Department of Mathematics and Statistics

Optimisation of Large Scale Network Problems

Mark Ted Grigoleit

This thesis is presented for the Degree of

Doctor of Philosophy

of

Curtin University of Technology

April 2008

Declaration

To the best of my knowledge and belief this thesis contains no material previously published by any other person except where due acknowledgement has been made. This thesis contains no material which has been accepted for the award of any other degree or diploma in any university.

Signature: ....

Date: ....

Summary

The Constrained Shortest Path Problem (CSPP) consists of finding the shortest path in a graph or network that satisfies one or more resource constraints. Without these constraints, the shortest path problem can be solved in polynomial time; with them, the CSPP is NP-hard and thus far no polynomial-time algorithms exist for solving it optimally.

The problem arises in a number of practical situations. In the case of vehicle path planning, the vehicle may be an aircraft flying through a region with obstacles such as mountains or radar detectors, with an upper bound on the fuel comsumption, the travel time or the risk of attack. The vehicle may be a submarine travelling through a region with sonar detectors, with a time or risk budget. These problems all involve a network which is a discrete model of the physical domain. Another example would be the routing of voice and data information in a communications network such as a mobile phone network, where the constraints may include maximum call delays or relay node capacities. This is a problem of current economic importance, and one for which time-sensitive solutions are not always available, especially if the networks are large.

We consider the simplest form of the problem, large grid networks with a single side constraint, which have been studied in the literature. This thesis explores the application of Constraint Programming combined with Lagrange Relaxation to achieve optimal or near-optimal solutions of the CSPP. The following is a brief outline of the contribution of this thesis.

Lagrange Relaxation may or may not achieve optimal or near-optimal results on its own. Often, large duality gaps are present. We make a simple modification to Dijkstra's algorithm that does not involve any additional computational work in order to generate an estimate of path time at every node. We then use this information to constrain the network along a bisecting meridian. The combination of Lagrange Relaxation (LR) and a heuristic for filtering along the meridian provide an aggressive method for finding near-optimal solutions in a short time.

Two network problems are studied in this work. The first is a Submarine Transit Path problem in which the transit field contains four sonar detectors at known locations, each with the same detection profile. The side constraint is the total transit time, with the submarine capable of 2 speeds. For the single-speed case, the initial LR duality gap may be as high as 30%. The first hybrid method uses a single centre meridian to constrain the network based on the unused time resource, and is able to produce solutions that are generally within 1% of optimal and always below 3%. Using the computation time for the initial Lagrange Relaxation as a baseline, the average computation time for the first hybrid method is about 30% to 50% higher, and the worst case CPU times are 2 to 4 times higher.

The second problem is a random valued network from the literature. Edge costs, times, and lengths are uniform, randomly generated integers in a given range. Since the values given in the literature problems do not yield problems with a high duality gap, the values are varied and from a population of approximately 100,000 problems only the worst 200 from each set are chosen for study. These problems have an initial LR duality gap as high as 40%.

A second hybrid method is developed, using values for the unused time resource and the lower bound values computed by Dijkstra's algorithm as part of the LR method. The computed values are then used to position multiple constraining meridians in order to allow LR to find better solutions. This second hybrid method is able to produce solutions that are generally within 0.1% of optimal, with computation times that are on average 2 times the initial Lagrange Relaxation time, and in the worst case only about 5 times higher.

The best method for solving the Constrained Shortest Path Problem reported in the literature thus far is the LRE-A method of Carlyle et al. (2007), which uses Lagrange Relaxation for preprocessing followed by a bounded search using aggregate constraints. We replace Lagrange Relaxation with the second hybrid method and show that optimal solutions are produced for both network problems with computation times that are between one and two orders of magnitude faster than LRE-A.

In addition, these hybrid methods combined with the bounded search are up to 2 orders of magnitude faster than the commercial CPlex package using a straightforward MILP formulation of the problem.

Finally, the second hybrid method is used as a preprocessing step on both network problems, prior to running CPlex. This preprocessing reduces the network size sufficiently to allow CPlex to solve all cases to optimality up to 3 orders of magnitude faster than without this preprocessing, and up to an order of magnitude faster than using Lagrange Relaxation for preprocessing.

Chapter 1 provides a review of the thesis and some terminology used. Chapter 2 reviews previous approaches to the CSPP, in particular the two current best methods. Chapter 3 applies Lagrange Relaxation to the Submarine Transit Path problem with 2 speeds, to provide a baseline for comparison. The problem is reduced to a single speed, which demonstrates the large duality gap problem possible with Lagrange Relaxation, and the first hybrid method is introduced.

Chapter 4 examines a grid network problem using randomly generated edge costs and weights, and introduces the second hybrid method. Chapter 5 then applies the second hybrid method to both network problems as a preprocessing step, using both CPlex and a bounded search method from the literature to solve to optimality.

The conclusion of this thesis and directions for future work are discussed in Chapter 6.

Acknowledgements

After a long and sometimes frustrating 15-year sojourn in the land of computing, with this thesis I am returning at last to my first love, mathematics. I would like to first thank Dr. Stephen Hill, with whom I worked in my last IT job, for helping me get the Research Assistant position at Curtin University and for his assistance and advice on this project. Secondly, I would like to thank my senior supervisor Professor Lou Caccetta for employing me in the Western Australia Centre of Excellence in Industrial Optimisation (WACEIO), and for providing support for the project.

Professor Caccetta's work has been supported in part by the Australian Research Council grant (DP 034639), and I am grateful to the ARC for their support. I would also like to thank the Defence Science Technology Organisation (DSTO) for providing the data on the sonar detector functions.

Apart from the generous financial and technical support from WACEIO and the Department of Mathematics, there have been those who provided emotional support every bit as necessary to the completion of this work. I would like to express my deep appreciation to the many fine whisky distilleries of Scotland, my AA sponsor, my parole officer and my psychotherapist. More seriously, without the lunchtime talks and dark humour of the two people who embody all these roles, I think I would have been too discouraged to continue. Thank you both, Carey and Stephen. You've helped more than you can know.

Contents

Chapter 1

Introduction

A great many practical problems of optimisation may be represented by a network model, consisting of nodes connected by edges. For example, we may wish to model the movement or transport of people through a rail or airline network, the path of a road through a mountainous area, the delivery of goods on a road network, or the transmission of data through a communications network. The nodes then represent destinations or way points, and the edges are the paths or connections between them.

One of the fundamental network problems is finding the best path from a starting node to a terminal node in a network. We may assign a cost to each edge, and then search for the path with the lowest cost. This is the classic *shortest path problem*, and there exist efficient polynomial time algorithms for it (eg. Dijkstra (1959)).

In practical problems, we usually have additional resources or weights associated with each edge, such as travel time or fuel consumption, which we need to place upper limits on. This turns the problem into the *constrained shortest path problem* which is known to be NP-complete (Garey and Jonhson (1979), p. 214). There are no known algorithms for solving this problem exactly in polynomial time.

What makes this class of problem interesting, both theoretically and in practise, is the computational complexity. The solution space, and therefore the time needed to search that solution space, grows exponentially with the size of the problem. Even with advances in computer technology, there will always be practical problems for which the networks are large and the solution times very long.

Very large networks may arise if we need to model, for example, the flight of an aircraft through airspace that is monitored with radar detectors, or the path of a submarine though a field of sonar detectors. These types of problems are modelled by laying a grid network over the transit area, and calculating the cost of each edge as the probability of being detected. The goal is to find the path which minimises the risk of detection while observing limits on the transit time, fuel consumption, or other limited resource. Of course, the more accurately we need to model the transit area, the larger this grid will be.

1.1 Contributions of the Thesis

Most approaches to the constrained shortest path problem (CSPP) invariably use a preprocessing step to remove infeasible edges and provide upper and lower bounds on the path cost, followed by a search of the remaining network to reduce the gap between these bounds and find optimal or near-optimal paths. The best known method for solving the CSPP reported in the literature is from Carlyle et al. (2007). This method uses Lagrange Relaxation with an enumeration of near-shortest paths.

Lagrange Relaxation is a powerful tool for the first step, although on problems that are not tightly constrained the network reduction is ineffective, and where the objective function is not convex there may be a large duality gap between the lower and upper bounds on path cost.

In this thesis we consider the CSPP on grid networks with a single side constraint. Two networks are studied. The first concerns the optimal path of a submarine through a region which contains passive sonar detectors, with an upper bound on the transit time. The submarine is capable of two speeds, and the location of the sonar detectors is known. The second problem is a grid network with randomly generated edge costs and times, similar to networks described in the literature. Two directions of transit are considered, one between opposite corners, as in the

Submarine Transit Path problem, and the other from any edge node on one side to any edge node on the opposite side.

In this thesis we explore the effectiveness of Lagrange Relaxation on these problems, identify problems for which Lagrange Relaxation produces solutions with a large duality gap because of local minima, and develop a heuristic for identifying local minima in the network responsible for the duality gap. We apply this heuristic in combination with Lagrange Relaxation to arrive at a hybrid method which produces improved upper and lower bounds, which in turn significantly speeds up the search phase. This approach dramatically improves the solution time for a selection of singly-constrained grid network problems that have a large initial duality gap.

1.2 Terminology

In this section we provide definitions of the basic terminology used to describe the network problems in this thesis.

A graph $G(V, E)$ consists of a set of vertices or nodes V and a set of connecting arcs or edges E. The number of nodes is defined as $|V| = n$ and the number of edges given by $|E| = m$.

An edge $e = (u, v) \in E$ connects node u and node v, has a cost $c_e \geq 0$ and one or more *weights* $f_{ie} \geq 0$ for $i \in I$, where I indexes a set of *side constraints*. Each side constraint i has a weight limit $g_i \geq 0$ defined.

A *directed path* from s to t is an ordered set of edges of the form
$E_p = \{(s, v_1), (v_1, v_2), ...(v_{k-1}, t)\}$. A path is *simple* if no vertices are repeated.

Given two distinct vertices $s, t \in V$, the CSPP seeks to find a simple path E_p from s to t such that the path cost $\sum_{e \in E_p} c_e$ is minimal and the side constraints $\sum_{e \in E_p} f_{ie} \leq g_i$ for all $i \in I$ are satisfied.

A *meridian* is defined as a set of nodes which bisect the network, such that any feasible path must pass through at least one of these nodes.

1.3 Outline of the Thesis

Chapter 2 provides a critical review of previous work. In particular, there are two approaches which are the current state of the art. These approaches are used as a basis for motivating the work of this thesis.

Chapter 3 introduces the Submarine Transit Path problem, which seeks to find the path with lowest cost which observes an upper bound on the transit time. The transit field is a square area of ocean 80 km on a side, which contains 4 identical passive sonar detectors arranged in 30 different location patterns. The submarine is considered to be capable of 2 speeds. Initial results of Lagrange Relaxation on the singly-constrained optimisation problem with 2 speeds are first produced as a baseline.

We then focus on the 1-speed version of the problem which has no parallel edges, and show how the Lagrange duality gap may be as high as 30%. The problem set is then expanded by introducing 3 new detector functions, two of which are smoothed variants of the original function and a third which is a monotonically decreasing function.

A hybrid method is developed which uses a systematic constraint on the unused transit time along a bisecting meridian, and Lagrange Relaxation is used to propagate these constraints. By introducing this temporary constraint, Lagrange Relaxation is then able to produce solutions which are generally within 1% of optimal, while taking on average only about twice as much CPU time as the initial application of Lagrange Relaxation.

Chapter 4 introduces two grid network problems based on problems reported in the literature. These problems have randomly generated edge costs and times, and a single side constraint. The structure of the two network problems is the same, but in the first problem the path traverses the network diagonally, and in the second network the path traverses horizontally. Problem sets of 5 different sizes are generated. From a population of roughly 100,000 problems for each size, a subset of the worst 200 cases for each network are chosen for study. These problems have an initial Lagrange Relaxation duality gap as high as 40%.

A second hybrid method is developed which expands on the first method, using values for the unused time resource and the Lagrange Relaxation lower bound information calculated at each node. This information is used to calculate a ratio which indicates where the Lagrange Relaxation solution is likely to be in a local minimum. The computed values are used to position multiple constraining meridians which allow Lagrange Relaxation to produce better solutions.

This second hybrid method is able to produce solutions for the random valued networks that are generally within 0.1% of optimal, with computation times that are on average only 2 times longer than the initial Lagrange Relaxation time.

Chapter 5 uses this second hybrid method as a preprocessing step on both the Submarine Transit Path problem and the two random valued network problems, and optimal solutions are produced using either CPlex or a bounded search. This hybrid method preprocessing is able to reduce the average CPlex CPU times by up to 3 orders of magnitude.

In Chapter 6 we present our conclusions and consider directions for future research.

Chapter 2

Background

This chapter provides a review of previous work in the literature on the constrained shortest path problem (CSPP). We focus on the most recent approach which uses Lagrange Relaxation with enumeration, and describe it in detail. Using key observations from previous approaches we motivate the approach used in this thesis.

2.1 Review of Previous Work

Approaches to the CSPP can be broadly divided into either exact methods or approximation schemes. Exact methods may further be divided into label setting algorithms based on dynamic programming, Lagrange Relaxation to produce upper and lower bounds on the path cost, and gap-closing methods which use bounded search or path enumeration. Preprocessing to reduce the problem size is often implicit in many approaches, but rarely mentioned on its own. These methods may also be combined. The following sections discuss the available methods described in the literature and their relative merits.

2.1.1 Approximation Schemes

Let P be a minimisation problem. Given $\epsilon > 0$, algorithm A is called an ϵ-*approximation algorithm* for P if it finds a feasible solution to any instance I of

P with value $A(I)$ such that

$$OPT(I) \leq A(I) \leq (1+\epsilon)OPT(I) \tag{2.1}$$

where $OPT(I)$ is optimal cost of instance I. An *approximation scheme* for P is a sequence of algorithms A_ϵ such that for all $\epsilon > 0$, A_ϵ is an ϵ-approximation algorithm for P. In addition, we say that A_ϵ is a *fully polynomial approximation scheme* (FPAS) if its complexity is a polynomial function of the length of the instance data and $\frac{1}{\epsilon}$.

Approximation schemes make use of *scaling and rounding* to reduce the range of the input data, thus reducing the complexity. However, because of this scaling and rounding, the solution obtained will not necessarily be a solution to the original problem. It will instead be an *approximate* one, with an error which can be bounded.

Hassin (1992) gives two fully polynomial approximation schemes for the CSPP with one side constraint, but does not give any computational results.

2.1.2 Dynamic Programming and Label Setting

Joksch (1966) described one of the first dynamic programming approaches. No computational results are given, but this is hardly surprising given the lack of computing resources available at the time. Dynamic programming involves a recursive method to calculate and update values at each node. These methods typically have pseudopolynomial time complexity, but may require a lot of storage space for intermediate results, and cannot always guarantee optimal results.

Aneja et al. (1983) describe a 2-step method which first reduces the network by removing nodes and edges which cannot be part of a feasible solution, then uses a label setting algorithm based on Dijkstra's algorithm (1959) to search to optimality. Space complexity results from having to store labels for each node. No computational results are given.

Desrochers and Soumis (1988) propose a label setting algorithm with pseudopolynomial time complexity, which was widely regarded as the best method for the CSPP for many years. However, their original computational work was carried out on a computer with very limited memory, and the resulting memory limitations occured on networks with as few as 250 nodes.

Dumitrescu and Boland (2003) analyse the effectiveness of a modified version of the label setting algorithm (LSA) of Desrochers and Soumis (1988) on a variety of different network problems. Problems include grid networks with randomly generated costs, and two real problems - a road path planning and a digital elevation model. A small number of problems in each class are considered (7, 8 and 28) with the largest being a grid of 450x300 nodes. A 10 minute time limit on computation is imposed.

Because of space complexity issues, none of the 200x200 problems with random edge costs could be solved by the modified label setting algorithm (MLSA) in the 10 minute limit. They observe that preprocessing using Lagrange Relaxation is much stronger than their original MLSA, and the preprocessing step often solves the problem without any further work. However, the space complexity would suggest that LSA is not suitable for very large network problems.

Zabarankin et al. (2002) model the path of an aircraft through a radar field using two methods: an analytical model based on calculus of variations, and a discrete network model. The models seek to find the lowest risk path of a single aircraft with a single radar. The network model has 1849 nodes and 28,056 edges. Experiments confirm that the discrete model can produce solutions within 1% of the optimal analytic solutions.

In Zabarankin et al. (2006) the same model is expanded to 3 dimensions and includes up to 25,662 nodes and 2.2 million edges. Once again, a label setting algorithm is used based on the MLSA of Dumitrescu and Boland (2003), using real values instead of integers. The number of labels produced can be very large, almost 10 million, and computation times on a Xeon 3.08 GHz processor with 3.37Gb of memory were as high as 192 minutes. Path smoothing to limit the path turn radius reduces the number of labels by up to 50%, thus reducing the solution times accordingly while incurring an insignificant increase in path cost. The authors comment that the number of labels that need to be generated is determined by how good the upper bound on cost is, and suggest (but do not implement) preprocessing based on Lagrange Relaxation.

2.1.3 Lagrange Relaxation

Everett (1963) is one of the first to describe the general approach of Lagrange Relaxation to combinatorial optimisation problems. He observes that Lagrange relaxation is well-suited to large problems, but that gaps may exist because the objective function may not be convex. Lagrange relaxation grew in popularity during the 1970s as it was applied to a broad range of combinatorial optimisation problems. Fisher (1981) gives a review of the developments.

Whether or not Lagrange Relaxation produces an optimal solution, its chief benefit is producing a lower bound on the path cost. Any feasible solution can provide the upper bound, and the optimal solution cost lies somewhere in this gap. Gap closing methods attempt to reduce this gap to zero, thus yielding an optimal solution.

2.1.4 2-Step Methods

Handler and Zang (1980) describe a 2-step method using Lagrange Relaxation to perform preprocessing followed by the k-shortest path algorithm of Yen (1971) to close the duality gap. The problems under study were small, containing at most 500 nodes, with randomly generated edge values and a single side constraint. Comparison of solution times with and without Lagrange Relaxation showed that Lagrange Relaxation can speed up the computation time by up to 2 orders of magnitude.

Beasley and Christofides (1989) use a similar 2-step method, applying preprocessing and a binary depth-first tree search. They test 24 problems of up to 500 nodes with either 1 or 10 side constraints. The preprocessing step alone solves the problem in about half the cases. Using Lagrange Relaxation they are able to reduce the network by up to 95%. All problems are solved within 30 seconds on a CDC-7600.

Mehlhorn and Ziegelmann (2000) propose yet another 2-step approach. They use a novel formulation of the dual that is similar to Lagrange Relaxation, but uses a geometric hull approach to find the best lower bounds. The second step is a gap closing method that uses spanning tree path ranking. Three problem sets are included: a road path planning problem with up to 77,059 nodes, a DEM problem with up to 40,000 nodes, and a curve approximation problem with up to 10,000 nodes. The authors claim to get much better bounds using the hull approach than the subgradient optimisation step of Lagrange Relaxation, thus leading to faster solution times with the gap-closing step. However, they terminate the subgradient optimisation early, which may produce non-optimal multipliers. Comparisons are made with label setting, k-shortest paths and the MILP formulation solved with CPlex. Their 2-step method is shown to be an order of magnitude faster than LSA on some problems. A rather short 5-minute computation time limit is imposed, which on a Sun Enterprise 333 MHz computer restricts the solution time so much that most of the k-shortest path and MILP problems are not solved. Full details are give in Ziegelmann (2001).

The approach of Carlyle et al. (2007) is of particular interest as it appears to be the leading method reported thus far. This approach combines Lagrange Relaxation with a bounded search to enumerate near-shortest paths. The bounded search is strengthened by aggregating the constraints to further limit the search. Grid networks with sizes up to 450x300 are considered, with 1 to 10 side constraints. On grid networks with randomly generated costs and a single side constraint, this method is found to be up to an order of magnitude faster than the label setting approach of Dumitrescu and Boland (2003). On problems with multiple constraints computation times are more variable, especially for problems with a large duality gap. One way to improve these cases is to refilter the network whenever the bounded search finds an improved solution that significantly improves the upper bound. Despite the added overhead, the approach with this reprocessing is invariably faster than the label setting approach.

The exponential time complexity of the bounded search can be a problem, however. To illustrate, on one grid problem of size 450x300 this approach takes only 0.3 seconds to find a solution that is within 0.5% of optimal, but a further 789 seconds to solve optimally, in a problem set where the average computation time is a only a few seconds. In an earlier 2005 draft of Carlyle et al. (2007), a decomposition method is proposed to handle this. A set of edges which bisect the network exactly in the middle – a meridian consisting of edges – is chosen, and on each pass only one of these edges is permitted. With the solution forced to use only one edge in the centre of the grid network, the authors report much better lower bounds, and the optimal solution is found in 61 seconds. By using multiplier information and improved upper bounds from preceeding passes, this is a form of reprocessing which helps to reduce the network and speed up the search. We will use elements of this decomposition in our approach.

2.1.5 Constraint Programming

Constraint programming, or the *constraint satisfaction problem*, is a relatively new approach to solving combinatorial problems. Arising from the field of computing science, constraint programming uses logic based tools to reason about discrete valued problems, implicitly eliminating infeasible regions of the solution space. Another way to describe it is rule-based domain reduction. A very good introduction to the topic is found in a recent book review by van Emden (2003). Hooker (2000) gives a good account of how constraint satisfaction applies to optimisation.

Several authors have applied constraint programming (CP) to the CSPP with some success. Sellmann (2003) provides a theoretical analysis of the time and space complexity of providing consistency checking in a CP approach to the CSPP which uses cost-based filtering. It is shown that relaxed consistency checking can be performed in the same time as a shortest path algorithm, making CP suitable for this type of problem.

Sellman and Fahle (2003) show how Lagrange Relaxation can be incorporated in a CP approach and apply this method to a multimedia recording problem which is NP-hard. Gellermann et al. (2005) apply the cost-based filtering approach to the CSPP and compare the effectiveness with the filtering of Aneja et al. (1983) and Beasley and Christofides (1989), showing that the CP approach of cost-based filtering is superior.

The central concept of CP is to select a constraint on the domain of a problem variable, then use a propagation algorithm to work out how the rest of the problem is affected by that choice. We will apply this principle in this thesis, using Lagrange Relaxation as the propagation algorithm.

2.1.6 Conclusion

Preprocessing plays an important, if not critical, role in first reducing the problem size. This is especially important given the exponential time complexity of any bounded search algorithm. In some cases preprocessing may succeed in finding an optimal solution on its own.

For larger problems, label setting methods may require too much space to store the potentially large number of labels, as observed in Dumitrescu and Boland (2003) and Zabarankin et al. (2006). The bounded search of Carlyle et al. (2007) is more attractive for this reason.

Lagrange Relaxation is a very powerful tool, producing upper and lower bounds on the solution cost which then help restrict the search. Much of the success of Lagrange Relaxation depends on how tight the bounds are, as large duality gaps may occur in practise.

And finally, a comment on computational power. It is difficult to directly compare previous computational results without implementing them, as computing power has advanced so much in recent years. Rough comparisons can be made by comparing the relative computing power of the computers that were used. One common measure of computing power is millions of operations per second (MIPS), but for mathematical analysis a more practical measure is to count the floating point operations per second (FLOPS). For example, the CDC 6600 used by Handler and Zang (1980) was a supercomputer introduced in 1964, was capable of 2-3 Mflops and had the equivalent of 0.94 Mbytes of memory. By comparison, a Pentium 4 desktop with a 3.0 GHz processor is capable of 4 Gflops, or about 3 orders of magnitude faster. In addition, the amount of memory available on a common desktop is many orders of magnitude greater than that of the former supercomputer.

The combination of processing power and memory storage available today makes it possible to consider problems of ever increasing size. It is on large problems of practical importance that we are able to test the limitations of available algorithms.

2.2 Our Approach

In this thesis we will use the approach of Carlyle et al. (2007) as a basis. Three key observations that motivate this decision are:

1. the effectiveness of Lagrange Relaxation in preprocessing
2. improvements to the bounded search provided by aggregating the constraints
3. the potential of the decomposition method to improve the bounds

Very few authors address the basic problem of what causes Lagrange Relaxation to have a duality gap in the first place. In order to find out how large duality gaps occur and how we may improve the solution times, we will deliberately choose problem instances where the duality gap is large, as these are the problems that pose the greatest challenge to gap closing methods.

We will use additional information gathered when finding the optimal Lagrange multiplier to predict where local minima are, and use a CP-like approach to achieve optimal or near-optimal upper bounds.

2.3 Problem Formulation and Methods

In this section we define the algorithms to be used in this thesis.

2.3.1 MILP Formulation

The CSPP may be expressed as an integer programming problem. Here we consider the grid based graph to be a network $G = (V,A)$, where V is the set of vertices or nodes and A is the set of arcs or edges. The model can be formulated so that G is directed or undirected. Associated with each edge e_{ij} from node i to node j there is a cost c_{ij}, and as we have only one edge resource which is the transit time, we denote this f_{ij}. Let g be the upper bound on path time. We assume non-negative values for cost and time. When G is directed, the CSPP can be specified as:

$$\min z = \sum_{(i,j)\in V} c_{ij}x_{ij} \tag{2.2}$$

subject to

$$\mathbf{fx} \leq \text{g} \tag{2.3}$$

$$x_{ij} \geq 0 \text{ and is integer.} \tag{2.4}$$

$$\sum_j x_{ij} - \sum_j x_{ji} = \begin{cases} 1, \text{if } i = \text{S}, \\ -1, \text{if } i = \text{T}, \\ 0 \text{ otherwise} \end{cases} \tag{2.5}$$

Equation (2.3) is the time constraint, and (2.5) ensures that a path from the starting node S to terminating node T is obtained. Note that the problem (2.2), (2.4) and (2.5) is just the standard shortest path problem which can be solved efficiently.

2.3.2 Lagrange Relaxation

The basic principle of Lagrangian Relaxation (LR) is to *relax* the problem by removing the side constraints and adding them to the objective function with a multiplier. Taking the single side contraint of equation (2.3), $\mathbf{fx} \leq \mathrm{g}$, and rearranging this to $(\mathbf{fx} - \mathrm{g}) \leq 0$, we observe that this term may be added to the objective function with a multiplier $\lambda \geq 0$ to give

$$z^* \geq z(\lambda) = \min \sum_{(i,j)\in V} c_{ij}x_{ij} + \lambda(\mathbf{fx} - \mathrm{g}) \tag{2.6}$$

where $z(\lambda)$ is the Lagranian dual function and forms the lower bound of the solution value z^*. The second step of the relaxation is to find the value of λ which gives the highest minimum value for $z(\lambda)$. Rearranging the terms we get

$$z(\lambda) = \min \sum_{(i,j)\in V} (c_{ij} + \lambda f_{ij})x_{ij} - \lambda\mathrm{g} \tag{2.7}$$

Note that equation (2.7) now expresses the edge costs with a modified cost $c'_{ij} = c_{ij} + \lambda f_{ij}$ that is a function of both the original edge cost c_{ij} and the weighted time f_{ij} of the edge, and that this relationship is linear. The search for λ now reduces to the task of setting a value for λ, calculating the modified edge costs c_{ij}' and using Dijkstra's algorithm to find the minimum value for $z(\lambda)$. The value of λ which maximises $z(\lambda)$ while preserving the relationship $(\mathbf{fx} - \mathrm{g}) \leq 0$ leads us to the solution z^*. If the gap between z^* and $z(\lambda)$ is zero, then z^* is optimal.

2.3.3 Dijkstra's Algorithm

At the heart of LR is the calculation of min $z(\lambda)$ which is done using Dijkstra's algorithm (Dijkstra (1959)). This is a well-known algorithm for finding the shortest path in a graph with directed edges and with non-negative edge weights. The pseudocode for this is shown in Algorithm 1.

Algorithm 1 Dijkstra's Algorithm

```
1:  set Q is empty
2:
3:  for each node v in Graph do
4:     if v is the source node then
5:        set v[total] to zero
6:        set v[state] to ACTIVE
7:        add v to Q
8:     else
9:        set v[total] to INFINITY
10:       set v[state] to IDLE
11:    end if
12: end for
13: repeat
14:    get from set Q vertex u with lowest total and state is not DONE
15:    for each edge e leaving u do
16:       get destination node v
17:       if state of v is IDLE then
18:          set v[state] to ACTIVE
19:          add v to Q
20:       end if
21:       if state of v is ACTIVE then
22:          cost = u[total] + e[cost]
23:          if  v[cost] > cost  then
24:             v[cost] = cost {update cost total for v}
25:          end if
26:       end if
27:    end for
28:    set u state DONE
29:    remove u from set Q
30: until set Q is empty
```

The first loop initialises the totals and state for each node, and adds the source node to the active set. For each iteration of the second loop, one node is chosen and every outgoing edge is explored. For each destination node, if the cost of reaching it through this edge is less than the current total, the total is updated. Then the intermediate source node is retired. Thus, on every iteration exactly one node is explored.

The time complexity of Dijkstra's algorithm depends on the method used to extract the next node to explore. If every node in the set Q is searched, then the worst case time complexity is $O(n^2)$. However, improvements are possible by using a special data structure to store the nodes that are active. Fredman and Tarjan (1987) show that by using a Fibonacci heap the time complexity may be reduced to $O(n \log n + m)$.

2.3.4 Modification to Dijkstra's Algorithm

The pseudocode for Dijkstra's algorithm in Algorithm 1 shows the calculation of the shortest path in terms of edge cost. However, we may use Dijkstra's algorithm to calculate the shortest path in terms of any edge variable, including the time or the Lagrangian modified cost.

In order to facilitate the analysis in this thesis, we make one small modification to Dijkstra's algorithm. At line 24 where the total cost for node v is updated, we may also update another total, the Lagrangian path time. This is especially useful when calculating the shortest path in terms of the Lagrangian modified cost $z(\lambda)$. Of course, the total for this time needs to be initialised to zero at line 5.

Whenever Dijkstra's algorithm is used to calculate the minimum $z(\lambda)$ path length, as a side effect this small modification will also update a time total. By applying Dijkstra's algorithm in both directions, we then get a node total for the minimum $z(\lambda)$ as well as an *estimate* of the time taken by the min $z(\lambda)$ path. Of course, this value is only an estimate, because the more a path deviates from the min $z(\lambda)$ path, the less accurate this time value will be.

2.3.5 Problem Reduction

Dijkstra's algorithm may be used in either direction, either to calculate the minimum path length from each node forward to the terminating node T, or from each node backwards to the source node S. Each edge has an array of values for cost, modified cost, and time, and we may use Dijkstra's algorithm to calculate the shortest path in terms of these edge variables. Each node also contains an array of values to store the minimum path lengths to S and T in terms of these edge variables. By applying Dijkstra's algorithm in both directions and then summing these node totals, we are able to determine whether a node can be on a feasible path.

For example, consider a node u. If we apply Dijkstra's algorithm in both directions to find the shortest path time, then node u will have the shortest time from S to u and from u to T. If that total is greater than the time upper bound, we may remove u since no time-feasible path can go through it.

Similarly, once we have a feasible solution with a path cost of z, we may compute the shortest path costs in both directions and any node u for which the minimum cost from S to u plus the minimum cost from u to T is greater than z may be eliminated.

Finally, once we have an optimal value for λ and $z(\lambda)$, we may calculate the shortest paths in both directions in terms of $z(\lambda)$ to S and T. Any node for which this total $z(\lambda)$ is greater than z we may eliminate.

Removing nodes in this manner is called *cost-based filtering.*

2.3.6 Multiplier Search

The purpose of the multiplier search is to find the value of λ which maximises the value of $z(\lambda)$ in Equation(2.7). For example, Figure 2.1 shows the values for z and $z(\lambda)$ as a function of λ for a typical case. The minimum value for z corresponds to a maximum value for $z(\lambda)$. The method we use is a modification of Kelley's cutting plane algorithm (Kelley (1960)) reported by Handler and Zang (1980).

The y-intercept of $z(\lambda)$ at λ=0 corresponds to z_{min}, the unconstrained shortest path cost. If we use a sufficiently large value of λ we can guarantee that $z(\lambda)$ will be less than z_{min}. Let us call this λ_{max}.

To find λ_{opt} we first get initial solutions using λ=0 and λ=λ_{max}. By using the edges of these two solutions we may approximate the tangent line to the $z(\lambda)$

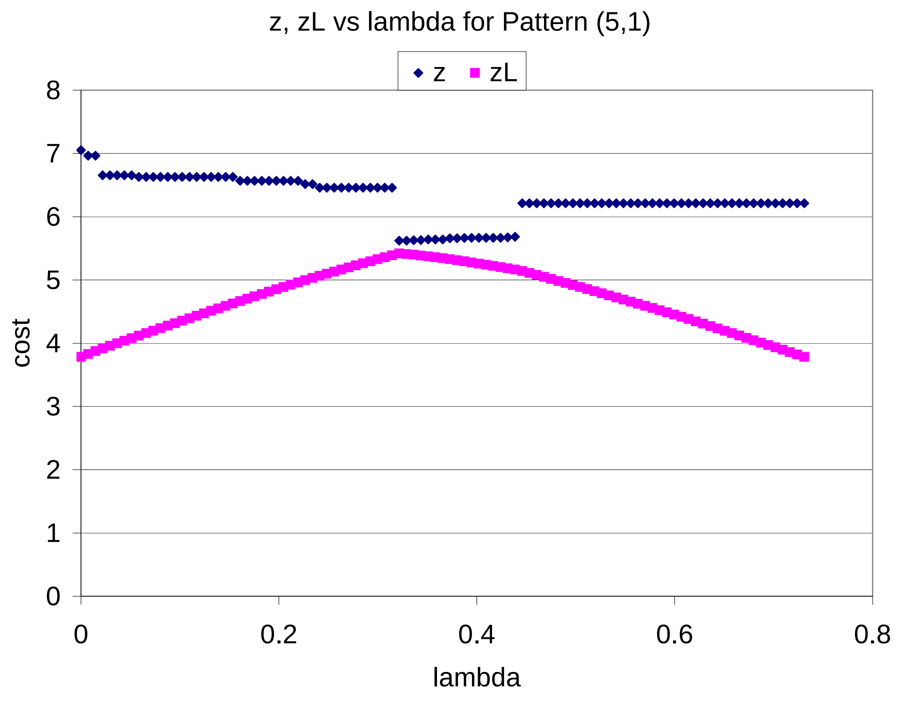


Figure 2.1: Initial Path Cost vs λ

curve at these points by allowing λ to vary. The intersection of these two lines then provides an updated value for λ_{opt}.

A straight line is given by $y = mx + b$, and the intersection point of two lines (where $y_1 = y_2$) is given by

$$m_1x + b_1 = m_2x + b_2 \tag{2.8}$$

or

$$x = \frac{(b_1 - b_2)}{(m_2 + m_1)} \tag{2.9}$$

Since b_1 and b_2 are simply the y-intercepts when x=0, this amounts to the path cost z which we calculate for our two solutions by setting λ=0. The slopes of the two tangent lines at λ=λ_{max} are given by

$$m_1 = \frac{(y_1 - b_1)}{\lambda_{max}} \tag{2.10}$$

$$m_2 = \frac{(y_2 - b_2)}{\lambda_{max}} \tag{2.11}$$

Using these values in (2.9) gives the new value for λ. We set the edge costs with this new λ and calculate a new $z(\lambda)$ path. Using the above method, we find the slope of this line and decide whether to update the left hand (positive slope) or right hand (negative slope) value of λ and initial path. This process is repeated until λ does not change. In practise this may take up to 8 iterations, but usually 4 to 6 iterations are sufficient. We thus have a fast method for calculating λ_{opt}.

2.3.7 Bounded Search

Carlyle et al. (2007) use a bounded search method to close the duality gap between the lower bound of $z(\lambda)$ and the upper bound $\bar{z}$. Although this type of search can have exponential time complexity, they describe an innovative approach which combines the constraints to further reduce the search time.

First, we describe the basic path enumeration algorithm. The initialisation steps are:

1. Computing the minimum distance $d(v)$ for each $v \in V$ to T with respect to the modified edge costs $c'_{ij} = c_{ij} + \lambda f_{ij}$
2. Computing the minimum distance $d_0(v)$ for each $v \in V$ to T with respect to edge costs c_{ij}
3. For each constraint $i \in I$, computing the minimum distance $d_i(v)$ for each $v \in V$ to T with respect to edge weights f_i

Let $E_P(u) = \{(s, v_1), (v_1, v_2), ..., (v_k - 1, u)\}$ denote a directed $s - u$ subpath. The path enumeration begins at S but extends subpath $E_P(u)$ along edge $e = (u, v)$ if and only if the following conditions hold:

1. $E_P(u) \cup \{e\}$ can be extended to a path for which the Lagrangianised length does not exceed $\bar{z}$, that is, $L(u) + (c_e + \sum_{i \in I} \lambda_i f_{ie}) + d(v) \leq \bar{z}$, where $L(u)$ denotes the Lagrangianised length of $E_P(u)$ and where, by convention, we define $L(s) = -\lambda\mathbf{g}$.

2. $E_P(u) \cup \{e\}$ can be extended to a path for which the true length is strictly less than $\bar{z}$, that is, $L_0(u) + c_e + d_0(v) < \bar{z}$, where $L_0(u)$ denotes the length of $E_P(u)$.

3. For all $i \in I$, $E_P(u) \cup \{e\}$ can be extended to a path for which the i-th weight does not exceed g_i, that is, $L_i(u) + f_{ie} + d_i(v) \leq g_i$, where $L_i(u)$ denotes the i-th total weight of $E_P(u)$.

This algorithm is actually a branch-and-bound procedure which uses a depth-first enumeration along with feasibility checks. The partial path is stored in a stack, and as the feasibility checks are performed the path is extended or contracted. Whenever the algorithm extends the path from S to T and an improved cost is found, $\bar{z}$ is updated which further limits the search. The lower bound $z(\lambda)$ is not updated.

Once an optimal λ is found, the path enumeration procedure checks that the following constraints are satisfied.

$$\mathbf{cx} < \bar{z} \tag{2.12}$$

$$-\lambda\mathbf{g} + (\mathbf{c} + \lambda F)\mathbf{x} \leq z(\lambda) \tag{2.13}$$

$$F\mathbf{x} \leq \mathbf{g} \tag{2.14}$$

The path is not extended along an edge, say $e' = (u, v)$ if doing so would violate any of the constraints (2.12), (2.13) and (2.14).

Let us label equations (2.12), (2.13) and (2.14) as [A], [B] and [C], respectively. These constraints may be combined with multipliers to π_i further limit path enumeration. The combinations possible are given in Equations (2.15) through (2.19).

$$\pi_1[\mathrm{C}] \tag{2.15}$$

$$\pi_2[\mathrm{A}] + \pi_2[\mathrm{B}] \tag{2.16}$$

$$\pi_3[\mathrm{A}] + \pi_1[\mathrm{C}] \tag{2.17}$$

$$\pi_3[\mathrm{B}] + \pi_1[\mathrm{C}] \tag{2.18}$$

$$\pi_3[\mathrm{A}] + \pi_3[\mathrm{B}] + \pi_1[\mathrm{C}] \tag{2.19}$$

The values for π_i reported in Carlyle et al. (2007) are $\pi_1 = (1/g_1...1/g_{|I|})$, $\pi_2 = 1$, and $\pi_3 = 1/z(\lambda)$. In this thesis we only consider problems with a single side constraint, thus $|I| = 1$, and (2.15) is redundant.

Substituting these values for π_i into the above equations yields the following additional path feasibility checks:

$$\begin{aligned} UB + LB &\leq 2\bar{z} && (2.20)\\ UB + \frac{TBz(\lambda)}{g} &\leq \bar{z} + z(\lambda) && (2.21)\\ LB + \frac{TBz(\lambda)}{g} &\leq \bar{z} + z(\lambda) && (2.22)\\ UB + LB + \frac{TBz(\lambda)}{g} &\leq 2\bar{z} + z(\lambda) && (2.23)\\ & && (2.24) \end{aligned}$$

where

$$\begin{aligned} UB &= L_0(u) + c_e + d_0(v) && (2.25)\\ LB &= L(u) + (c_e + \lambda f_e) + d(v) && (2.26)\\ TB &= L_1(u) + f_e + d_1(v) && (2.27) \end{aligned}$$

2.3.8 Computing Platform

All software for this work was developed using Microsoft Visual C++ version 6, and run on a 3.0 GHz Pentium 4 desktop computer with 2 Gb of RAM running Windows XP Service Pack 2.

Chapter 3

Submarine Transit Path Problem

This chapter explores the application of Lagrange Relaxation (LR) to the Submarine Transit Path problem, a constrained shortest path problem with a single side constraint. One appealing feature of this problem is that it can easily be scaled up to produce large network problems while retaining the underlying topology. The original model includes two vehicle speeds, which introduces parallel edges to the network. On this model, LR is shown to produced near optimal results in most cases.

In addition, we also consider the problem with only one vehicle speed, which has no parallel edges. In this model, LR often produces solutions with very large duality gaps, making this a problem of interest.

We examine the cause of these large duality gaps, and propose an algorithm which combines aspects of constraint programming with LR to produce a hybrid method for arriving at near optimal solutions in every case, with computation times that are up to several orders of magnitude faster than CPlex.

3.1 Problem Description

The task of planning a submarine path through a field of sonar detectors has strategic importance in marine defence. Caccetta et al. (2007) approach this problem using a 2-phase method. First, a simple heuristic is first used to generate

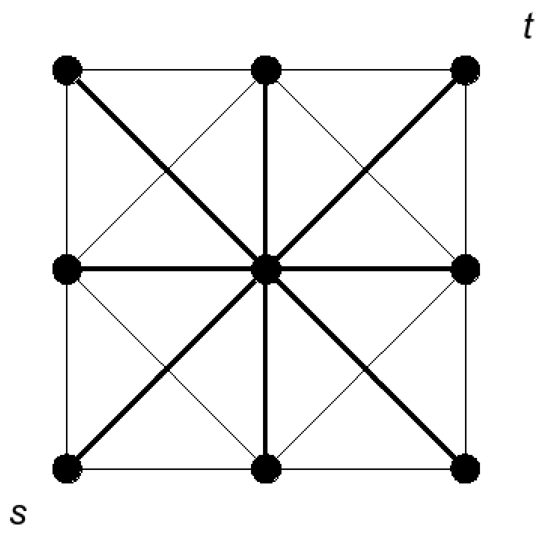


Figure 3.1: Submarine Transit Path grid

an intial path on a discretised approximation of the problem modeled with a grid network, and then an optimal control model of the original problem is solved using MISER3 (Jennings et al. (2004)). As with the aircraft flight planning problems studied by Zabarankin et al. (2006) and Carlyle et al. (2007), these types of vehicle path planning problems in a risk environment lend themselves very well to discrete modeling. We will use a similar grid network model in this work.

The field of traversal is a square area of ocean 80 km on a side. The path starts in the lower left corner and ends at the upper right corner. A grid network consisting of directed horizontal, vertical and diagonal edges is superimposed on this area. Although the submarine speed can take any value up to the maximum speed possible, in order to limit the size of the resulting network model we will restrict our analysis to a small number of discrete speed values.

We consider two possible vehicle speeds, 8 km/h and 14 km/h. Figure 3.1 shows the structure of edges at each node in the 2-D grid network. For a 2-speed Submarine Transit Path network there are 4 edges between any two adjacent nodes, in each direction an edge for the 8 km/h and the 14km/h speed. With a nodes per side, the total number of vertices is given by

$$|V| = a^2 \tag{3.1}$$

and the total number of edges is given by

$$|E| = 8(a-1)(2a-1) = 16a^2 - 24a + 8 \tag{3.2}$$

As a increases, the number of vertices is roughly proportional to the number of edges. For example, if $a = 41$ then there are 1681 nodes and 25,920 edges, and the ratio of edges to nodes is 15.42. This corresponds to a scale of 2 km per horizontal or vertical edge.

The transit field contains four identical sonar detectors placed randomly at nodes in the network. For our test purposes we use a reference set of 30 detector patterns, each containing 4 sonar detectors. The full list of detector location patterns is given in Appendix A.

Let g_{min} be the shortest possible path time, and g_{max} be the transit time of the shortest unconstrained path. We want to provide 4 different time bounds between these extremes. Each of the 30 problem sets has the same lower bound on the time constraint $g_{min} = 11.43$ hours and an upper bound g_{max} that varies by problem set but is around 19 hours. We use 4 different time constraints per detector pattern, with upper bounds g_i chosen by

$$g_i = g_{min} + \alpha(g_{max} - g_{min}) \tag{3.3}$$

where α={0.2, 0.4, 0.6, 0.8}. This gives a total of 120 test cases.

Several simplifying assumptions are made:

- The detectors are stationary
- The vehicle depth is not important
- There are no islands or other obstructions to travel or to signals
- Ocean currents, which affect speed, are ignored
- Vehicle propulsion issues (battery charge and fuel) are not considered
- The vehicle may only travel at one of 2 speeds along each edge
- There is no cost to change speeds
- All detectors share the same detector function.
- All detectors are independent, ie. do not communicate with one another

The single constraint for this problem is an upper bound on the total transit time. A solution consists of a list of nodes and edges from the start node s to the terminal node t. A feasible solution is one for which the total transit time is less than or equal to the upper time bound.

The optimisation problem is to minimise the cost function within the given transit time constraint. The cost associated with each edge is determined by the detector function, which is discussed in the next section.

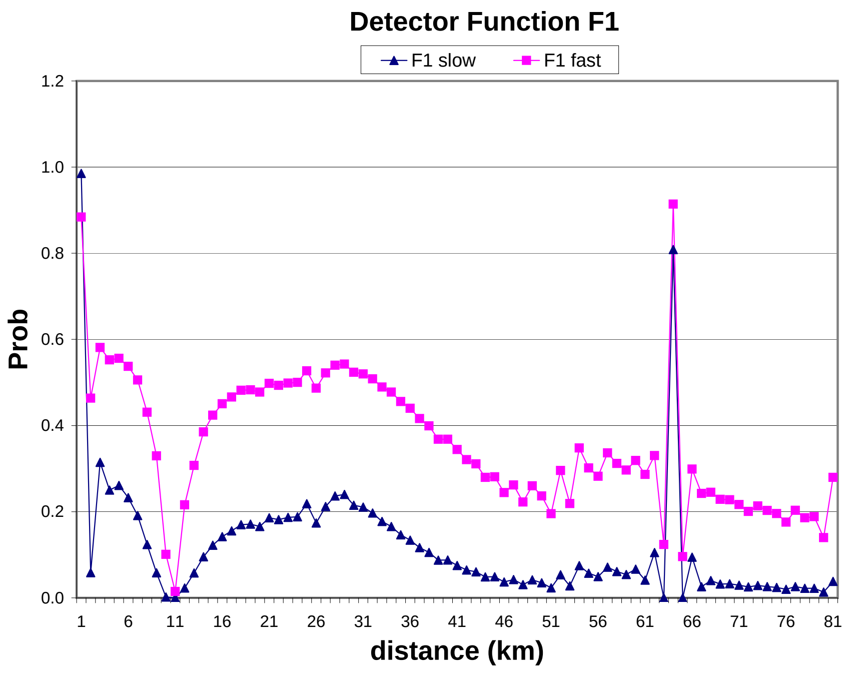


Figure 3.2: Original detector function F1

3.1.1 Detector Functions

The probability of detection depends on many factors. A passive sonar detector is essentially a microphone that picks up undersea sounds, and the transmission of sound in water depends on many factors including temperature, salinity, depth, currents and even the shape of the ocean floor. Details of how these many factors affect undersea acoustics are described in Etter (1991). Analytic models are often not available and so, in practise, values based on observed measurements are computed for discrete ranges.

One such probability function as reported by Hallam (1997) is given in Figure 3.2. This is based on the source and detector both at a depth of 6 meters, with the detector tuned to receive an acoustic frequency of 190 Hz. There are different plots for the 8 km/h (slow) and 14 km/h (fast) speeds. Obviously, the faster a submarine travels the more noise it makes, hence the higher probability for the fast speed.

The general trend is for the probability to decrease with distance, however there are several interesting features that highlight the difficulty of modeling underwater acoustics. There is a minimum near 11 km which is indicative of a dead zone, and a surprisingly sharp peak at 64 km. This peak is an example of how sound waves under water can converge.

The probability of detection at each node in the network may be computed as a function of the distance from each of the 4 detectors. Suppose that sensors are located at coordinates (x_i, y_i), $i = 1, 2, 3, 4$, and the target at point (x_t, y_t). The distance from the target to sensor i is given by

$$r_i = \sqrt{(x_t - x_i)^2 + (y_t - y_i)^2} \tag{3.4}$$

and the probability of being detected at point (x_t, y_t) by sensor i is given by the probability function in Figure 3.2 with r_i as the distance. As we have 2 possible speeds with different probability functions, let us define $D_z(r_i)$ as the probability of being detected at distance r_i while travelling at speed z, where $z = 1$ for the slow speed and $z = 2$ for the fast speed.

Assuming the detectors are independent, the combined probability of detection at a given point (x_t, y_t) for a given speed z is given by

$$P_z(x_t, y_t) = \prod_{i=1}^{4} D_z(r_i) \tag{3.5}$$

The probability fields for the first test case (pattern 1, time bound 1) for both the slow (8 kmph) and fast (14 kmph) speeds are shown in Figure 3.3 and Figure 3.4, respectively. Note how the peak in the probability function at 64 km manifests as a ring of high probability in the transit field.

There are two ways to model path risk reported in the literature. The first involves the routing of military aircraft through a region of radar detectors, and has been studied by Zabarankin et al. (2006) and Carlyle et al. (2007). In both of these approaches, the probability of detection along an edge p_e is calculated by integrating with respect to the distance travelled along the edge. In this approach, the speed of the aircraft does not affect the detection probability, only the distance along which the aircraft is exposed to the radar. The probability of *not* being detected is therefore $(1 - p_e)$.

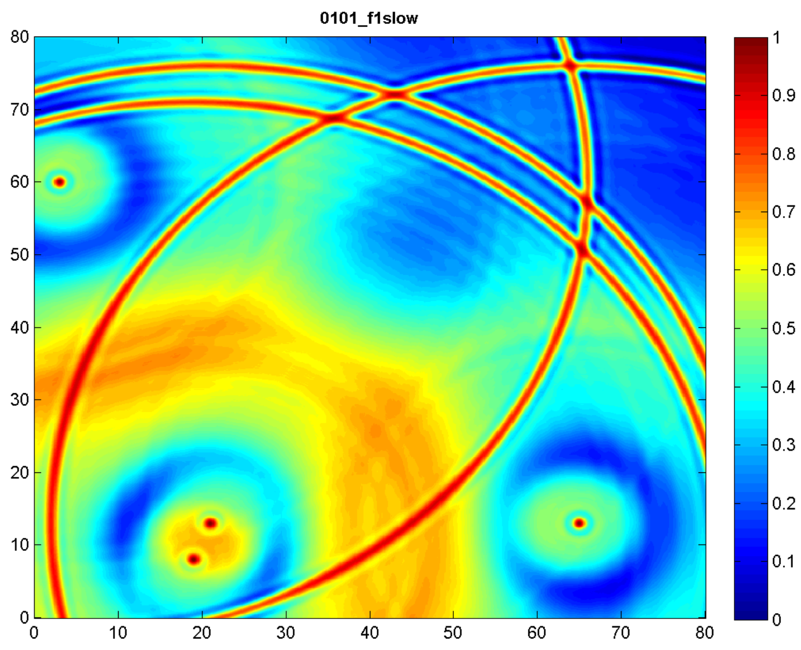


Figure 3.3: Sample probability field for 8 kmph

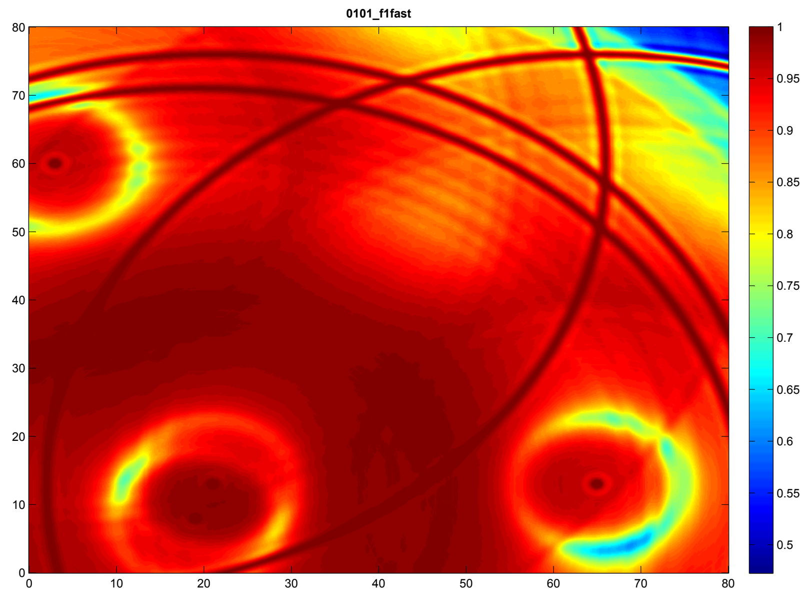


Figure 3.4: Sample probability field for 14 kmph

Assuming that the probabilities of detection for each edge are independent, the probability of mission success is then the product of the individual edge probabilities along the path. For a path from s to t using edges E, this is

$$\max P_{st} = \prod_{e \in E} (1 - p_e) \tag{3.6}$$

A logarithmic transformation turns this product into a summation given by

$$\min P_{st} = \sum_{e \in E} -log(1 - p_e) \tag{3.7}$$

since $-log(1 - p_e) \geq 0$.

The second approach to modeling path risk, the approach we will use, is reported by Hallam (1997) in solving the Submarine Transit Path problem. Unlike radar detection, the risk of sonar detection is a function of vehicle speed, as seen in Figure 3.2. Instead of integrating the probability of detection in terms of edge length, the edge probability p_e may be considered to be a function of the time spent at that probability.

We construct the edge costs c_e as follows. Equation (3.5) gives us the instantaneous probability of detection while travelling through node (x_t, y_t) at speed z. An edge e_{AB} going from node A to node B has an average probability of

$$P_{AB} = \frac{(P_A + P_B)}{2} \tag{3.8}$$

and we consider the contribution of this edge to the total cumulative detection probability to be a function of the time spent at that probability P_{AB}. If the transit time of an edge is t_e, then the edge cost c_e is given by

$$c_e = t_e p_e \tag{3.9}$$

If the total path transit time is T, then the total probability of detection along a path from s to t is given by

$$P_{st} = \frac{\sum_{e \in E} t_e p_e}{T} \tag{3.10}$$

This gives us a *time-weighted average probability.* If we assume that the path transit time is close to the upper bound on path time, and therefore constant for a given problem, we may ignore the transit time term T in the denominator.

With this one detector function, 30 detector patterns and 4 time bounds for each, we have a set of 120 test cases. We would like to expand this set and explore how variations in the detector function may affect the optimal solution.

The first step is to remove the peak at 64 km by manually replacing the unusually high and low values in this region to produce a smoother profile in this section. This produces F2 which is shown in Figure 3.5.

The next step is to apply general smoothing on F2. This is achieved by averaging each value and its immediate neighbours,

$$\overline{P_z}(r) = \frac{P_z(r-1) + P_z(r) + P_z(r+1)}{3}, r \in \{2, 3, ..., 79\} \tag{3.11}$$

This smoothing is applied twice. The values at either extreme, $P_z(0)$ and $P_z(80)$, are left unchanged. This produces the function values for F3 as shown in Figure 3.6.

Finally, we also construct a very smooth, monotonic detector function F5 as a reference (Figure 3.7). This last function F5 is similar to the detection probability of a radar used in problems of path planning for aircraft.

The effect of these different detector functions on the transit field are illustrated in Figure 3.8 through Figure 3.11. These plots show the transit field probabilities at the slow speed for detector pattern 1.

For F2 and F3 the time bounds do not need to change so we use the same time bounds as F1, but for F5 we must calculate a new set of time bounds. If g_{max} is the time of the unconstrained shortest path and g_{min} is the shortest possible path time, then our time bounds are chosen such that

$$g_i = g_{min} + \alpha(g_{max} - g_{min}) \tag{3.12}$$

where α={0.2, 0.4, 0.6, 0.8} and i={1,2,3,4}.

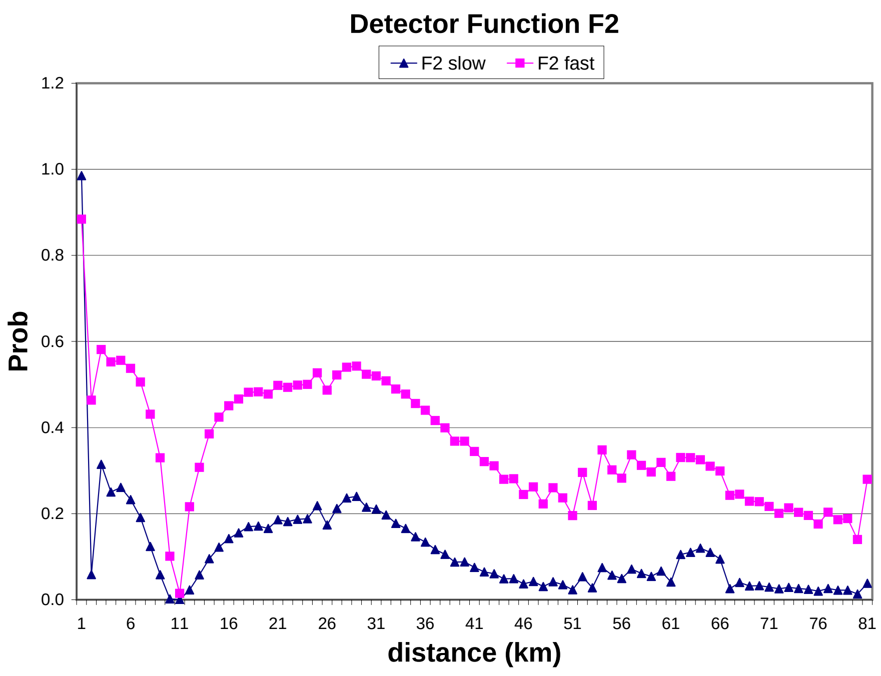


Figure 3.5: Original minus peak

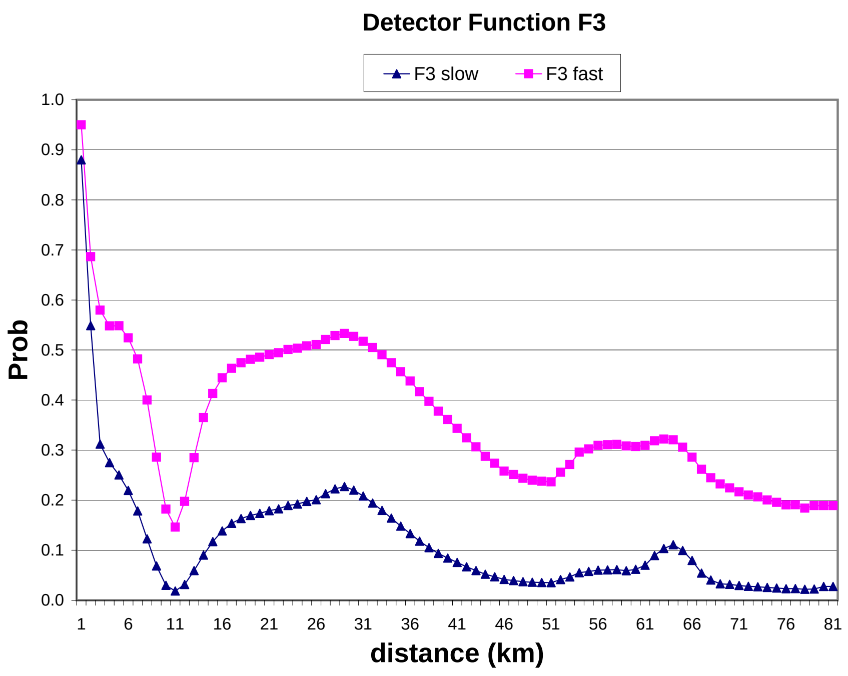


Figure 3.6: Original minus peak and smoothed

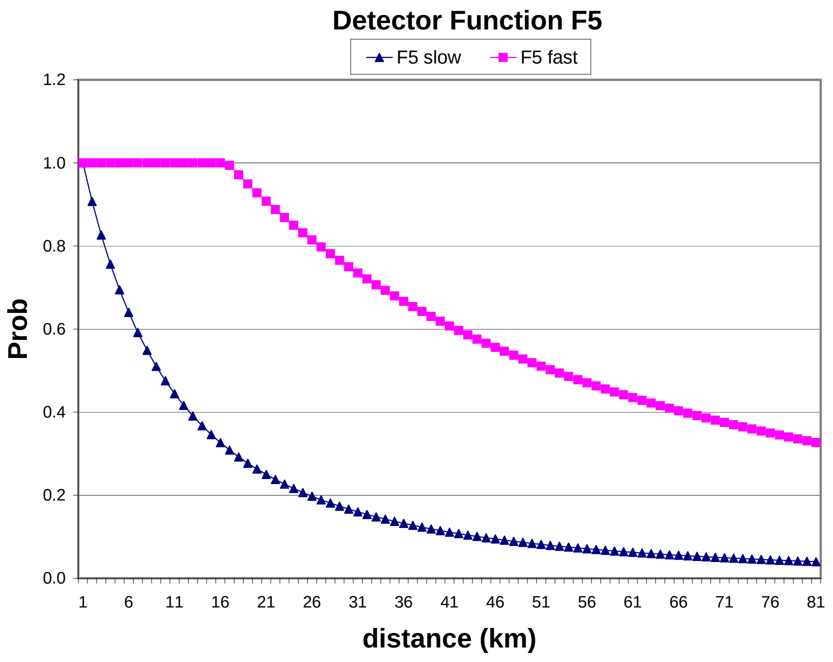


Figure 3.7: Very smooth function

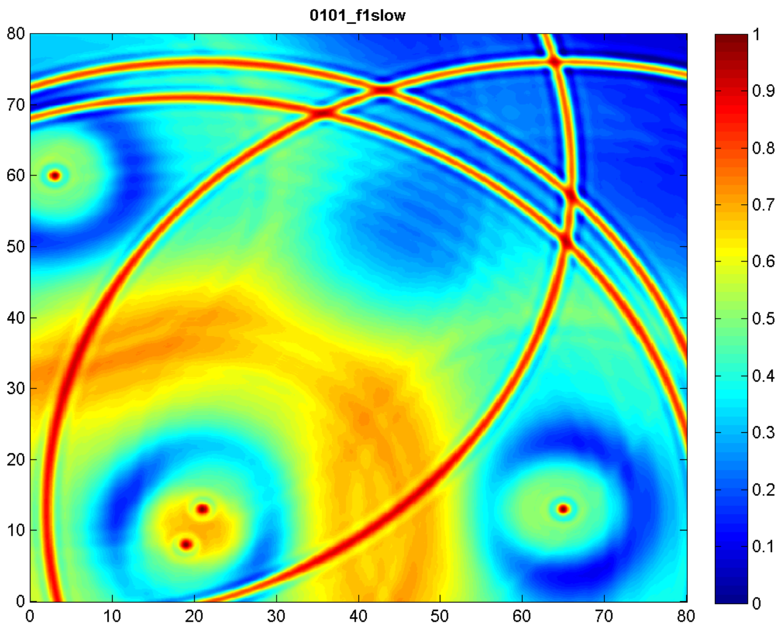


Figure 3.8: F1 sample probability field for 8 kmph

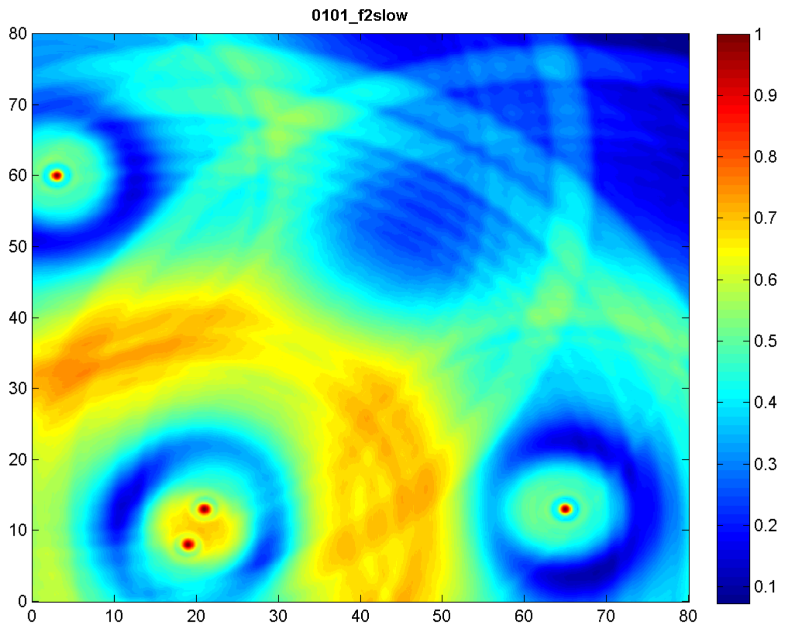


Figure 3.9: F2 sample probability field for 8 kmph

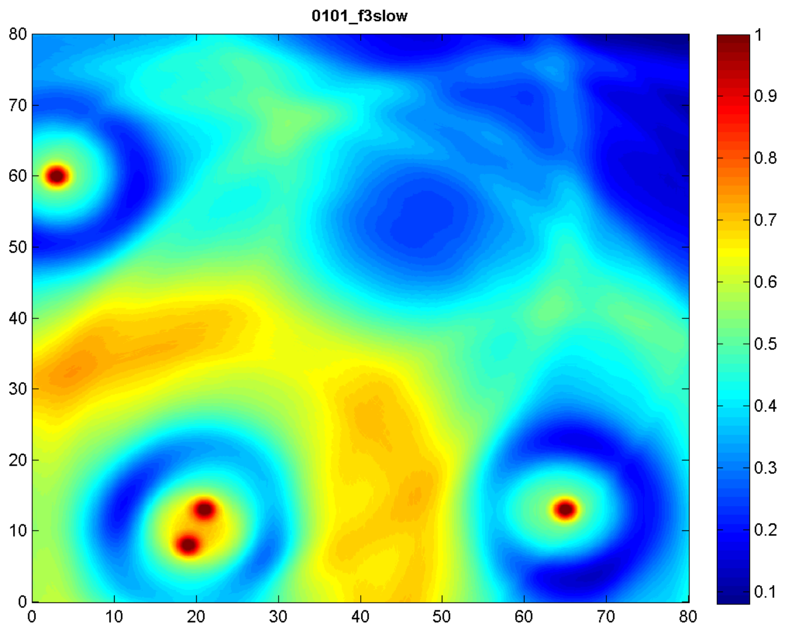


Figure 3.10: F3 sample probability field for 8 kmph

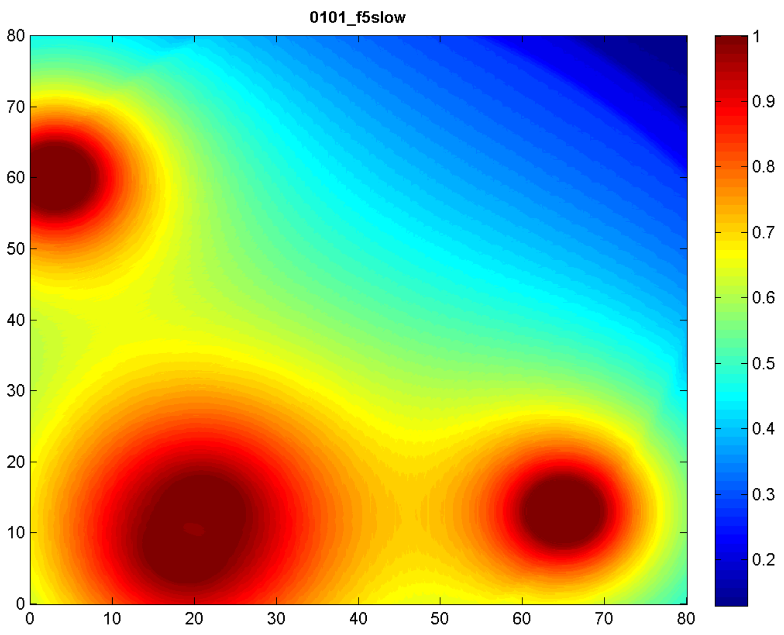


Figure 3.11: F5 sample probability field for 8 kmph

For each detector function and problem size we have 30 detector patterns and 4 time bounds, and we refer to each problem case by (pattern, time). For example, (5,3) refers to detector pattern 5 with time bound 3 (α=0.6).

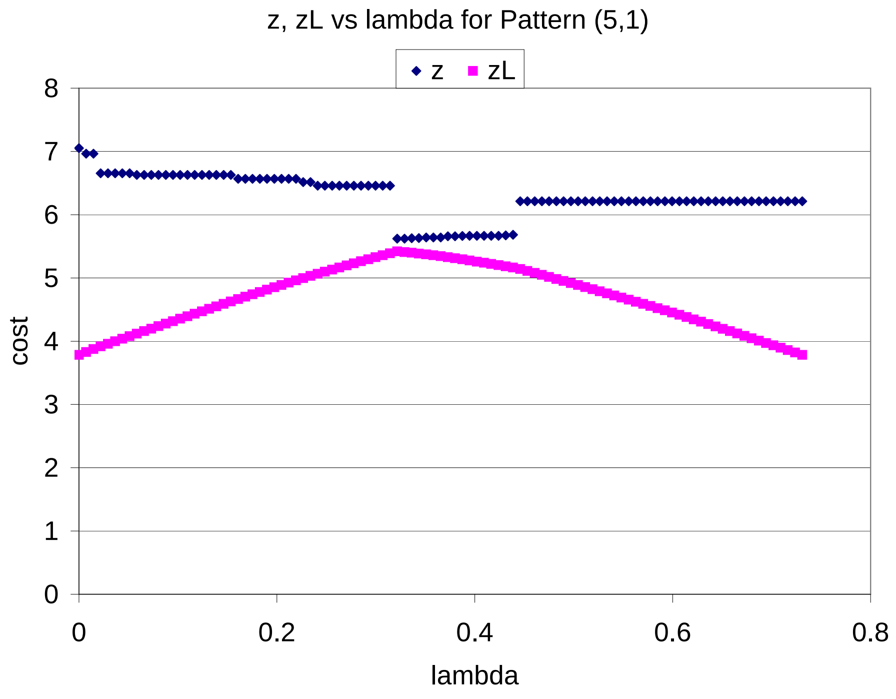


Figure 3.12: Lagrange dual $z(\lambda)$ and path cost z vs λ

3.1.2 Multiplier Search

Figure 3.12 once again shows a sample plot of the Lagrange dual $z(\lambda)$ and corresponding path cost z as a function of λ. Note how there are discontinuities in the plot for z, which indicate that the function $\sum_{e \in E} c_e$ is not convex.

For singly constrained problems such as ours, the Lagrange multiplier search using Kelley's cutting plane algorithm is very fast. However, in practise it is possible for the algorithm to cycle between two intersecting tangent lines. This is especially true if the $z(\lambda)$ curve is not smooth. Since we want to make sure that we have a value for λ that is close to optimal, we add a bisection search to ensure that the value for λ we achieve is within 1% of optimal.

3.1.3 CPlex Reference

Reference solutions were obtained using CPlex version 9 on the mixed-integer linear program (MILP) model given by (2.2) through (2.5). The edge cost values were truncated to 3 significant figures (ie. 0.000 to 0.999) in order to allow CPlex

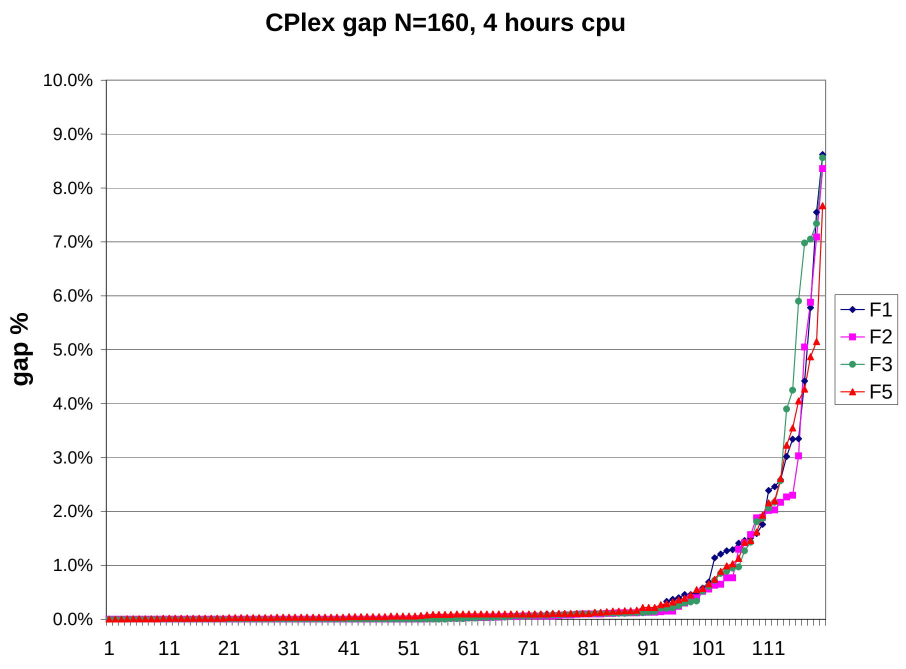


Figure 3.13: CPlex gap (sorted) for N=160 with 4 hr time limit

to converge more easily. The code for both methods was written in C++ using STL container classes, and run on a 3.0 GHz Pentium 4 desktop computer with 2 Gb of RAM running Windows XP SP2. The tolerance between the best integer solution and the lower bound was set to 0.01% and CPlex was allowed to run for a maximum of 4 hours, or until this gap was reached. Table 3.1 shows the average CPU times.

	N			
	20	40	80	160
F1	0.519	25.97	1047.5	3730
F2	0.591	11.60	870.7	3326
F3	0.656	16.25	908.6	3641
F5	0.402	6.76	1188.6	3987

Table 3.1: CPlex Average CPU Times (seconds)

For problem sizes N={20, 40, 80} the 4 hour time limit was sufficient to achieve optimal results. However, for N=160 this was not always the case. Figure 3.13 shows the CPlex gap for all detector functions for N=160. F1 has 19 cases where the gap is greater than 1%, F2 has 15 cases, F3 has 15 cases, and F5 has 16 such cases, with the worst case being just above 8%. Despite these gaps for the largest problem size, we will use these CPlex solutions as reference values, with the knowledge that the values for N=160 may not be optimal.

These reference solutions provide path costs z_{opt} which we can then use to calculate the *relative error* (RE) as

$$\text{RE} \ = 100 * (z^* - z_{opt})/z_{opt} \tag{3.13}$$

We also calculate the LR duality gap as

$$\text{LGAP} \ = 100 * (z^* - z(\lambda))/z(\lambda) \tag{3.14}$$

Both these terms will be used as percentages throughout. In addition, the networks are all square arrays with N edges per side.

3.2 Initial LR Results

3.2.1 2-speed case

We implement Lagrange Relaxation using the formulation for $z(\lambda)$ given in Equation 2.7. Dijkstra's algorithm shown in Algorithm 1 is used to calculate the shortest path using the modified edge costs provided by λ, and the multiplier search method of Section 2.3.6 is used to find the optimal multiplier λ_{opt}.

Table 3.2 shows the LR results on the 2-speed Submarine Transit Path problem. For each detector function and for N={20,40,80,160}, the average and maximum values of RE, LGAP and CPU seconds are given for each set of 120 cases.

		LR					
		%RE		%LGAP		CPU (sec)	
		avg	**max**	**avg**	**max**	**avg**	**max**
F1	**20**	0.36	5.49	1.11	11.18	0.015	0.032
	40	0.45	5.62	1.14	11.08	0.093	0.141
	80	0.35	6.16	0.96	11.69	0.446	0.656
	160	0.32	4.73	1.05	10.37	2.377	3.906
F2	**20**	0.28	5.79	1.02	9.46	0.015	0.032
	40	0.29	6.28	0.88	9.57	0.090	0.141
	80	0.31	6.36	0.84	9.53	0.414	0.672
	160	0.24	5.99	0.86	10.34	2.219	3.625
F3	**20**	0.30	5.46	1.03	9.70	0.015	0.032
	40	0.35	5.52	0.95	9.56	0.088	0.156
	80	0.27	5.54	0.83	9.65	0.417	0.672
	160	0.10	2.28	0.80	9.40	2.168	3.765
F5	**20**	0.17	3.88	1.20	10.47	0.017	0.032
	40	0.13	3.12	0.92	9.38	0.114	0.141
	80	0.10	2.54	0.84	10.03	0.573	0.687
	160	0.09	2.63	0.77	9.11	3.128	3.516

Table 3.2: LR results for 2-speed networks, all problem sets

For all problem sets, the average RE is below 0.5%, and the average LGAP is around 1%. In about one third of all cases for F1, F2 and F3 the upper time bound was not active, resulting in a lower average. Instead of the average value, we focus our attention on the maximum values. For F1, F2 an F3 the maximum %RE is generally less than 6%, and below 4% for F5. On the other hand, the

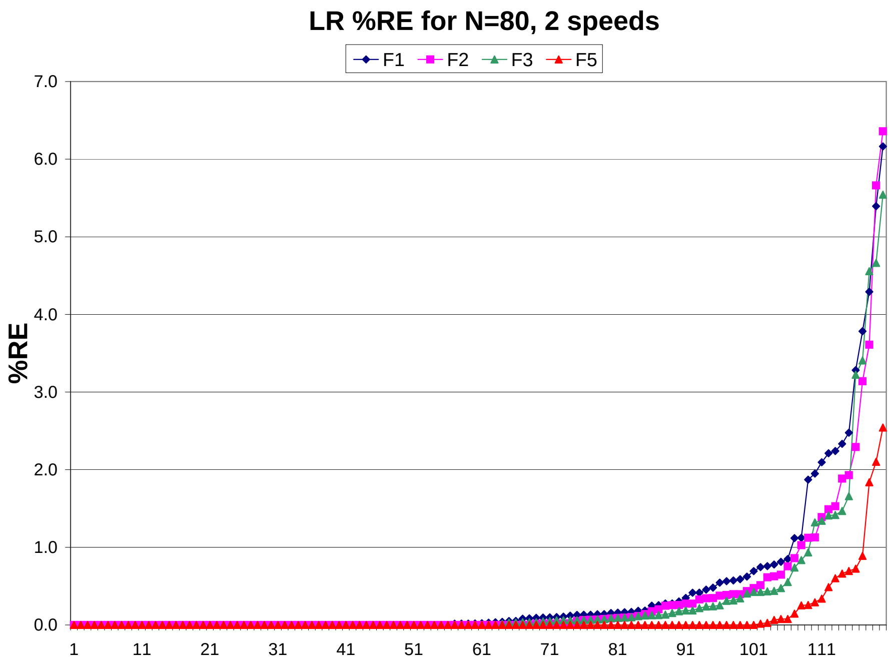


Figure 3.14: LR %RE for all functions with N=80, sorted

maximum value for the LR duality gap is around 10%, so we don't always have very strong bounds on our LR results. This illustrates the central problem of LR.

The CPU times given include the problem setup time and the LR solution. For those problems that have an active time bound, the number of calls to Dijkstra's algorithm is between 17 and 18. For the largest problems, the maximum total CPU time is less than 4 seconds. While we are not yet producing optimal solutions, these times are 3 orders of magnitude smaller than CPlex, indicating at least the potential for LR to produce optimal or near-optimal solutions in much less time.

In order to see what proportion of the problem cases have a high RE, we may plot the RE for all 120 cases for each detector function for a chosen size. Figure 3.14 shows this for N=80, sorted by ascending RE. The results for F1, F2 and F3 are very similar, indicating that the relative smoothness of these functions has only a slight affect on the optimality of LR. Only the very smooth F5 shows a significantly lower RE, the worst case of 2.54% being about half that of the other functions. Note that for all detector functions, with the exception of a small number of cases, the RE is below 3%.

Similarly, we may plot the LGAP for N=80 for all detector functions, as shown

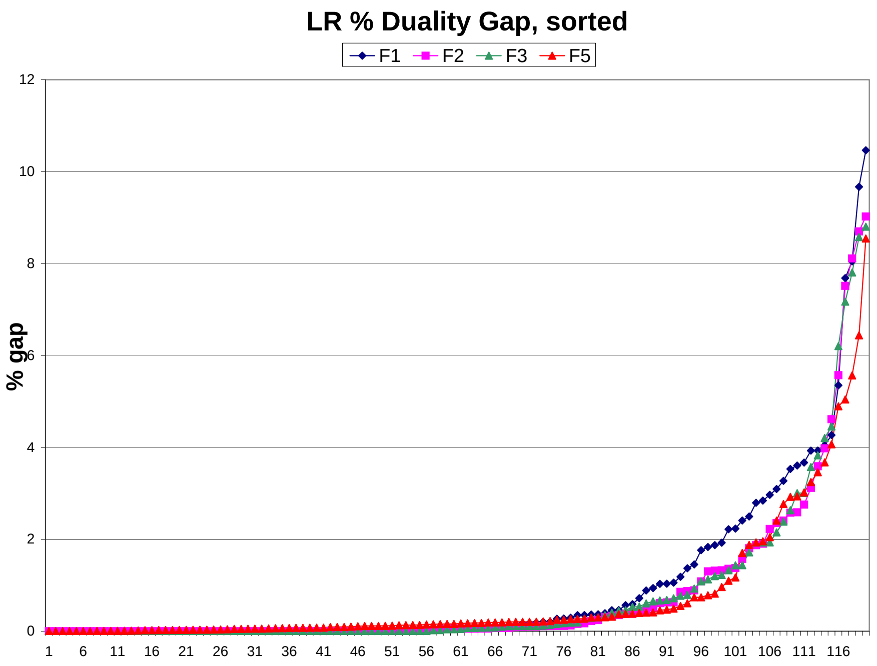


Figure 3.15: LR duality gap for N=80, sorted

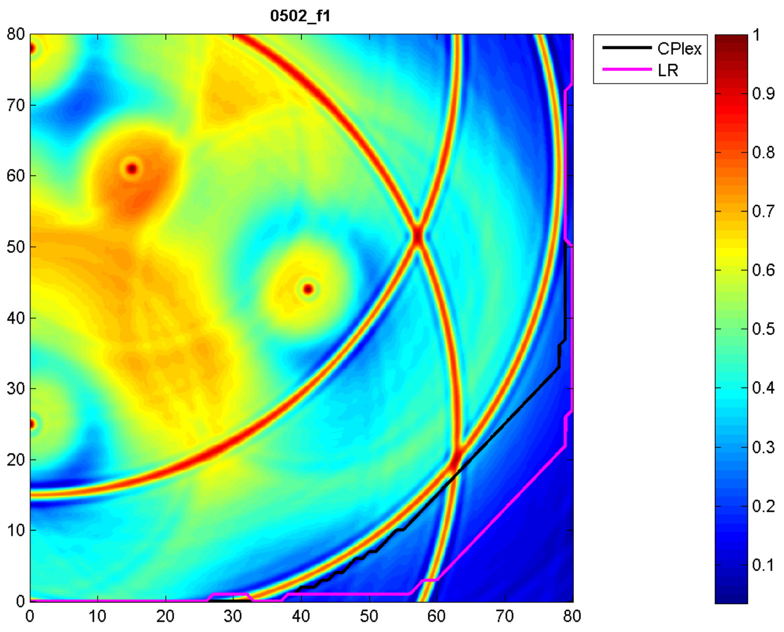


Figure 3.16: LR and CPlex paths for F1-80 (5,2)

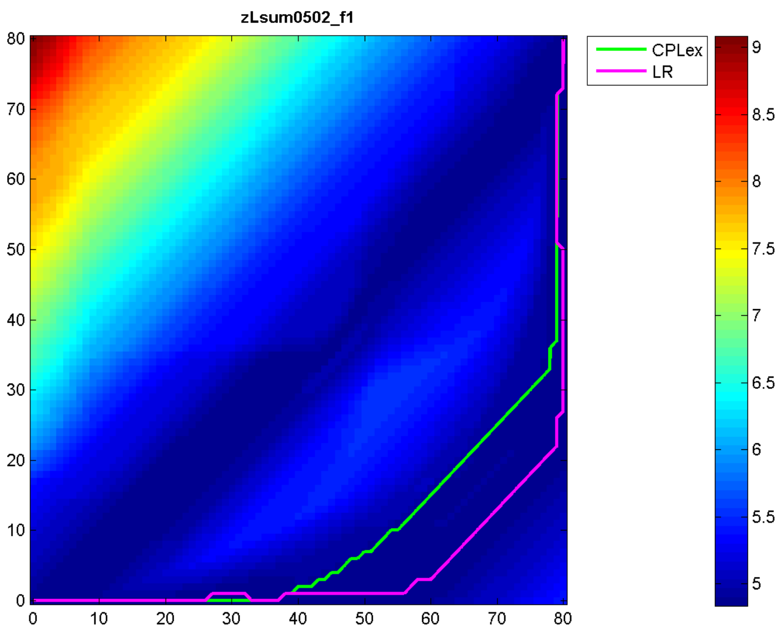


Figure 3.17: $z(\lambda)$ node sums for F1-80 (5,2)

in Figure 3.15. Again, only a small number of cases have a LGAP over 5%.

The worst case of F1-80 is (5,2), with RE=6.16%. Figure 3.16 shows the CPlex and LR paths on a plot of the transit field in terms of node probabilities, and Figure 3.17 shows the same in terms of $z(\lambda)$ node sums. The value at each node represents the sum of the minimum $z(\lambda)$ distances to S and T. Note how much smoother the plot of $z(\lambda)$ is by comparison.

Plots such as Figure 3.17 are useful for analysis because they show the regions where LR will find the best path. If we interpret the $z(\lambda)$ values as conventional 3-D topography, with high values corresponding to high elevations, then LR will find a path through one of the valleys, shown here in dark blue.

3.2.2 1-speed case

In most published works on the constrained shortest path problem, networks under study do not have any parallel edges. In fact, not having any parallel edges is a requirement. To see how much the parallel edges may be contributing to the results, we may take away all the fast edges from the network, leaving only the slow edges. To differentiate these problem sets from the 2-speed case, we add the suffix 's' to the problem size. For example, F1-80 refers to the 2-speed case, and F1-80s refers to the 1-speed case.

Of course, we then need to calculate new time bounds. If g_{max} is the time of the unconstrained shortest path and g_{min} is the time of the shortest possible path from s to t, then our time bounds are chosen such that

$$g_i = g_{min} + \alpha(g_{max} - g_{min}) \tag{3.15}$$

where α=$\{0.2,\ 0.4,\ 0.6,\ 0.8\}$. Once again, CPlex is used to generate a set of reference solutions. Table 3.3 shows the CPlex duality gap and the CPU times for all problem sets.

Fcn	N	Gap		CPU (sec)	
		avg	max	avg	max
F1	20s	0.00	0.00	0.39	3.69
	40s	0.00	0.00	9.90	134.38
	80s	0.13	5.64	2088.00	43299.10
	160s	2.04	10.83	47048.96	173013.00
F2	20s	0.00	0.00	0.21	2.63
	40s	0.00	0.00	4.71	133.41
	80s	0.18	5.40	922.39	14400.70
F3	20s	0.00	0.00	0.21	1.19
	40s	0.00	0.00	5.90	208.77
	80s	0.16	6.14	788.11	14400.60
F5	20s	0.00	0.09	0.19	1.20
	40s	0.02	0.10	2.47	32.00
	80s	0.05	2.16	389.52	14400.50
	160s	0.63	9.27	3491.47	14410.70

Table 3.3: CPlex times for 1-speed problems

For the N=160 problem size we only produce CPlex reference solutions for F1 and F5. We use a time limit of 4 hours for all problem sets except for F1-160, in which the time limit is raised to 48 hours.

Table 3.4 shows the RE, LGAP and CPU times for all problem sets. Both average and maximum values are given.

For the 1-speed problem, the maximum RE and LGAP values are significantly higher that for the 2-speed problem, while the CPU times are roughly the same. These 1-speed LR results clearly demonstrate the basic problem with Lagrange

		LR					
		%RE		%LGAP		CPU (sec)	
		avg	**max**	**avg**	**max**	**avg**	**max**
F1	**20s**	1.98	28.45	4.233	31.152	0.015	0.031
	40s	2.65	38.92	4.750	40.423	0.088	0.125
	80s	2.81	29.37	4.539	31.037	0.507	0.750
	160s	2.83	29.62	4.593	30.803	3.190	6.641
F2	**20s**	1.90	27.78	3.872	29.935	0.015	0.032
	40s	2.40	28.20	4.056	30.115	0.086	0.125
	80s	2.58	28.54	3.990	30.558	0.449	0.718
F3	**20s**	1.82	28.53	3.661	30.277	0.014	0.032
	40s	2.11	28.50	3.652	30.110	0.081	0.125
	80s	2.40	28.59	3.738	30.265	0.407	0.734
F5	**20s**	1.68	17.55	3.038	19.185	0.014	0.031
	40s	1.83	30.44	2.817	33.111	0.091	0.140
	80s	2.14	32.14	2.965	33.163	0.503	0.766
	160s	2.03	23.88	3.053	32.450	2.972	4.782

Table 3.4: LR results for 1-speed networks, all problem sets

Relaxation: the possibility for high duality gaps.

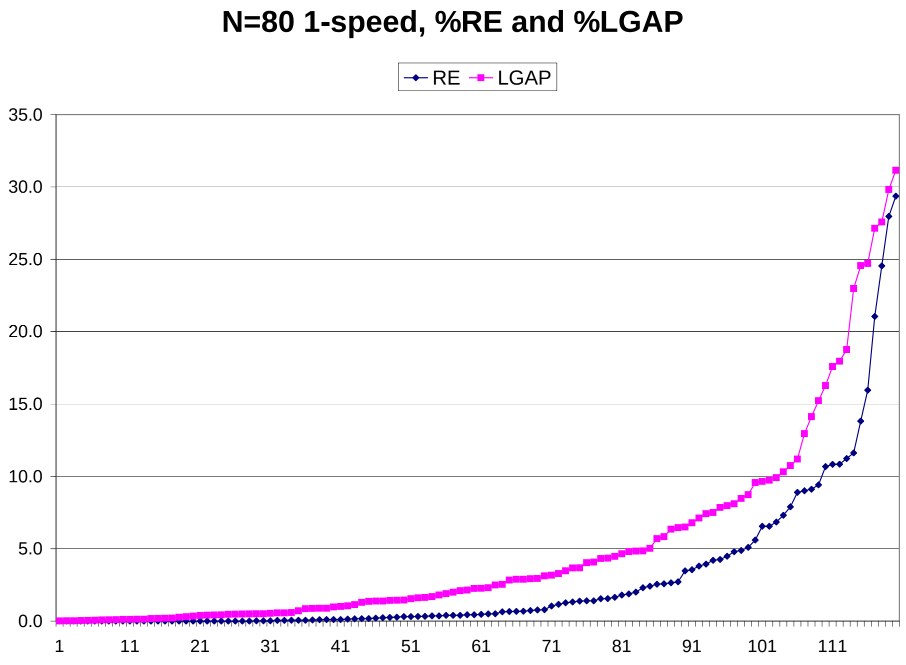


Figure 3.18: LR RE and Lgap for 120 cases, N=80, 1 speed

Figure 3.18 shows a graph of the RE and LGAP for N=80, in sorted order as before. Clearly, these results are much worse than for the 2-speed case. On the one hand, the time bounds are always active and yet we have 71 out of 120 cases where the RE is below 1%. But in 20 out of 120 cases the RE is over 5%, with the worst case being 29.37% above optimal. The parallel edges do indeed have an effect on the LR results.

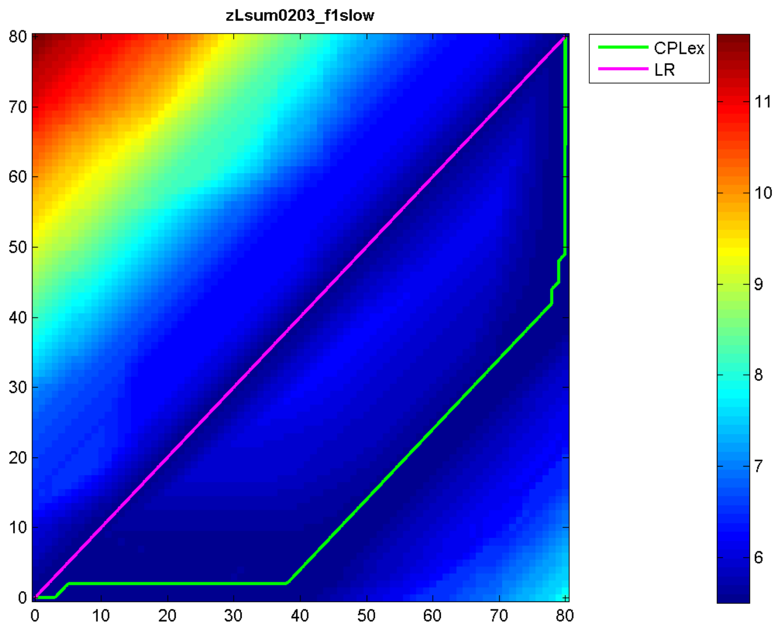


Figure 3.19: LR solution for (2,3), N=80 1-speed

Figure 3.19 shows a plot of the F1-80s worst case, (2,3), which has RE=29.37%. Note how there are two valleys or channels of low $z(\lambda)$, with the LR path going through the centre and the CPlex path going the longer way through the second channel.

3.2.3 LR Gap Problem

So far, we have applied LR to the Submarine Transit Path problem, a constrained shortest path problem with a single time constraint, using a model with 2 speeds and one with just a single speed. For the original problem containing parallel edges, the resulting solutions were generally within 3% of optimal. For the reduced network case with no parallel edges, however, the results are not nearly so good. The worst case RE rises from 6.16% to 29.37%, with almost half the cases having RE $> 3\%$.

This raises several questions. Why is it that LR works much better on the network with parallel edges than the one without? What causes LR to produce solutions with a very high duality gap?

Now that we have established that there can be a problem with LR duality gaps, the next section explores these questions and develops a hybrid method to improve the performance of Lagrange Relaxation.

3.3 Hybrid Approach

3.3.1 LR with 1-speed Network

The first thing to note is that the maximum LGAP is very high for all 1-speed problem sets, with a worst case of 40.423% for F1-40s. The corresponding maximum RE for this set is 38.92%. A typical case from F1-40s is (2,3), which has a RE=31.83%. Figure 3.20 shows the node sums for $z(\lambda)$ using the optimal value of λ for this problem case, with both CPlex and LR solutions. In this plot there are two distinct regions of min $z(\lambda)$, one straight through the centre (the path taken by LR) and another region in the lower right quadrant (the path produced

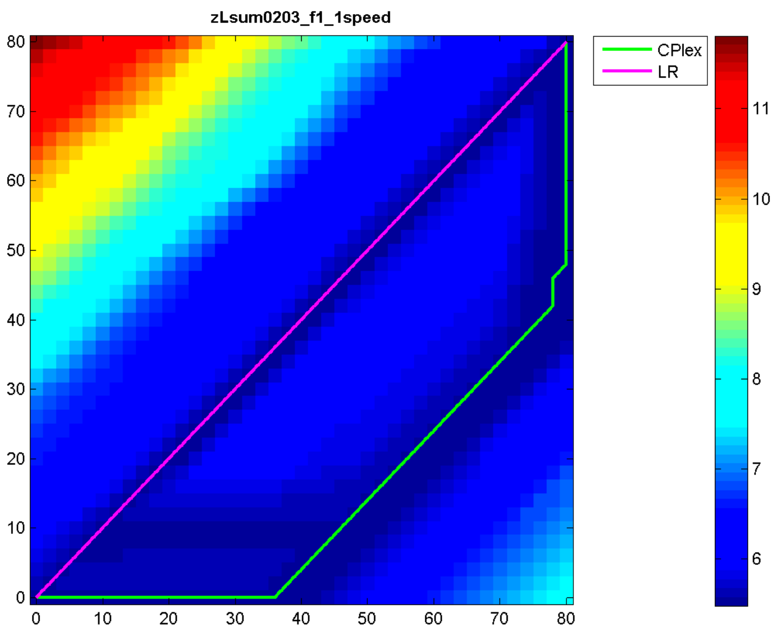


Figure 3.20: $z(\lambda)$ sums for F1-40s (2,3)

by CPlex).

The LR path has a cost of 7.181 and a transit time of 14.306 hours, while the optimal CPlex solution has a cost of 5.447 and a transit time of 16.788 hours. The upper bound on time for this problem is 16.825. The LR solution uses only 85% of the available time, but the CPlex path uses 99.8% of the time. We may call this unused time the *slack time*, and express it as a percentage of the upper time bound. Thus, the LR solution for (2,3) has a slack time of about 15%. We may also express the slack time as a fraction of the upper time bound, and call this the *slack time margin*. For example, the slack time margin for (2,3) would be 0.15.

What causes LR to find such a suboptimal solution? This is not a software bug, but a result of how LR works. Recall that each edge has a cost c and a time t, and we define the modified cost as $c^{'} = c + \lambda t$. For any given value of λ we calculate the shortest path in terms of $c^{'}$, which gives us the value $z(\lambda)$. The value of λ which gives us the highest value for min $z(\lambda)$ is considered the optimal value for λ, and the corresponding value for z^{*} is therefore correct, although it is not optimal.

To see why, note the two valleys in the $z(\lambda)$ plot of Figure 3.20. LR calculates the min $z(\lambda)$ solution as the one going straight through the centre, where the min $z(\lambda)$ value is 5.35571. The second valley, where CPlex finds the optimal path, has a slightly higher min $z(\lambda)$ value of 5.42019. In searching for a global minimum for z, we instead get a local minimum. This outcome is the same for all of the 1-speed cases where the RE and LGAP are both very high.

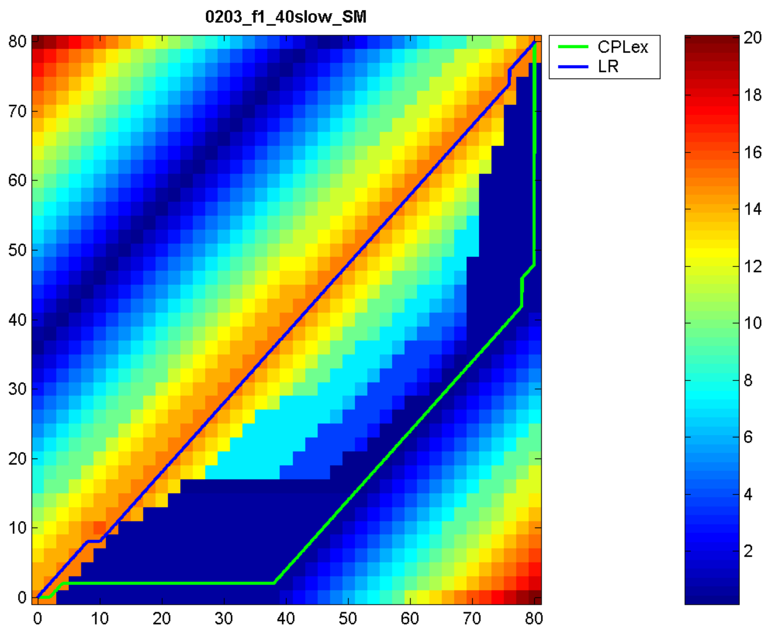


Figure 3.21: Slack time for problem F1-40s (2,3)

Let us look now at the plot of slack time for our sample problem F1-40s (2,3). By applying Dijkstra's algorithm in both directions on the modified edge costs, we have been able to calculate, for each node v, the total of the min $z(\lambda)$ paths from v back to S and from v forward to T. Our simple modification to Dijkstra's algorithm also stores the corresponding totals for time. Thus we are able to plot the slack time through every node in the network.

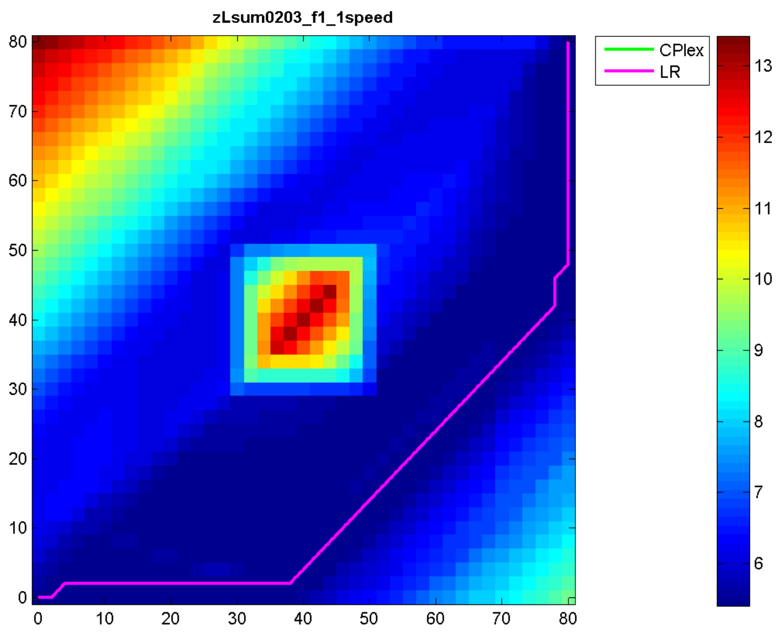


Figure 3.22: $z(\lambda)$ sums for (2,3) with island

Figure 3.21 shows such a plot of slack time for our (2,3) problem from F1-40s. To avoid negative numbers, we plot the absolute value of the slack time as a percentage of the time upper bound.

Comparing this plot with Figure 3.20, notice how the area of high slack time through the centre (where the LR path is) corresponds to the valley of min $z(\lambda)$ through the centre, and that the valley with the slightly higher $z(\lambda)$ valley (where the CPlex path is) has a much lower slack time.

A simple experiment is proposed: let us introduce an artificial island of high cost to the middle of the network, effectively blocking any min $z(\lambda)$ path through the centre. If we make this island large enough, we make it impossible for an LR path to lie in that region. We then apply the LR method as before on this modified transit field.

The result, shown in Figure 3.22, is striking. With LR applied to this modified network we achieve an optimal result, with the CPlex path and the LR path overlapping completely.

We can see from this example how LR fails to find an optimal solution because of a local minima. The slack time along the resulting LR path is very high. We may be able to use this slack time information to determine whether LR has fallen into a local minima. Although this is only one case, we see the possibility to achieve optimal results on a network which has been reduced to avoid non-optimal paths.

It seems reasonable to assume that the optimal path is one which uses as much of the time allowed in order to provide the lowest path cost. We may consider adding an additional penalty function to give a weight to the unused time, but this approach involves calculating yet another multiplier. We will explore another approach, one that keeps the original LR formulation but temporarily modifies the network based on slack time.

The next section looks at the slack time in more detail.

3.3.2 Slack Time and the Duality Gap

What we know about the Submarine Transit Path problem so far is that the LR method often produces paths which pass through areas of the field where the slack time (as we can calculate it) is high, and the optimal paths pass through areas where this slack time is lower. If we were able to apply a constraint on the amount of slack time allowed, we may then use LR to perform constraint propagation.

If we go back to our definition of z^* and $z(\lambda)$, we might see how to formulate the duality gap in terms of slack time.

$$\begin{aligned}\text{duality gap} &= z^* - z(\lambda)\\ &= \sum_{(i,j)\in V} c_{ij}x_{ij} - \Big(\sum_{(i,j)\in V}(c_{ij}+\lambda f_{ij})x_{ij} - \lambda\mathrm{g}\Big)\\ &= \lambda\mathrm{g} - \sum_{(i,j)\in V}\lambda f_{ij}x_{ij}\\ &= \lambda\Big(\mathrm{g} - \sum_{(i,j)\in V} f_{ij}x_{ij}\Big)\\ &= \lambda(\text{slackTime}) \qquad (3.16)\end{aligned}$$

From this we see that, for a given LR path using the optimal value of λ, the duality gap is a function of λ and the slack time. An easy assumption would be that to minimise the duality gap all we have to do is minimise the slack time. However, there are two problems with this. The first is that it is not sufficient to simply minimise the slack time. Inspection of the CPlex reference solutions for the F1-40s data set shows that the slack time varies from 0.01% to 13.02%, with an average of 1.62%, so it is possible to have an optimal solution that has a non-zero LR duality gap.

Remember that the core idea of LR is to balance the edge costs with the edge times subject to the upper bound on the total time. A path that tries to use *all* of the allowed time may fail to minimise the path cost. Secondly, further inspection of the CPlex paths reveals that the slack time is only minimal along part of the path, and generally in the centre of the transit field.

Is there a general correlation between the LR duality gap and the slack time? Figure 3.23 shows a plot of the LR gap and the slack time for problem set F1-80s. Although there is quite a spread in the data points, for slack time values over 2% there seems to be a rough correlation, enough to suggest that if were to reduce the slack time of a solution we might also be able to lower the LR duality gap, and thus provide tighter bounds on the solution.

In order to explore this approach, we need to find a way to use the slack time to constrain the network.

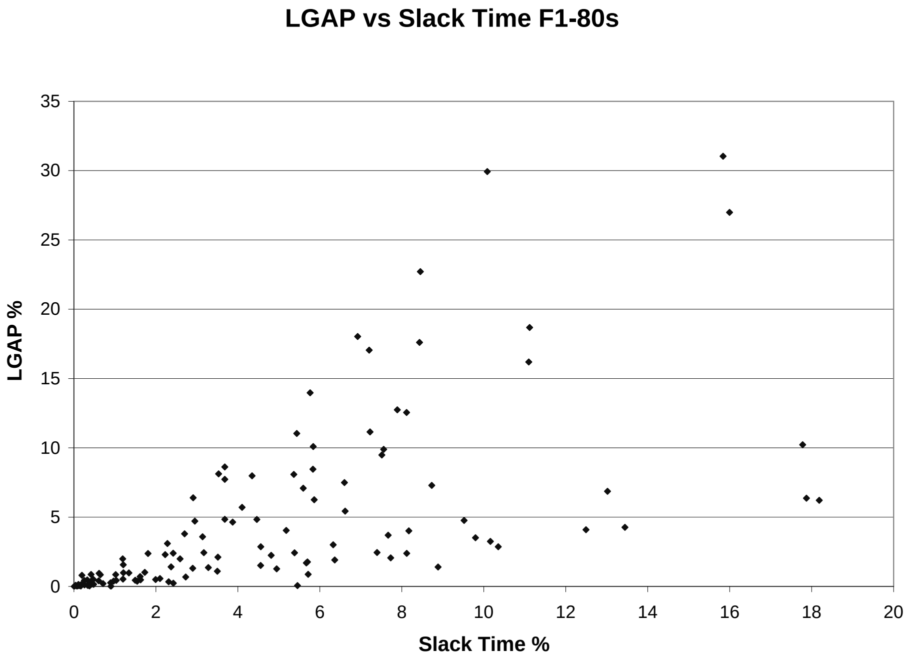


Figure 3.23: LR duality gap vs slack time for F1-80s

3.3.3 The Hybrid LR-CP Approach

If we look again at the slack times of the transit field in Figure 3.21 we see that the lowest values tend to appear in the middle region, and the highest values are found in the upper left and lower right corners. The slack time values near the origin S and the terminus T don't appear to be very useful. Therefore we focus our attention on the line that bisects the transit field through the middle from the upper left to the lower right, and call this the *meridian*.

For a square grid with N nodes per side, we define meridian nodes as those for which $x + y = N - 1$, which describes a line from (0,N-1) to (N-1, 0). Since we also have diagonal edges in this network which might pass through this line of nodes, we also add the row of nodes on either side, defined by $x + y + 1 = N - 1$ and $x + y - 1 = N - 1$. After initially applying LR to the network and summing the slack times at each node, we can then calculate the slack time for nodes along this meridian. By disabling or enabling nodes on this meridian according to a given slack time threshold, we can then constrain the crossing point of the LR path.

We then select a range of slack time threshold or slack margin (SM) values be-

Slack Time Margin	Path Cost
0.200	7.181
0.190	7.181
0.180	7.181
0.170	7.181
0.160	7.181
0.150	7.181
0.140	7.299
0.130	7.299
0.120	6.645
0.110	6.645
0.100	6.645
0.090	7.047
0.080	6.645
0.070	**5.447**
0.060	**5.447**
0.050	**5.447**
0.040	**5.447**
0.030	**5.447**
0.020	**5.447**
0.010	**5.447**
0.009	**5.447**
0.008	**5.447**
0.007	**5.447**
0.006	**5.447**
0.005	**5.447**
0.004	**5.447**
0.003	**5.447**
0.002	**5.447**

Table 3.5: Path cost vs Slack Time Margin for F1-40s (2,3)

tween 0.20 and 0.001 of the upper time bound g. The 29 values used (with the exception of 0.001) are shown in Table 3.5.

Starting with the highest threshold of 0.20, we disable any meridian nodes which have a slack time greater than $0.20g$, and then apply the LR method to the modified network. For each successive threshold value we find the resulting LR path and store it. At the point where the LR path is no longer within the time bound or no feasible path exists, we stop. At the end of this process we select the best solution from the ones stored and using the cost of this path as the the z upper bound, we filter out nodes which are suboptimal. Applying LR then finds the final path.

As this approach is similar to Constraint Programming (CP), in which the domain of the problem (in this case, the nodes and edges of our network) is reduced by propagating constraints (such as the limit on slack time), we therefore call this the LR-CP approach. The algorithm is shown in Algorithm 2.

Algorithm 2 LR-CP method

```
 1: Apply LR, store solution
 2: Set initial threshold to 0.20
 3: repeat
 4:     disable meridian nodes with slack time greater than the threshold
 5:     Apply LR, store solution
 6:     Decrease slack time threshold to next value
 7: until (no feasible solution possible) OR (SM = 0.001)
 8: Get best solution cost $z$
 9: Enable all meridian nodes
10: Apply LR
11: Remove any nodes which have $z(\lambda) > z$
12: Apply LR
13: Report solution
```

Note that we temporarily disable meridian nodes while progressively constraining the slack time along the meridian. When all the thresholds have been applied, we re-enable all meridian nodes and use the lowest path cost found to filter out nodes which we now know to be infeasible based on $z(\lambda)$.

Table 3.5 shows the LR path cost for each threshold value for problem F1-40s (2,3). Notice how the path cost both increases and decreases as the threshold varies, finally reaching a minimal value with a threshold of 0.07. The thresh-

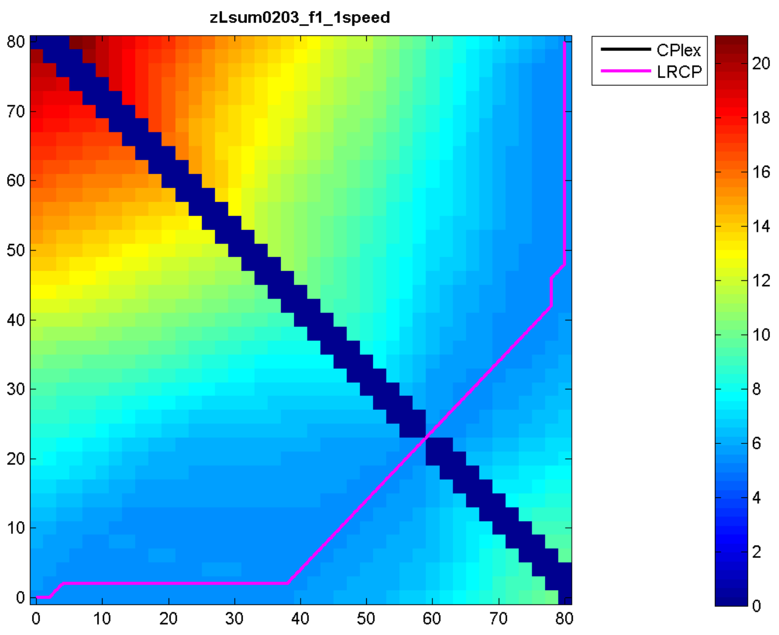


Figure 3.24: Optimal solution using LR-CP on F1-40s (2,3)

old 0.001 is not shown because no solution is possible when the slack time is constrained this much. Figure 3.24 shows the resulting LR-CP path, which is optimal. The meridian is shown only to illustrate its location in the transit field.

Applying LR-CP with 29 thresholds on all 120 problems for F1-40s, we get an average RE of 0.0087% and a maximum of 0.533%. This is a dramatic improvement over LR, which had an average and maximum RE of 2.65% and 38.92%, respectively. In addition, the average cpu time for LR-CP is 0.096 seconds, only slightly higher than the average of 0.088 seconds for LR.

The LR-CP method has been shown to work well on this one problem set of F1-40s. In the next section we apply it to all 1-speed problem sets.

3.3.4 Results of LR-CP on 1-speed Network

Before discussing the full set of results, a comment on the 1-speed reference values generated by CPlex is in order. As with the 2-speed network, the CPlex reference solutions for N=20 and N=40 are solved optimally, with no duality gap. For N=80 there are a few cases where CPlex did not run to completion within the time limit of 4 hours, and for N=160 a significant number of solutions still had a duality gap of more than 3% after 4 hours. It was decided to keep the 4 hour time limit for CPlex, and use these reference solutions as is (we will revisit the optimality of these solutions in the final chapter). The resulting reference solutions for the N=160 problem set are therefore not all optimal, and a negative %RE indicates those cases where LR-CP is able to find better solutions than CPlex did in the time allowed.

Looking then at the 1-speed networks using the all detector functions and the sizes N={20,40,80,160}, the results of LR-CP with 29 thresholds are summarised in Table 3.6.

Fcn	Size	%RE		%LGAP		CPU (sec)	
		avg	max	avg	max	avg	max
F1	**20s**	0.000	0.000	0.067	1.928	0.019	0.047
	40s	0.009	0.533	0.119	1.721	0.112	0.343
	80s	0.016	0.896	0.097	1.888	0.826	3.782
	160s	-0.107	1.053	0.094	2.347	5.137	19.625
F2	**20s**	0.000	0.000	0.033	1.875	0.017	0.063
	40s	0.006	0.536	0.035	0.842	0.103	0.391
	80s	0.037	1.689	0.060	1.689	0.630	3.282
F3	**20s**	0.000	0.000	0.024	1.144	0.015	0.032
	40s	0.000	0.000	0.017	0.544	0.095	0.375
	80s	0.011	0.989	0.027	1.102	0.535	2.063
F5	**20s**	0.000	0.000	0.000	0.022	0.017	0.047
	40s	0.000	0.000	0.000	0.001	0.100	0.187
	80s	-0.001	0.000	0.000	0.001	0.603	1.515
	160s	-0.188	0.000	0.000	0.007	3.383	9.625

Table 3.6: LR-CP results for 1-speed networks, 29 thresholds

Comparing these results to the LR results of Table 3.4, we see a remarkable improvement. The maximum RE has dropped from around 30% to no more than 1.689% for F2-80s, the LGAP has been reduced from a maximum of 40.423% to 2.347% for F1-160s, and the average CPU time has only been increased by about 50%. Note the negative values for average %RE, which are due to the CPlex reference values not being optimal.

With the LGAP less than 3% in all cases, we can say that these results are near-optimal. The next step is to see if we can reduce the CPU time while retaining the low LGAP. That is the subject of the following section.

3.3.5 Refinements

With every slack time threshold we use we must apply LR, which means optimising the value for λ, and if our goal is to minimise the CPU time then we need to question how many thresholds are really necessary. The original set of thresholds has 29 values. We may reduce this set by about half to 15, again to 8 and finally to 4. Table 3.7 shows the four resulting SM threshold sets, with SM29 being the original one.

SM29	SM15	SM08	SM04
0.200	0.200	0.200	
0.190			
0.180			
0.170	0.175		
0.160			
0.150	0.150		
0.140			
0.130			
0.120	0.125		
0.110			
0.100		0.100	0.100
0.090			
0.080			
0.070	0.075		
0.060			
0.050	0.050	0.050	0.050
0.040			
0.030			
0.020		0.025	0.025
0.010	0.010	0.010	
0.009			
0.008			
0.007	0.0075		
0.006			
0.005	0.005	0.005	0.005
0.004	0.004		
0.003	0.003	0.003	
0.002	0.002		
0.001	0.001	0.001	

Table 3.7: Slack time threshold sets

We are interested in what effect these different sets have on RE and the CPU time. We therefore apply the LR-CP method with each threshold set, and plot the results for F1 across all network sizes. The results are shown in Table 3.8 and Table 3.9.

	%RE	N20	N40	N80	N160
LR	avg	1.98	2.65	2.81	2.83
	MAX	**28.45**	**38.92**	**29.37**	**29.62**
SM04	avg	0.00	0.06	0.02	-0.06
	MAX	**0.56**	**3.82**	**1.12**	**2.40**
SM08	avg	0.00	0.01	0.01	-0.10
	MAX	**0.56**	**0.53**	**0.77**	**1.05**
SM15	avg	0.00	0.01	0.02	-0.11
	MAX	**0.56**	**0.53**	**0.90**	**1.05**
SM29	avg	0.00	0.01	0.02	-0.11
	MAX	**0.00**	**0.53**	**0.90**	**1.05**

Table 3.8: LR-CP on F1 1-speed with varying thresholds, %RE

	CPU (sec)	N20	N40	N80	N160
LR	avg	0.01	0.09	0.51	3.19
	MAX	**0.03**	**0.13**	**0.75**	**6.64**
SM04	avg	0.02	0.09	0.57	3.29
	MAX	**0.03**	**0.19**	**1.28**	**7.34**
SM08	avg	0.02	0.10	0.61	3.58
	MAX	**0.03**	**0.25**	**1.56**	**9.25**
SM15	avg	0.02	0.10	0.68	4.10
	MAX	**0.05**	**0.27**	**2.27**	**12.66**
SM29	avg	0.02	0.11	0.83	5.14
	MAX	**0.05**	**0.34**	**3.78**	**19.63**

Table 3.9: LR-CP on F1 1-speed with varying thresholds, CPU

Observe first how dramatically SM04 improves both the average and maximum RE over LR for all cases, with a gradual improvement for SM08, SM15 and SM29. The average RE changes little after SM08. The best tradeoff between number of thresholds (and therefore CPU time) and optimality is achieved with SM08.

With the SM08 threshold set chosen, we now apply LR-CP to all the 1-speed problem sets. Table 3.10 shows the final results of this, and provides the original LR results for comparison.

Fcn	Size	**LR**				**LRCP-sm08**			
		%RE		CPU (sec)		%RE		CPU (sec)	
		avg	max	avg	max	avg	max	avg	max
F1	**20s**	1.98	28.45	0.015	0.031	0.00	0.56	0.016	0.032
	40s	2.65	38.92	0.088	0.125	0.01	0.53	0.096	0.203
	80s	2.81	29.37	0.507	0.750	0.01	0.77	0.589	1.516
	160s	2.83	29.62	3.190	6.641	-0.10	1.05	3.711	9.359
F2	**20s**	1.90	27.78	0.015	0.032	0.00	0.00	0.016	0.032
	40s	2.40	28.20	0.086	0.125	0.01	0.54	0.093	0.234
	80s	2.58	28.54	0.449	0.718	0.04	1.69	0.497	1.266
F3	**20s**	1.82	28.53	0.014	0.032	0.00	0.00	0.014	0.032
	40s	2.11	28.50	0.081	0.125	0.00	0.00	0.088	0.235
	80s	2.40	28.59	0.407	0.734	0.01	0.99	0.441	1.094
F5	**20s**	1.68	17.55	0.014	0.031	0.00	0.00	0.016	0.032
	40s	1.83	30.44	0.091	0.140	0.00	0.00	0.094	0.141
	80s	2.14	32.14	0.503	0.766	0.00	0.00	0.526	0.922
	160s	2.03	23.88	2.972	4.782	-0.19	0.00	3.139	6.313

Table 3.10: LR and LR-CP SM08 results for 1-speed networks

Using LR-CP with 8 thresholds, we have been able to reduce the average RE on the 1-speed networks from almost 3% with LR to less than 0.05%, with average CPU times that are only up to 16% longer than LR. In the worst case, for F1-80s, the maximum CPU time is double that for LR.

3.3.6 LR-CP on the 2-speed network

Returning to the original 2-speed Submarine Transit Path problem, we apply the same sets of SM thresholds and compare the average and maximum LR-CP RE with LR, shown in Table 3.11 and Table 3.12.

	%RE	N20	N40	N80	N160
LR	avg	0.36	0.45	0.35	0.32
	max	**5.49**	**5.62**	**6.16**	**4.73**
SM04	avg	0.11	0.12	0.06	-0.06
	max	**3.54**	**2.99**	**1.69**	**1.50**
SM08	avg	0.07	0.11	0.05	-0.07
	max	**1.15**	**2.99**	**1.31**	**0.28**
SM15	avg	0.05	0.09	0.05	-0.06
	max	**0.96**	**0.91**	**1.31**	**0.28**
SM29	avg	0.05	0.08	0.05	-0.07
	max	**0.96**	**0.91**	**1.31**	**0.26**

Table 3.11: LR-CP on F1 2-speed with varying thresholds, %RE

	CPU (sec)	N20	N40	N80	N160
LR	avg	0.02	0.09	0.45	2.38
	max	**0.03**	**0.14**	**0.66**	**3.91**
SM04	avg	0.02	0.11	0.54	2.96
	max	**0.05**	**0.22**	**1.48**	**8.34**
SM08	avg	0.02	0.11	0.58	3.20
	max	**0.03**	**0.27**	**1.83**	**9.81**
SM15	avg	0.02	0.12	0.67	3.67
	max	**0.05**	**0.39**	**2.58**	**12.94**
SM29	avg	0.02	0.15	1.04	4.88
	max	**0.06**	**0.64**	**4.91**	**22.36**

Table 3.12: LR-CP on F1 2-speed with varying thresholds, CPU

As with the 1-speed networks, we see a large drop in RE with SM04, a further improvement with SM08, and very little change after that. The average RE values for N={20,40,80} are quite close together, and the overall RE drops with N=160. This drop in RE reflects the quality of the CPlex solutions at the larger sizes, with CPlex optimality decreasing as the network size increases.

Once again, as was the case for the 1-speed networks, the optimum tradeoff between RE and CPU time occurs with SM08.

With the SM08 threshold set chosen, we now apply LR-CP to all the 2-speed problem sets. Table 3.13 shows the final results of this, and provides the original LR results for comparison.

Fcn	Size	**LR**				**LRCP-sm08**			
		%RE		CPU (sec)		%RE		CPU (sec)	
		avg	max	avg	max	avg	max	avg	max
F1	**20**	0.36	5.49	0.015	0.032	0.07	1.15	0.018	0.032
	40	0.45	5.62	0.093	0.141	0.11	2.99	0.111	0.266
	80	0.35	6.16	0.446	0.656	0.05	1.31	0.580	1.828
	160	0.32	4.73	2.377	3.906	-0.07	0.28	3.203	9.813
F2	**20**	0.28	5.79	0.015	0.032	0.07	1.55	0.018	0.032
	40	0.29	6.28	0.090	0.141	0.05	0.97	0.105	0.328
	80	0.31	6.36	0.414	0.672	0.02	0.55	0.534	1.719
	160	0.24	5.99	2.219	3.625	-0.07	0.89	2.868	9.328
F3	**20**	0.30	5.46	0.015	0.032	0.05	1.11	0.018	0.047
	40	0.35	5.52	0.088	0.156	0.04	1.24	0.106	0.391
	80	0.27	5.54	0.417	0.672	0.02	0.46	0.540	2.453
	160	0.10	2.28	2.168	3.765	-0.12	0.43	2.793	11.594
F5	**20**	0.17	3.88	0.017	0.032	0.01	0.34	0.019	0.046
	40	0.13	3.12	0.114	0.141	0.01	0.46	0.123	0.250
	80	0.10	2.54	0.573	0.687	-0.01	0.34	0.645	1.750
	160	0.09	2.63	3.128	3.516	-0.03	0.25	3.529	9.094

Table 3.13: LR and LR-CP SM08 results for 2-speed networks

The LR-CP results for the 2-speed network are similar in quality to the 1-speed case. The average RE, which was already better for the 2-speed case, has been reduced from a worst case of 0.45% to 0.11%, while the maximum RE has been reduced from 6.36% to 2.99%. The average CPU time has increased by at most 29%, while the maximum CPU time has increased by a factor of 3.65. Considering the overall improvement in RE, the increase in CPU of less than 4 times over LR makes this a computationally effective method.

3.4 Summary

In this chapter we introduced the first of two network problems studied in this thesis, the Submarine Transit Path problem, which is a singly constrained minimisation problem. Applying Lagrange Relaxation to the original 2-speed problem revealed that the average RE was less than 1% with a maximum of about 6% across 16 problem sets.

By reducing the network problem to have only one speed, the results are very different. The average RE rises to about 3%, but the maximum RE rises to almost 40%, with a maximum LR duality gap just over 40%. Examination of the resulting solutions shows a connection between high LR duality gap and an excessive slack time (unused resource). In its straightforward implementation, Lagrange Relaxation may find local minima instead of global minima.

A hybrid method is developed using a bisecting row of nodes, called a meridian, and a system of successively constraining the slack time along this meridian while using LR to produce updated solutions on the now restricted network. The result is improved solutions in every case. The best cost z produced is then used by LR to filter the network of local minima, allowing LR to find much improved solutions.

This hybrid method, relying on a combination of LR and ideas from constraint programming, we call LR-CP. With a selected list of 8 slack time thresholds, this method is able to find optimal or near-optimal solutions for both the 2-speed and 1-speed Submarine Transit Path problems with CPU times that are in the worst case only 2 to 4 times longer than with LR, and on average are 30% to 50% higher than LR.

Chapter 4

Random Valued Networks

In this chapter we turn our attention to grid networks similar to those which have been studied in the literature. In contrast to the networks of the Submarine Transit Path problem, which have smoothly varying edge costs and uniform edge times, these grid networks have randomly generated edge costs and times.

We consider two variants, one with start and end nodes on opposite corners of the grid, and one with the path starting at any node on one side and ending at any node on the opposite side. As with the Submarine Transit Path problem, we construct a minimum path cost problem with a single side constraint, and focus on problems for which the initial Lagrange Relaxation duality gap is the largest.

4.1 Network Structures

The random valued networks are structured the same as those studied by Dumitrescu and Boland (2003) and Carlyle et al. (2007). Each vertex in the grid has up to three directed edges connecting it to adjacent vertices above, below and to the right. Figure 4.1 shows the first network, Net1, which has the path start at s in the lower left corner and the terminus at the upper right corner. The second network, Net2, is shown in Figure 4.2. This network allows the path to start at any vertex along the left edge and terminate at any vertex along the right edge. The starting vertex s is external to the grid and is connected with zero-cost edges to every vertex along the left side; the terminating vertex t is also external to the

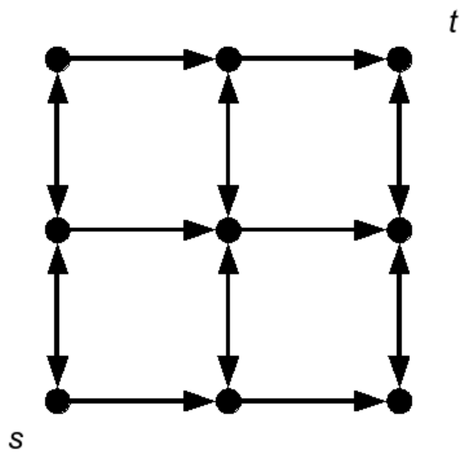


Figure 4.1: Network Net1

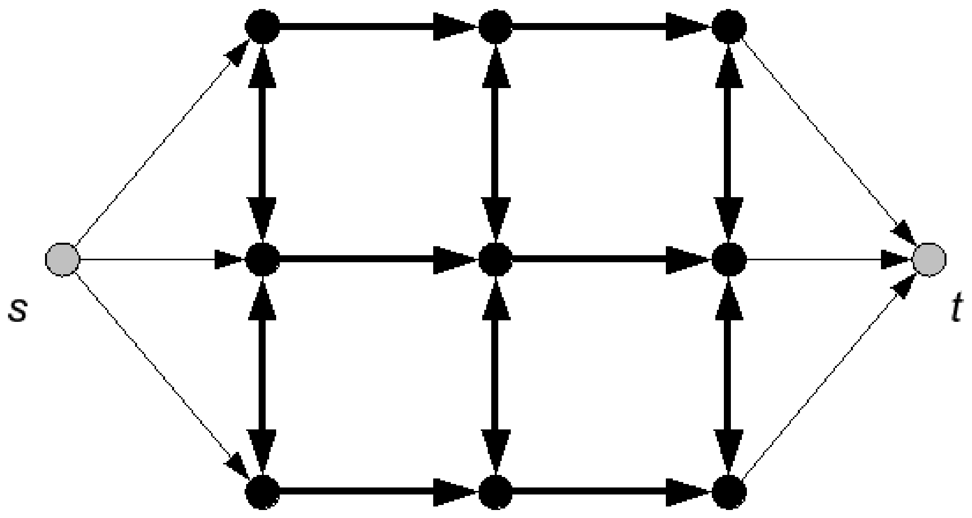


Figure 4.2: Network Net2

grid and is connected to every vertex along the right side with zero-cost edges.

Values for edge weights and lengths are set using the built-in C language function *rand()*, which produces pseudorandom integers. As with all pseudorandom number generators, this function is first initialised with a seed value, a 15-bit integer from 0 to 32,767. Using such a pseudorandom function allows the networks to be constructed repeatably at runtime without the need to store and read in a large array of values.

The cost of a path from s to t is the sum of all the edge costs along the path.

Constraining the problem is a single side constraint, an upper bound on the total path time. Time bounds are calculated as follows: if g_{max} is the time of the unconstrained shortest path and g_{min} is the shortest possible path time, then our time bounds are chosen such that

$$g_i = g_{min} + \alpha(g_{max} - g_{min}) \tag{4.1}$$

where α={0.05, 0.50, 0.95}.

With a set of 32,768 possible networks with 3 time bounds each, this provides a population of 98,304 possible problem cases. We denote each problem (c, t) by the seed value c, which is an integer between 0 and 32,767, and the time bound t which is 1, 2 or 3. For example, a problem labeled (3777, 1) is generated using the seed value 3,777 and has the low time bound produced with α=0.05, while problem (3777, 3) uses the high time bound produced with α=0.95. Since we only consider square grids, the size N of the network indicates the number of vertices along each side. Thus, N50 denotes a square grid with 2,500 vertices.

The random valued networks studied in other works use vertical edge weights in the range [1,10], horizontal edge weights in the range [80,100], and all edge lengths in the range [1,10]. Using these values, the first pass of LR on Net1 N50 revealed, however, that of the 98,304 test cases *not a single case* had an initial LR duality gap above 5%. Network Net2 N50 had only 26 such cases in total, the worst of these having a duality gap of only 7.2%. Of interest in this work are cases in which the initial duality gap is quite large, since this is the type of problem that prevents LR from finding near-optimal solutions. To that end, we change the horizontal edge weights from [80,100] to be in the range [10,30] with the assumption that on average this balances the horizontal and vertical edge weights.

Applying LR to all 98,304 test cases of both network types of size N50 with these new edge weights reveals a very different situation. Net1 now has 1896 cases with a duality gap more than 5% and Net2 has 5314 such cases. The Net1 cases have a duality gap of between 9.0 and 16.6%, the Net2 cases between 12.7% and 28.7%. In terms of RE, the worst case for Net1 is (6591,2) with a 16.1% RE and 16.6%

LR duality gap, and the Net2 worst case is (10161,1) with 24.3% RE and 28.7% LR duality gap.

Table 4.1 lists for N={25,50,100,200} the number of cases which have an initial LR duality gap greater than 5%.

	N25	N50	N100	N200
Net1	5,325	1,896	495	85
Net2	11,010	5,314	2,148	523

Table 4.1: Number of cases with LR duality gap over 5% for Net1 and Net2

It is immediately obvious that as the size of the network increases the number of large duality gap cases decreases. Net1 N200 has only 85 such cases where the gap is greater than 5%. For the purposes of our study we choose the 200 cases (where we have that many) from each network with the largest duality gap. This group is the T200 set. In addition to the top 200 cases, we also select for N50 a second set of 200 cases which have an initial duality gap of about 3%. These problems are the low 200 or L200 cases.

CPlex reference solutions are produced as before, and solved to optimality. Table 4.2 shows the CPU times for the T200 cases of both Net1 and Net2 for N={25,50,100,200}.

		CPlex CPU (sec)	
	N	avg	max
Net1	25	0.38	2.41
	50	2.24	15.86
	100	31.64	163.66
	200	544.51	1635.91
Net2	25	0.40	3.94
	50	3.25	16.75
	100	35.96	236.92
	200	441.24	2355.69

Table 4.2: CPlex CPU times for T200 cases, Net1 and Net2

The average CPU values for Net1 and Net2 are comparable, although the maximum times for Net2 are about 44% longer for N100 and N200.

4.2 Initial LR and LR-CP Results

We focus our initial attention on the N50 sizes for each network. The first step is to apply LR and LR-CP with the eight thresholds of SM08 to the T200 and L200 test sets of both Net1 and Net2 to see how far from optimal the initial solutions are. For LR-CP we use a single centre meridian as for the Submarine Transit Path problem in Chapter 3. Table 4.3 summarises the LR and LR-CP results.

	LR %RE			LR-CP %RE		
	min	avg	max	min	avg	max
Net1 T200	3.65	8.96	**16.07**	0.00	1.59	**11.06**
Net1 L200	1.36	2.47	**3.11**	0.00	1.38	**3.04**
Net2 T200	3.85	11.07	**24.27**	0.00	1.21	**10.89**
Net2 L200	0.21	2.10	**3.03**	0.00	1.04	**2.98**

Table 4.3: Net1 and Net2 N50 initial LR and LR-CP results, 8 thresholds

As might be expected from the large initial LR duality gaps in the T200 cases, the LR solutions are far from optimal. The initial LR solutions for Net1 T200 vary from 3.65% to 16.07% above optimal, and the Net2 T200 cases range from 3.85% to 24.27% above optimal. The L200 cases, with a low LR duality gap, have low RE values but on average are still 2.47% and 2.10% for Net1 and Net2, respectively.

LR-CP is able to produce a significant improvement over LR, bringing the Net1 T200 average RE down from 8.96% to 1.59%, and for Net2 T200 from 11.07% down to an average of 1.21%. Still, there remain high maximum values for RE, 11.06% for Net1 and and 10.89% for Net2.

These LR-CP results are not as good as were seen with the Submarine Transit Path problem, where a single centre meridian was able to bring the RE% below 3% for every case. It appears that a single centre meridian is not sufficient for these networks.

Two problems are chosen for comment.

Figure 4.3 shows that for Net1 (15878,1) in the lower left quadrant and the upper right quadrant there are two competing channels of low $z(\lambda)$ for a near-optimal path to take. The LR path has RE = 11.06%, and the LR-CP path only lowers this slightly to 10.22%.

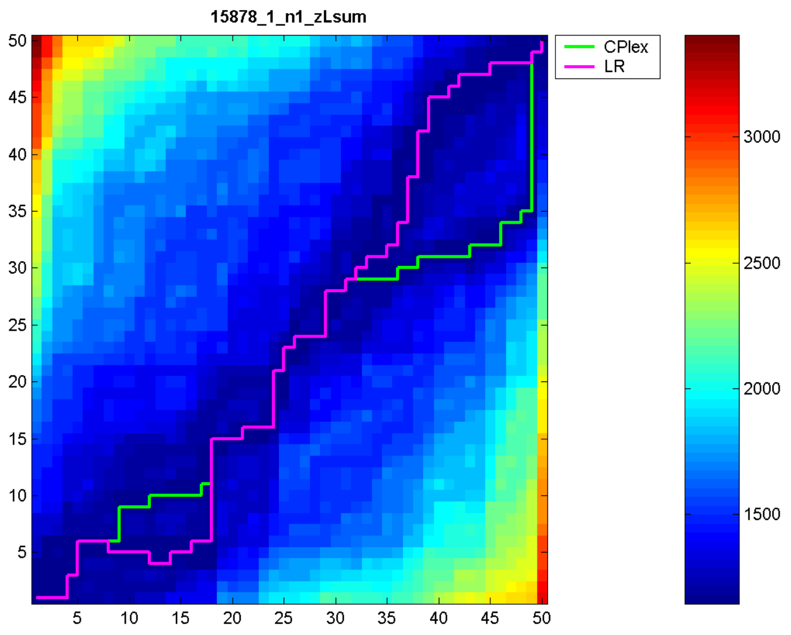


Figure 4.3: Net1 N50 worst case (15878,1) $z(\lambda)$ sums

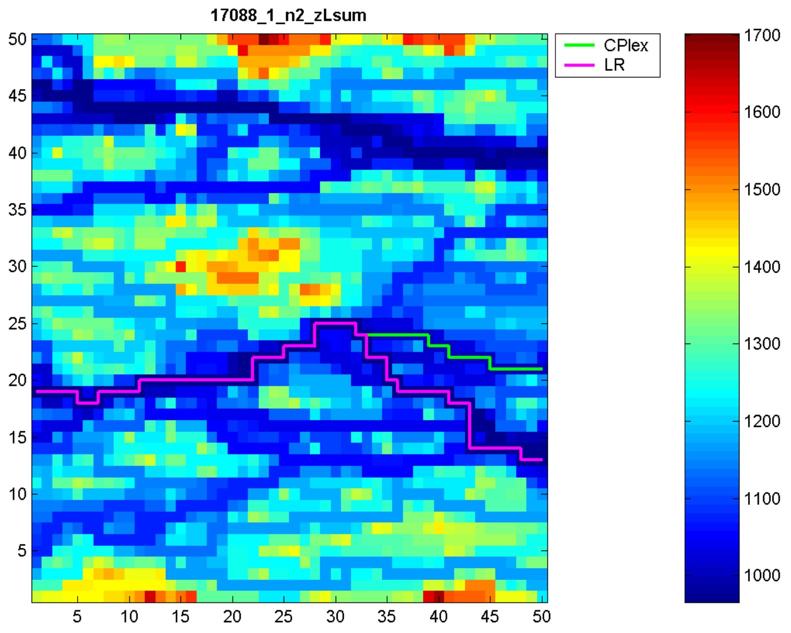


Figure 4.4: Net2 N50 worst case (17088,1) $z(\lambda)$ sums

The same situation occurs with problem Net2 (17088,1) as shown in Figure 4.4. The LR and CPlex paths diverge on the right side of the plot, the LR path taking the branch with lower $z(\lambda)$. The LR path has RE = 13.17%, the LR-CP path (not shown) lowering this again only slightly to 10.89%.

One property both of these plots share is that the areas where parallel valleys of low $z(\lambda)$ appear are not in the centre. Also, they do not run the whole length of the field but are more local. It is hardly surprising, then, that LR-CP with a centre meridian was not able to improve these solutions much.

These two cases demonstrate one thing clearly: for these grid networks with random edge costs, we cannot rely on a single meridian that bisects the grid in the centre. The reason why this is so has to do with the nature of the network. In the Submarine Transit Path problem, the edge costs were determined by a small number of sources which covered the entire field. As a result, the $z(\lambda)$ sum plots had wide, distinct valleys of low $z(\lambda)$ that stretched the entire length of the field. Also, the peak slack times tended to occur in the centre, making that the logical place to put the meridian.

In these networks, the edge costs and edge times are not smooth but random. There can be many parallel channels on the $z(\lambda)$ plot, and they can occur anywhere in the field.

4.3 LR-CP Meridian Scan

In this section, to show if the LR-CP method can achieve near-optimal results on these problems using a single meridian we test all possible meridians, assess the results, and consider if we might be able to predict where the best meridian might be placed.

For network Net1 the meridian is a diagonal line of vertices bisecting the network at the midway line, such that $x + y = N - 1$. Because we do not have diagonal edges, this meridian is sufficient to provide an adequate barrier. We may apply an offset from the centre, describing a line given by $x+y+\text{offset} = N-1$, of between -20 and +20 to give a total of 41 different meridians. Each of these meridians is used in an iteration of the LR-CP method, using 8 thresholds of {0.2, 0.1, 0.05, 0.025, 0.010, 0.005, 0.003, 0.001}, and the best solution is reported. Similarly, for Net2 we may apply an offset from the centre to the meridian, but because the path may start at any node along the left edge of the grid, we may use offsets in the larger range of -25 to +24. The meridian is a vertical column of nodes given by $x = N/2 + \text{offset}$ where $x = 0..N - 1$.

Table 4.4 and Table 4.5 show the LR and LR-CP meridian scan results for N50.

	LR %RE			LR-CP mscan %RE		
	min	avg	max	min	avg	max
Net1 N50 T200	3.65	8.843	**16.07**	0	1.071	**10.60**
Net1 N50 L200	1.36	2.455	**3.04**	0	1.188	**3.04**
Net2 N50 T200	3.85	10.989	**24.27**	0	0.361	**6.68**
Net2 N50 L200	0.21	2.071	**3.02**	0	0.675	**2.73**

Table 4.4: Initial LR and LR-CP mscan %RE, 8 thresholds

	LR CPU (sec)			LR-CP mscan CPU (sec)		
	min	avg	max	min	avg	max
Net1 50T	0.031	0.052	**0.079**	0.094	0.356	**1.391**
Net1 50L	0.046	0.053	**0.094**	0.109	0.338	**1.141**
Net2 50T	0.031	0.056	**0.079**	0.078	0.270	**2.375**
Net2 50L	0.046	0.052	**0.078**	0.078	0.236	**1.438**

Table 4.5: Initial LR and LR-CP mscan CPU, 8 thresholds

These results are only a minor improvement on LR-CP with a single centre meridian. Note that the average RE for Net1 N50 T200 has been reduced from 1.59% to 1.07%, with a maximum RE of 10.60%. The Net2 results are better, the average %RE for N50 T200 dropping from 1.21% to 0.361%, with a maximum of 6.68%. Not as good as we might have expected, however. It appears that for these random valued networks a single meridian is not sufficient, no matter where it is placed.

Another consideration is the CPU time required for such a meridian scan. Note how the average meridian scan CPU times are 4.5 to 6 times longer than for LR, for which we see a modest improvement in the maximum RE.

```
BSCAN:   115 1: 00000000000000000000000000001100000000000 :
BSCAN:   294 2: 00000000011111000000000000000000000000000 :
BSCAN:   384 1: 00000000000000001111111111111000000000000 :
BSCAN:   761 1: 00000000000000000000000000000001000000000 :
BSCAN:   934 1: 00000111111111111111111111111111000000000 :
BSCAN:  1038 1: 00000100000011010001000010000000000000000 :
BSCAN:  1165 1: 00000011000000000000000000000000000000000 :
BSCAN:  1169 1: 01111111111111111111111110000000000000000 :
BSCAN:  1276 1: 01111111100111111111111111111111111111111 :
BSCAN:  1355 1: 00000001111111111000111111010000000011000 :
BSCAN:  1913 1: 00000000000000000000011111100000000000000 :
BSCAN:  2196 1: 00000000000000000000000111111000000000000 :
BSCAN:  2234 1: 00000000000000000110011100000000000000000 :
BSCAN:  2398 1: 00001111111111111111111110000000000000000 :
BSCAN:  2429 1: 01000000000000000000000000000000000000000 :
BSCAN:  2439 1: 00000000000000000000000000000010111100000 :
BSCAN:  2601 1: 00000000000001101100000000000000000000000 :
BSCAN:  3777 1: 00000000000000000011000011000000111111111 :
```

Figure 4.5: Net1 N50 T200 cases, selection of BSCAN results

In the cases where we do get near-optimal results, where do these near-optimal meridians occur? Each row of Figure 4.5 shows a sample of the results, indicating which meridians produced a solution with the same cost as the best one. Each row shows the 41 meridian results from offset=[-20,20] and a '1' if that meridian produced the best result.

From this there appears at first to be no clear pattern. Some cases have only one or two successful meridians, while others such as (1276,1) can have them almost anywhere.

```
BSCAN:   192 1: 00000000000000000000000000000000000100000000000000 :
BSCAN:   431 1: 01100000000000000000000000000000000000000000000000 :
BSCAN:   444 2: 00000000000000000000001111111000000000000000000000 :
BSCAN:   463 1: 01011100011111000011001111111110011111111111110010 :
BSCAN:   667 1: 00000000000000011110000000000000011111111111111110 :
BSCAN:   803 1: 01111111111111111111111111111111111111111111111110 :
BSCAN:  1575 2: 00000000000000000000000000001111110000000000111000 :
BSCAN:  1913 1: 01111111111111111111111111111111111111111111111110 :
BSCAN:  2138 1: 01111111111111111111111111111111111111111111111110 :
BSCAN:  2258 1: 01111111111111111101111111111111111111000000000010 :
BSCAN:  2396 1: 00000000000000000001000000000000000000000000000000 :
BSCAN:  2413 1: 01111001101111111111111101111111111111111111111110 :
BSCAN:  2843 2: 00000111111000000000000000000000000000000000000000 :
BSCAN:  2879 1: 01111111111111111111111111111111111111111111111110 :
BSCAN:  2983 1: 00000000000000000000001000000000000000000000000000 :
BSCAN:  3079 1: 01111111111111111111111111111111111000000000000000 :
BSCAN:  3126 1: 00000000000000000000000000000000000100000000000000 :
BSCAN:  4710 1: 00000000000000000000000000000000000000000111111000 :
BSCAN:  4817 1: 00000000000000000000000000111111111100000000000000 :
BSCAN:  5092 1: 01111111111111111111111111111111111111111111111110 :
```

Figure 4.6: Net2 N50 T200 cases, selection of BSCAN results

Figure 4.6 shows a sample of the Net2 results, with the same wide variation in where the best meridians can occur.

In conclusion, using the LR-CP method on these random valued grid networks does not produce the quality of results we saw for the Submarine Transit Path problem. There seems to be no pattern in where to place the meridian. Trying all possible meridians takes too long, and even then we might need more than one meridian. Another approach is needed, and the answer is to be found by comparing the LR paths with the optimal paths produced with CPlex. This is the subject of the following section.

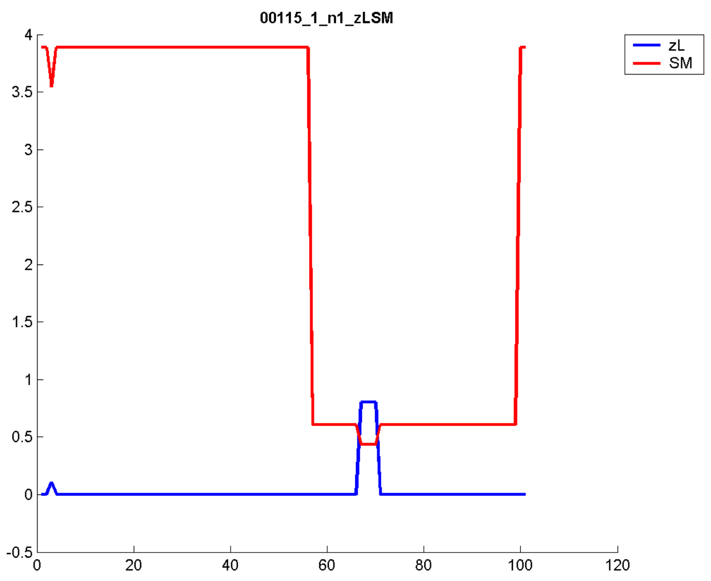
4.4 Second Hybrid Method

In this section we look at the optimal CPlex paths, to see what properties they have that might give us clues as to why the LR paths do not follow the optimal paths. From this analysis we will be able to calculate a value for each node in the network, after a single application of LR, which will allow us to position meridians that effectively filter out non-optimal paths.

4.4.1 Theory

Recall that in our implementation of Dijkstra's algorithm (DA) we have made a modification so that when we sum the shortest $z(\lambda)$ value for a node we also store the sum of the edge times leading to that node. If we apply DA in both directions we then have, for each node, forward and reverse totals for $z(\lambda)$ and for the transit time corresponding to the min $z(\lambda)$ paths.

Furthermore, the nodes along the path of the LR solution have the property that the two-way totals for $z(\lambda)$ are constant and the total time is also constant. If we look at the optimal paths produced by CPlex we find this is not the case. Rather, the $z(\lambda)$ values rise and fall as does the slack time as defined by the DA path times. If we calculate the deviations from min $z(\lambda)$ and the value of slack time along the CPlex paths, we observe that in most cases the $z(\lambda)$ value will rise slightly at the same place that slack time falls. This is shown for (115,1) and (1165,1) in Figure 4.7.



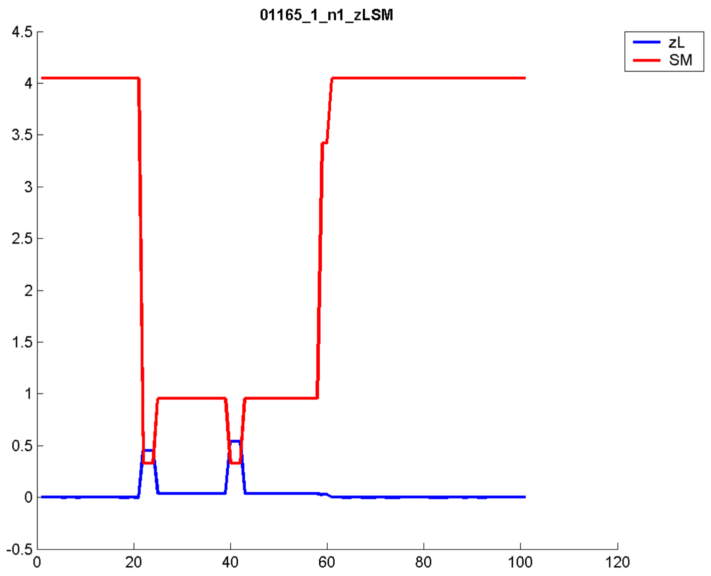


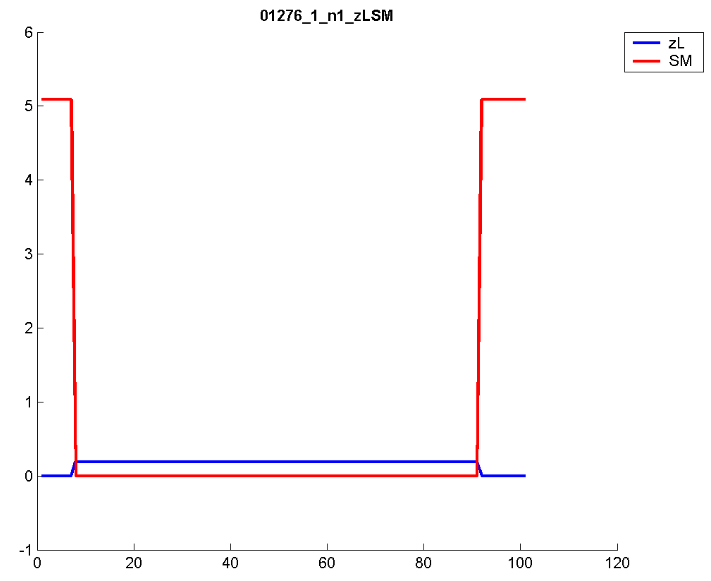
Figure 4.7: Plots of $z(\lambda)$ and slack time margin (SM) along CPlex path, expressed as percentage deviations from the LR path values. SM is the percentage of slack time, zL is the percent rise in $z(\lambda)$ above $z(\lambda)_{min}$.

Now consider just the two Net1 cases (115,1) and (1165,1) below. In either case there are only 2 meridians where the best solution is found.

```
BSCAN:   115 1: 000000000000000000000000000001100000000000 :
BSCAN:  1165 1: 000000110000000000000000000000000000000000 :
BSCAN:  1276 1: 011111111001111111111111111111111111111111 :
BSCAN:  1355 1: 000000011111111110001111111010000000011000 :
BSCAN:  2196 1: 000000000000000000000001111110000000000000 :
BSCAN:  3777 1: 000000000000000000110000110000001111111111 :
```

Referring back to Figure 4.7, note how the areas of the plot where $z(\lambda)$ rises and slack time drops correspond to the meridian offset where LR-CP found the best solution. This indicates that the optimal path is one in which more of the available time is used in areas that allow the $z(\lambda)$ value to rise only slightly. This general behaviour is observed in every case, and is similar to the behaviour observed in the previous chapter, where the optimal path lay in a region with slightly higher $z(\lambda)$ but much lower slack time.

The corresponding plots for the remaining four cases are shown in Figure 4.8 and Figure 4.9.



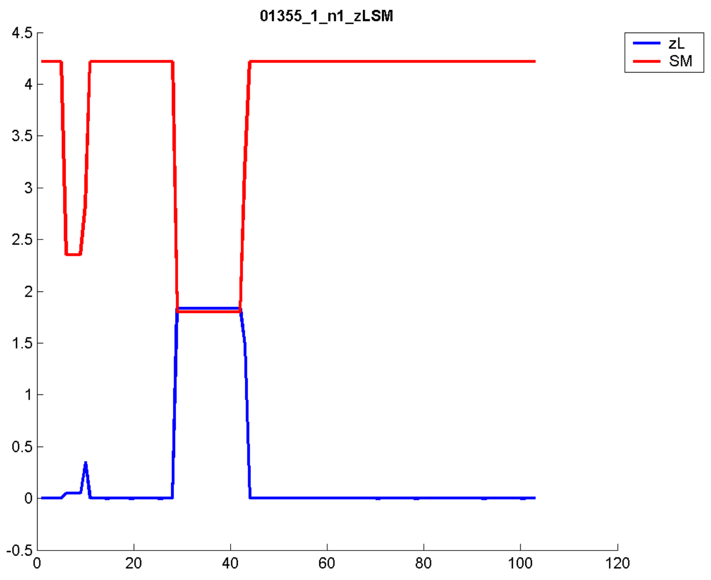
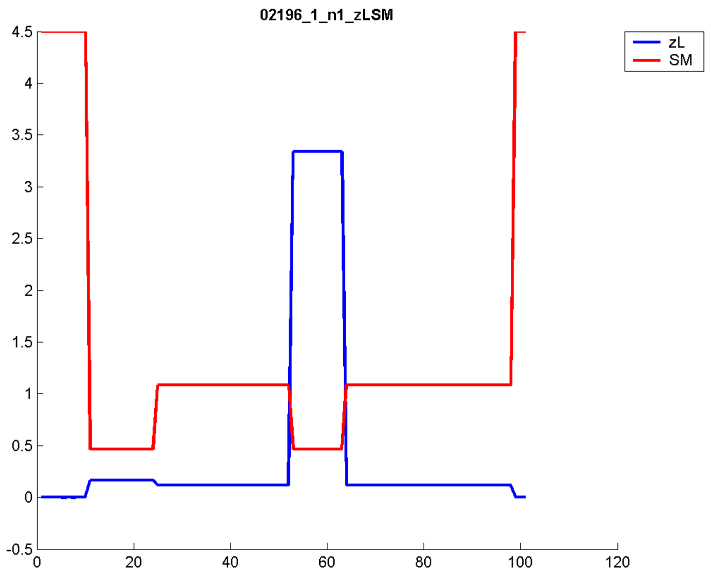


Figure 4.8: Plots of $z(\lambda)$ and slack time margin (SM) along CPlex path for Net1 (1276,1) and (1355,1).



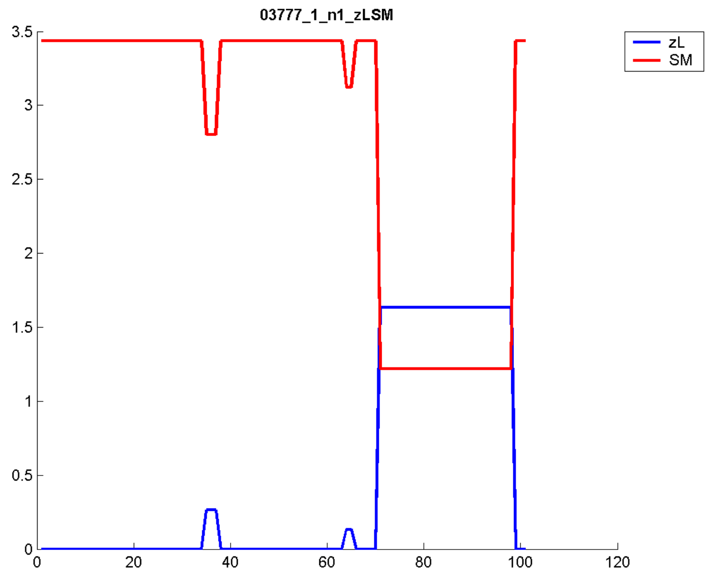


Figure 4.9: Plots of $z(\lambda)$ and slack time margin (SM) along CPlex path for Net1 (2196,1) and (3777,1).

Making use of this property, we may calculate a new parameter R given by

$$\text{Ratio R} = \frac{(T_2 - T_1)}{(z(\lambda)_2 - z(\lambda)_1)} \tag{4.2}$$

where T is the node sum of time calculated by DA and $z(\lambda)$ is the two-way sum of the modified edge costs for each node. If T_1 is the time of the LR path and $z(\lambda)_1$ is the $z(\lambda)$ of that path, which are constant for a given value of λ, then for every node with T_2 and $z(\lambda)_2$ we may then calculate the value of R. The value zL_1 is the minimum for $z(\lambda)$ and $T_2 \geq T_1$, so the value of R will always be positive.

What this R value gives us is a ratio of path time to change in $z(\lambda)$, and where this ratio is highest is an indicator of places in the network where the path time rises higher (thus using more of the available time) while the $z(\lambda)$ value rises slightly. These should indicate where the parallel valleys are in the $z(\lambda)$ plot. Figure 4.10 shows a plot of these R values for Net1 case (1165,1).

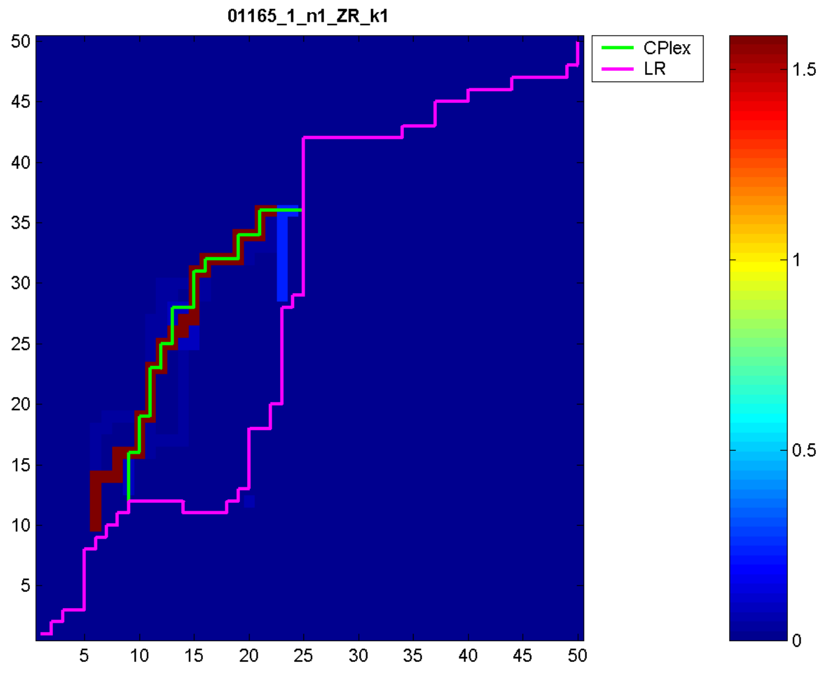


Figure 4.10: Net1 (1165,1) ZR with key 1

Notice where the CPlex path and the LR path diverge is where the R value is the highest. The term 'key 1' will be explained shortly. Figure 4.11 shows the plot of R values for Net1 case (15878,1), which was previously identified as the worst

Net1 T200 case with a RE of 11.06% and was shown in Figure 4.3. In this plot there are two branches where the LR and CPlex paths diverge, and both of the CPlex branches show a higher R value.

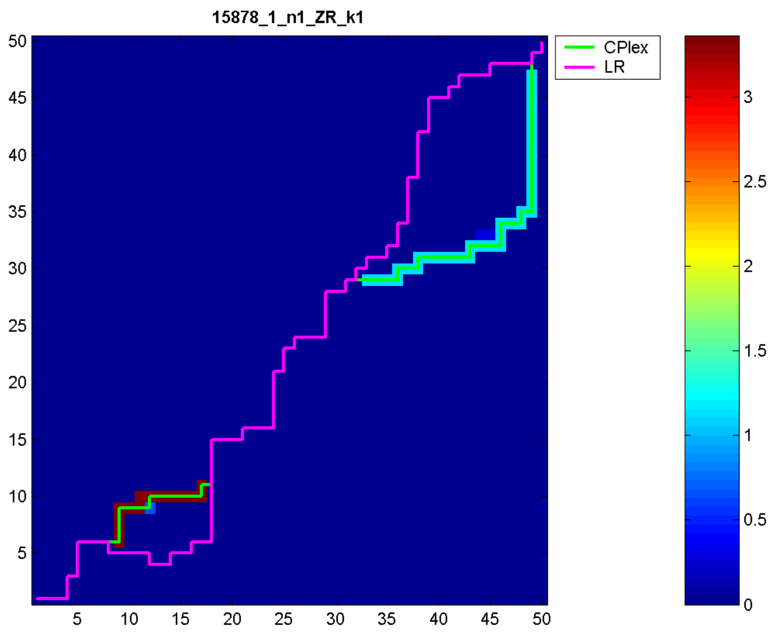


Figure 4.11: Net1 (15878,1) ZR with key 1

Now that we have established a link between our R value and the possible location of optimal paths, we proceed with the application and testing of this approach.

4.4.2 Procedure

Now that we can potentially identify areas of the network where the LR path deviates from the optimal path, we need some way to use this information. We may use a meridian to temporarily constrain the network as before, but we then face two questions: where to place the meridian, and what value of R is appropriate?

From the plots shown so far it appears that the best place to use a meridian to constrain the network is at the point where the distance between the LR path and the path of high R value is the greatest.

Let us set a node flag INI to true if the node lies on the initial LR path, and we set another flag RMAX to true if that node has a high value of R (which we define shortly). We have two choices here: we can disable all but the meridian

RMAX nodes and force the LR path to go through that small opening, or we can disable only the meridian INI nodes and allow LR to find an alternate path without them. We call the first approach *aggressive* and the second approach *conservative*. This is our first variable.

The first step after applying LR is to calculate the R values for every feasible node in the network. We also calculate the mean $\overline{R}$ and the standard deviation σ given by

$$\sigma = \frac{1}{N}\sqrt{\sum_i (R_i - \overline{R})^2} \tag{4.3}$$

We set the threshold for RMAX as $R_{TH} = \overline{R} + i\sigma$, for $i = [5, 4, 3, 2, 1, 0]$ where i is the highest value for which the maximum value of R is greater than R_{TH}. For any node with $R > R_{TH}$ we set the RMAX flag. We may also choose to arbitrarily set $R_{TH} = \overline{R}$. This choice becomes our second variable.

After calculating the R values, we then examine the nodes in each meridian to see what the maximum distance is between the INI node and any RMAX node on that meridian. After scanning all meridians, we select the one with the largest such distance. We do this in order to find the place where the potential optimal path differs the most from the initial LR path.

Before proceeding to the algorithm, it must be noted that there are cases where the R value does *not* indicate the optimal path. One such case is Net1 (3777,1) shown in Figure 4.12.

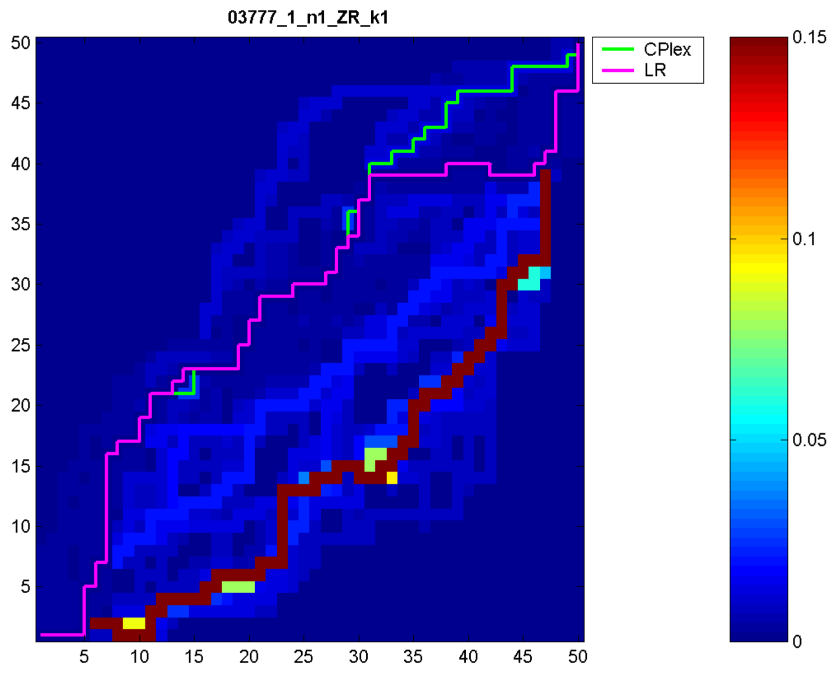


Figure 4.12: Net1 (3777,1) ZR with key 1

Note how there is a region of high R value that does not correspond to the optimal path. This arises because our calculation of the path time using DA allows the time to exceed the time upper bound. If we disable the nodes where this is the case before we calculate R, we get the situation shown in Figure 4.13.

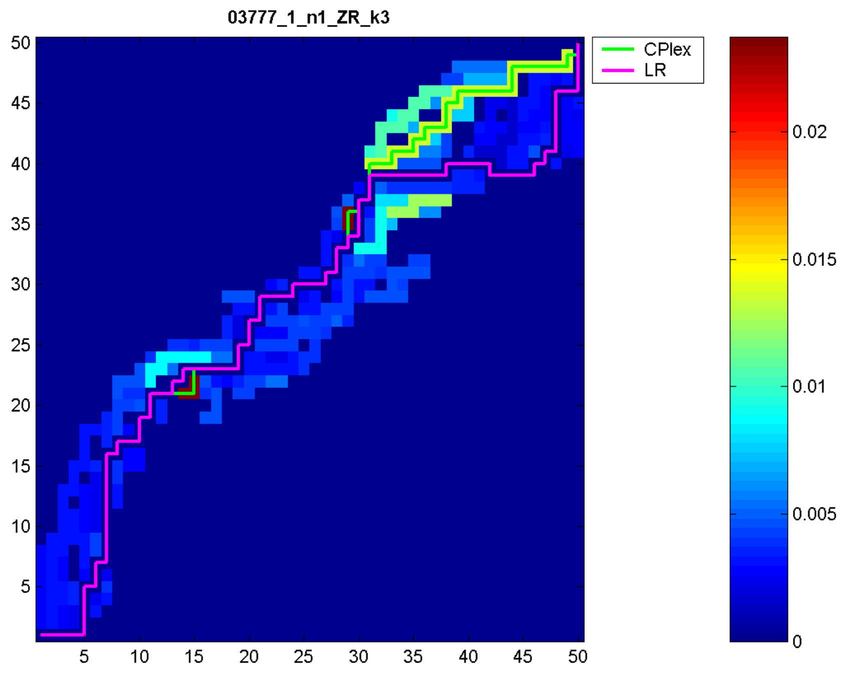


Figure 4.13: Net1 (3777,1) ZR with key 3

Once again, the optimal path is seen to pass through an area of high R. The choice of whether to ignore nodes for which the time total exceeds the time bound is our third variable.

Figure 4.14 and Figure 4.15 show Net2 case (10161,1). Notice that key 3 is able to identify the optimal path exactly.

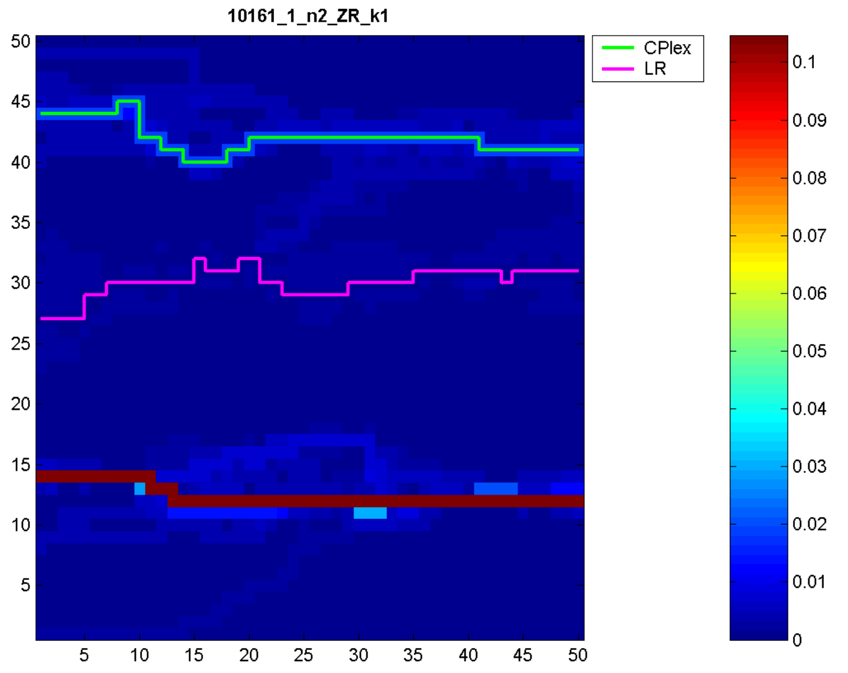


Figure 4.14: Net2 (10161,1) ZR with key 1

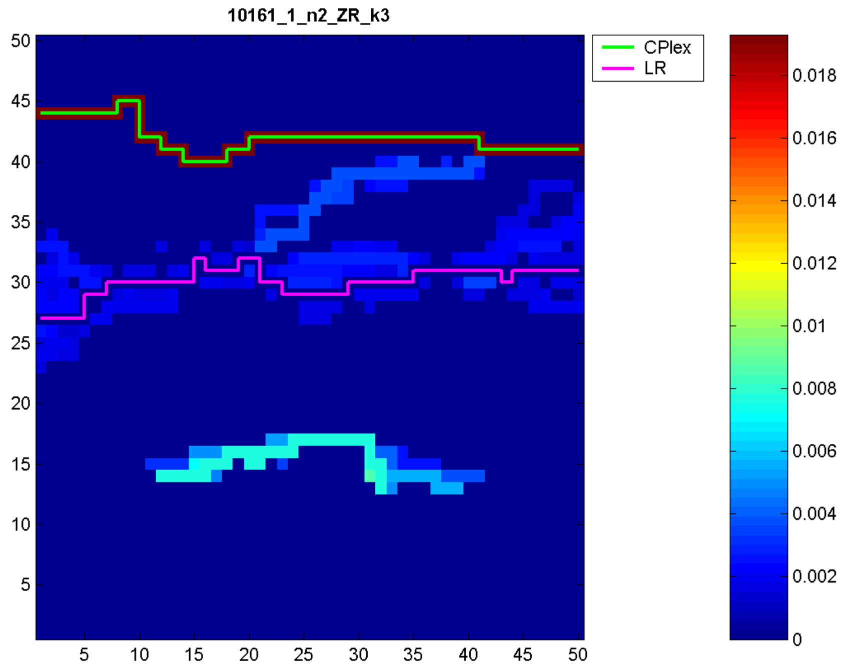


Figure 4.15: Net2 (10161,1) ZR with key 3

	R_{TH}		Slack Time		
key	Max	Avg	$ST < 0$	$ST \geq 0$	Aggressive
1	X		X		X
2	X		X		
3	X			X	X
4	X			X	
5		X	X		X
6		X	X		
7		X		X	X
8		X		X	

Table 4.6: The 8 key values

We thus have three variables to consider: (i) the *aggressive* and *conservative* application of meridians, (ii) allowing or disallowing nodes for which the calculated path time based on $z(\lambda)$ is within the time upper bound or not, and (iii) the choice of R_{TH}. This leads us to define 8 key values, shown in Table 4.6.

The meaning of k1 and k3 in Figure 4.12 and Figure 4.13 is now clear. The only difference between key 1 and key 3 is that key 1 allows the slack time to be negative. In other words, with key 1 the path time as calculated by our modifications to Dijkstra's algorithm are allowed to exceed the time upper bound.

To apply a meridian, we disable the nodes in question (either just the INI nodes, or all but the RMAX nodes) and apply LR. There are several possible outcomes:

1. LR fails - critical nodes have been disabled
2. LR succeeds but cost is higher
3. LR succeeds and cost is lower

If LR fails, then we have disabled a critical node and need to invert the state of the disabled flag for every node on the meridian. We apply LR again and move on to the next meridian.

If the resulting path cost is higher, we have made the wrong move. We invert the disabled flag for every node on the meridian, and apply LR again. If the path cost is still higher, then we unset the disabled flags, mark this meridian as having been tried, and move on to the next one.

The final algorithm is given in Algorithm 3.

Algorithm 3 ZR Method: Applying R-valued meridians

```
1:  apply LR, get initial path and cost
2:  for $i = 1$ to 10 do
3:     calculate R for all nodes
4:     set R threshold
5:     set RMAX flags
6:     scan all meridians and calculate distance from INI to RMAX (distR)
7:     select meridian with max distR
8:     if aggressive then
9:        disable all meridian nodes except RMAX
10:    else
11:       disable all INI meridian nodes
12:    end if
13:    apply LR
14:    if new cost is lower then
15:       keep this barrier
16:       store cost
17:    else
18:       if LR fails then
19:          invert meridian flags
20:          apply LR
21:       else if LR cost is higher than initial cost then
22:          if aggressive then
23:             disable all but INI nodes
24:          else
25:             invert meridian flags
26:          end if
27:          apply LR
28:       end if
29:       if new LR cost is still higher than initial cost then
30:          unset all meridian flags
31:          mark this meridian as done
32:          apply LR, store cost
33:       end if
34:    end if
35: end for
```

4.4.3 Key tests

In order to see which keys are most effective, we begin by applying the ZR method of Algorithm 3 with a maximum of 10 passes on the Net1 T200 cases, using each of the 8 keys on its own, ignoring the CPU times for now. Table 4.7 shows the total, mean and maximum RE for all 200 cases, with the initial LR results included for comparison. Table 4.8 shows these same results for Net2 N50.

%RE	LR	k1	k2	k3	k4	k5	k6	k7	k8
total	1730.5	109.6	196.3	167.9	234.0	111.6	206.1	444.6	296.4
avg	8.65	**0.55**	0.98	0.84	1.17	0.56	1.03	2.22	1.48
max	16.07	10.37	7.19	7.68	6.82	4.67	6.48	11.20	7.93

Table 4.7: Net1 N50 T200 LR-ZR %RE using 8 separate keys, 10 passes

%RE	LR	k1	k2	k3	k4	k5	k6	k7	k8
total	2213.5	176.4	311.0	335.4	376.8	183.2	342.3	451.1	329.3
avg	11.07	**0.88**	1.55	1.68	1.88	0.92	1.71	2.26	1.65
max	24.27	8.03	11.58	11.87	16.17	9.96	17.54	12.69	18.11

Table 4.8: Net2 T200 LR-ZR %RE using 8 separate keys, 10 passes

It is remarkable that by using key 1 and performing 10 passes (which is to say, testing 10 meridians), the average RE for Net1 has been reduced from 8.96% to 0.55%, and for Net2 from 11.07% to 0.88%. However, the lowest maximum RE for Net1 is still 4.67% with key 5. While no one key seems to solve every case, many cases which could not be solved optimally by one key are solved by using another. Recall the difference between key 1 and key 3: the slack time is allowed to be negative for 1, and strictly non-negative for 3.

Hence it is only a question of which combination of keys we need to use. If we consider the minimum RE values among a combination of keys from Table 4.7 we see in Table 4.9 the best RE values obtained if we were to use that combination of key values in sequence, that is, to apply Algorithm 3 with each of the keys individually.

%RE	all	k1357	k2468	k1234	k5678	k135	k13	k123	k12
total	21.90	30.73	105.52	27.73	40.59	37.84	41.89	29.91	50.67
avg	**0.11**	0.15	0.53	**0.14**	0.19	0.20	0.21	0.15	0.25
max	**1.40**	1.40	6.48	**1.40**	1.45	1.75	2.87	1.40	3.90

Table 4.9: Net1 N50 T200 %RE using different keys combinations, 10 passes

Using all 8 keys it is possible to reduce the average RE to 0.11% and the worst case %RE down to 1.40% for Net1, as shown in the first column. We may also achieve nearly the same result using only the 4 keys of k1234. The k1357 combination represents all the aggressive approaches, while k2468 uses all the conservative approaches; the aggressive approach appears to perform better. However, the mix of aggressive k13 and conservative k24 in k1234 gives us the best overall result if we had to choose only 4 keys.

Similarly, Table 4.10 shows for Net2 how the worst case RE can be reduced from 8.03% to 2.49% by using all 8 keys, while the key combination k1234 can achieve nearly the same result.

%RE	all	k1357	k2468	k1234	k5678	k135	k13	k123	k12
total	21.82	43.76	166.68	30.88	54.31	60.89	66.64	34.71	62.25
avg	**0.11**	0.22	0.83	**0.15**	0.27	0.30	0.33	0.17	0.31
max	**2.49**	5.10	8.34	**2.85**	5.07	5.97	5.97	2.85	3.14

Table 4.10: Net2 N50 T200 %RE using different keys combinations, 10 passes

Extending these key tests to the T200 sets of all sizes for both Net1 and Net2, we now present the single key test results in Table 4.11 and the combination key test results in Table 4.12.

Net	Size	RE	**LR**	**k1**	**k2**	**k3**	**k4**	**k5**	**k6**	**k7**	**k8**
Net1	25	avg	10.78	0.24	0.23	0.80	0.66	0.31	0.29	1.47	1.33
		max	**22.08**	**4.64**	**4.64**	**6.83**	**5.32**	**3.90**	**3.90**	**10.93**	**9.19**
	50	avg	8.84	0.22	0.19	0.84	0.60	0.28	0.28	1.68	1.36
		max	**16.07**	**3.72**	**3.72**	**7.68**	**4.39**	**3.06**	**3.06**	**9.58**	**9.58**
	100	avg	6.84	0.23	0.21	0.63	0.43	0.29	0.25	1.67	1.41
		max	**12.68**	**6.95**	**6.95**	**5.12**	**2.72**	**3.02**	**2.61**	**7.25**	**6.26**
	200	avg	5.29	0.05	0.05	0.38	0.39	0.34	0.28	1.34	0.97
		max	**8.13**	**1.17**	**1.17**	**1.72**	**2.37**	**1.75**	**1.43**	**3.94**	**4.54**
Net2	25	avg	14.31	0.96	1.35	2.11	1.71	0.63	0.84	2.32	2.16
		max	**36.98**	**14.94**	**21.29**	**15.88**	**15.23**	**11.75**	**11.59**	**14.63**	**16.32**
	50	avg	10.99	0.37	0.40	1.06	0.85	0.36	0.36	1.39	1.26
		max	**24.27**	**7.45**	**6.65**	**10.88**	**12.40**	**5.04**	**4.66**	**8.89**	**10.25**
	100	avg	9.20	0.34	0.43	1.24	0.88	0.41	0.50	1.90	1.40
		max	**16.17**	**8.22**	**9.05**	**7.83**	**8.29**	**7.72**	**9.05**	**9.70**	**9.70**
	200	avg	6.95	0.17	0.26	1.04	0.85	0.50	0.45	1.75	1.35
		max	**10.28**	**5.48**	**7.35**	**6.27**	**6.11**	**4.13**	**7.35**	**7.31**	**5.13**

Table 4.11: Single key tests %RE on T200 cases, 10 passes

Net	Size	RE	**all**	**k1357**	**k2468**	**k1234**	**k5678**	**k123**	**k12**	**k13**	**k135**
Net1	25	avg	0.06	0.09	0.08	0.09	0.17	0.10	0.19	0.12	0.10
		max	**1.45**	**1.45**	**1.54**	**1.45**	**3.56**	**1.45**	**4.64**	**1.52**	**1.45**
	50	avg	0.07	0.08	0.10	0.09	0.16	0.11	0.18	0.14	0.09
		max	**1.32**	**1.32**	**1.32**	**1.85**	**1.75**	**2.54**	**3.72**	**3.02**	**1.32**
	100	avg	0.04	0.05	0.06	0.06	0.16	0.07	0.19	0.08	0.06
		max	**0.78**	**1.29**	**1.67**	**0.78**	**1.38**	**1.29**	**6.95**	**1.62**	**1.29**
	200	avg	0.05	0.05	0.05	0.05	0.28	0.05	0.05	0.05	0.05
		max	**1.17**	**1.17**	**1.17**	**1.17**	**1.43**	**1.17**	**1.17**	**1.17**	**1.17**
Net2	25	avg	0.21	0.35	0.45	0.52	0.37	0.61	0.79	0.69	0.38
		max	**4.15**	**8.17**	**6.43**	**14.94**	**4.62**	**14.94**	**14.94**	**14.94**	**9.25**
	50	avg	0.09	0.14	0.16	0.12	0.20	0.16	0.24	0.23	0.17
		max	**2.60**	**4.44**	**3.81**	**2.70**	**3.69**	**6.19**	**6.65**	**6.19**	**4.44**
	100	avg	0.07	0.10	0.17	0.10	0.23	0.13	0.26	0.14	0.11
		max	**2.61**	**2.61**	**3.39**	**2.61**	**3.39**	**3.32**	**7.72**	**3.32**	**2.61**
	200	avg	0.07	0.10	0.12	0.08	0.27	0.09	0.11	0.11	0.10
		max	**2.08**	**2.08**	**2.08**	**2.08**	**2.08**	**2.08**	**2.08**	**2.08**	**2.08**

Table 4.12: Combination key tests %RE on T200 cases, 10 passes

In Table 4.11 we see a comparison between LR and the individual key tests. In every key test, the RE is lower than for LR, sometimes substantially so. In addition, key 1 appears to deliver the consistently lowest average RE of any key. We will use this information shortly.

In Table 4.12 we see the effect of key combinations, including the use of all 8 keys, in reducing the maximum and average RE. Using all 8 keys, we see uniformly good results, with the average RE less than 0.1% for all but the Net2 N25 set. For Net1, the maximum RE using all 8 keys is always less than 1.5%, while for Net2 the maximum RE is slightly higher. These results are nonetheless impressive, given that the initial LR results for these data sets have average RE of between 5.29% and 14.31%.

So far we have not used any preprocessing to reduce the network size, as we have thus far been concerned only with achieving better solutions. If the best results are produced by using all 8 keys, then we need to consider the computation time of applying the ZR method with each of the 8 keys.

Recall that if we have a feasible solution cost z, we may eliminate any node from the network if the $z(\lambda)$ sum through that node is higher than z. Since we are able to use LR-ZR to reduce the RE more than LR, and therefore lower the path cost z, it follows that LR-ZR will be able to reduce the network more than LR.

Net	Size	Num	N1 % avg	N2 % avg
Net1	25	200	46.24	16.99
	50	200	36.48	10.55
	100	200	35.96	6.48
	200	85	35.82	3.64
Net2	25	200	52.30	16.74
	50	200	31.04	7.40
	100	200	34.35	4.65
	200	200	50.52	3.02

Table 4.13: Network reduction using 3 passes with key 1

The LR-ZR method using key 1 is able to offer the most consistent reduction in RE. If we were to first apply LR-ZR with key 1 and only use a small number of passes, we might be able to substantially reduce the network prior to applying all 8 keys. Table 4.13 shows the average network reduction on all T200 cases using

LR compared with using 3 passes of LR-ZR with key 1.

The N1 column shows the percentage of nodes remaining after applying LR, then using the LR path cost to filter out infeasible nodes. Similarly, the column N2 shows the corresponding reduction using the path cost achieved after 3 passes of LR-ZR with key 1. Note that LR is only able to reduce the network on average to about 30%, while LR-ZR is able to achieve an average of 3%, or up to an order of magnitude better. With the network thus reduced, we also reduce the CPU time of applying LR-ZR with the entire set of 8 keys.

Also, it is observed from experimental results that most cases do not improve much after 5 passes of LR-ZR. Thus, our final LR-ZR method will consist of 3 inital passes with key 1, followed by up to 7 passes with each of the 8 keys. The steps are outlined in Algorithm 4, which uses Algorithm 3 as a subroutine.

Algorithm 4 ZR Method: Applying LR-ZR

```
 1: apply LR, get initial path and cost $z$
 2: filter infeasible nodes using $z$
 3: apply LR-ZR, 3 passes with key 1
 4: store cost $z$ and path
 5: remove all meridians
 6: apply LR
 7: filter infeasible nodes using $z$
 8: for $i = 1$ to 8 do
 9:     apply LR-ZR, 5 passes with key $i$
10:     store cost $z$ and path
11:     remove all meridians
12:     apply LR
13:     retrieve best path and cost $z$
14:     filter infeasible nodes using $z$
15: end for
16: remove all meridians
17: apply LR
18: retrieve best path and cost $z$
19: filter infeasible nodes using $z$
20: apply LR
21: report final cost $z$ and path
```

The results of this approach are discussed in the next section.

4.5 LR-ZR Results for Net1 and Net2

Table 4.14 shows the RE, LGAP and CPU average and maximum values for all T200 problem sets of Net1 and Net2 using LR, and Table 4.15 shows the corresponding results using LR-ZR.

				LR					
				%RE		%LGAP		CPU (sec)	
Net	Size	Num	N2 % avg	avg	max	avg	max	avg	max
Net1	25	200	46.24	10.78	22.08	14.811	23.140	0.008	0.016
	50	200	36.48	8.84	16.07	10.388	16.485	0.052	0.079
	100	200	35.96	6.84	12.68	7.439	14.211	0.495	0.719
	200	85	35.82	5.29	8.13	5.586	8.357	3.671	3.828
Net2	25	200	52.30	14.31	36.98	21.777	40.614	0.009	0.031
	50	200	31.04	10.99	24.27	14.998	28.453	0.056	0.079
	100	200	34.35	9.20	16.17	10.474	18.044	0.572	0.828
	200	200	50.52	6.95	10.28	7.421	10.717	4.065	4.313

Table 4.14: LR results for Net1 and Net2 T200

				LR-ZR					
				%RE		%LGAP		CPU (sec)	
Net	Size	Num	N2 % avg	avg	max	avg	max	avg	max
Net1	25	200	12.24	0.07	1.62	0.821	7.019	0.025	0.078
	50	200	7.14	0.08	2.54	0.633	4.511	0.109	0.296
	100	200	4.78	0.05	0.56	0.441	2.108	0.877	2.000
	200	85	3.21	0.07	0.29	0.344	1.168	6.131	9.234
Net2	25	200	9.26	0.28	7.20	1.248	10.890	0.026	0.141
	50	200	4.15	0.08	2.69	0.663	9.469	0.102	0.468
	100	200	2.51	0.08	2.61	0.518	3.859	0.939	3.938
	200	200	2.10	0.09	2.08	0.443	2.812	8.740	20.97

Table 4.15: LR-ZR results for Net1 and Net2 T200

The column N2 indicates the average size of the network after filtering out infeasible nodes. For the larger network sizes N100 and N200, LR-ZR is able to reduce the network by over 95%, where LR is only able to reduce by at most 65%.

The average LR-ZR RE is below 0.1% for all test sets except for Net2 N25, which has an average of 0.28%. Again, with the exception of Net2 N25, the maximum RE is below 3%. The average %LGAP is similarly low for all test sets, however the maximum values are still quite high for the smaller sizes of N25 and N50. Since this is our only measure of optimality when we do not have CPlex reference solutions, these LGAP figures are not sufficient to give a confident bound on optimality. Doing this is the subject of the next chapter.

The CPU times for LR-ZR are on average about double than for LR, and in the worst case are only about 5 times higher. Note that the average LR-ZR CPU times for N50 are roughly one third smaller than using the LR-CP meridian scan in Section 4.3, and the maximum CPU times for N50 are about 5 times smaller for LR-ZR. In addition to taking less CPU time, the LR-ZR method is able to produce far better results than LR-CP on these random valued grid networks. Even though there is much more processing involved in the LR-ZR method, our use of preprocessing effectively reduces the network so that we can afford to run LR-ZR with all 8 keys on a strongly reduced network.

Finally, there is a clear trend in the RE and LGAP figures: as the network size increases, these values decrease. This is consistent with the decrease in number of cases which have an initial LR duality gap over 5% as the network size increases, as was shown previously in Table 4.1. Recall that for Net1 N200, we only had 85 such cases out of a population of almost 100,000 possible problems. The conclusion we may draw is that for grid networks with randomly generated edge costs and times, as the network size increases LR-ZR is able to produce correspondingly better solutions.

4.6 Summary

In this chapter we have considered random valued grid networks similar to those studied in the literature. We have altered the randomly generated edge weights to produce problems with much higher initial duality gaps, from which we select the worst 200 cases of each size out of a possible set of almost 100,000. Thus, we are considering only the very worst problems, ones for which LR is unable to produce quality solutions.

The LR-CP hybrid method was shown to be insufficient for these networks, as the single central meridian does not always intersect an area where the slack time is high. In fact, the areas where the LR and CPlex (optimal) paths diverge may occur in several local places, unlike in the networks of the Submarine Transit Path problem.

We therefore have developed a second hybrid method, which uses information gathered by our modified Dijkstra's algorithm, to calculate a ratio value R which we use to predict where the LR path may not be optimal. Using this ratio to position meridians, we temporarily constrain the network and apply LR in order to seek improved paths with lower cost z. These improved z values are then used to further reduce the network of suboptimal nodes.

The resulting method uses a combination of LR preprocessing and two applications of the LR-ZR method to produce optimal or near-optimal results on the selected problem sets, which represent for each problem size the worst 200 cases out of a population of almost 100,000 problems. The average CPU times for LR-ZR are about 2 times higher than LR, and in the worst case almost 5 times higher.

Chapter 5

Using LR-ZR For Preprocessing

In this thesis we have developed two hybrid methods based on Lagrange Relaxation, the LR-CP method which was applied to the Submarine Transit Path problem, and the LR-ZR method which was developed for the random valued network problem. Although the LR-ZR method is able to produce optimal or near-optimal solutions for the random valued networks, the gap between the solution cost z and the LR lower bound $z(\lambda)$ may still be significant, and thus we cannot always guarantee that LR-ZR will find an optimal solution. If a guarantee of optimality is required, then we need an additional processing step to close this gap. We consider two methods to do this, the bounded search method of Carlyle et al. (2007) and the commercial LP solver package CPlex. In this sense, we are using LR-ZR as a preprocessing step before gap closing.

First, we introduce the bounded search method LRE-A of Carlyle et al. (2007), which uses Lagrange Relaxation preprocessing and path enumeration with aggregated constraints to perform a bounded search. We then apply LR-ZR as a preprocessing step on the random valued network problems and close the gap with both the bounded search and CPlex. Secondly, we apply the LR-ZR method (with the LRE-A search) to the Submarine Transit Path problem, and compare how well this method performs on the two distinct types of networks we have studied.

We begin with a discussion of the LRE-A method on the random valued grid networks.

5.1 Random Valued Networks

The first step is to apply LRE-A to all random valued network problem sets, in order to determine how well the bounded search method works with only LR preprocessing. We use a duality gap tolerance of 1% and set a search time limit of 300 seconds. For comparison, we also use CPlex with no preprocessing, set the gap tolerance to 1% and allow it to run to completion. Table 5.1 shows the results.

	Num	N2	LRE-A %RE		CPU (sec)		No.	CPlex 1% CPU	
		avg	avg	max	avg	max	TO	avg	max
Net1 25	200	46%	0.03	0.70	0.009	0.016	0	0.23	1.53
50	200	36%	0.11	0.88	0.070	0.454	0	2.13	11.11
100	200	36%	0.48	7.20	9.701	300.7	2	29.6	124.8
200	85	36%	1.15	5.37	136.3	304.1	27	525.7	1582.1
Net2 25	200	52%	0.00	0.00	0.009	0.031	0	0.26	2.56
50	200	31%	0.02	0.53	0.059	0.079	0	3.10	15.09
100	200	34%	0.14	0.95	2.494	150.1	0	31.78	215.3
200	200	51%	0.70	3.66	49.66	304.5	20	420.2	1939.5

Table 5.1: T200 LRE-A results with 1% tolerance and 300 sec time limit

The column N2 shows the average size of the reduced network after LR preprocessing. The average ranges from 31% to 52%, indicating that for these problems LR is not that effective in filtering out non-optimal nodes. This is hardly surprising, given the large initial duality gaps of these problem sets. The column "No. TO" shows how many cases timed out, reaching the 300 second time limit during the LRE-A search. For Net1 N200, 27 of the 85 problem cases reached the time limit, or almost one third of the cases in this problem set.

The weakness of LR filtering is reflected in the CPU times needed for the search. For the N25 and N50 sizes, all cases ran to optimality, while in the N200 problem sets there were a significant number of cases which reached the time limit before reaching 1% optimality. The maximum RE is over 5% for these cases. However, comparing CPU times for the N25 and N50 problem sets which did complete within the time limit, we see that LRE-A is on average about 20 times faster than CPlex. For the N100 and N200 problem sets the ratio of average CPlex CPU times to average LRE-A times is much lower, between 3 and 6, indicating that the bounded search performance declines as the networks increase in size.

	Num	CPU1 Init	CPU2 Setup	CPU3 PP	CPU4 Search	CPU5 Total
Net1 50	200	0.014	0.030	0.008	0.018	0.070
100	200	0.087	0.316	0.094	9.205	9.701
200	85	0.522	2.478	0.726	132.596	136.321
Net2 50	200	0.014	0.034	0.007	0.003	0.059
100	200	0.093	0.376	0.105	1.920	2.494
200	200	0.558	2.721	0.797	45.586	49.663

Table 5.2: T200 LRE-A CPU averages

Table 5.2 shows the breakdown of average CPU times for each part of the LRE-A method. CPU1 is the problem initialisation time, CPU2 is the time to apply LR for the first time, CPU3 is the preprocessing time to filter out infeasible nodes and edges, and CPU4 is the bounded search time. CPU5 is the total processing time. The values for CPU1, CPU2 and CPU3 for Net1 and Net2 are similar, but the CPU4 values vary considerably. For the N100 and N200 problem sizes, the bounded search time CPU4 makes up most of the processing time. For Net1 N200 the search time accounts for 97% of the total average CPU time.

The next step is to replace the LR preprocessing step with the LR-ZR method, in order to find near-optimal values for the z upper bound, and then apply the bounded search as before. We denote this approach as LRE-ZR, and although we omit the 'A' we are still using aggregate constraints. We set the gap tolerance to 0% and search for optimality. We also solve these problems using CPlex with a gap tolerance of 0%, and let the problems run to completion. No preprocessing is used with CPlex. Table 5.3 shows these results.

	Num	N2	LRE-ZR CPU (sec)		No.	CPlex CPU	
		avg	avg	max	TO	avg	max
Net1 25	200	12.2 %	0.025	0.078	0	0.38	2.41
50	200	7.14 %	0.111	0.313	0	2.26	15.9
100	200	4.78 %	0.889	2.000	0	32.05	163.7
200	85	3.21 %	13.62	212.99	0	556.8	1635.9
Net2 25	200	9.26 %	0.027	0.157	0	0.40	3.94
50	200	4.15 %	0.102	0.468	0	3.25	16.8
100	200	2.51 %	0.938	3.953	0	35.98	236.9
200	200	2.10 %	9.011	28.08	0	447.8	2355.7

Table 5.3: T200 LRE-ZR results, 0% tolerance and 300 sec time limit

As before, the N2 column shows the average network size remaining after preprocessing, and the "No. TO" column shows the number of cases that timed out during the search (ie. none). The first thing to note is how much lower the N2 values are, compared with the previous table. The best average has been reduced from 31% to 2.1%. Indeed, as the network size increases, the N2 average decreases.

If we compare the CPlex run times, the result of running CPlex on the original networks without preprocessing, we see that the average CPU times for the Net1 and Net2 N200 sets are 41 and 49.7 times faster, respectively, using LRE-ZR. Also, comparing the average CPU times for LRE-A from Table 5.1 with the LRE-ZR times in Table 5.3 we see that LRE-ZR can be an order of magnitude faster than LRE-A.

All problems run to optimality within the time limit, so there are no timeout cases. The longest time is about 213 seconds, from the Net1 200 set. Of the 85 cases in this set, all but 10 complete in under 10 seconds. Excluding these 10 cases, the average search time for the remaining problems in this set is about 4% of the total running time.

		CPU1	CPU2	CPU3	CPU4	CPU5
	Num	Init	Setup	LR-ZR	Search	Total
Net1 50	200	0.015	0.031	0.065	0.000	0.111
100	200	0.086	0.316	0.477	**0.010**	0.889
200	85	0.515	2.449	3.179	**7.473**	13.616
Net2 50	200	0.015	0.034	0.054	0.000	0.102
100	200	0.093	0.375	0.470	**0.001**	0.938
200	200	0.558	2.736	5.533	**0.184**	9.011

Table 5.4: T200 LRE-ZR CPU averages

Table 5.4 shows, in the same manner as in Table 5.2, the breakdown of average CPU times for each part of the LRE-ZR method. The values for CPU1, CPU2 and CPU3 for Net1 and Net2 are similar, but the CPU4 values for Net2 are an order of magnitude smaller than for Net1. The reason for this difference is not immediately clear. However, recall that Net1 has paths that traverse the network diagonally, while the Net2 paths traverse horizontally. This means that Net2 paths will generally be shorter than the Net1 paths, resulting in a shorter search tree.

		CPlex no pp		CPlex LRpp		CPlex LR-ZR pp		LRE-ZR	
	Num	avg	max	avg	max	avg	max	avg	max
Net1 25	200	0.38	2.41	0.11	1.70	0.09	0.53	0.025	0.078
50	200	2.26	15.86	0.59	8.05	0.35	1.14	0.111	0.313
100	200	32.05	163.66	4.60	39.27	1.88	4.06	0.889	2.000
200	85	**556.79**	1635.91	**70.22**	253.37	**10.91**	18.70	**13.616**	212.985
Net2 25	200	0.40	3.94	0.15	1.47	0.09	0.84	0.027	0.157
50	200	3.25	16.75	0.83	16.84	0.31	1.42	0.102	0.468
100	200	35.98	236.92	4.97	52.15	1.80	6.50	0.938	3.953
200	200	**447.78**	2355.69	**115.14**	554.19	**13.98**	28.67	**9.011**	28.078

Table 5.5: CPlex CPU times for random valued networks Net1 and Net2, T200 cases

			LRE-ZR 1%				LRE-ZR 0%	
	Num	N2	%RE		CPU (sec)		CPU (sec)	
		avg	avg	max	avg	max	avg	max
Net1 50	1896	7.80%	0.043	0.782	0.106	0.328	0.107	0.344
100	495	4.92%	0.051	0.628	0.872	2.860	0.878	3.328
200	85	3.21%	0.065	0.294	8.537	212.9	14.735	213.0
Net2 50	5314	4.67%	0.034	0.938	0.102	0.438	0.102	0.484
100	2148	3.02%	0.053	0.852	0.985	4.86	0.987	4.91
200	523	2.18%	0.063	0.620	8.504	59.21	8.772	59.77

Table 5.6: Top5 LRE-ZR results, 1% and 0% tolerance with 300 sec time limit

So far we have not considered the effect of preprocessing on CPlex solution times. We have two methods for preprocessing, LR and LR-ZR. Table 5.5 shows the average and maximum CPU times for CPlex to solve to optimality the T200 cases for Net1 and Net2, with comparison values for the LRE-ZR method also solving to optimality.

The columns represent CPlex with no preprocessing, with LR preprocessing, and with LR-ZR preprocessing. The final columns show the CPU times for LRE-ZR. Just applying LR preprocessing dramatically reduces the CPU times for CPlex, while LR-ZR preprocessing further reduces the CPU time. For example, Net1 N200 average CPU time for CPlex with no preprocessing is 556.79 seconds, which reduces to 70.22 seconds with LR preprocessing, and is further reduced to 10.91 seconds with LR-ZR preprocessing. The corresponding average CPU time for LRE-ZR is 13.616 seconds, with a maximum time of about 213 seconds. In this problem set of 85 there are 3 cases which take more than 126 seconds to solve. Without these 3 cases, the average total solution time is 7.98 seconds. These 3 cases illustrate the problem with using a bounded search method that has, in the worst case, exponential time complexity.

Finally, we apply the LRE-ZR method to all the Net1 and Net2 cases with an initial LR duality gap over 5%. The number of such cases for N={25,50,100,200} was previously presented in Table 4.1. We ignore the N25 size since these problems are small enough to be solved easily by LRE-A. The LRE-ZR results for N={50,100,200}, using 1% and 0% tolerances, are shown in Table 5.6.

Across these larger problem sets, the N2 averages are consistent with the previous sets of 200 cases, as are the average CPU times to find optimality. What is interesting is how close the running times are with a 1% tolerance and a 0% tolerance. This indicates that the difference in CPU time to complete the search to optimality is not significant. For Net2 the average time taken to do the search is less than 4% of the total time, as was noted for the Net1 200 set. Thus for most problems, the search time is small compared to the preprocessing time. This is in marked contrast to the LRE-A method of Table 5.1, where the search time accounts for up to 97% of the total CPU time. The LR-ZR method is thus very effective at reducing the network size and thus dramatically speeding up the bounded search.

5.2 Submarine Transit Path Problem

In the previous section we applied LRE-A to the random valued network problems, and then compared this with LR-ZR preprocessing followed by the bounded search. In this section we repeat the same steps with the 1-speed Submarine Transit Path problem, using both LR-CP and LR-ZR to do the preprocessing.

The first step is to apply LRE-A, and again we use a time limit of 300 seconds but a tolerance of 0%. The results are shown in Table 5.7.

		N2	LRE-A %RE		CPU (sec)		No.
Fcn	Size	avg	avg	max	avg	max	TO
F1	20s	29.5 %	0.00	0.00	0.015	0.032	0
	40s	19.3 %	0.00	0.00	1.185	80.703	0
	80s	15.8 %	0.61	21.80	88.1	301.0	31
F2	20s	33.4 %	0.00	0.00	0.015	0.032	0
	40s	27.6 %	0.00	0.00	1.139	72.500	0
	80s	30.0 %	0.64	24.53	70.6	301.0	26
F3	20s	40.8 %	0.00	0.00	0.014	0.031	0
	40s	37.6 %	0.00	0.00	0.902	50.485	0
	80s	38.9 %	0.59	22.22	71.9	301.1	26
F5	20s	23.6 %	0.00	0.00	0.015	0.032	0
	40s	15.8 %	0.00	0.00	0.098	0.250	0
	80s	14.7 %	0.20	9.26	35.9	300.9	11

Table 5.7: 1-speed Submarine Transit Path Problem, LRE with 0% tolerance and 300 sec time limit

The N20 and N40 cases are all solved to optimality, but the N80 cases start to have timeout problems. Each of these problem sets has 120 cases, so to have 31 of them exceed the 5 minute time limit indicates that LR is unable to reduce the network size sufficiently.

Before applying the bounded search with LR-ZR as a preprocessing step, let us first apply LR-ZR on its own. Applying LR-ZR to the Submarine Transit Path problem is a straightforward matter, as the layout of the grid network for this problem is essentially the same as for the random valued network Net1. However, one small difference is the use of diagonal edges which necessitate using a wider meridian to produce an effective barrier. As before, the meridian consists of 3 rows of nodes. No other changes are required to the LR-ZR method.

The results, shown in Table 5.8, compare the results of LR-ZR with those of LR-CP using the set of 8 slack time thresholds. The initial LR results are included for comparison.

Fcn	Size	LR				LR-ZR				LR-CP sm08			
		%RE		CPU		%RE		CPU		%RE		CPU	
		avg	max	avg	max	avg	max	avg	max	avg	max	avg	max
F1	20s	1.98	28.45	0.015	0.031	0.00	0.00	0.016	0.032	0.00	0.56	0.016	0.032
	40s	2.65	38.92	0.088	0.125	0.00	0.00	0.099	0.250	0.01	0.53	0.096	0.203
	80s	2.81	29.37	0.507	0.750	-0.01	0.38	0.689	1.750	0.01	0.77	0.589	1.516
	160s	2.83	29.62	3.190	6.641	-0.14	0.50	3.706	11.172	-0.10	1.05	3.711	9.359
F2	20s	1.90	27.78	0.015	0.032	0.00	0.00	0.015	0.032	0.00	0.00	0.016	0.032
	40s	2.40	28.20	0.086	0.125	0.00	0.33	0.092	0.203	0.01	0.54	0.093	0.234
	80s	2.58	28.54	0.449	0.718	0.00	0.34	0.497	1.532	0.04	1.69	0.497	1.266
F3	20s	1.82	28.53	0.014	0.032	0.00	0.00	0.015	0.032	0.00	0.00	0.014	0.032
	40s	2.11	28.50	0.081	0.125	0.00	0.00	0.087	0.172	0.00	0.00	0.088	0.235
	80s	2.40	28.59	0.407	0.734	0.00	0.99	0.438	1.344	0.01	0.99	0.441	1.094
F5	20s	1.68	17.55	0.014	0.031	0.00	0.00	0.015	0.032	0.00	0.00	0.016	0.032
	40s	1.83	30.44	0.091	0.140	0.00	0.00	0.094	0.141	0.00	0.00	0.094	0.141
	80s	2.14	32.14	0.503	0.766	0.00	0.00	0.522	0.875	0.00	0.00	0.526	0.922
	160s	2.03	23.88	2.972	4.782	-0.19	0.00	3.099	5.813	-0.19	0.00	3.139	6.313

Table 5.8: Comparison of LR-ZR and LR-CP on 1-speed networks

The small negative RE averages for N80 and N160 indicate that LR-ZR and LR-CP are each able to find better solutions than CPlex did in the given time. LR-ZR is able to produce a maximum RE below 1% for all problem sets, which is better than the LR-CP worst-case maximum RE of 1.69% for F2-80s, while using approximately the same CPU time. This is impressive given the extra computation needed by LR-ZR.

Also note that for F5 both LR-ZR and LR-CP are able to find optimal solutions for all problem sizes without a bounded search. For the F5-160s problem set, there are 3 cases for which CPlex, having run for 4 hours, still had a gap of more than 5%. For these cases, both LR-ZR and LR-CP were able to find optimal solutions in about 5 seconds.

Adding the LRE-A bounded search method to find optimality after LR-ZR and LR-CP preprocessing, we get the results shown in Table 5.9.

A search time limit of 300 seconds is used, and this limit is reached in only one problem set, F1-160s. For LRE-ZR the time limit is reached a total of 10 times, and for LRE-CP a total of 11 times. Even though the time limit for the bounded search was reached in these 21 cases, in no case were the solutions worse than the CPlex results.

Fcn	Size	LR				LRE-ZR				LRE-CP sm08			
		%RE		CPU		%RE		CPU		%RE		CPU	
		avg	max	avg	max	avg	max	avg	max	avg	max	avg	max
F1	20s	1.98	28.45	0.015	0.031	0.00	0.00	0.016	0.032	0.00	0.00	0.016	0.032
	40s	2.65	38.92	0.088	0.125	0.00	0.00	0.099	0.234	0.00	0.00	0.097	0.203
	80s	2.81	29.37	0.507	0.750	-0.02	0.00	0.609	2.031	-0.02	0.00	0.615	2.016
	160s	2.83	29.62	3.190	6.641	-0.15	0.00	32.024	**311.23**	-0.13	0.19	34.438	**309.56**
F2	20s	1.90	27.78	0.015	0.032	0.00	0.00	0.015	0.047	0.00	0.00	0.016	0.032
	40s	2.40	28.20	0.086	0.125	0.00	0.00	0.093	0.188	0.00	0.00	0.093	0.219
	80s	2.58	28.54	0.449	0.718	0.00	0.00	0.516	1.860	0.00	0.00	0.525	2.906
F3	20s	1.82	28.53	0.014	0.032	0.00	0.00	0.015	0.032	0.00	0.00	0.014	0.032
	40s	2.11	28.50	0.081	0.125	0.00	0.00	0.088	0.188	0.00	0.00	0.087	0.234
	80s	2.40	28.59	0.407	0.734	0.00	0.00	0.457	1.531	0.00	0.00	0.462	1.844
F5	20s	1.68	17.55	0.014	0.031	0.00	0.00	0.018	0.032	0.00	0.00	0.014	0.032
	40s	1.83	30.44	0.091	0.140	0.00	0.00	0.095	0.141	0.00	0.00	0.095	0.141
	80s	2.14	32.14	0.503	0.766	0.00	0.00	0.528	0.984	0.00	0.00	0.529	0.922
	160s	2.03	23.88	2.972	4.782	-0.19	0.00	3.329	28.86	-0.19	0.00	3.369	28.890

Table 5.9: 1-speed Submarine Transit Path Problem, comparison of LRE-ZR and LRE-CP

Finally, we consider how LR-ZR may be used as a preprocessing step for CPlex. Recall that even with a time limit of up to 48 hours (for F1-160s), CPlex was unable to close the duality gap in many cases, especially for the N80 and N160 problem sizes. For example, the largest duality gap after 48 hours of computation was 10.83% for one case in F1-160s. Throughout this work we have been using these original CPlex reference solutions for the Submarine Transit Path problem with the knowledge that they may not be optimal. Now we have the chance to see if LR-ZR preprocessing can speed up CPlex.

Table 5.10 shows the results of using LR and LR-ZR as preprocessing steps prior to solving the mixed integer LP with CPlex with a 1 hour time limit. Where preprocessing has been used, the CPU times include this preprocessing time.

Fcn	Size	No preprocessing				LR preproc.		LR-ZR preproc.	
		Gap		CPU (sec)		CPU (sec)		CPU (sec)	
		avg	max	avg	max	avg	max	avg	max
F1	20s	0.00	0.00	0.39	3.69	0.05	0.14	0.05	0.16
	40s	0.00	0.00	9.90	134.38	0.30	3.58	0.25	0.56
	80s	0.13	5.64	2088.00	43299.10	1.71	18.06	1.45	4.69
	160s	2.04	10.83	47048.96	173013.00	**75.01**	3604.47	**7.52**	22.11
F2	20s	0.00	0.00	0.21	2.63	0.05	0.11	0.06	0.19
	40s	0.00	0.00	4.71	133.41	0.27	0.53	0.27	0.53
	80s	0.18	5.40	922.39	14400.70	1.96	6.59	1.87	5.20
F3	20s	0.00	0.00	0.21	1.19	0.06	0.11	0.06	0.11
	40s	0.00	0.00	5.90	208.77	0.31	0.56	0.31	0.55
	80s	0.16	6.14	788.11	14400.60	2.29	6.27	2.30	5.48
F5	20s	0.00	0.09	0.19	1.20	0.05	0.09	0.05	0.09
	40s	0.02	0.10	2.47	32.00	0.23	0.56	0.23	0.59
	80s	0.05	2.16	389.52	14400.50	1.33	5.41	1.37	5.41
	160s	0.63	9.27	3491.47	14410.70	13.04	81.63	13.29	81.53

Table 5.10: 1-speed Submarine Transit Path Problem, CPlex times with LR and LR-ZR preprocessing

The first surprise is that, with either LR or LR-ZR preprocessing, the average CPlex CPU times are up to 3 orders of magnitude faster for F1-80s. The second surprise is how close the average and maximum CPU times are for both preprocessing methods, with one notable exception: F1-160s. The large maximum time using LR preprocessing is due to one specific problem case, F1-160s (13,4). With no preprocessing, after 48 hours CPlex still had a gap of 4.88%. With only LR preprocessing, CPlex gets gap of 2.05% after 1 hour and 0.6% after 4 hours. But with LR-ZR preprocessing, which takes 8.515 seconds, CPlex is then able to find the optimal solution in a further 0.469 seconds, for a total time of 8.984 seconds. (By way of comparison, LRE-ZR was able to achieve an optimal result for this problem in 6.14 seconds.)

The most-improved case is (3,3) from the F1-160s set. The original CPlex solution (without preprocessing, taking 48 hours) had a gap of 8.9%. With LR preprocessing CPlex takes 23.265 to find an optimal solution; the combination of LR-ZR preprocessing and CPlex takes 8.156 seconds, and both methods have a duality gap of 0.004%. LRE-ZR finds the optimal solution in 4.844 seconds with a duality gap of 0.004%.

These results clearly demonstrate the value of using either LR or LR-ZR preprocessing to improve the CPlex running times on large problems. They also show that the LR-ZR method, while able to achieve near-optimal results on its own, is also superior to LR in preprocessing a large grid network prior to running CPlex.

5.3 Summary

In this chapter we have used the LR-ZR method as a preprocessing step prior to using a bounded search or CPlex to close the duality gap and find optimal solutions. In both cases, LR-ZR is superior to LR as a preprocessing method as it is more effective at reducing the network size.

On the random valued network problems Net1 and Net2,

- The combination of LR-ZR preprocessing and the bounded search, LRE-ZR, is up to 2 orders of magnitude faster than CPlex with no preprocessing.
- On larger network sizes, LRE-ZR is an order of magnitude faster than LRE-A.
- When LR-ZR is used as a preprocessing step for CPlex, the average CPU times are reduced by up to 3 orders of magnitude.
- Compared to LR preprocessing, LR-ZR preprocessing is able to reduce the average CPU times for CPlex a further 6 to 8 times.

On the Submarine Transit Path problem,

- LRE-ZR is up to 2 orders of magnitude faster than LRE-A
- LR and LR-ZR are comparable as a preprocessing step in reducing the CPlex CPU time.
- For one problem set (F1-160s), LR-ZR may further reduce the average CPlex CPU time by an order of magnitude, and the maximum CPU time is reduced by 2 orders of magnitude.

As a preprocessing step, LR-ZR allows CPlex to find optimal solutions for large network problems in a very short time, improving on Lagrange Relaxation preprocessing by up to an order of magnitude.

The LR-ZR method has been shown to be effective on both the Submarine Transit Path problem, with its smoothly varying edge costs, and the grid network problems Net1 and Net2 which have randomly varying edge costs and weights.

And finally, the LR-ZR method developed in this thesis, when used in conjunction with the bounded search method of Carlyle et al. (2007), improves on the best reported method from the literature by one to two orders of magnitude.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The Constrained Shortest Path Problem (CSPP) is a well-known problem in optimisation and is known to be NP-hard. Until recently, label setting methods have been regarded as the best approach, but these methods suffer from space complexity issues which limit the size of networks that may be considered. More recent work has focussed on using Lagrange Relaxation to perform preprocessing, with a bounded search to close the remaining duality gap. Lagrange Relaxation is a simple yet powerful technique for solving constrained optimisation problems, but it may not achieve optimal or near-optimal results on its own. Often, large duality gaps may be present, and the bounded search may then take exponential time.

This thesis builds on the latter approach. We make a simple modification to Dijkstra's algorithm, one that does not involve any additional work, in order to generate an estimate of path time at every node. We then combine this information with a heuristic method for determining where to apply a bisecting meridian, which temporarily constrains the network to find improved solutions. This hybrid method, combining Lagrange Relaxation with a simple heuristic based on constraint programming, is shown to be superior at reducing the network at the preprocessing stage, delivering optimal or near-optmal solutions with computation times that are only a few times longer than the initial Lagrange Relaxation time.

The following is a brief outline of the contribution of this thesis.

In this thesis we consider rectangular grid networks with a single side constraint, similar to the kind that have been studied in the literature. We select problems that have a large duality gap when Lagrange Relaxation is applied, as these are the problems of interest. We study a large number of problems from two different types of grid network problems.

The first network studied is a Submarine Transit Path problem in which the transit field contains four sonar detectors at known locations, each with the same detection profile, and the submarine capable of 2 speeds. The single side constraint is an upper bound on the total transit time. For the original 2-speed case the initial duality gap may be as high as 10%, but for the 1-speed case the initial duality gap may be as high as 30%. The first hybrid LR-CP method uses a single central bisecting meridian along which an additional constraint on slack time is applied, using Lagrange Relaxation to propagate this constraint. This method is able to produce improved solutions that are generally within 1% of optimal. Using the computation time for the initial Lagrange Relaxation as a baseline, the total computation time for the first hybrid method is on average about 2 times this.

The second problem is a grid network from the literature. Edge costs, times, and lengths are randomly generated. As the values given in the literature problems do not yield problems with a sufficiently high duality gap, the ranges of the random edge values are varied slightly and from a population of approximately 100,000 possible problems, only the worst 200 from each problem size are chosen for study. These problems have an initial duality gap as high as 40%. The second hybrid method, LR-ZR, uses a calculated ratio and multiple bisecting meridians, and is able to produce solutions that are generally within 0.1% of optimal, with computation times that are on average 2 to 4 times the initial Lagrange Relaxation time.

The second hybrid method, LR-ZR, is also used as a preprocessing step on both network problems. This method is able to reduce the network up to 10 times more than Lagrange Relaxation on its own. For the Submarine Transit Path problem, when LR-ZR is used to perform preprocessing the average CPU time for CPlex to solve to optimality is reduced by up to 3 orders of magnitude. Compared with Lagrange Relaxation preprocessing, the LR-ZR preprocessing is able to further reduce the CPlex CPU time by up to an order of magnitude on one problem set.

On the random valued network problems, Net1 and Net2, LR-ZR preprocessing is able to reduce average CPlex CPU times by 6 to 8 times more than if Lagrange Relaxation preprocessing is used.

A bounded search method from the literature, which uses aggregate constraints to limit the search, is also used to close the gap and find optimal solutions. For the random valued network problems, LR-ZR preprocessing combined with the bounded search is able to find optimal solutions up to 50 times faster than using CPlex without preprocessing. Compared to the LRE-A method of Carlyle et al. (2007), this LRE-ZR method is up to an order of magnitude faster on the random valued networks, and up to 2 orders of magnitude faster on the Submarine Transit Path problem.

6.2 Future Work

In this thesis we have limited the problems under study to square grid networks with a single side constraint, in order to fully test the effectiveness of these hybrid methods on simple problems.

One obvious extension is to include problems with more than one side constraint. With multiple side constraints, finding Lagrange multipliers cannot be done with Kelley's cutting plane algorithm, but must use another method such as subgradient optimisation. This may require more computation, leading to longer computation times for Lagrange Relaxation, which is the central algorithm used in this thesis. In addition, subgradient optimisation may or may not provide the same quality of lower bounds, an issue that would need to be explored.

Future work may also include networks that are not grids, that is, networks with a more arbitrary structure. The two hybrid methods described in this work rely on the existence of a bisecting meridian to temporarily constrain the network. As long as such a meridian can be constructed for any given network, there is every reason to believe that these hybrid methods could be applied to any network structure or problem.

Appendix A

Sonar Detector Locations

The coordinates x, y for the 4 detectors in each of the 30 patterns are given below. The values are km from the origin, with x and y in the range 0 to 79.

```
static int detectorArray[][numDetectors][2] = {
   { {65,13}, { 3,60}, {21,13}, {19, 8} }, //  1
   { { 7,20}, {38,66}, {35,45}, {28,56} }, //  2
   { {17,25}, {69,41}, {55,27}, {37,45} }, //  3
   { {55,62}, {53,75}, { 6,20}, {59,49} }, //  4
   { {15,61}, { 0,25}, {41,44}, { 0,78} }, //  5
   { {65,16}, {18,22}, {23,60}, {79,10} }, //  6
   { {18,33}, {66,76}, {17,78}, {37,15} }, //  7
   { {77,21}, {60,75}, {51,55}, { 7, 9} }, //  8
   { {35,22}, {48,12}, {21,47}, {56, 7} }, //  9
   { { 6,35}, {55,58}, {10,38}, { 8,39} }, // 10
   { {64,31}, {14,10}, {48, 4}, {42, 5} }, // 11
   { {26,57}, {57,29}, {22,32}, {21,58} }, // 12
   { { 9,62}, {47,45}, {50,41}, { 1,58} }, // 13
   { {49,34}, {62, 7}, {30,33}, {53,71} }, // 14
   { {74, 3}, {18,22}, {32,54}, {68,56} }, // 15
   { {12,66}, {44,56}, {76,15}, { 5,28} }, // 16
   { {66,20}, {75, 3}, { 7,55}, {66,72} }, // 17
   { {37,11}, {71,26}, {51,67}, {17,50} }, // 18
```

```
    { {44,70}, {65,46}, {36, 5}, {53,61} }, // 19
    { {75,27}, {49,47}, {33,26}, {72,56} }, // 20
    { {77,51}, {15, 4}, {61,65}, {53,24} }, // 21
    { {38,28}, {37,65}, {18,62}, { 3,31} }, // 22
    { {70,17}, { 6,29}, {62,58}, {35, 9} }, // 23
    { {22,24}, {68, 3}, {73, 0}, {18,13} }, // 24
    { {52,43}, {69,66}, {49,35}, {47,13} }, // 25
    { {79,48}, {39, 6}, {23,74}, {25,65} }, // 26
    { { 4,24}, {59,29}, {32,71}, {38,64} }, // 27
    { {78,51}, {14,75}, {16,18}, {57,25} }, // 28
    { {28,69}, {25,11}, {39, 8}, { 5, 4} }, // 29
    { {12,61}, {49,33}, {26,43}, {51,38} }  // 30
};
```

Bibliography

[1] Y.P. Aneja, V. Aggarwal, and K.P.K. Nair. Shortest chain subject to side constraints. *Networks*, 13:295–302, 1983.

[2] Y.P. Aneja and K.P.K. Nair. The constrained shortest path problem. *Naval Research Logistics Quarterly*, 25:549–553, 1978.

[3] M.O. Ball, T.L. Magnanti, C.L. Monma, and G. Nemhausser. Network routing. In *Handbooks in Operations Research and Management Science Vol 8*. North Holland, Amsterdam, 1995.

[4] J.E. Beasley and N. Christofides. An algorithm for the resource constrained shortest path problem. *Networks*, 19:379–394, 1989.

[5] Dimitri P. Bersekas. *Convex Analysis and Optimization.* Athena Scientific, Belmont, Massachusetts, 2003.

[6] Thomas H. Byers and Michael S. Waterman. Determining all optimal and near-optimal solutions when solving shortest path problems by dynamic programming. *Operations Research*, 32(6):1381–1384, November 1984.

[7] L. Caccetta, I. Loosen, and V. Rehbock. Computational methods for a class of discrete valued optimal control problems. In *Proceedings of the 7th International Conference on Optimization: Techniques and Applications (ICOTA7), Kobe, Japan*, Curtin University of Technology, Western Australia, 2007.

[8] W. Matthew Carlyle and R. Kevin Wood. Lagrangian relaxation and enumeration for solving constrained shortest-path problems. In *Proceedings*

of the 38th Annual ORSNZ Conference, Operations Research Department, Naval Postgraduate School, Monterey, California, 2003.

[9] W.M. Carlyle, J.O. Royset, and R.K. Wood. Lagrangian relaxation and enumeration for solving constrained shortest-path problems. in review, March 2007.

[10] W.M. Carlyle, J.O. Royset, and R.K. Wood. Routing military aircraft with a constrained shortest-path algorithm. in review, April 2007.

[11] W.M. Carlyle and R.K. Wood. Near-shortest and k-shortest simple paths. *Networks*, 46:98–109, 2003.

[12] M. Desrochers and F. Soumis. A generalized permanent labelling algorithm for the shortest path problem with time windows. *INFOR*, 26(3):191–212, 1988.

[13] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[14] I. Dumitrescu and N. Boland. Algorithms for the weight constrained shortest path problem. *International Transactions in Operations Research*, 8:15–29, 2001.

[15] I. Dumitrescu and N. Boland. Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem. *Networks*, 42(3):135–153, 2003.

[16] Paul C. Etter. *Underwater Acoustic Modeling: Principles, Techniques and Applications.* Elsevier Applied Science, 1991.

[17] M.L. Fisher. The lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–18, January 1981.

[18] B.L. Fox and D.M. Landi. Searching for the multiplier in one-constraint optimization problems. *Operations Research*, 18(2):253–262, March 1970.

[19] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimisation algorithms. *Journal of the ACM*, 34(3):596–615, July 1987.

[20] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the Association for Computing Machinery*, 34(3):596–615, July 1987.

[21] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.M. Freeman and Co., San Francisco, 1979.

[22] T. Gellermann, M. Sellmann, and R. Wright. Shorter path constraints for the resource constrained shortest path problem. In *Lecture Notes in Computer Science, Issue 3524*, pages 201–216. Springer-Verlag, Germany, 2005.

[23] C.L. Hallam. *Hierarchical Path Generation: An Application to Submarine Transit Paths.* Honours Dissertation, Murdoch University, Western Australia, 1997.

[24] G. Handler and I. Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10:293–310, 1980.

[25] R. Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations Research*, 17(1):36–42, February 1992.

[26] J. Hooker. *Logic-Based Methods for Optimization - Combining Optimization and Constraint Satisfaction.* John Wiley and Sons, New York, 2000.

[27] Hugh Everett III. Generalized lagrange multiplier method for solving problems of optimum allocation of resources. *Operations Research*, 11(3):399–417, May 1963.

[28] J.M. Jaffe. Algorithms for finding paths with multiple constraints. *Networks*, 14:95–116, 1984.

[29] L.S. Jennings, M.E. Fisher, K.L. Teo, , and C.J. Goh. *MISER3 Optimal Control Software: Theory and User Manual Version 3.* Department of Mathematics, The University of Western Australia, Nedlands, WA 6907, Australia, 2004.

[30] H.C. Joksch. The shortest route problem with constraints. *Journal of Mathematical Analysis and Applications*, 14:191–197, 1966.

[31] J.E. Kelley Jr. The cutting-plane method for solving convex programs. *Journal of the SIAM*, 8(4):703–712, December 1960.

[32] Myungsoo Jun and Raffaello D'Andrea. Path planning for unmanned aerial vehicles in uncertain and adversarial environments. In S. Butenko et al., editor, *Cooperative Control: Models, Applications and Algorithms*, chapter 6, pages 95–110. Kluwer Academic Publishers, 2003.

[33] F. Kuipers, T. Korkmaz, M. Krunz, and P. Van Mieghem. Performance evaluation of constraint-based path selection algorithms. *IEEE Network*, 2004.

[34] Kurt Mehlhorn and Mark Ziegelmann. Resource constrained shortest paths. In *European Symposium on Algorithms*, pages 326–337, 2000.

[35] George L. Nemhauser and Zev Ullmann. A note on the generalized lagrange multiplier solution to an integer programming problem. *Operations Research*, 16(2):450–453, 1968.

[36] A.K. Pujari, S. Agarwal, and V.P. Gulati. A note on the constrained shortest path problem. *Naval Research Logistics Quarterly*, 31:87–89, 1984.

[37] C. Ribeiro and M. Minoux. Solving hard constrained shortest path problems by lagrangian relaxation and branch-and-bound algorithms. *Methods of Operations Research*, 53:303–316, 1986.

[38] R.T. Rockafellar. Lagrange multipliers and optimality. *SAIM Review*, 35(2):183–238, June 1993.

[39] M. Sellmann. Cost-based filtering for shorter path constraints. In *Proceedings of the 9th International Conference on Principles and Practise of Constraint Programming (CP). Volume LNCS 2833., Springer-Verlag*, pages 694–708, 2003.

[40] Meinolf Sellmann and Torsten Fahle. Constraint programming based lagrangian relaxation for the automatic recording problem. *Annals of Operations Research*, 118:17–33, 2003.

[41] C.C. Skiscim and B.L. Golden. Solving k-shortest and constrained shortest path problems efficiently. *Annals of Operations Research*, 20:249–282, 1989.

[42] Maarten van Emden. Review of: Principles of constraint programming by krzysztof r. apt. available online, published in SIAM Review 2006, 48(2), 2003.

[43] J.Y. Yen. Finding the k-shortest, loopless paths in a network. *Management Science*, 17:711–715, 1971.

[44] Michael Zabarankin, Stanislav Uryasev, and Robert Murphey. Aircraft routing under the risk of detection. *Wiley InterScience*, 2006.

[45] Michael Zabarankin, Stanislav Uryasev, and Panos Pardalos. Optimal risk path algorithms. In R. Murphey and P.M. Paradalos, editors, *Cooperative Control and Optimisation*, chapter 13, pages 273–298. Kluwer Academic Publishers, 2002.

[46] Mark Ziegelmann. *Constrained Shortest Paths and Related Problems.* PhD thesis, Universitaet des Saarlandes, 2001.