Automated dig-limit optimization through Simulated Annealing

Thijs Hanemaaijer

Automated dig-limit optimization through Simulated Annealing

by

Thijs Hanemaaijer

in fulfillment of the requirements for the degree of

Master of Science
in
Applied Earth Sciences

at the Delft University of Technology,
to be defended publicly on the 17th of December 2018 at 16:00

Thesis committee:

Dr. M.W.N. Buxton (Chair)	TU Delft Resource Engineering
Prof. Dr. M.A. Hicks	TU Delft Geo-Engineering
Dr. H. Hajibeygi	TU Delft Petroleum Engineering
Dr. M. Soleymani Shishvan (Supervisor)	TU Delft Resource Engineering
J.R. van Duijvenbode, MSc.	TU Delft Resource Engineering

Department of Geoscience & Engineering
Delft University of Technology

Abstract

As ore-bodies become more complex and difficult to extract, mining operations need to increase their efficiency and performance. Dig-limit design is part of the short-term mine planning in open pit mines. Dig-limits are the boundaries between material destinations in a mining bench. Common practice in the mining industry is that these are created manually by a mining engineer, this results in subjective and sub-optimal dig-limits.
In this thesis an automated optimization program was written to optimize these dig-limit designs. The optimization of dig-limits is a combinatorial optimization problem which has proven to be NP-hard, it cannot be solved exact in a reasonable time-frame. Therefore the meta-heuristic method of simulated annealing has been used for the optimization program. A meta-heuristic optimization method is a problem independent method that uses a smart searching algorithm, such that not the whole solution space needs to be investigated. This cannot guarantee an absolute optimal solution but will produce a near optimal solution within reasonable computing time.
Simulated annealing uses an initial solution, from where it makes a change into a neighboring solution. The new solution is then accepted or rejected based on the difference in the objective value. If the objective value increases, the solution is always accepted, if it decreases, the solution may still be accepted with a certain probability. This probability depends on the magnitude of the difference in objective value, and on the temperature. The temperature is a control parameter for the acceptance of worse solutions, and is regulated by the cooling schedule. Initially the temperature will be very high, such that most worse solutions are accepted and the algorithm makes a broad search of the solution space. During the course of the algorithm the temperature is lowered by the cooling schedule and less worse solutions are accepted. This will make the algorithm converge into the found optimum. The constraints in the simulated annealing program were enforced by applying a penalty to the objective value for constraint violations.
Different varieties for the simulated annealing program have been tested to investigate their applicability for dig-limit optimization and their performance. These options are written as interchangeable modules which are all compatible to each other and can easily be implemented in the program. Different methods were investigated for the initial solution, the constraint penalty, the cooling schedule, the perturbation mechanism and the stop criteria of the algorithm. The best performing combination of modules was a random initial solution, with a quadratic cooling schedule, a random perturbation which stops after a certain number of consecutive rejected perturbations.
To reduce or prevent frequent switching between destinations, a specific module for the penalty function was created. This module could successfully and in a controlled manner discourage the switching between destinations, while making the final solution comply to the spatial mining constraints. This will make the algorithm more applicable to mining operations where dilution is a problem. Another successful option was a pre-optimized initial solution in combination with a greedy algorithm, which does not accept any worse solutions. This created high quality dig-limit designs in a considerable smaller amount of computation time.
The resulting automated dig-limit optimization program was successful in creating near-optimal dig-limit designs. The program is flexible and can be adjusted for multiple material destinations, multiple ore-types and different shapes and sizes for the spatial mining constraints. The objective function can be adjusted for different optimization goals. This makes the program potentially applicable for different types of open-pit mining operations.

Acknowledgements

First of all I would like to thank my initial supervisor Tom Wambeke, who introduced the topic of dig-limit optimization to me. He laid down the framework for the program that was written and gave me essential guidance to get familiar with the programming language python. I would also like to thank Jeroen van Duivenbode, for his assistance with the programming and for his constructive feedback throughout my thesis.

Furthermore I would like to thank Masoud Soleymani Shishvan, for taking over supervising me in the final stages of my thesis and giving guidance and feedback on the writing of this thesis. I would like to thank Mike Buxton, Micheal Hicks and Hadi Hajibeygi, for being part of my thesis committee and reviewing my work.

Thijs Hanemaaijer

Delft, 3 December 2018

Contents

Nomenclature

Abbreviations

MMU	Minimal mining unit
SA	Simulated annealing
SMU	Selective mining unit

Subscripts

0	Initial
k	Current
m	Mining
n	Final
p	Processing

Parameters

α	Cooling factor	$[-]$
χ	Acceptance ratio	$[-]$
ΔE	Change in objective value between solutions	$[-]$
c	Cost	$[\$/t]$
g	Mineral grade	$[g/t]$
n	Total amount of cycles	$[-]$
p	Mineral price	$[\$/g]$
r	Recovery factor	$[-]$
T	Temperature, control parameter	$[-]$
z_c	Cut-off grade	$[g/t]$

List of Figures

List of Tables

1 Introduction

The generation of dig-limits is a key aspect of short-term mine scheduling. Dig-limits are the boundaries between material destination zones, they define what material is processed, and what material is send to the waste dump. The optimization of these boundaries can significantly affect the economic value of a mining operation.
To explain the purpose of this thesis, in this chapter mine planning and the role of dig-limits within this process are introduced. The problems that are encountered with the current conventional manner of the creation of dig-limits will be discussed, as well as the advantages of automated optimization methods. The objectives and the scope of this thesis will be outlined at the end of the chapter.

1.1 Mine planning

Mine planning is an important process in mining engineering. It starts with a geological block model containing the estimated grades and ore types of the ore material. Upon the information in this model it is decided whether or not a block should be mined, when it should be mined, and if it is mined, to which destination it should be sent. This problem has many different solutions and it is common practise to try to maximize the net present value (NPV) of the mine while dealing with constraints like production, blending criteria and environmental regulations (Dagdelen, 2001). The NPV is the sum of the present value of all future cash-flows, thus where revenue created in the future is calculated for its worth today (Hustrulid et al., 2013).

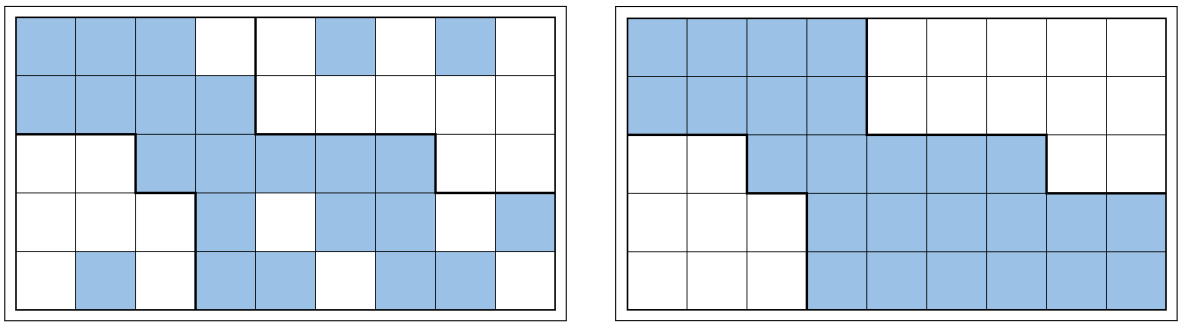
Mine planning is commonly divided into three levels: the long-term planning, the medium-term planning and the short-term planning. Long-term planning is the initial strategic planning, it is based on the estimation or simulation of block models from often sparse drill-hole data. The blocks in these models have a large block size and are called planning blocks. The main goal of long-term planning is to generate a mine plan to maximize the net present value, taking into account the macro economic sensitivity of the project as well as long term market forecasts(Ruiseco et al., 2016).
Medium-term planning connects the mining operations to the long-term mine plan. It tries to maximize the compliance to the long-term plan and uses more detailed forecasts and previous knowledge of the mining operation. For example it manages the planning for drilling and blasting, the hauling and the equipment maintenance (Sari and Kumral, 2018).
The short-term planning focuses on the daily mining activities, implementing the medium-term plan and handling daily problems and difficulties encountered in the mining operation. Also the required production rates, quality, equipment use, throughput of the processing plant and recovery are managed by short term planning (Sari and Kumral, 2018).
The short-term mine planning uses blast-hole data to estimate the ore grades and lithology inside a mining bench. These blast-hole data are much denser than the drill-hole data. This enables for a detailed detection of grade variations on a smaller scale (Ruiseco et al., 2016).

The data from the blast-holes are assigned to volumes called Selective Mining Units (SMU's) which are defined as the smallest block on which ore or waste selection is made (Sinclair and Blackwell, 2002). There are different interpretations as to what volumes these should be, which are mostly based on the blast-hole spacing and the bench height. Sometimes also the equipment selectivity is taken into account in the SMU definition (Sinclair and Blackwell, 2002). For this thesis a SMU is defined as the spacing between the blast-holes and the height of the mining bench, such that every blast-hole is the center of a single SMU. Dig-limits are part of the short-term planning, using these SMU's to create day-to-day operational mine plans.

1.2 Dig-limits

After the drilling of blast-holes, more detailed grade distribution information is available about the mining bench. These blast-hole data enable more precise scheduling of mining sequences and ore-waste classification. However, in reality an ore-waste classification on SMU-scale is not practicable. Firstly the mining equipment is very large to mine single SMU's and send them to different destinations (Sari et al., 2017). Secondly, if it was possible to mine individual SMU's there would be a significant dilution and ore-loss due to frequent ore-waste boundaries, and thirdly, this frequent switching between ore and waste mining will have a great impact on the production rate (Ruiseco et al., 2016).

For this reason, dig-limits, also referred to as dig-lines are created. Dig-limits group SMU's together complying to the spatial constraints of the mining equipment, and assigning them to a certain destination. The dimensions of this minimum group of SMU's will from now on be called the minimal mining unit (MMU). Dig-limits are the boundaries separating the material in the mining bench that is send to different destinations, for example to the processing plant, the leach heap or the waste dump. This grouping of SMU's and sending them to a single destination will automatically result in the misplacement of certain SMU's. Some high-grade SMU's initially classified as ore will be sent to the waste dump, which is called (ore)loss, and the other way around, some low-grade SMU's classified as waste will be sent to the processing plant, which is called dilution. Figure 1.1 illustrates a grid that is divided on SMU basis, and the same grid that is divided by dig-limits, with a clustering size of 2x2 SMU's.



(a) SMU selection (b) Dig-limit

Figure 1.1: Illustration of dig-limit clustering

The blue blocks are assigned as ore SMU's, and the white blocks the waste SMU's. On the left the blocks are assigned to their individual optimal destination, and on the right they are grouped together. As can be seen, some waste blocks are turned into ore, and the other way around.

1.3 Purpose

1.3.1 Problem statement

Deposits that are currently being mined are decreasing in grade and are becoming more complex, often different ore-types and multiple metals are extracted. This results in more complex processing and multiple possible destinations for the mined material therefore the short-term scheduling in these mining operations is also becoming more difficult. Multiple metal mining is also becoming more important in the mining industry as a way of risk mitigation and increasing the value of an operation (Ruiseco and Kumral, 2017).

Common practice in the mining industry is that the dig-limits are drawn manually by a pit-geologist. This is a labour intensive task which results in subjective and therefore often sub-optimal dig-limits (Ruiseco et al., 2016). The resulting dig-limits can approach the optimal when a geologist is very experienced, but still there is effort and time loss due to the daily need to create dig-limits. Automated optimization of dig-limit designs can significantly increase the efficiency and revenue of a mining operation. This is especially the case for high value metal operations, or operations where ore loss and dilution are a problem (Sari and Kumral, 2018).

The design of dig-limits is a combinatorial optimization problem, there are many variables influencing the quality of a solution. These kinds of problems require a lot of computing time to be solved by an exact method. To make these problems more manageable, heuristic search methods are used. These methods do not always result in the ultimate optimal solution but try to approach the optimal solution within a reasonable time-frame (Rutenbar, 1989).

1.3.2 Research objectives

The main objective of this thesis is to develop an automated dig-limit optimization method based on simulated annealing.

Sub-objectives:

1. The program must be able to handle complex deposits with multiple ore types, destinations and selectivity sizes.
2. A mining direction preference will be included in the program.
3. The dig-limits can be used to decrease frequent switching between ore and waste mining.
4. The sensitivity of the program and the results to different parameters will be investigated.

1.3.3 Scope

This thesis focuses on the development of a dig-limit optimization program, applicable to open-pit mining operations. The program is based on simulated annealing and multiple variable plugins are created for the optimization program and are tested and compared on their performance. The data that will be used as input for the tests are multiple subsets from the artificial Walker Lake data-set, containing x- and y-coordinates and grades. The sensitivity of the program to different parameters and different data-sets are investigated.

Out of scope:

- The effect of movement from blasting is not incorporated in this research, this will not have influence on the program and the comparative results. For this research it is assumed that the input grade control model is corrected for blasting.
- A real-life case is not investigated in this thesis, as it is not available, and will not influence the performance of the algorithm. The Walkerlake data-set is deemed realistic enough to prove the performance of the program.

1.3.4 Thesis organization

In chapter 2 the literature review is presented. The subject of optimization is shortly introduced, then the method of simulated annealing is discussed in detail, starting with the global working of the method, and after that every part of a simulated annealing algorithm is elaborated on. At the end of chapter 2 the literature on dig-limit optimization will be discussed, as well as the originality and the approach of this research.
Chapter 3 will explain how the theory of simulated annealing was applied to the dig-limit optimization problem. All the different options for the algorithm that are created and compared will be introduced, and the working of the program will be explained. The cases that are used to investigate the performance and applicability of the program are introduced at the end of the chapter.
Chapter 4 will show and discuss all the results of the tests and comparisons between the different algorithm options on their effect on the algorithm, and the possible consequences for the application in mining operations.
In chapter 5 the results are concluded upon and recommendations for further study are suggested.
The python code of the modules that are created in this thesis can be found in the appendix A.

2 Background

2.1 Optimization

The main goal of this thesis is to optimize the dig-limit designs of open-pit mining benches. This is a combinatorial optimization problem, meaning that there are a finite set of objects that need to be combined satisfying given conditions (Sierksma and Ghosh, 2009). The combined group of objects satisfying these conditions is called a (feasible) solution. From all different possible solutions, called the solution space, it is tried to find the optimal one. To define which of these solutions is optimal, a quantification of the quality of these solutions is required. This is assigned by the objective function, the function which is minimized or maximized. There are two approaches applicable to solve these combinatorial optimization problems.

The first option is using a standard optimization algorithm, this will find a guaranteed optimal solution to the problem. An example of such an algorithm is an Exhaustive-search or Brute-force search algorithm, which systematically checks all possible solutions for the optimal one. This will always result in an exact optimal solution, but may take an (unrealistically) long computation time (Van Laarhoven and Aarts, 1987). In many real-life cases and in dig-limit design as well, the optimization problem is NP-complete (Sari and Kumral, 2018). NP-complete means that the known techniques to acquire an exact optimal solution need an exponentially increasing number of steps when the problem become larger. These problems will quickly become too large and will take an impossible computational effort to be solved exact, and often brute force methods will not suffice to solve them (Rutenbar, 1989).

The second option is using a heuristic, or approximation algorithm, which gives a near-optimal solution in a reasonable time-frame, but which can never be guaranteed to be optimal (Laarhoven van and Aarts, 1987). The definition of an heuristic is: criteria, methods or principles for deciding upon a course of action that is probably the best or most effective way to achieve a certain goal (Pearl, 1984). In optimization this is used to reduce the computational time and prevent the requirement to search the whole solution space. An easy implementation of an heuristic method is the descent algorithm, a local search algorithm which wants to find a local minimum. It starts with an initial solution which may or may not be random, and searches the neighbourhood accepting only solutions with a lower objective value. When no lower solution can be found the algorithm is terminated and the local minimum is found (Eglese, 1990).

These types of optimization algorithms are also called 'greedy' and have the issue that it is never certain whether the found minimum is a local minimum or the global minimum. This is illustrated in figure 2.1. The ball represents the current solution, with a descent algorithm only downhill movements are allowed. This means that when it reaches the first minimum, it can not get out and the search is terminated, without finding the global optimum. An option to use these types of algorithms to find the global optimum is running it many times from different random initial solutions, and remembering the best solution. This way the whole solution space can be sampled. But still, when the problems

are getting larger and more complicated, they may require an exorbitant number of starting points to properly search the whole solution space. This will consume a lot of computing time and still can not guarantee an optimal solution (Rutenbar, 1989).
This problem is overcome in simulated annealing by allowing uphill moves with a certain probability. While simulated annealing still cannot guarantee an optimal solution, it handles the local minimum problem. This optimization method will be explained in depth in the next section 2.2.

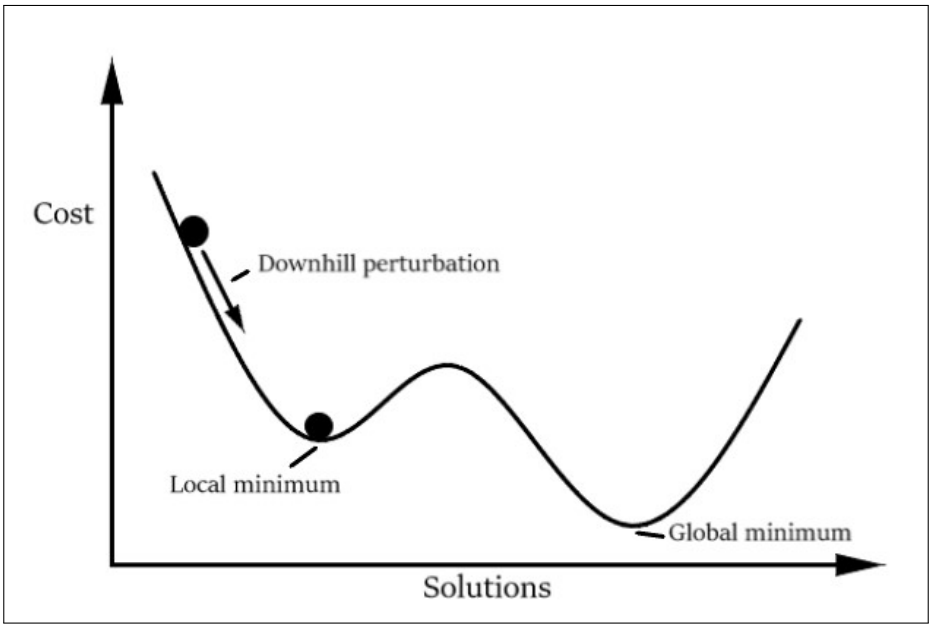


Figure 2.1: Local and global minima. Adapted from:(Rutenbar, 1989)

2.2 Simulated Annealing

Simulated annealing is a meta-heuristic computational method that is often used for the optimization of combinatorial problems. The method of simulated annealing was first investigated by Kirkpatrick et al. (1983) and Černỳ (1985) as a method to find the global minimum of a cost function that might have multiple local minima.
A key feature of the algorithm is that it allows for uphill movements, where it might accept a worse solution in order to search a larger solution space and to prevent convergence into local minima or maxima. The method is based on the annealing process of metals (Rutenbar, 1989).

2.2.1 Annealing of solids

Simulated annealing is based on the analogy between combinatorial optimization problems and the physical process of the annealing of solids. Annealing is the process of recrystallizing a solid into a low(er) energy state. This is accomplished by heating the solid to a maximum temperature which is just above the melting point. At this temperature all the molecules of the solid will go into the liquid phase and are able to move freely and rearrange themselves. When step-wise cooled down following a cooling schedule, and given enough time at certain temperatures, the molecules will rearrange themselves in a way to minimize the Gibbs free energy (Van Laarhoven and Aarts, 1987). When cooled too rapidly the solid will not be able to reach the ground-state, which is the minimum energy configuration of the solid, but will get stuck into a local optimal structure with defects. The different states of the solid in annealing can be compared to different feasible solutions of the combinatorial optimization problem in simulated annealing, and the energy of these states corresponds to the function that needs to be optimized (Eglese, 1990). The rapid

cooling and getting stuck in a locally optimal structure can be compared to the getting stuck in a local minimum in simulated annealing when the temperature is lowered too quickly. In the analogy the descent algorithm mentioned earlier can be compared to directly cooling the solid to its freezing point (Szu and Hartley, 1987)(Aarts et al., 2005).

2.2.2 Simulated annealing theory

In simulated annealing, the process of annealing is projected on combinatorial optimization problems. The goal is to find a solution (configuration of particles) that minimizes the cost (energy). To simulate how the system in annealing reaches thermodynamic equilibrium at each temperature stage in the cooling schedule, Metropolis et al. (1953) created the Metropolis algorithm, which is displayed in figure 2.3 (Youssef et al., 2001).
The algorithm makes a random perturbation, simulating a particle which rearranges itself, and compares it to the prior configuration. The change in energy (cost) ΔE between the two configurations (solutions) is then calculated. If $\Delta E < 0$, the new configuration is always accepted. If $\Delta E > 0$ and thus the new configuration has a higher energy state, the movement might still be accepted if the temperature is high enough and the ΔE is not too big. In physical annealing, these perturbations to a higher energy state also happen with a higher probability at high temperatures. To simulate this, the Metropolis algorithm uses a Boltzmann distribution, which is shown below in equation 2.1 and figure 2.2 (Ledesma et al., 2008)(Glover and Greenberg, 1989).
Thus the probability of accepting an uphill solution increases with a smaller ΔE or a higher temperature. This is implemented into the algorithm by generating a uniform random number between 0 and 1, if $P(accept)$ is higher than the random number, the new solution is accepted (Rutenbar, 1989).

$$P(accept) = e^{-\Delta E/T} \tag{2.1}$$

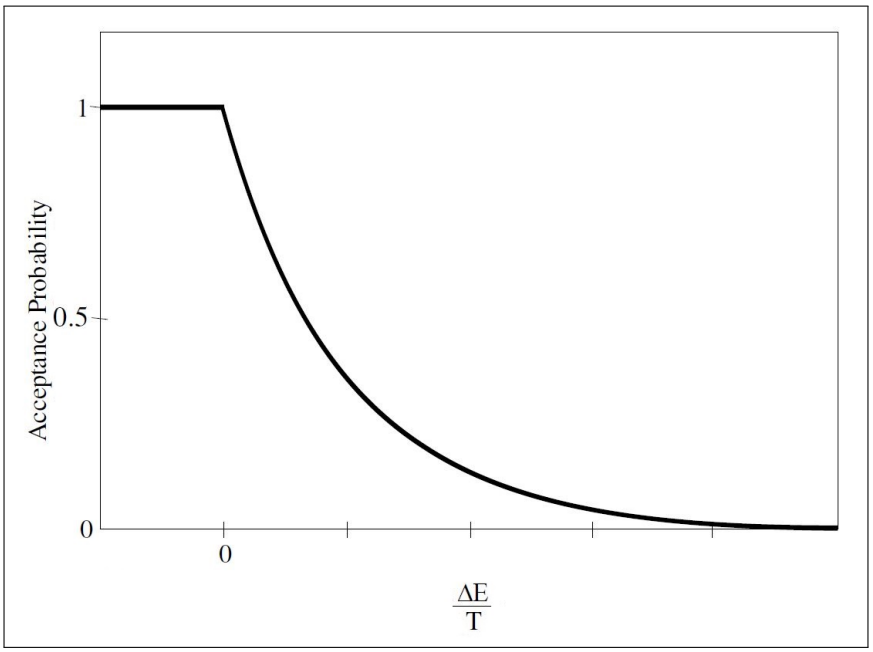


Figure 2.2: Boltzmann distribution (Ledesma et al., 2008).

This algorithm is ran at systematically lowered temperatures, simulating the material reaching equilibrium at each temperature. In real annealing this temperature is a physical factor, in simulated annealing

N = Number of moves to attempt at temperature T
T = Current temperature

for n = 1 **to** N:

NewS = New solution after perturbation
CurS = Current solution before perturbation
ΔE = Energy change between NewS and CurS
if $\Delta E < 0$:
CurS = NewS
else:
if : RANDOM(0,1) $< e^{-\Delta E/T}$
CurS = NewS
else:
CurS = CurS

Figure 2.3: Metropolis algorithm. Adapted from (Youssef et al., 2001).

it is a control parameter to influence the acceptance of uphill moves. By adjusting the 'cooling schedule' the acceptance of uphill moves in the course of the algorithm is regulated (Eglese, 1990). The cooling schedule will be further elaborated upon in section 2.2.2.4.

A simulated annealing algorithm is built up from different components, which can be divided in generic and problem specific components. The generic components need to be implemented in any application of simulated annealing, they are the initial temperature, the cooling schedule, the number of trials per temperature cycle, and the stopping criteria. The problem specific components, which need to be adapted for each specific application of simulated annealing, are the solution space, the initial solution, the perturbation mechanism and the constraints (Johnson et al., 1989).
The algorithm starts with an initial solution from where the algorithm will search for improvement. Then there are constraints to which the solution should comply. Based on the constraints and the solution, an objective value is assigned by the objective function, this is the function that needs to be optimized. Upon the outcome of the objective function the solution is accepted or rejected by the metropolis algorithm, based on the magnitude of the change of the objective value and the current temperature. A cooling schedule is used to control the temperature, and thus the acceptance probability of the algorithm. When a solution is accepted or rejected, it is again changed by a perturbation into a new, neighbouring solution, and the whole cycle from constraint to perturbation is repeated until the stop criteria are reached (Youssef et al., 2001)(Aarts et al., 2005). The process of simulated annealing is demonstrated in pseudo code in figure 2.4 and each component of simulated annealing will be discussed in detail in the next sections.

2.2.2.1 Initial solution

The simulated annealing algorithm requires an initial solution as input. This initial solution can either be a random feasible solution or can be pre-optimized, this depends on the type of problem that needs to be solved (Eglese, 1990). In the case of dig-limit optimization this might be a solution made by hand or by a simple algorithm. A random solution might need more time to converge to the optimal solution, but will probably result in a good search of the solution space. Whereas a pre-optimized solution will probably converge faster, but might nudge the algorithm into a local minimum. If the temperature is kept high in the start of the algorithm the pre-optimized initial solution might be a waste of computing time as the initial search will accept a lot of uphill solutions and 'create' a random solution before converging. Thus

n = Current perturbation
N = Number of perturbations at temperature T
m = Current cycle
M = Total number of cycles
T = Current temperature
S_0 = Initial solution
T_0 = Initial temperature
α = Cooling factor < 0

Start Algorithm

$m = 0$
$T = T_0$
$CurS = S_0$

while $m \leq M$:
$n = 0$

while $n \leq N$:

$NewS$ = New solution after perturbation
$ObjValue$ = Cost difference between NewS and CurS
if $ObjValue < 0$:
$CurS = NewS$
else:
if : $RANDOM(0,1) < e^{-ObjValue/T}$
$CurS = NewS$
else:
$CurS = CurS$
$n = n + 1$
$T = T * \alpha$
$m = m + 1$

Figure 2.4: Simulated Annealing. Adapted from (Youssef et al., 2001).

with a pre-optimized initial solution the initial temperature should be reduced (Eglese, 1990).

2.2.2.2 Constraints

If the problem that needs to be optimized is constrained, there are two ways of implementing this into the algorithm. A hard-constraint can be implemented, this is a constraint that can not be violated. This is done by constraining the solution space, such that the algorithm will only search for feasible solutions. However, depending on the optimization problem this is often difficult to implement (Eglese, 1990). Hard constraints might also impede the searching capability of the algorithm as is explained and illustrated in section 3.2.

The other option is a soft-constraint, this constraint can be violated but against a cost, a penalty is applied for constraint violations, allowing the algorithm to search for unfeasible solutions. This will give an easier perturbation mechanism and create a smoother solution space with less deep minima (Eglese, 1990). The magnitude of the penalty is a trade-off between making sure the constraints are complied to

in the end of the algorithm and the capability of the algorithm to search the solution space. When the penalty of a violation is very high the penalty will almost act like a hard-constraint, almost no solutions with penalty violations will be accepted. This will limit the search of the solution space as a temporary violation might eventually lead to an improved final solution.

The penalty can also be used to fulfill a goal rather than an absolute restriction, for instance to discourage certain behaviour of the solution. In this case a lower penalty can be applied for these violations. This can also be combined with a 'hard' penalty constraint (Jeffcoat and Bulfin, 1993).

2.2.2.3 Objective function

The objective function is the measure of quality of the solutions found by the simulated annealing algorithm. It is the driving factor of the optimization, every decision in the algorithm is made by the max- or minimization of the objective value. This is different for each application of simulated annealing and the objective function for dig-limit optimization will therefore be discussed in section 2.3 Dig-limit quality.

2.2.2.4 Cooling schedule

The cooling schedule regulates the temperature of the algorithm. This is a control parameter which influences the probability of acceptance of 'uphill', moves to worse solutions. From equation 2.1, the Boltzmann distribution, it can be derived that with a decreasing temperature, the share of the perturbations that is rejected will increase. This will cause the algorithm to make less far fetched moves and to potentially converge into a local minimum (Kalivas, 1992).

This makes the cooling schedule one of the most important components of a simulated annealing algorithm, as it will have a great influence on the performance of the algorithm (Bertsimas et al., 1993). This is shown by Van Laarhoven and Aarts (1987) who compared three different cooling schedules and found that the quality of the resulting solutions can deviate as much as 10%. Because of its importance a lot of research has been conducted on different types of cooling schedules.

A cooling schedule consists of an initial temperature, a decrement-function to lower the temperature, and the number of iterations at each temperature. Collins et al. (1988) classified a number of different schedules, which different authors have used. Some of these schedules will be discussed in this section.

Kirkpatrick et al. (1983) proposed a simple form of cooling schedule, which starts by an empirical approach to determine an appropriate initial temperature. The initial temperature T_0 should be high enough that virtually any uphill perturbation is accepted, thus $e^{-\Delta E/T_0} \simeq 1$. To determine the initial temperature they choose a large value for T_0 and perform a number of perturbations. The acceptance ratio χ, which is defined as the number of accepted perturbations divided by the total number of perturbations, is calculated. If χ is smaller than a certain value χ_0 the initial temperature is doubled, this is repeated until $\chi > \chi_0$. They suggest an acceptance ratio of $\chi_0 = 0.8$. If $\chi \gg \chi_0$ too many uphill perturbations will be accepted and the algorithm will make a random walk in the solution space without converging. On the other hand if $\chi < \chi_0$ too many uphill perturbations will be rejected and the algorithm will quickly converge into a local minimum (Kalivas, 1992).

The initial temperature that is required depends on the value of the solutions and the penalty that is applied to the solutions. These factors influence the difference in objective value after perturbations, which following the Boltzmann equation 2.1 influences the probability to accept a worse solution. Therefore, for each optimization case an appropriate initial temperature should be determined.

After the initial temperature is determined, the cooling schedule must be chosen. There are two main types of cooling schedules, multiplicative and additive. Multiplicative cooling schedules use a factor that reduces the current temperature. An often used, basic example of such a schedule is shown in equation 2.2, where k is the current cycle, T_k is the current temperature, and α is a factor typically in the range

$\alpha \in [0.5, 0.99]$ (Kalivas, 1992)(Van Laarhoven and Aarts, 1987).

$$T_{k+1} = \alpha * T_k, \qquad k = 0, 1, 2, ..., \qquad \alpha < 1 \tag{2.2}$$

Additive cooling schedules use the total number of cycles and a fixed final temperature. A basic example of such a schedule was proposed by Golden and Skiscim (1986), they take 0 as final temperature and divide the interval from T_0 to 0 into a fixed number of cycles K. This schedule has a constant decrease in temperature instead of a constant factor of decrease and is called a linear cooling schedule, it is shown in equation 2.3. Where T_k is the current temperature, k is the current cycle, K is the total number of cycles, and T_0 is the initial temperature.

$$T_k = \frac{K - k}{K} * T_0, \qquad k = 1, ..., K \tag{2.3}$$

In this thesis the second type of cooling schedules was used, because they are easier to manipulate and control.
To determine the number of iterations $N(k)$ that is required at each temperature T_k there are different techniques found in literature. Kirkpatrick et al. (1983) sets $N(k)$ to a sufficient number of perturbations that needs to be accepted, although when the temperature approaches zero, almost no perturbations will be accepted, and $N(k) \to \infty$. For this reason an upper bound on $N(k)$ is required. Another simple and often used option is choosing $N(k)$ as a fixed number based upon the size of the problem (Van Laarhoven and Aarts, 1987).

2.2.2.5 Perturbation

The perturbation mechanism, also called the neighbourhood structure is also an important factor for the success and the efficiency of the algorithm. The neighbourhood search is a very problem specific part of simulated annealing. It is highly dependent on how the solution space looks like and what is defined as a neighbouring solution (Jeffcoat and Bulfin, 1993).

For dig-limit optimization the neighbourhood is defined by changing the destination of an m x n frame. This way the new solution will be very similar to the old solution, but still changed and possibly improved. Other options that are investigated are changing multiple frames per perturbation or given a preference to certain frames to be changed. This will be further explained and discussed in chapter 3.

2.2.2.6 Stop criterion

The stop-criteria determine when the algorithm will terminate. The main idea is that the algorithm is terminated when the improvement of the solution is no longer expected to increase significantly when the algorithm continues.

The easiest way to implement a stop criterion is by letting it run a certain number of cycles. The downside of this method is that it might result in a premature termination where the algorithm hasn't found the global optimum yet, or in a waste of computing time where the algorithm has found the optimal solution and is stuck for a long time. Another method is to terminate the algorithm when there is a certain number of consecutive perturbations that are rejected. This way the algorithm wont unnecessarily continue without improvement (Van Laarhoven and Aarts, 1987).

Huang (1986) suggests another stop criterion. At the end of each temperature cycle, after $N(t)$ trials, the difference between the maximum and minimum cost value of all accepted perturbations is compared to the maximal difference in cost after any of the perturbations. If these are the same, the perturbations in the cycle are all of similar cost and the temperature is set to zero to induce a descent algorithm into the local minimum.

2.2.3 Quality of the simulated annealing algorithm

Approximation algorithms, like the simulated annealing algorithm are generally tested on their performance by the quality of the final solution, thus the difference between the cost of the final solution and the cost of the global optimal solution, and the computation time required by the algorithm to reach this final solution (Van Laarhoven and Aarts, 1987).
In this thesis the performance of the algorithm will be tested empirically, by running many different configurations of the algorithm and comparing the results on their quality and computation time. The exact global optimal solution of the cases is not known and thus cannot be used to asses the algorithm quality.

2.3 Dig-limit quality

To optimize a dig-limit design, first it needs to be defined how the quality of a dig-limit should be assessed. This defined quality needs to be quantified and translated into the objective function of the algorithm. There are multiple methods found in the literature to quantify the quality of dig-limits.
A common way to assign a value to a mining block is by using the marginal cut-off grade. This is the mineral grade at which the revenue of the ore is equal to the processing cost, see equation 2.4 (Verly, 2005).

$$p * r * z_c - c_m - c_p = -c_m \longleftrightarrow z_c = \frac{c_p}{p * r} \tag{2.4}$$

Where p is the mineral price $[\$/g]$, r is the recovery factor, z_c is the cut-off grade $[g/t]$, c_m is the mining cost$[\$/t]$, and c_p is the processing cost $[\$/t]$. Any additional cost can easily be incorporated in the equation by increasing the marginal cut-off grade.
When applying this equation to the blocks, and only the destinations ore and waste are used, there are four possible scenarios. A block of grade z can be correctly assigned as ore, $z > z_c$. a block can be falsely assigned as ore, $z \leq z_c$. A block can be correctly assigned as waste, $z \leq z_c$. Or a block can be falsely assigned as waste, $z > z_c$.
The undiscounted value of blocks can be defined in two manners, the potential value and the recovered value. The recovered value is the value of the block as it is destined. For blocks send to the processing plant this is $(p*r*z_c-c_m-c_p)$ and for blocks send to the waste dump this is $(-c_m)$. The potential value of a block is the maximum value that the block could be worth given it is send to the correct destination. This means the potential and recovered values are equal for correctly classified blocks. (Verly, 2005)

A minimum loss method was proposed by Isaaks (1990) and Srivastava (1987). They assign a destination to minimize the expected loss. The loss is defined by the potential value minus the recovered value.
The misclassification of SMU's can occur in two ways, either a waste block is classified as ore and is send to the processing plant, or an ore block can be classified as waste and send to the waste dump. The second kind of misclassification generally results in a greater loss of profit because in this case none of the valuable material in the block will be recovered and all the potential profit will be lost, while the mining costs are still made. In the other case when processing a waste block, money is lost due to processing cost, but still some valuable material may be recovered (Isaaks et al., 2014).
Isaaks et al. (2014) quantifies the dig-line quality by assigning loss functions to estimation errors where the loss due to underestimation of blocks is greater than the loss due to overestimation. The loss of a group of SMU's with different ore types is calculated by Isaaks et al. (2014) calculating the potential revenue of each SMU with the equations 2.5, 2.6 and 2.7, where P is the net revenue, \$ is the metal price, Z is the block grade, Ri the recovery factor of ore i, B_i is the break-even cost of ore i, Zc_i is the cut-off grade and n the number of ore types.

$$P_i = \$ * Z * R_i - B_i \quad i = 1, ..., n \tag{2.5}$$

Where:

$$B_1 = \$ * Zc_1 * R_1 \tag{2.6}$$

$$B_{i+1} = \$ * (Zc_{i+1} * R_{i+1} - Zc_{i+1} * R_i) + B_i \quad i = 1, ..., n \tag{2.7}$$

The recovery factor R_i is specific for each ore type. The loss functions of a cluster of SMU's are then determined for each SMU's possible destinations. The optimal destination is the one with the minimal loss. This method is applicable to multiple ore-types and destinations.(Isaaks et al., 2014)

2.4 Dig-limit optimization

The prior research that has been conducted into the optimization of ore-waste classification and dig-limit designs will be discussed in this section, as well as the originality of this research and the research approach.

2.4.1 Prior research

One of the first ideas of dig-limit optimization was proposed by Allard et al. (1994), they pointed out that especially when the cut-off grade is high relative to the mean grade, with a heterogeneous grade distribution, there are high grade blocks which will be separated from mine-able clusters of blocks and must be mined together with surrounding waste-grade blocks. This results in loss and dilution, thus the value of a block does not only depend on its own grade, but on the grade of the surrounding blocks as well. They concluded that a connectivity index which indicates the number of clusters and how they are connected, should be used in combination with the classical grade-tonnage curve to make a proper mining reserve estimate.
Richmond and Beasley (2004) used a greedy iterative construction heuristic with a stochastic demolition and reconstruction strategy for ore selection problems. They repeatedly discard the destination of a number of random blocks and recalculate the best destination for these blocks, accepting the new solution if the objective value has improved. During the algorithm the number of random blocks that is changed is decreased, until a stopping criterion is reached. Their method outperformed a constructive-simulated annealing hybrid, but does not incorporate equipment size constraints.
Wilde and Deutsch (2015) proposed a new method called Feasibility Grade Control. Their method assigns each SMU to a mining unit of 2, 3 or 4 SMU's, which are send to the same destination. The algorithm randomly changes the unit of a block by switching it with a neighbouring mining unit. If the profit of the solution increases, the change is accepted. The process is iterated until all blocks have been changed a given number of times. This is a greedy algorithm, and can therefore get stuck in local optima. The method also requires an initial hand-drawn dig-limit solution.
Tabesh and Askari-Nasab (2013) used agglomerative hierarchical clustering, which starts at each block being an individual cluster, and merging the most similar blocks together until all blocks are clustered or a stopping criterion is met. They created a similarity index between blocks by dividing the similarity measures by the dissimilarity measures, which are normalized to make the different measures comparable. The measures used are grade, shape, destination and rock-type. Their algorithm can not control the mining constraints and small clusters or sharp corners need to be removed after the algorithm is completed.
Norrena and Deutsch (2000) were the first to use simulated annealing to optimize dig-limit designs. They used an objective function to optimize the profit, and penalized small angles in the dig-limits, which are not 'digable' due to equipment size. The 'digability' is defined as a measure of difficulty by which a dig-limit can be extracted. The magnitude of the small angle penalty depends on the size of the mining equipment that is used. Their perturbation mechanism moves a single vertex of the dig-limit polygon to a new point within a certain range, accepting the mining of partial blocks. Their dig-limits compared well to hand-drawn dig-limit designs, however their method can not handle multiple ore and waste zones. This is because it cannot handle a waste dig-limit polygon inside of an ore dig-limit polygon, and it

cannot handle the merging or splitting of the dig-limit polygons. This means that the initial amount of polygons would need to be defined beforehand and cannot change during the optimization.
Neufeld et al. (2003) created a semi-automatic program for dig-limit optimization, based on simulated annealing. They maximize the profit of the dig-limit solution. The program can handle multiple ore types and destinations, but requires a manually input initial dig-limit solution. Their method can handle only a single dig-limit at a time, when multiple dig-limits are required the program must be run multiple times, each time requiring an initial solution, and the profit of every dig-limit polygon is summed.
Isaaks et al. (2014) used simulated annealing, in which they implemented a minimal mining width, grouping a row of SMU's together to comply to the mining equipment constraints. They optimized the dig-limit design for minimum loss, rather than maximum profit. They noted that the loss resulting from the misclassification of SMU's is greater for assigning ore grade blocks to the waste dump, then for assigning waste grade blocks to the processing plant. When the ore grade block is send to the waste dump, the whole value of the block is lost, as when a waste grade block is send to the processing plant, still some valuable material is recovered. Therefore they assigned asymmetric loss-functions to all possible destinations of each SMU in the cluster, the destination with the smallest loss is assigned as the optimal destination of the cluster. The method is applicable on multiple ore-types and destinations.
Sari and Kumral (2018) used mixed linear integer programming to do solve the dig-limit optimization problem. They maximize the revenue of a bench while complying to a square equipment constraint of n x n SMU's. Their method moves a n x n frame over each SMU and makes sure it belongs to a frame where all SMU's are send to the same destination. Their method does result in the certain optimal solution, but it takes a lot of computing time, therefore it probably wont be practical for real mining operations. It can be very useful as a benchmark for (meta-)heuristic methods such as simulated annealing, which cannot guarantee an optimal result.
Ruiseco et al. (2016) used genetic algorithms to optimize the dig-limit design. Their method uses a large 'population' of solutions which are 'bred', combined to generate new solutions. All solutions are ranked by their 'fitness' which is the objective value of the solution. High ranked solutions have better chances to multiply than lower ranked solutions. The mining equipment constraint is implemented by subtracting a penalty from the objective function for constraint violations, similar to simulated annealing. They took the profit of the solution minus the penalty as the objective value which is to be optimized. Their perturbation mechanism changes a single line of n x 1 or 1 x n SMU's and changes its destination. Their constraint only looks in the perpendicular axes of the block that is investigated, and not a square frame, this will cause minor mining constraint violations especially at oblique borders. Their method was compared to hand-drawn dig-limit designs and performed well.
Ruiseco and Kumral (2017) researched the impact of equipment sizing on the dig-limit optimization problem. As the equipment size increases, the mining costs decrease, but also the selectivity decreases. They used dig-limit optimization to find an optimal equipment size for a single mining bench. They show that the relationship between equipment size and profitability of the dig-limit design is non-linear and shows severe breaking points as the selectivity gets lower.
Dagasan et al. (2018) used pilot point optimization in combination with simulated annealing to optimize ore-waste boundaries in laterite-type bauxite deposits. Their method minimizes the ore loss and dilution by optimizing the boundary topography between the ore and waste layers. This problem shows similarity to did-limit optimization but the main difference is that it optimizes the location of a single horizontal boundary, thus their method only works for stratified deposits. They assumed a constant grade for all ore material in their research.

2.4.2 Originality of this thesis

There have been many attempts at optimizing the dig-limit problem, however there has not yet been a perfect solution. Often the existing methods are not able to create mine-able dig-limits, are not computationally feasible for real life application in the mining industry, or do not optimize the profit. The most promising results are from Isaaks et al. (2014), who also used simulated annealing to minimize

the ore loss. However, their method and results are not described as it has a commercial purpose. Ruiseco et al. (2016) successfully applied genetic algorithms to the dig-limit problem, their method was successful compared against manually drawn dig-limits, but still there are some constraint issues.
In this thesis simulated annealing is used to optimize the dig-limit problem for profit, and different methodologies are compared and described in detail. A meta-heuristic method is used in this thesis, since solving the problem exact has proven to be NP-Hard, especially for large and complex deposits (Sari and Kumral, 2018). The choice of simulated annealing was based on the fact that it has a single current solution, whereas genetic algorithms have a large population of current solutions, for all of which the objective value needs to be evaluated. This makes simulated annealing easier to implement and probably less computationally intensive than genetic algorithms. Simulated annealing has been used for the optimization of dig-limits but has never been properly described and discussed.
The resulting algorithm has successfully handled the dig-limit optimization problem while fully complying to the spatial mining constraints. It manages this in a very reasonable computation time which can make it applicable for use in the mining industry. Furthermore a method has been created to discourage destination boundaries and frequent switching between destinations.

2.4.3 Thesis approach

The main objective of this thesis is to create an automated dig-limit optimization program. The programming language that was used is Python 3.6. To investigate which implementations of the simulated annealing algorithm work best on the dig-limit optimization problem, different implementations of parts of the simulated annealing algorithm were created and tested. The program consists of a framework in which modules of these different implementations can be combined. The program and the modules that were created are discussed in chapter 3.
To asses the performance and the quality of the algorithm, the different modules were tested against a base-case. The resulting dig-limit design, final objective value and the objective value, penalty and real-value of the solution in the course of the algorithm were investigated. The results are discussed in chapter 4.

3 Methodology

Multiple variations in parameters and modules have been tested on their performance in the simulated annealing algorithm. This was done in comparison to a base-case which will be described at the end of this chapter. All module-strategies of the algorithm that were investigated are stated in this chapter as well as the mining and algorithm parameters that were used. The results of all the tests that were performed are reported in chapter 4. It is important to note that in this chapter when an SMU is called ore, it means its destination is the processing plant, not necessarily that the grade of the SMU lies above the cut-off grade. And the other way around for SMU's called waste. The automated dig-limit optimization program was written in python 3.6, the program and how it works is explained in this chapter. The python code for the different modules is shown in appendix A.

The algorithm consists of different parts that will be discussed in this chapter. The algorithm starts with an initial solution, then the constraint penalty that is applied in the objective function is calculated. The objective function calculates the objective value from the profit of the solution minus the penalty. Then the algorithm either accepts or rejects the solution. The temperature is lowered by the cooling schedule and the solution is perturbed into a neighboring solution. The new solution is then again penalized and the cycle continues until the stopping criteria are met.

3.1 Initial solution

Two types of initial solutions were compared in this thesis. In both methods the grid, defining a mining bench, is divided in clusters of blocks with a dimension of the input minimal mining unit (MMU), which is defined as the m x n spatial mining constraint. The blocks inside these clusters are assigned to a single destination. The division of the grid in clusters is required to ensure that the initial solution has a constraint-penalty of zero. This prevents the requirement to perform an initial, computing-time intensive penalty calculation of the whole grid.

The first strategy is that each cluster is send to a single random destination, this will have a low computing time and will probably give a good search of the whole solution space. The second strategy is to calculate and assign the optimal destination for the cluster i.e. the destination resulting in the maximum value of all the blocks in the cluster combined. This way a quick feasible pre-optimized dig-limit design is generated from which the algorithm can take off. This dig-limit is not optimal, but will have a significantly higher objective value than a random initial solution. This may give a faster convergence, but also might nudge the algorithm towards a local optimum by discouraging a broad search of the solution space. If the second option is used the initial temperature should be lowered, otherwise the initial search will accept too much worse solutions and the solution will be randomized, wasting the advantage in computing time.

3.2 Constraints

As explained in section 2.2.2.2, there are two manners to enforce constraints in a simulated annealing algorithm. A hard constraint, or a soft constraint in the form of a penalty can be used. In the case of

dig-limit optimization, the first option is difficult to implement. The solution space should be limited, which would mean that the perturbations need to comply to the constraints, this will make it much more complex to program, or it will waste a lot of time on rejected perturbations that don't comply to the constraints. It will also make it harder to deviate from the current solution and explore the whole solution space. This is illustrated in figure 3.1 with a 2x2 MMU constraint.

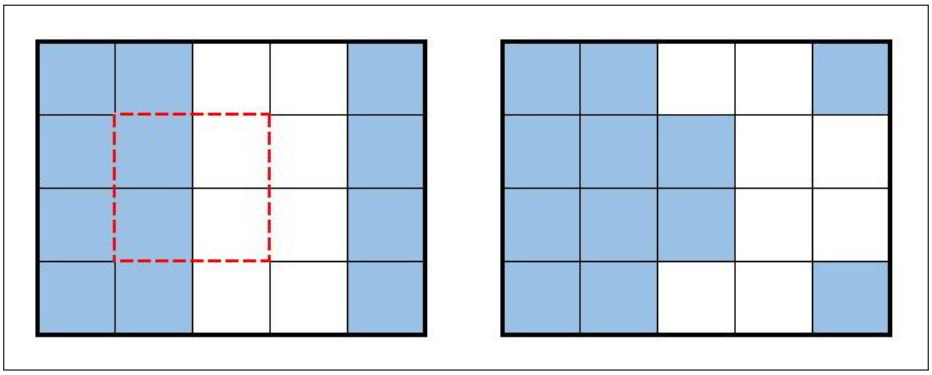


Figure 3.1: Constraining the solution space, on the left the current solution and on the right the optimal solution.

Imagine the grid extends outside of these blocks. With a hard-constraint the perturbation of changing the blocks in the red square on the left grid to blue would not be allowed because it would leave a single row of two white blocks, which would violate the 2x2 MMU constraint. To obtain the hypothetical optimal solution in the right figure, first another perturbation is required which will make it much more unlikely to occur. The right corner blocks in the right figure are disregarded in this example and are thought of as part of clusters outside the figure. The algorithm has been tested with a simple form of hard constraint, where only solutions that comply to the constraints are accepted. This resulted as expected in a long run time with a low final objective value, meaning that a hard constraint limits the searching capability of the simulated annealing algorithm.

For these reasons it is chosen to apply a penalty to enforce the constraints of the algorithm. This penalty can also be used to discourage 'unwanted' behaviour of the algorithm, for instance it is not desired to switch frequently between ore and waste mining, as this will slow down the production and will increase dilution. By applying a low penalty to switching, this behaviour is still allowed but will only occur if the value of switching is higher than the penalty that it would cause.
The penalties assigned by the algorithm are calculated via 'misfit' values. A misfit value is assigned to all possible frames in the grid. A frame is any possible group of blocks with the dimensions of the MMU. The misfit value is defined as the number of blocks in any possible frame, that is not assigned to the same destination as the largest group of blocks in the frame. The misfit value is independent from the destination of the misfits, it only indicates how many blocks in the frame are not send to the largest destination group. This is illustrated in figure 3.2 with a frame size (MMU) of 3x3 SMU's.
The frames are identified by their bottom left block, to which the misfit value is assigned. In this case the highlighted frame has five blue blocks, and four white blocks. Thus the largest destination group is blue, and a misfit value of four is assigned to the frame. A list of the misfits of all possible frames in the grid is created at the start of the algorithm, and is updated after each perturbation.
This list can be used in two ways, the first is for the discouraging of switching between destinations. This is simply done by summing the misfit values of all frames in the grid. Boundaries between destinations will result in frames with misfit values which are then penalized with a certain factor to control the grade of discouragement. This penalty factor will generally be low because it is not a hard mining constraint but rather unfavourable behaviour.
The second way is to check for each block if they lie inside a frame with a zero-misfit value. If this is

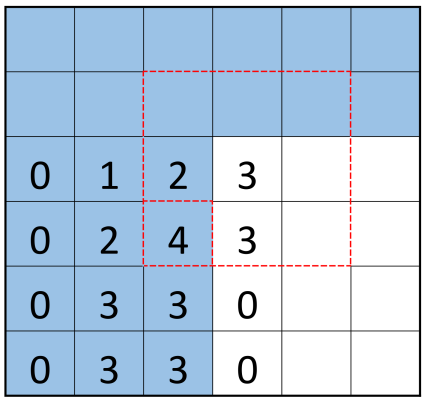


Figure 3.2: Misfits of a frame.

true it means that the mining constraint is not violated. If the block does not lie in a zero-misfit frame the lowest misfit value of all the frames the block lies in, is assigned as penalty to the block. Again this penalty is multiplied by a penalty factor, which in this case should be chosen high as this penalty should prevent that the constraint is violated in the final stages of the algorithm.

Three different approaches to the final penalty assignment were tested and compared in this thesis. The first was using only the sum of all misfit values, this is expected to highly discourage ore-waste switching, but will also discourage equipment constraint violations. In the second approach only the minimal-misfit values of the blocks were taken into account. This is expected to result in a feasible dig-limit design, but ore-waste switching will not be taken into account by this approach. The third approach was a combination of both, with a low factor to discourage the switching between ore and waste and a high factor to prevent the violation of the mining constraints.

The magnitude of the penalty that is required to ensure no violations occur depends on several factors. The main requirement of the penalty is that it influences the objective value in such an extent, that a violation will always cost more than it might increase the objective value. This logically depends on the value that is assigned to a solution. In the case of dig-limit optimization this will be the mining parameters and the grade values in the grid. The penalty that is required will need to be adjusted in each individual case. In this thesis it was investigated how a good penalty can be derived depending on the mining parameters and the grade values of the grid.
To determine an appropriate penalty factor for the mining constraints that were used in this thesis, the algorithm was run for a number of cycles and the compliance to the constraints was assessed. During these runs the temperature of the algorithm was initially kept high for a short time to allow the algorithm to violate the constraints, and then quickly lowered to zero, to force convergence early without searching for the most optimal solution. The algorithm was run for a certain amount of cycles at temperature zero to converge into the local minimum. Then the compliance to the constraints was quantified by dividing the total final penalty by the penalty factor that was used. To find a correlation between an appropriate penalty factor and the grade values of the grid (mining bench) different penalty factors for three cases were investigated. The results are shown in section 4.1

3.3 Objective value

The objective value in the program is calculated by the value of the solution, minus the penalty assigned to the solution. In this thesis the value of the solution was defined by the profit. This was done for simplicity and mining companies might create their own criteria for the value of a solution, according to their knowledge of the operation or requirements for the processing plant.

The profit of a solution was calculated for each SMU using equation 3.1. Where p is the mineral price $[\$/g]$, $r(g)_i$ is the recovery factor for blocks send to destination i, a recovery curve which depends on the grade can also be used, but in this thesis a constant factor was used for simplicity, g is the mineral grade $[g/t]$ of the SMU, mc is the mining cost $[\$/t]$, and pc_i is the processing cost $[\$/t]$ for destination i.

$$Blockvalue(x, y) = (p * r(g)_i * g) - (mc + pc_i) \tag{3.1}$$

The objective value of the entire grid of mxn blocks was calculated by equation 3.2.

$$Objectivevalue = \sum(Blockvalue(x, y)) - penalty \quad x = 1, ..., m, \quad y = 1, ..., n \tag{3.2}$$

3.4 Cooling schedule

As stated in section 2.2 the cooling schedule is an important part of the algorithm and will have a considerable influence on the convergence of the algorithm. For this reason the effect of different cooling schedules is investigated extensively. A cooling schedule consists of an initial temperature and a cooling schedule which controls the rate of cooling in the course of the algorithm.

For the initial temperature it is important that it is high enough to allow uphill moves at the start of the algorithm, in order to ensure a good search of the whole solution space. However when the initial temperature is too high it can waste processing time (Youssef et al., 2001). The initial temperature used in this thesis was decided upon empirically using the method described by Kirkpatrick et al. (1983), which is explained in section 2.2.2.4.

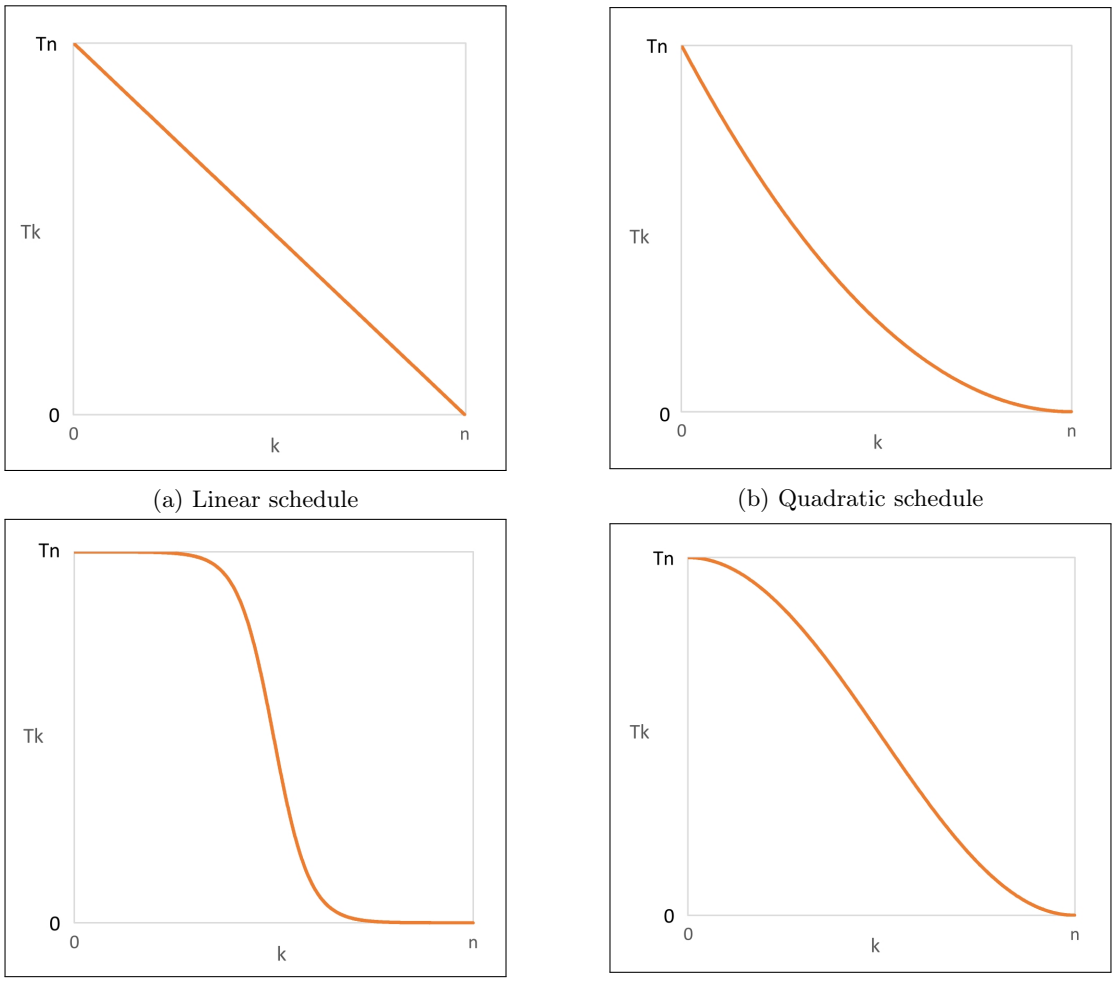
The algorithm was run for 2000 trials at a constant initial temperature, the acceptance ratio χ was calculated by dividing the number of accepted perturbations by the total number of perturbations. The desired acceptance ratio for an initial temperature was chosen to be $\chi_0 \approx 0.8$, such that at the start of the algorithm around 80% of all perturbations are accepted. If $\chi < \chi_0$ after 2000 trials, the initial temperature was doubled and the algorithm was ran again.

The cooling schedule itself will also have a large influence on the convergence of the algorithm. There are different schedule strategies that are each different in their initial search of the solution space and their rate of convergence. If the schedule remains at a high temperature for a longer time, the search of the solution space will be better, but the convergence will be slower, which might waste computation time. If the temperature is lowered too fast or too early, there will be a bigger risk that the algorithm converges into a local minimum.

The schedules that were tested are; linear cooling, quadratic cooling, exponential cooling, and trigonometric cooling (Luke, 2007). These schedules are displayed in figure 3.3 and their formulas in table 3.1, where T_k is the current temperature, T_0 is the initial temperature, T_n is the final temperature, k is the cycle number and, n is the total number of cycles in the temperature schedule.
At the end of the algorithm, when the number of cycles of the temperature schedule are completed and the temperature has become zero, the algorithm will act as a greedy algorithm. The temperature is kept at zero for a certain amount of extra cycles or amount of consecutive rejected cycles to give a deeper and more certain descent into the local optimum.

3.5 Perturbations

The perturbation mechanism controls the way the current solution looks for a neighbouring solution. This can have a big influence on the convergence speed. There were three different strategies investigated



(a) Linear schedule

(b) Quadratic schedule

(c) Exponential schedule

(d) Trigonometric schedule

Figure 3.3: Cooling schedules (Luke, 2007)

in this thesis. The first will take any random frame in the grid and change the destination of all blocks in the frame to a random destination.
The second strategy chooses with a certain probability between choosing a random frame in the grid, or changing a boundary frame, and changes the destination to random. A boundary frame is a frame that lies on a destination boundary. This might increase the convergence rate of the algorithm. This strategy uses the misfit list to identify all frames at a destination boundary and chooses a random frame from them. This method might make the algorithm converge earlier and will decrease the 'useless' perturbation of frames that are already send to their optimal destination.
The third strategy will take any random frame in the grid and change the destination of all blocks in the frame to the optimal destination of the combined blocks. This was expected to give a fast convergence, but it might encounter problems with removing penalty violations, as for each possible frame only one possible destination can be assigned.

3.6 Stop

Two strategies were used in the stop criteria of the algorithm, either the algorithm was simply halted after a certain number of cycles have been completed, or after a certain number of consecutive new

Table 3.1: Cooling schedules (Luke, 2007)

Schedule type	Formula
Linear	$T_k = T_n + (T_0 - T_n)\left(\frac{n-k}{n}\right)$
Quadratic	$T_k = T_n + (T_0 - T_n)\left(\frac{n-k}{n}\right)^2$
Exponential	$T_k = T_n + (T_0 - T_n)\left(\frac{1}{1+e^{\frac{2\ln(T_0-T_n)}{n}(k-\frac{1}{2}n)}}\right)$
Trigonometric	$T_k = T_n + \frac{1}{2}(T_0 - T_n)\left(1+\cos\frac{k\pi}{n}\right)$

solutions have been rejected by the metropolis algorithm. Different numbers of cycles have been tested and compared on their improvement of the solution versus the added time. For the second option also different numbers of rejected cycles have been tried to find a good computing time versus improvement ratio.

3.7 Parameters

The parameters that are used in the simulated annealing program can be divided into two types: the algorithm parameters and the mining parameters.

The algorithm parameters influence the performance of the program, these are the initial temperature, number of cycles, number of trials per cycle and, penalty factors. These parameters were tested extensively in this thesis to research their effect.

The mining parameters are used to calculate the value of a SMU. These parameters are the mining cost, processing cost, recovery, and mineral price, or any other parameter influencing the objective value. These parameters will not directly influence the performance of the program, but are taken into account to make the program applicable for real-life dig-limit-optimization scenarios.

The mining parameters were kept constant most runs, this was done to make comparisons between the algorithms easier. The mining parameters will primarily affect the objective value, and the required penalty factor. To investigate the influence of the cut-off grade, different processing costs were compared.

For simplicity, the mining cost of waste and ore mining is kept the same in this thesis, and each SMU is assumed to be one tonne in weight. The program is flexible about the number of material destinations and recovery factors, but in this thesis only two destinations are used, the processing plant and waste dump. The recovery factor and the processing cost of material that is send to the waste dump were set to zero. The mining parameters that were used are shown in table 3.2.

Table 3.2: Mining parameters

Parameter	Value	Unit
Mineral price	50	[$/g]
Recovery factor waste	0	[-]
Recovery factor ore	0.8	[-]
Mining cost waste	1000	[$/t]
Mining cost ore	1000	[$/t]
Processing cost waste	0	[$/t]
Processing cost ore	1000	[$/t]

3.8 Dig-limit optimization program

This section will discuss the dig-limit optimization program that was written for this thesis. The program consists of a framework in which modules of all different strategies that were investigated can be implemented. The different modules are all compatible to each other, so different combinations of strategies could be tested. A diagram of how the program works is shown in figure 3.5.

The input of the program should be delivered in GSLIB format. This is a commonly used format for geostatistical software used to store grid information.

The following convention is used as input for the program:

- The first line will contain the number of blocks in the x, y, and z direction in the grid.
- The second line is a numerical value **n** specifying the number of numerical variables in the data file.
- The following **n** lines are character identification labels describing each variable.
- The following lines from **n + 3** until the end of the file contain the data points. They will have **n** numerical values per line, separated by commas[,] and with points [.] as decimal-separators.

An example of a GSLIB format input file with a grid size of 60 x 30 x 1, an X and Y coordinate and two grade values is displayed in figure 3.4.

```
60 30 1
4
X
Y
grades-U
grades-V
1,1,4.125,179.11
2,1,4.119,190.67
3,1,4.169,190.42
4,1,8.174,241.37
5,1,10.811,265.31
6,1,20.692,332.21
7,1,39.348,404.15
```

Figure 3.4: GSLIB input format.

The program is run in three steps, step 1 will create a project folder with a grid template and plugins template file. The grid template file contains the specifications and the file location of the input block model, and the plugins template file is used to select the modules of the algorithm that will be used. Step 2 will read the plugins template file and read out from the templates file that is created for every plugin, which parameters will be required for the plugins that were selected. From these templates it creates a configure template file into the project folder, in this file all parameters required by the chosen plugins are stored. Step 3 will then read the input .gslib file, create the grid object, configure and build the algorithm, and run the optimization algorithm according to all specifications and parameters that were selected in steps 1 and 2. These steps are found in the top-left of figure 3.5.

The algorithm will first create an initial solution, assigning a destination to all nodes that are in the grid and stores these in a nodes data-frame that also contains the grade values of the nodes. Then it calculates the misfit values of all the possible frames in the solution and stores these misfits in the misfits data-frame. It will also calculate the minimal misfit value of all nodes in the grid, this is the minimal misfit value of

any frame a node lies inside of, and stores this in the nodes data-frame. From these data-frames it will calculate the penalty. The objective value of the solution is calculated with the destinations data-frame, the grade values and the assigned penalty using the equations 3.1 and 3.2. The new objective value is then compared to the old objective value and the new solution is accepted or rejected by the Metropolis algorithm. If the new solution is accepted it is saved in the nodes data-frame as the 'old solution'. If the new solution is rejected it is forgotten and the old solution remains the 'old solution'. After this, it is checked if the stop criteria are met, if they are met the algorithm is terminated and the old solution is the final solution. If they are not met the algorithm continues by lowering the temperature. After the temperature is lowered the old solution is perturbed into a new solution, and this new solution is saved in the nodes data-frame as 'new solution'. The misfits data-frame and the minimal misfits are updated for all the frames and nodes that were affected by the perturbation, this way the whole misfit data-frame doesn't have to be calculated each cycle, saving computing time. These updated misfits data-frame and minimal misfits are then used to calculate the new penalty and the whole cycle starts again. This is continued until the stop criteria are met.

3.8.1 Modules summary

The different modules that were created for the program and were compared in this thesis are summarized in table 3.3. There are no modules created for the hard constraint, this was because of their limitations on the searching capability of the algorithm, as explained in section 3.2.

Table 3.3: Simulated annealing modules

Module	Code	Description
Initial solution	a01	Pre-optimized initial solution
	a02	Random initial solution
Hard constraint	b00	-
Penalty function	c01	Sum of all misfit values
	c02	Minimal misfit value per node
	c03	Combination of c01 and c02
Objective function	d01	SMU value
Accept/reject	e01	Metropolis algorithm
Temperature	f01	Linear cooling
	f02	Quadratic cooling
	f03	Exponential cooling
	f04	Trigonometric cooling
Perturbation	g01	Random frame to optimal destination
	g02	Random frame to random destination
	g03	Certain probability to select boundary frame
Stop	h01	Number of cycles
	h02	Number of cycles or consecutive rejects

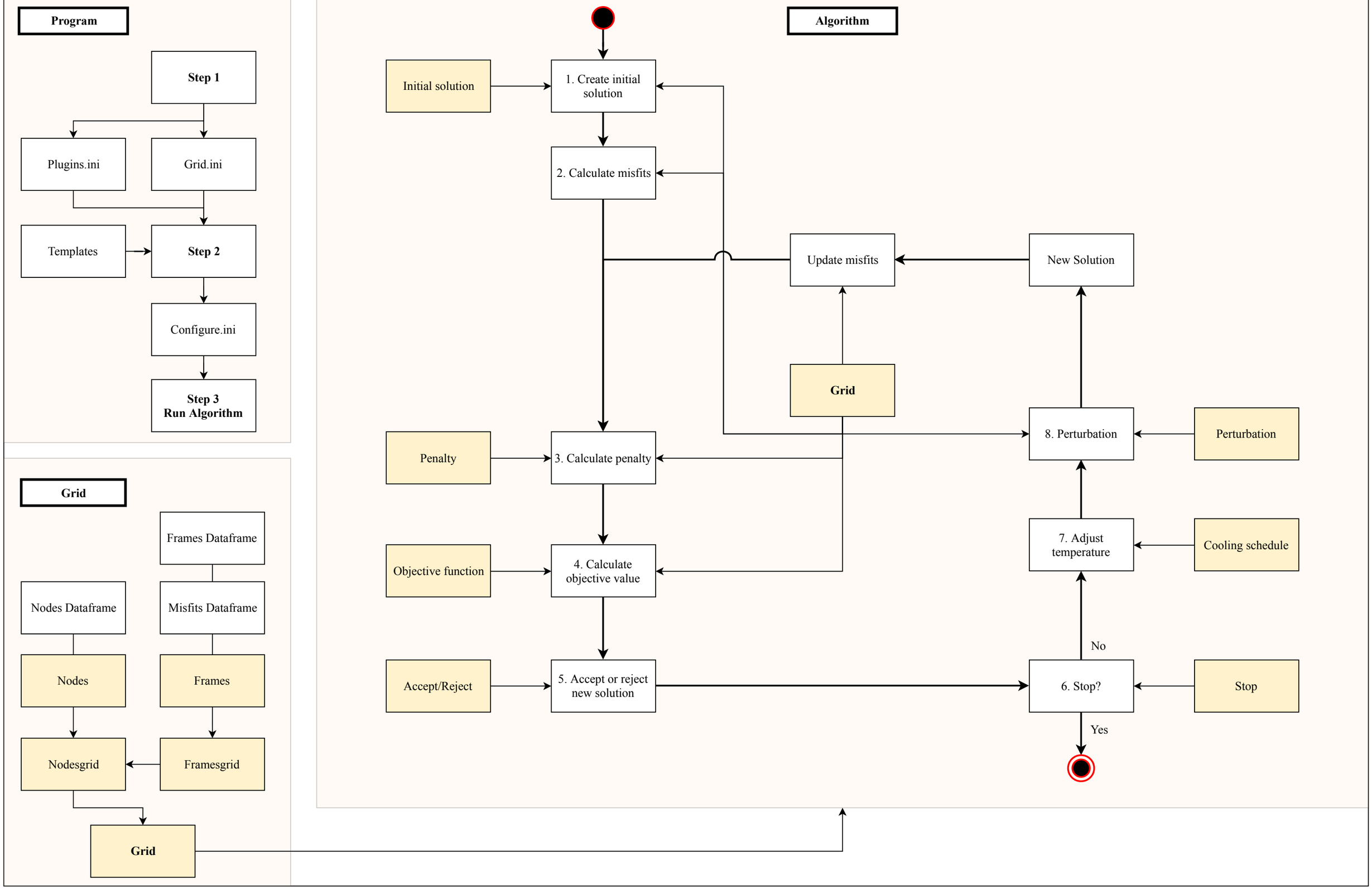


Figure 3.5: Program diagram.

3.8.2 Applicability program

The program is very flexible in its applicability, to make it useful for any mining operation. The objective function can be formulated in any way the value is to be assigned to a block, this can be customized for any mining operation. The program can handle any number of destinations, and any number of ore-types. However, this will have influence on the computation time as the possibilities will increase significantly, this influence is not investigated in this thesis. The penalty system of the program allows for any MMU size and dimensions, and a mining direction can be easily incorporated by choosing a m x n MMU. It is, however, impossible to accept both m x n and n x m clusters in the same solution, so the minimal mining width can not be twisted. The frequent switching between destinations can be reduced by applying a small penalty to destination boundaries. The program can use different block sizes and tonnages to calculate the objective value.

3.8.3 Computer specifications

The tests were run on a laptop, the exact time of the algorithms cannot be compared due to the fact that different algorithms were run simultaneously or other programs have been running on the laptop at the same time as the algorithm. This will have influenced the time performance of the algorithms. To compare the results on their computing time performance the number of cycles was used instead of time. The cycle time is mainly dependent on the cluster size, so when this is kept constant the results of different runs can still be compared on their time performance. To give an indication of the computational complexity of the algorithm the specifications of the laptop that was used are listed in table 3.4. A run of the algorithm on the base case, which is described in section 4.4, takes around 70 minutes to complete.

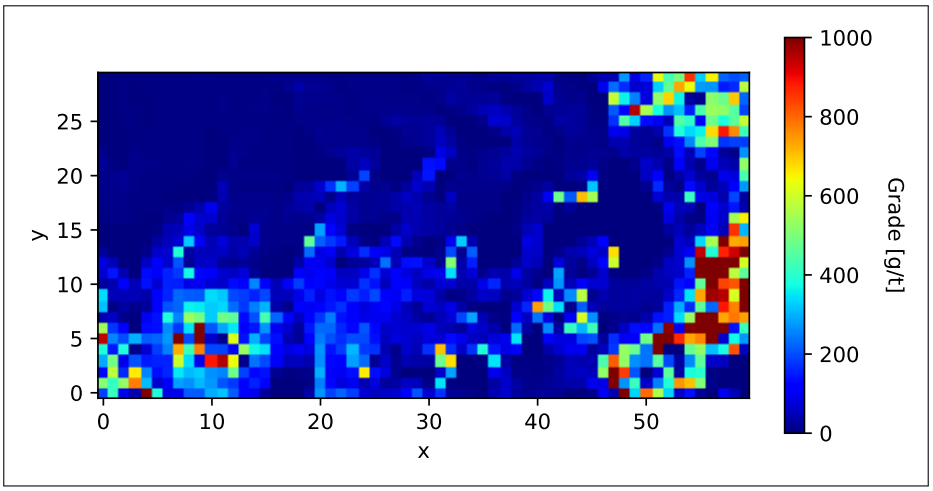
Table 3.4: Laptop specifications

Specifications	**Type**
Processor	Intel(R) Core(TM) i7-3740QM CPU @ 2.70GHz
RAM memory	28.0 GB
Operating system	Windows 10 Pro 64-bits
Hard drive	Samsung SSD 840 EVO 250GB

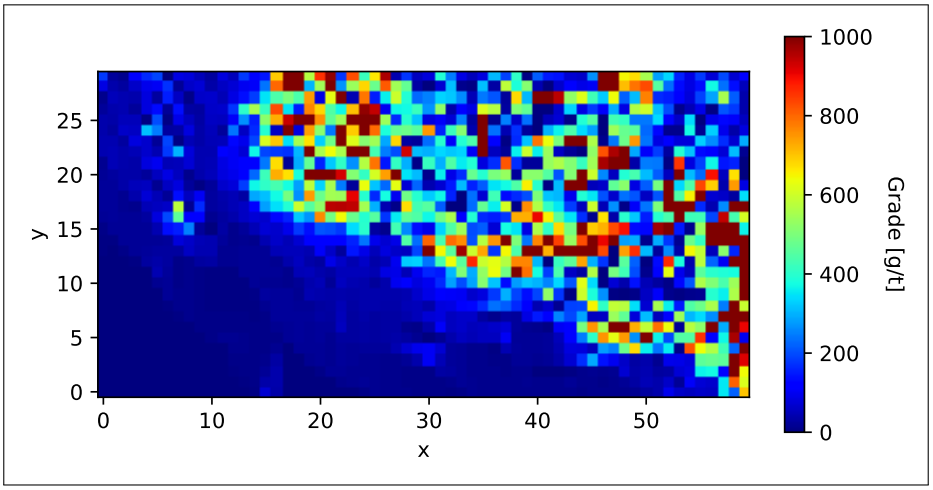
3.9 Cases

To assess the performance of the program, five different cases were investigated. They were used to find and compare appropriate penalty factors and the required number of cycles. These are dependent on the grid size, the grade values of the SMU's and the mining parameters. Also the applicability of the program on grids with different grade distributions was assessed. The cases were created from different parts of the Walkerlake-dataset.

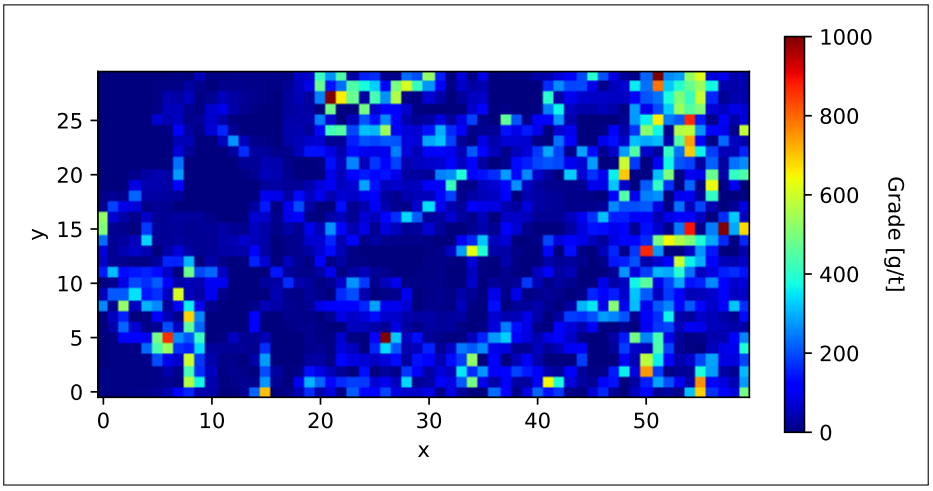
The first three cases all have a grid size of 60 x 30 x 1 SMU's. The dimensions of the SMU's is irrelevant for the optimization program, but meters can be assumed. Grade maps of the first three cases are displayed in figure 3.6. These three cases were chosen because of their different grade distributions. Case 1 has a heterogeneous grade distribution with some high grade pockets. Case 2 has a distinct border between the high grade and low grade areas, and case 3 has a more homogeneous relatively low-grade distribution. The maximum, average and median grade values of these cases are shown in table 3.5.



(a) Grademap case1



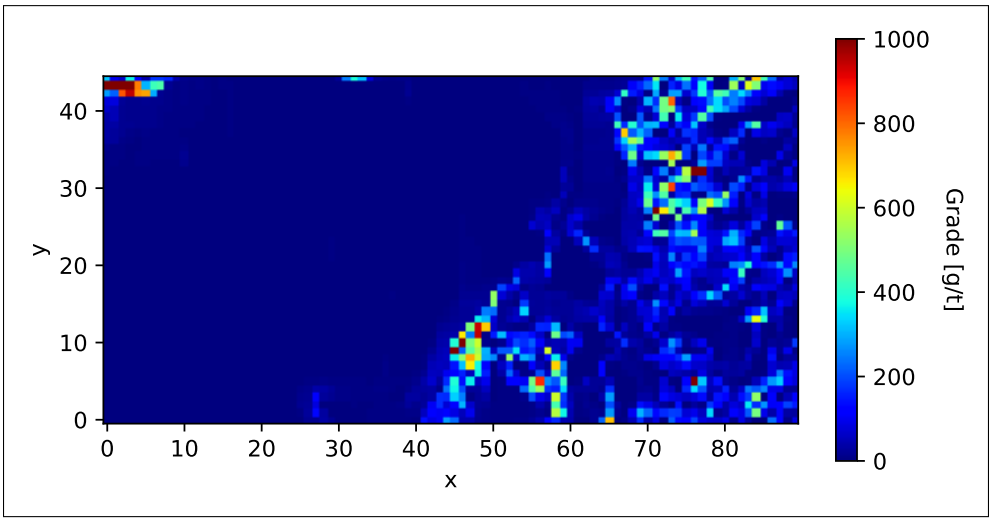
(b) Grademap case2



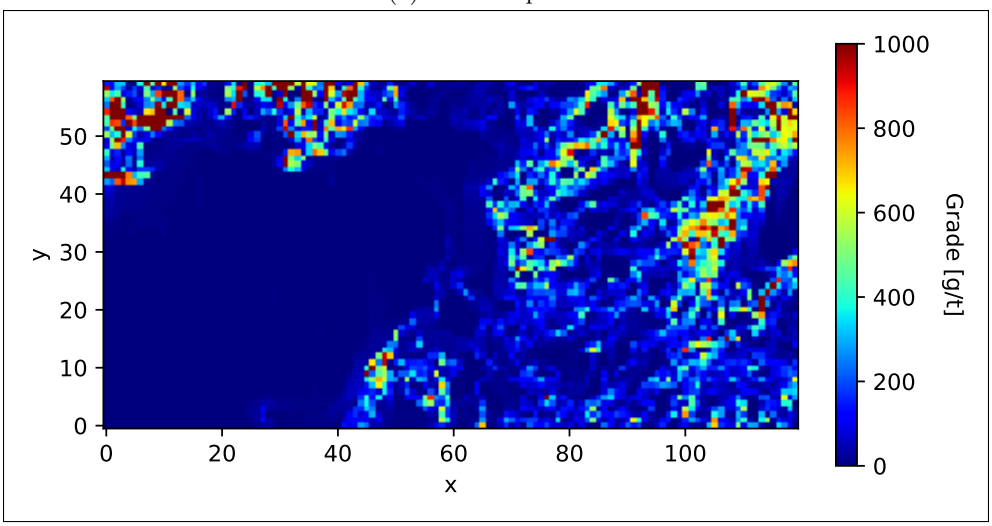
(c) Grademap case3

Figure 3.6: Grademaps cases 1-3

Case 4 and 5 have a larger grid size, of 90 x 45 x 1 and 120 x 60 x 1 SMU's respectively. These cases were used to investigate the effect of the grid size on the required number of cycles of the algorithm. Grade maps of these grids are shown in figure 3.7, and the maximum, average and median grade values of these cases are shown in table 3.5.



(a) Grademap case4



(b) Grademap case5

Figure 3.7: Grademaps cases 4 & 5

Table 3.5: Grade distribution cases 1-5

	Case1	**Case2**	**Case3**	**Case4**	**Case5**
Maximum grade [g/t]	2616.31	3515.96	1278.21	2602.65	3162.61
Average grade [g/t]	125.48	252.44	101.93	46.78	121.44
Median grade [g/t]	32.28	86.84	46.13	14.55	2.21

4 Results and discussion

4.1 Penalty factor

The penalty factor should be adapted to each different case, since it should force the solution to comply to the mining constraints. To ensure the solution complies to the constraints, the penalty factor could be chosen as the maximal profit difference between different destinations, this depends on the grade values of the grid and the mining parameters. However, choosing a penalty factor this high is often unnecessary and if the penalty factor is chosen too high it will affect the searching capability of the algorithm.

To investigate the appropriate height for the penalty factor to make sure the algorithm removes all penalty violations in the final phase of the run, the algorithm was run at different penalty factors for case 1-3. The algorithm was run at a high temperature for an amount of cycles to ensure violations of the mining constraint, and then the temperature was lowered to zero to let the algorithm converge until 1500 consecutive perturbations were rejected. This is the same stopping criteria that was used in the base-case. When the temperature is zero, no more worse solutions will be accepted and the algorithm will focus increasing the objective value by removing the penalty violations, given that the penalty is high enough. Table 4.1 shows the grades and the profit difference between sending the blocks to the waste dump or to the processing plant of the maximum, average and median grade value of the blocks in case 1, 2 and 3. Table 4.2 shows the number of blocks violating the mining constraint after running the algorithm and letting it converge.

Table 4.1: Grade values cases

	Case1	**Case2**	**Case3**
Maximum grade [g/t]	2616.31	3515.96	1278.21
Average grade [g/t]	125.48	252.44	101.93
Median grade [g/t]	32.28	86.84	46.13
Maximum profit difference [$]	103652.30	139638.50	50128.36
Average profit difference [$]	4019.33	9097.54	3077.11
Median profit difference [$]	291.40	2473.68	845.26

As can be noticed the penalty violations are not mainly dependent on the height of the grade values of the grid, as case 2 has the highest grade values but requires the smallest penalty factor, and case 3 has the lowest grade values and requires the largest penalty factor to remove the constraint violations.
This indicates that the magnitude of the penalty factor is more dependent on the grade distribution than on the grade values. As can be seen in the grade maps of cases 1-3 in figure 3.6, case 2 has a sharp and defined boundary between the high grade and low grade areas, whereas case 3 has a more homogeneous grade distribution. Both the grade values and the grade distribution of case 1 lie in between those of case

Table 4.2: Penalty factor

	Constraint violations		
Penalty factor	**Case1**	**Case2**	**Case3**
500	14	4	31
625	18	0	16
750	9	1	11
875	4	0	3
1000	5	0	4
1125	0	0	7
1250	0	0	9
1500	0	-	0
2000	0	-	2
2500	-	-	1
3000	-	-	5
3500	-	-	3
4000	-	-	1
4500	-	-	0
5000	-	-	0

2 and 3, and also the required penalty factor lies in between those of case 2 and 3.

This can be explained by the need to group high and low grade blocks together in more spread out grade distributions. To illustrate this effect figure 4.1 shows a 4 x 4 cluster of 14 waste blocks and 2 ore blocks. When the value of these two ore blocks is higher than the penalty that is assigned to them for violating the mining constraint, but lower than the added processing cost for 14 waste blocks, sending the group to a single destination will decrease the objective value.

For example: the white blocks have a value of 0, and the blue blocks have a value of 15000$. The penalty factor is 5000, the mining cost 1000$ and the processing cost is also 1000$.

The objective value of current solution is 16 times the mining cost, 2 times the processing cost for the blocks send to the processing plant, 2 penalty violations of -5000, and the sum of the value of all blocks: $16 * -1000 + 2 * -1000 + 2 * -5000 + 2 * 15000 = 2000\$$

If the perturbation changes the frame to waste, the objective value of the solution would be 16 times the mining cost of -1000: $16 * -1000 = -16000\$$.

If the perturbation changes the frame to ore, the objective value of the solution would be, sixteen times the mining and processing cost, plus the sum of the value of all blocks : $16*-1000+16*-1000+2*15000 = -2000\$$

Since the current value of the solution is higher than the values of sending the frame to a single destination, the perturbation of this group of blocks to a single destination to remove the constraint violation is unlikely to happen with a penalty factor of 5000, and even impossible when the temperature is 0 and no worse solutions are accepted.

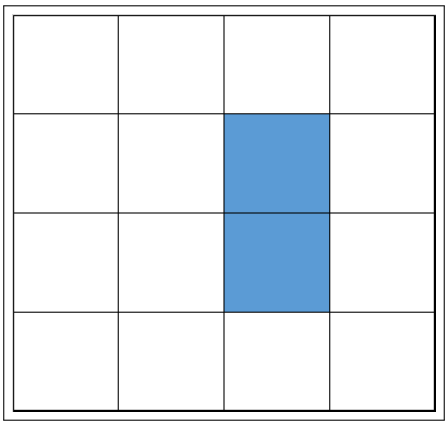


Figure 4.1: Changing a group of blocks.

For the testing of the algorithm a penalty factor of 5000 was chosen, as this will force the solution to comply to the mining constraint and did not seem to impact the searching capability.

4.2 Initial temperature

To find an appropriate initial temperature with an initial acceptance ratio of ≈ 0.8, the algorithm was run at a constant temperature of 1000, for 2000 perturbations. The initial temperature was doubled every time the acceptance ratio was lower than the required ratio. The resulting graph of temperature versus acceptance ratio of case 1, with the base-case parameters is shown in figure 4.2. Based on this graph an initial temperature of 100000 was chosen for the algorithm.

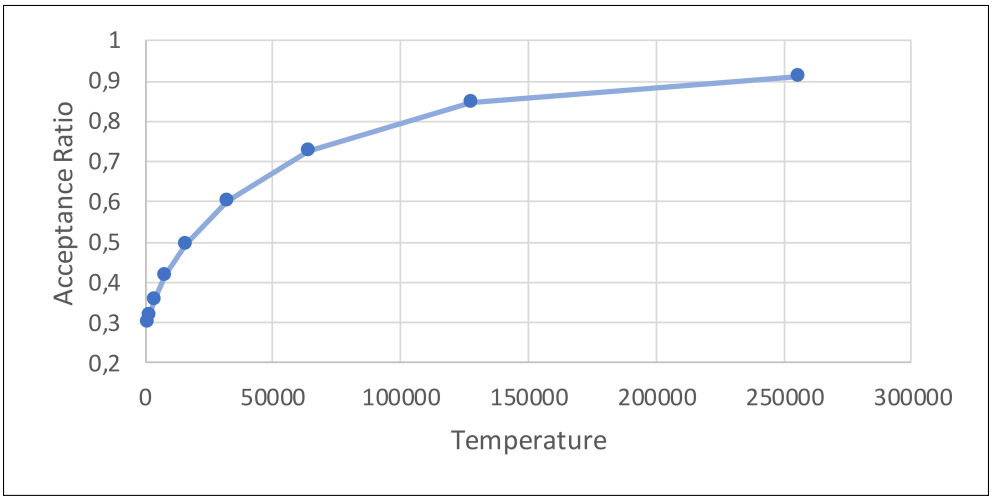


Figure 4.2: Acceptance ratio at different temperatures.

As can be seen in the graph, the acceptance ratio of the algorithm has an exponential decay with a decreasing temperature. This means that when, for example a linear cooling schedule would be used, the acceptance ratio will remain high throughout most of the cycles and only near the end of the algorithm will it decrease quickly to zero. This will result in less time for the algorithm to converge into the local optimum at the end of the algorithm, and to remove penalty violations. This will need to be taken into account when considering a temperature schedule.

4.3 Stop criteria

The amount of perturbations that the algorithm needs to perform to initially give a proper search of the solution space and eventually converge into the optimum is also case dependent. The extent of the solution space depends on the size of the grid, the cluster size, and the number of possible destinations to which the blocks are assigned. In this thesis only two destinations were used, ore and waste. Two types of stopping criteria were investigated.

The first type is a fixed number of cycles, this is the total number of cycles, meaning the number of cycles of the temperature schedule and the number of cycles the algorithm is ran at temperature zero to converge into the found optimum. To asses how much perturbations are necessary to properly search the solution space of a 60 x 30 nodes grid, with a MMU-size of 3 x 3 SMU's, and two material destinations, different number of trials at each cycle were used. The amount of temperature cycles was chosen at 1000, and the post-temperature schedule cycles at 200. For example, if 5 trials per cycle are chosen the total number of perturbations is 6000, and with 25 trials per cycle the total is 30000. The results are shown if figure 4.3.

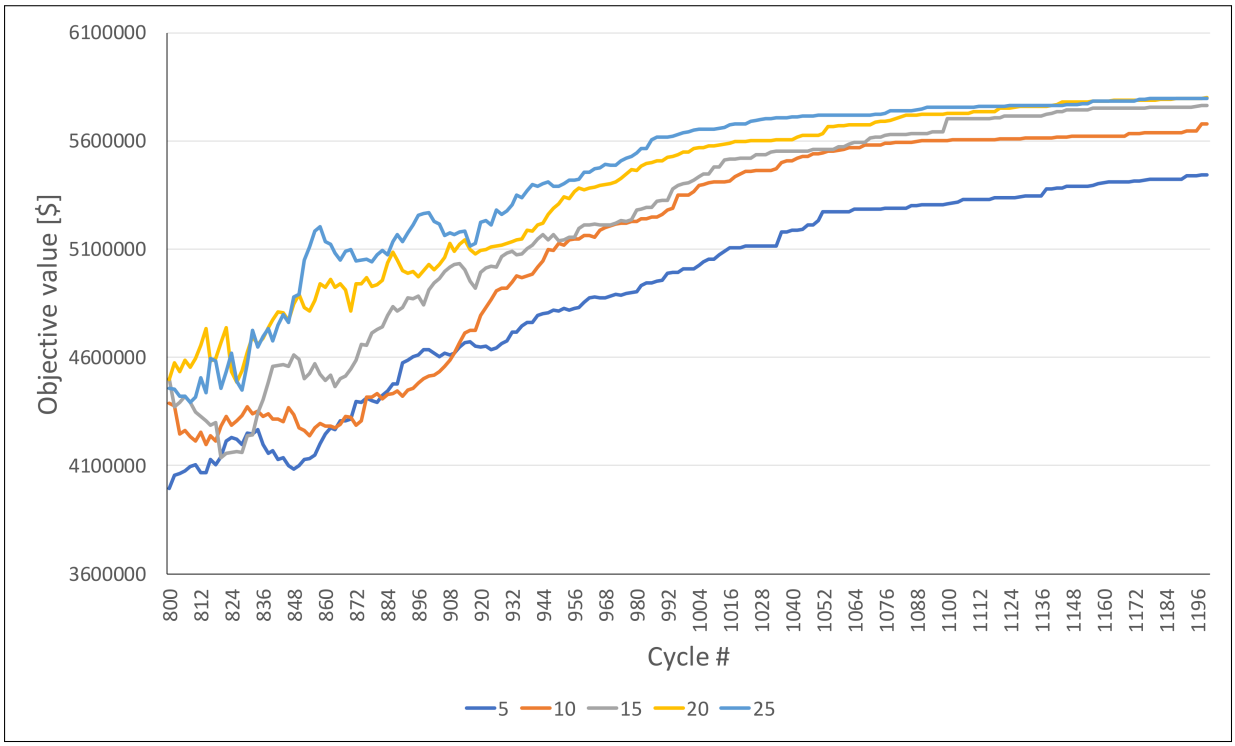


Figure 4.3: Effect of number of trials per cycle on solution improvement.

The runs at 5 and 10 trials per cycle have clearly not approached the global optimum, as the other solutions have a higher objective value. The runs at 15, 20 and 25 trials per cycle do converge to a similar optimum. The amount of trials per cycle that was used in this thesis was 15, because this will result in a near optimal solution within a reasonable computing time. In real life applications of the algorithm, where more powerful computers can be used and the stakes are higher, a higher amount of trials per cycle might be used to ensure a better search of the solution space and a better convergence into the found local optimum.
The second type of stop criteria that was investigated terminates the algorithm after a certain number of consecutive perturbations is rejected. This typically happens after the algorithm has completed the

cooling schedule, the temperature is zero and no worse solutions are accepted anymore. The amount of consecutive perturbations regulates for how many perturbations the algorithm will search for a better solution without improvement. Different numbers of consecutive perturbations were tested and their increase in objective value versus the amount of cycles the algorithm needed to run were compared. The results of these runs from cycle 1000, when the temperature reaches zero, until the termination of the algorithm are shown in figure 4.4.

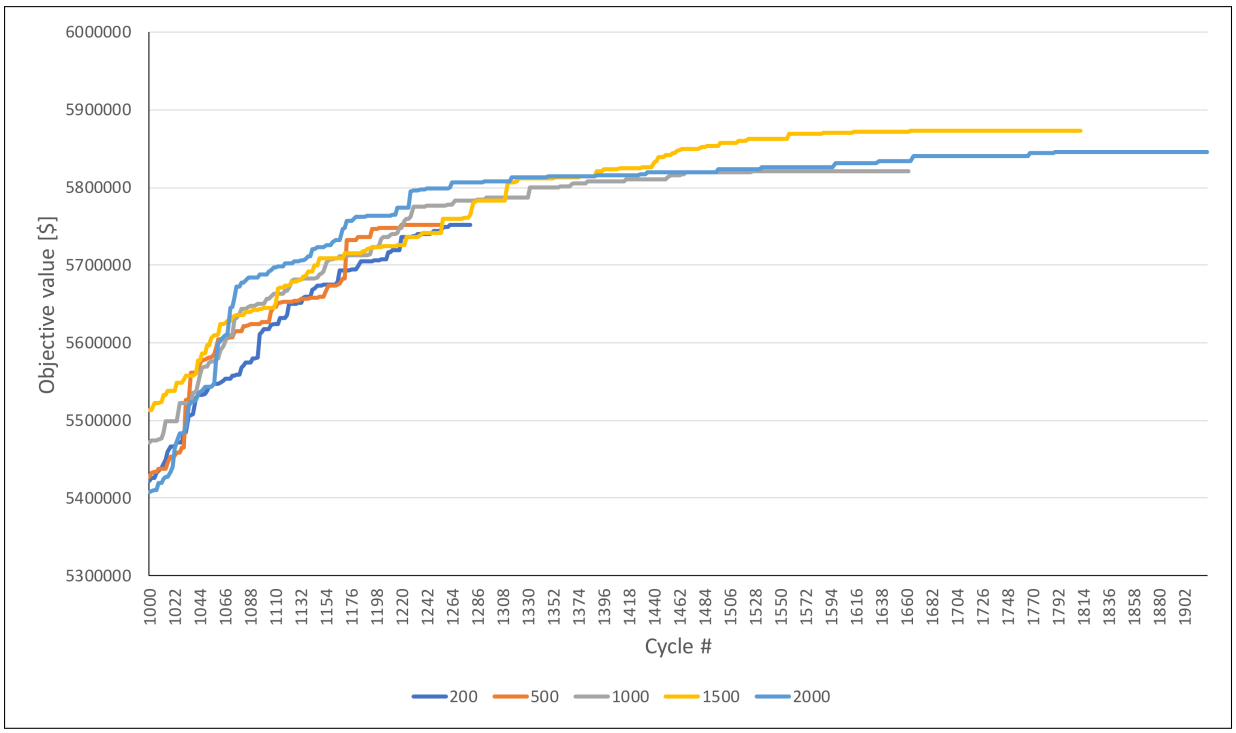


Figure 4.4: Effect of number of consecutive rejects on solution improvement.

Based on these results a consecutive number of rejects of 1500 was chosen, the improvement of the solution versus the extra run time was decent. The 2000 case shows little improvement over a significantly longer run-time. In real-life cases, the number of consecutive rejects before algorithm termination might be increased to ensure a better convergence. As it will always ensure a better result at only the cost of computing time.

4.4 Base-case

To compare the influence of all modules and parameters on the performance of the algorithm, a base-case was defined against which single or combinations of alterations were compared. For all runs of the algorithm in this chapter, unless stated otherwise, the base-case setup was used. The base-case comprises the most standard and basic modules that were created for the algorithm. The MMU size of the base-case is 3 x 3 SMU's. Based on the results of the previous sections in this chapter the penalty factor was chosen at 5000, the initial temperature at 100000, and the algorithm was ran for 15 trials per cycle until 1500 consecutive perturbations were rejected. The temperature schedule was run for 1000 cycles after which the algorithm was run at temperature 0, until the stop-criterion was met. The modules and the parameter values of the base-case are listed in tables 4.3 and 4.4.

The penalty value, the objective value and the real value of the solution during the course of a run of the base-case on case 1 are shown in figure 4.5. These results show that the algorithm is working properly.

Table 4.3: Base-case modules

Module	Code	Description
Initial solution	a02	Random initial solution
Penalty function	c02	Minimal misfit value per node
Objective function	d01	SMU value
Accept/reject	e01	Metropolis algorithm
Temperature	f01	Linear cooling
Perturbation	g02	Random frame to random destination
Stop	h02	Number of cycles or consecutive rejects

Table 4.4: Base-case parameters

Parameter	Value	Unit
Initial temperature	100000	[-]
Final temperature	0	[-]
Number cycles in schedule	1000	[-]
Trials per cycle	15	[-]
Number of consecutive rejects before stop	1500	[-]
Penalty factor	5000	[-]

In the early stages of the algorithm the penalty is quickly increased and the objective value decreases, this indicates that the algorithm is accepting worse solutions, violating the constraints, and is searching the solution space. In the course of the algorithm it can be noticed that the objective value gradually increases, and that the magnitude of the decreases in the objective value are becoming smaller as the temperature decreases. In the end of the temperature schedule at cycle 1000 there are no more worse solutions accepted and the algorithm converges into the found local optimum. At the end of the algorithm there is no more penalty assigned to the solution, meaning the constraint is not violated anymore.
The final results of the base-case runs on cases 1-3 are summarized in table 4.5. The resulting dig-limits are shown in figures 4.6, 4.7 and 4.8, where blue is the color of the blocks that are send to the processing plant, and white the blocks send to the waste dump. As it is not possible to compare the results to the unknown global optima of the cases, they are compared to the free-selection values. Free-selection means that every block is send to its individual optimal destination, and the mining constraints are disregarded. These values are higher than the real-global optima but still give some indication of the performance of the algorithm. The free selection values of cases 1-3 are shown in table 4.6.

Table 4.5: Base-case results

	Case 1	Case 2	Case 3	Unit
Objective value	5877871	14794671	3986693	[$]
Penalty	0	0	0	[$]
Real value	5877871	14794671	3986693	[$]
Number of perturbations	28770	34380	27690	[-]
Run time	70	95	61	[min]

Table 4.6: Free-selection values

Case	Value	Unit
1	5995648	[$]
2	14869035	[$]
3	4226063	[$]

Figure 4.5: Course of base-case run on case 1.

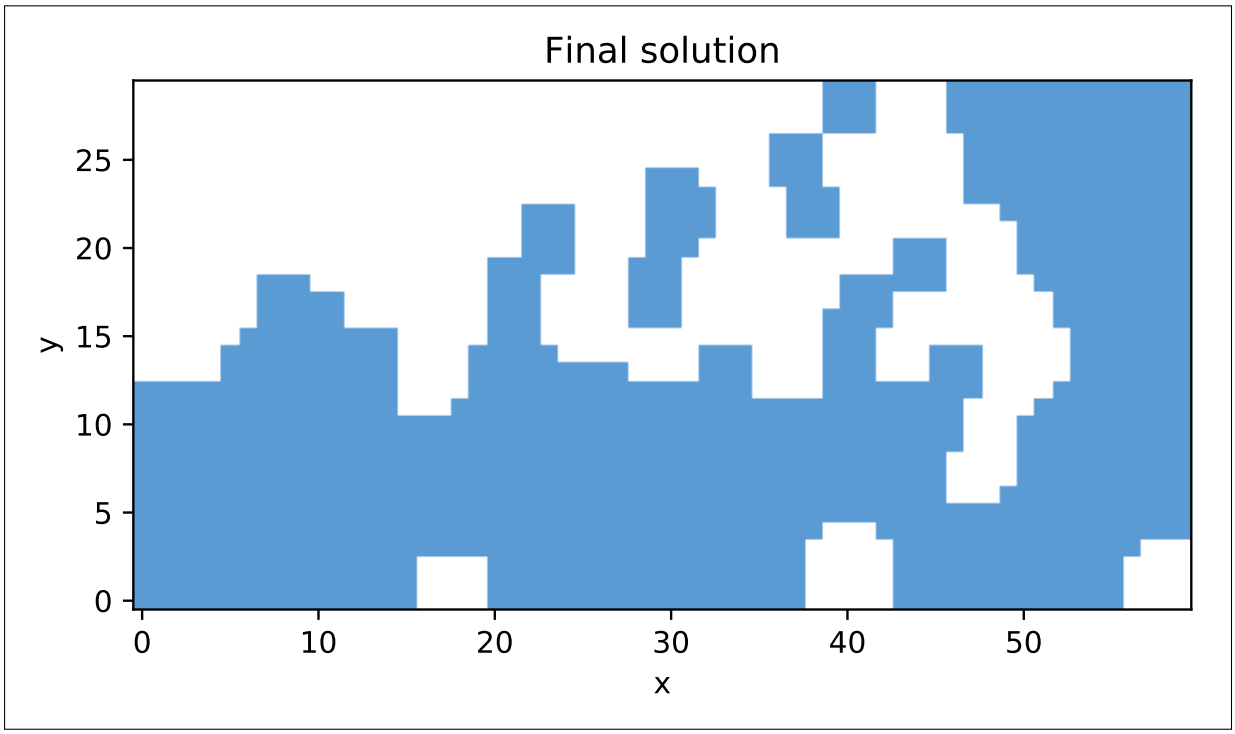


Figure 4.6: Dig-limit design of the base-case on case 1.

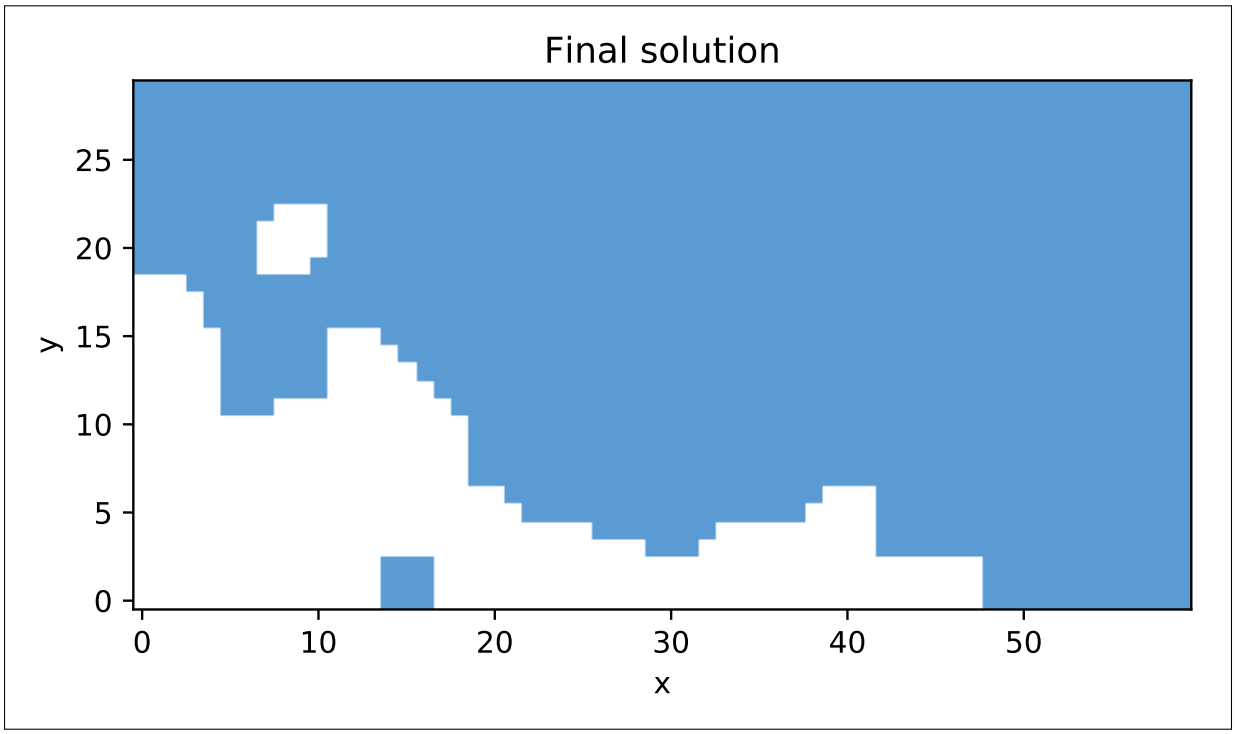


Figure 4.7: Dig-limit design of the base-case on case 2.

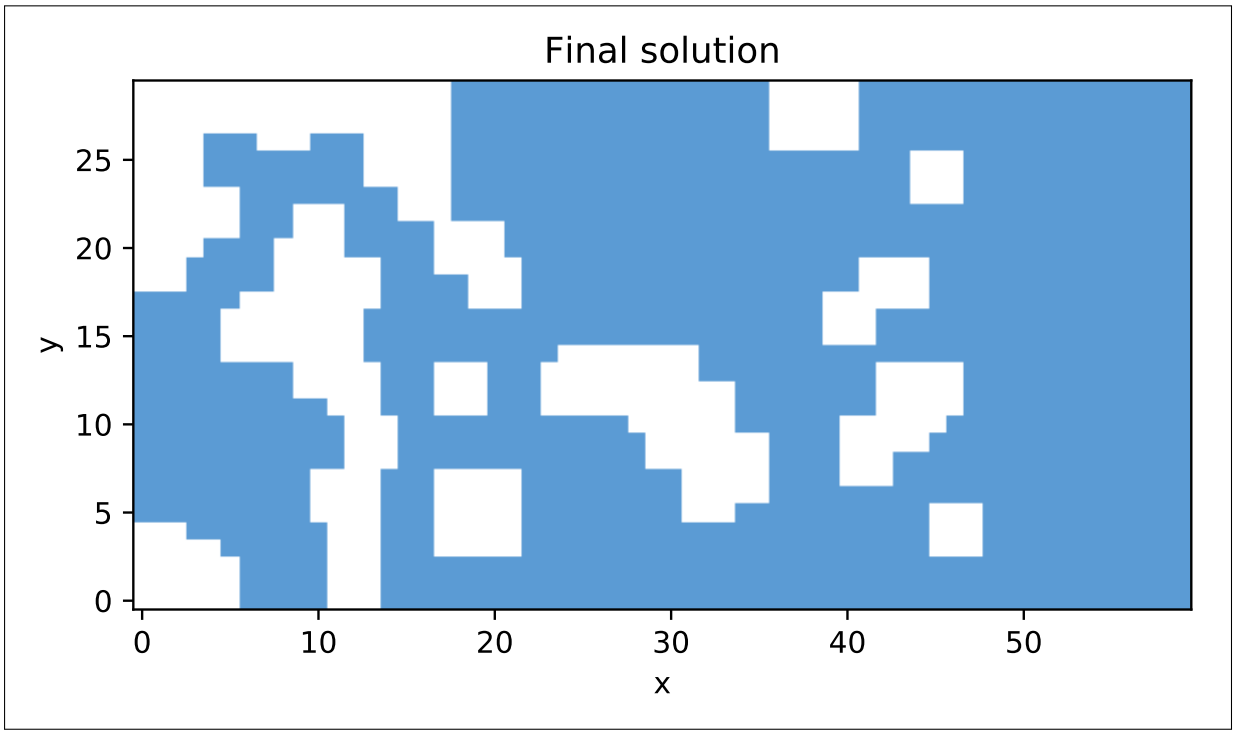


Figure 4.8: Dig-limit design of the base-case on case 3.

When comparing the dig-limit designs to the grade-maps of cases 1-3 in figure 3.6 it is clear that the ore in the dig-limit designs match the high grade areas of the grade-maps. The deviation from the free-selection values is 1.96% for case 1, 0.50% for case 2, and 5.66% for case 3. These objective values of the solutions lie close to the free-selection values, which are higher than the global optimum. The difference in deviation between the cases can be explained by the grade distribution of the cases. Case 2 has a sharp boundary between the high and low-grade areas, thus the optimal dig-limit will be closer to free-selection. Case 3 has a very heterogeneous distribution, where high and low grade blocks are mixed together, in this case the difference between the free-selection value and the global optimum of the dig-limit solution will be bigger, as more blocks will need to be send to a not-optimal destination. Case 1 lies between case 2 and 3 in both grade distribution and deviation from the free-selection value. Since the global optimum of the dig-limit solution always lies below the free-selection value, the 0.50% deviation of case 2 indicates that the algorithm approaches the global optimum very well, at least with less than 0.50% deviation.

In conclusion the algorithm works properly, it fully complies to the spatial mining constraints and the resulting dig-limits have a value close to the free-selection values. It manages these results within a reasonable computing time. It can be noticed that although the algorithm works as it should, there are a lot of smaller destination groups. This might be unfavorable due to the dilution that occurs at destination boundaries and decrease in productivity due to destination switching. To handle these smaller groups, an alternative penalty function has been proposed to reduce these, the results of this function are presented in section 4.6.2.

4.5 Initial solution

Two strategies were tested for the initial solution. The initial solution is the take-off point for the algorithm. When the algorithm is working properly, and the initial search of the solution space is extensive, the method of the initial solution should not influence the final result. Nevertheless different options were investigated to research if there are possible combinations to decrease the run time of the algorithm. The general approach to assign an initial solution was to divide the grid in MMU-size clusters, and assigning a destination to these clusters.

The base case uses a randomized initial solution, which assigns a random destination to the clusters. This initial solution will generally have a low objective value, from which a good, unbiased search of the solution space can take place. However, with little extra computing time, the initial solution can be pre-optimized. This was done by calculating the profit from all the blocks in a frame for all possible destination and assigning the highest profit destination.

When a pre-optimized solution is used, and the base-case initial temperature of 100000 is used, where most of the worse solutions are accepted, the pre-optimized initial solution is wasted as the algorithm will still perform a random search of the whole solution space in the early stages of the algorithm. To investigate the effect of the pre-optimized initial solution at different initial temperatures on the course of the algorithm, it was run at different initial temperatures. The results are shown in figure 4.9 and table 4.7.

Table 4.7: Pre-optimized initial solution at different initial temperatures

Initial temperature	**Objective value [$]**	**Number of perturbations**
100000	5859027	29190
25000	5875978	27450
10000	5874269	23670
5000	5875158	25200

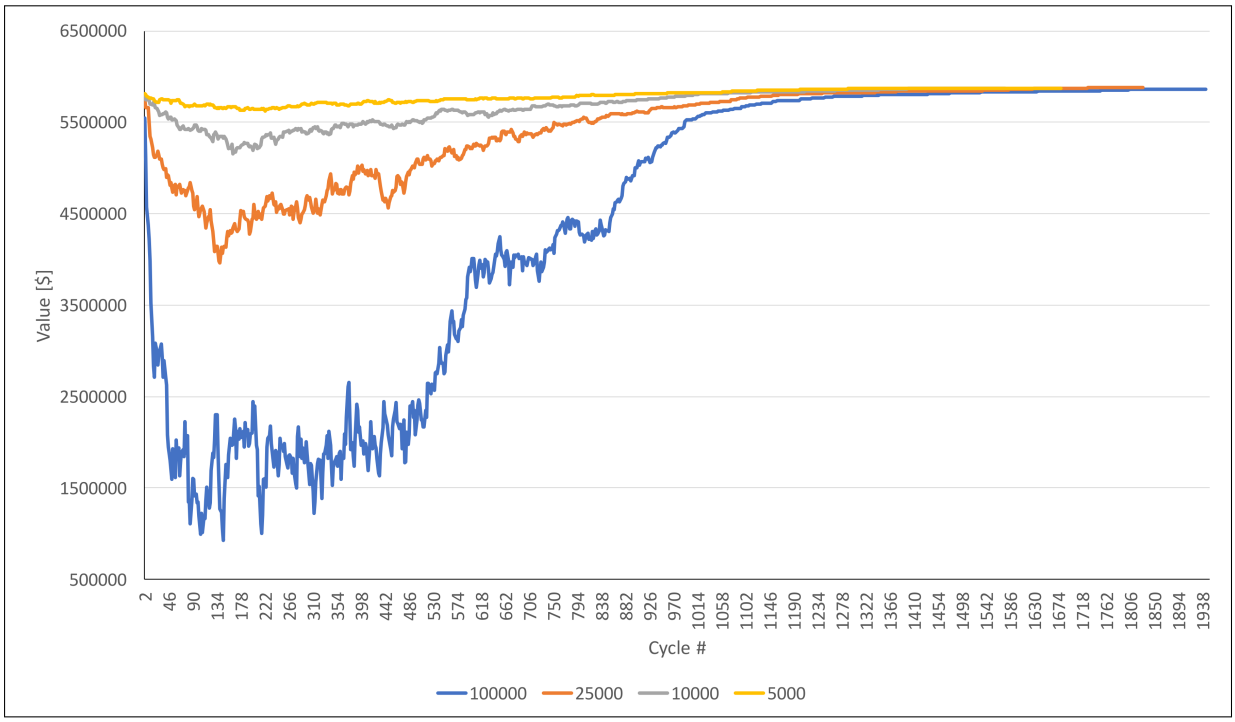


Figure 4.9: Pre-optimized initial solution at different initial temperatures.

It can be noticed that when the initial temperature of 100000 is used, the objective value first decreases to about 10% of its starting value, meaning that the pre-optimized solution is wasted. The final objective values of the algorithm do not change much with the different initial temperatures, and the number of perturbations increases with a higher initial temperature but not by a large amount. This discourages the use for a pre-optimized solution in combination with the normal simulated annealing algorithm.

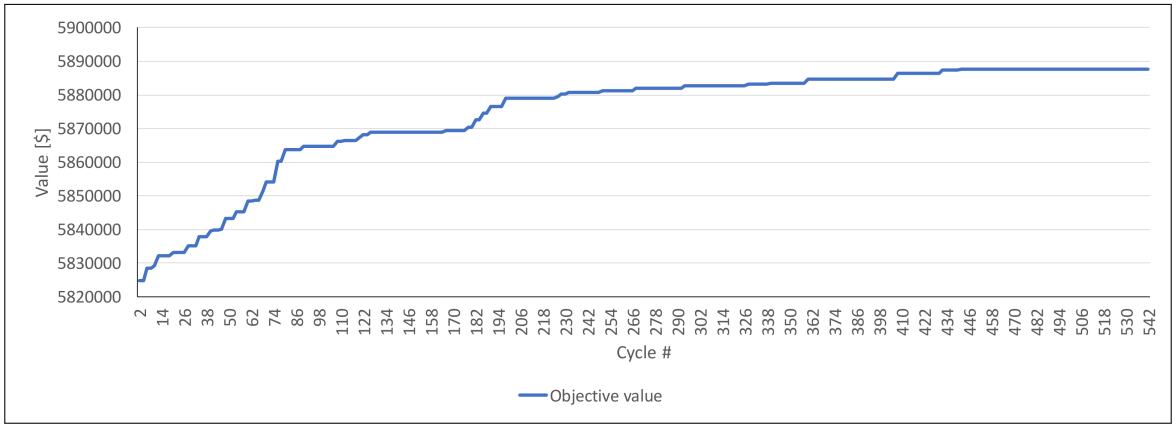
4.5.1 Greedy algorithm

To test how well the pre-optimized solution approaches the global optimum, and to investigate if the pre-optimized solution might be used to limit the computing time, the algorithm was run as a greedy algorithm from the pre-optimized solution. The algorithm is greedy when the initial temperature is zero, and only better solutions are accepted over the whole course of the algorithm. This will directly focus all computing time into converging into the local optimum from where the initial solution takes off. To test the applicability of a greedy algorithm to find a near optimal solution for dig-limit design, it was tested on different cases. Cases 1 to 3 were investigated and the results of these runs are shown in table 4.8 and figure 4.10.

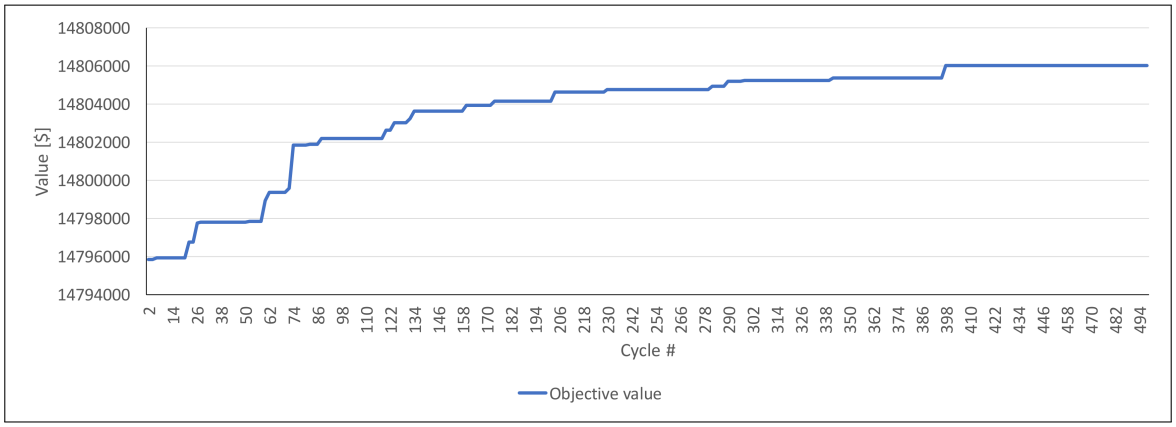
In every case the greedy algorithm has resulted in a better solution than the base-case. It has reached these solutions in a considerable less amount of perturbations, in all three cases in less than a third of the perturbations the base-case required. This indicates that the pre-optimized solution approaches the global optimum better than the random and gradual search performed in the early stages of the base-case simulated annealing algorithm. Figure 4.11 shows the dig-limit design of the initial, pre-optimized, solution and of the final solution of the greedy algorithm on case 1. It can be notice that they are very similar and mainly the boundaries have shifted. Using the pre-optimization with a greedy algorithm might be an interesting combination that can save a significant amount of computing time and result in a high quality dig-limit design.

Table 4.8: Greedy-run results on case 1-3

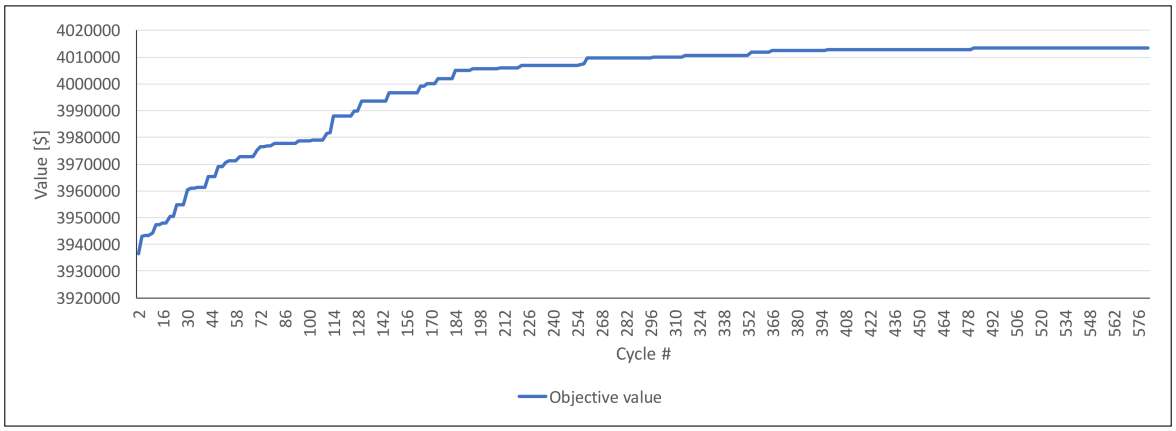
	Case 1	Case 2	Case3
Objective value [$]	5887604	14806052	4013510
Number of perturbations	8130	7470	8730



(a) Objective value of greedy-run on case 1

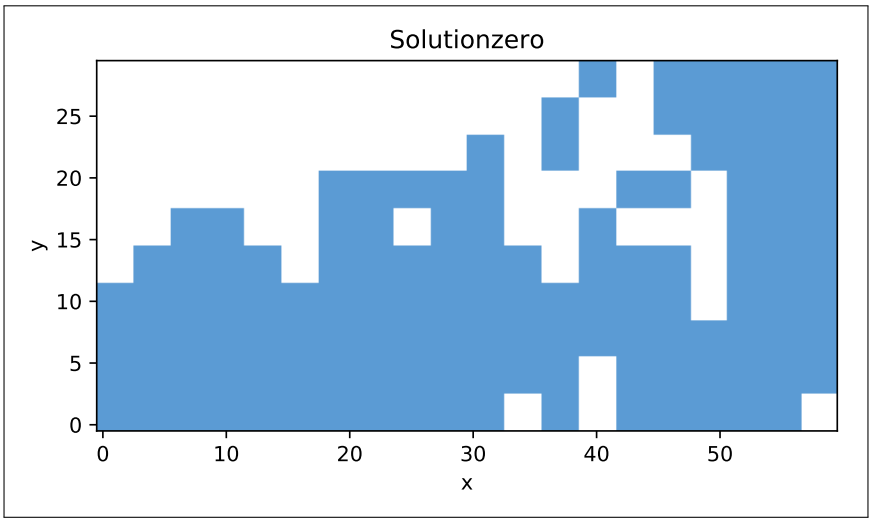


(b) Objective value of greedy-run on case 2

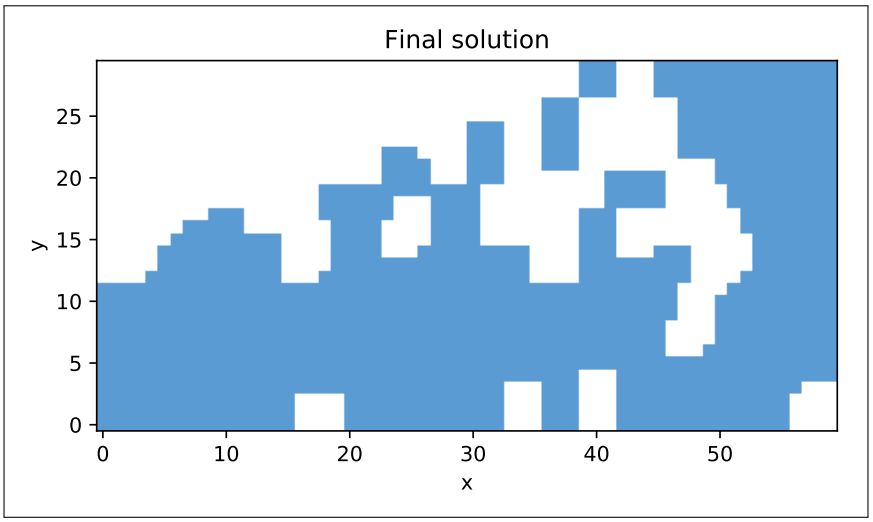


(c) Objective value of greedy-run on case 3

Figure 4.10: Objective value of greedy-runs on cases 1-3



(a) Pre-optimized initial solution



(b) Final solution

Figure 4.11: Initial and final solution of greedy algorithm on case 1

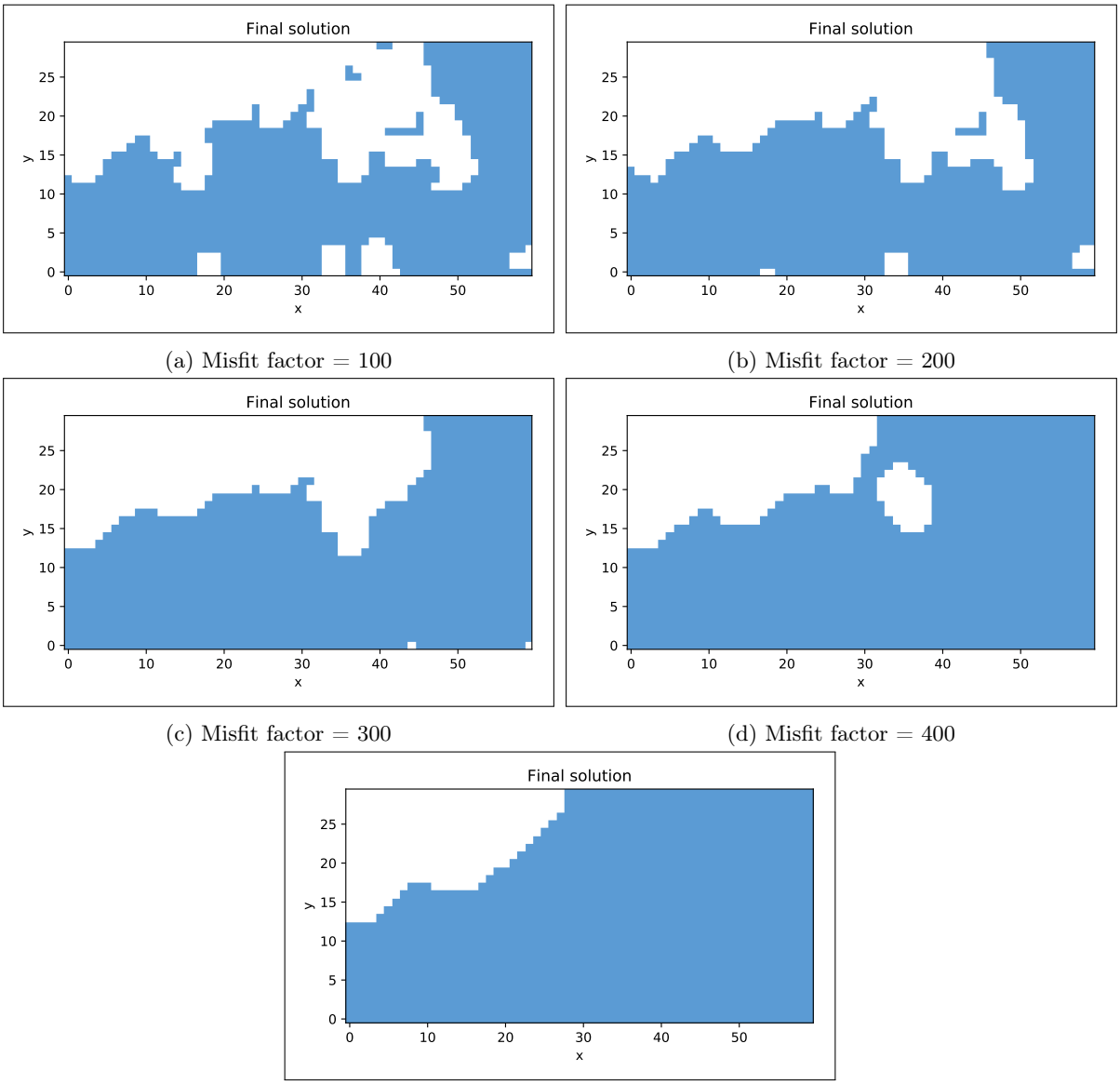
4.6 Penalty

For the penalty function, three different options were investigated. The m x n MMU constraint has proven to be successful with an appropriate penalty factor in section 4.1. The other options for applying a penalty to the solutions are the sum of the misfit values, and a combination of the MMU constraint and the sum of the misfit values, which should combine the compliance to the mining constraints and the discouragement of destination boundaries.

4.6.1 Sum of misfit values

To reduce the frequent switching between material destinations, for instance ore and waste, and to reduce dilution by decreasing the total length of boundaries between destinations, the sum of the misfit values

as penalty function was investigated. The misfit values are appointed to all possible frames in the grid that violate the mining constraint, this will always happen at destination boundaries. This means that the sum of the misfit values as a penalty will discourage both the constraint violations and destination boundaries. To investigate if and how the sum of the misfit values manages this, it was run at different penalty factors. The penalty factor on the total amount of misfits is from this point called the 'misfit factor'. The resulting dig-limit designs of the runs is shown in figure 4.12.



(a) Misfit factor = 100

(b) Misfit factor = 200

(c) Misfit factor = 300

(d) Misfit factor = 400

(e) Misfit factor = 500

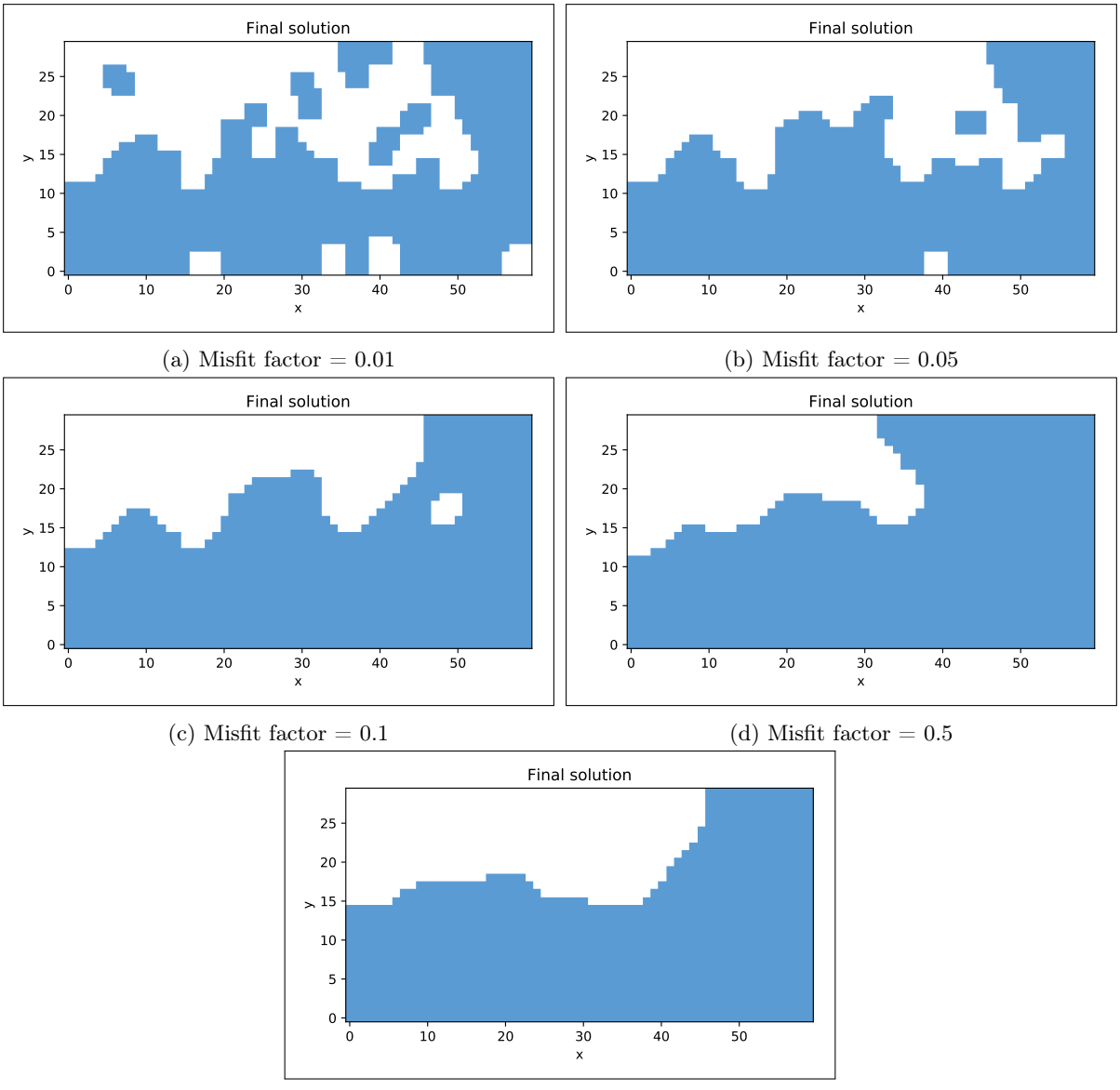
Figure 4.12: Dig-limit designs with different misfit factors for c01 on case 1.

The misfit factors of 100 and 200 do not significantly decrease the destination boundaries, and they also do not make the solution comply to the mining constraints. The misfit factors of 300 and 400 already reduces the destination switching but do not successfully remove the mining constraint violations. The misfit factor of 500 leaves only 2 destination zones with no constraint violations. The sum of the misfit

values does not seem to be a good option for the penalty on its own. It is difficult to control the discouragement of destination boundaries while making sure it complies to the mining constraints.

4.6.2 Constraint penalty and the sum of the misfit values

To attempt to regulate the discouragement of destination switching in combination with forcing the algorithm to comply to the mining constraints, the cluster penalty was combined with a penalty on the total amount of misfits. Multiple misfit factors were used to investigate the effect of the misfit factor on the ore waste boundaries. The resulting dig-limits are shown in figure 4.13.



(a) Misfit factor = 0.01

(b) Misfit factor = 0.05

(c) Misfit factor = 0.1

(d) Misfit factor = 0.5

(e) Misfit factor = 1

Figure 4.13: Dig-limit designs with different misfit factors for c03 on case 1.

The misfit factor of 0.01 does not significantly affect the dig-limit design. A misfit factor of 0.05 removes most of the smaller clusters of blocks with a different destination, but still has a very tortuous boundary between the ore and the waste. The boundary is becoming significantly smoother with a misfit factor of 0.1, and with a misfit factor of 0.5 all smaller clusters of a different destination were eliminated and only 2 destination zones were left. These results are very promising for the use of the algorithm to discourage the destination switching and it means that the extent of the discouragement can be properly regulated by the misfit factor.
To investigate the applicability of the combination of the time saving greedy optimization from a pre-optimized initial solution and the destination switching discouragement, the algorithm was run at temperature 0 from the pre-optimized initial solution with a misfit factor of 0.5. The pre-optimized initial solution does not take the penalty into account, meaning it will disregard the destination boundaries. The resulting dig-limit is shown in figure 4.14. The dig-limit design does manage reduce the boundaries and also complies to the mining constraints. This indicates that the faster, greedy method is also applicable to destination boundary discouragement.

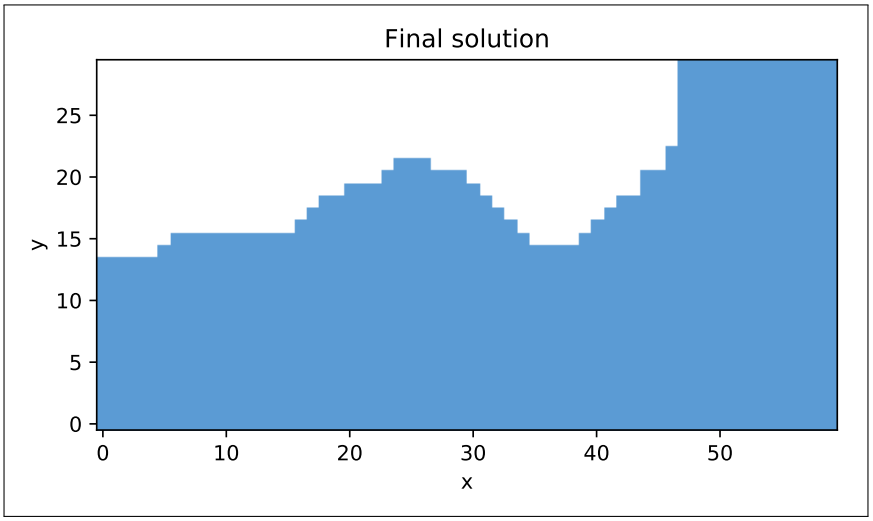


Figure 4.14: Dig-limit design of c03 in combination with greedy algorithm

4.7 Temperature schedule

Four different temperature schedules were investigated in this thesis, which have been introduced in chapter 3.4. The investigated schedules are a linear schedule, a quadratic schedule, an exponential schedule and a trigonometric schedule. The different schedules were tested on case 1. The linear schedule was used in the base-case, the courses of the runs of the other three schedules are shown in figures 4.15, 4.16 and 4.17.
The quadratic cooling schedule cools down fairly quickly, and remains on lower temperatures for a longer amount of cycles. This can be seen in figure 4.15, the convergence starts early and major worse solutions are no longer accepted after the first 110 cycles. The resulting solution is the best solution that was found by the algorithm in this thesis, with an objective value of 5890759 $.

The exponential cooling schedule remains at a high temperature for a longer time, and then quickly cools down to a very low temperature. This results in a long search of the solution space, followed by a quick convergence into the local optimum where no major worse solutions are accepted anymore. This means that the convergence into the local optimum is very abrupt, and the take-off of the convergence happens from a more random solution. This is not preferable for this application of simulated annealing. The final objective value was 5854672 $.

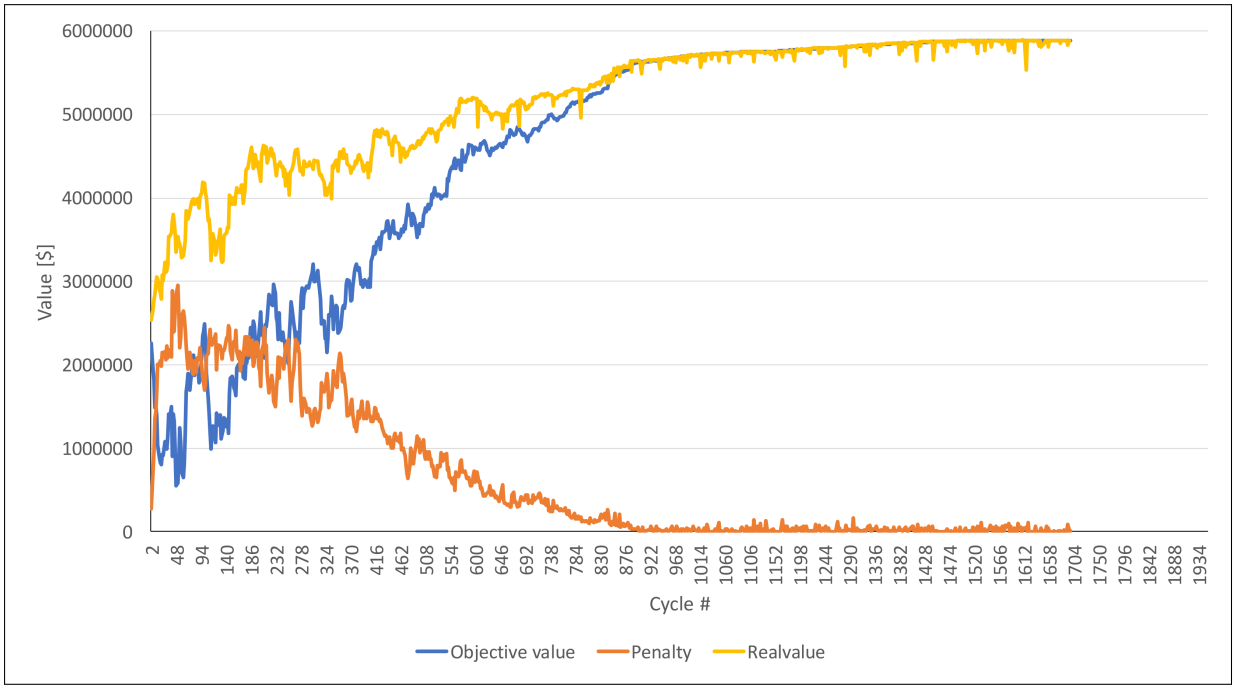


Figure 4.15: Course of run with quadratic cooling schedule.

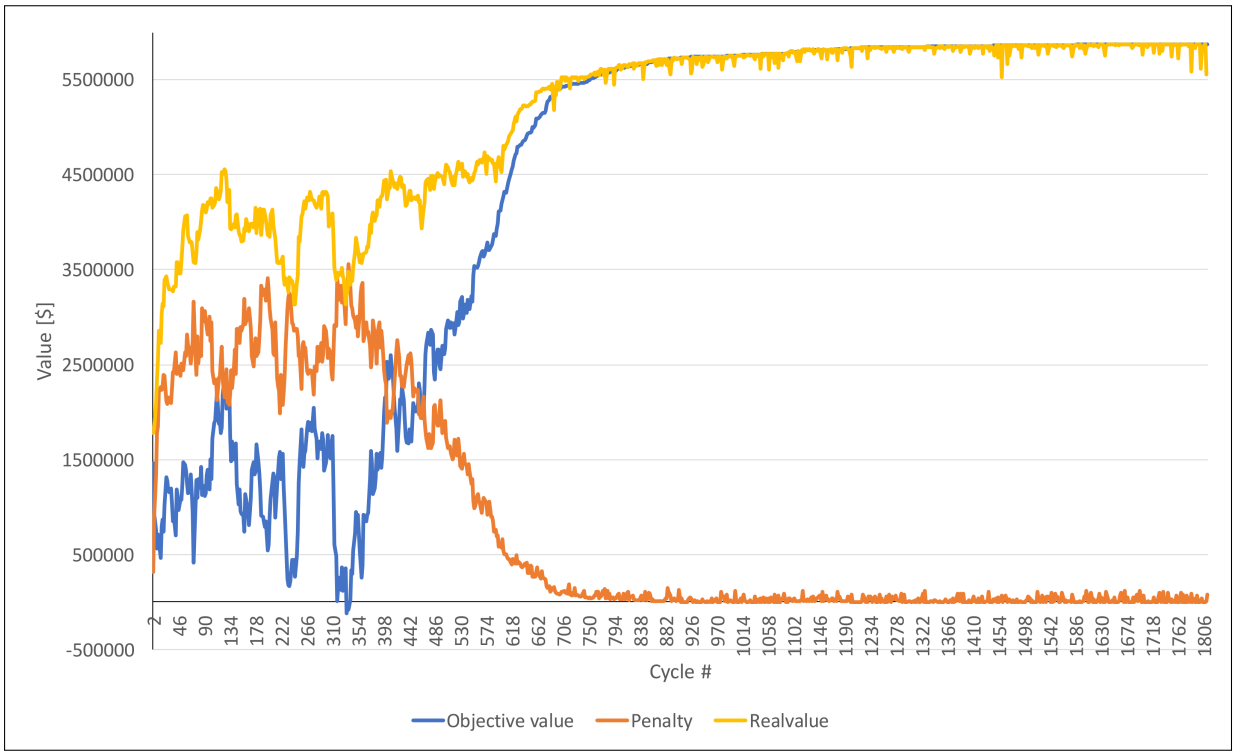


Figure 4.16: Course of run with exponential cooling schedule.

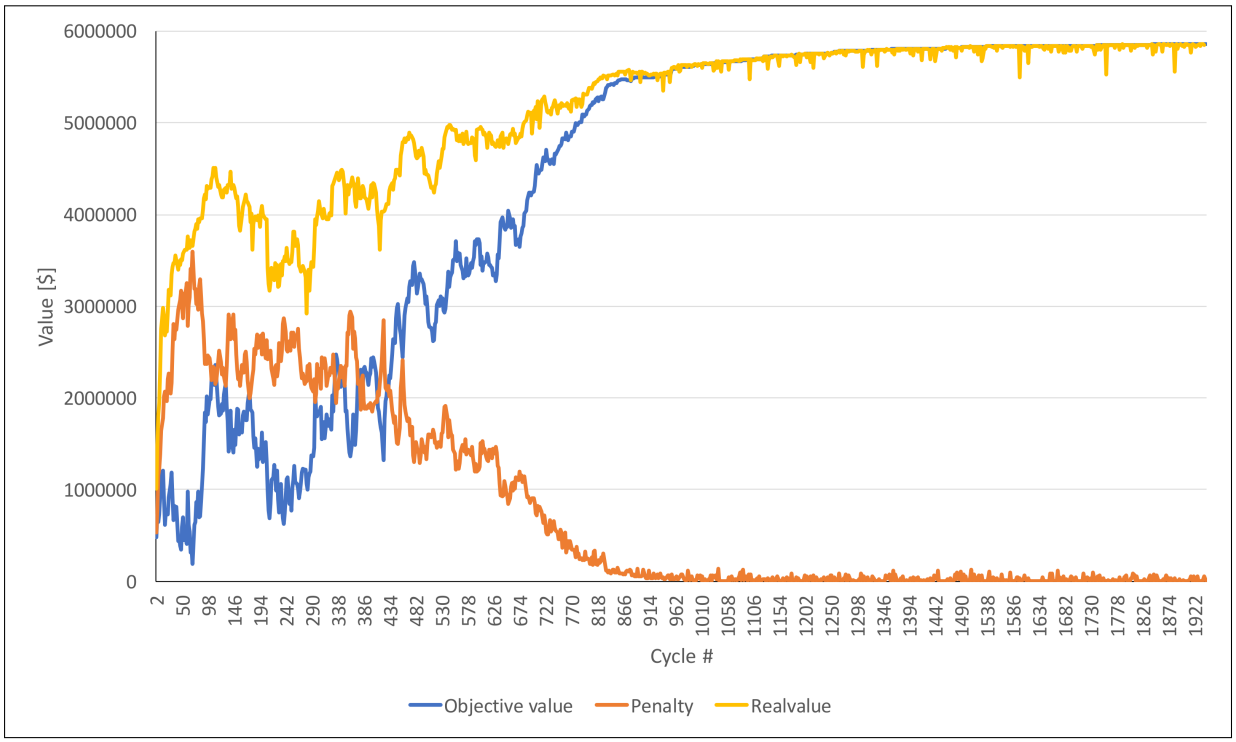


Figure 4.17: Course of run with trigonometric cooling schedule.

The trigonometric cooling schedule is somewhere in between the exponential and the linear cooling schedules. It remain a little longer at the high and low temperatures and spends less time at the middle temperatures, but still accepts some worse solutions to escape from local optima. The final objective value was 5856291 $.

The quadratic schedule seems the most successful schedule for the application on dig-limit optimization. This might be explained by the fact that it counteracts the exponential decay in the acceptance ratio with decreasing temperatures, by spending less cycles at the higher temperatures.

4.8 Perturbation

For the perturbation mechanism, three different options were investigated. The first option is to select a random frame in the grid and change its destination to the one that delivers the maximal profit, but disregards the penalty it might cause. The second mechanism also selects a random frame in the grid, but changes its destination to random, this is the method that is used in the base-case. The third mechanism has a certain probability to select a frame that lies on a destination boundary, or else it will select a random frame in the grid. In both cases the destination is changed to random.

The course of the run of the first mechanism is shown in figure 4.18. The real value of the solution is quickly increased in the first 100 cycles of the algorithm. This is expected since when the algorithm has covered most of the grid once, most of the blocks are send to an optimized destination. The decrease of the penalty is slowly, this might be explained by the fact that every possible frame can only be send to a single destination, this will make it more difficult for the algorithm to remove constraint violations. The objective value of the final solution was 5890655 $, this is very close to the most optimal solution that was found by the algorithm.
The third perturbation mechanism was run at different probabilities for selecting a boundary frame. The

objective values of the solutions in the course of the algorithm are shown in figure 4.19. No significant difference in the convergence of the algorithm could be detected.

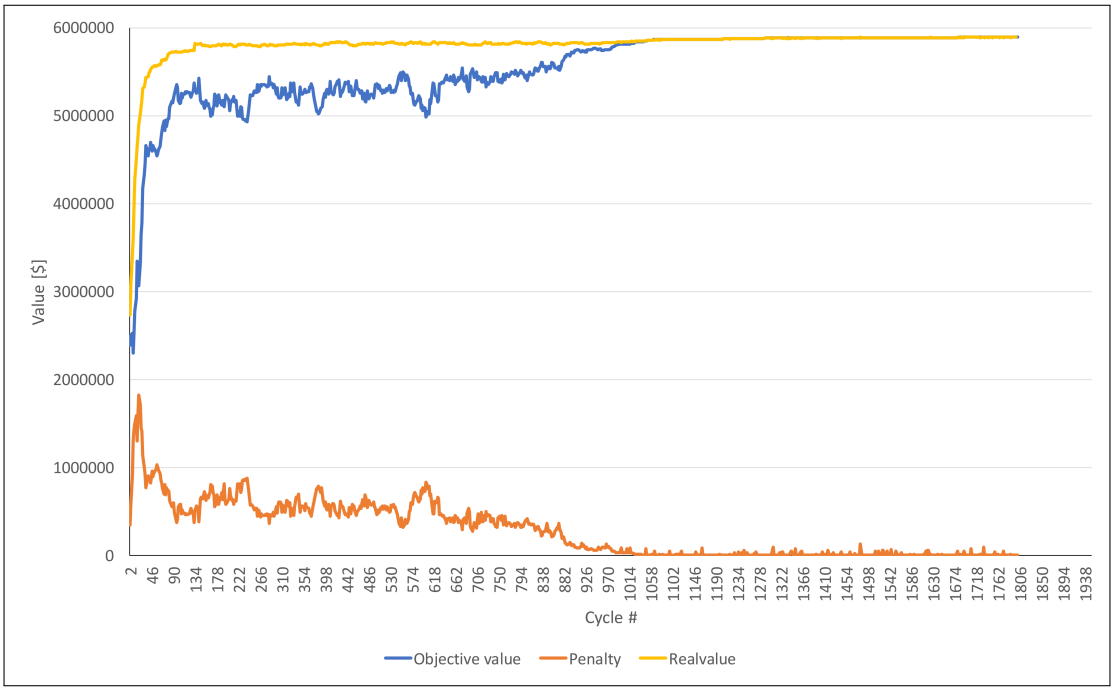


Figure 4.18: Course of run with perturbation to optimal destination.

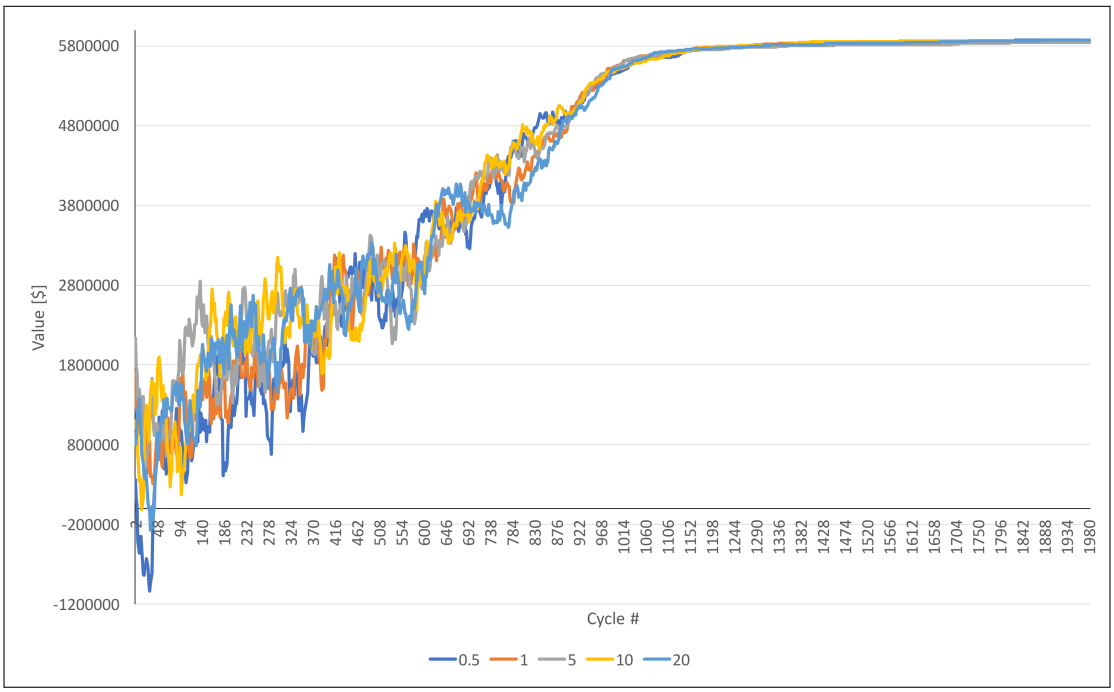


Figure 4.19: Objective value of runs at different probabilities for boundary preference.

To investigate the effect of a preference for boundary frames on the run of the algorithm on cases with a more distinct grade distribution the third method was also run on case 2 with a preference probability of 20%. When compared to the base-case on case 2, there could no significant influence on the convergence of the algorithm be detected, and the objective values of the final solutions were very similar.

4.9 MMU size

To test the applicability of multiple MMU sizes for the mining constraints of the algorithm, multiple options were tested on case 1. The MMU sizes of 2x2, 3x3, 5x3, 5x5 and 6x6 have been tested. The increase in MMU size has a significant effect on the cycle-time of the algorithm. The computing time of one cycle of the algorithm increased exponentially with increasing MMU size. This is mainly caused by the updating of the misfit values for all frames that were affected by the perturbation. With an increasing frame size the increase in the area of effect also increases. A 2x2 perturbation requires to update 9 misfit frames, and a 5x5 perturbation requires to update 81 frames. Opposite to this longer cycle-time, the solution space becomes smaller with larger MMU sizes, therefore the algorithm might require less cycles to converge into the optimum. The objective values of the course of the runs for the different MMU sizes are shown in figure 4.20.

The runs were all performed with the same amount of cycles and trials per cycle, and the same temperature schedule, therefore the eventual convergence into the found optimum takes place in the same range of cycles. It can be noted that the larger the MMU-size, the earlier the solutions reach a higher objective value. This can be explained by the fact that the solution space is smaller and that sending a large group of blocks to a worse destination will impact the objective value in a larger extent than a small group will. This makes it more unlikely to accept worse perturbations and means that the initial temperature of the algorithm is MMU-size dependent and should be adjusted for each MMU-size. The larger the MMU-size, the lower the final objective value becomes, this is expected because it is essentially the same as decreasing the selectivity, more SMU's will be send to a not-optimal destination.

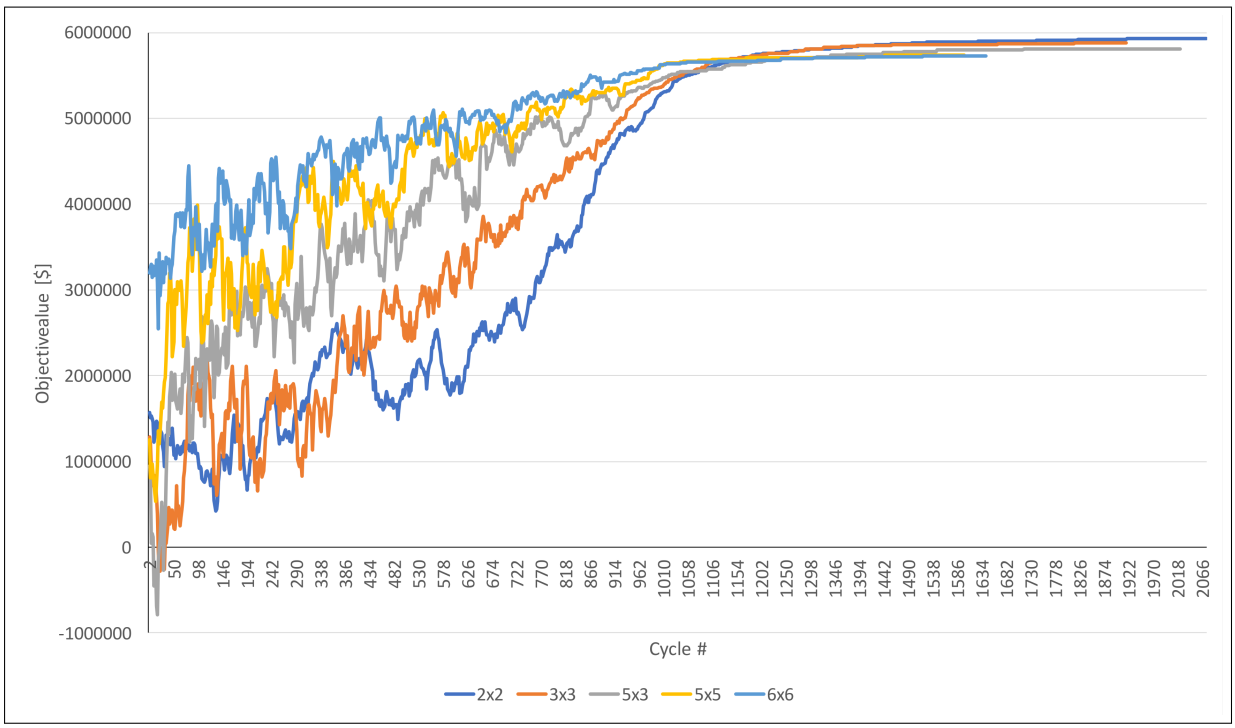


Figure 4.20: Objective value of runs at different cluster sizes.

4.10 Cut-off grade

To investigate the sensitivity of the algorithm on the cut-off grade, the algorithm was run at different processing costs. The objective value of the course of the runs is shown in figure 4.21. As is expected the final objective value becomes lower with increasing processing cost, less SMU's will be sent to the processing plant and the SMU's that are send there create less profit. The runs with a higher processing cost of 5000+$/t converge in a slightly earlier stage. This might be caused by the decrease in boundary blocks, as most of the grid is sent to the waste dump. The penalty factor was not adjusted for the higher processing cost runs, this caused some minor constraint violations in the final solution of the algorithm. The consequences of a changing cut-off grade on the algorithm are small, but the results show that the penalty factor and the amount of cycles do need to be defined for every different case if the differences become significant. This probably wont be necessary within the same mining operation, since the mining cost, processing cost and grade will be similar.

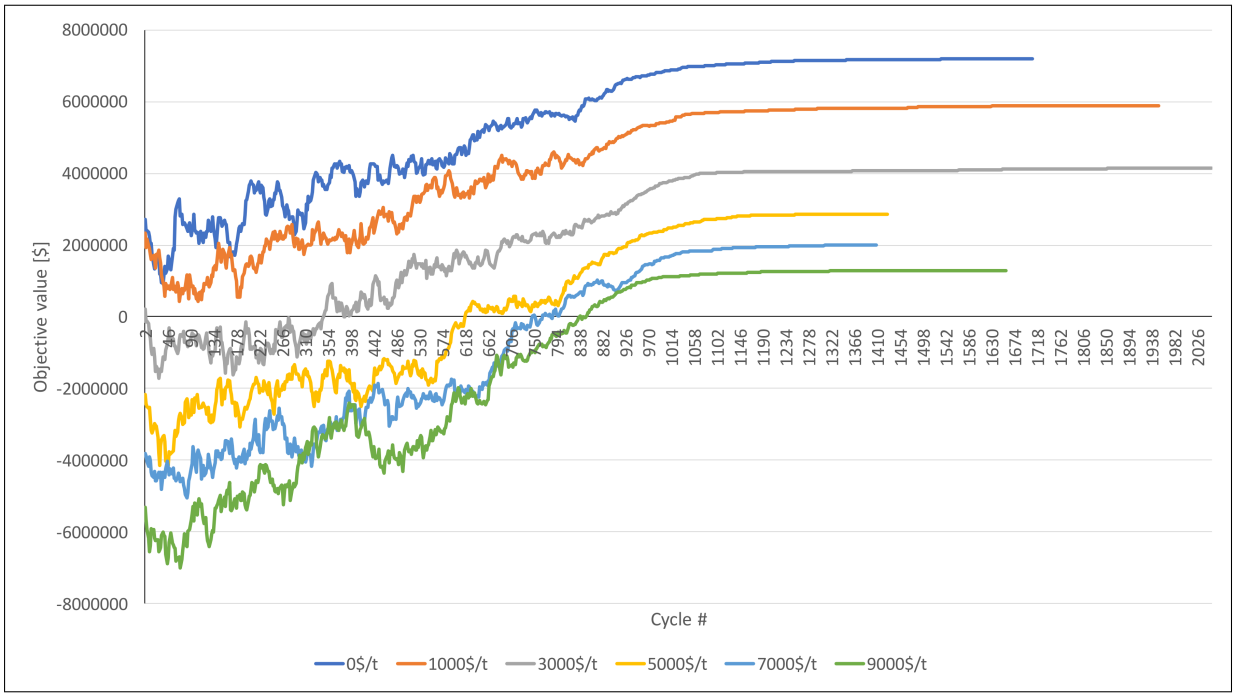


Figure 4.21: Objective value of runs at different processing costs.

5 Conclusions and Recommendations

5.1 Conclusions

Ore deposits are becoming more complex and difficult to extract, as a result mining operations need to increase their efficiency and performance. Dig-limit design is a part of the short-term open-pit mine planning. Current common practice is that these dig-limits are drawn by hand by mining engineers or geologists. This results in subjective and often sub-optimal dig-limit designs. To make this process more efficient, and to optimize the profit of mining operations, in this thesis, a new, automated method for the design and optimization of dig-limits is presented.

The optimization problem of dig-limit designs is a combinatorial optimization problem which is proven to be NP-complete. Exact optimization methods will therefore be unfeasible to solve the problem as computing time will quickly become a problem with increasing grid sizes. For this thesis the heuristic method of simulated annealing was used to create optimized dig-limit designs, this method results in near-optimal solutions in a reasonable computing time.

The program that was created for this thesis consists of a framework for simulated annealing in which many variations can be implemented and were tested. The program works properly and creates automated, optimized dig-limit designs that comply to the spatial mining constraints. It manages this in a very reasonable computing time, making it applicable for the day-to-day mining operations and an interesting alternative for hand-drawn dig-limit designs.

Different options to built the algorithm have been tested which resulted in multiple interesting options for the application in mining operations.
When a high initial temperature is used and the algorithm is run in its normal setup, doing an extensive initial search of the solution space, the initial solution that is input into the algorithm does not have influence on the result, and a pre-optimized solution will be wasted. Therefore a random initial solution should be used when running the full algorithm.

The most effective cooling schedule that was tested was the quadratic cooling schedule, this ensured a good convergence into the local optimum at an early stage of the algorithm, resulting in the best objective value.

To reduce the switching between destination boundaries, an alternative penalty function was used. This performed very well and the discouragement of destination switching can be properly regulated, while still complying to the spatial mining constraints. This makes the program an interesting option especially for mining operations with a heterogeneous ore-body where dilution is a problem.

An alternative, greedy variation of the algorithm performed very well in combination with the pre-

optimized initial solution. It resulted in high quality solutions with a computing time of more than three times shorter than the standard algorithm. This method was also applicable in combination with the destination boundary discouragement.

The required penalty factor, misfit factor, initial temperature and the stop criteria are dependent on the grade values of the SMU's as well as the mining parameters. These are case specific and need to be determined before the algorithm can be used. However, the algorithm is found to be robust to small variations and within a mining operation they will probably only have to be determined once.

The program is flexible and can be adjusted for multiple material destinations and multiple ore-types, and the objective function can be adjusted for different optimization goals. These options haven't been tested in this thesis and will affect the computation time of the algorithm. Different shapes and sizes for the spatial mining constraints are also possible in the program and have been tested successfully. This makes the program potentially applicable for all types of open-pit mining operations.

5.2 Recommendations

- The program should be tested on a real-life case, where it is compared to the performance of hand-drawn dig-limit designs of engineers.
- The program might be written more efficient, the updating of the misfits consumes a lot of computing time, especially with larger cluster sizes when it needs to update more frames after each perturbation.
- The boundary cases should be changed, the grid might not be dividable in m x n frames, this should be possible without creating an initial penalty violation. Also null nodes might be used to handle grids that are not rectangular in shape.
- A new cooling schedule with a faster initial decrease in temperature might be tested, as the quadratic cooling schedule performed best, and the initial search of the algorithm does not seem to have as much influence on the final solution as the final convergence.
- The combination of a pre-optimized initial solution and a greedy optimization algorithm might be further investigated, as it shows real promise to be the most efficient dig-limit design optimization tool.
- More complicated perturbation combinations might be investigated, for instance a variation of the probability for boundary frame preference.
- To make the program use-able for the real-life application in mining engineering, an interface should be created such that the operator does not require python knowledge to use the program. This interface should include options to variate the parameters, and the objective function.
- The perturbation mechanism g01, where the selected frame is sent to its optimal destination should be further investigated for its performance and the ability to remove constraint violations.
- The stopping criteria could be made improvement dependent, such that the algorithm terminates after no more significant improvement is reached. This still does not take away the requirement to test for an appropriate number of temperature cycles and trails per cycle.
- If possible, the method should be compared to an exact optimization method, to indicate how well the algorithm approaches the global optimum.

Bibliography

Aarts, E., Korst, J. and Michiels, W. (2005), Simulated annealing, *in* 'Search methodologies', Springer, pp. 187–210.

Allard, D., Armstrong, M. and Kleingeld, W. (1994), 'The need for a connectivity index in mining geostatistics', *Geostatistics for the Next Century* **6**, 293–302.

Bertsimas, D., Tsitsiklis, J. et al. (1993), 'Simulated annealing', *Statistical science* **8**(1), 10–15.

Černỳ, V. (1985), 'Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm', *Journal of optimization theory and applications* **45**(1), 41–51.

Collins, N., Eglese, R. and Golden, B. (1988), 'Simulated annealing–an annotated bibliography', *American Journal of Mathematical and Management Sciences* **8**(3-4), 209–307.

Dagasan, Y., Renard, P., Straubhaar, J., Erten, O. and Topal, E. (2018), 'Pilot point optimization of mining boundaries for lateritic metal deposits: Finding the trade-off between dilution and ore loss', *Natural Resources Research* pp. 1–19.

Dagdelen, K. (2001), 'Open pit optimization-strategies for improving economics of mining projects through mine planning', *17th International Mining Congress and Exhibition of Turkey* pp. 117–121.

Eglese, R. (1990), 'Simulated annealing: a tool for operational research', *European journal of operational research* **46**(3), 271–281.

Glover, F. and Greenberg, H. J. (1989), 'New approaches for heuristic search: A bilateral linkage with artificial intelligence', *European Journal of Operational Research* **39**(2), 119–130.

Golden, B. L. and Skiscim, C. C. (1986), 'Using simulated annealing to solve routing and location problems', *Naval Research Logistics Quarterly* **33**(2), 261–279.

Huang, M. (1986), 'An efficient general cooling schedule for simulated annealing', *Proc. ICCAD, Santa Clara, USA* pp. 381–384.

Hustrulid, W. A., Kuchta, M. and Martin, R. K. (2013), *Open pit mine planning and design, two volume set & CD-ROM pack*, CRC Press.

Isaaks, E. (1990), 'The application of monte carlo methods to the analysis of spatially correlated data', *PhD Dissertation* p. 213.

Isaaks, E., Treloar, I. and Elenbaas, T. (2014), 'Optimum dig lines for open pit grade control', *9th International mining geology conference, Adelaide, South Australia* .

Jeffcoat, D. E. and Bulfin, R. L. (1993), 'Simulated annealing for resource-constrained scheduling', *European Journal of Operational Research* **70**(1), 43–51.

Johnson, D. S., Aragon, C. R., McGeoch, L. A. and Schevon, C. (1989), 'Optimization by simulated annealing: An experimental evaluation; part i, graph partitioning', *Operations research* **37**(6), 865–892.

Kalivas, J. H. (1992), 'Optimization using variations of simulated annealing', *Chemometrics and intelligent laboratory systems* **15**(1), 1–12.

Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P. (1983), 'Optimization by simulated annealing', *science* **220**(4598), 671–680.

Laarhoven van, P. J. and Aarts, E. H. (1987), *Simulated annealing: Theory and applications*, Springer.

Ledesma, S., Aviña, G. and Sanchez, R. (2008), Practical considerations for simulated annealing implementation, *in* 'Simulated annealing', pp. 402–420.

Luke, B. (2007), '*Simulated Annealing Cooling Schedules*'. `http://www.btluke.com/simanf1.html` [Accessed: 2018-10-20].

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. and Teller, E. (1953), 'Equation of state calculations by fast computing machines', *The journal of chemical physics* **21**(6), 1087–1092.

Neufeld, C. T., Norrena, K. P. and Deutsch, C. V. (2003), Semi-automatic dig limit generation, Technical report, Department of Civil and Environmental Engineering, University of Alberta.

Norrena, K. and Deutsch, C. (2000), 'Automatic determination of dig limits subject to geostatistical, economical and equipment constraints', *Center for Computational Geostatistics (CCG), University of Alberta, Edmonton, Alberta, Canada* .

Pearl, J. (1984), *Heuristics: intelligent search strategies for computer problem solving*, Addison-Wesley Publishing Company.

Richmond, A. and Beasley, J. (2004), 'An iterative construction heuristic for the ore selection problem', *Journal of Heuristics* **10**(2), 153–167.

Ruiseco, J. R. and Kumral, M. (2017), 'A practical approach to mine equipment sizing in relation to dig-limit optimization in complex orebodies: multi-rock type, multi-process, and multi-metal case', *Natural Resources Research* **26**(1), 23–35.

Ruiseco, J. R., Williams, J. and Kumral, M. (2016), 'Optimizing ore–waste dig-limits as part of operational mine planning through genetic algorithms', *Natural Resources Research* **25**(4), 473–485.

Rutenbar, R. A. (1989), 'Simulated annealing algorithms: An overview', *IEEE Circuits and*

Devices Magazine **5**(1), 19–26.

Sari, Y. A. and Kumral, M. (2018), 'Dig-limits optimization through mixed-integer linear programming in open-pit mines', *Journal of the Operational Research Society* **69**(2), 171–182.

Sari, Y., Germain, N. and Kumral, M. (2017), Short-term production planning for multi-metal open-pit mines with equipment size constraints, Technical report.

Sierksma, G. and Ghosh, D. (2009), *Networks in action: text and computer exercises in network optimization*, Vol. 140, Springer Science & Business Media.

Sinclair, A. and Blackwell, G. (2002), *Applied Mineral Inventory Estimation*, Cambridge University Press.

Srivastava, R. (1987), 'Minimum variance or maximum profitability?', *CIM Bulletin* **80**(901), 63–68.

Szu, H. and Hartley, R. (1987), 'Fast simulated annealing', *Physics letters A* **122**(3-4), 157–162.

Tabesh, M. and Askari-Nasab, H. (2013), 'Automatic creation of mining polygons using hierarchical clustering techniques', *Journal of Mining Science* **49**(3), 426–440.

Van Laarhoven, P. J. and Aarts, E. H. (1987), *Simulated annealing: Theory and applications*, Springer.

Verly, G. (2005), 'Grade control classification of ore and waste: A critical review of estimation and simulation based procedures', *Mathematical Geology* **37**, 451–475.

Wilde, B. and Deutsch, C. V. (2015), A short note comparing feasibility grade control with dig limit grade control, Technical report.

Youssef, H., Sait, S. M. and Adiche, H. (2001), 'Evolutionary algorithms, simulated annealing and tabu search: a comparative study', *Engineering Applications of Artificial Intelligence* **14**(2), 167–181.