Applying Lagrangian Relaxation to the Submarine Transit Problem

L. Caccetta[a], M. Grigoleit[a], and V. Rehbock[a]

Abstract

A new approach to the constrained shortest path problem (CSPP) is applied in the context of submarine path planning through a region of sonar detectors. This uses fast, convergent methods to find the optimal Lagrange multiplier and Dijkstra's algorithm to find initial solutions. On a test set of 120 cases, the resulting paths are almost always within 3% of optimal, with solution times under 1 second on a 2.4 GHz desktop computer. Given the performance of this method on problems with over 6400 nodes, extension to much larger problem sizes appears feasible.

Introduction

The task of planning a submarine path through a field of sonar detectors has strategic importance in marine defence. This is a difficult problem for many reasons: the constrained shortest path problem (CSPP) is known to be NP-complete; results reported to date tend to involve relatively small networks, up to 1000 nodes (to adequately model ocean conditions requires much larger networks); and computation time is critical, especially in tactical situations.

Of particular importance, then, is the tradeoff between computation time and optimality. The desired goal is to find near optimal solutions with minimal computation time. We are primarily concerned with very large networks, of at least 6400 nodes, which allow us to model a square transit field with an 80 km side. This gives a resolution of 1.0 km per edge.

This paper explores the application of Lagrangian relaxation to the submarine transit problem in order to evaluate how well it can produce near optimal results on large networks in minimal time.

Problem Definition

The *field of traversal* is represented as a square array of nodes, with horizontal, vertical and diagonal arcs between adjacent nodes. The vehicle is assumed to travel from the origin (0,0) in the lower left to (*N, N*) in the upper right hand corner. Each sonar detector is a point in the field, with the probability of detection at each node in the *NxN* array based on its distance from the detector. We consider two possible vehicle speeds, 8 and 14 km/h. The detection probability as a function of distance is given in Figure 1. Notice that the probability is very high at the origin then generally drops off with distance. Note the second peak at about 64 km which may be a feature of underwater reflection and resonance.

A path consists of a list of adjacent nodes from start to finish. There is no cost associated with changing speed from one node to the next.

Several simplifying assumptions are made including:

- 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
- All detectors share the same detector function.

The transit field is modelled as a square array of nodes in which every pair of adjacent nodes is connected by 4 edges for both speeds in both directions. This includes diagonal edges as well as horizontal and vertical ones. An array 80km square contains 6561 nodes and 103,040 edges.

[a] West Australian Centre of Excellence in Industrial Optimisation (WACEIO), Department of Mathematics and Statistics, Curtin University, GPO Box U 1987, Perth, WA 6845, Australia

Each edge has a cost, which is the probability of detection (calculated from the given detection ans sensor location patterns), and a transit time. The only constraint is an upper bound on the transit time. The optimisation problem is formulated as minimising the cost function within the given transit time constraint.

For test purposes we use a reference set of 30 detector patterns, each containing 4 detectors. Each of the 30 problem sets has the same lower bound on the time constraint (11.43 hours) and an upper bound that varies but is around 19 hours. We use 4 different time constraints per problem, chosen at 20, 40, 60 and 80 percent of the difference between upper and lower bounds, giving a total of 120 test cases.

Approach

The approach we describe in this paper is derived from the Lagrangian relaxation (LR) method described by Carlyle & Wood in [2]. For a cost function given by

$$z = \sum_i c_i x_i$$

we add the single constraint (transit time of each edge) into the objective function to give

$$zL = \sum_i (c_i + \lambda F_i) x_i - \lambda g$$

where
c is the edge cost
c' is the modified edge cost, c + λF
x is the binary variable for edge *i* (0 means edge not used)
λ is the Lagrange muliplier
F is the transit time of the edge
g is the upper time bound
z is the cost of the path using original edge costs c
zL is the cost of the path using modified edge costs c'

The two main techniques involved are: (1) the calculation of the Lagrange multiplier λ; and (2) the application of Dijkstra's algorithm to find the shortest path using modified edge costs. We discuss the calculation of λ later.

We have implemented Dijkstra's algorithm with a couple of modifications. The first is the use of an insertion sort to manage the list of active nodes under consideration, as opposed to using a Fibonacci heap [3]. Secondly, we exploit the structure of the regular grid network to discard nodes after they fall behind the wave front, thus leaving O(*n*) nodes in the list. We thus achieve speeds comparable to F-heaps without the code complexity.

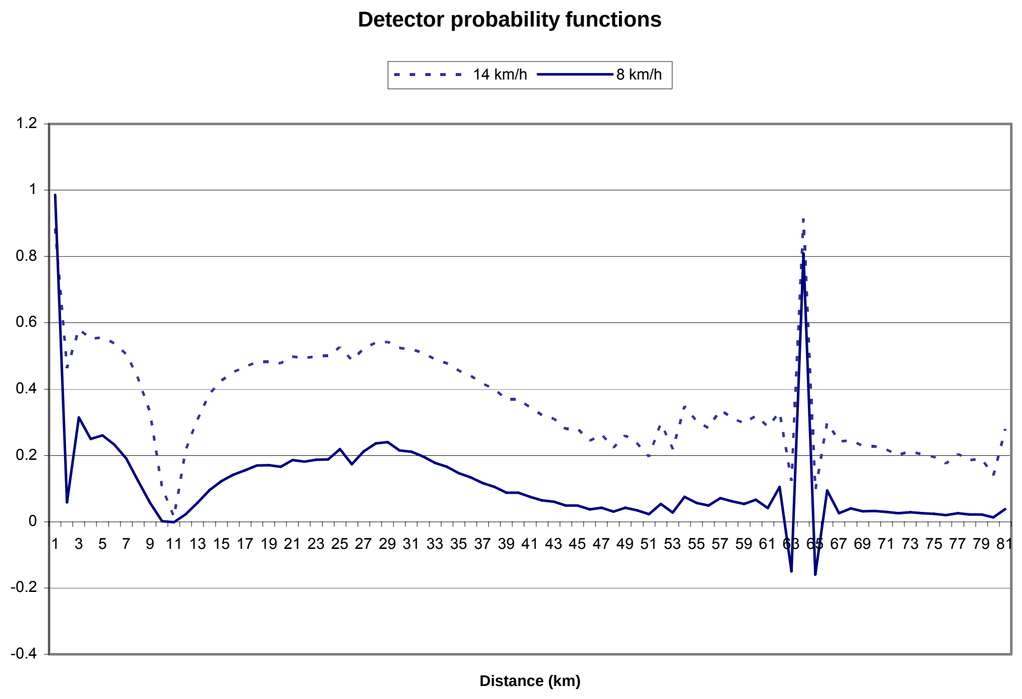

Figure 1 - Detection probability vs Distance (km)

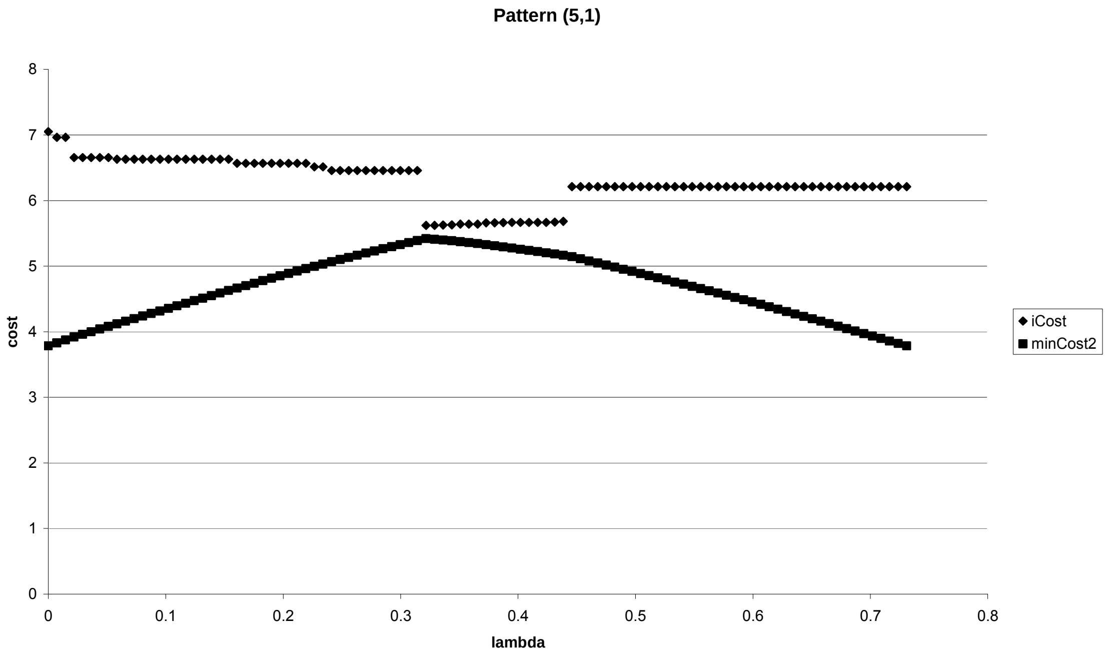

Figure 2 – initial path cost and minCost2 vs λ

The first pass of the method involves calculating the shortest path using Dijkstra's algorithm on the original edge costs. The cost of this path is z_{min}. If the transit time of this path is within the upper time bound, we have an optimal solution. If not, we proceed with the Lagrangian relaxation (LR).

LR involves finding the value of λ for which the cost of the shortest path using the modified edge costs – minCost2 – is maximum. As can be seen from Figure 2, this corresponds to the minimum value for z. The graph shows a plot of initial path cost (iCost) and the minCost2 values as a function of λ. The value iCost is simply the cost of the minCost2 path using the original edge costs.

What we need is an efficient way to find the value of λ_{max} at the peak of the minCost2 curve. The y-intercept of this curve at λ=0 corresponds to z_{min}. At the right of the curve we want to find the value of λ at which minCost2 is again equal to z_{min}. This we call λ_{min}. Knowing these two values for λ will help us find λ_{max}.

First, here is the method for finding λ_{min}.

```
set λ=1.0
do:
   set all modified edge costs c'
      using this λ
   apply Dijkstra's algorithm to
      get zL
   extract the resulting path
      based on c' edge costs
   using this set of edges, vary
      λ downwards until zL is just
      below minCost
   set new value of λ
until λ doesn't change from the
previous iteration.
```

The calculation that uses the edge costs of an initial solution and varies λ can be done in O(n) time. This usually takes only 3 or 4 iterations.

To calculate λ_{max} we first take the two endpoints of the curve where λ=0 and λ=λ_{min} and get initial mincost2 solutions using these two values of λ. We approximate tangent lines to the curve at these points by calculating the zL value of λ=0.01 and λ=0.95* λ_{min}. and then find the intersection point of these two lines. This gives us a first approximation of λ_{max}. We then update either the left or right tangent line based on the slope at that point being negative or positive, and update the right or left value of λ. This is repeated until λ does not change. In practise this may take up to 8 iterations, but usually 4 to 6 are sufficient.

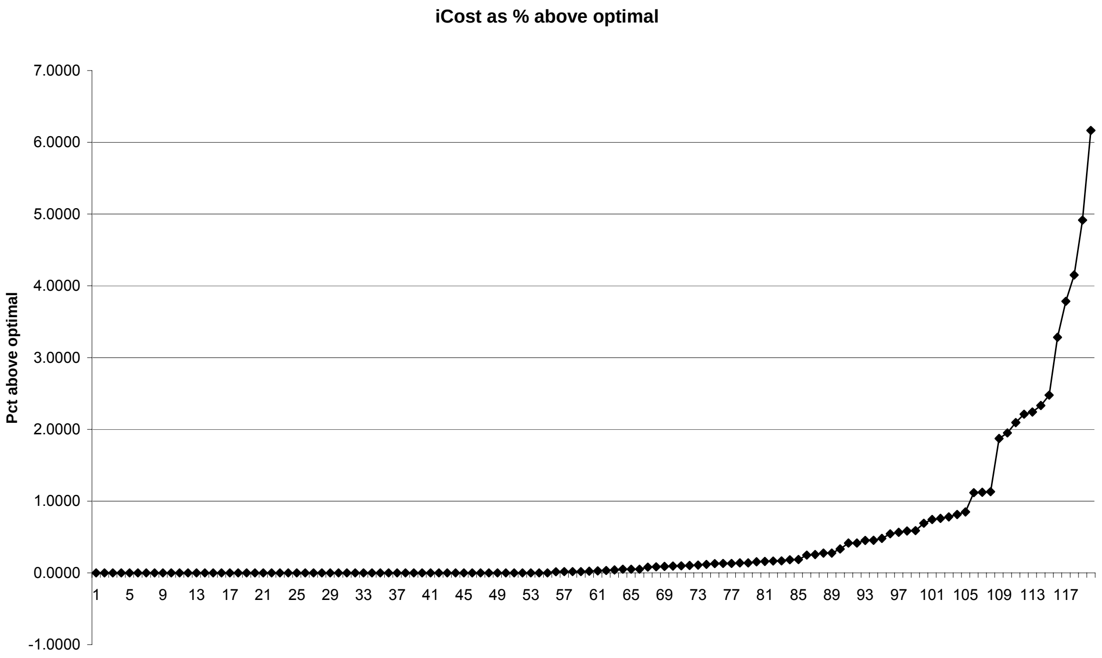

Figure 3 – initial path cost above optimal for all 120 cases

```
do:
  calc the intersection point of
    the two tangent lines
  set all modified edge costs c'
    using this λ
  apply Dijkstra's algorithm to
    get zL for that λ based on
    the slope of the tangent at
    that point, update either
    the right or left value for
    λ
until λ does not change
```

We thus have fast methods for calculating both λ_{min} and λ_{max}.

Reference values

For comparison purposes, a mixed-integer linear program (MILP) model was used to generate optimal solutions with CPlex. We use 3 significant figures for all edge costs (eg. 0.000 to 0.999) in order to allow CPlex to converge more easily. The code for both methods was written in C++ using STL container classes, and run on a 2.4 GHz desktop computer.

Initial Paths

If we limit our interest to initial paths for the 80x80 problem, we get the results shown in Figure 3 and Table 1. Note that all but 5 of the 120 results are within 3% of optimal, with the worst case just over 6% above optimal. The 5 results above 3% are the result of only 3 detector patterns.

Time bound	CPlex cpu (sec)	LR cpu (sec)	LR % above optimal
0.20	619.90	0.70	0.56
0.40	527.48	0.63	0.82
0.60	506.78	0.52	0.30
0.80	59.54	0.40	0.12
All Problems	428.42	0.56	0.45

Table 1 – Summary of 80x80 results

The computation time for each case is about half a second on average, and about half of this time is taken in calculating the initial edge costs during setup.

There is some variation by time bound, but not as great as one might expect. What is impressive is the fact that, except for a small number of cases, we get results that are within 3% of optimal and our CPU times are 2 to 3 orders of magnitude smaller than using CPlex.

Of the 120 cases, one third of initial shortest path solutions will be within the time bound, and thus be optimal. One third will be under time, and one third will be over the time bound. However, because we have fast and slow edges in parallel, it is always possible to adjust the initial path using edge swaps to adjust the path time up or down to be within the time bound.

In general, these initial paths follow very closely the optimal paths generated by CPlex.

The CPU time for the LR method is dominated by number of times that Dijkstra's algorithm is called, which on average is called 17 times per problem. Across problem sizes of n = (20, 40, 80, 120, 160) nodes per edge, we observe $O(n^3)$ computation time.

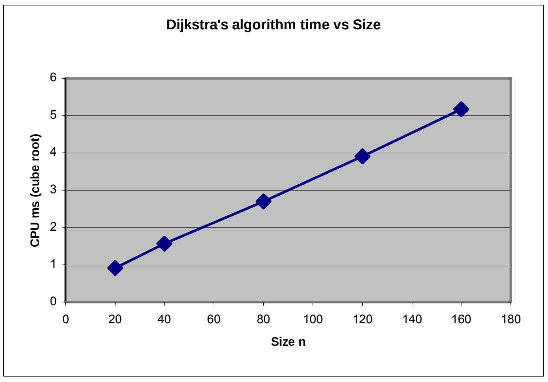

Figure 4 - Dijkstra's algorithm time vs grid size

As an upper limit, consider scale of 250m per edge, with 321 nodes per side of an 80km grid, we have 103,041 nodes and 1,640,960 edges. If Dijkstra's algorithm is called 17 times and takes approximately 1 second, this gives an estimated solution time of about 17 seconds. In practise, a problem of this size is solved by the LR method with an average time of about 16 seconds. This corresponds to a MILP problem of 1.6m decision variables and 103,041 constraints.

There is a duality gap between minCost2 and optCost. For half of the cases, this is below 0.1% but for 10/120 cases it ranges from 2 to 8%. The gap between minCost2 and iCost is similar, but the optimal cost may lie anywhere within this gap, and thus is not an especially good predictor of optimality.

Search results

The tree-based enumeration search described in [2] doesn't work very well here. The tree-based enumeration depends on having good bounds for z, zL, and time (the side constraint). Because of the structure of this problem with parallel fast/slow edges, it is always possible to find shorter (quicker) paths, so the intermediate time bound is of no use here. This is one practical problem for which an optimal λ does not exponentially reduce the solution space, as claimed in [2].

Despite the long search times, a time limit of 5 minutes was implemented and the search run on all 120 cases for 3 different scales. The results are shown in Table 2 below. Even for the 20x20 size, there is one case that takes an inordinately long time.

Conclusions and Further Work

We have applied the Lagrangian relaxation approach described by Carlyle & Wood to the submarine transit problem. We have developed a quick way to find the Lagrange multiplier for a singly constrained problem, and the computation times are 2 to 3 orders of magnitude faster than using CPlex, while achieving results that are usually withing 3% of optimal.

Unfortunately, the bounded tree search of Carlyle & Wood is not suitable for this problem due to weak time bounds and the duality gap that exists in this problem.

Further work might look at problems with more than one side constraint, and also finding a better search method to improve on the already good initial solutions while keeping the computation time small.

	CPU times for LR search – number of cases			
Size	< 1 sec	1-60 sec	1-5 min	> 5 min
20 x 20	118	1	0	1
40 x 40	89	11	4	16
80 x 80	72	12	1	35

Table 2 – Summary of search times

Acknowledgements

This project has been supported by the Australian Research Council grant (DP 034639). We are grateful to the ARC for their support. The authors would also like to thank the Defence Science Technology Organisation for providing the data on the sonar detector functions.

References

[1] V. Rehbock, L. Caccetta, C.L. Hallam and R. O'Dowd, Optimal Submarine Transit Paths Through Sonar Fields, Department of Mathematics and Statistics, Curtin University of Technology, Research Report, 2000.

[2] W. Matthew Carlyle, R. Kevin Wood, Lagrangian Relaxation and Enumeration for Solving Constrained Shortest-Path Problems, Operations Research Department, Naval Postgraduate School, Monterey, California, 2003.

[3] Michael L. Fredman, Robert Endre Tarjan, Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms, Journal of the Association for Computing Machinery, Vol. 34, No. 3, July 1987, Pages 596-615.