Optimizing Paths for Autonomous Vehicles Using Evolutionary Algorithms

TOMMY LINDGREN

MASTER OF SCIENCE PROGRAMME
Computer Science

Luleå University of Technology
Department of Computer Science and Electrical Engineering
Division of Computer Science and Networking
Swedish mining company LKAB uses autonomous vehicles to transport ore at their production areas. The traversal paths of the vehicles are today manually constructed in a CAD program by an engineer. A previously developed physical model of the vehicles suggests that the paths can be improved to allow better traversal times by optimizing their smoothness, i.e. the integral of the derivative curvature squared over the arc-length. This thesis presents an evolutionary algorithm that represents the path as a parametric B-spline and optimizes its smoothness using an evolutionary strategy. The algorithm is evaluated on a few test cases and the results indicate that time savings up to 10–20% are possible for longer paths.
This Master's Thesis was written during the winter 05/06 as a final examination of the MSc. programme in Computer Science and Engineering at Luleå University of Technology. The author wish to thank a few people that have been important in the development of this thesis: my supervisor Håkan Jonsson for his support and guidance; Inge Söderqvist for his helpful comments; Björn Zachrisson for choosing a related and crucial topic as his thesis; and Rickard Nilsson for letting us use his software and giving us a hand in integrating it with our code. I’m also grateful to Mats Strömsten at LKAB for making this thesis possible in the first place.

Tommy Lindgren
CONTENTS

CHAPTER 1: INTRODUCTION
   1.1 Background .............................................. 1
   1.2 Purpose ................................................. 2
   1.3 Delimitations ............................................ 2

CHAPTER 2: THEORY
   2.1 Problem description ................................... 5
   2.2 The B-Spline ............................................ 6
   2.3 Previous approach ...................................... 6
   2.4 A brief overview of evolutionary algorithms ............. 7
   2.5 Evolutionary algorithms and splines ..................... 9

CHAPTER 3: AN EVOLUTIONARY STRATEGY
   3.1 Overview of the algorithm ............................. 11
   3.2 Drive path representation ............................. 12
   3.3 The initial population .................................. 13
   3.4 The objective function ................................ 13
   3.5 The penalty function ................................... 14
   3.6 Selection ................................................. 18
   3.7 Recombination ........................................... 18
   3.8 Mutation ................................................ 19
   3.9 HDF generation ......................................... 22

CHAPTER 4: METHOD
   4.1 Implementation details ................................. 25
   4.2 Safety margin ........................................... 25
   4.3 Test cases ............................................... 25
   4.4 Comparison to the MATLAB implementation .......... 27

CHAPTER 5: EVALUATION
   5.1 The safety margins .................................... 29
   5.2 The ES algorithm ....................................... 29
   5.3 The MATLAB implementation .......................... 35

CHAPTER 6: DISCUSSION


This chapter gives a background to the problem treated in this report and explains the thesis’ purpose.

1.1 Background

The Swedish mining company LKAB (LuossavaaraKiirunavaara AktieBolag) uses autonomous vehicles to transport the ore inside the mines. Today the vehicles follow drive paths designed by an engineer. LKAB has in recent years financed research at the Department of Computer Science and Electrical Engineering at Luleå University of Technology to investigate ways of improving the quality of these manually made drive paths, specifically decreasing the traversal time. Tomas Berglund has, as a part of his licentiate thesis [1], written an experimental path optimizer in MATLAB. The optimizer showed potential but had a few limitations and was difficult to use. Mats Staffanson has written a Master’s Thesis [2] in which he addressed some problems of practical nature, such as reading map files and performing evaluation of drive paths, but also developed the physical model of the vehicles.

During the spring of ’05 the results of Berglund’s and Staffanson’s works were combined in a project course [3]. A project group, in which the author was included, wrote a program (uninspiringly named “Prototype X”) that made it easier to use Berglund’s optimizer, and the optimized paths could be evaluated using Staffanson’s methods. Some of the limitations of the optimizer were still present in the program. The most significant are the following:

- The mine walls must be monotone, in the sense that for each $x$ coordinate there must be exactly one corresponding $y$ coordinate. This is not always the case, even if the mine is rotated to an arbitrary angle. This means that it is not possible to optimize paths in some parts of the mine, which is of course not acceptable.

- Before an optimization is started, the user has to help the optimizer by performing a knot
placement (more about this later in the thesis). Choosing a good placement is not only
time-consuming but also requires some experience of the user. The placement affects the
outcome of the optimization and a bad placement may cause the optimization to fail. The
user should not have to deal with such tiresome tasks. The knot placement is actually a
difficult nonlinear optimization problem in itself [4].

- The performance of the MATLAB code is unsatisfactory — computation times exceeding
several hours are not uncommon when optimizing longer paths.

Correcting the above mentioned drawbacks in the existing implementation may be difficult,
at least without sacrificing more computation time. For example, giving up the monotonicity
requirement doubles the number of parameters to be optimized and it may also require a new
way of ensuring that the path stays inside the allowed area. Since it is appears difficult to
succeed in achieving the goals by improving the current MATLAB implementation, it would be
interesting to explore a different approach to path optimization. One such possible direction is
evolutionary algorithms, a class of search algorithms inspired by the biological evolution that
has proven successful in many hard optimization problems.

1.2 Purpose

The purpose of this Master’s Thesis is to investigate which evolutionary algorithms that are
suitable for this type of problem and then implement and evaluate one such algorithm. The
evaluation will examine the reliability of the algorithm and its execution time, but of course
also show how much time that can be gained by optimizing paths. The results will be compared
to those of Berglund’s implementation.

1.3 Delimitations

The subject touches many areas and evolutionary algorithms are a large topic so delimitations
are necessary. Firstly, this thesis will use the model developed by Berglund and Staffanson
as it is. The model is a simplification and refinements are imaginable but not addressed here.
Secondly, evolutionary algorithms are very flexible and come in many variants. A complete
survey of all variants is not doable in the given time frame. Moreover, evolutionary algorithms
are generally hard to analyze. As Renner and Ekárt [5] write about genetic algorithms (GA),
one type of evolutionary algorithms:

“‘To predict the GAs’ behavior, it would be useful to have a mathematical characterization
of how they work. Unfortunately, it is very difficult—if not impossible—to predict
how a stochastic search method like GAs will perform on a specific problem in a
complex, highly nonlinear domain. [. . .] When applying GAs to more complex
design problems, a complete theoretical analysis is not possible, and in such cases
ingenious GA design ideas in the choice of representation and operators are needed to ensure evolution toward good solutions.”

Thus we will resort to empirical studies of the properties of the algorithms in this thesis. In addition to the defects of the optimizer listed above, there is another critical problem, not directly related to the optimizer. To make sure that the vehicles do not collide with objects inside the mine, the optimizer applies a safety margin to the mine map. The method that has been used to generate the margins is very simplistic; it does not even give guarantees on the minimum distance between the path and the mine walls. This problem will not be treated in this thesis, but in another Master’s Thesis, performed parallely to this, by Björn Zachrisson [6].
In this chapter, a formal description of the problem is given, and the previous approach is described. We also dive into the topic of evolutionary computing and review related works.

### 2.1 Problem description

LKAB uses the electrical wheel loader Tamrock Toro 2500 as LHD (Load-Haul-Dump) vehicles. These are possible to control autonomously using a system called HUNS (High-speed Underground Navigation System). To construct drives paths for the vehicles an engineer loads a mine map stored in the DXF (Drawing Interchange Format) file format into LDS (a CAD system).

A mine map has two important types of areas: the drifts where the ore is loaded and the shafts (see figure A.1). Since the vehicles are powered by electricity, they feed a cable as they drive. To transport the ore between the areas without running over the cable, the vehicles must drive forwards to the loading area, return to the source of the cable by driving backwards, and continue to the shaft in a similar manner.

When the engineer is done designing the paths, they are exported to the HDF (HUNS Definition Format) file format and then imported into the control system. But before this, we can do operations on them. The paths are nothing but a chain of segments and it is possible to replace some of these if we ensure that the new segments have the correct directions and curvature\(^1\) properties at the joints. This gives us an opportunity to improve the paths. The paths the engineer designs have somewhat sharp turns. As we will see later, smoother turns mean that the vehicle can travel faster. We can express the problem a bit more formally as: given the following parameters

- a simple polygon \( P \) with holes (representing the mine with a safety margin applied),

---

\(^1\)Curvature is the amount by which a geometric object deviates from being flat, e.g. if a curve at one point has a curvature of 0, then the curve is straight at that point.
• two end points \( p_{\text{start}} \) and \( p_{\text{end}} \),
• two angles \( \theta_{\text{start}} \) and \( \theta_{\text{end}} \) (the direction of the drive path at the end points),
• two curvatures \( \kappa_{\text{start}} \) and \( \kappa_{\text{end}} \),
• two curvature derivatives \( \kappa'_{\text{start}} \) and \( \kappa'_{\text{end}} \).

compute a drive path, expressed in the HDF format, from \( p_{\text{start}} \) to \( p_{\text{end}} \), residing inside \( P \) and which ends satisfy the constraints specified by \( \theta_{\text{start}}, \kappa_{\text{start}}, \kappa'_{\text{start}} \) and \( \theta_{\text{end}}, \kappa_{\text{end}}, \kappa'_{\text{end}} \), that minimizes the time needed for a Tamrock Toro 2500 machine to complete the path.

2.2 The B-Spline

Since B-splines are a reoccurring topic in this thesis, a short introduction is given here. B-splines, short for basis splines, are a generalization of Bézier curves, a parametric curve commonly used in computer graphics. An \( n \)-dimensional B-spline curve constitutes of \( L \) piecewise polynomials of degree \( k \), and a nondecreasing knot sequence

\[
\{u_0, \ldots, u_{L+2k-2}\}
\]

and a coefficient matrix \( D \) of size \( n \times L \). When \( n = 2 \) the coefficients are said to form a control polygon, whose vertices are

\[
(d_{0,i}, d_{1,i}) \quad i = 0, \ldots, L + k - 1.
\]

The vertices of the polygon are also called control points or de Boor points. The curve is only defined over the domain

\[
[u_{k-1}, \ldots, u_{L+k-1}]
\]

and the knots \( u_i \) are usually in the interval \([0, 1]\), which represents the arc-length of the curve. 1-dimensional B-splines are called univariate. 2-dimensional, parametric B-splines consists effectively of two univariate B-splines.

The B-spline has a number of nice properties. The most important one is that the curve is differentiable \( k - 1 \) times at the knots \( u_i \) and infinitely often differentiable at all other points. This makes the curve look smooth. Why this is desirable is explained in the next section.

2.3 Previous approach

Berglund’s prototype code is built around MATLAB’s \texttt{fmincon}, an optimization solver that can handle equality and inequality constraints. The equality constraints constitute of the curvature constraints at the path ends and the inequality constraints make sure the path is kept inside
the mine. The inequality constraints, in this implementation, require that the mine walls are monotone.

Since it is slightly complicated to compute the traversal time, Berglund optimizes the smoothness of the path instead. Smoothness, defined in section 3.4, is closely related to the traversal time since the highest allowed gear at a certain point depends on the curvature derivative. The drive path itself is (during the optimization) represented by a univariate B-spline of degree 4. B-splines have good smoothness properties and the high degree guarantees that the curvature derivative will be continuous. A continuous curvature derivative is assumed to reduce the stress of the steering mechanism of the vehicle.

To express the path as a B-spline, we must find the B-spline’s knot sequence and coefficients. Berglund lets the user place the knots and then feeds the smoothness function, the constraints and the user specified knot sequence to \texttt{fmincon}, which finds the optimal B-spline coefficients.

2.4 A brief overview of evolutionary algorithms

“[. . .] one general law leading to the advancement of all organic beings—namely, multiply, vary, let the strongest live and the weakest die.”

Charles Darwin, \textit{The Origin of Species}, 1859

The introductory prestudy of this Master’s Thesis suggested that evolutionary algorithms (EA) have great potential. EA are a type of search algorithms that use mechanisms inspired by the biological evolution, such as reproduction, mutation, recombination, natural selection and survival of the fittest. The first use of EA was probably in the 60’s and several types of EA have since then seen the light of the day, of which \textit{genetic algorithms} are the most famous variant. The last decades EA have proven to be successful in a wide range of problems [7]. EA’s popularity is explained by that they usually are simple but still can solve hard optimization problems that traditional methods have troubles with.

One key feature of EA are that they maintain a set of possible solutions during the optimization. The set is called \textit{population} and the solutions \textit{individuals} (or \textit{chromosomes}). Each individual has a \textit{genome} that represents the solution. The genome can be \textit{binary} or \textit{continuous}, i.e. the solution can be encoded as ones and zeros or constitute directly of real values. Before an optimization is performed, an \textit{initial population} is created. The individuals in this population are usually uniformly distributed over the search space (the set of all possible feasible and infeasible solutions). Each individual is then evaluated subject to a certain \textit{fitness function}. The optimization is terminated if any of the individuals has a sufficiently good fitness and satisfies all given constraints. If no feasible individual is found, then a new population (or \textit{generation}) is created by applying one or many \textit{genetic operators} to the old population and the

\footnote{In mathematical literature, “smoothness” normally refers to the continuity of function derivatives. In this thesis, smoothness is, loosely expressed, a measurement of how much a path turns, i.e. how much the vehicle has to change the position of its steering wheel as it travels along the path. A smoothness of zero means that the path is either straight or a perfect circle (segment).}
Figure 2.1: Schematic flowchart of an evolutionary algorithm.

evaluation step is repeated (see figure 2.1). The population will then, assuming the algorithm is well-construed, converge to an optimum which hopefully is a global one.

Some EA utilize elitism which means that the best individuals of the last generation are kept untouched in the new generation. This guarantees that good solutions are not lost and that may speed up the optimization. But why is not elitism always used? It may seem unwise to throw away the best solutions found so far. But if the objective function has many local optima it is generally a good idea to not use elitism, since elitism makes it harder to escape local optima.

Constraints in EA are usually handled by giving infeasible solutions worse fitness. Each constraint is transformed to an objective function or penalty function, that estimates how much an individual violates the constraint, and then the fitness is defined as a weighted-sum of the objective functions. Constraint-handling is explained more in detail in section 3.5.

Several variants of EA have been developed over the years. What differs them are mainly how the genetic operators are implemented. The three most common operators are selection, mutation and reproduction (or recombination). All EA use some form of mutation, which means that random changes are inserted into the individuals genomes. Mutation is important because it introduces variance in the population. Reproduction means that (parts of) a genome of an old individual is entered into individuals of the next generation. This makes it possible for promising solutions to survive and evolve over successive generations. Recombination is a variant of reproduction where new individuals are created by using genome from two or more old individuals, which means that (good) traits can be inherited from different parents. To choose the parent individuals is called selection and can be done different ways, but always by somehow favoring individuals with good fitness.

The prestudy suggested to examine two EA types, Differential Evolution and Evolutionary Strategies, which have in common that both operate on continuous genomes. Differential Evolution (DE) is a relatively new algorithm developed by Price and Storn that has proven useful when the fitness function is continuous [8]. Creating new individuals in DE is done by randomly choosing two individuals in the population, calculating the vector difference between their genomes and adding it to the genome of a third individual. This kind of mutation gives DE a good performance. Sadly, DE was not very successful on this problem. The population converged too quickly and the optimization ended prematurely. The exact cause for this was
Evolutionary Strategies (ES) were probably the first type of EA used in practice and were developed by Rechenberg and Schwefel to replace a traditional optimization algorithm for solving design problems [9, 10]. The original algorithm used a population of only two individuals and elitism was enforced. The genomes were mutated by adding normal distributed perturbations to the parameters. ES have since its introduction evolved a lot and have today many similarities with genetic algorithms [11], the EA that resembles biological evolution the most. The use of recombination and self-adaptive perturbations are the most noticeably improvements. A very simple ES algorithm was implemented during the prestudy, and the result was sufficiently promising to develop a more advanced version. This version is described in detail in chapter 3.

2.5 Evolutionary algorithms and splines

Using EA to optimize splines (or Bézier curves) in some sense has been done before. Márkus et al. show how spline operations (such as data fitting) can be replaced with GA-based methods [12, 13]. Goldenthal and Bercovier explore the usage of GA for interpolation and approximation subject to approximation error, length, curvature and elastic energy [14]. Sarfraz and Raza [15] and Rueckert et al. [16] provide two examples of contour fitting using GA and Markov-Random-Fields-based techniques, respectively. Obayashi et al. use multi-objective GA and B-splines for super-sonic wing-shape optimization [17, 18]. Yoshimoto et al. find optimal knot placements using GA with binary genomes (no control points are optimized though).

The algorithms presented in these works are almost all GA (or similar EA) with continuous genomes consisting of knot sequences and/or control points. Most of the problems they solve are variations of data fitting without any additional constraints, though. This is a bit different from our problem, where the spline can move freely inside a narrow area, except at the constrained ends. However, constraint enforcement is generally orthogonal to the choice of EA algorithm, so it was decided to go for the GA-like ES algorithm anyway.
A detailed description of the presented algorithm is given in this chapter.

3.1 Overview of the algorithm

The algorithm presented in this thesis is a so called multi-membered \((\mu, \lambda)\)-ES, which means that the algorithm maintains, using an Evolutionary Strategy, a population of \(\lambda\) individuals that are created from \(\mu\) parents. Expressed in pseudo code, the outline of the algorithm looks like this:

\[
\begin{align*}
t &= 0; \\
\text{initialize}(P(t)); \\
\text{repeat} \\
&\hspace{1em} \text{evaluate}(P(t)); \\
&\hspace{1em} \text{if terminate}(P(t)) \\
&\hspace{2em} \text{break}; \\
&\hspace{1em} t = t + 1; \\
&\hspace{1em} P(t) = \text{select}(P(t-1)); \\
&\hspace{1em} \text{recombine}(P(t)); \\
&\hspace{1em} \text{mutate}(P(t)); \\
\text{end}
\end{align*}
\]

where \(P(t)\) denotes the current population at iteration step \(t\). The initialization of the population is explained in section 3.3. In the evaluation step, the fitness of the individuals is determined. The fitness reflects how well an individual solves a given problem. Generally, when solving constrained nonlinear optimization problems we want to find a solution vector \(\tilde{s}\) which minimizes an objective function \(f(\tilde{s})\) subject to
where $g_i(\vec{s})$ are inequality constraints and $h_j(\vec{s})$ equality constraints. But an EA doesn’t
directly know about constraints; the EA just optimizes the average fitness of the population.
Therefore we must define a fitness function that determines how well a certain individual satisfy
the given constraints. The fitness function has the general form

$$\text{fitness}(\vec{s}) = f(\vec{s}) + p(\vec{s})$$

(3.3)

where $f(\vec{s})$ is the objective function above and $p(\vec{s})$ is a penalty function, giving individuals
that do not satisfy the constraints a worse fitness. The choices of $f(\vec{s})$ and $p(\vec{s})$ are discussed
in section 3.4 and 3.5, respectively, and $\vec{s}$, i.e. the representation of the solution, is defined in
section 3.2. Note that we want to $\text{maximize}$ the fitness, but that the fitness is defined as a cost
function, which we want to $\text{minimize}$.

The fitness is used for choosing parent candidates in the selection step, but this information
is also normally used for deciding when the optimization should terminate. A simple variant
is to terminate when the fitness reaches a certain level. This is not usable in our case, as the
fitness level where the solutions get acceptable can differ vastly between different drive paths,
and we lack a method to get good estimates. A slightly more sophisticated way is to look at
how the fitness changes for each iteration. When changes get sufficiently small it is futile to
continue with the optimization. However, this is not reliable either, since the fitness does not
always improve every iteration and it can even occasionally get worse. The most promising
way seems to be to compare the best fitness of the population with the average fitness, and
terminate when difference get sufficiently small. But, again, also here it is difficult to tell what
is “sufficiently”. So in this thesis we resort to terminating the optimization after a fixed number
of iterations.

The reproduction steps, parent selection, genome recombination and mutation, are discussed
in section 3.6, 3.7 and 3.8.

### 3.2 Drive path representation

As mentioned in section 2.3, Berglund represents the drive path as a univariate B-spline of
degree 4 because of its smoothness properties. But a univariate spline is monotone, i.e. for
each $x$ coordinate there is exactly one corresponding $y$ coordinate, and not every drive path
satisfy this. To represent all possible drive paths we must use parametric splines. The 2-
dimensional, parametric B-spline is defined by a knot sequence $\vec{u}$ and a coefficient matrix $D$
of height 2. Thus our solution vector (or genome) is represented as

$$\vec{s} = [u_0, u_1, \ldots, u_{L+2k-2}, d_{0,0}, d_{0,1}, \ldots, d_{0,L+k-1}, d_{1,0}, d_{1,1}, \ldots, d_{1,L+k-1}]^T$$
where $u_i$ are the knots, $d_{0,i}$ and $d_{1,i}$ the $x$ and $y$ coordinates for the control points, $L$ the number of spline segments and $k$ the degree of the spline, in accordance with the terminology and definitions in section 2.2.

### 3.3 The initial population

The first generation in EA is normally populated by individuals uniformly distributed over the search space. This is not practicable in our case, as the feasible space, the set of feasible solutions, is very small compared to the search space. If the individuals in the population are too far from being feasible, the algorithm may fail to make the population converge. To create a population of reasonably good individuals, splines are fitted through interpolation points spread along the shortest path between $p_{\text{start}}$ and $p_{\text{end}}$. The point distribution is not completely uniform. Corners where the shortest path tangents the mine polygon (the polygon that represents the mine walls with safety margin applied) are favored and there is minimum allowed distance of 5 m between the points. The average point distance is about 8 m. As a final touch, to increase the variance, all points are shifted randomly, at maximum half the average point distance, along the shortest path. These generated splines are far from perfect solutions but quite close to be feasible.

### 3.4 The objective function

As stated earlier, Berglund chooses to optimize the smoothness of the path rather than the traversal time, which would be the obvious objective function. There are mainly two reasons for this. The first one is that the smoothness is easier and faster to compute. Both the MATLAB algorithm and the ES algorithm need to evaluate the objective function a large number of times, and this contributes greatly to the total execution time. The other reason is that the traversal time function is not continuous, and conventional optimization algorithms, that compute gradients, may have a problem with this.

But as we shall see, the smoothness is related to the traversal time. The vehicle has four gears and each gear has a top speed. As Staffanson explains, the top speed corresponds to a maximum allowed curvature derivative. The two measurements, the speed $v$ and curvature derivative $\frac{d\kappa}{ds}$, relate according to

$$
 v = \frac{C_1}{2C_2 \cos^2(\arctan(C_2\kappa)) \frac{d\kappa}{ds}},
$$

where $C_1$ and $C_2$ are constants that depend on the properties of the vehicle. The velocity and the curvature derivative are clearly in inverse proportion and minimizing the curvature derivative would thus allow higher velocities. Smoothness, here defined as

$$
 \int (\frac{d\kappa}{ds}(s))^2 ds,
$$

(3.5)
i.e. the integral over the arc-length of the curvature derivative squared, now appears as a sensible choice of objective function instead of traversal time. Besides being continuous, Berglund has empirically shown that the function is convex (in his case) and therefore a unique solution exists. This property is probably lost in our problem, though, because of the nonlinearity introduced by the optimization of the knot sequence.

The objective function needs to satisfy one more condition. Ideally, the terms in the right-hand side of equation (3.3) should be of about equal size. If one of them is relatively much larger, the algorithm may fail to minimize the other. The solution is to transform the terms to a predefined interval. Let $F(\vec{s}_i)$ be the smoothness value for a certain individual $\vec{s}_i$. The actual objective function is then

$$f(\vec{s}_i) = \left( \frac{F(\vec{s}_i) - \min \{F(\vec{s}_j)\}}{\max \{F(\vec{s}_j)\} - \min \{F(\vec{s}_j)\} + \epsilon} \right)^2,$$

where $\vec{s}_j$ are all individuals in the population and $\epsilon$ is a very small value so we avoid the risk of dividing by zero. This function has the range $[0, 1]$, i.e. the best individual gets an objective value of 0 and the worst 1. The value is also squared to further favor good solutions. The penalty function will undergo a similar transformation.

A note on the implementation. The smoothness expression (3.5) is estimated by evaluating points along the spline (about 2–3 points/meter), computing the curvature for each point using

$$\kappa = \frac{\dddot{x}y - \dddot{y}x}{(\dddot{x}^2 + \dddot{y}^2)^{3/2}}.$$  \hspace{1cm} (3.7)$$

and then sum the curvature derivatives. All derivatives are numerically determined with forward differences.

### 3.5 The penalty function

There is a wide range of ways to deal with constraints [19, 20]. The most commonly used and explored idea is to treat the deviation from a constraint goal as a penalty cost. Actually, our previously defined inequality constraints $g_i(\vec{s})$ could be used directly as objective functions, which we then sum and add to the fitness value. But as with the objective function we want to do some transformations first. Before that we note that the equality constraints $h_j(\vec{s})$ can be rewritten as inequality constraints on the form

$$|h_j(\vec{s})| - \epsilon \leq 0,$$

where $\epsilon$ is a small threshold value. Since the constraint functions can have different ranges we want to transform them so they give penalties on a predictable interval. We achieve this by using the same transformation applied on the objective function. Given a certain individual $\vec{s}_i$ and a constraint function $g(\vec{s}_i)$ we define the transformed constraint $\hat{g}(\vec{s})$ as
3.5. The Penalty Function

\[
\hat{g}(\vec{s}) = \left( \frac{g(\vec{s}_i) - \min \{g(\vec{s}_j)\}}{\max \{g(\vec{s}_j)\} - \min \{g(\vec{s}_j)\} + \epsilon} \right)^2, \quad (3.9)
\]

where \(\vec{s}_j\) and \(\epsilon\) are as in equation (3.6). We can now define the penalty function as

\[
p(\vec{s}_i) = A \sum_{j=1}^{q} w_j \hat{g}_j(\vec{s}_i), \quad (3.10)
\]

where \(A\) is a severity factor, \(w_j\) are constraint weights and \(q\) the number of inequality constraints after the transformation of the equality constraints. The severity factor controls how much more important the constraints are than the objective function. This constant should ideally have a value so that \(f(\vec{s}) \approx p(\vec{s})\). But if \(A\) is too low there is a risk that the algorithm fails in producing a solution that satisfies the constraints. A value of 50 showed sufficient in my tests.

The constraint weights are assumed to be normalized so that

\[
\sum_{j=1}^{q} w_j = 1 \quad (3.11)
\]

and they control how the importance is distributed among the constraints (some constraints are more critical than other). It is not always easy to say how this distribution should look like in advance, and it is possible that what is considered optimal changes during the optimization. Therefore we will utilize adaptive weights, a method inspired by Bean’s and Hadj-Alouane’s adaptive penalties\[20\], which use feedback from the search process to find optimal values. All weights are initially equal (or guessed), and then, after every iteration the weights are adjusted. If \(\vec{b}_t\) is the most fit individual in the \(t\):th generation, then the adjusted weights \(w'_j\) are computed in the following manner:

\[
\hat{w}_j = \begin{cases} 
\beta \cdot w_j, & \text{if } g_j(\vec{b}_t) \geq g_j(\vec{b}_{t-1}) \\
 w_j, & \text{otherwise} 
\end{cases} \quad (3.12)
\]

\[
w'_j = \frac{\hat{w}_j}{\sum_{j=1}^{q} \hat{w}_j}, \quad (3.13)
\]

i.e. the \(j\):th weight is multiplied with a constant \(\beta > 1\) if the \(j\):th constraint violation has not decreased, and then the weights are renormalized so that equation (3.11) is satisfied.

The actual constraints taken into consideration in this problem are summarized in table 1. There are two additional constraints, maximum allowed steering angle and maximum allowed steering angle derivative, that were left out, as they never were violated.

Constraint \(g_1\) is a measurement of how much the path that is outside allowed area. The constraint goal of 0 means that the path is completely inside the safety margin. An approximation is used here, though. Preferably outside would measure the length of the parts outside the
An interesting alternative way of dealing with constraints is to completely eliminate them by always generating feasible solutions. This is normally an expensive approach but is sometimes viable. This technique was tested by using a spline interpolation function (Concur from the Fortran library Fitpack) that can satisfy derivative constraints. The curvature constraints can easily be transformed to derivative constraints since the curvature corresponds to a circle with certain radius, and the circle has simple parametric equations from which the sought derivatives can be found. This approach was unfortunately not very successful; it was slow (as expected) and produced suboptimal solutions. The failure might partly be explained by the high curve degree Concur needs (degree $2n+1$ is required to satisfy $n$ derivative constraints).
3.5. THE PENALTY FUNCTION

Figure 3.1: The evolution of a path population.
3.6 Selection

Before we can apply genetic operators such as recombination and mutation to create new individuals, we must select parents. The parents should have good fitness (to increase the likeliness that the offspring is fit too) and be moderately genetically related (some genetic distance increases the chance of picking up new good traits but too much leads to bad offspring). The common approach is to give each individual a certain chance of being selected based on the fitness. Older books on the subject recommend roulette selection, where the chance is equal to the fitness of the individual divided by the total fitness of the population. The problem with this method is that it favors fit individuals too much; the population converges too quickly and may get stuck in a local minimum. Therefore it is nowadays more popular to use some kind of ranking, where the chance of getting selected does not directly depend on the fitness, but on order of the fitness compared to the other individuals. One such elegant method is tournament selection: simply randomly choose two individuals from the population and keep the one with best fitness. It can be easily shown (the proof is irrelevant here though) that this gives a linear probability function

\[
\text{prob}(r, \lambda) = \frac{2\lambda - 2r + 1}{\lambda^2},
\]

where \( r \) is the rank of the individual and \( \lambda \) is the population size. The best individual has rank 1 and thus has the probability \( \frac{2\lambda - 2 + 1}{\lambda^2} \) of being selected while the least fit individual, with rank \( \lambda \), has the probability \( \frac{1}{\lambda^2} \).

This ES algorithm uses tournament selection with one modification: the parents are selected from the 25% best individuals only. The rest of the population has usually degenerated because of bad mutations.

Another implementation detail is that the population size \( \lambda \) is not kept constant during the optimization. It was noticed that the beginning of the optimization is of great importance to the outcome, and therefore we use a larger population at this phase, about 400 individuals. As the constraints are satisfied, \( \lambda \) is exponentially decreased until it reaches a minimum allowed value of 100 individuals. This procedure seems to give a good trade-off between reliability and execution time.

3.7 Recombination

Recombination increases the chance that good traits survives to the next generation. The two common recombination classes in ES are discrete recombination and intermediate recombination. The former class resembles the chromosomal crossover process in biological evolution and means that new genomes are generated by using sequences from different parent genomes. One popular variant is one-point crossover shown in figure 3.2(a), where the first part of the new genome comes from parent 1 and the second from parent 2. Such “cut-and-paste” operations are not that usable in our case, though. The smoothness of a spline is very sensitive.
3.8 Mutation

Mutation in ES algorithms is done by adding perturbations to the parameters. The perturbations are usually normal distributions and each parameter to be optimized has its own standard deviance $\sigma_i$. The deviances should initially be large, so that the search space is scanned thoroughly, and then decrease as the population approaches the final solution. In one early publication on ES, a general rule was given concerning the size of the perturbations, namely that the size is optimal when about a fifth of the mutations are successful. If the success rate is higher, then the deviances should be decreased, otherwise increased. Modern ES algorithms use a slightly more advanced self-adaption mechanism: the deviances $\sigma_i$ are treated as a genome, i.e. we let also the deviances undergo genetic operations such as crossover and mutation! The crossover operator used is exactly the same as the one for the parameters which is described in section 3.7. The mutation operator is, on the other hand, completely different.
Figure 3.3: Examples of discrete, intermediate and transition recombination of B-spline genomes. Dashed lines are control polygons.
3.8. Mutation

Before we mutate the real genome, the actual solution vector, we mutate the deviances using

\[ s = \exp(N(0, \tau)) \]  
\[ \sigma'_i = \sigma_i \cdot \exp(N(0, \tau')) \cdot s, \]  

where \( N \) is the normal distribution and \( \tau \) and \( \tau' \) are defined as

\[ \tau = 1/\sqrt{2n}, \]  
\[ \tau' = 1/\sqrt{2\sqrt{n}}, \]

where \( n \) is the genome length. Using our new deviances \( \sigma'_i \) we can mutate the real genome by adding \( N(0, \sigma'_i) \) perturbations to the parameters. One deviance per optimization parameter means that we have separate deviances for \( x \) and \( y \) coordinates of the control points. The deviances form a mutation ellipsoid, in which the majority of the mutations will reside in. See figure 3.4(a). In theory, separate deviances are good since we can control with greater accuracy what areas of the search space we want to explore. During the testing, I noticed some tendencies to get stuck in local minima, though. Instead of keeping track of separate deviances I found it more reliable to derive them from an “amplitude deviance” and a “composite angle”, i.e. each control point \((x_i, y_i)\) has a deviance \( \sigma_{a,i} \) and an angle \( \alpha_i \) from which we can compute the separate deviance \( \sigma_{x,i}, \sigma_{y,i} \) using

\[ \sigma_{x,i} = \sigma_{a,i} \cos \alpha_i \]  
\[ \sigma_{y,i} = \sigma_{a,i} \sin \alpha_i. \]

\( \sigma_{a,i} \) is mutated using equations (3.16) and (3.17) and \( \alpha_i \) undergo mutation by adding \( N(0, \beta) \) perturbations, where \( \beta = 0.1 \) has shown to be a good choice.

Optimal placements of the control points often lie close to an infeasible region. To improve the search of the boundaries of this region it is possible to rotate the mutation ellipsoid using the transformation

\[ \hat{\sigma}_{x,i} = \sigma_{x,i} \cos \theta_i + \sigma_{y,i} \sin \theta_i \]  
\[ \hat{\sigma}_{y,i} = -\sigma_{x,i} \sin \theta_i + \sigma_{y,i} \cos \theta_i \]

where \( \theta_i \) is the rotation angle. See figure 3.4(b). These \( \theta_i \) undergo the same mutation as \( \alpha_i \).

To summarize, besides the actual genome (the control points and the knot sequence), we have a “shadow genome” consisting of amplitude deviances \( \sigma_{a,i} \), composite angles \( \alpha_i \) and rotation angles \( \theta_i \). The initial values of the amplitude deviances are 0.5 m except for the first and last
one, which are set to 0.2 m. The composite angles are initially set to $\frac{\pi}{4}$, which means that all $x$ and $y$ deviances are of equal size in the beginning. The rotation angles are randomly initiated.

There is one possibility EA offers that was not utilized: if we want we can add or remove genome parameters, i.e. to change the length of the solution vector, during the optimization, as a mutation. This is actually desirable, as it is hard to estimate the optimal length in advance. A longer solution vector means that we can express the path more freely and thus produce better results. However, more genome parameters require more computation time. This option was not utilized, though, for the sake of simplicity. Keeping all genomes of equal size make other tasks, such as recombination, less complicated. It can also be tricky to rebuild the B-spline when changing the genome length.

## 3.9 HDF generation

When the optimization has terminated and found a solution (a B-spline) it must be converted to the HDF format, which defines the path as chain of segments. Each segment has polynomial of degree 7 describing that part of the path. The segments also carry some other data, such as maximum allowed velocity for the part. Because of mechanical reasons, the segments must not be shorter than 3 m.

The conversion would ideally be done by directly converting the B-spline parameters to polynomial coefficients. This task is non-trivial to solve and because of lack of time the polynomials are now constructed simply by using a least-squares fitting method. This gives paths that can be imported into Prototype X and Staffanson's program. However, these paths are not suitable to perform speed and time estimations on, as the fitting gives small discontinuities...
at the joints, which can give rise to bad curvature derivatives. Instead the maximum allowed segment velocity is computed directly from the spline representation.
This chapter explains how the presented algorithm will be evaluated. For convenience, we refer to the ES algorithm presented in this thesis as POES (Path Optimizing Evolutionary Strategy) and Berglund’s algorithm as FPOA (fmincon-based Path Optimizing Algorithm) from this chapter and onwards.

4.1 Implementation details

POES is implemented entirely in Python [21], except for a small piece of code that is written in C because of performance reasons. The choice of Python is motivated by two reasons: it allows agile development, making it excellent for prototyping, and there are good numerics and plotting packages freely available. The spline functions is provided by the numeric package SciPy [22] and plots are produced by matplotlib [23].

4.2 Safety margin

The safety margins are generated by Zachrisson’s code, working in shortest-path mode, which means that it takes two parameters specifying the margin in inner and outer curves. The vehicle tends to drift a bit outwards in the curves, why a stricter margin is required in outer curves. Note that the algorithm and Staffanson’s physical model treat the vehicle as a point. This means that the width of the vehicle must be included in the safety margins.

4.3 Test cases

To evaluate POES it has been applied on a some real scenarios. Unfortunately, at the time these tests were performed, the safety margin code was not fully implemented and it was impossible
to generate margins for arbitrarily mine areas. This means that the evaluation will only consider a few cases. Characteristics for the chosen test paths are shown in table 1. Notable missing cases are:

- paths with different end curvature constraints (all chosen paths have curvature constraints equals to zero),
- shorter paths (that may better reflect a real scenario),
- a path without turns (to verify that POES can produce straight paths) and
- paths from other maps. The chosen paths are from one map named 82039 provided by LKAB (see figure A.1).

In addition to these omissions, the safety margin code was not taking the reflector information, that is supplied in the HDF files, into consideration. These reflectors are placed along the mine walls and the minimum distance between the path and the reflectors should be at least 1 dm. To ensure this, we will do a manual review of the safety margins.

Now on to a description of the cases. Case 1a (see figure B.1 for the safety margin) will show the upper limits of the algorithm. The path is long and has three turns (it is in the turns that time can be saved). The margins used are 2.5 m at inner curves and 3.0 at outer curves (or 0.5 m and 1.0 m in reality, if the maximum width of the vehicle is about 4 m), which is quite strict. A closer examination reveals that the original, hand-drawn HDF paths intersect the generated margins slightly at the inner curves. This stricter margin evens out the inaccuracy of the outside estimate. Case 1b (figure B.2) is the same path but with a sloppy margin. We want to know how much 0.5 m extra space on each side affects the traversal time. Case 1c has the same safety margin as 1a, but the constraint thresholds are higher. It might be so,
that the accuracy used by the HDF format is unreasonably high (e.g. angles have an accuracy of 1/100th degrees), and that it makes it tougher for the algorithms. To test this theory, the thresholds in this test are increased by a factor of 10. Since the previous paths are curvy it would be interesting to see how the algorithm performs on a path with fewer turns. Case 2a (figure B.3) is a straight path except for one turn. Case 2b and 2c (figure B.4) are analogous to 1b and 1c.

On these cases, we want to study result of the following measurements:

**Smoothness** How does the spread of the objective function look like? Do we always hit the same minimum?

**Traversal time** According to the model, do we get improvements of the traversal times? Is there a strong correlation between the smoothness and the traversal time? Note that the traversal time is for only one way (from $p_{start}$ to $p_{end}$) and does not include e.g. loading and dumping time.

**Execution time** One of the drawbacks with the FPOA implementation was the execution time. Is this POES implementation performing better?

**Path length** Optimized paths are more curvy and therefore a bit longer, which reduces the time savings. The impact is probably neglectable, but it would nevertheless be interesting to get figures on how great the loss is.

Since the algorithm is non-deterministic we must execute it several times on each case to get reliable statistical figures. 50 times is assumed to give a good overview of the behavior of the algorithm.

EA algorithms normally use fitness information to decide when the optimization should terminate, but as explained in section 3 we were forced to be less sophisticated: in the cases above each run goes on for 1000 iterations.

When the optimization has terminated, the best individual in the last generation is normally selected as the final solution. But this EA doesn’t use elitism, so the evolution of the population can occasionally go backwards. Instead POES keeps track of the best individual seen during the optimization and returns that.

### 4.4 Comparison to the MATLAB implementation

Because of the numerous differences it is difficult to make a fair comparison between POES and FPOA. A short summary:

- They use different types of safety margins. FPOA uses a very simple method that just translates the mine walls. This type of margin does not give any guarantees on the minimum distance between the walls and the path, and is thus not as strict as it should be.
- POES uses parametric splines instead of univariate splines. This means that the optimization parameters are at least twice as many, which should affect the execution time.

- POES also optimizes the knot sequence which gives further optimization parameters. The knot optimization is also a nonlinear problem, which are generally harder to solve. On the other hand, FPOA relies on a user supplied knot sequence which most likely is far from being optimal.

- FPOA uses somewhat stricter thresholds. Originally it used thresholds of $10^{-8}$, which is much stricter than necessary. For my comparison I have set the constraint thresholds (and the similar objection function tolerance) to $10^{-5}$, and the execution times seems to have dropped significantly by this. But the execution times cannot be compared directly anyway, as the two algorithms are run on different systems (due to MATLAB’s licence). POES was executed on an unloaded P4 2.26 GHz while MATLAB ran on a slower, semi-loaded Sun Fire 480R 4x900 MHz.

Nevertheless, we shall perform some optimizations with FPOA and do a comparison with POES. The evaluation of FPOA will be less rigorous than POES’, though, since its optimization procedure has not been automated.
In this chapter, the results of the evaluation are presented.

5.1 The safety margins

Before we look at the results we make a quick visual inspection of the safety margins (see appendix B). In particular we are interested in if it is a problem that the safety margin code does not take reflectors into consideration. In our four different safety margins (two path cases with two different margin distances) it appears that all reflectors are on acceptable distance from the safety margin (and thus also the path).

5.2 The ES algorithm

Let us start the evaluation of POES by looking at the most interesting part: the traversal times (see the first columns of table 1). We immediately notice a couple of things. In most cases, the ES algorithm produces a significantly optimized path (e.g. improvements of 12–20% for case 1a), but there are runs when the result is worse than the HDF path (case 2a and 2b), and the spread is large. On all test paths we have standard deviances larger than one second, and the largest difference between the best and worst run is almost ten seconds (case 1b). Being a stochastic algorithm, POES is expected to give slightly different result for every run, but this is clearly not an acceptable spread.

The algorithm does not optimize the traversal time directly, but the smoothness. Do we see the same spread there? A quick glance at the middle section of table 1 reveals that the smoothness fluctuates too. One explanation for this could be that 1000 iterations is simply not enough. Evolutionary algorithms tend to get ineffective when they approach the final solution and it is common to hook up a traditional local optimizer to speed up the end phase of the optimization.
### Table 1: Results for POES.

<table>
<thead>
<tr>
<th>Case</th>
<th>Traversal time(^a) (s)</th>
<th>Smoothness (rad/m(^2) (\times) 10(^4))</th>
<th>Execution time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>best</td>
<td>avg</td>
<td>worst</td>
</tr>
<tr>
<td>1a</td>
<td>62.38 (-20.0%)</td>
<td>65.61 (-15.9%)</td>
<td>68.60 (-12.1%)</td>
</tr>
<tr>
<td>1b</td>
<td>51.63 (-33.8%)</td>
<td>54.17 (-30.6%)</td>
<td>60.59 (-22.3%)</td>
</tr>
<tr>
<td>1c</td>
<td>62.33 (-20.1%)</td>
<td>65.38 (-16.2%)</td>
<td>69.13 (-11.4%)</td>
</tr>
<tr>
<td>2a</td>
<td>39.01 (-12.4%)</td>
<td>41.17 (-7.5%)</td>
<td>44.57 (+0.1%)</td>
</tr>
<tr>
<td>2b</td>
<td>38.05 (-14.5%)</td>
<td>39.97 (-10.2%)</td>
<td>46.00 (+3.3%)</td>
</tr>
<tr>
<td>2c</td>
<td>39.01 (-12.4%)</td>
<td>40.42 (-9.2%)</td>
<td>43.16 (-3.1%)</td>
</tr>
</tbody>
</table>

\(^a\)Traversals times are according to the model. Percental differences are against the traversal time of the corresponding HDF path for that case.
5.2. THE ES ALGORITHM

Letting the optimization run for another few thousand iterations does indeed seem to reduce the standard deviance. But at the same time, the traversal times surprisingly get somewhat worse. It turns out that the correlation between the smoothness and traversal time is suspiciously weak. Two interesting examples of the relationship are shown as scatter plots in figure 5.1(a) and 5.1(b). The data for the first plot comes from case 1a, and almost no correlation is discernible here. The fact that the objective function (the smoothness) does not use gear information can contribute to this. Each gear has a maximum allowed curvature derivative, and when this limit is reached, the vehicle has to shift to a lower gear. The paths often have a such shape that the curvature derivative runs close to one limit, which results in that the path sometimes breaks the limit (and has to shift to a lower gear) and sometimes not (and can keep running at the higher gear). See figure 5.2 for an example. Every time the path breaks a limit a couple of seconds are lost, and thus this has a large impact on the traversal time. Furthermore, the vehicle cannot shift gear instantly; there has to be at least 3 m between every gear shift. For now, this is achieved by naively dividing the path into 3 m long segments, where the top speed for each segment depends on its worst curvature derivative, i.e. the vehicle must not run faster than the slowest point of the segment allows. Simply dividing the path uniformly is not optimal and may worsen the spread, although this contribution is probably small.

The second scatter plot, for case 2a, hints at another problem: the algorithm sometimes converges towards different optima, which may have the about same smoothness but different traversal times (because of different curvature derivative characteristics). The two optima in case 2a are vaguely noticeable if we plot the solutions from all 50 runs in the same figure (see figure 5.3). The source of this multimodularity could be the optimization of the knot sequence, but there are also other suspects. It is not completely investigated how going from univariate to parametric splines affects the objective function, and the penalty function may have multiple minima (e.g. because of the discrete outside constraint).

One more problem can spotted in the second scatter plot. A few percent of the runs result in solutions that are distinctly worse than the rest. The exact cause of this has not been determined. Initially, the suspicions were directed at the evaluation of the spline. Some of these “outliers” have in common that two or more of their control points for some reason get very close. The average point distance is about 0.3–0.5 m and if the control points get close, bends in the path could go undetected. But increasing the point density did not seem to make this problem go away. It may instead be a combination of the above mentioned problems, or just that POES becomes ineffective when this happens.

In case 1b and 2b we explore how the traversal time is affected by the safety margin. A 0.5 m smaller margin (on each side) further shortens the (average) traversal time in both cases. In case 1b we see improvements up to 33%. But at the same time the standard deviance gets somewhat larger. Case 2b actually gets a higher worst time. More surprisingly, this generous safety margin resulted in about twice as long execution times (see last columns in table 1). My interpretation of this is that the given extra space reduces the pressure on POES to minimize the mutation perturbations. Larger perturbations result in slower convergence and it will take longer time before the constraints are satisfied, and this in turn holds up the population size
Figure 5.1: Scatter plots showing the relationship between smoothness and traversal time for cases 1a and 2a. The correlations are 0.14 and 0.08, respectively.
Figure 5.2: Curvature derivative plots for two different paths from case 1a. The close-to-horizontal lines mark where the vehicle must shift gear. As seen in the circles, one of the paths has to shift to a lower gear at two times, which gives that path a much worse traversal time (68.6 s compared to 62.4 s) when its smoothness is only marginally worse (0.00154 compared to 0.00148).
(because of the mechanism described in section 3.6) and with that also the execution time.

The execution times for case 1a/1c and 2a/2c are about 18 and 13 minutes, respectively, which was a minor disappointment. The very first (elitistic) version of the ES algorithm had much better performance (a few minutes on cases equivalent to 1a). During the development of POES, the performance has been sacrificed for the sake of reliability.

In case 1c and 2c the constraints thresholds were looser. The purpose was to see if the optimization was heavily constrained by the default thresholds. The results indicate that this hypothesis should be rejected. The traversal times and the execution times do get somewhat better but the gain is mostly neglectable.

Figure 5.3: The solutions from all 50 runs for case 2a. N.B. the aspect ratio is not 1:1.
5.3 The MATLAB implementation

We end the evaluation of POES by establishing that the optimized paths are a bit longer, about 0.5–2 m (see table 2). If the average velocity is around 3 m/s we estimate that the extra length is equivalent to a loss of not more than 1 s, a relatively small price when we in some cases gain more than 10 s.

<table>
<thead>
<tr>
<th>Case</th>
<th></th>
<th>Path length (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>best</td>
<td>average</td>
</tr>
<tr>
<td>1a</td>
<td>218.89</td>
<td>219.08</td>
</tr>
<tr>
<td>1b</td>
<td>218.20</td>
<td>218.55</td>
</tr>
<tr>
<td>1c</td>
<td>219.00</td>
<td>219.09</td>
</tr>
<tr>
<td>2a</td>
<td>150.29</td>
<td>150.33</td>
</tr>
<tr>
<td>2b</td>
<td>150.11</td>
<td>150.63</td>
</tr>
<tr>
<td>2c</td>
<td>150.30</td>
<td>150.35</td>
</tr>
</tbody>
</table>

Table 2: Path lengths.

The evaluation of FPOA was a bit troublesome. Many of the optimization runs failed to converge within 1000 iterations, especially at tougher safety margins. For this reason the safety margin was kept at 2.0 m (i.e. no effective margin at all when taking the vehicle width into account).

When FPOA did converge, it usually needed less than 60 iterations. The execution time seemed mostly dependent on the number of knots. At 15 knots, the optimization was usually done in 30 minutes, while 20 knots led to execution times around 60–90 minutes and with 25 knots it could go on for almost 10 hours. But there were also exceptions, like one run with 28 knots that completed in 134 minutes.

The traversal times for the produced paths varied greatly too. In case 1 (HDF stretch 99–139), the traversal times were 68.42–71.12 s, an improvement of 8.9–12.3% compared to the manually constructed path. In case 2 (HDF stretch 99-215), FPOA had serious problems. The produced paths always took the turn in the beginning in strange way, resulting in very bad curvature derivatives that violated the limit even at the first gear. It is unclear what caused this. It is likely not a fundamental problem of FPOA, but something that can be fixed with fine tuning (perhaps giving it another start point, or improving the naive safety margin).
CHAPTER 6
Discussion

The last chapter concludes the work and discusses possible future improvements and research.

6.1 Conclusions

This thesis presents an evolutionary strategy for optimizing drive paths for electrical wheel loaders, specifically Tamrock Toro 2500. The introduction mentioned three goals we wanted to achieve: make the optimization procedure user friendly, remove the limitations concerning monotonicity and reduce the execution time. The two first goals may be considered reached. The new algorithm does not require any expertise from the user (who now does not have to perform tasks such as knot placement), and it works on arbitrarily mine parts (assuming a safety margin can be provided). Whether the current implementation of POES is significantly faster than the old MATLAB implementation is debatable. The performance of Berglund’s code seem to improve when using sane thresholds and, considering it was ran on slower hardware, its execution times were sometimes competitive. But on the other hand, FPOA solved a somewhat easier problem and, as we saw in the evaluation, there are cases when it fails to produce a feasible solution, a behavior not observed with POES. In that regard POES clearly is an improvement.

6.2 Future work

A few tasks concerning POES are left uncompleted. The first is to reimplement in Java to make the integration with Prototype X smoother. The move to Java makes it also easier for users to install the program, as extra dependencies (such as Python and SciPy) are avoided. The core of the algorithm is already ported. What is missing are the HDF generation and code for spline interpolation. The HDF generation could be implemented fairly easy using the least-squares method, but since its accuracy is not satisfactory and the performance is not very impressive
it would be preferable to develop an algorithm for converting the spline parameters directly to polynomials. At the same time it might be advisable to study if it is worth to replace the code that now uniformly selects the segment joints to something smarter. The generation of the first population requires spline interpolation. Some spline code for this purpose already exists (a Java port of Farin’s C code [24]), but it only supports cubic splines (degree 3). We need support for quartic splines (degree 4).

Another implementation detail that could need a review is the computation of the derivatives. We saw in the evaluation that the difference between a “good” and “bad” optimized path is sometimes small. To eliminate the risk that numerical errors propagate through the calculations it would be desirable to replace the derivative computation (that relies on forward differences of the evaluation points) with one that use the spline parameters directly.

A quick profiling study of the Java code shows that it will be a bit faster than the Python+C version (although this is using cubic splines). The largest bottleneck seems to be the routine that compute normal distributions, which at first is somewhat surprising. But remember that every iteration, all individuals are mutated, and the mutation requires several random values for every optimization parameter, which ends up being a lot. Even if the implementation of this routine comes from CERN’s Java package for “high performance generation of random numbers” [25] it should be investigated if it is possible to speed up the generation of normal distributions. Other computationally heavy routines are spline evaluation and the outside constraint. The latter can probably be improved, subject to both performance and accuracy, perhaps by making use of triangulation.

The evaluation identified a couple of problems with the algorithm itself. There were indications on that the smoothness function, while giving optimized paths, was not an ideal objective function. Since this ES has a potential to handle non-continuous objective functions it would be interesting to see if minimizing the time function directly would lead to improvements. This could also make it easier to decide when the optimization should terminate, as we then can make a direct interpretation of the objective function. If the termination problem would appear insoluble, a fallback solution could be to continuously update the user about how the optimization proceeds and give him/her the possibility to halt if the process seems to have stagnated.

An advanced feature that did not make into POES was mutation of the genome length. If it is possible to rebuild the B-spline in an efficient way, it would be very interesting to see how this would affect the traversal time and the general reliability of the algorithm. Rebuilding the B-spline without changing the genome length could possibly fix the problem with control points getting close.
Figure A.1: Map 82039 and paths from corresponding HDF file. The ore is fetched from the upper vertical drifts and dumped in the shafts in the lower, shorter drifts.
Figure B.1: Safety margin for case 1a and 1c.
Figure B.2: Safety margin for case 1b.
Figure B.3: Safety margin for case 2a and 2c.
Figure B.4: Safety margin for case 2b.
REFERENCES


