1.03839548 \cdot 10^{-7}$	21370	$1.03373045 \cdot 10^{-7}$	110.1	$6.02967578 \cdot 10^{-6}$
2	70.0	$1.03863220 \cdot 10^{-7}$	8885	$1.02567204 \cdot 10^{-7}$	73.1	$7.69194733 \cdot 10^{-7}$
3	73.4	$1.03917929 \cdot 10^{-7}$	31385	$1.03568951 \cdot 10^{-7}$	64.3	$9.72387552 \cdot 10^{-6}$
4	67.9	$1.03833229 \cdot 10^{-7}$	24953	$1.03604080 \cdot 10^{-7}$	71.8	$1.11822343 \cdot 10^{-7}$
5	68.3	$1.03862863 \cdot 10^{-7}$	22623	$1.03602976 \cdot 10^{-7}$	131.4	$1.53267032 \cdot 10^{-7}$
6	71.7	$1.03579836 \cdot 10^{-7}$	29959	$1.03735833 \cdot 10^{-7}$	124.5	$1.33527398 \cdot 10^{-7}$

Table 11: Result for different initial guesses for one time step

The objection function value, *fval*, shown in Table 11 is not the objective function value provided from a local minimum found but is the best value found for each solver. No global minima were found for any of the solvers, but possible local minima were found for all solvers for some initial guesses due to the solvers stopping prematurely in feasible points. The runs, where possible local minima were found, are marked blue in the table. In the case of a blue value for both MultiStart and GlobalSearch, it was only one of all points investigated that resulted in a possible local minimum. The reason for stopping prematurely for GlobalSearch

and MultiStart was due to reaching *MaxFunctionEvaluations* or *MaxIterations*. For PatternSearch it was due to the mesh size being less than the mesh tolerance.

For the two solvers that can be run in parallel, MultiStart and PatternSearch, the time difference for a parallel and a non-parallel run is shown in Figure 5 in real time. The non-parallel versions of both solvers were matched so that the real time would be around the same for the solvers, to better compare the speedup. This meant one time step for MultiStart and 820 for PatternSearch. This resulted in a real time of $\sim 3400s$. The solvers were run with initial guess 1. The resulting speedup is approximately 10 for MultiStart and no speedup for PatternSearch.

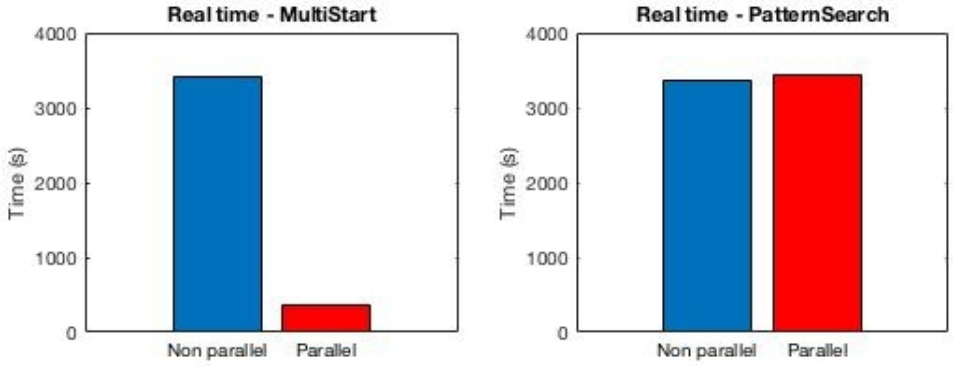

Figure 5: Real time for single core and parallel for MultiStart and PatternSearch.

4.4 Comparison of MultiStart and *fmincon*

A final test to compare the solvers *fmincon* and MultiStart is shown in Table 12. The tests were done for one time step and initial guess 1.

Solver	Real time (s)	Constraint violation	Total energy consumption
MultiStart	2487	$1.39456850 \cdot 10^{-3}$	$5.70491403 \cdot 10^{-6}$
fmincon	331	$1.86485444 \cdot 10^{-3}$	$5.65509923 \cdot 10^{-6}$

Table 12: Comparison of MultiStart and *fmincon* for ten time steps with initial guess 1.

5 Discussion

5.1 Improvement of the model

The interior-point algorithm was the fastest and resulted in the lowest total energy consumption, see Table 3. However, it also had the highest average total constraint violation

per server. The algorithm that gave the lowest average total constraint violation per server was active set, but was the slowest. We therefor found SQP to be the best algorithm for this problem, as in [1].

By doing the changes described in 4.1, we achieved a time reduction of approximately 21% in real time from the original code, see Table 4. Worth mentioning is that the function *spdiags* uses memory more efficiently than the combination of *sparse* and *diag* and is therefor usually the recommended choice. The time reduction achieved was deemed significant enough to sacrifice some memory and was therefore kept.

5.2 Evaluation of the performance of *fmincon*

The solver *fmincon* showed bad performance in time and accuracy in [1]. To investigate this behavior, a series of test were performed on the different input options for *fmincon*.

In Figure 3 for the first three iterations there is a steep increase in the objective function value. We suspect that the cause of this increase is that the corresponding parameter sets for these objective function values were less feasible and had a higher first order optimality, so the solver keeps searching. The search region *fmincon* then enters is flat and the solver stopped due to the *MaxFunctionEvaluations* tolerance. In Figure 4 the objective function values decreases for the first ten iterations to then increase again. The reason for it not stopping at the lowest function value we believe again is because of the feasibility and first order optimality. For each iteration, in both Figure 3 and Figure 4, the feasibility and first order optimality improved. Also, the changes in the function value might seem large in the plots but in full scale the changes in the objective function is small.

Table 5 shows three different values for the minimum change in the step length for *fmincon*. These tests were done to see if forcing the solver to take larger steps would explore a larger area of the objective function. However, these tests did not change the result much and the convergence history mostly remained the same for each test, except for the smallest value for the *FiniteDifferenceStepSize* 10^{-2}. For that value, *fmincon* stopped due to the size of the current step in feasible points for all time steps, resulting in faster run time since fewer iterations were made.

Three different values for the maximum number of function evaluations were tested, see Table 6. For the default value of $2 \cdot 10^4$ evaluations, most time steps stopped due to the number of function evaluations and for $4 \cdot 10^4$ evaluations, none of them did. Instead, the solver stopped due to the size of the current step being less than the step size tolerance. Between zero and five solutions were infeasible for each value of the step size tolerance where 10^{-10} was the only one with zero. Looking at the total average constraint violation per server and the total energy consumption, the values does not improve for neither the increase in function evaluations nor the decrease for the step size tolerance. This is due to the fact that no local minima were found resulting in the solver stopping at arbitrary points.

In Table 8 the six different initial guesses in Table 2 were tested for *fmincon* with 25 time steps. No local minima were found and the convergence history remained similar for all time steps and initial guesses.

The convergence history remained similar and no local minima were found for any tests. This indicates that the objective function has similar properties everywhere. From these tests the conclusion was drawn that the objective function is flat and also that it is not multi-modal since no local minima were found.

In Table 9 the average total constraint violation per server decreases with smaller values for *TolCon*. This is a expected result since a smaller tolerance in constraint violation causes *fmincon* to evaluate the found parameter set with a higher accuracy. This results in longer run times as well and also causes the region of feasible parameter sets to be smaller resulting in more infeasible points. From these tests it was decided to use the value 10^{-1} for *TolCon* in tests since it was relatively fast and had a smaller average total constraint violation per server compared to the value 1.

In Table 10 the result from running *fmincon* with and without parallel processing is seen. Running on a single core resulted in a run time of 1057s and running with twelve workers took slightly longer time, 1099s. In a different testing environment using two workers, an anticipated speedup of 1.5 was achieved. The reason for not getting any speedup in the testing environment chosen for this project might be that it is a Linux host which can hold several users. There might be limitations in place for users which affects the result. For example, there could be limits in how much of the CPU each user can utilize. However, a speedup was achieved with MultiStart, see Figure 5, which means that it is possible to run MATLAB with parallel processing in the testing environment. Though, *fmincon* and MultiStart do not use parallelism in the same way since MultiStart starts the different start points in parallel and *fmincon* only runs the computation of the gradient in parallel. This difference in parallelism usage might be the explanation to why the result differ for the two solvers.

Furthermore, *fmincon* is a deterministic solver, meaning that the result from one run using the same parameters on the same problem should result in the same solution. This was not the case for this problem set up, which suggest that something, like the objective function or the constraints, are stochastic. The cause of this might be the Simulink model, used to simulate the server temperatures. If the Simulink model is stochastic, it would cause the problem set up to be so as well since the server temperatures are used in the optimization.

When using a different testing environment, different solutions were produced. Using a different version of MATLAB, R2016b, for example, produced more infeasible points than the version in this project, R2018a. This said, the testing environment affects the result of the model in both timings and accuracy.

5.3 Evaluation of the performance of global solvers

Looking at Table 11, it was concluded that GlobalSearch can be excluded from the discussion due to it being time inefficient compared to the other solvers. Running GlobalSearch with multiple time steps would consume a lot of time since the solver can not be run in parallel in the same way as MultiStart. In a different testing environment with multiple cores, GlobalSearch using parallelized fmincon could result in speedup. From the tests done, the conclusion was drawn that the choice of global solvers to consider should be MultiStart and PatternSearch.

PatternSearch converges to infeasible points for three of the six tested initial guesses, see Table 11, and stops prematurely with possible local minimum for the other guesses. MultiStart on the other hand had four initial guesses resulting in premature stops where the constraints were not satisfied, and two which gave possible local minimum. No initial guesses resulted in only infeasible points. Comparing the result of speedup between the two global solvers, as Figure 5 show, MultiStart is the only solver where the parallel version resulted in speedup. This is likely due to PatternSearch only doing at most three iterations per time step. In parallel this makes a small difference in time, it can even add time due to the parallel pool, parpool, starting up. When running MultiStart in parallel the solver starts *fmincon* in different points for different workers, making the advantage of parallelism greater. Taking everything in consideration MultiStart can be seen as the best of the global solvers.

5.4 Comparison of *fmincon* and MultiStart

The two solvers to compare are the global solver MultiStart and the local solver fmincon. MultiStart starts *fmincon* in multiple points which takes around 8 times longer time, for 50 start points, then to run it only once which can be seen in Table 12. The total energy consumption is higher for MultiStart and since the difference in average total constraint violation per server between the solvers is not significant enough, the best solver for the problem is *fmincon*.

6 Conclusion

In this project the goal was to improve the model, evaluate the performance of *fmincon* and evaluate the performance of three global solvers. The results from improving the model showed that a bottleneck of the code could be found in the function *Dynamics*, concerning matrix builds. This was improved, resulting in approximately 21% time reduction.

The most time-consuming part of the model was the *fmincon* optimization routine. One of the reasons for *fmincon* being time-consuming was the behavior of the objective function which is flat, resulting in a high number of function evaluations. Changes of the initial guess and the *fmincon* options did not result in any improvement of the result, which further proves the flatness of the objective function. The three *fmincon* algorithms uses different

methods of iterating over the objective function and no minima being found, due to flatness, results in the solver stopping in arbitrary points.

When evaluating the global solvers it was found that MultiStart was the best solver for our problem. GlobalSearch was time inefficient and PatternSearch produced infeasible points resulting in bad performance. No global solution was found for any of the solvers.

By comparing the real time, the average total constraint violation and the total energy consumption for *fmincon* and MultiStart it was concluded that *fmincon* is the better solver for this problem.

6.1 Suggestion for future work

Based on the results of this project suggestions for future research is:

- Testing the model in a different testing environment with multiple cores.
- Implementing the model in a different programming environment or language, and usage of other solvers.
- Further relaxing of the constraints to expand the feasible region.

7 Bibliography

[1] Berglund, E. (2017). *LQR and MPC control of a simulated data center*. Master thesis, KTH Royal Institute of Technology, Sweden

[2] Camacho E.F. and Bordons C. (1999). *Model Predictive Control*, Springer-Verlag London Limited

[3] MathWorks. (2018). *fmincon* https://se.mathworks.com/help/optim/ug/fmincon.html

[4] MathWorks. (2018). *Constrained Nonlinear Optimization Algorithms* https://se.mathworks.com/help/optim/ug/constrained-nonlinear-optimization-algorithms.html

[5] MathWorks. (2018). *How GlobalSearch and MultiStart Work* https://se.mathworks.com/help/gads/how-globalsearch-and-multistart-work.html?s_tid=srchtitle

[6] MathWorks. (2018). *Nonlinear Constraint Solver Algorithm*. https://se.mathworks.com/help/gads/description-of-nonlinear-constraint-solver.html

[7] MathWorks. (2018). *Run Code on Parallel Pools* https://se.mathworks.com/help/distcomp/run-code-on-parallel-pools.html

[8] Kozola, S. (2009). *Improving Optimization Performance with Parallel Computing*. https://se.mathworks.com/company/newsletters/articles/improving-optimization-performance-with-parallel-computing.html

[9] *Serial and parallel approaches to gradient estimation* (2009). Accessed January 2019: https://se.mathworks.com/company/newsletters/articles/improving-optimization-performance-with-parallel-computing.html

[10] *Parallel processing work flow* (2018). Accessed January 2019: https://computing.llnl.gov/tutorials/parallel_comp/