Multi-fidelity Algorithms for the Horizontal Alignment Problem in Road Design

by

Mahdi Aziz

B.Sc., Khayyam Institute of Higher Education, 2010

M.Sc., Amirkabir University of Technology, 2013

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The College of Graduate Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Okanagan)

July 2018

© Mahdi Aziz, 2018

The following individuals certify that they have read, and recommend to the College of Graduate Studies for acceptance, a thesis/dissertation entitled:

Multi-fidelity Algorithms for the Horizontal Alignment Problem in Road Design
submitted by Mahdi Aziz in partial fulfilment of the requirements of the degree of Master of Science

Dr. Yves Lucet, Irving K. Barber School of Arts and Science
Supervisor

Dr. Warren Hare, Irving K. Barber School of Arts and Science
Co-supervisor

Dr. Ramon Lawrence, Irving K. Barber School of Arts and Science
Supervisory Committee Member

Dr. Michael Friedlander, Department of Computer Science, UBC
University Examiner

Abstract

Multi-fidelity refers to methods that employ low fidelity and high fidelity models in their design so as to lower the computation time and obtain high accuracy simultaneously. In particular, the algorithms that use the multi-fidelity technique in their design adaptively controls the error of the objective function. As they reach better positions in the search space, they update the objective function and use a new version with better precision. In this thesis, we study the multi-fidelity algorithms for solving the horizontal alignment problem in road design. The algorithms that we study here are a generalized pattern search with adaptive precision control and a trust region algorithm for unconstrained problems with controlled error. At first, in order to make a fair comparison, we tune the parameters of each algorithm on 5 small roads. We then test the algorithms on 35 roads, ranging from small to very large roads. The results of the comparison demonstrate that the use of the multi-fidelity framework helps the horizontal alignment algorithms reduce their computation time by an average factor of 2.52 while preserving the quality of solutions.

Lay Summary

We provide faster algorithms to design a road with the minimum cost that satisfies regulatory, safety, and environmental constraints. Numerical experiments on a test set of 35 roads provided by our industrial partners validate the algorithms and show a speedup of 2.52 while keeping the approximation error within an acceptable tolerance.

Table of Contents

Abstract	iii
Lay Summary	iv
Table of Contents	v
List of Tables	vii
List of Figures	viii
Acknowledgements	x
Dedication	xi
Chapter 1: Introduction	1
1.1 Road design optimization	1
1.1.1 Horizontal alignment problem in road design	2
1.2 Multi-fidelity algorithms	4
1.3 Motivation	6
1.4 Organization of this thesis	7
Chapter 2: Multi-fidelity algorithms	8
2.1 Generalized pattern search algorithms with adaptive precision control	8
2.1.1 Mathematical background	8
2.1.2 Generalized pattern search: general framework	10
2.1.3 Generalized pattern search: an adaptive precision control version	12
2.2 A trust region algorithm with controlled error	15
2.2.1 Creating a surrogate model using support vector regression model	20

TABLE OF CONTENTS

2.2.2	Find the approximate minimum of the model	21
Chapter 3:	Multi-fidelity algorithms for solving the HA . . .	22
3.1	Terminology	22
3.2	Objective function: vertical alignment problem	25
3.3	A surrogate model for the VA problem	26
3.4	A general approach	28
3.5	A multi-fidelity GPS algorithm for the HA problem	29
3.6	A multi-fidelity trust region algorithm for the HA problem . .	30
Chapter 4:	Experimental results	33
4.1	Road set	33
4.2	The effect of the multi-fidelity framework on the GPS algorithm	34
4.2.1	Parameter tuning	34
4.2.2	Experimental results on 35 roads	37
4.2.3	The effect of the multi-fidelity on the GPS algorithm for different classes of roads	39
4.3	The effect of the multi-fidelity framework on trust region al- gorithm	42
4.3.1	Parameter tuning	42
4.3.2	Experimental results on 35 roads	44
4.3.3	The effect of the multi-fidelity on the TR algorithm for different classes of roads	45
4.4	Comparison with a standard solver	48
4.4.1	Comparison with NOMAD for different classes of roads	50
Chapter 5:	Conclusion	55
Bibliography		58
Appendix		63
	Appendix A: Data Tables	63

List of Tables

Table 3.1	A mapping table from ϵ_k to N	28
Table 4.1	Roads tested on different workstation PC computers. . .	34
Table 4.2	Parameter values for GPS0 and GPS1 algorithms . . .	35
Table 4.3	Speed up and cost difference for the GPS algorithm . .	38
Table 4.4	Optimization failure for the GPS algorithm	39
Table 4.5	Categorization of roads based on the number of IPs . .	39
Table 4.6	Speed up for GPS algorithm for two categories of IPs	40
Table 4.7	Cost difference for GPS for two categories of IPs . . .	40
Table 4.8	Categorization of roads based on the number of stations	41
Table 4.9	Speed up for GPS for groups of stations	41
Table 4.10	Cost difference for GPS for groups of stations	42
Table 4.11	Parameter values for TRSVR0 and TRSVR1 algorithms	44
Table 4.12	Speed up and Cost difference for the TRSVR	45
Table 4.13	Speed up for the TRSVR algorithm for categories of IPs	46
Table 4.14	Cost difference for the TRSVR algorithm for cate- gories of IPs	46
Table 4.15	Speed up for the TRSVR algorithm for groups of stations	47
Table 4.16	Cost difference of TRSVR algorithm for groups of sta- tions	48
Table 4.17	The parameter values for NOMAD	49
Table A.1	GPS0 for the HA problem across all roads	64
Table A.2	GPS1 for the HA problem across all roads	65
Table A.3	TRSVR0 for the HA problem across all roads	66
Table A.4	TRSVR1 for the HA problem across all roads	67
Table A.5	NOMAD0 for the HA problem across all roads	68
Table A.6	Comparison of GPS0 and GPS1 algorithms	69
Table A.7	Comparison of TRSVR0 and TRSVR1 algorithms . . .	69
Table A.8	Cost comparison among 5 HA algorithms	70
Table A.9	Time comparison among 5 HA algorithms	71

List of Figures

Figure 2.1	Positive basis sets and a positive spanning set in $\mathbb{R}^2$.	9
Figure 3.1	A satellite view of a road	23
Figure 3.2	A sample road with stations and vertical and horizontal offset points	24
Figure 3.3	A road with four segments and 14 stations	25
Figure 3.4	A corridor with five IPs represented by blue stars . .	25
Figure 3.5	The objective function	26
Figure 3.6	The flowchart of a general approach for solving the horizontal alignment problem	29
Figure 3.7	The flowchart of the general multi-fidelity approach .	30
Figure 3.8	The trajectory plot of the GPS algorithm for Road 35	31
Figure 3.9	The trajectory plot of the trust region algorithm for Road 35	32
Figure 4.1	The performance profile of GPS0 algorithm	36
Figure 4.2	The performance profile of GPS1 algorithm	37
Figure 4.3	The horizontal alignment by GPS algorithms	43
Figure 4.4	The performance profile of TRSVR0 algorithm	44
Figure 4.5	The performance profile of TRSVR1 algorithm	45
Figure 4.6	The horizontal alignment by TRSVR algorithms . . .	48
Figure 4.7	The performance profile of NOMAD	49
Figure 4.8	The performance profile of 5 different algorithms . . .	50
Figure 4.9	The performance profile of 5 different algorithms for Category C1 roads	51
Figure 4.10	The performance profile of 5 algorithms for Category C2 roads	52
Figure 4.11	The performance profile of 5 algorithms for Group 1 roads	52
Figure 4.12	The performance profile of 5 algorithms for Group 2 roads	53

LIST OF FIGURES

Figure 4.13	The performance profile of 5 different algorithms for Group 3 roads	53
-------------	--	----

Acknowledgements

I would like to thank Dr. Yves Lucet as my main supervisor for continuous help and guidance for this project. He has always been very welcoming to help during the project. And, Dr. Warren Hare as my co-supervisor gave me very helpful comments and guidance during the project.

Special thanks to The University of British Columbia, Softree Technical System Inc., and the federal government of Canada for providing me financial support. This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) through Collaborative Research and Development grant #CRDPJ 479316-15 sponsored by Softree Technical Systems Inc. Additional funding was provided by the Graduate Entrance Scholarship and the University Graduate Fellowship from the University of British Columbia. Part of the computation in this research was carried out, using a software library provided by Softree Technical System Inc. Part of the research was performed in the Computer-Aided Convex Analysis (CA2) laboratory funded by a Leaders Opportunity Fund (LOF, John R. Evans Leaders Fund Funding for research infrastructure) from the Canada Foundation for Innovation (CFI) and by a British Columbia Knowledge Development Fund (BCKDF).

Dedication

I would like to dedicate my work to M.Z.R. and K.L.

Chapter 1

Introduction

In this chapter, we provide a brief overview of the road design optimization and multi-fidelity algorithms, and our motivation to solve the horizontal alignment in road design optimization.

1.1 Road design optimization

Road design optimization is a hot topic in the design of transportation systems as it has significantly contributed to the development of today's economy. For instance, according to one analysis published by the Canadian Government, 15.9 billion dollars were spent by the territorial, provincial and federal agencies to build and extend roads in 2015/2016. Moreover, in 2015/2016 more than 50 percent of revenue gained by the territorial, provincial and federal government comes from road transportation [25].

Road design optimization refers to the problem of finding a series of tangents and curves between two given points in a given area while minimizing the construction cost between these points and following all the design constraints given. The problem can also be defined as the positioning of cross section points between two given starting and end points in a road such that all associated costs are minimized, and all design constraints are satisfied.

There are many costs associated with road design. However, there are five principal costs that need to be taken into account: planning, design and administrative cost; construction cost; maintenance cost; social, political and environmental cost; and user cost [20]. In road design, an alignment refers to a curve between two specified points in a three-dimensional space. The horizontal projection between those points is called the horizontal alignment. The vertical alignment is the profile aspect of the road, and it comprises crest and sag curves, and the straight grade lines connecting these curves.

The road design optimization problem usually can be categorized into three sub-problems: earthwork optimization, vertical alignment optimization and horizontal alignment optimization. These sub-problems are closely related to one another where in order to solve one you need to take care of

the other two sub-problems first. Given a fixed vertical alignment, earthwork optimization can be defined as the problem of minimizing the overall cost of hauling, excavation and embankment. Given specified quantities of earth materials and a given ground profile, the objective is to haul, embank or excavate earth materials in a way that the ultimate cost would be minimized. If for a given road the ground profile is less than the road profile, then we need to fill it with allowed materials. Otherwise, we need to cut those materials from the ground. There is also the cost for moving (hauling) materials from cutting areas to filling areas, which is called hauling cost. The objective is to minimize the sum of the embankment, excavating and hauling costs.

Given a fixed horizontal alignment, vertical alignment optimization refers to the problem of minimizing the earthwork cost while following all vertical design constraints. The major design constraints for the vertical alignment are the rate of curvature and the minimum and maximum of allowable grades. The allowable grades are determined by some factors, including the type of terrain (the topology of a road) and the maximum speed limit [13].

The horizontal alignment, on the other hand, refers to the problem of finding the best feasible sets of curves and tangential line segments between two given points such that the earthwork cost is minimized and all associated horizontal and vertical design constraints are satisfied. The principal horizontal design constraints are the minimum radius of curvature of the circular arcs that is essential for safety requirements. Further details on the road design problem can be found in [20].

1.1.1 Horizontal alignment optimization problem in road design

Optimizing the horizontal alignment in road design is a challenging task as the problem is extremely complicated and intertwined with earthwork and vertical alignment optimization problems. There are a few optimization algorithms that have been proposed for solving the horizontal optimization problem. Some of them considered the horizontal alignment along with vertical alignment optimization problem and solve the problems simultaneously. For example, Aruga [2] proposed a tabu search for optimizing the vertical and horizontal alignments at the same time. They presented two different ways to create the initial solutions, one is to manually select the initial solution and the other to use Dijkstra algorithm to create the initial solution. Using the initial solution, the tabu search creates the alternative horizontal and vertical alignments in a small number of iterations. They

showed that using the tabu search they can reduce the computational cost significantly. While the tabu search algorithm returns a better horizontal optimization quickly, due to its nature it does not necessarily converge to a stationary point. Another example is genetic algorithm in conjunction with GIS (geographic information system) that was proposed to find a feasible alignment for the horizontal problem in road design [33]. Like the tabu search algorithm, the genetic algorithm suffers from non-deterministic solutions. However, the authors [33] have shown that the algorithm can provide an acceptable solution for the 3-d dimensional alignment optimization problem very fast.

In contrast to heuristic algorithms that do not guarantee (even local) optimality, Hirpa et al. [15] proposed a method that simultaneously solved the horizontal and vertical alignments together while converging to the local minimizer. In fact, they considered the problem as the optimization of three-dimensional road alignments where the objective function is considered as a bi-objective function. The first objective is to minimize the cost of the length of the road and the second one is to minimize the cost of the volume of earthwork. Since the problem is computationally expensive, they proposed a surrogate model to reduce the cost of simulating the objective function and used NOMAD [3] (Nonlinear optimization with Mesh Adaptive Direct search) to solve the multi-objective problem efficiently.

Considering the horizontal alignment as a multi-level optimization problem, Mondal [23] studied two derivative-free optimization solvers, namely NOMAD and HOPSPACK (Hybrid Optimization Parallel Search PACKAGE) to solve the horizontal alignment optimization in road design. They designed a multi-layer framework such that in the inner layer the algorithm solves the earthwork optimization problem that satisfies the vertical design constraints for a fixed alignment. Then, in the outer layer, the algorithm solves the horizontal alignment problem satisfying all horizontal design constraints. They have shown that using NOMAD, their model guarantees local optimality. However, even though their model can solve the horizontal alignment effectively, it takes a long time to evaluate the objective function (optimizing the vertical alignment), resulting in huge computation time in the horizontal alignment optimization. In this thesis, we apply a surrogate [14] that removes some cross sections in the road so that the evaluation of the objective function is less costly. In the next chapter, we use algorithms that employ a multi-fidelity framework, and in Chapter 4 we show that using this surrogate in the multi-fidelity framework the algorithm can achieve huge time saving as well as good solution quality.

1.2 Multi-fidelity algorithms

Surrogate-based optimization algorithms are a class of optimization algorithms that is concerned with the utilization of surrogate functions (models) to find a solution faster. These algorithms are usually employed when evaluating the objective function of a problem is prohibitively expensive such as the horizontal alignment in road design [29].

A surrogate function is an approximation of a true objective function that shares some similarities with the true objective function. Ideally, a surrogate function is faster to compute than the true objective function. Furthermore, the local minimizers of the surrogate function should be close to the local minimizers of the true objective function [4].

Solving real-world problems such as designing the wings of aircraft or solving the vertical optimization problem in road design are usually difficult and time-consuming. For such problems, performing even a small number of function evaluations is highly time-consuming. For this reason, multi-fidelity algorithms that use different levels of accuracy in function evaluations were introduced. They are the combination of low fidelity and high fidelity models such that depending on the problem at hand, they use high fidelity, low fidelity or a combination of both. The low fidelity and the surrogate terms are occasionally used interchangeably. However, there is a subtle difference between them. A surrogate function can be a fine approximation of the true objective function, which is considered as a high fidelity model. Alternatively, it can be a coarse approximation of the true objective function, which can be called a low fidelity model.

The main objective of multi-fidelity algorithms is to reduce the computation time of the optimization by using a low fidelity model and increasing the quality of the solutions by taking advantage of the high fidelity model at some point in the search process. These algorithms use a surrogate function(s) (a low fidelity model) instead of the true objective function (the high fidelity model) to evaluate the problem, and as they make progress toward the better solutions, they use more accurate surrogate functions. Finally, the algorithm stops as it reaches a local minimizer.

There have been numerous multi-fidelity algorithms designed for a wide range of purposes, ranging from aerodynamic designs [18] to mechanical engineering designs [17]. For example, in [16] authors designed a multi-level fidelity surrogate model for solving an engineering design problem. They demonstrated that using variable-fidelity meta-modeling, the algorithm can find more accurate engineering designs while having the same simulation cost.

One of the important aspects of multi-fidelity algorithms is the model management strategy. This strategy is about deciding which surrogate model should be used at a particular time of the search process. In terms of the model management strategy, authors in [27] classified the multi-fidelity algorithms into three strategies: adaptation, filtering and fusion. Adaptation strategy refers to adapting the low fidelity model to the problem at hand using the information gathered from the high fidelity model. For example, authors in [21] used the Kriging model as the low fidelity model in the algorithm where the results of the Kriging model is updated and improved using the results of the high fidelity model provided. In particular, the algorithm starts with a crude initial Kriging model to solve the problem. At the next iteration, the high fidelity model is utilized to provide a more accurate input for the low fidelity model. Using this input, the Kriging model provides better and more accurate interpolation.

On the other hand, the fusion strategy refers to the methods that utilize low and high fidelity models simultaneously. An example of this strategy is the co-Kriging method where the output of the low fidelity Kriging method is combined with the output of the high fidelity model at each iteration of the algorithm. One of the benefits of this technique is that no *a priori* information about the problem is required [26].

Lastly, methods based on the filtering work in a way that the high fidelity is employed when more precision is required [10]. This technique usually starts with a low-accuracy surrogate model where the objective is to minimize the surrogate model. As the algorithm converges to better points in the search space, the surrogate model utilizes the higher accuracy. Examples of this technique can be found in [28, 31, 36].

Another important aspect of multi-fidelity algorithms is how the high fidelity model is converted to a low fidelity one. When it comes to converting high fidelity models to low fidelity ones, there are many techniques including dimensionality reduction, linearization, less refinement, simpler geometry, simpler physics, etc. [10]. In this thesis, in order to generate the low fidelity model for the objective function of our problem, we apply the less refinement technique in Section 3.3 where less number of cross sections are used for optimizing the vertical alignment for a fixed horizontal alignment.

As authors in [10] stated multi-fidelity algorithms have more heavily been used for mechanical, aerospace engineering problems such as designing an engineering system [17] or the wings of aircraft [16, 18]. However, in recent years some researchers have utilized the idea of multi-fidelity in other fields. For example, Polak and Wetter in [28] proposed a new version of the Generalized Pattern Search (GPS) for unconstrained optimization problems

that adaptively controls the error of the objective function. In particular, the algorithm begins with the low fidelity version of the true-objective function. Then as the algorithm converges to better points in the search space, a higher fidelity model is employed. Using this technique, they have proved that even though the objective function is not accurate, the algorithm will still converge to a local minimizer. The algorithm was employed to simulate the energy consumption of a building. They showed that it can solve the problem more quickly than using the algorithm with only the high fidelity model. Another example of multi-fidelity algorithms used in other fields is the implied volatility estimation where authors [31] showed that using a trust region algorithm based on the support vector regression model, they can solve the problem much faster than surrogate-free version. Using the idea of multi-fidelity, they proved that the algorithm converges to the stationary point.

In this thesis, we apply both GPS with adaptive control [28] and the trust region algorithm with controlled error [31] to solve the horizontal problem in road design. The algorithms we choose fall into multi-fidelity hierarchical methods as the algorithms do not employ high fidelity models at each iteration, and they use the high fidelity models when they are close enough to the stationary point. Moreover, the algorithms can also be considered as filtering multi-fidelity algorithms as the model management strategy of the algorithm decides when the high fidelity model or low fidelity model should be used.

1.3 Motivation

Evaluating the design of the horizontal alignment is extremely expensive in terms of both time and memory consumption. Using a portion of cross sections to solve the vertical alignment was shown to be a good surrogate for the true objective function [14]. In [14], authors showed that we can approximate the vertical alignment in road design with 5 percent error when almost 95 percent of cross sections of roads are removed, resulting in huge time-saving. In this thesis, we apply this surrogate to decrease the size of the vertical alignment, which results in solving the vertical alignment at a faster pace. As a result we can solve the horizontal alignment faster than using the true objective function. In particular, using the idea of adaptive multi-fidelity, we first solve the horizontal alignment problem with low precision and as the algorithm find better horizontal alignments, a more accurate surrogate is used, thereby saving the computation time while maintaining

the overall accuracy at a reasonable level.

1.4 Organization of this thesis

The remainder of this thesis is structured as follows. In Chapter 2, we discuss two multi-fidelity algorithms. In Chapter 3, we explain how we apply these two multi-fidelity algorithms to solve the horizontal alignment in road design. In Chapter 4, we report the results of the algorithms, and compare the results with the versions of the algorithms without the multi-fidelity approach on 35 road problems. The results clearly demonstrate that when applying the multi-fidelity framework to our algorithms, we save significant computation time. In Chapter 5, we conclude the thesis and shed some lights on future works.

Chapter 2

Multi-fidelity algorithms

In this chapter, we discuss the multi-fidelity algorithms that we apply to the horizontal alignment in road design.

2.1 Generalized pattern search algorithms with adaptive precision control

In this section, a Generalized Pattern Search (GPS) for unconstrained optimization problems with adaptive precision is studied [28]. The algorithm is suitable for computationally expensive optimization problems. In order to understand the algorithm properly, we should first recall some basic concepts in Linear Algebra.

2.1.1 Mathematical background

Definition in this section are from [4].

Definition 2.1. *Linearly Independent.* The set of vectors $\{v_1, v_2, \dots, v_m\}$ where $v_m \in \mathbb{R}^d$ is said to be linearly independent if and only if there is only one solution to the following equation:

$$\sum_{i=0}^m \lambda_i v_i = 0. \quad (2.1)$$

Definition 2.2. *Span.* For a set of vectors $\Omega \subset \mathbb{R}^d$, its span is defined as

$$\text{Span}(\Omega) \triangleq \left\{ \sum_{i=1}^m \lambda_i \theta_i : m \geq 1, \lambda_i \in \mathbb{R}, \theta_i \in \Omega \right\}. \quad (2.2)$$

Definition 2.3. *Basis.* A set of vectors Ω is called a basis of $\mathbb{R}^d$ if and only if it is linearly independent and $\text{Span}(\Omega) = \mathbb{R}^d$.

One important feature of a basis is that it must have exactly d vectors.

Figure 2.1: Two positive basis sets and a positive spanning set in $\mathbb{R}^2$

Definition 2.4. *Positive Spanning Set.* A set of the non-negative linear combination of vectors in Ω is said to be a positive spanning set if and only if $\text{PSpan}(\Omega) = \mathbb{R}^d$ where

$$\text{PSpan}(\Omega) \triangleq \left\{ \sum_{i=1}^m \lambda_i \theta_i : m \geq 1, \lambda_i \geq 0, \theta_i \in \Omega \right\}. \quad (2.3)$$

Definition 2.5. *Positive Linearly Independent.* A set of vectors Ω is said to be positive linearly independent if and only if for any $\theta \in \Omega$, $\theta \notin \text{PSpan}(\Omega \setminus \{\theta\})$.

It means that a positive linearly independent set does not possess excessive vectors.

Definition 2.6. *Positive Basis.* A set is said to be a positive basis if and only if it is positive linearly independent and a positive spanning of $\mathbb{R}^d$.

Figure 2.1 shows two examples of positive basis sets and a positive spanning set in $\mathbb{R}^2$. As shown in Figure 2.1, the first two sets are the positive basis as their linear combination covers the whole space in $\mathbb{R}^2$, and it is not possible to remove one of the directions in the aforementioned sets and get a positive spanning set. However, the last one is only a positive spanning set and not a positive basis, as by removing two directions in the middle, the remaining directions can still cover the whole space in $\mathbb{R}^2$. Therefore, the set is not positive linearly independent and as a result not a positive basis.

Definition 2.7. *Canonical Maximal Positive Basis.* The canonical maximal positive basis of $\mathbb{R}^d$ is defined as

$$\Omega_{max} \triangleq \{e_1, e_2, \dots, e_d, -e_1, \dots, -e_d\} \quad (2.4)$$

where e_i is the i^{th} unit vector.

2.1. Generalized pattern search algorithms with adaptive precision control

Definition 2.8. *Canonical Minimal Positive Basis.* The canonical minimal positive basis of $\mathbb{R}^d$ is defined as

$$\Omega_{min} \triangleq \{e_1, e_2, \dots, e_d, -(\sum_{i=1}^d e_i)/d\} \quad (2.5)$$

where e_i is the i^{th} unit vector.

Definition 2.9. *Mesh.* A Mesh M is a set of vectors that is defined as

$$M(x, \delta) \triangleq \{x + \delta D_y : y \in \mathbb{N}^p\} \subset \mathbb{R}^d, \quad (2.6)$$

where $x \in \mathbb{R}^d$, $\delta \in \mathbb{R}_+$, $D_y \subset D$, $D = GZ \in \mathbb{R}^{d,p}$, $G \in \mathbb{R}^{d,d}$ is an invertible matrix, $Z \in \mathbb{Z}^{d,p}$, and the columns of Z constructs a positive spanning set for $\mathbb{R}^d$. We also define the set of non-negative real numbers $\mathbb{R}_+$ as $\mathbb{R}_+ = \{x \in \mathbb{R} : x \geq 0\}$.

Definition 2.10. *Box-constrained Optimization Problem.* The function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is defined as

$$\min_x \{f(x) : x \in R^d, l_i \leq x_i \leq u_i, i = 1, \dots, d\}, \quad (2.7)$$

where l, u are the lower and upper bounds, respectively.

Definition 2.11. *Box-constrained Optimization Problem with Error.* The function $\tilde{f}(x, \epsilon)$ with ϵ error is defined as,

$$\min_x \{\tilde{f}(x, \epsilon) : x \in R^d, \epsilon \in \mathbb{R}_+, l_i \leq x_i \leq u_i\} \quad (2.8)$$

where $\tilde{f} : \mathbb{R}^d \times \mathbb{R}_+ \rightarrow \mathbb{R}$, $x \in \mathbb{R}^d$, $\epsilon \in \mathbb{R}_+$ and $\mathbb{R}_+$.

Definition 2.12. *Local Search Set:* The local search set $\iota : (X \times \mathbb{R}_+) \rightarrow \mathbb{R}_+$ is defined as,

$$\iota(x, \delta) \triangleq \{x + \delta \Omega_{max}\}, \quad (2.9)$$

where $X \subset \mathbb{R}^d$ and Ω_{max} is the canonical maximal positive basis.

2.1.2 Generalized pattern search: general framework

The generalized pattern search (GPS) is a famous DFO methods that can be applied to both unconstrained and constrained optimization problems [28]. It belongs to the subclass of DFO methods, called direct search

2.1. Generalized pattern search algorithms with adaptive precision control

methods. It starts with an incumbent solution and explores the search space by generating many trial points. The key component of this algorithm is the poll step where the success or failure of this algorithm heavily relies on the set of polling directions. If the set of polling directions is properly chosen, the algorithm converges to a local minimizer. A good poll set must follow three conditions. First, it should have only a few vectors. Second, every half-space of $\mathbb{R}^d$ should be covered by the vectors in the poll set. Third, in terms of size, all vectors in the poll set should be similar [4]. When the poll set is a positive basis (Definition 2.6), the first two conditions are satisfied. So it is logical to create the poll set as a positive basis. There are many techniques to create a positive basis, ranging from minimal to maximal positive basis. Using the positive basis ensures that there exist at least one vector in the set that is a descent direction [4, Theorem 6.5]. Algorithm 1 shows the general framework for GPS algorithm.

Algorithm 1: The generalized pattern search (general framework)

Input: $f : \mathbb{R}^d \rightarrow \mathbb{R}$, starting point x_0 .

Step 0-initialization:

$\delta_0 \in (0, \infty)$	an initial mesh size parameter,
$D = GZ$	a positive spanning matrix,
$Z \in \mathbb{Z}^{d \times p}$	a $d \times p$ matrix of integer numbers,
$\tau_{inc} > 1$	a mesh size increasing parameter,
$\tau_{dec} \in (0, 1)$	a mesh size decreasing parameter,
$k = 0$	an iteration counter,
$\epsilon_{stop} \in (0, \infty)$	a stopping tolerance.

Step 1-search step: construct E_k a finite subset of the mesh M_k .

if $f(x) < f(x_k)$ for some $x \in E_k$, then

$x_{k+1} = x$, $\delta_{k+1} = \tau_{inc} \delta_k$, $k = k + 1$ and go to Step 3.

else go to Step 2.

Step 2-poll: select a positive spanning set ι_k from columns of D:

if $f(x) < f(x_k)$ for some $x \in P_k = \{x_k + \delta_k d : d \in \iota_k\}$, then

$x_{k+1} = x$, $\delta_{k+1} = \tau_{inc} \delta_k$.

else $x_{k+1} = x_k$, $\delta_{k+1} = \tau_{dec} \delta_k$.

Step 3-stop:

if $\delta_k \leq \epsilon_{stop}$, then stop.

else $k = k + 1$ and go to Step 1.

The detailed description of each step of the general version of GPS algorithm is as follows:

2.1. Generalized pattern search algorithms with adaptive precision control

In **Step 0**, all the parameters of the algorithm are initialized, including the initial mesh size δ_0 , the mesh size increasing and decreasing parameters τ_{inc} , τ_{dec} and the stopping condition ϵ_{stop} .

In **Step 1**, the algorithm performs the search process on the points created on E_k . More specifically, at each iteration, the algorithm generates E_k as the subset of the Mesh M_k . Then the algorithm explores the points at E_k , trying to find a better point. The search process can be carried out using many techniques, ranging from heuristic algorithms to even another GPS algorithm inside of the main GPS algorithm [28]. As long as the points created lie on the E_k , the algorithm converges to a stationary point [4, p. 120]. It is even possible not to employ the search step, and the algorithm still converges [28]. The main objective of the search process is to help the algorithm to converge faster to a local minimizer.

In **Step 2**, the polling step is performed. At first, a positive spanning set (ι_k) from columns of the polling points is constructed. If there is a point in ι_k , which is better than the current point, it takes the place of the current point and the mesh size will be expanded by the factor of τ_{inc} . Otherwise, the size of the mesh is decreased by the factor of τ_{dec} .

In **Step 3**, if the size of the mesh is small enough, the algorithm is stopped. Otherwise, the algorithm goes to Step 1.

Using the proper polling set (positive basis), the generalized pattern search was shown to converge to a stationary point [35].

2.1.3 Generalized pattern search: an adaptive precision control version

In 2006, Polak and Wetter [28] proposed a new version of GPS algorithm that adaptively controls the error. The algorithm is a multi-fidelity GPS algorithm where the precision of the objective function is adaptively adjusted. In particular, the algorithm starts off with the objective function with low accuracy (low fidelity model). As the algorithm converges, it employs a smaller size for the mesh size parameter, and higher fidelity model is used. It was proven that when a proper poll set (positive basis) is selected and the error decreases/increases similar to the mesh size parameter, the algorithm converges to a stationary point [28].

The algorithm considers three conditions that make it different from the general version of GPS algorithm. First, it assumes that the objective function is not precise and contains error. Second, the error of objective function can be controlled. Third, the more computation time spent on the objective function, the more accuracy is gained. The main objective

2.1. Generalized pattern search algorithms with adaptive precision control

of this algorithm is to adaptively control the error so that the algorithm converges to the stationary point. The pseudo-code for the GPS algorithm with adaptive precision control is presented in Algorithm 2.

Algorithm 2: The GPS with adaptive precision control (GPS 1)

Input: $f : \mathbb{R}^d \rightarrow \mathbb{R}$, starting point x_0 .

Step 0-initialization:

$\epsilon_0 \in (0, 1)$	the initial error value,
$\delta_0 \in (0, \infty)$	an initial mesh size parameter,
$D = GZ$	a positive spanning matrix,
$Z \in \mathbb{Z}^{d \times p}$	a $d \times p$ matrix of integer numbers,
$\tau_{inc} > 1$	a mesh size increasing parameter,
$\tau_{dec} \in (0, 1)$	a mesh size decreasing parameter,
$k = 0$	an iteration counter,
$\epsilon_{stop} \in (0, \infty)$	a stopping tolerance,
$E \in \mathbb{N}$	an error control parameter.

Step 1-global search: construct the global search set E_k as a finite subset of the mesh M_k .

if $\tilde{f}(x, \epsilon_k) - \tilde{f}(x_k, \epsilon_k) < -\zeta \epsilon_k$ for at least one $x \in E_k$, then

$x_{k+1} = x, \delta_{k+1} = \delta_k, \epsilon_{k+1} = \epsilon_k$ and go to Step 3.

else go to Step 2.

Step 2-poll: select a positive spanning set $\iota_k \subseteq D$:

if $\tilde{f}(x, \epsilon_k) - \tilde{f}(x_k, \epsilon_k) < -\zeta \epsilon_k$ for at least one $x \in \iota_k$, then

$x_{k+1} = x, \delta_{k+1} = \delta_k, \epsilon_{k+1} = \epsilon_k$.

else

$x_{k+1} = x_k, \delta_{k+1} = \tau_{dec} \delta_k$.

if E sequential poll failures have occurred, then

$\epsilon_{k+1} = \tau_{dec} \epsilon_k$.

else

$\epsilon_{k+1} = \epsilon_k$.

Step 3-stop:

if $\delta_k \leq \epsilon_{stop}$, then stop.

else $k = k + 1$ and go to Step 1.

The algorithm is similar to the general version of GPS except that it has a strategy to adaptively control the error, and it uses the surrogate model instead of the true objective to evaluate each point. To control the error, the algorithm defines a new parameter called ϵ_k that determines the precision

of the objective function.

One other important difference between Algorithm 2 and the general version of GPS is that Algorithm 2 does not expand δ_{k+1} at successful iterations.

The description of the algorithm is as follows.

In **Step 0**, the iteration counter k and the mesh size adjustment parameter δ_0 are initialized. Moreover, the initial error of the objective function, ϵ_0 , is initialized with a large value. The reason behind choosing a large value for the error at initial iterations is that we assume that the initial points are far away from a local optimizer at initial iterations.

In **Step 1**, similar to the general framework of GPS, the algorithm carries out the search process on E_k , which is a subset of the Mesh M_k . In contrast to the general version of the GPS algorithm, this algorithm evaluates the points in E_k using the surrogate function. Note that in this study, we do not employ the search step as it has not shown positive results in the outcome of our preliminary studies.

In **Step 2**, we first select a subset of the D as our local search set. Then we do the poll step on the selected local set (ι_k). To select ι_k , Pollak and Wetter [28] suggested using Canonical Maximal Positive Basis. In this study, we employ this technique to generate polling points at each iteration. If we can find at least one point in ι_k , which is significantly better than the incumbent solution, then we replace the incumbent solution with that point. The reason that we use the word ‘significantly better’ is that since the difference between the incumbent solution and x should be less than $-\zeta \cdot \epsilon_k$ and $\zeta \geq 0$, x should be better than the current solution by a factor of ζ . If we cannot find any point, we update δ_{k+1} by τ_{dec} . Furthermore, if the algorithm cannot find any better points at E sequential iterations, the error ϵ_{k+1} will be updated by τ_{dec} . It is worth noting that both δ_k and ϵ_k are updated when a poll failure occurs. So, by linking the δ_k and ϵ_k updates, the stopping only occurs when the error is small. It was shown that using this idea, the algorithm converges to a local minimizer [28].

In **Step 3**, similar to the general version of GPS, the algorithm terminates if δ_k is small enough. Otherwise, it goes to Step 1.

In [28], the authors showed that the proposed GPS algorithm converges to a stationary point as long as the cost function f is locally Lipschitz. They have shown that even if f is approximated by a family of discontinuous surrogate functions, the algorithm will eventually converge to stationary points of f .

2.2 A trust region algorithm for unconstrained optimization problems with controlled error

Model-based DFOs are different from the GPS algorithm; they build a surrogate model function from the sample points and its objective function values. It is worth indicating that model-based algorithms are good if the objective function is differentiable and the model created is easy to minimize [4]. There are two different groups of model-based algorithms: gradient-descent and trust region algorithms. In contrast to gradient-descent algorithms where the model is employed to find the descent direction of the objective function, in trust region ones the model is minimized approximately considering trust region constraints [4]. A general framework for the trust region algorithm is presented in Algorithm 3.

In **Step 0**, the parameters of the algorithm including Δ_0 , k , N_l , N_u are initialized. The trust region size Δ is one of the most important parameters of a trust region algorithm that has a great impact on the performance of the algorithm. In addition, we need to set the minimum (N_l) and maximum (N_u) size parameters for the sample set. These parameters control how many sample points are in the sample set. The minimum size of the sample set usually is set to $d + 1$ where d is the size of the optimization problem and the maximum size of the sample set (N_u) can be set to a large integer number.

The next sub-step is to construct the sample set (X). There are many different ways to generate a sample set. It can be created randomly or using a positive basis [4]. For example, Takaki [31] used a random method to generate sample points. In [28], authors suggested constructing sample points using the canonical maximal positive basis (Definition 2.7).

In **Step 1**, we create a surrogate model that approximates the behavior of the objective function. There are numerous ways to create a surrogate of an objective function. It can be created using a linear or quadratic interpolation or regression models. Readers are referred to [6] to find more information.

In **Step 2**, the approximate minimum of the model within the trust region is found. This procedure is usually called trust region sub-problem and has been discussed extensively in many papers [6, 12, 32].

In **Step 3**, if the model is successful (meaning that the approximate minimum x_k^* found in Step 2 is better than the current best point x_k), the trust region will be expanded. Otherwise, it will shrink.

In **Step 4**, if the sample set has enough space, the model minimum x_k^*

Algorithm 3: The trust region algorithm (general framework)

Step 0 : initialization - choose trust region parameters

$k = 0$ an iteration counter,
 $\Delta_0 \in (0, \infty)$ an initial trust region size parameter,
 $0 < N_l \leq N_u$ the minimum and maximum size of sample set,
 construct sample set $X = \{x_1, x_2, \dots, x_l\}$ where $x_l \in \mathbb{R}^d$,
 set $x_0 \in \arg \min_{x \in X} (f(x))$.

Step 1: construct the model $m_k(x)$ using the sample set X

Step 2: find the approximate minimum of the model

compute $x_{k^*} \in \arg \min_{x_k \in B_k} m_k(x_k)$.

Step 3: update the trust region radius

if the model $m_k(x)$ is successful, then

$$\Delta_{k+1} > \Delta_k.$$

else

$$\Delta_{k+1} < \Delta_k.$$

Step 4: update the sample set

if $|X| < N_u$, then

insert x_{k^*} into X .

else

replace x_{k^*} with $\arg \max_{x \in X} (f(x))$.

set $x_{k+1} \in \arg \min_{x_j \in X} (f(x_j))$.

remove any $x \in X$ such that $\|x - x_{k+1}\| > \Delta_k$.

while($|X| < N_l$)

generate a point inside of the trust region.

Step 5: stop

if Δ_k or the gradient of the model is small, then stop.

else $k = k + 1$ and go to Step 1.

is inserted to the sample set X . Otherwise, the model minimum x_{k^*} takes the place of the worst point in X . Then we need to update the trust region accordingly. To do that, we first find the minimum of the sample set and set it to x_{k+1} . And then all points in S that are outside of the trust region will be removed from the sample set. If the number of remaining points in X is less than N_l , new points are generated until the number of points in X is more than or equal to N_l . The new points can be generated using random

sampling [31], Latin Hypercube sampling [19] or using positive basis [28].

Algorithm 3 is a general framework for any type of trust region algorithms. However, the framework does not consider the error of the objective function and how the algorithm handles it. Here, we study the trust region algorithm that addresses unconstrained optimization problems with controlled error [31].

The pseudo-code for this algorithm is presented in Algorithm 4.

Algorithm 4 is similar to Algorithm 3 in several aspects. However, there are several major differences. First, the algorithm considers point-wise error ϵ_k in the objective function where ϵ_k is adjusted according to the trust region size parameter Δ_k . Second, the algorithm considers a penalty parameter C_k for every point in the sample set S . This penalty parameter C_k comes into play when the surrogate model is constructed from the sample set. In particular, the sample points with better performance have fewer penalty values, thereby having more impact on the construction of the surrogate model. Third, the algorithm does not only consider x_l in S to construct surrogate model $m_k(\cdot)$ but also penalty and error parameters. Finally, for every point x_k in the sample set we store three corresponding values $(x_k, y_k, \epsilon_k, C_k)$.

In **Step 0**, we initialize the parameters of the algorithm where model quality control parameters (η_0, η_1) are set to $(0.1, 0.75)$, the trust region increasing and decreasing parameters $\gamma_{inc}, \gamma_{dec}$ to 0.5 and 2, the iteration counter k to 0, the minimum size of the sample set N_l to $d+1$ and the initial solution x_0 to the best point in the sample set. The initial error parameter ϵ_0 is set to 0.1. Other parameters are set similar to Algorithm 3.

The next sub-step is to construct the sample set S . Similar to the general framework of the trust region algorithm, the sample points in S can be created using a random method [31], Latin Hypercube sampling [19] or a positive basis. However, in contrast to the general version of the trust-region algorithm that we only store x value for each point, in the trust region algorithm with controlled error, we store the error and penalty values for each point. Here, to construct the sample points in S , we choose the canonical minimal positive basis (Definition 2.8) as our preliminary studies have shown that it would be a good choice. It is worth noting that for every point in S , we initialize ϵ_l and C_l with ϵ_0 and C_0 , respectively.

In **Step 1**, we build a surrogate model using the support vector regression model. Since we assume that the objective function contains error, the surrogate model considers pointwise error (ϵ_k) in its design. Moreover, in order to penalize points that are farther away from the minimum of the trust region, it considers a penalty parameter C that adaptively adjusts the effect of every point in the sample set. The penalty parameter is updated as in

Algorithm 4: The proposed trust region algorithm

Step 0 : Initialization - choose trust region parameters

$0 \leq \eta_0 \leq \eta_1 < 1$ model quality control parameters,
 $\epsilon_m \in (0, 1)$ a model control parameter,
 $\gamma_{dec} \in (0, 1)$ a trust region decreasing parameter,
 $\gamma_{inc} > 1$ a trust region increasing parameter,
 ϵ_0 the initial error parameter,
 C_0 the initial penalty parameter,
construct sample set $S = \{s_1, s_2, \dots, s_l\}$ where $s_l = (x_l, y_l, e_l, c_l)$,
set $k, \Delta_0, N_l, N_u, x_0$ similar to Algorithm 3.

Step 1: construct the model $m_k(x)$ from sample set S

if $(\max(\|\nabla m_k(x)\|, \lambda_{\min}(\nabla^2 m_k(x))) < \epsilon_m)$, then
label the model as unsuccessful, $\rho_k = 1$ and got to Step 4.

Step 2: find the approximate minimum of the model

compute $x_{k^*} \in \arg \min_{x_k \in B_k} m_k(x)$,
evaluate $y_{k^*} = \tilde{f}(x_{k^*}, \epsilon_{k^*})$ with $\epsilon_{k^*} = \epsilon_k$.

Step 3: evaluate the quality of the model

compute the ratio: $\rho_k = \frac{y_k - y_{k^*}}{m_k(x_k) - m_k(x_{k^*})}$.

Step 4: update the trust region radius

successful step: if $\rho_k \geq \eta_1$, then $\Delta_{k+1} = \gamma_{inc} \Delta_k$.
unsuccessful step: if $\rho_k \leq \eta_0$, then $\Delta_{k+1} = \gamma_{dec} \Delta_k$.
else $\Delta_{k+1} = \Delta_k$.

Step 5: adjust the accuracy ϵ_k and penalty parameter C_k

$\epsilon_{k+1} = o(\Delta_k)$, $C_{k+1} = C_0/\epsilon_{k+1}$.

Step 6: find the best solution

$x_{k+1} \in \arg \min_{x_j \in (S \cup x_k^*)} (\tilde{f}(x_j, \epsilon_k))$

Step 7: update process

if $|S| < N_u$, then
insert $(x_{k^*}, y_{k^*}, \epsilon_{k^*}, C_k)$ into S .
else
replace $(x_{k^*}, y_{k^*}, \epsilon_{k^*}, C_k)$ with the worst point in S .
remove any $x \in X$ such that $\|x - x_{k+1}\| > \Delta_k$.
while $(|X| < N_l)$
generate a point inside of the trust region.
update the penalty parameter $C_l \quad \forall C_l \in S$.

Step 8: stop similar to Algorithm 3.

Equation (2.2).

The detailed description about how the model is built appears in Section 2.2.1.

After building the model, we need to check whether the model is successful or not. If $\max(\|\nabla m_k(x)\|, \lambda_{\min}(\nabla^2 m_k(x))) < \epsilon_m$, we label the model as unsuccessful and go to Step 4. In other words, if the model is almost a constant function, the algorithm does not accept the model and define it as an unsuccessful one and shrink the trust region radius in Step 4. In the formula, $\lambda_{\min}$ is the minimum eigenvalue of the Hessian of the model.

In **Step 2**, the approximate minimum of the model is found. Since we want to find the approximate minimum of the model within the hypersphere without any external constraints, we can use several different methods such as dogleg, two-dimensional subspace, Steihaug's methods [31] or truncated conjugate gradient method [6]. In this study, we use the truncated conjugate gradient as it provides the approximate minimum of the model quickly with good accuracy. The discussion about it appears in Section 2.2.2.

In **Step 3**, the quality of the model is evaluated. To compute the quality of the model, we use the objective function value with ϵ_k (the error value at the current iteration).

In **Step 4**, if the model is successful (the quality of the model is greater than η_1), the current trust region radius Δ_{k+1} will be expanded. On the other hand, if the quality is not good enough, the trust region radius will shrink.

In **Step 5**, the error is updated based on the trust region radius, Δ_k . Because we do not have the derivative of the objective function, we cannot find the distance between a point x to the solution. It was shown that [7] when x_k is generated by a DFO algorithm with exact objective function, $k \rightarrow \infty$, $\Delta_k \rightarrow 0$ then $\|\nabla f(x_k)\| \rightarrow 0$ too. For this reason, it would be a good idea to use Δ_k (the trust region radius) for the measure. For updating the penalty parameter, C_0/ϵ_{k+1} is used. This formula makes sure that the penalty parameter becomes larger for x_{k^*} with higher accuracy and vice versa.

In **Step 6**, the best solution among the sample points in S and the approximate minimum of the model x_k^* is found and is assigned to x_{k+1} .

In **Step 7** If the size of the sample set is less than the maximum size N_u , x_k^* and all its corresponding values (y_{k^*} , ϵ_{k^*} , C_k) are inserted to the sample set S . If not, then x_k^* and its corresponding values will take the place of the worst point in S .

For updating the sample set, we need to first measure the distance between all points in the sample set to x_{k+1} (the best point). Then if the

number of sample points inside the new trust region is less than $d + 1$, then the farthest points from x_{k+1} will be replaced with newly generated sample points. Note that the new sample points must be inside of the new trust regions to be accepted.

When N is less than N_u , it is probable that some points would be outside of the trust region thereby deteriorating the quality of the model. To surmount this, we reduce the penalty of these points, making them have less effect on the model by reducing C_i . So for any C_i corresponding to $x_i \in S$ and $\|x_i - x_{k+1}\| > \Delta_{k+1}$, we update

$$C_i = 10^{-\frac{\|x_i - x_{k+1}\|}{\Delta_k}} C_0.$$

2.2.1 Creating a surrogate model using support vector regression model

In this section, we study the construction of a surrogate model using a support vector regression model [31]. Suppose $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is the true objective function. Our goal here is to find a quadratic surrogate model that approximates the behavior of f . Here, we define the model $m : \mathbb{R}^d \rightarrow \mathbb{R}$ as follows,

$$m(x) = w^\top \phi(x),$$

where $w \in \mathbb{R}^N$ is a coefficient vector, $N = (1/2)(d + 1)(d + 2)$, d is the dimension of the problem and $\phi(x)$ is a mapping function defined as,

$$\phi(x) = [1, x_1, x_2, \dots, x_n, x_1^2, x_1x_2, \dots, x_{n-1}x_n, x_n^2]^\top \in \mathbb{R}^N. \quad (2.10)$$

The next step is to calculate w as follows,

$$w = P^T(\alpha^+ - \alpha^-), \quad (2.11)$$

where P is computed as,

$$P = [\phi(x_1), \phi(x_2), \dots, \phi(x_l)] \in \mathbb{R}^{l \times N}$$

In (2.11), α^+ and α^- are computed by finding an approximate minimizer of the following quadratic programming problem,

$$\arg \min_{\alpha^+, \alpha^-} \frac{1}{2} \begin{pmatrix} \alpha^+ \\ \alpha^- \end{pmatrix}^\top \begin{pmatrix} K & -K \\ -K & K \end{pmatrix} \begin{pmatrix} \alpha^+ \\ \alpha^- \end{pmatrix} + \begin{pmatrix} \epsilon - y \\ \epsilon + y \end{pmatrix}^\top \begin{pmatrix} \alpha^+ \\ \alpha^- \end{pmatrix}, \quad (2.12)$$

subject to $0 \leq \alpha^+, \alpha^- \leq C$ and $\sum_{i=1}^l (\alpha_i^+ - \alpha_i^-) = 0$.

In (2.12), K is a matrix defined as: $K = PP^T$, y is a vector of the function values of the sample points ($y_i = \tilde{f}(x_i, \epsilon_i)$) where $i = 1, \dots, l$, $\epsilon = [\epsilon_1, \epsilon_2, \dots, \epsilon_l]$ and $C = [C_1, C_2, \dots, C_l]$.

2.2.2 Find the approximate minimum of the model

In this section, we briefly overview the problem of the approximating minimum of a function inside an hypersphere. Suppose $m_k(x)$ is the model function for the iteration k . The goal is to find the approximate minimum x_k^* of the quadratic model function $m_k(x)$ within trust region B_k such that x_k^* is defined as,

$$x_k^* = \arg \min_{x \in B_k} m_k(x), \quad (2.13)$$

where B_k is a hypersphere that is defined as, $B_k = \{x \in \mathbb{R}^d : \|x - x_k\| \leq \Delta_k\}$ and Δ_k is the current trust region radius. This problem is called trust region sub-problem and has been addressed by many research papers [6, 12, 32]. Since the problem is differentiable ($m \in C^2$), many different algorithms such as dogleg, Steihaug's approach, the truncated conjugate method can be good choices. Since truncated conjugate gradient method [6] is a fast and an accurate technique to solve this problem, we use it in all our simulations.¹

¹For the truncated conjugate gradient method, I use the implementation provided by Dr. Yves Lucet, which follows [6, Section 7.5.1].

Chapter 3

Multi-fidelity algorithms for solving the horizontal alignment in road design

Building a high precision algorithm with low computational cost has always been an interesting topic for researchers. For this purpose, multi-fidelity algorithms that employ different levels of accuracy have come into play. The main idea of these algorithms is to spend more computation time on the areas of the search space that are close enough to local minimizers. The general framework for these algorithms is as follows. First, the multi-fidelity algorithm starts with a surrogate model with low accuracy. As the algorithm converges to better points in the search space, the surrogate model gradually increases its accuracy. Finally, the algorithm stops as it reaches a local minimizer.

In this chapter, we first formulate the horizontal optimization problem in road design, and then discuss how multi-fidelity algorithms are applied to the problem.

3.1 Terminology

To solve the horizontal alignment, we need to first find an optimal curve in a given corridor. The corridor is usually specified by civil engineers, and it shows the feasible region for the horizontal and vertical alignments. In every given corridor, there are two points at the very start and at the very end of a road called, starting and ending points. The center-line alignment that connects these two points is called baseline. Figure 3.1 shows the satellite view of a road that contains the corridor, the baseline and the starting and ending points.

The baseline is often discretized into some data points called base data points. For each base data point, there are some horizontal and vertical points that are called offset data points. If the points associated with a

3.1. Terminology

Figure 3.1: A satellite view of a road that represents a corridor, a baseline and the starting and ending points.

base data point are generated in the horizontal fashion, they are dubbed as horizontal offset points. In a similar vein, if they are the displacement of a base data point in the vertical fashion, they are called vertical offset points. Another similar concept is the station where it is defined as base data points along with all its associated vertical and horizontal offset points. Figure 3.2 shows a road with stations and their corresponding vertical and horizontal

3.1. Terminology

offsets as well as the baseline.²

Figure 3.2: A sample road with stations and their corresponding vertical and horizontal offset points [23]

Another important concept is the intersection point. Since optimizing the horizontal alignment for every station is extremely time-consuming, specifically for long roads, it was shown [23] that using circular curve and tangential line segments instead of stations to optimize the HA problem is a faster approach. Each segment contains some consecutive stations, and the number of segments is usually dependent on the size and complexity of a road. Figure 3.3 shows a road with four segments and 14 stations where a segment is represented by a red rectangular box and a station with some consecutive blue circles. In Figure 3.3, instead of optimizing the road with 14 stations, we can use two tangential segments and two circular curvature segments. Each segment often contains at least one Intersection Points (IPs). By moving these IPs, a Horizontal Alignment (HA) algorithm can find the optimal horizontal alignment in a corridor. Figure 3.4 shows a corridor with five IPs represented by blue stars.

²The figure is taken from [23] with permission from the author

Figure 3.3: A road with four segments and 14 stations

Figure 3.4: A corridor with five IPs represented by blue stars

3.2 Objective function: vertical alignment problem

The objective function for the horizontal alignment problem is the Vertical Alignment (VA) problem for a fixed horizontal alignment. The VA problem is about finding a series of parabolic curves and vertical tangential lines for a particular horizontal alignment. The main objective of the VA

problem is to find the minimum of the earthwork costs while following all the safety design constraints. Here, we consider the VA problem as a black-box optimization problem where the input parameters are IPs, the station sets and the output is the optimized VA. Figure 3.5 shows the box diagram of the objective function of the VA problem. Readers are referred to [23,

Figure 3.5: The objective function

Chapter 3] to gain more information about how the VA problem is modeled.

It should be noted that if the horizontal alignment generated from the IP set (x_k) is outside of the corridor or any other safety design constraints are violated, the VA problem will return infinity as the output.

Note that the VA problem is a mixed-integer linear programming problem, and therefore it can be solved by a global solvers such as CPLEX [8].

3.3 Merging stations: a surrogate model for the vertical alignment problem

The backbone of a multi-fidelity algorithm is the surrogate model that approximates the true objective function. In road design, finding the optimal vertical alignment for a given horizontal alignment often takes minutes or even hours for a large and complex road. Moreover, for solving a horizontal problem, we need to find the vertical optimization problem many times. So it would be essential to find a surrogate for the vertical alignment optimization problem. In [14], Hare, Jaberipour and Lucet proposed two surrogate models to approximate the true objective function. Both surrogate models are based

3.3. A surrogate model for the VA problem

on the idea of the reduction of the size of mixed-integer program needed to tackle the vertical alignment optimization problem at the expense of decreasing the accuracy of the final earthwork cost.

The first surrogate method, uniform merging technique (we call it $\tilde{f}(\cdot, \epsilon)$), removes some stations from the station set in the hope of reducing the computation time. Assume there is a road with $n + 1$ stations and with the station set $S = \{s_0, s_1, \dots, s_n\}$. The method reduces the station set to

$$S_m = \{s_0, s_N, s_{2N}, \dots, s_{\lfloor \frac{n}{N} \rfloor N}\}, \quad (3.1)$$

where N varies between $N = 1$, for no reduction, to $N = n - 2$ for reducing the road to a single station.

The uniform merging method decreases the number of stations by a factor of $\lfloor \frac{n}{N} \rfloor + 1$.

Even though the method is simple, the experiments carried out on 40 roads have shown that it can decrease the solution time by a factor of 8 with a small loss of accuracy [14]. In this study, we will use this technique as a surrogate model for reducing the station set, and as a result, solving the vertical alignment optimization problem faster.

We discussed how the surrogate is created. But we did not mention how exactly the surrogate is related to the true objective function. Algorithm 5 shows how the surrogate model is utilized in the vertical optimization problem. In **Step 1**, we determine N by the input ϵ_k . To do this, we use

Algorithm 5: The merging station approach

Input:

$\tilde{f} : \mathbb{R}^d \times \mathbb{R}_+ \rightarrow \mathbb{R}_+$	the surrogate model,
ϵ_k	the error value at iteration k ,
x_k	the IP set at iteration k ,
S	the station set with no reduction.

Step 1: Determining N by ϵ_k using Table 3.1.

Step 2:

if S_m not exist for a particular N , then

merge station set S and create S_m by uniform merging method.

else

retrieve S_m for a particular N .

Step 3: Perform vertical alignment optimization on S_m for x_k .

Table 3.1 that was obtained from experiments carried out on 40 roads [14].

3.4. A general approach

Table 3.1: A mapping table from ϵ_k to N

Error (ϵ_k)	N
$\epsilon_k > 0.12$	20
$0.06 \leq \epsilon_k \leq 0.12$	10
$0.034 \leq \epsilon_k < 0.06$	6
$0.01 \leq \epsilon_k < 0.034$	4
$0 < \epsilon_k < 0.01$	2
$\epsilon_k == 0$	1

In **Step 2**, in order to avoid wasting computation time merging stations for a particular error, we save the merged station set (S_m), and its corresponding meta-data on disk after performing the uniform merging method. If S_m exists for a particular N , we retrieve it from the disk.

In **Step 3**, we perform the vertical alignment on S_m for the given IP set (x_k).

3.4 A general approach

In this section, we describe how HA algorithms are applied to solve the horizontal alignment problem. At first, the input data including the road data profile, road meta-data information along with IPs positions are provided. Using this information, the vertical alignment optimization finds the optimal vertical alignment for a given horizontal alignment. If the horizontal alignment (HA) is optimal, then we terminate the algorithm; otherwise, the HA algorithm continues to change the IPs until either an optimal HA is found or the computation time is exhausted. Figure 3.6 shows the flow chart of the general approach.

The multi-fidelity framework for solving the horizontal alignment problem is similar to the general version except that we need to first determine how close the incumbent solution is to the local minimizer. To find this we can use either the mesh size δ for the GPS algorithm or the trust region size Δ for the trust region algorithm. As the algorithm gets closer to a local minimizer, the mesh size or the trust region size becomes smaller and vice versa. Based on the premise of the closeness of the incumbent solution to a local minimizer, the algorithm decides to adopt a surrogate with high or low accuracy for the vertical optimization problem. Figure 3.7 shows the flowchart of the general multi-fidelity approach for solving the horizontal alignment optimization problem in road design.

Figure 3.6: The flowchart of a general approach for solving the horizontal alignment problem

3.5 A multi-fidelity GPS algorithm for the horizontal alignment optimization problem in road design

The GPS algorithm designed for solving the horizontal alignment without a multi-fidelity framework is similar to Algorithm 1 described in Section 2.1.3. So we do not discuss it here. However, the multi-fidelity version of the GPS algorithm is a little bit different from Algorithm 2. Here, for the surrogate model $\tilde{f}(\cdot, \epsilon)$ the uniform merging method is employed and for the objective function, the vertical alignment optimization is used. Furthermore, every point x in the local search set ι_k represents IPs for a given road.

Figure 3.8 demonstrates the trajectory plot for the multi-fidelity GPS algorithms (with and without surrogate) for Road 35 with one IP. In the trajectory plot, the contour lines represent the function value not the topography of the road. In Figure 3.8, the initial point is represented by the red solid dot and a polling point by a hollow blue circle.

Figure 3.7: The flowchart of the general multi-fidelity approach

As shown in Figure 3.8, both GPS algorithms (with and without surrogate) begins with large cost (\$33,389) and as they progress, they converge to better positions with lower cost ($\approx$ \$30,000).

3.6 A multi-fidelity trust region algorithm for the horizontal alignment optimization problem in road design

In this section, we study the multi-fidelity trust region algorithm for the HA problem in road design.

3.6. A multi-fidelity trust region algorithm for the HA problem

Figure 3.8: The trajectory plot of the GPS algorithm for Road 35

The algorithm is similar to Algorithm 4 except in Step 6. After evaluating the sample points with the surrogate function, we evaluate the result again with the true objective function before setting it to x_{k+1} . In particular, we first create and evaluate the sample set based on the surrogate function. And, when we want to pick x_0 , we evaluate the best point in the sample set using the true objective function. And at each iteration, after finding the approximate minimizer of the model, we evaluate it using the true objective function.

3.6. A multi-fidelity trust region algorithm for the HA problem

Figure 3.9: The trajectory plot of the trust region algorithm for Road 35

To see how the trust region algorithm progresses, Figure 3.9 shows the trajectory plot of two versions of the trust region algorithm for the HA problem for Road 35. In Figure 3.9, the initial point is represented by the red solid dot, and the approximate minimizer of the trust region (x_k^*) by the blue solid dot. As shown in Figure 3.9, both versions of the trust region algorithm start off with large vertical alignment costs, and as they progress, they converge to better points.

Chapter 4

Experimental results

In this chapter, we study the effect of the multi-fidelity framework for the two derivative-free optimization algorithms on 35 different roads. Then we compare the results with NOMAD [9] (Nonlinear Optimization by Mesh Adaptive Direct Search) as a standard solver.

For carrying out this experiment, we run the algorithms on three different workstation PCs where all the computers run on Windows 7 64 bits. The first computer uses Intel core i7 2.8 GHz CPU with 16 GB RAM. We use this computer to run the HA algorithms on small roads. The second computer is equipped with Intel Xeon 3.5 GHz CPU with 64GB RAM. Finally, the last workstation PC has the same configuration as the second computer. The last two workstation PCs are employed to run the algorithms on medium to large roads.

In terms of the framework and the programming environment, we designed and implemented an integrated framework that is based on MATLAB 2017 and Visual Studio 2013.

4.1 Road set

We gathered a set of 35 roads from our industrial partner (Softree Technical System Inc. abbreviated as Softree) ranging from small to very large roads. The information we gathered contains the initial alignment (X- and Y-coordinates and radius of IPs), the station set and cut/fill material information. Table 4.1 provides a summary of each road, including the number of IPs and the number of stations for each road.

As shown in Table 4.1, most of the roads contain 1 to 10 IPs. Since the radius of IPs are fixed, the number of IPs is exactly 1/2 the number of variables in the resulting HA problem, e.g, Road 1 with 2 IPs contains 4 variables.

4.2. The effect of the multi-fidelity framework on the GPS algorithm

Table 4.1: Roads tested on different workstation PC computers.

Computer 1		
#	# of IPs	# of Stations
15	13	203
17	2	228
18	2	234
21	8	523
22	2	470
23	3	97
24	3	95
31	17	367
34	9	941

Computer 2		
#	# of IPs	# of Stations
1	2	49
2	1	33
3	5	107
4	3	114
5	3	114
6	2	116
7	2	120
8	2	120
9	7	178
10	2	120
11	2	129
13	4	184
14	2	222
16	2	227
20	19	368
25	2	98
26	2	11
27	2	952
28	5	1,164
30	15	446
32	12	9,343

Computer 3		
#	# of IPs	# of Stations
12	8	303
19	17	252
25	2	98
29	2	2,288
33	19	1,168
35	1	24

4.2 The effect of the multi-fidelity framework on the GPS algorithm

In order to study the effect of the GPS algorithm, we carry out our experiments with the algorithm presented in Algorithm 2 where ϵ_0 is set to 0 for the GPS without multi-fidelity and set to 0.1 for the GPS with the multi-fidelity framework. Note that when ϵ_0 is set to 0, the GPS algorithm always evaluates the true objective function.

4.2.1 Parameter tuning

In order to make a fair comparison, we need to follow some rules [5]. First, we need to give the same number of function evaluations to both versions of the GPS algorithm. The number of function evaluations has a great impact on the performance of algorithms. The number of function evaluations is recommended to be dependent on the size of the problem as the size of the problem becomes larger, the problem becomes more difficult to solve and needs more computation time. However, it should also be a trade-off between the maximum time allowed to solve the optimization problem and the size of the HA problem. Based on our initial experiments, we set the maximum number of function evaluations for all problems and both versions of GPS as follows,

4.2. The effect of the multi-fidelity framework on the GPS algorithm

Table 4.2: Parameter values for GPS0 and GPS1 algorithms

Parameter	GPS0	GPS1
ϵ_0	0	0.1
E	0	$\{1, 2, 5, 10\}$
δ_0	$\{5, 2.5, 1.25\}$	$\{5, 2.5, 1.25\}$
τ_{dec}	$\{0.01, 0.1, 0.3, 0.5, 0.7, 0.9\}$	$\{0.01, 0.1, 0.3, 0.5, 0.7, 0.9\}$

$$F_{\text{eval}} = 100 \min(|IP|^2, 5 |IP|) \quad (4.1)$$

where F_{eval} is the maximum number of function evaluations and $|IP|$ is the size of IP set.

The next rule is to use the same initial IP set for all the algorithms. We use the initial IP set that is provided by Softree for all the algorithms.

When it comes to the maximum allowed computation time to solve the HA problem, we set it to 24 hours for solving each road and 1 hour for calling each function call.

Finally, we need to tune the parameters of both algorithms. The multi-fidelity GPS algorithm presented in Algorithm 2 has four major parameters, including the initial error value parameter ϵ_0 , the initial mesh size δ_0 , the mesh size decreasing parameter τ_{dec} and the error control policy. Both GPS and TRSVR algorithms use the same error control policy such that at each E unsuccessful iterations, the error will be decreased by a factor of τ_{dec} .

In the GPS algorithm, ϵ_0 is set to 0.1 according to what the original paper [28] suggested. The rest of parameters are tuned in this study. We tune δ_0 , τ_{dec} and E , for the multi-fidelity version and tune the δ_0 , τ_{dec} for the GPS without the multi-fidelity. Note that since for the GPS without the multi-fidelity ϵ_0 is set to 0, it does not matter what value the error decreasing parameter is. Table 4.2 shows the parameter values tested for each version of the GPS algorithm. Note that from now on, we call the GPS without the multi-fidelity framework “GPS0” and the GPS with the multi-fidelity framework “GPS1”.

As shown in Figure 4.2, for GPS0 we have 18 different configurations. We call the first configuration “config0” and the next one “config1” and so on.

In order to find out which configuration is the best for GPS0, we use the performance profile [24] that can easily tell you which solver or algorithm is the best over a set of problems. A performance profile visualizes the competence of an algorithm by comparing it with other algorithms in a set

4.2. The effect of the multi-fidelity framework on the GPS algorithm

of problems. In fact, when you have a set of algorithms and a set of problems, and you like to know the efficiency and robustness of the algorithms in one compact figure, you can use performance profile. In fact, the performance profile shows that the percentage of the problems that a solver has the best performance among all the other solvers. The algorithm that can solve the highest percentage of the problems at the starting point is called the most efficient algorithm, and the algorithm that can solve most of the problems in total is called the most robust one. To gain more information about how to compute the performance profile, readers are referred to [4].

We tune the parameters δ_0 and τ_{dec} on five small roads (Road 1, Road 2, Road 5, Road 8, Road 35). Figure 4.1 shows the performance profile of GPS0 for 18 different configurations on five small roads.

Figure 4.1: The performance profile of GPS0 algorithm for 18 different configuration on five small roads. The legend only shows the top three configurations.

As shown in Figure 4.1, when the tolerance τ is 10 percent, the fastest configurations are “config4” and “config10” and the most robust one is “config16” where it solved all of the 5 problems. On the other hand, when τ becomes 5 percent, “config4” and “config10” seem to work similarly. However, when we zoom in, “config10” seems to work faster. Therefore, we chose “config10” over “config4”, which means that we choose $\tau_{dec} = 0.5$, $\delta_0 = 5$ for GPS0 algorithm.

Figure 4.2 shows the performance profile of GPS1 algorithm for 72 dif-

4.2. The effect of the multi-fidelity framework on the GPS algorithm

ferent configurations on five small roads.

Figure 4.2: The performance profile of GPS1 algorithm for 72 different configuration on five small roads. The legend only shows the best configuration.

As shown in Figure 4.2, “config37” is the fastest and the most robust configuration for both tolerances. So we choose this configuration (config37= $\delta_0 = 5$, $E = 1$, $\tau_{dec} = 0.5$) for all of our simulations.

4.2.2 Experimental results on 35 roads

After setting the parameters, we carry out our experiments on both versions of the GPS algorithm on 35 different roads presented in Table 4.1. We record the computation time, the optimized VA cost and the memory for all of the roads. The detailed description of results for GPS0 and GPS1 algorithms is presented in Table A.1 and Table A.2, respectively.

Table 4.3 shows the summary results of the speed up and cost difference for both versions of the GPS algorithm on 35 roads.

Notice that the speed up factor is calculated by dividing the computation time of GPS1 algorithm by the computation time of GPS0 algorithm. If one of the algorithms fails to solve the problem, we do not calculate the speed up factor for that problem. Moreover, the cost difference is calculated by subtracting the optimized VA cost obtained by GPS0 algorithm from the one obtained by GPS1 algorithm and then dividing it by the optimized VA cost of the initial alignment.

4.2. The effect of the multi-fidelity framework on the GPS algorithm

Table 4.3: Speed up and cost difference for the GPS algorithm

Speed factor	# of Roads	Cost Difference(%)	# of Roads
< 0.7	1	< 0	3
0.7 to 0.9	1	0.0 to 1.0	17
0.9 to 1.1	1	1.0 to 2.0	4
1.1 to 2.0	7	2.0 to 5.0	1
2.0 to 3.0	9	> 5	0
> 3	6		All roads
	All roads	Average	0.41
Average	2.52		

As shown in Table 4.3, there is only one road for which the use of the multi-fidelity version of the GPS increases the computation time significantly. As shown in Table A.6, this road is Road 26, which has only 11 stations and 2 IPs. The reason behind this computation time increase is that the surrogate function may not provide a good approximation of the true objective function. From Table 4.3, we can also see that for 2 roads, GPS0 algorithm performs better than GPS1 algorithm in terms of computation time. However, for other 21 roads, GPS1 algorithm outperforms GPS0 algorithm when it comes to computation time. In average, using the multi-fidelity framework improves the speed of the GPS algorithm by a factor of 1.52 while keeping the cost difference at 0.41 percent.

Note that the average speed up is calculated by adding up the speed up for each road and then dividing the total speed up by all the roads tested. In a similar vein, to compute the average cost difference, we add up the cost difference between GPS0 and GPS1 algorithms, and then divide the total cost difference by the total number of roads.

If you add up the number of roads in either Table 4.3, you can see that it is 25. The reason is there are some roads that GPS1 or both algorithms fail to solve due to the function evaluation exhaustion or the time-out policy (see Table A.1 or Table A.2). Table 4.4 shows the summary results for two versions of the GPS algorithm for the roads they fail to solve.

As shown in Table 4.4, GPS1 solves 4 more problems than GPS0. For finding more details about the speed up and the cost difference of GPS0 and GPS1 for every single road, see Table A.6.

4.2. The effect of the multi-fidelity framework on the GPS algorithm

Table 4.4: Optimization failure due to Time or number of function evaluations (F_e) budget exhaustion for the GPS algorithm

Algorithm	GPS0 (# of Roads)	GPS1 (# of roads)
F_e exhaustion	0	0
Time-out	9	5
Total	9	5

Table 4.5: Categorization of roads based on the number of IPs

Category	# of IPs	# of Roads
C1	1-2	17
C2	3+	18

4.2.3 The effect of the multi-fidelity on the GPS algorithm for different classes of roads

In order to study the effect of the multi-fidelity framework on different types of roads, we categorize the 35 roads based on the number of stations and the number of IPs. As the number of stations increases, the computation time for solving the VA problem increases. In addition, the number of IPs affects the number of function evaluations an algorithm requires to solve the HA problem. As the number of IPs increases, the number of variables increases, resulting in requiring more function evaluations. Table 4.5 shows the proposed categorization of roads based on the number of IPs. As shown in Table 4.5, most roads in our dataset contain one or two IPs (2 to 4 variables) and the rest have 3 to 19 IPs.

Table 4.6 show the summary results for the speed up of GPS0 and GPS1 algorithms for two different categories of IPs. As shown in Table 4.6, the speed up of the GPS algorithm is 94 percent for Category C1 roads and 240 percent for Category C2 roads in the average case. From Table 4.6, we can realize when the size of IP set is small (Category C1 roads), the multi-fidelity framework helps the GPS algorithm by almost a factor of 2. However, as the the size of the IP set grows, the multi-fidelity framework helps the GPS algorithm more significantly. Furthermore, we can also see when the size of the HA problem is small (Category C1 roads), the multi-fidelity framework makes the GPS algorithm slower in 2 of the roads. But, when the size of the HA problem is large (Category C2 roads), the multi-fidelity does not help the algorithm in only 1 road, and there is no road that it causes the algorithm to have more computation time.

4.2. The effect of the multi-fidelity framework on the GPS algorithm

Table 4.6: Speed up for GPS algorithm for two categories of IPs

Speed factor	# of Roads (C1)	# of Roads (C2)
< 0.7	1	0
0.7 to 0.9	1	0
0.9 to 1.1	0	1
1.1 to 2	6	1
2 to 3	6	3
> 3	1	5
	C1	C2
Average	1.94	3.40

Table 4.7: Cost difference for GPS algorithm for two categories of IPs

Cost Difference(%)	# of Roads (C1)	# of Roads (C2)
< 0	2	0
0 to 1	11	6
1 to 2	2	1
2 to 5	0	0
> 5	0	0
	C1	C2
Average	0.17	0.78

Table 4.7 represents the summary results of cost difference for GPS0 and GPS1 algorithms for two different categories of IPs. As shown in Table 4.7, for Category C1 roads, the cost difference between GPS0 and GPS1 algorithms is 0.17 percent in the average case, and there is no road that the cost difference is actually not within the acceptable tolerance range (0-5 percent range). Furthermore, for Category C2 roads, the cost difference between GPS0 and GPS1 algorithms is 0.78 in the average case. Interestingly, from Table 4.6 and Table 4.7, we can realize as the number of IPs increases, the speed up of GPS algorithm increases and the cost difference does not increase significantly in the average case.

We now study the effect of the multi-fidelity paradigm on the GPS algorithm for different sizes of the station set. Table 4.8 shows the categorization of 35 roads based on the number of stations. As shown in Table 4.8, most of the roads belong to Group G2 roads where the number of stations is between 100 to 250 stations.

Table 4.9 summarizes the speed up of GPS0 and GPS1 algorithms for different groups of stations. As shown in Table 4.9, in the average case,

4.2. The effect of the multi-fidelity framework on the GPS algorithm

Table 4.8: Categorization of roads based on the number of stations

Group	# of Stations	# of Roads
G1	1-100	7
G2	101-250	15
G3	251+	13

Table 4.9: Speed up for the GPS algorithm for different groups of stations

Speed factor	# of Roads (G1)	# of Roads (G2)	# of Roads (G3)
< 0.7	1	0	0
0.7 to 0.9	1	0	0
0.9 to 1.1	0	0	1
1.1 to 2	1	6	0
2 to 3	2	5	2
> 3	2	4	0
	G1	G2	G3
Average	3.03	2.38	2.10

the multi-fidelity framework improves the speed of the GPS algorithm by a factor of 3, 2.38 and 2.10, respectively. As the number of stations increases, the speed up of the GPS algorithm decreases in the average case. However, the number of roads in Groups G1, Group G2 and Group G3 are different, and maybe by having more roads in Group G1, the results would be different.

Table 4.10 presents the summary results of cost difference between two versions of the GPS algorithm for different groups of stations. As shown in Table 4.10, the cost difference between GPS0 and GPS1 algorithms is almost 2 percent in the average case for Group 1 roads while 0.17 and 0.41 percent, respectively, for Group G2 and Group G3 roads.

From Table 4.9 and Table 4.10, we can see that by increasing the number of stations, the multi-fidelity framework reduces the computation time while the error is in an acceptable range, similar to what we saw in Table 4.6 and Table 4.7.

Even though the results of the GPS algorithm for both Group G2 and Group G3 roads are similar (in both groups, we have more than 100 percent speed up improvement, and less than 1 percent cost difference), the GPS algorithm could only solve 3 out of 13 for Group G3 roads while 15 out of 15 for Group G2 roads, respectively.

4.3. The effect of the multi-fidelity framework on trust region algorithm

Table 4.10: Cost difference for the GPS algorithm for different groups of stations

Cost Difference	# of Roads (G1)	# of Roads (G2)	# of Roads (G3)
< 0	2	1	0
0 to 1	3	11	3
1 to 2	1	3	0
2 to 5	1	0	0
> 5	0	0	0
	G1	G2	G3
Average	0.42	0.17	0.27

In order to see what the final results look like, Figure 4.3 shows the optimal horizontal alignments obtained by both versions of the GPS algorithm for Road 1. Note that for both GPS0 and GPS1 algorithms, the best alignment is obtained when the horizontal alignment touches the corridor. Since the corridor is given from the input (our industrial partner), it cannot be expanded to get better results.

4.3 The effect of the multi-fidelity framework on the trust region algorithm

In this section, we study the effect of the multi-fidelity paradigm on the Trust Region (TR) algorithms. To do so, we employ the algorithm presented in Algorithm 4 for the TR algorithm with the multi-fidelity framework. In addition, for the TR algorithm without multi-fidelity framework, we set ϵ_0 to zero, making the error value disappear for the whole optimization process. We call the former Trust Region Support Vector Regression-based algorithm with multi-fidelity (TRSVR1) and the latter Trust Region Support Vector Regression-based algorithm without multi-fidelity (TRSVR0).

For the maximum of number of function evaluation, the maximum time allowed to solve the HA problem and the maximum time allowed for each function call, we follow the same rules as Section 4.2.1.

4.3.1 Parameter tuning

There are many parameters in the TR algorithm. In this section, in order to make a fair comparison, we tune the parameters that have a major

4.3. The effect of the multi-fidelity framework on trust region algorithm

Figure 4.3: The horizontal alignment carried out by GPS0 and GPS1 algorithms for Road 1

impact on the performance of the TR algorithm. The parameters we tune are the initial trust region size Δ_0 and the trust region decreasing parameter γ_{dec} for TRSVR0 algorithm, and Δ_0 , γ_{dec} and the error control parameter E for TRSVR1 algorithm. The final trust region size is set to 0.01 for both versions of the TR algorithms. The rest of parameters, including $(\gamma_{inc}, \eta_0, \eta_1, \epsilon_m, C_0, N_l \text{ and } N_u)$ are set according to [31]. Table 4.11 shows the parameters values considered in the tuning process. We tune the parameters on 5 small roads following the same process as Section 4.2.1.

Figure 4.4 and Figure 4.5 show the performance profile for the different configurations for both TRSVR0 and TRSVR1 algorithms, respectively. As shown in Figure 4.4, for TRSVR0 algorithm for both 10 and 5 percent tolerance, the “config1” ($\Delta_0 = 5$, $\gamma_{dec} = 0.1$) is the fastest and the most robust configuration. Therefore, “config1” is chosen for TRSVR0 algorithm for all the simulation. As shown in Figure 4.5, for TRSVR1 algorithm

4.3. The effect of the multi-fidelity framework on trust region algorithm

Table 4.11: Parameter values for TRSVR0 and TRSVR1 algorithms

Parameter	TRSVR0	TRSVR1
ϵ_0	0	0.1
E	0	$\{1, 2, 5, 10\}$
Δ_0	$\{5, 2.5, 1.25\}$	$\{5, 2.5, 1.25\}$
γ_{dec}	$\{0.1, 0.3, 0.5, 0.7, 0.9\}$	$\{0.1, 0.3, 0.5, 0.7, 0.9\}$

Figure 4.4: The performance profile of TRSVR0 algorithm for 18 different configuration on five small roads. The legend only shows the best configuration.

for both 10 and 5 percent tolerance, the “config1” is the most robust and the fastest among all configurations. Therefore, for TRSVR1 algorithm we choose “config1” ($E = 1$, $\Delta_0 = 5$, $\gamma_{dec} = 0.1$) for all the simulations.

4.3.2 Experimental results on 35 roads

After tuning the parameters, we carry out an experiment on 35 roads to study the effect of the multi-fidelity framework on the TR algorithm. In order to determine how much the effect of the multi-fidelity is, we collect the time and the cost of the horizontal alignment for each algorithm.

Table 4.12 summarizes the results of speed up and cost difference of two versions of the TRSVR algorithm for the 35 roads presented in Table 4.1.

As shown in Table 4.12, the multi-fidelity framework improves the speed of the TR algorithm by 31 percent while the cost difference is 0.40 percent

4.3. The effect of the multi-fidelity framework on trust region algorithm

Figure 4.5: The performance profile of TRSVR1 algorithm for 60 different configuration on five small roads. The legend only shows the best configuration.

in the average case. In particular, the multi-fidelity only decreases the speed of the algorithm only in 2 road, and in 6 roads it does not change the speed of the algorithm, and in 26 roads it improves the speed of the algorithm.

4.3.3 The effect of the multi-fidelity on the TR algorithm for different classes of roads

In order to show how much the number of IPs and the number of stations have effect on the performance of the multi-fidelity approach for the TR

Table 4.12: Speed up and Cost difference for the TRSVR algorithm

Speed factor	# of Roads	Cost Difference(%)	# of Roads
< 0.7	1	< 0	2
0.7 to 0.9	1	0.0 to 1.0	27
0.9 to 1.1	6	1.0 to 2.0	3
1.1 to 2.0	26	2.0 to 5.0	2
2.0 to 3.0	0	> 5	0
> 3	0		All roads
	All roads	Average	0.40
Average	1.31		

4.3. The effect of the multi-fidelity framework on trust region algorithm

Table 4.13: Speed up for TRSVR Algorithms for two categories of IPs

Speed factor	# of Roads	# of Roads
< 0.7	1	0
0.7 to 0.9	0	1
0.9 to 1.1	2	4
1.1 to 2	14	12
2 to 3	0	0
> 3	0	0
	C1	C2
Average	1.31	1.30

Table 4.14: Cost difference for TRSVR algorithm for 2 categories of IPs

Cost Difference(%)	# of Roads	# of Roads
< 0	1	1
0 to 1	13	14
1 to 2	1	2
2 to 5	2	0
> 5	0	0
	C1	C2
Average	0.50	0.30

algorithm, we run TRSVR0 and TRSVR1 algorithms on different categories of roads presented in Table 4.5 and Table 4.8.

Table 4.15 shows the speed up summary results of TRSVR0 and TRSVR1 algorithms for Category C1 to Category C2 roads, respectively. As shown in Table 4.15, in contrast to the GPS algorithm that the number of IPs has great effect on the performance of the multi-fidelity framework, in the TRSVR algorithm it seems that it does not have significant effect. In particular, the average speed up of the TRSVR algorithm is 30 and 31 percent for Category C1 and Category C2 in the average case.

Table 4.14 show the cost difference between TRSVR0 and TRSVR1 algorithms for Category C1 to Category C2 roads, respectively. We can see from Table 4.14 that there is no significant cost difference between TRSVR0 and TRSVR1 algorithms in the average case as it is less than 1 percent for both categories.

We studied the effect of multi-fidelity framework for the TRSVR algorithm for different categories of IPs. We now study the effect of the multi-fidelity paradigm for different groups of stations following the same

4.3. The effect of the multi-fidelity framework on trust region algorithm

Table 4.15: Speed up for the TRSVR algorithm for 3 different groups of stations

Speed factor	# of Roads (G1)	# of Roads (G2)	# of Roads (G3)
< 0.7	1	0	0
0.7 to 0.9	1	0	0
0.9 to 1.1	1	1	4
1.1 to 2	4	14	8
2 to 3	0	0	0
> 3	0	0	0
	G1	G2	G3
Average	1.15	1.39	1.33

categorization of stations in Table 4.8.

Table 4.15 represents the summary results of speed up of the TRSVR algorithm for 3 different groups of stations. As shown in Table 4.15, for Group G1 roads, there is a subtle difference between TRSVR0 and TRSVR1 algorithms in terms of computation time. However, for Group G2 and Group G3, TRSVR1 algorithm is faster than TRSVR0 algorithm by 39 and 33 percent, respectively, in the average case. We can also see that in contrast to the GPS algorithm that could only solve 3 out 13 roads for Group G3 roads, the TRSVR algorithm solves 12 out of 13 problems for Group G3 roads.

Table 4.16 demonstrates the summary results of cost difference between TRSVR0 and TRSVR1 algorithms for different groups of stations. As shown in Table 4.16, the cost difference between TRSVR0 and TRSVR1 is 0.64, 44 and 0.20, respectively, in the average case.

In total, from Table 4.13 and Table 4.15, we can realize that the size of IP set has a less effect on the efficiency of the TRSVR algorithm than the size of station set.

To see the final results obtained by TRSVR0 and TRSVR1 algorithms, we plot the horizontal alignment for Road 1. Figure 4.6 show the optimal horizontal alignments obtained by both versions of the TRSVR algorithm for Road 1.

4.4. Comparison with a standard solver

Table 4.16: Cost difference of TRSVR algorithm for different groups of stations

Cost Difference	# of Roads (G1)	# of Roads (G2)	# of Roads (G3)
< 0	1	0	1
0 to 1	4	13	10
1 to 2	1	2	0
2 to 5	1	0	1
> 5	0	0	0
	G1	G2	G3
Average	0.64	0.44	0.20

Figure 4.6: The horizontal alignment carried out by TRSVR algorithms for Road 1

4.4 Comparison with a standard solver

In order to compare the GPS and the TRSVR algorithms with a standard solver, we compare them with NOMAD [9]. The mesh adaptive direct search algorithm used in NOMAD solver contains several parameters that have a

4.4. Comparison with a standard solver

Table 4.17: The parameter values for NOMAD

Parameter	NOMAD
Initial mesh size	$\{1, 2, 5, 10\}$
The mesh quadratic radius factor	$\{1.1, 2, 3, 4\}$
The mesh coarsening exponent	$\{1, 2, 3, 4\}$

drastic effect on the performance of the algorithm. Among these parameters, we tune 3 parameters, set 4 parameters and leave the others with default values suggested in [22]. The parameters we tune are the initial mesh size, the mesh coarsening exponent and the mesh quadratic radius factor. The parameters we set are the final mesh size, the maximum allowed time to solve the problem and the maximum number of function evaluations, having the same values as mentioned in Section 4.2.1. We also set the mesh refining exponent to the inverse values of the mesh coarsening exponent.

Table 4.4 shows the parameters values for the tuned parameters. In order to compare these configuration, we follow the same process as Section 4.2.1.

Figure 4.7: The performance profile of NOMAD for 64 different configurations on five small roads. The legend only shows the top two configurations.

Figure 4.7 shows the performance profile of NOMAD for 64 different configurations on five small roads. As shown in Figure 4.7, both “config3” and “config15” seem to perform better than the other configurations. However, as we zoom in the figure, we realize that “config3” is more efficient than “config15”. Therefore, we choose “config3” = initial mesh size = 5, the

mesh quadratic radius factor = 1.1 and the mesh coarsening exponent = 3.

After tuning the parameters of NOMAD, we test all the algorithms presented with it. To compare the results obtained by NOMAD and the rest of the algorithm, we use the performance profile. Figure 4.8 shows the performance profile of all the five algorithms on 35 roads.

Figure 4.8: The performance profile of 5 different algorithms across all roads

As shown in Table 4.8, the fastest algorithms are TRSVR0 and TRSVR1 for the tolerance of 5 percent while the most robust algorithm is NOMAD. However, as the tolerance decreases, the performance of both versions of TRSVR algorithms plunges down drastically. We can also see that GPS1 algorithm is faster and more robust than GPS0. However, note that the performance profile does not imply relative ranking, i.e. if we remove the best algorithm in a set of algorithms, we may see a shift in the ranking among the rest of algorithms. Readers can find more information about the switching phenomenon in [11].

4.4.1 Comparison with NOMAD for different classes of roads

As mentioned earlier, we categorize the 35 roads based on the number of IPs (see Table 4.5) and the number of stations (see Table 4.8).

Figure 4.9 shows the performance profile of all the 5 algorithms for Category C1 roads. As shown in Figure 4.9, for Category C1 and when the error is 5 percent, the most efficient algorithm is TRSVR1 and the most robust

4.4. Comparison with a standard solver

ones are GPS1 and NOMAD. However, as the error decreases, the performance of TRSVR0 and TRSVR1 algorithms drastically decreases. For the GPS algorithm, GPS1 is faster and more robust than GPS0 in 5 percent error but GPS0 can solve more problems in Category C1 roads in 1 percent error.

Figure 4.9: The performance profile of 5 different algorithms for Category C1 roads

Figure 4.10 shows the performance profile of all of the five algorithms for Category C2 roads. As shown in Figure 4.10, similar to Category C1, for $\tau = 5\%$ TRSVR1 is the fastest algorithm, and the most robust algorithms are GPS1 and NOMAD for 5 percent and only NOMAD for 1 percent tolerance. We can also see that GPS1 algorithm is much faster and more robust than GPS0 algorithm. From Figure 4.9 to Figure 4.10, we see that from Category C1 to Category 2 roads, the most robust algorithm is NOMAD and the most efficient one is TRSVR1. Also, note that as the size of the IP set increases, GPS1 algorithm becomes more efficient, comparing to GPS0 algorithm.

In the next step, we study the effect of the station set size on the performance and the efficiency of the algorithms. We use the same categorization for the station set as presented in Table 4.8. Figure 4.11 shows the performance profile of the 5 algorithms for Group G1 roads. As shown in Figure 4.11, for 5 percent accuracy, the most efficient algorithm is TRSVR1 and the most robust one is NOMAD. However, when more accuracy is required (1 percent tolerance), the fastest and the most robust algorithm is NOMAD.

4.4. Comparison with a standard solver

Figure 4.10: The performance profile of 5 algorithms for Category C2 roads

Figure 4.11: The performance profile of 5 different algorithms for Group 1 roads

Figure 4.12 shows the performance profile of the algorithms for Group G2 roads. As shown in Figure 4.12, for 5 percent accuracy, the most robust algorithm is NOMAD and the most efficient one is TRSVR1 algorithm. However, for 1 percent tolerance, the most robust and the most efficient one is NOMAD algorithm.

Figure 4.13 shows the performance profile of the 5 different algorithms

4.4. Comparison with a standard solver

Figure 4.12: The performance profile of 5 different algorithms for Group 2 roads

for Group G3 roads. As shown in Figure 4.13, for 5 percent tolerance, the most efficient algorithm is TRSVR1 and the most robust one is NOMAD. However, for the 1 percent tolerance, the most efficient and the most robust one is NOMAD algorithm.

Figure 4.13: The performance profile of 5 different algorithms for Group 3 roads

Altogether, from Figure 4.9 to Figure 4.13 we can conclude that TRSVR1 algorithm is the fastest algorithm among all of the five algorithms for different sizes of IPs and different sizes of station set when 5 percent tolerance is required. However, when 1 percent tolerance is needed, the most efficient algorithm is either NOMAD or GPS1 for most of categories of IPs and groups of the station set. Moreover, the most robust algorithms are also either GPS1 or NOMAD algorithm that can solve most of the problems.

Chapter 5

Conclusion

The horizontal alignment optimization problem is a time-consuming and complicated optimization problem. Solving this problem can take hours to solve for Derivative-Free Optimization (DFO) algorithms. In this thesis, we studied the use of multi-fidelity framework for some DFO algorithms in the HA problem in order to save computational time. The algorithms we studied are generalized pattern search and trust region algorithms. The multi-fidelity refers to use surrogate models with different precision in the hope of saving the computation time while keeping the approximation error within an acceptable tolerance.

In order to test the multi-fidelity framework, we studied both algorithms on 35 different roads. The experimental results showed that the multi-fidelity framework increases the speed of the GPS algorithm by an average factor of 2.52 while increasing the speed of the trust region algorithm by an average factor of 1.31.

We also categorize the roads based on the number of intersection points and the number of stations. We demonstrate that when the size of the HA problem is small, the multi-fidelity framework did not help the HA algorithm, and even it deteriorates the speed of the algorithm. However, as the HA problem becomes larger, the multi-fidelity framework provides improvement in terms of speed. This is helpful since we already have good HA algorithms to optimize the small roads, and our industrial partner is more interested to see the computation time improvement when it comes to large roads.

Finally, we study the HA algorithms in terms of number of stations. Interestingly, we see that there is a direct correlation between the number of stations and the effect of the multi-fidelity framework. As the number of stations increases, the multi-fidelity framework provides greater improvement in terms of computation time. This is particularly important as more than half of the roads takes more than 1 hour to solve for the GPS algorithm.

Recommendations for future research

Even though the multi-fidelity framework helps the GPS TRSVR algorithms to improve their speed, there is still a room for further improvement.

In the GPS algorithm presented, we only considered the contraction for the mesh size. While this assumption makes it easier to prove the convergence of the algorithm, it may also deteriorate the performance of the algorithm. Furthermore, we utilized the basic poll strategy with maximal positive basis for the construction of polling directions. This strategy makes the GPS algorithm evaluate all possible polling directions, thereby spending too much computation time on potentially unsuccessful polling directions. In the future, we can use the opportunistic polling with ordering that was shown [1] to reduce the computation time of the GPS algorithm significantly.

In the TRSVR algorithm presented, we consider the minimal positive basis for the construction of the sample points in the trust region. Sometimes using the maximal positive basis or different strategies might provide better results. Furthermore, for the TRSVR algorithm, we only studied the initial trust region size, trust region size decreasing and error control parameters. The TRSVR algorithm has other parameters such as parameters for controlling the size of sample set. These parameters often have noticeable impact on the performance of the algorithm, and by tuning them we might obtain better results.

In our multi-fidelity framework, we used the true objective function in two cases: when we are close enough to a local minimizer and at the end of each iteration to check the quality of the new solution. Even though using the true objective function in these cases helps the algorithm not to go far away from the promising regions of the search space, it increases the computation time of the algorithm. In the future, it might be worth to use the true objective function in only one or neither of the cases to reduce the computation time of the algorithm.

In terms of optimality, both the GPS and the TRSVR algorithms studied, guarantee local optimality. In the future, if we are interested in global optimality, we can choose global optimization algorithms such as branch and cut approach [34] to find the global minimizer. Since the road design optimization problem is non-convex mixed-integer non-linear programming problem, we can use global optimization solvers such as BARON [30] to solve the small size roads, and for large-size roads, we can use the solution obtained by a global solver as a initial point for the GPS or the TRSVR algorithm.

In our settings, we use a serial computing environment in both the GPS and the TRSVR algorithms. One potential future direction can be using the parallel computing environment in the algorithms. In fact, we can implement a multi-start GPS or TRSVR algorithm where for each core the algorithm can choose one starting point.

Bibliography

- [1] Mark A Abramson, Charles Audet, and John Dennis. Generalized pattern search algorithms: unconstrained and constrained cases. In *IMA Workshop*, 2003. → pages 56
- [2] Kazuhiro Aruga. Tabu search optimization of horizontal and vertical alignments of forest roads. *Journal of Forest Research*, 10(4):275–284, Aug 2005. → pages 2
- [3] Charles Audet and John E Dennis Jr. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on optimization*, 17(1):188–217, 2006. → pages 3
- [4] Charles Audet and Warren Hare. *Derivative-Free and Blackbox Optimization*. Springer Series in Operations Research and Financial Engineering. Springer International Publishing, 2017. → pages 4, 8, 11, 12, 15, 36
- [5] Vahid Beiranvand, Warren Hare, and Yves Lucet. Best practices for comparing optimization algorithms. *Optimization and Engineering*, 18(4):815–848, 2017. → pages 34
- [6] Andrew R Conn, Nicholas IM Gould, and Philippe L Toint. *Trust region methods*, volume 1. SIAM, 2000. → pages 15, 19, 21
- [7] Andrew R Conn, Katya Scheinberg, and Philippe L Toint. On the convergence of derivative-free methods for unconstrained optimization. *Approximation theory and optimization: tributes to MJD Powell*, pages 83–108, 1997. → pages 19
- [8] IBM ILOG CPLEX. V12. 1: Users manual for cplex. *International Business Machines Corporation*, 46(53):157, 2009. → pages 26
- [9] Jonathan Currie and David Wilson. Opti: lowering the barrier between open source optimizers and the industrial matlab user. *Foundations of Computer-aided Process Operations*, 24:32, 2012. → pages 33, 48

- [10] M Giselle Fernández-Godino, Chanyoung Park, Nam-Ho Kim, and Raphael T Haftka. Review of multi-fidelity models. In *ECCOMAS Congress 2016*, 2016. → pages 5
- [11] Nicholas Gould and Jennifer Scott. A note on performance profiles for benchmarking software. *ACM Transactions on Mathematical Software (TOMS)*, 43(2):15, 2016. → pages 50
- [12] Nicholas IM Gould, Stefano Lucidi, Massimo Roma, and Philippe L Toint. Solving the trust-region subproblem using the lanczos method. *SIAM Journal on Optimization*, 9(2):504–525, 1999. → pages 15, 21
- [13] Michael W Hancock and Bud Wright. A policy on geometric design of highways and streets, 2013. → pages 2
- [14] Warren Hare, Majid Jaberipour, and Yves Lucet. Multi-fidelity surrogates for vertical alignment optimization in road design. *in submission*. → pages 3, 6, 26, 27
- [15] Dessalegn Amenu Hirpa. Simultaneous optimization of vertical and horizontal road alignments. Master’s thesis, University of British Columbia, 2014. → pages 3
- [16] Jiexiang Hu, Qi Zhou, Ping Jiang, Xinyu Shao, and Tingli Xie. An adaptive sampling method for variable-fidelity surrogate models using improved hierarchical kriging. *Engineering Optimization*, 50(1):145–163, 2018. → pages 4, 5
- [17] Edward Huang, Jie Xu, Si Zhang, and Chun-Hung Chen. Multi-fidelity model integration for engineering design. *Procedia Computer Science*, 44:336 – 344, 2015. → pages 4, 5
- [18] Likeng Huang, Zhenghong Gao, and Dehu Zhang. Research on multi-fidelity aerodynamic optimization methods. *Chinese Journal of Aeronautics*, 26(2):279–286, 2013. → pages 4, 5
- [19] Ronald L Iman. Latin hypercube sampling. *Wiley StatsRef: Statistics Reference Online*, 2008. → pages 17
- [20] Manoj Kumar Jha, Paul Schonfeld, and Jyh-Cherng Jong. *Intelligent Road Design*. Advances in transport. WIT Press, 2006. → pages 1, 2
- [21] Donald R Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21(4):345–383, 2001. → pages 5

- [22] Sébastien Le Digabel and Christophe Tribes. *NOMAD User Guide: Version 3.5*. Groupe d'études et de recherche en analyse des décisions (GERAD), 2009. → pages 49
- [23] Sukanto Mondal. Horizontal alignment optimization in road design. Master's thesis, University of British Columbia, 2014. → pages 3, 24, 26
- [24] Jorge J Moré and Stefan M Wild. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization*, 20(1):172–191, 2009. → pages 35
- [25] Minister of Transport. Transportation in Canada 2016. <https://www.tc.gc.ca/eng/policy/transportation-canada-2016.html#the-role-of-transportation-in-the-economy>, 2017. → pages 1
- [26] Chanyoung Park, Raphael T Haftka, and Nam-Ho Kim. Experience with several multi-fidelity surrogate frameworks. In *11th World congress on structural and multidisciplinary optimization, Sydney, Australia*, pages 7–12, 2015. → pages 5
- [27] Benjamin Peherstorfer, Karen Willcox, and Max Gunzburger. Survey of multifidelity methods in uncertainty propagation, inference, and optimization. Technical report, Aerospace Computational Design Laboratory, Department of Aeronautics and Astronautics, Massachusetts, 2016. → pages 5
- [28] Elijah Polak and Michael Wetter. Precision control for generalized pattern search algorithms with adaptive precision function evaluations. *SIAM Journal on Optimization*, 16(3):650–669, 2006. → pages 5, 6, 8, 10, 12, 14, 15, 17, 35
- [29] Yasha Pushak. Road design optimization with a surrogate function. Master's thesis, University of British Columbia, Apr 2015. → pages 4
- [30] Nikolaos V Sahinidis. Baron: A general purpose global optimization software package. *Journal of Global Optimization*, 8(2):201–205, 1996. → pages 56
- [31] Jun Takaki and Nobuo Yamashita. A derivative-free trust-region algorithm for unconstrained optimization with controlled error. *Numerical Algebra, Control & Optimization*, 1(1):117–145, 2011. → pages 5, 6, 15, 17, 19, 20, 43

- [32] Pham Dinh Tao and Le Thi Hoai An. A dc optimization algorithm for solving the trust-region subproblem. *SIAM Journal on Optimization*, 8(2):476–505, 1998. → pages 15, 21
- [33] Chan Weng Tat and Fan Tao. Using gis and genetic algorithm in highway alignment optimization. In *Proceedings of the 2003 IEEE International Conference on Intelligent Transportation Systems*, volume 2, pages 1563–1567 vol.2, Oct 2003. → pages 3
- [34] Mohit Tawarmalani and Nikolaos V Sahinidis. A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming*, 103(2):225–249, 2005. → pages 56
- [35] Virginia Torczon. On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, 7(1):1–25, 1997. → pages 12
- [36] Michael Wetter and Elijah Polak. A convergent optimization method using pattern search algorithms with adaptive precision simulation. *Building Services Engineering Research and Technology*, 25(4):327–338, 2004. → pages 5

Appendix

Appendix A

Data Tables

Here, we present the detailed results for GPS0, GPS1, TRSVR0, TRSVR1 and NOMAD algorithms on 35 roads.

It should be noted that in Table A.1 and Table A.2, Table A.3, Table A.4 and Table A.5, “Cost” is the optimized VA cost obtained, “Time” is the total computation time measured, “Status” and “Reason” are the status and reason that an algorithm terminates, “Memory” is the memory used by a computer to solve the problem, “True FC” and “Surrogate FC” are the number of surrogate and true function calls used for the algorithm to solve the HA problem.

In Table A.1 and Table A.2, “Mesh Small” refers to the fact that the GPS algorithm stops as reaches the final mesh size. Furthermore, “ F_e exhaustion” refers to the fact the algorithm terminates because it hits the upper bound for the number of function evaluations, and “24h T.” demonstrates that the algorithm reaches the maximum computation time before finding a local minimizer.

In Table A.3 and Table A.4, “TR Small” means that the algorithm stops because the trust region is small.

In Table A.8, “Initial Solution” refers to the VA cost obtained for the initial alignment.

Appendix A. Data Tables

Table A.1: The detailed results of GPS0 for the HA problem across all roads

GPS0							
#	Cost(\$)	Time(s)	Status	Reason	Memory(mb)	True FC(#)	Surrogate FC(#)
1	252,975	334	Success	Mesh Small	1,645	106	0
2	44,458	160	Success	Mesh Small	1,450	41	0
3	537,873	5,023	Success	Mesh Small	1,643	690	0
4	127,411	1,211	Success	Mesh Small	1,645	150	0
5	8,554,830	1,054	Success	Mesh Small	1,636	137	0
6	10,951,800	762	Success	Mesh Small	1,643	90	0
7	5,889,700	1,069	Success	Mesh Small	1,649	130	0
8	7,273,130	1,729	Success	Mesh Small	1,654	144	0
9	363,861	5,538	Success	Mesh Small	1,600	463	0
10	6,986,830	1,598	Success	Mesh Small	1,653	94	0
11	6,082,110	1,533	Success	Mesh Small	1,651	128	0
12	204,763	17,273	Success	Mesh Small	1,702	694	0
13	650,128	4,566	Success	Mesh Small	1,193	164	0
14	278,496	1,320	Success	Mesh Small	1,275	53	0
15	1,514,740	43,276	Success	Mesh Small	1,395	1,732	0
16	356,061	1,242	Success	Mesh Small	1,268	49	0
17	244,938	1,767	Success	Mesh Small	1,622	56	0
18	2,853,810	3,626	Success	Mesh Small	1,632	106	0
19	614,144	68,809	Success	Mesh Small	1,608	3,482	0
20	121,374	-	Fail	24h T.	-	-	-
21	3,088,040	48,963	Success	Mesh Small	1,369	589	0
22	230,578	5,504	Success	Mesh Small	1,594	59	0
23	149,631	7,550	Success	Mesh Small	1,596	479	0
24	222,118	3,030	Success	Mesh Small	1,601	148	0
25	222,550	789	Success	Mesh Small	1,441	65	0
26	244,800	195	Success	Mesh Small	1,436	86	0
27	377,133	-	Fail	24h T.	-	-	-
28	568,715	-	Fail	24h T.	-	-	-
29	11,012,600	-	Fail	24h T.	-	-	-
30	1,016,780	-	Fail	24h T.	-	-	-
31	1,075,830	-	Fail	24h T.	-	-	-
32	47,097,000	-	Fail	24h T.	-	-	-
33	30,894,600	-	Fail	24h T.	-	-	-
34	30,155,800	-	Fail	24h T.	-	-	-
35	28,887	73	Success	Mesh Small	1,665	35	0

Appendix A. Data Tables

Table A.2: The detailed results of GPS1 for the HA problem across all roads

GPS1							
#	Cost(\$)	Time(s)	Status	Reason	Memory(mb)	True FC(#)	Surrogate FC(#)
1	251,160	226	Success	Mesh Small	1,298	21	169
2	44,752	39	Success	Mesh Small	1,284	10	41
3	543,235	1,504	Success	Mesh Small	1,288	33	661
4	127,346	563	Success	Mesh Small	1,284	21	253
5	8,604,670	613	Success	Mesh Small	1,285	24	289
6	10,953,600	481	Success	Mesh Small	1,285	27	217
7	5,897,510	508	Success	Mesh Small	1,284	22	177
8	7,327,620	1,078	Success	Mesh Small	1,283	30	241
9	367,957	1,428	Success	Mesh Small	1,287	19	533
10	6,996,660	969	Success	Mesh Small	1,290	21	169
11	6,163,850	841	Success	Mesh Small	1,290	20	161
12	204,763	18,124	Success	Mesh Small	1,247	37	1,185
13	650,128	1,280	Success	Mesh Small	1,289	19	305
14	278,496	577	Success	Mesh Small	1,289	11	89
15	1,545,050	12,700	Success	Mesh Small	1,303	50	2,601
16	356,061	520	Success	Mesh Small	1,288	10	81
17	244,938	799	Success	Mesh Small	1,305	11	89
18	2,853,810	1,860	Success	Mesh Small	1,312	26	209
19	614,144	-	Fail	24h Timeout	-	-	-
20	131,313	15,516	Success	Mesh Small	1,346	22	1,673
21	3,115,360	17,052	Success	Mesh Small	1,268	44	1,409
22	230,578	2,221	Success	Mesh Small	1,240	13	105
23	152,889	746	Success	Mesh Small	1,240	10	121
24	223,508	1,468	Success	Mesh Small	1,244	23	277
25	222,550	937	Success	Mesh Small	1,247	12	97
26	247,984	316	Success	Mesh Small	1,341	9	73
27	391,768	-	Fail	24h Timeout	-	-	-
28	563,671	26,550	Success	Mesh Small	1,353	15	301
29	11,012,600	-	Fail	24h Timeout	-	-	-
30	1,028,300	20,355	Success	Mesh Small	1,350	34	2,041
31	1,084,200	68,432	Success	Mesh Small	1,246	44	2,993
32	47,035,100	52,576	Success	Mesh Small	1,350	38	1,825
33	30,894,600	-	Fail	24h Timeout	-	-	-
34	29,678,800	-	Fail	24h Timeout	-	-	-
35	28,579	36	Success	Mesh Small	1,530	13	40

Appendix A. Data Tables

Table A.3: The detailed results of TRSVR0 for the HA problem across all roads

#	TRSVR0						
	Cost(\$)	Time(s)	Status	Reason	Memory(mb)	True FC(#)	Surrogate FC(#)
1	262,740	82	Success	TR Small	1,655	31	0
2	46,223	41	Success	TR Small	1,472	20	0
3	560,863	343	Success	TR Small	1,652	60	0
4	136,550	280	Success	TR Small	1,652	39	0
5	8,876,460	243	Success	TR Small	1,653	41	0
6	11,183,200	183	Success	TR Small	1,652	31	0
7	6,035,530	248	Success	TR Small	1,669	30	0
8	7,468,290	325	Success	TR Small	1,697	30	0
9	377,635	1,001	Success	TR Small	1,644	77	0
10	7,224,780	273	Success	TR Small	1,651	18	0
11	6,160,100	330	Success	TR Small	1,641	31	0
12	215,049	1,776	Success	TR Small	1,721	90	0
13	719,287	753	Success	TR Small	1,261	50	0
14	288,204	626	Success	TR Small	1,277	30	0
15	1,630,980	2,419	Success	TR Small	1,456	137	0
16	365,717	630	Success	TR Small	1,282	31	0
17	250,031	776	Success	TR Small	1,675	30	0
18	3,038,370	831	Success	TR Small	1,652	31	0
19	848,507	2,741	Success	TR Small	1,655	175	0
20	154,308	5,522	Success	TR Small	1,407	198	0
21	3,214,460	6,772	Success	TR Small	1,419	90	0
22	238,498	2,303	Success	TR Small	1,659	31	0
23	153,501	481	Success	TR Small	1,659	41	0
24	239,318	801	Success	TR Small	1,658	40	0
25	223,173	163	Success	TR Small	1,470	15	0
26	255,320	55	Success	TR Small	1,440	31	0
27	386,715	70,888	Success	TR Small	1,302	12	0
28	584,928	36,686	Success	TR Small	1,330	60	0
29	11,077,600	81,564	Success	TR Small	1,741	29	0
30	1,049,540	7,218	Success	TR Small	1,280	161	0
31	1,126,310	14,163	Success	TR Small	1,554	174	0
32	47,319,700	17,589	Success	TR Small	1,455	121	0
33	30,860,100	-	Fail	24h T.	-	-	-
34	30,781,500	65,649	Success	TR Small	1,315	100	0
35	30,982	32	Success	TR Small	1,718	21	0

Appendix A. Data Tables

Table A.4: The detailed results of TRSVR1 for the HA problem across all roads

#	TRSVR1						Surrogate FC(#)
	Cost(\$)	Time(s)	Status	Reason	Memory(mb)	True FC(#)	
1	266,295	63	Success	TR Small	1,654	4	28
2	46,543	26	Success	TR Small	1,469	4	18
3	562,267	256	Success	TR Small	1,652	4	58
4	138,497	194	Success	TR Small	1,653	4	38
5	8,957,230	182	Success	TR Small	1,654	4	37
6	11,193,700	140	Success	TR Small	1,651	4	28
7	6,052,440	186	Success	TR Small	1,703	4	27
8	7,474,790	218	Success	TR Small	1,689	4	27
9	378,078	679	Success	TR Small	1,641	4	78
10	7,260,590	139	Success	TR Small	1,651	2	17
11	6,194,380	259	Success	TR Small	1,638	4	27
12	215,049	1,765	Success	TR Small	1,723	4	88
13	722,599	833	Success	TR Small	1,265	4	48
14	288,741	439	Success	TR Small	1,281	4	28
15	1,652,720	1,623	Success	TR Small	1,454	4	137
16	365,717	453	Success	TR Small	1,284	4	28
17	250,031	584	Success	TR Small	1,662	4	28
18	3,056,910	572	Success	TR Small	1,653	4	28
19	848,700	2,775	Success	TR Small	1,659	4	178
20	155,122	3,679	Success	TR Small	1,417	4	198
21	3,214,870	4,256	Success	TR Small	1,409	4	88
22	238,610	1,581	Success	TR Small	1,659	4	28
23	153,748	396	Success	TR Small	1,669	4	38
24	240,306	977	Success	TR Small	1,661	4	38
25	223,173	175	Success	TR Small	1,470	2	18
26	254,333	109	Success	TR Small	1,440	4	27
27	395,426	54,644	Success	TR Small	1,260	4	27
28	585,648	24,535	Success	TR Small	1,331	4	58
29	11,077,600	80,765	Success	TR Small	1,743	4	27
30	1,051,290	4,782	Success	TR Small	1,273	4	158
31	1,128,230	8,189	Success	TR Small	1,556	4	177
32	47,320,300	12,063	Success	TR Small	1,458	4	128
33	30,860,100	-	Fail	24h T.	-	-	-
34	30,547,400	68,147	Success	TR Small	1,254	4	98
35	31,789	25	Success	TR Small	1,721	4	18

Appendix A. Data Tables

Table A.5: The detailed results of NOMAD0 for the HA problem across all roads

NOMAD0							
#	Cost(\$)	Time(s)	Status	Reason	Memory(mb)	True FC(#)	Surrogate FC(#)
1	254,376	156	Success	Mesh Small	-	35	0
2	43,842	237	Success	Mesh Small	-	53	0
3	538,078	1,269	Success	Mesh Small	-	203	0
4	126,946	1,101	Success	Mesh Small	-	159	0
5	8,571,510	1,190	Success	Mesh Small	-	176	0
6	10,942,000	557	Success	Mesh Small	-	81	0
7	5,910,740	454	Success	Mesh Small	-	60	0
8	7,269,320	1,420	Success	Mesh Small	-	149	0
9	361,757	2,707	Success	Mesh Small	-	327	0
10	6,992,860	1,213	Success	Mesh Small	-	106	0
11	6,085,130	927	Success	Mesh Small	-	100	0
12	207,808	5,271	Success	Mesh Small	-	421	0
13	642,831	1,010	Success	Mesh Small	-	101	0
14	270,265	910	Success	Mesh Small	-	67	0
15	1,518,510	20,595	Success	Mesh Small	-	1,688	0
16	361,737	460	Success	Mesh Small	-	35	0
17	245,352	597	Success	Mesh Small	-	35	0
18	2,870,000	3,088	Success	Mesh Small	-	166	0
19	658,138	23,377	Success	Mesh Small	-	2,162	0
20	123,617	45,854	Success	Mesh Small	-	2,869	0
21	3,109,310	17,074	Success	Mesh Small	-	521	0
22	231,022	6,460	Success	Mesh Small	-	123	0
23	150,613	2,191	Success	Mesh Small	-	197	0
24	222,485	870	Success	Mesh Small	-	93	0
25	220,982	532	Success	Mesh Small	-	62	0
26	239,810	431	Success	Mesh Small	-	111	0
27	375,012	38,572	Success	Mesh Small	-	64	0
28	564,436	39,969	Success	Mesh Small	-	137	0
29	10,939,800	79,867	Success	Mesh Small	-	83	0
30	1,022,910	32,574	Success	Mesh Small	-	1,143	0
31	1,078,410	75,087	Success	Mesh Small	-	1,570	0
32	47,136,900	-	Fail	24h T.	-	-	-
33	30,784,400	-	Fail	24h T.	-	-	-
34	30,034,800	-	Fail	24h T.	-	-	-
35	28,889	183	Success	Mesh Small	-	45	0

Appendix A. Data Tables

Table A.6: Comparison of GPS0 and GPS1 algorithms

#	GPS0		GPS1		Comparison	
	Cost(\$)	Time(s)	Cost(\$)	Time(s)	Diff(%)	Speedup
1	252,975	334	251,160	226	-0.67	1.48
2	44,458	160	44,752	39	0.60	4.06
3	537,873	5,023	543,235	1,504	0.93	3.34
4	127,411	1,211	127,346	563	-0.05	2.15
5	8,554,830	1,054	8,604,670	613	0.56	1.72
6	10,951,800	762	10,953,600	481	0.02	1.58
7	5,889,700	1,069	5,897,510	508	0.13	2.11
8	7,273,130	1,729	7,327,620	1,078	0.73	1.60
9	363,861	5,538	367,957	1,428	1.03	3.88
10	6,986,830	1,598	6,996,660	969	0.13	1.65
11	6,082,110	1,533	6,163,850	841	1.32	1.82
12	204,763	17,273	204,763	18,124	0.00	0.95
13	650,128	4,566	650,128	1,280	0.00	3.57
14	278,496	1,320	278,496	577	0.00	2.29
15	1,514,740	43,276	1,545,050	12,700	1.83	3.41
16	356,061	1,242	356,061	520	0.00	2.39
17	244,938	1,767	244,938	799	0.00	2.21
18	2,853,810	3,626	2,853,810	1,860	0.00	1.95
21	3,088,040	48,963	3,115,360	17,052	0.84	2.87
22	230,578	5,504	230,578	2,221	0.00	2.48
23	149,631	7,550	152,889	746	2.11	10.12
24	222,118	3,030	223,508	1,468	0.56	2.06
25	222,550	789	222,550	937	0.00	0.84
26	244,800	195	247,984	316	1.24	0.62
35	28,887	73	28,579	36	-0.91	2.00

Table A.7: Comparison of TRSVR0 and TRSVR1 algorithms

#	TRSVR0		TRSVR1		Comparison	
	Cost(\$)	Time(s)	Cost(\$)	Time(s)	Diff(%)	Speedup
1	262,740	82	266,295	63	1.31	1.31
2	46,223	41	46,543	26	0.66	1.57
3	560,863	343	562,267	256	0.24	1.34
4	136,550	280	138,497	194	1.37	1.44
5	8,876,460	243	8,957,230	182	0.90	1.33
6	11,183,200	183	11,193,700	140	0.09	1.30
7	6,035,530	248	6,052,440	186	0.28	1.33
8	7,468,290	325	7,474,790	218	0.09	1.49
9	377,635	1,001	378,078	679	0.11	1.47
10	7,224,780	273	7,260,590	139	0.49	1.97
11	6,160,100	330	6,194,380	259	0.55	1.28
12	215,049	1,776	215,049	1,765	0.00	1.01
13	719,287	753	722,599	833	0.44	0.90
14	288,204	626	288,741	439	0.18	1.43
15	1,630,980	2,419	1,652,720	1,623	1.31	1.49
16	365,717	630	365,717	453	0.00	1.39
17	250,031	776	250,031	584	0.00	1.33
18	3,038,370	831	3,056,910	572	0.60	1.45
19	848,507	2,741	848,700	2,775	0.02	0.99
20	154,308	5,522	155,122	3,679	0.49	1.50
21	3,214,460	6,772	3,214,870	4,256	0.01	1.59
22	238,498	2,303	238,610	1,581	0.05	1.46
23	153,501	481	153,748	396	0.16	1.21
24	239,318	801	240,306	977	0.40	0.82
25	223,173	163	223,173	175	0.00	0.93
26	255,320	55	254,333	109	-0.38	0.50
27	386,715	70,888	395,426	54,644	2.20	1.30
28	584,928	36,686	585,648	24,535	0.12	1.50
29	11,077,600	81,564	11,077,600	80,765	0.00	1.01
30	1,049,540	7,218	1,051,290	4,782	0.17	1.51
31	1,126,310	14,163	1,128,230	8,189	0.17	1.73
32	47,319,700	17,589	47,320,300	12,063	0.00	1.46
34	30,781,500	65,649	30,547,400	68,147	-0.76	0.96
35	30,982	32	31,789	25	2.37	1.26

Appendix A. Data Tables

Table A.8: Comparison of VA cost obtained by 5 HA algorithms

	GPS 0	GPS 1	TRSVR 0	TRSVR 1	NOMAD	Initial Solution
#	Cost(\$)	Cost(\$)	Cost(\$)	Cost(\$)	Cost(\$)	Cost(\$)
1	252,975	251,160	262,740	266,295	254,376	270,431
2	44,458	44,752	46,223	46,543	43,842	48,835
3	537,873	543,235	560,863	562,267	538,078	575,425
4	127,411	127,346	136,550	138,497	126,946	142,318
5	8,554,830	8,604,670	8,876,460	8,957,230	8,571,510	8,957,990
6	10,951,800	10,953,600	11,183,200	11,193,700	10,942,000	11,263,800
7	5,889,700	5,897,510	6,035,530	6,052,440	5,910,740	6,063,630
8	7,273,130	7,327,620	7,468,290	7,474,790	7,269,320	7,499,100
9	363,861	367,957	377,635	378,078	361,757	396,965
10	6,986,830	6,996,660	7,224,780	7,260,590	6,992,860	7,287,410
11	6,082,110	6,163,850	6,160,100	6,194,380	6,085,130	6,207,560
12	204,763	204,763	215,049	215,049	207,808	218,411
13	650,128	650,128	719,287	722,599	642,831	752,019
14	278,496	278,496	288,204	288,741	270,265	294,034
15	1,514,740	1,545,050	1,630,980	1,652,720	1,518,510	1,655,240
16	356,061	356,061	365,717	365,717	361,737	382,428
17	244,938	244,938	250,031	250,031	245,352	255,164
18	2,853,810	2,853,810	3,038,370	3,056,910	2,870,000	3,065,770
19	614,144	614,144	848,507	848,700	658,138	922,876
20	121,374	131,313	154,308	155,122	123,617	167,372
21	3,088,040	3,115,360	3,214,460	3,214,870	3,109,310	3,262,090
22	230,578	230,578	238,498	238,610	231,022	240,637
23	149,631	152,889	153,501	153,748	150,613	154,332
24	222,118	223,508	239,318	240,306	222,485	246,076
25	222,550	222,550	223,173	223,173	220,982	227,522
26	244,800	247,984	255,320	254,333	239,810	256,546
27	377,133	391,768	386,715	395,426	375,012	396,176
28	568,715	563,671	584,928	585,648	564,436	599,988
29	11,012,600	11,012,600	11,077,600	11,077,600	10,939,800	11,167,800
30	1,016,780	1,028,300	1,049,540	1,051,290	1,022,910	1,055,270
31	1,075,830	1,084,200	1,126,310	1,128,230	1,078,410	1,129,760
32	47,097,000	47,035,100	47,319,700	47,320,300	47,136,900	47,345,800
33	30,894,600	30,894,600	30,860,100	30,860,100	30,784,400	31,208,100
34	30,155,800	29,678,800	30,781,500	30,547,400	30,034,800	30,991,100
35	28,887	28,579	30,982	31,789	28,889	33,989

Appendix A. Data Tables

Table A.9: Comparison of computation time obtained by 5 HA algorithms

	GPS 0	GPS 1	TRSVR 0	TRSVR 1	NOMAD
#	Time(s)	Time(s)	Time(s)	Time(s)	Time(s)
1	334	226	82	63	156
2	160	39	41	26	237
3	5,023	1,504	343	256	1,269
4	1,211	563	280	194	1,101
5	1,054	613	243	182	1,190
6	762	481	183	140	557
7	1,069	508	248	186	454
8	1,729	1,078	325	218	1,420
9	5,538	1,428	1,001	679	2,707
10	1,598	969	273	139	1,213
11	1,533	841	330	259	927
12	17,273	18,124	1,776	1,765	5,271
13	4,566	1,280	753	833	1,010
14	1,320	577	626	439	910
15	43,276	12,700	2,419	1,623	20,595
16	1,242	520	630	453	460
17	1,767	799	776	584	597
18	3,626	1,860	831	572	3,088
19	68,809	24h T.	2,741	2,775	23,377
20	24h T.	15,516	5,522	3,679	45,854
21	48,963	17,052	6,772	4,256	17,074
22	5,504	2,221	2,303	1,581	6,460
23	7,550	746	481	396	2,191
24	3,030	1,468	801	977	870
25	789	937	163	175	532
26	195	316	55	109	431
27	24h T.	24h T.	70,888	54,644	38,572
28	24h T.	26,550	36,686	24,535	39,969
29	24h T.	24h T.	81,564	80,765	79,867
30	24h T.	20,355	7,218	4,782	32,574
31	24h T.	68,432	14,163	8,189	75,087
32	24h T.	52,576	17,589	12,063	24h T.
33	24h T.	24h T.	24h T.	24h T.	24h T.
34	24h T.	24h T.	65,649	68,147	24h T.
35	73	36	32	25	183