UPPSALA
UNIVERSITET

# Parallel Global Optimization of ABB's metal process models using Matlab

Adam Saxén, Karl Bengtsson Bernander

**Project in Computational Science: Report**

January 2014

PROJECT REPORT



Institutionen för informationsteknologi

# Contents

# 1 Introduction

The focus of this report is on analysing three of Matlab's global optimization solvers when applied to a mathematical model of an industrial process called hot rolling. The problem to be optimized can be described as a non-convex, non-linear optimization problem with linear and nonlinear constraints. The report covers implementation of the solvers and performance analysis.

## 1.1 Goal

Our goal is to investigate three of Matlab's global optimization solvers when applied to ABB's ADM (Adaptive Dimension Model), and evaluate their performance in terms of accuracy, execution time and parallel speedup; accuracy measures how consistent a solver is in finding the global minimum; an accuracy of 50 % means that the global minimum is found in 50 % of the runs.

Choosing a solver that performs well is challenging, where the best configuration heavily depends on the problem at hand. This research is an important step to achieve a larger goal set by ABB, to be able to perform high performance optimization on the cloud.

## 1.2 The ADM - Hot rolling

Adaptive Dimension Model, for short ADM, is a thermo-mechanical model which describes hot rolling, a type of metalworking process. During the process metal slabs are heated and the reshaped by passing the metal through a series of stands with pairs of rolls. The end product consists of long pieces of cooled metal, fitted for a new purpose.

As with all industrial processes the goal is to produce a good quality product while minimizing production cost. There are many parameters for the process, such as roll gap and rolling speed, and these are referred to as the rolling schedule. Finding a suitable schedule, in terms of e.g. low power consumption, proves hard and costly if done by trial and error. The ADM helps the operators to find the optimal rolling schedule based on what asset they want to optimize.

# 2 Theory

## 2.1 Optimization

In this report optimization refers to numerical optimization, i.e a mathematical technique to find optima, minimum/maximum values, of a function. The problem to be optimized is defined by an objective function, f(x), subject to a set of constraints that limit the domain of acceptable vectors of optimization variables $x$. The standard optimization problem, by convention, is defined as a

minimization problem

$$\begin{aligned} & \min_x f(\mathbf{x}) \\ \text{s.t } & G_i(\mathbf{x}) = 0, \quad i = 1, \cdots, m \\ & H_j(\mathbf{x}) \leq 0, \quad j = 1, \cdots, n \\ & \mathbf{x}_l \leq \mathbf{x} \leq \mathbf{x}_u \end{aligned} \tag{1}$$

where $G_i$ are equality constraints and $H_j$ are inequality constraints. These constraints can be linear or nonlinear, where the last constraint of eq.(1) shows the upper and lower bounds of x which is linear. A vector $\mathbf{x}_l \leq \mathbf{x} \leq \mathbf{x}_u$, that satisfies $G_i$ and $H_j$ $\forall i, j$ is called a feasible point. All points that are feasible belong to the feasible set or feasible region and is denoted $D$.

A feasible point is called a local optimizer $x^*$ if $f(x^*) \leq f(x)$ holds for all $\mathbf{x}$ in the feasible region confined by $|x - x^*| \leq \delta, \quad \delta > 0$ (i.e. in some region around $x^*$ all function values are greater than or equal to the value at the optimizer). A global optimizer has the same definition but the confined region will be equal to the entire feasible set $D$. An optimum is referred to as the pair $(x^*, f(x^*))$. Figure 1 illustrates these concepts in one dimension.

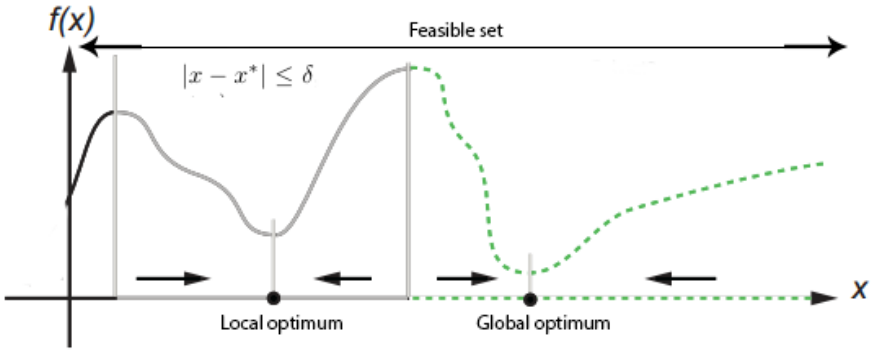


Figure 1: Global optimum and local optimum for an arbritary function with a feasible set in one dimension

Optimization problems are classified depending on the characteristics of the objective function $f(x)$ and the constraints $G_i, H_j, x_l \leq x \leq x_u$. For example if no constraints exist then eq. (1) is referred to as an unconstrained optimization problem, otherwise the problem is known as a constrained optimization problem. Nonlinearity and linearity of $f(x)$ and the constraints are an important part when classifying the problem. When linear constraints and a linear objective function exist the problem can be solved with well studied methods like the Simplex Method.

Nonlinear problems on the other hand require a more sophisticated approach. The method of Lagrangian multipliers is one example of a method that handles nonlinear $f(x)$ and nonlinear constraints. The method finds feasible points that fulfill the first order optimality conditions, i.e. a neccessary but not suf-

ficient condition for optimality. The conditions are called the Karush Kuhn Tucker(KKT) conditions[1]

$$\nabla f(x) + \sum_{i=1}^{m} \lambda_i \nabla G_i(x) + \sum_{j=1}^{n} \mu_j \nabla H_j(x) = 0 \tag{2}$$

$$G_i(x) = 0, \quad i = 1, ..., m \tag{3}$$

$$H_j(x) \leq 0, \quad j = 1, ..., n \tag{4}$$

$$\mu_j \geq 0, \quad j = 1, ..., n \tag{5}$$

$$\mu_j H_j(x) = 0, \quad j = 1, ..., n \tag{6}$$

Lagrangian multipliers $\lambda_i$ and $\mu_i$ make it possible to combine the contraints $G_i$ and $H_i$, with the objective function $f(x)$. Equation(2) implicitly states that the gradients of $f(x)$ and the gradients of the constraints are parallel; this does not imply that the vectors are of the same magnitude, thus $\lambda_i$ and $\mu$ are required and scales the gradients in order to fulfill the condition. $\mu_i$ regulates when the constraints $H_j$ are active(i.e. act as equality constraints). If an optimizer $x^*$ activates a constraint $H_j$ the gradients,$\nabla f(x)$ and $\nabla H_i(x)$, with respect to $x$ have opposite directions which implies that $\mu$ must be positive. The last eq.(6) is the complementary slackness condition.

The ADM solves constrained nonlinear optimization problems and tries to find a local optimum, i.e. sub-optimal rolling schedule (optimizer) and optimal objective function. The optimization variable x is a 33-dimensional vector. This classification determines the choice of solution methods.

## 2.2 Optimization methods

There exists multiple methods for finding optimum points, such as evaluating gradients in order to descend towards the optimum or by computing and comparing function values according to certain patterns. Regardless of what method is used, an initial point $x_0$ needs to be supplied. This point will act as the starting point from where the method descends towards the optimum; $x_0$ should be feasible.

### 2.2.1 Gradient-based methods

As the name suggests, these methods use the first and second derivative, Jacobians and Hessians, to find local minima. There are several methods where two will be mentioned in this report, namely Sequential Quadratic Programming(SQP) and the Active-set method. These methods are well suited to solve the constrained nonlinear optimization problems of this project.

Both SQP and Active-set solve a sequence of Quadratic Program subproblems in order to find a local optimum. The methods use the Lagrangian method with KKT eq.(2) and transforms the original nonlinear optimization problem into a

[1](Kuhn, H. W.; Tucker, A. W. (1951). "Nonlinear programming". Proceedings of 2nd Berkeley Symposium. Berkeley: University of California Press. pp. 481-492. MR 47303)

Quadratic programing subproblem

$$\begin{aligned} & min_d \frac{1}{2} d^T H_k d + \nabla f(x_k)^T d \\ & s.t. \quad \nabla G_i(x_k)^T d + G_i(x_k) = 0, \quad i = 1, ..., m \\ & \qquad \nabla H_j(x_k)^T d + H_i(x_k) \leq 0, \quad j = 1, ..., n \end{aligned} \tag{7}$$

The subproblem is an approximation obtained by linearizing constraints and approximating the modified objective function (lagrange function) quadratically. For every iteration $k$ a new Hessian $H_k$ is computed and the subproblem (7), is solved. The search direction $d_k$ is used to form a new point closer to the local optimum:

$$x_{k+1} = x_k + d_k$$

The process is iterated to construct a sequence of subproblems that in theory will converge to a local optimizer $x^*$.

Gradient-based methods are limited by the requirement that the objective function and constraints are continuous and have continuous gradients. The ADM exhibits discontinuities that are made continuous using a smoothing technique. The next type of methods called Direct search methods don't need the gradient or Hessian when finding an optimum.

### 2.2.2 Direct Search methods

Direct Search methods do not use any information about the gradient. Instead, they systematically search the area near the current point, looking for feasible points with lower objective function value.[2] There are many different techniques of searching the feasible set, where the choice depends on the problem at hand. In general, these methods can be preferred in situations where the objective function is not differentiable. However, since no directional information is used from the gradient, the methods could exhibit poor convergence to the global minimum.

### 2.2.3 Challenges

In non-convex optimization a hard problem is to determine if an optimum also is the global optimum. There are several ways to solve this problem, but in short it comes down to searching the set $D$ as thoroughly as possible. This could, for example, mean applying a method multiple times for multiple initial points and comparing local optima, or based on a single initial point search $D$ in a manner that efficiently covers the entire feasible set. Regardless of how the search is done the found solution will with higher probability be the global optimum.

There are a vast set of methods available. To solve an optimization problem with one method can be straight forward, but finding and tuning a method that solves it efficiently is a challenge, and requires indepth analysis and testing of

[2] `http://www.mathworks.se/help/gads/what-is-direct-search.html`

many methods applied to the problem. There exists no shortcut, every new optimization problem needs to be analyzed separately.

## 2.3 Optimization methods in Matlab

The ADM is implemented in Matlab and is solved with a local method called *fmincon*. This predetermines that all methods used in this report will run in Matlab. There exists a couple of Matlab toolboxes that contain optimization methods, and they will be referred to as solvers in this report.

### 2.3.1 Local optimization solver - fmincon

The fmincon solver is a gradient-based method used to find the local minima of constrained non-linear multi-variable functions. Fmincon begins from an initial guess, iterates according to a given update scheme, and finishes when a stopping criteria is met. The final iteration is a local minimum if the first-order necessary and second-order sufficient conditions are fulfilled, where the second-order condition states that the Hessian must be positive definite. It is worth to note that fmincon often can handle an infeasible starting guess by solving a linear programming problem for the initial guess and the constraints.

The update scheme depends on the sub-algorithm used by fmincon, of which there are four: SQP, Active-set, interior-point and trust-region-reflective. The last requires a pre-defined gradient, which does not exist in the ADM, and is therefore not explored. All schemes rely on using the gradient to find a descent direction.

SQP (Sequential Quadratic Programming) and Active-set both use the Quadratic program subproblem (7) that approximates an optimization problem transformed by the Lagrangian method and KKT eq.(2). The sub-algorithms follow the same procedure:

1. Compute the gradient and update the Hessian $H_k$ of eq.(7) using the BFGS update scheme.
2. Set up and solve a quadratic program to find descent direction
3. Perform line search to find a proper step length in the descent direction

The Hessian $H_k$ is computed using a quasi-Newton update scheme called BFGS[3]. The method is called quasi-Newton since it does not computed $H_k$ exactly but approximate it with BFGS:

$$H_{k+1} = H_k + \frac{q_k q_k^T}{q_k^T s_k} - \frac{H_k s_k s_k^T H_k^T}{s_k^T H_k s_k} \tag{8}$$

[3]Broyden-Fletcher-Goldfarb-Shanno

where

$$
\begin{aligned}
s_k &= x_{k+1} - x_k \\
q_k &= \nabla f(x_{k+1}) - \nabla f(x_k) && (9) \\
&= \nabla f(x_{k+1}) + \sum_{i=1}^{m} \mu_i \nabla G_i(x_{k+1}) + \sum_{j=1}^{n} \lambda_j \nabla H_j(x_{k+1}) && (10) \\
&\quad - \Big(\nabla f(x_k) + \sum_{i=1}^{m} \mu_i \nabla G_i(x_k) + \sum_{j=1}^{n} \lambda_j \nabla H_j(x_k)\Big)
\end{aligned}
$$

Solving the subproblem is done in two steps. First feasibility is checked, if it fails, a feasible point is calculated by solving a linear programming problem for the point and the constraints. The second step involves the generation of an iterative sequence of feasible points that converge to the solution. Finally, a line search is used to take an appropriate step length by scaling the descent direction $d_k$ with $\alpha$ for the next iterate:

$$x_{k+1} = x_k + \alpha d_k \tag{11}$$

Both subalgorithms ensure that the Hessian $H_k$ is positive definite by setting the initial $H_0$ Hessian to be positive definite.

There are some differences between SQP and Active-set; Active-set can take intermediate infeasible steps outside bounds, which could be unbeneficial if the objective function or constraints are undefined or complex outside the feasible region; SQP can, during its iterations, take a step that fails. This means that an objective function or constraint returns the value Inf, NaN or a complex value. If this happens SQP will attempt to take a smaller step. The linear algebra routines used to solve the QP subproblem are more efficient in terms of memory usage and speed for SQP than for Active-set.

The interior-point subalgorithm approaches the solution from interior of the feasible region, by replacing the QP with a barrier method which is solved for decreasing $\mu$.[4]

At every iteration the algorithm can use one of two steps to find an optimum, called direct step and conjugate gradient step. The direct step tries to solve the KKT eq:(2) via a linear approximation. Conjugate gradient step uses a trust region approach where $f(x)$ in eq:(1) is approximated with a new function $p$ that behaves similar to the original function $f(x)$ in a neighbourhood $N$ of the current point $x$. This neighbourhood is called the trust region.

Fmincon has a parallel mode for the computation of gradients. For more information about fmincon and its subalgorithms see footnote.[5]

[4] Optimization toolbox user guide 2013, Mathworks

[5] `http://www.mathworks.se/help/optim/ug/fmincon.html` and `http://www.mathworks.se/help/optim/ug/choosing-a-solver.html`

### 2.3.2 Global optimization solvers - MultiStart, GlobalSearch and Patternsearch

Multistart, as the name implies, runs fmincon from multiple starting points. The results of each fmincon run is saved in a vector, and in the end the best result is selected as the global minimum. Multistart can use other local solvers beside fmincon, such as fminunc and lsqnonlin, but only fmincon will be considered since its the only local solver that handles constrained problems. Parallelization of Multistart performs well since the individual runs are independent; the different runs can easily be divided among multiple CPU:s.

The solver executes a number of starting points $k$. The points are supplied by creating $k-1$ random uniformly distrbuted points within the upper and lower bounds of the optimization problem, without necessarily fulfilling the constraints. The last point $x_0$ is supplied as an parameter to Multistart. For the ADM, with its many nonlinear constraints, the generated points are most likely not feasible.

GlobalSearch, like Multistart, uses fmincon to find the global minimum, but in a more sophisticated way. The solver needs an initial starting point $x_0$ and a set of potential starting points, generated with a scatter-search algorithm. The solver scores a subset of the points by running a score function, defined as the sum of the objective function at a point and a multiple of the sum of the constraint violations. GlobalSearch will initially run fmincon for two points, $x_0$ and the subset point $x_{s1}$ with the lowest score.

In the next step GlobalSearch creates two basins, assumed to be spherical, with $x_0$ and $x_{s1}$ as centers. Two counters are associated to the solver and keep track of how many points lay within a basin of attraction and how many that have a score function greater than a certain threshold $\mathbf{t}$. The solver can now determine if a trial point $\mathbf{p}$ should be evaluated by fmincon. The following criteria need to be fulfilled in order to run fmincon for point $\mathbf{p}$:

- $\mathbf{p}$ can't be in any existing basin
- the score of $\mathbf{p}$ must be less than the threshold $\mathbf{t}$

GlobalSearch repeatedly examines the list of trial points and throws away points that don't generate a better solution. It will finish when it runs out of trial points or when a predefined maximum time is reached. Globalsearch has no built in support for parallelism, but supports the parallelism of fmincon.

In Matlab the direct search methods are implemented in the Patternsearch solver, named after how it searches for new points in a pattern around the current point. This pattern is made up of a set of vectors which can be fixed or randomized for each iteration, depending on the search algorithm. The vectors are multiplied with a scalar multiple, generating a mesh of so called candidate points. The next point is selected by testing for feasibility and a lower objective function value. Either the first successful candidate point is chosen, or the lowest among all is chosen (complete polling). If no point is feasible or has lower objective function value, the mesh is contracted. If a feasible point with lower

objective function is found ($x_{success}$), the next point becomes the $x_{success}$ and the mesh is expanded. This continues until the stopping criteria are met.

Patternsearch is highly customizable with parameters controlling e.g. the search algorithm, complete polling and mesh contraction/expansion rates. Also, a poll and search algorithm can be used in a consecutive manner to further improve the search. If the poll method finds a successful point, the search method will be skipped.

If non-linear constraints are present, the Augmented Lagrangian Patternsearch is used, where nonlinear and linear constraints are handled separately. The nonlinear constraints are combined with the objective function and penalty parameters using the Lagrangian method to formulate a subproblem. The solver then minimizes a series of subproblems, updating the Lagrangians or penalty factor depending on whether or not a solution is found. The initial penalty and increase factors are input parameters to Patternsearch [6].

# 3 Method

The method for reaching our goal can be viewed as consisting of the following steps:

- Implement the three global solvers as a framework in Matlab
- Analyse fmincon and establish a reference solution
- Analyse - parameter selection, testing, accuracy, speedup etc
  - MultiStart
  - GlobalSearch
  - Patternsearch

Parameter selection means systematically exploring the parameter space for each global solver; to find and focus on performance dependent parameters.

There are two objective functions that will be used during the project: objective function 63 and objective function 71. They represent power per production speed and grain size, respectively.

## 3.1 Matlab framework

Since ADM is implemented in Matlab, it can utilize the built-in toolboxes for optimization and parallelization. The framework also uses mex files which allow non-Matlab code (C,C++,python) to be invoked in the model; certain functions are computationally heavy and are improved by being replaced by mexed files. Lastly, the framework contains three toolboxes shown in Table 1.

[6] `http://www.mathworks.se/help/gads/description-of-the-nonlinear-constraint-solver_bqf9jvg-1.html`

Table 1: The toolboxes used and what they contain

| Toolbox | Description |
|---|---|
| Optimization toolbox | Used for finding local minimums (fmincon) |
| Global optimization toolbox | Contains Global solvers |
| Parallel computing toolbox | Used to run solvers in parallel |

### 3.1.1 Testing environment

The framework was run on a server at the IT Department, Uppsala University. All results presented in this report have been created with the following computer specifications:

- CPU: AMD Opteron (Bulldozer) 6282SE, 2.6 GHz, 16-core, dual socket
- Memory: 128 GB
- Operating system: Scientific Linux 6.3
- Matlab 2012b

Uppsala University has a license for running up to 12 workers in parallel.

### 3.1.2 The Run framework

The run framework was developed in this project to enable the user to easily switch between different global solvers and change the degree of parallelization, without having to alter the code. The user configures a 'run' by assigning a set of parameters in *run_setup.mat*. The possible parameters to vary are shown in Table 2. The user can define several 'runs' that the framework then will perform unsupervised. To start the program the user types *Start_Run_Framework.m*. The number of optima found, execution time and more results are stored in *run_results.mat*.

In Table 2, *Id* is a number for keeping track of a specific test, *parallel* is how many workers are used, and *solver* is one of the three global solvers. *Smooth* and *relaxation* are parameters relating to the physics in the ADM; where *smooth* is how much discontinuities should be smoothed and *relaxation* is how strict the equality constraints are. The rest of the parameters are specific to fmincon or for each global solver. More information about what the parameters are can be found in Mathwork's Global optimization User guide 2013.

Table 2: Possible parameter selection for the global optimization solvers in 'the run framework'

| **General** | **Multistart** | **GlobalSearch** | **Patternsearch** |
|---|---|---|---|
| id | ms_numInitPt | gs_basinRadiusFactor | ps_cache |
| solver | | gs_numStageOnePoints | ps_completePoll |
| algorithm | | gs_numTrialPoints | ps_completeSearch |
| tolX | | gs_penaltyThresholdFactor | ps_initialPenalty |
| tolFun | | gs_maxWaitCycle | ps_meshAccelerator |
| tolCon | | gs_startPointsToRun | ps_meshContraction |
| parallel | | gs_distanceThresholdFactor | ps_meshExpansion |
| maxIter | | | ps_penaltyFactor |
| smooth | | | ps_pollMethod |
| relaxation | | | ps_searchMethod |
| maxFunEvals | | | ps_tolMesh |
| tolBind | | | |

The choice of parameters to test is based on what is believed to affect the performance of the solvers, after studying how the solvers work according to Mathwork's global optimization toolbox user guide 2013.

### 3.1.3 Solver exitflags

Exitflags are an output from the solvers implemented in Matlab. It is an integer identifying the reason why a solver terminates. For example an exitflag of 1 when running fmincon indicates that the solver exited because of a local optimum was found. There are several integers for different solvers and algorithms, but in general a positive exitflags indicates a successfull run with a local or global minimum found. Negative integers mark that no minimum was found, not finding feasible points, or errors occuring in the solver. More information about exitflags can be found in Mathworks global optimization toolbox and optimization toolbox user guides.

## 3.2 Establish a reference solution

A reference solution is needed to evaluate the performance of the solvers. Before this is done fmincon needs to generate reliable results, since it is a part of both MultiStart and Globalsearch. The default parameter settings for the ADM are used and the tolerances and optimization algorithms are varied to see how the local solution is affected. The ADM automatically generates a feasible initial point $x_0$ and since fmincon later, through global solvers, runs for multiple feasible and most likely infeasible random points, there is an interest to see how the algorithms affect execution time of fmincon for different feasible and infeasible $x_0$. Running Mulitstart for 10 points, $x_0$ and 9 random start points, and taking the average execution time of succesfull runs(i.e runs that return a positive exitflag) an performance estimate of the algorithms can be done.

Then Multistart will run with the found parameter settings for a very large number of starting points, both for objective function 63 and 71; Mathworks recommends increasing the number of start points as a thourough approach for finding a better solution.[7] GlobalSearch will also be used for a similar amount of trial points. This should increase the probability of finding the global optimum, since Multistart and Globalsearch generate and test the points in different ways.[8] The global reference solution will be regarded as the best global minimum found, either by Globalsearch or Multistart.

## 3.3 The solvers

The first priority when analysing the solvers is to find parameters that result in high accuracy, i.e. consistent in finding the global minimum. Then comes execution speed and finally speedup, where the solvers parallel scalability will be tested.

### 3.3.1 MultiStart

Multistart runs fmincon for each of the **n** random initial values. The number of **n** needed to find the global minimum with high probability is then calculated. This is done with a convergence test, i.e. determining the distribution of how many points converge to the global optimum, how many fail etc. This also yields information about the accuracy of Multistart.

Parallelization is built into Multistart and runs several fmincon calls simultaneously, one for each worker. The support is activated by setting the parameter *parallel* in *run_setup.mat* to a number larger than 0. The value represents the number of workers to be used; where 0 means serial execution, 1 means serially but in matlabs parallel environment, etc. Since multistart generates random initial points several tests were done, and the results were averaged.

### 3.3.2 GlobalSearch

Being a more sophisticated solver, GlobalSearch has more parameters to explore. By systematically varying interesting parameters according to Table 2, the solvers performance could be analyzed. Unlike MultiStart, Globalsearch doesn't have built-in parallelization support, but it uses fmincon which can parallize the computations of the gradients.

### 3.3.3 Patternsearch

Patternsearch has a large amount of parameters, many with large ranges, meaning a huge parameter space. Two approaches to investigate this space were developed: one systematic, and one stochastic by a custom-made genetic algorithm. The relevant parameters that were to be investigated are listed in Table 2.

[7] Mathworks Global Optimization Toolbox Users's Guide (2013b) - Page 3-64
[8] `http://www.mathworks.se/help/gads/how-globalsearch-and-multistart-work.html`

For the systematic approach, parameters thought to have the largest impact on the performance of patternsearch were varied, while the others were kept constant. For the stochastic approach, a genetic algorithm was developed. It works by randomizing sets of parameters, running each in patternsearch, comparing found objective function values, and letting the best ones mate for the next generation. After running for a number of generations, the fittest parameters for finding the global optimum consistently should have been found. For a more detailed explanation, see Algorithm 1.

**Result**: Sorted list of objective function values along with their coordinates and parameters for patternsearch

Initialize N individuals, where each individual has a randomized set of parameters for patternsearch within some bounds. The same initial guess is used for all individuals;
**for** *For M generations* **do**
  **for** *For N individuals* **do**
    Execute patternsearch for the current individual;
  **end**
  Sort all individuals by lowest objective function value.
  Natural selection: select the best individuals, in terms of lowest feasible objective value. Let them mate, exchanging their parameters randomly for the individuals in the next generation. Introduce mutations to preserve genetic diversity.
**end**

**Algorithm 1:** Genetic algorithm for patternsearch

# 4 Results

The objective function values that are presented in Results are scaled values with physical interpretation that is not discussed in this report. The reader should only focus on the relative sizes of the numbers.

## 4.1 Establish a reference solution

The local solver fmincon, run through Multistart with 1 feasible inital point $x_0$ and 9 random bounded points, converged to **0.37792** for obj. 71 and to **0.60868** for obj. 63, regardless of what algorithm was used. The number of decimals indicate the accuracy of what all algorithms could achieve. When considering the mean execution time of the three algorithms, Active-set was the fastest, then SQP and lastly Tnterior-point, as seen in Fig 2. Since Interior-point was considerably slower, sometimes up to twice the execution time when minimizing obj. 63 or obj. 71, and exhibited a higher function value at the optimum (compared to the 6th decimal); a decision was made to exclude the algorithm from further analysis.

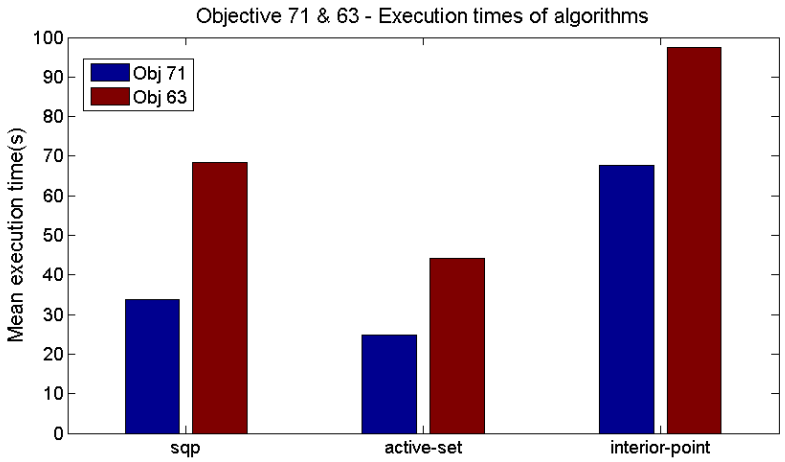


Figure 2: Comparison between fmincon algorithms SQP, Active-set and interior-point with default settings. One feasible inital point $x_0$ and 9 randomly generated points ,satisfying bounds, were used in Multistart.

When studying how to set the stopping criteria (controlled by a number of parameters) for fmincon it was found that the default tolerance settings in Matlab yielded good results, see Figure 3 and Table 3. All of the tolerances were varied by an equal amount and the result showed that the value of the local minimum drastically increased for tolerances of $10^{-5}$ and larger; the other subplots show how the solution behaves when varying one tolerance at a time while setting the rest to default. The exitflags were positive (1 or 2) for all tolerances below $10^{-5}$. For higher tolerance values the solver would eventually end with a negative exitflag.

Table 3: Default tolerances for fmincon, with function as stopping criteria. Constraint tolerance is how strictly the constraints have to be fulfilled, function tolerance is the difference in objective function evaluations between iterations, X tolerance is the difference in the coordinates of the point between iterations.

| **Tolerance** | **Value** |
|---|---|
| Constraint tolerance (TolCon) | $10^{-6}$ |
| Function tolerance (TolFun) | $10^{-6}$ |
| X tolerance (TolX) | $10^{-6}$ |

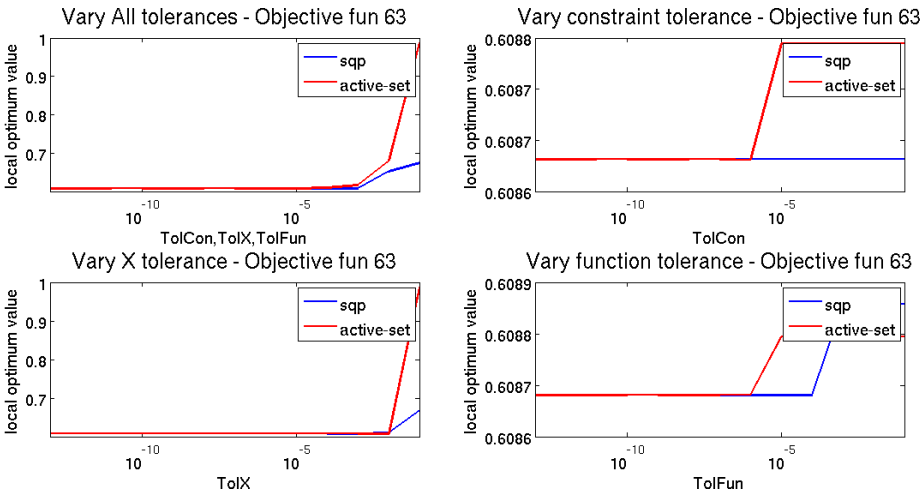


Figure 3: Tolerance test to analyze the impact of constraint, objective and x tolerance on the solution

Execution times are stable for tolerances below $10^{-5}$ and drop considerably for values larger than $10^{-5}$. Plots are found in appendix section (B).

When searching for a global reference solution, fmincon was configured to use Active-set and default settings. Active-set had a good execution time and an ability to find the best local minimum regardless of which objective function was used. MultiStart ran for 10000 random initial points, uniformly distributed between the bounds of the input variables, but not necessarily feasible with respect to the other constraints; see section 3.3.1. Globalsearch ran for 1000 stage 1 points and 5000 trial points, generated by the scatter-search algorithm. The results are found in Table 4.

Table 4: Found reference solutions for obj 71 and obj 63

| **Objective function** | **MultiStart** | **GlobalSearch** | **Reference solution** |
|---|---|---|---|
| 71 | 0.3779 | 0.3779 | 0.3779 |
| 63 | 0.6087 | 0.6087 | 0.6087 |

## 4.2 Gradient-based global solvers

### 4.2.1 Multistart

All of the multistart tests used initial guesses from a random uniform distribution between the bounds of the input variables, but they were not necessarily feasible with respect to the other constraints; see Section 2.3.2.

The first Multistart test was designed to determine the distribution of converged solutions, given a set of starting points. It is seen in Figure 4 for both

objective functions with the SQP and Active-set sub-algorithms, using 10000 initial guesses. It is clear that more than 80 % find the global optimum up to the 4th decimal (exitflag 1), with the exception of Active-set for objective function 63, where the corresponding number is 37 %. However, for the remaining points, 46% converge to the global optimum to within the 3rd decimal. The stopping criteria could probably be adjusted to bring these up to the 4th decimal as well, bringing the success rate up to over 80 % in this case as well.

Finally, speedup was tested for both algorithms and sub-algorithms, using 120 initial guesses. The test was repeated 5 times, and the mean execution times were analyzed. Looking at the results in Figure 5, it is clear that sub-linear speedup(i.e. speedup that does not scale linearly with the number of workers) is achieved in all cases, with a maximum speedup of around 10x for 12 workers. Speedup was also tested for fewer initial guesses; for 24 initial guesses, maximum speedup was reduced to around 5x for 6 workers.

The mean execution times for one worker with 120 initial guesses are found in Table 5.A large number of points were used, so the distribution of failed runs were approximately the same as for the convergence test (around 17 %) for all five tests. The maximum coefficient of variation (standard deviation of the 5 tests divided by mean) of execution times (for a fixed number of workers) was around 5 %.

Table 5: MultiStart execution times in seconds for 120 points, one worker, mean values of five separate runs.

| **Problem / Algorithm** | **SQP(s)** | **Active-set(s)** |
|---|---|---|
| 71 | 2350 | 2100 |
| 63 | 5100 | 3800 |

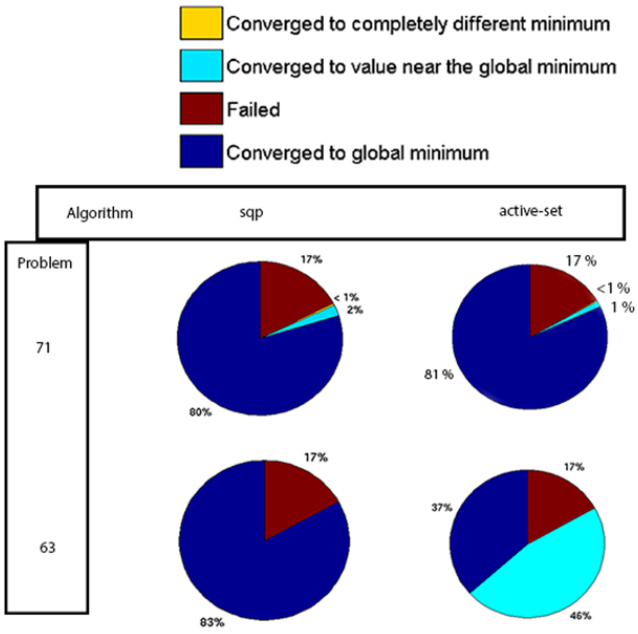


Figure 4: Multistart convergence distributions. 10000 initial guesses were used. In this case, convergence to global minimum was defined to having a correct 4th decimal when rounded; converging to near the global optimum was defined to having a correct 3rd decimal when rounded. A very small number of completely different minima were found; these had objective function values of around 2 times the global optimum, with positive exitflags greater than 1.

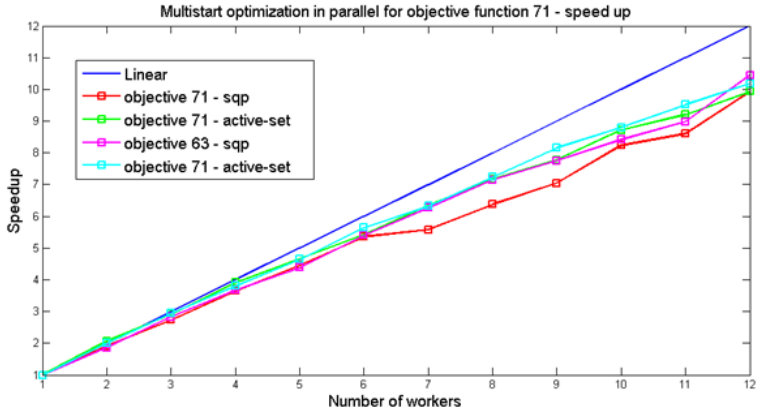


Figure 5: MultiStart speedup for 120 points. The values shown are mean values of 5 separate tests.

### 4.2.2 GlobalSearch

Globalsearch unfortunately terminated erroneously when using mex-files. The solver always executed fmincon for the initial guess, evaluated a few objective functions, and then stopped with a message saying that 'the execution was successful', 'that fmincon had run once', and that 'all trial points have been analyzed'. Since fmincon should be run at least twice in the globalsearch algorithm, something failed. When the mex-files were removed, the solver worked as expected, but was slow(up to several days for many initial guesses), meaning that the time constraints of the project would reduce the number of parameters that could be tested. It was decided to focus on the evaluation of the accuracy and speedup of Globalsearch; fine-tuning to see if execution times could be reduced was given a low priority.

The solver always found the global optimum for all our test and was therefore deemed as an accurate solver. Speedup was tested for a small number of points (50 stage 1 points and 150 trial points); the resulting mean values for 5 separate tests can be seen in Figure 6. Execution times are shown in Table 6. The maximum coefficient of variation (standard deviation of the 5 tests divided by mean) of execution times (for a fixed number of workers) was around 10 %.

Table 6: Globalsearch execution times in seconds for 120 points, 1 worker, mean values of 5 separate runs.

| **Problem / Algorithm** | **SQP(s)** | **Active-set(s)** |
|---|---|---|
| 71 | 1650 | 1250 |
| 63 | 5200 | 4050 |

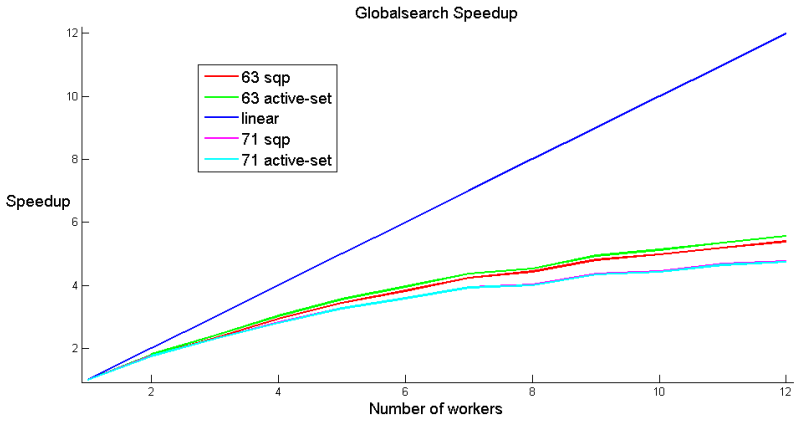


Figure 6: Speedup for globalsearch. For each number of workers, 50 stage 1 points and 150 trial points were used in globalsearch. This was done 5 times and the mean values are shown. The maximum coefficient of variation was around 10 %.

## 4.3 Patternsearch

### 4.3.1 Systematic approach

For the systematic approach, the default parameters for patternsearch were used, parameters were varied one by one to see how the solution varied. Figure 7 illustrates a common result when running patternsearch for obj. 71 with default settings and a feasible initial guess. It serves as a good example when explaining parameter selection.

| Iter | f-count | f(x) | constraint | MeshSize | Method |
|---|---|---|---|---|---|
| 0 | 1 | 0.995463 | 1.669e-16 | 1 | |
| 1 | 20 | 0.995463 | 1.669e-16 | 0.001 | Increase penalty |
| 2 | 8595 | 0.53696 | 0.0002947 | 1e-05 | Increase penalty |
| 3 | 14188 | 0.529401 | 6.486e-05 | 1e-07 | Increase penalty |
| 4 | 26267 | 0.529053 | 8.712e-07 | 1e-09 | Increase penalty |

Figure 7: A common result for patternsearch when run on obj. 71

**Mesh tolerance (TolMesh)**
Mesh tolerance is the parameter that causes patternsearch to stop, fulfilling the stopping criteria, i.e. the sub-problems continually are solved until the mesh size is below the tolerance value. In Figure 7 the number of iterations can easily be extended by increasing TolMesh. Execution time was roughly 700s; decreasing the TolMesh to $10^{-20}$ resulted in 5 iterations and close to no improvement in the function value, *0.529046*.

**Bind tolerance (TolBind)**
Bind tolerance determines if the search direction must include the constraint boundaries. If the distance from the current point is less than TolBind the constraint becomes active. Mathwork recommends setting TolBind equal to or larger than the maximum tolerance of 'TolMesh', 'TolX' and 'TolFun'.[9] Thus the value was varied between $10^{-6}$ to $10^{-1}$. No improvement of either the function value or execution time was found.

Several additional parameters were tested but the global minimum was never found. All in all, the systematic approach yielded the following best feasible objective function values:

- Obj 71: best value: *0.5290* exitflag: 1
- Obj 63: best value: *0.6789* exitflag: -2

These are about 57 % and 12 % higher, respectively, than the global reference solutions.

[9] `http://www.mathworks.se/products/global-optimization/examples.html?file=/products/demos/shipping/globaloptim/psoptionsdemo.html`

### 4.3.2 Genetic Algorithm

The genetic algorithm was tested with patternsearch on both objective functions 63 and 71. It was implemented with 8 generations of 30 individuals each. The mean objective function value for each generation is shown in figures 8 and 9. Unfortunately, the mex-files sometime aborted the solver because of a domain error in the ADM. These individuals were excluded from the results. The best objective function values found were *0.6364* and *0.4867*, respectively. This should be compared to the global minima of *0.6087* and *0.3779*. Of the solvers that did not abort because of the mex error, the exit flags were around 70% 0's (meaning the maximum number of function evaluation or time limit had been reached) and 30% 1's (meaning the mesh and constraints tolerances had been fulfilled).

The parameters that had generated the lowest objective function values were then tested with patternsearch for execution time and speedup. The means of five runs were analyzed. The execution times for one worker were around 1580 seconds for 63, and 610 seconds for 71. The maximum standard deviation of execution times were around 2 %. However, execution time was one of the active stopping criteria. The speedup is shown in figure 10; a maximum speedup of 3 was achieved for 9 workers.

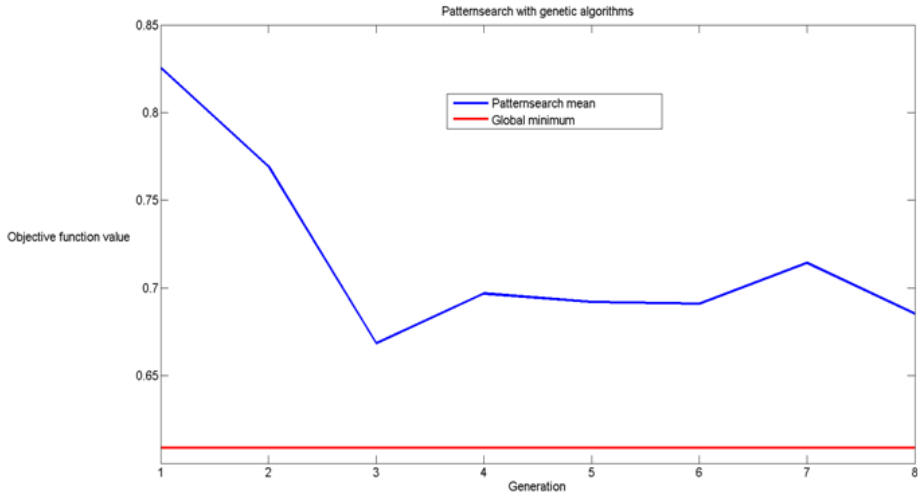


Figure 8: The performance of the genetic algorithm for objective function 63. Each generation consisted of 30 individuals; shown are the mean final objective function value for each generation. The best individual found had an objective function value of 0.6430.

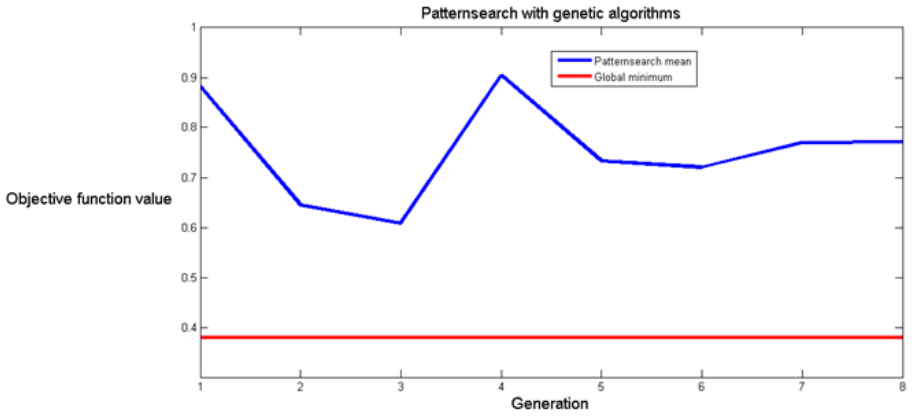


Figure 9: The performance of the genetic algorithm for objective function 71. Each generation consisted of 30 individuals; shown are the mean final objective function value for each generation The best individual found had an objective function value of 0.4867

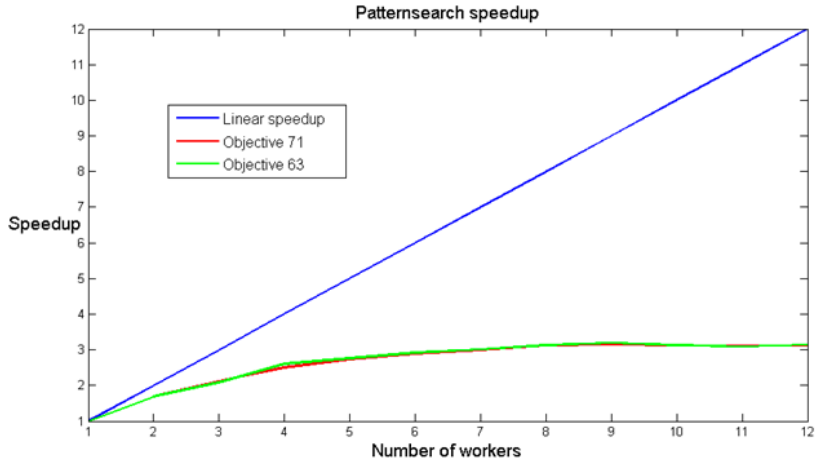


Figure 10: Speedup for both objective functions. The values shown are mean values of five tests. Both peaked at around 3 speedup for 9 workers.

### 4.3.3 Supplementary tests

To find out why patternsearch failed to find the global optimum, some supplementary tests were carried out.

When Patternsearch narrows in on a certain value it tends to get stuck, unable to improve further. We tested the hypothesis that if the initial guess is close to the global minimum, it should be easier for the algorithm to find it. The result shows that the solver does not progress to the global optimum of 0.3779; see

figure 11.

| Iter | f-count | f(x) | constraint | MeshSize | Method |
|---|---|---|---|---|---|
| 0 | 1 | 0.378762 | 2.086e-12 | 1 | |
| 1 | 14 | 0.378762 | 2.086e-12 | 0.01 | Increase penalty |
| 2 | 344 | 0.378762 | 2.086e-12 | 0.001 | Increase penalty |
| 3 | 784 | 0.378762 | 2.086e-12 | 0.0001 | Increase penalty |
| 4 | 1279 | 0.378762 | 2.086e-12 | 1e-05 | Increase penalty |
| 5 | 2049 | 0.378762 | 9.714e-07 | 1e-06 | Increase penalty |
| Optimization terminated: mesh size less than options.TolMesh and constraint violation is less than options.TolCon. | | | | | |

Figure 11: Patternsearch with an initial guess close to the global optimum for objective function 71.

There is a relaxation parameter that determines how strict the equality constraint are in the ADM; these constraints represent mass flow continuity in the ADM. By increasing this relaxation parameter (default value 0%), the equality constraints are transformed into inequality constraints, meaning an enlarged feasible region, which should make it easier for patternsearch to find the global optimum. Multistart was also tested to determine a correct new global minimum, since increasing the feasible region will most likely result in a new global minimum. The results are shown in table 7.

Table 7: Results from changing relaxation, objective function 71

| **Relaxation** | **Patternsearch** | **Multistart** | Difference (relative) |
|---|---|---|---|
| 5% | 0.9379 | 0.3137 | 0.6242 (2.99) |
| 25% | 0.3823 | 0.2014 | 0.1809 (1.90) |
| 50% | 0.2642 | 0.1321 | 0.1321 (2.00) |

# 5 Discussion

## 5.1 MultiStart

MultiStart performs well in the tests. The solver generates points within bounds and uses them as starting guess regardless of if they are feasible or infeasible, finding the global optimum in more than 80 % of the cases; the exception is Active-set for objective function 63; however, the stopping criteria could probably be adjusted. This would bring up the percentage that finds the global optimum within the 4th decimal to 80 % as well. An 80 % success rate with 10000 initial points indicates a very accurate method.

Execution time for 120 starting points was around 2100-2400 seconds, depending on the algorithm and objective function 71. This yields an average execution time per point of around 20 seconds.

The speedup is almost linear, reaching a maximum of 10 for 12 workers. However, for a lower amount of points, the speedup is reduced; for 24 starting points, the maximum speedup found was around 5. This happens because the amount of work per worker is not sufficient, causing the overhead due to the workers to be larger than the gained speedup. This illustrates the importance of assigning enough work when running in parallel.

There was a case when the subsolver fmincon found different optimum depending on what algorithm that was used. Appendix A contains figure 13 that illustrates an offset of 0.02 in the y-axis between the two algorithms Active-set and SQP. This is likely due to the feasible initial point $x_0$, which converges slighlty different for the two algorithms. If Mulitstart for 10 starting points is run there is no difference in the global optimum for the two algorithms.

## 5.2 GlobalSearch

Globalsearch appears to be very accurate. The tests always found the global optimum. Furthermore, since it relies on fmincon just like Multistart, it should have similar convergence properties, meaning that it probably would suffice to test a very small number of points, as discussed under the Multistart subheading. The execution times are unfortunately very large. However, since mex-files had to be turned off, these results should be treated with care when compared with the other methods, that naturally are much faster.

The speedup was found to be a maximum of 5x for 12 workers; this is higher than predicted. Considering that it lacks native parallelization support, the speedup of Globalsearch should not be better than that of fmincon. Running fmincon for up to 12 workers resulted in a speedup of 2x. The explanation has to do with the mex-files. GlobalSearch was configured to run without mex-files and hence was more computationally heavy compared to fmincon with mex-files. The speedup measure favours non-optimized code which means that globalsearch will have a larger parallel section, hence more work/worker than fmincon for the same problem.

GlobalSearch did not work with mex-files. When GlobalSearch fails it always executes fmincon once for the initial guess, evaluates a few objective functions, and then stops with a message saying that the execution was successful and that all trial points had been analyzed. This is strange since GlobalSearch will always run fmincon at least twice, once for the initial $x_0$ and one more for the stage 1 points. It is possible that the mex-files somehow interfere with the generation of trial points(scattering), preventing any trial points from being created.

### 5.2.1 Mex files

Mex-files should be used since they reduce the execution time significantly, but they also exhibit some difference compared to the original matlab code. For instance when running fmincon with and without mex-files the found solution is the same but the number of iterations sometimes differ. There is also a tendency of errors occurring, concerning domain errors and complex values, with mex-files. A recommendation is to perform a thorough debugging of the mex-files, while running Globalsearch, to help clarify the odd behaviour.

## 5.3 Patternsearch

### 5.3.1 Convergence problem

For both the systematic and stochastic approach no parameter values were found that made patternsearch converge to the global minimum. The results where feasible points with objective function values of around 4 % for obj. 71 and 28 % for obj. 63 when compared to the global reference solution.

Patternsearch did manage to find better results with the genetic algorithm. The solution for both obj 63 and obj 71 were found by a combination of poll method and search method. In both cases one of the methods were of type MADpositiveBasis which gives a certain randomness to the generated pattern. The random search direction could, in some cases, manage to find a small feasible 'path' towards a better solution.

The equality constraints $G_i$ of the optimization problem are clearly the reason to why patternsearch struggles to find the global optimimum. From Figure 7 one can see that the penalty is increased for every subproblem. This is due to candidate points being close to one of the many constraints. Repeatedly increasing the penalty will eventually confine the movement of patternsearch to the point when only small steps are possible, hence the minor improvements for iteration 3-4 in 7. Systematic tests were done, varying the initial penalty and penalty factor parameters, which affected the execution times, but no improvement of the global solution could be seen. Fine tuning these parameters could however lead to patternsearch finding the global optimum.

A test where the relaxation parameters were increased was done in order to enlarge the feasible region and make it easier for Patternsearch to find the global minimum. The results in Table 7 hints about this being the case. The difference between the true global solution found by MulitStart and the solution found by Patternsearch decreases when the equality constraints are relaxed. This could indicate that Patternsearch can find a path to the global minimum easier.

Previous results [1], based on a similar model, done outside the frame of this project, indicate that Patternsearch performs well when other global solvers struggle. An important difference between the two models are that ADM has equality constraints and the other model hasn't.

Patternsearch is still interesting to use with the ADM. The solver could be combined with e.g. Multistart creating a hybrid solver that might result in better performance (high accuracy and acceptable speedup). More specifically Patternsearch could be used to supply Multistart with a better initial point by searching for an optimum with the equality constraints turned off; this should be no problem for patternsearch. Then the obtained optimizer $x^*$(feasible w.r.t. the inequality constraints) is supplied as an intial point to Multistart with the equality constraints turned on. This could make the subsolver fmincon converge to a minimum faster than with the regular approach, because the initial point might be close to the optimum.

To summarize, Patternsearch never converges to the global minimum, meaning that accuracy can not be measured for the solver. The convergence problem and slow execution time is most likely caused by the strict equality constraints. This makes it difficult for the solver, without the aid of gradients, to find its way to the global optimum. In order to improve the solver, the genetic algorithm can be further refined in e.g. how inheritance works by finding parameters that converge, or that a careful analysis of the penalty parameters could give better convergence. Lastly, a hybrid solver would be interesting to implement and analyze.

#### 5.3.2 Speedup

Patternsearch exhibited execution times in the same order of magnitude as the other solvers, but these results are essentially meaningless since execution time was used as an active stopping criteria. The speedup for Patternsearch is more interesting; a maximum of around 3 was achieved for 9 workers. The reason for this poor speedup is the large serial part of Patternsearch; only the computations of the objective functions of the candidate points can be parallelized.

# 6 Conclusions

## 6.1 Current state

Our main findings are presented in Table 8. Multistart and Globalsearch are both very accurate, but only Multistart has reasonable execution time. It is important to note that execution time is the least interesting result, since Globalsearch did not work with mex-files and patternsearch did not converge. They cannot be compared in any meaningful way, but it has been shown that Multistart is reasonably accurate and fast when running 10 starting points. In terms of speedup Multistart is by far the best, this is visualized in Figure 12.

Based on these results Multistart is the best solver in terms of performance and speedup when used with the ADM. It is a good candidate for implementing with the ADM on a cloud architecture, running global optimization efficiently and accurately on the internet.

Table 8: A summary of the performance of each solver.

| Solver | Best value found | Accuracy | Max Speedup |
|---|---|---|---|
| Multistart | Global optimum | 80 % | 10x |
| Globalsearch | Global optimum | 100 % | 5x |
| Patternsearch | 4 % (obj. 63) and 28 % (obj.71) higher than global optimum | 0 % | 3x |

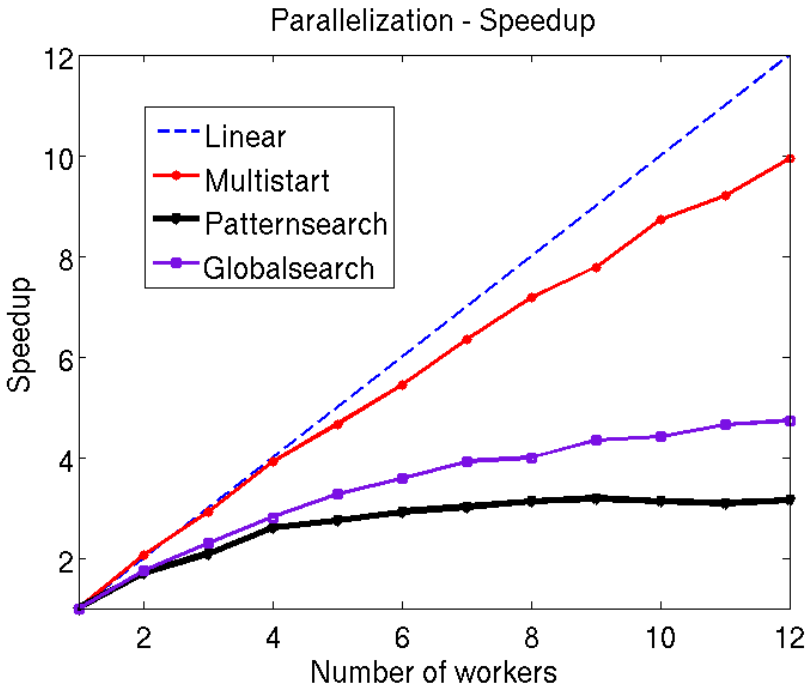


Figure 12: Speedup curves of the three solvers for problem 71

## 6.2 Directions for future work

It is recommended to further investigate Patternsearch in order to improve its performance, especially investigating the penalty parameters. It is also suggested that a proper optimization of the matlab code is performed. It is possible to reduce the computational complexity of the code and thereby remove the need of mex-files, solving the problems regarding Globalsearch. Finally, analysing the hybrid solver could give a new approach to generating better starting points for e.g. Multistart.

# 7 Acknowledgement

We would like to thank our supervisors Kateryna Mishchenko and Anders Daneryd for introducing us to the project and for their support. We also like to thank Maya Neytcheva for administrating the project course that this project was a part of.

# A Appendix

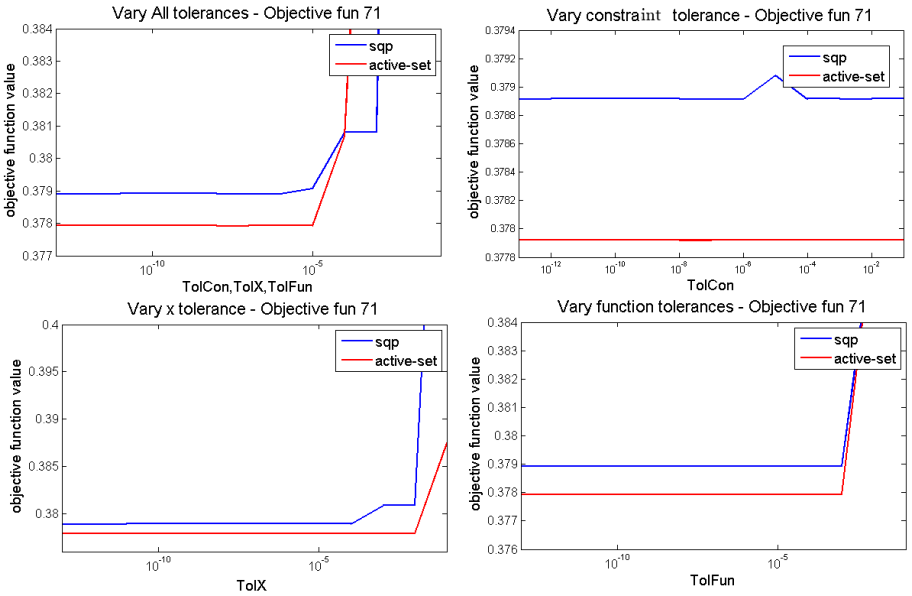


Figure 13: Tolerance test to see how constraint, objective and x tolerance affect the solution

# B Appendix

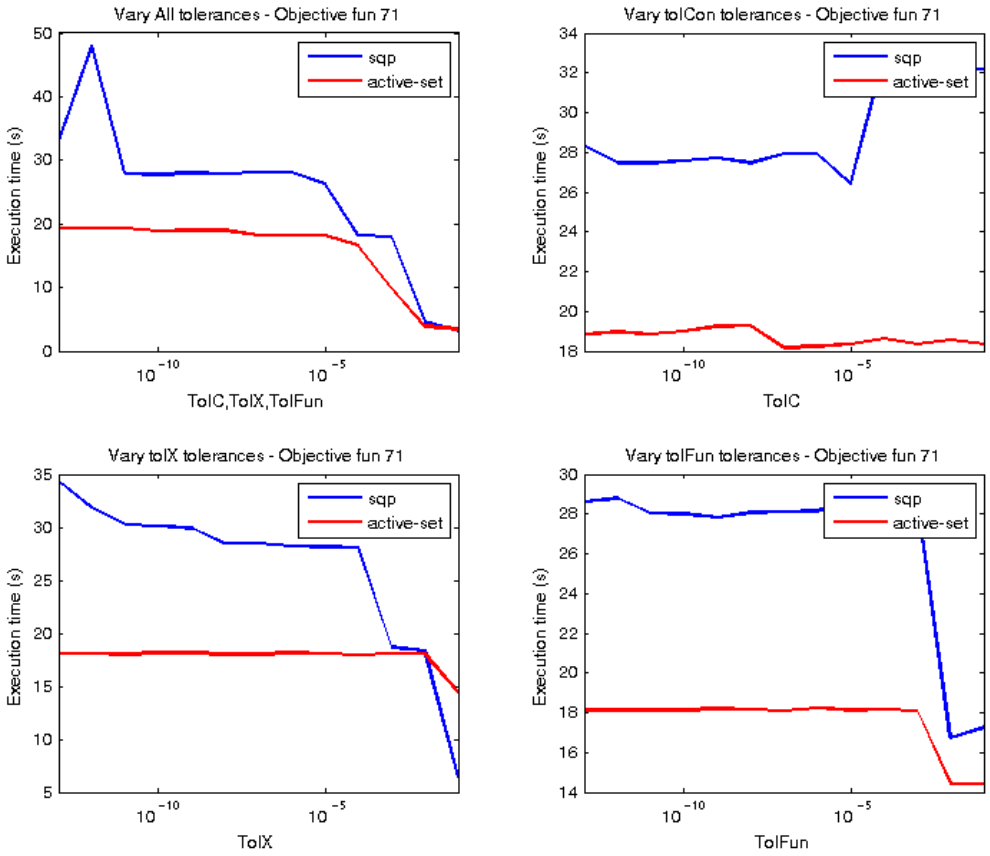


Figure 14: How execution time depends on choice of tolerance

# References

[1] Joakim Agnarsson, Mikael Sunde, Inna Ermilova, *Parallel Optimization In Matlab; 2012;* . Department of information Technology Uppsala university