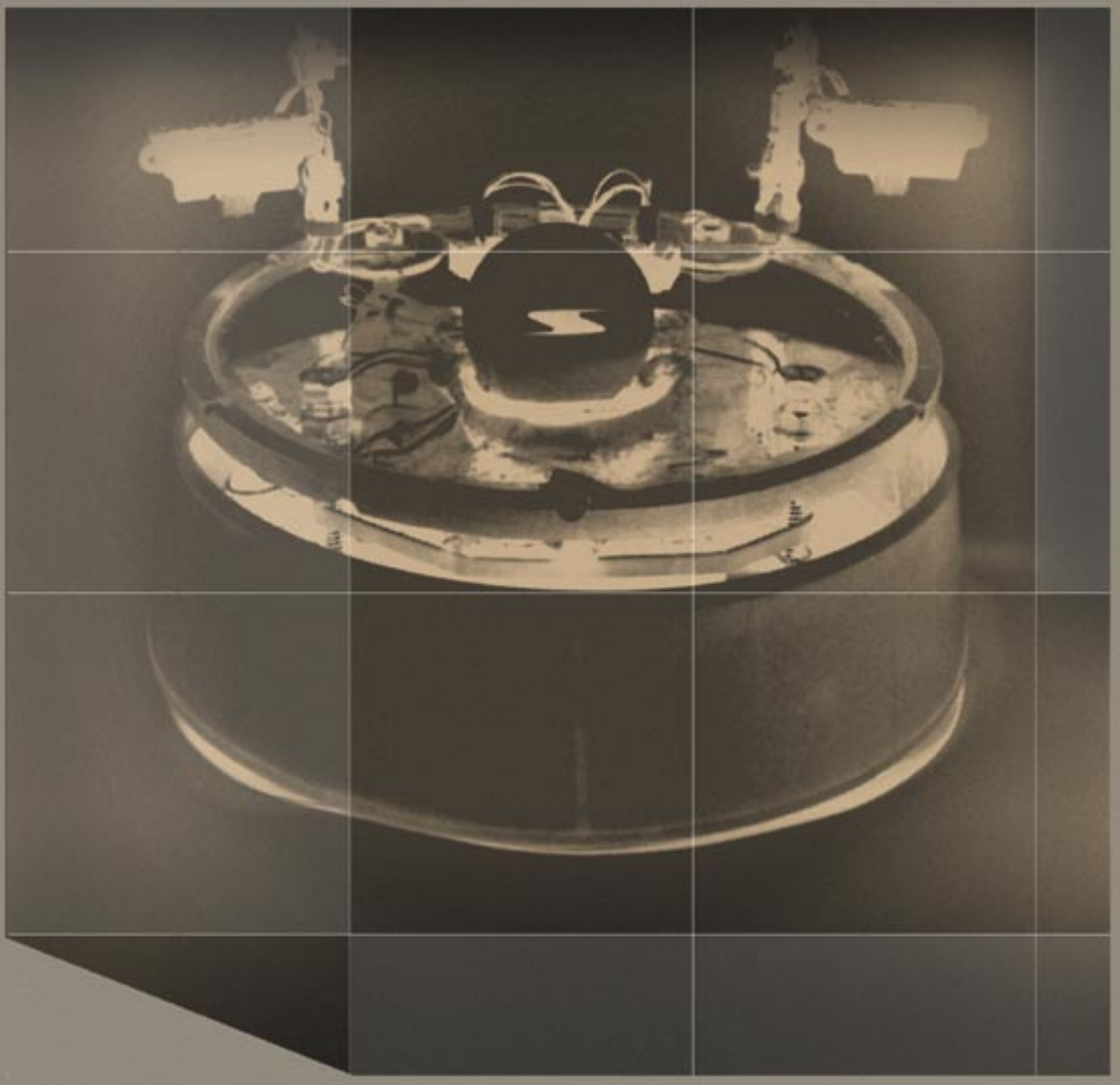


A DYNAMICAL PROGRAMMING SOLUTION FOR SHORTEST PATH ITINERARIES IN ROBOTICS

author: martin talbot

ryerson university

supervisor: dr. randall pyke

A DYNAMICAL PROGRAMMING SOLUTION FOR SHORTEST PATH ITINERARIES IN ROBOTICS

Author: Martin Talbot
mt@140k.com

Supervisor: Dr Randall Pyke

April 2003

TABLE OF CONTENTS

0. ABSTRACT

In robotics, more precisely Autonomous Mobile Robotics (AMR), robots, much like human beings, are confronted regularly with the problem of finding the best path to take from a source location to a destination location. This is an optimization concern, since the robot wants to minimize its cost in time or in energy while achieving its goal. Different algorithms exist for shortest path computation; the famous Dijkstra's Shortest Path Algorithm will solve single-source shortest path problems in near linear time[1]. However, for certain complex optimization path-planning problems, this algorithm alone is insufficient. We will study a dynamical formulation using Integer Programming (IP) to solve complex path-planning problems in robotics.

1. OVERVIEW

Figure 1 represents a map; the reader can imagine a building's floor viewed from above. The nodes symbolize the possible starting and arriving locations allowed (source and destination), and the edges represent the possible paths that the robot can take to travel from node to node (location to location). The values in the boxes attached to each pair of edges indicate the cost to travel along them; in this case, we assume the units are minutes. Note that integers are used for simplicity.

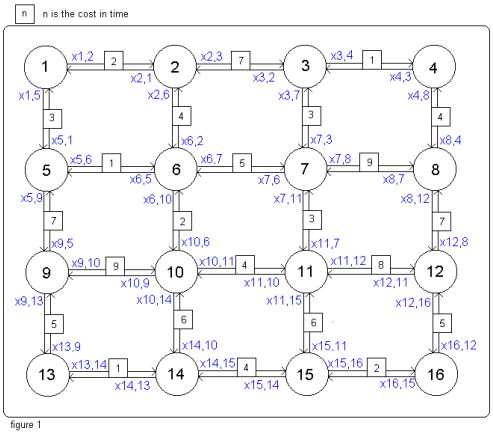


figure 1

A **non-complex shortest** or **trivial shortest path** problem is the shortest path computation between a source and a destination. For example, referring to Figure 1, finding the shortest path between node 1 and node 7, or node 9 and node 10. The next paragraph presents a more complex shortest path problem.

Imagine a robot having to perform security runs at different locations in a building (figure 1). Presume that a robot who is presently at node 1 is asked to visit node 13, node 11, and node 8 in any order, then stop and wait at node 7 until it gets a new mission. From this, the robot must start at its actual position (node 1), finish at node 7 and visit nodes 13, 11 and 8 in such a way that the sum of the costs of the shortest paths between these five nodes must be optimized to a minimum. We define such missions as "itineraries" or "tours". An **itinerary** or a **tour** requires a two-stage optimization algorithm. It is the optimal sequence or concatenation of trivial shortest paths for a mission. In the particular example above, the tour should be expressed as the optimal concatenation of 4 shortest paths. This problem is known to be NP-complete[2]. We remark on similarities to the Travelling Sales Person (TSP) problem, although with an important distinction (for an example, see [14] chapter 9). A typical TSP problem has n cities to visit, and each city must be visited exactly once in a way to minimize the total distance travelled (cost), and avoiding **subtours**[3]. Here we have 16 "cities", but only 5 must be visited; the other 11 are transitory nodes/cities that may or may no be visited. This problem is complex, and becomes more difficult if some edges must be avoided during a tour, for instance because of construction in the corridor `x1,5` and `x5,1`.

This security run scenario shows that Dijkstra's Shortest Path algorithm alone is not enough to solve the problem under consideration. Using the best algorithms available, NP-complete problems can be solved in exponential running time on nondeterministic machines [13]. Unfortunately, because of the 11 transitory-node particularities, I could not find an algorithm that would solve such a problem at once. Since a traditional shortest path algorithm such as Dijkstra's can only solve one part of the problem, another distinct algorithm is needed to compute the final solution. However, this is not true if IP is used. I will present how the simplex algorithm can be used as a unique algorithm to solve such intricate problems. Before we start, I would like to talk briefly about performance, sensitivity analysis, and duality.

2. ALGORITHM'S PERFORMANCE

Although the average empirical complexity of the simplex algorithm is $O(m^2n)$, in practice the computation has been observed to take roughly m iterations, and seldom more than $3m$ iterations[4] [8]. Furthermore, the type of problem addressed here implies sparse matrices; this favours fast computation. However, computational performance is not an issue. The ratio *path_computation_time / robot_travelling_time* is so small that we should worry more about software's design homogeneity than speed and space complexity.[5] Will we implement n distinct algorithms to solve our n optimization problems or will we implement a single one (i.e. the simplex algorithm in a base class using an Object Oriented Programming Language), and utilize inheritance & polymorphism to solve all our problems? If we decide to go with the simplex algorithm, then there is no need to reinvent the wheel; there are efficient implementations of the simplex algorithm (e.g. Matlab).

---

1 $O(mn \log n)$

2 Problems for which a solution can be checked in polynomial time are said to belong to class NP. NP-complete problems have the property that if any of these problems can be solved by a polynomial worst-case time algorithm, then all can be solved by polynomial worst-case time algorithms. It is accepted, but not proven that no NP-complete problems can be solved in polynomial time [5].

3 A subtour is a round trip that does not include all the nodes that are to be visited

4 Where the coefficient matrix **A** (constraints matrix) is m x n

5 Memory is cheap and computers are fast

3. SENSITIVITY ANALYSIS AND DUALITY

Two of the most important topics in Linear Programming are Sensitivity Analysis and Duality. Unfortunately, we will not discuss them much here, except to mention that the use of sensitivity analysis will speed up the computation as follows. In order to initiate the simplex algorithm for the first trivial shortest path problem posed, we require the introduction of artificial variables. As the simplex algorithm proceeds, these variables will eventually leave the basis. When they do, they can be eliminated, and this has the effect of reducing the size of the matrix involved in the computation. As we know, the simplex method can be visualized in tableau format, with the final solution referred to as the *final tableau*. The final tableau expresses the solution for the trivial shortest path. The idea is to restart from this final tableau to find the next trivial shortest path. In order to do that, we must alter the final tableau's right-hand-side to express a new starting source (which is the previous destination) and a new destination (see formulation B, section 12). Because this action makes the current tableau infeasible (negative values on the right-hand-side), we use the dual simplex method to regain feasibility. When feasibility is regained, we apply the simplex algorithm, this time with possibly less artificial variables than we had previously. Thus, using sensitivity analysis we can start from the basis obtained from the solution of the previous path without introducing artificial variables. The speed at which the simplex method finds the optimal solution depends polynomially on the number of variables in the problem, so the fewer variables needed the faster will be the computation of the solution.

4. IP MATHEMATICAL MODEL

Let us now take a look at the IP mathematical model that we will be using to optimize our tour. I have opted for the MCNFP (Minimum Cost Network Flow Problems) model.

Below, I will define the model, and I will describe the symbols in relation to our specific AMR shortest path problem, which is derived from Figure 1. I will build a mathematical model to solve a trivial shortest path problem, and I will briefly explain the responsibilities of the different sets of constraints. Finally, we will examine how edges can be avoided or forced. Later, we will solve the security run problem mentioned in section 1, using the knowledge gained from sections 5, 6, 7, 8, 9, 10 and 11.

5. MCNFP MODEL DEFINITION

Minimize:	$\sum_{\forall\ edges} c_{ij}\, x_{ij}$	*(objective function)*
		(constraints)
s.t	$x_{ij} - x_{ki} = b_i$	*(for each node i in the network)*
	$L_{ij} \le x_{ij} \le U_{ij}$	*(for each edge in the network)*

6. MCNFP SYMBOL DESCRIPTION

c_{ij} represents the cost of travelling the edge *ij*. For example, in Figure 1, travelling along the edge `x1,2` costs 2 minutes, while travelling along the edge `x2,3` costs 7 minutes.

x_{ij} represents the number of units sent through edge *ij*. Here, we have one robot, and therefore a single unit. Hence, x is either 1 or 0. Simply put, either the robot takes this edge (x = 1) or does not (x = 0).

b_i represents the net supply. Again, there is only one robot. If *i* is a source (starting node), it has value 1 at node *i*. If *i* is a destination, it has value -1 at node *i*. If node *i* is neither a source nor a destination, then it must be 0, because it is a transitory node and what enters must leave.

L_{ij} represents the lower bound on the flow through edge *ij,* and U_{ij} represents the upper bound. Here, the lower bound is zero, and the upper bound is 1. Not more than one robot can travel on edge *ij,* and not less than zero.

7. TRIVIAL SHORTEST PATH PROBLEM

Now, let us consider a trivial shortest path problem and build an appropriate IP formulation for this problem. Imagine our robot needs to find the shortest path between its actual position (say node 1), and the destination (say node 10 - refer to Figure 1). The following mathematical model in Formulation A expresses this scenario:

(Objective function)

```
MINIMIZE Z =

2 X1,2    +  2 X2,1    +  3 X1,5    +  3 X5,1    +  7 X2,3    +  7 X3,2    +  4 X2,6    +
4 X6,2    +  1 X3,4 +  +  1 X4,3    +  3 X3,7    +  3 X7,3    +  4 X4,8    +  4 X8,4    +
1 X5,6    +  1 X6,5    +  7 X5,9    +  7 X9,5    +  5 X6,7    +  5 X7,6    +  2 X6,10   +
2 X10,6   +  9 X7,8    +  9 X8,7    +  3 X7,11   +  3 X11,7   +  7 X8,12   +  7 X12,8   +
9 X9,10   +  9 X10,9   +  5 X9,13   +  5 X13,9   +  4 X10,11  +  4 X11,10  +  6 X10,14  +
6 X14,10  +  8 X11,12  +  8 X12,11  +  6 X11,15  +  6 X15,11  +  5 X12,16  +  5 X16,12  +
1 X13,14  +  1 X14,13  +  4 X14,15  +  4 X15,14  +  2 X15,16  +  2 X16,15
```

SUBJECT TO

(Subject to the following constraints)

```
 1)    X1,2   - X2,1   + X1,5   - X5,1                                          = 1
 2)  - X1,2   + X2,1   + X2,3   - X3,2   + X2,6   - X6,2                        = 0
 3)  - X2,3   + X3,2   + X3,4   - X4,3   + X3,7   - X7,3                        = 0
 4)  - X3,4   + X4,3   + X4,8   - X8,4                                          = 0
 5)  - X1,5   + X5,1   + X5,6   - X6,5   + X5,9   - X9,5                        = 0
 6)  - X2,6   + X6,2   - X5,6   + X6,5   + X6,7   - X7,6   + X6,10 - X10,6      = 0
 7)  - X3,7   + X7,3   - X6,7   + X7,6   + X7,8   - X8,7   + X7,11 - X11,7      = 0
 8)  - X4,8   + X8,4   - X7,8   + X8,7   + X8,12  - X12,8                       = 0
 9)  - X5,9   + X9,5   + X9,10  - X10,9  + X9,13  - X13,9                       = 0
10)  - X6,10  + X10,6  - X9,10  + X10,9  + X10,11 - X11,10 + X10,14 - X14,10    = -1
11)  - X7,11  + X11,7  - X10,11 + X11,10 + X11,12 - X12,11 + X11,15 - X15,11    = 0
12)  - X8,12  + X8,12  - X11,12 + X12,11 + X12,16 - X16,12                      = 0
13)  - X9,13  + X13,9  + X13,14 - X14,13                                        = 0
14)  - X10,14 + X14,10 - X13,14 + X14,13 + X14,15 - X15,14                      = 0
15)  - X11,15 + X15,11 - X14,15 + X15,14 + X15,16 - X16,15                      = 0
16)  - X12,16 + X16,12 - X15,16 + X16,15                                        = 0
17)  ∀ Xij = {0, 1}
```

FORMULATION A (For trivial shortest Path, source node1, destination node 10)

8. RESPONSIBILITES OF CONSTRAINTS

The objective function ($c \bullet x$) represents the sum of the costs of all possible paths in the network. The robot wants to minimize this function, therefore the costly edges such as $x_{7,8}$ and $x_{9,10}$ must be avoided as much as possible. The best way to do so is to set these variables equal to zero.

The constraints 1 to 16 ($x_{ij} - x_{ki} = b_i$) express each node's function: source, destination or transitory nodes. Every node is defined by its in-points (set negative) and out-points (set positive). Here, the robot is interested in travelling from node 1 to node 10. The starting node and the destination node must be unique. These are expressed on the right hand side of the constraints. Thus, b_1 (the starting node) must be set to 1, b_{10} (the destination node) set to –1, and all the other b_i's must be zero, since they are transitory nodes. Most importantly, constraints 1 to 16 guarantee that the returned source/destination optimized path will be connected, as it should be for the solution to represent an actual path traversed by the robot.

The constraint 17 ($L_{ij} \leq x_{ij} \leq U_{ij}$) limits the robot from travelling the same edge more than once and less than zero. With trivial shortest path problems, each edge can only be used once. The robot either takes the path (1) or does not (0). Results such as ½ would not have any meaning.

9. MECHANICS OF THE SIMPLEX ALGORITHM

Excellent textbooks exist that cover the simplex algorithm in detail [8,14]; as a result, the mechanics will not be discussed here. LINDO, MATLAB and other software can be used to solve Formulation A. In reality, a problem of this size would be tedious to compute by hand. In order to perform the simplex algorithm, the creation of more than 80 new variables, in addition to the 48 in Formulation A, are required.

10. AVOIDING AN EDGE OR A NODE

We will now discuss two different mathematical procedures to address edge avoidance. As stated earlier, a corridor could be blocked. The first method increases the cost of the edge(s) to avoid. Remember that we are solving for an optimal minimum. For instance (Figure 1), to avoid edge `X3,4` we could increase its cost so much that the algorithm will do everything it can to steer clear from it. In LP, this value is referred as big M. M symbolizes a very large value. Here M could be the sum of all the costs, 216 = 2 * (2 + 7 + 1 + 3 + 4 + 3 + 4 + 1 + 5 + 9 + 7 + 2 + 3 + 7 + 9 + 4 + 8 + 5 + 6 + 6 + 5 + 1 + 4 + 2). Thus, simply replacing the cost of 1 by 216 will force the algorithm to propose a path without `X3,4` in it.

The second method introduces new constraints to eliminate paths, or nodes. As mentioned above, a node is defined by its in-points and out-points. Cutting a node's in-points should make it "disappear" from the network. Suppose that node 3 must be avoided. This can be attained by appending the two binding constraints `X2,3` = 0 and `X7,3` = 0 to the initial formulation's set of constraints (see Figure 2). This will force the simplex algorithm to find a new shortest path that does not include the edges `X2,3` or `X7,3` in the solution. Hence, node 3 will never be visited.

Keep in mind that adding constraints is more costly in computational power than changing the costs. We remember from our discussion above that the average empirical complexity of the Simplex Method is $O(m^2n)$. Adding a new constraint increases *m* by 1. Changing the cost leaves *m* (and *n*) unchanged.

11. FORCING A NODE

Since each node is defined by its in-points and out-points, it should be simple to write constraints making sure that the robot goes "in" node *n*, then "out" node *n*, at least once in the network. This should force a visit, and it will. However, it will not necessarily guarantee a fully connected solution; subtours may appear. Consider this scenario, where node 1 is the source, node 11 is the destination, and node 3 is a node that the robot must visit in between (Refer to Figure 2). Appending the following two constraints to Formulation A will secure a visit to node 3.

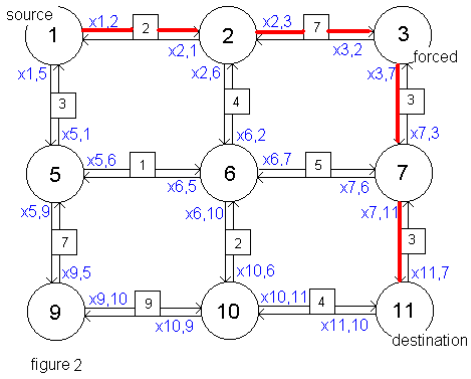


figure 2

```
3a) X2,3  +  X7,3      >= 1
3b) X3,2  +  X3,7      >= 1
```

3a) is saying: *the robot must either borrow edge `X2,3` or `X7,3` at least once, not less. The robot must absolutely "get in" node 3 by one of these two edges.*

3b) is saying: *the robot must either borrow edge `X3,2` or `X3,7` at least once, not less. The robot must absolutely "get out" node 3 by one of these two edges.*

With these additional constraints, the optimal path has length z = 15 minutes. We can follow the path returned by the simplex algorithm in Figure 2 (red arrows). We notice that the cost is **2** `X1,2` + **7** `X2,3` + **3** `X3,7` + **3** `X7,11` = **15** where the edge variables `X1,2 = X2,3 = X3,7 = X7,11 = 1` and all the other variables equal 0. We also notice the absence of subtours, which is desirable.

Now, consider Figure 3. Let us alter edges `X1,2` and `X2,1` by increasing their costs by 1; all the rest of the formulation stays the same as in Figure 2. Note that because of this, the algorithm returned two subtours, namely **3** `X3,7` + **3** `X7,3` and **3** `X1,5` + **1** `X5,6` + **2** `X6,10` + **4** `X10,11`. Since all these variables equal 1, adding the cost of the two subtours together gives an answer of z = 16 minutes. The reader should verify that if the robot had kept the previous path taken in Figure 2 (with the new altered cost for edges **3** `X1,2` and **3** `X2,1`), the final cost would have been 16 as well; **3** `X1,2` + **7** `X2,3` + **3** `X3,7` + **3** `X7,11` = **16**. Why, then, did the algorithm choose a subtour?

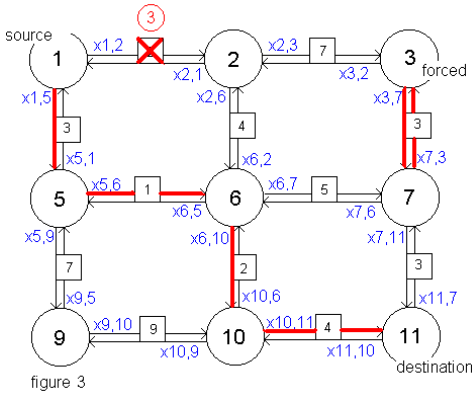


figure 3

This example shows that with this approach the algorithm is working exclusively to minimize z, while guaranteeing only that the path begins at node 1 and ends at node 10. Constraints 3a) and 3b) force the algorithm to find a path while including two edges[6] in the solution, but these constraints are not sufficient to formulate the problem that we want to solve here. Simply put, these constraints do not specify that the solution be fully connected. Hence, if subtours occur, the algorithm will not care as long as z is at the lowest cost possible. Although this approach is ineffectual here, we will see later that simple constraints like 3a) and 3b) will help us to solve the last round of our security run problem in Section 13. Also, it is worth

[6] Here, two edges incident to node 3 where indegree of node 3 = 1, and outdegree of node 3 =1

mentioning that there exist a formulation called the **Miller-Tucker-Zemlin (MTZ)** formulation [15], which adds J additional variables, J is the total number of nodes, and K additional constraints, K is the total number of edges, to the original formulation. This clever idea prevents TSP's subtours, however we will see in section 14 that this formulation does not return an optimal solution for our particular problem. So, it seems that adding constraints to force transitory nodes is not so simple. The problem gets trickier as we force more and more transitory nodes, and we recall that this security run problem involves three of them. Our MCNFP model, as it stands, cannot solve such complexity in one run. In order to solve this problem at once, we would possibly have to re-think the upper bound, the source, the destination constraints, and so on, which is not an easy task. Nevertheless, nothing stops us from using the same algorithm repeatedly. Besides, this is what we would have done if we had used Dijkstra's shortest path algorithm.

12. SOLVING A COMPLEX SHORTEST PATH PROBLEM

We are now ready to attack the security run problem introduced in section 1. As we recall, referring to Figure 1, we want to start from node 1 and finish at node 7, while visiting nodes 8, 11 and 13 in any order, all at the cheapest cost possible. Thus, a shortest path must be introduced between each of the five nodes to be visited in the sequence of n – 1 shortest paths[7]. Moreover, we need to consider all the possible sequences. We know that the sequences will always start at node 1 and finish at node 7. This narrows down the problem. For example, a possible solution could be Sequence 1, 11, 8, 13, 7. Another could be 1, 8, 11, 13, 7, and so on. A total of twelve distinct edges exist, but some are redundant to compute; for instance, node 8 to node 13 is equivalent in cost to node 13 to node 8. Thus, to work out this problem correctly, we will only need to run the simplex algorithm 9 times and find the trivial shortest paths for the paths 1→8, 1→11, 1→13, 8→7, 8↔11, 8↔13, 11↔13, 11→7 and 13→7. Figure 4 helps us see a clear picture of the problem. The question marks in Figure 4 will be replaced by the time (z) returned by the algorithm for each trivial shortest path computed. In other words, Figure 4 and Figure 5 are a high-level representation of Figure 1; they abstract the paths to be taken between the nodes, and only express the node-to-node optimal costs. We see that Tableau 1 is the matrix representation of Figure 5.

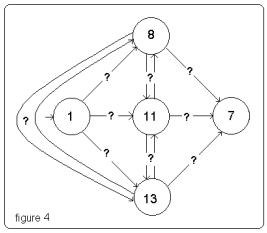


figure 4

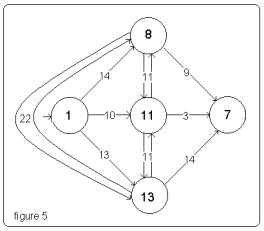


figure 5

Tableau 1	node 8	node 11	node 13	node 7
node 1	14	10	13	-
node 8	-	11	22	9
node 11	11	-	11	3
node 13	22	11	-	14

Also, we know that the simplex algorithm returns more than just the best cost; it also returns the connected source/destination path associated with that cost. Since we will need these paths later, we should store them in memory. If we were using C++, we could define the class **Path** to hold a shortest path and its corresponding cost. The instance variable **Path**::shortestPath could be an array of **EdgeVariable** objects, and the instance variable **Path**::z could be a primitive double. These **Path** objects could be organized in a **Course** class, having for data member a two-dimensional array of **Path** instances. This matrix could be similar to the ones shown in Tableau 1 and Tableau 2.

Using a formulation almost identical to Formulation A, we can now compute the shortest path between node 1 and node 8 (using Figure 1). We notice that the only difference between Formulation A and Formulation B below is the new destination (node 8 instead of node 10). I have highlighted the two adjustments. Note that nothing else has changed.

```
 1)    X1,2   - X2,1   + X1,5   - X5,1                                                    = 1
 2)  - X1,2   + X2,1   + X2,3   - X3,2   + X2,6   - X6,2                                 = 0
 3)  - X2,3   + X3,2   + X3,4   - X4,3   + X3,7   - X7,3                                 = 0
 4)  - X3,4   + X4,3   + X4,8   - X8,4                                                   = 0
 5)  - X1,5   + X5,1   + X5,6   - X6,5   + X5,9   - X9,5                                 = 0
 6)  - X2,6   + X6,2   - X5,6   + X6,5   + X6,7   - X7,6   + X6,10 - X10,6               = 0
 7)  - X3,7   + X7,3   - X6,7   + X7,6   + X7,8   - X8,7   + X7,11 - X11,7               = 0
 8)  - X4,8   + X8,4   - X7,8   + X8,7   + X8,12  - X12,8                                = -1
 9)  - X5,9   + X9,5   + X9,10  - X10,9  + X9,13  - X13,9                                = 0
10)  - X6,10  + X10,6  - X9,10  + X10,9  + X10,11 - X11,10 + X10,14 - X14,10             = 0
11)  - X7,11  + X11,7  - X10,11 + X11,10 + X11,12 - X12,11 + X11,15 - X15,11             = 0
12)  - X8,12  + X8,12  - X11,12 + X12,11 + X12,16 - X16,12                               = 0
13)  - X9,13  + X13,9  + X13,14 - X14,13                                                 = 0
14)  - X10,14 + X14,10 - X13,14 + X14,13 + X14,15 - X15,14                               = 0
15)  - X11,15 + X15,11 - X14,15 + X15,14 + X15,16 - X16,15                               = 0
16)  - X12,16 + X16,12 - X15,16 + X16,15                                                 = 0
```

FORMULATION B (Same as formulation A, but with a new destination, node 8)

[7] n is the number of nodes to visit including the source and destination nodes.

As we compute a shortest path, we instantiate a new **Path** object to store the results. Twelve **Path** objects will be instantiated, but we will only run the algorithm 9 times with the 9 distinct sources-destinations shown in Figure 5. Conveniently, only slight modifications to Formulation B will be carried out between each run. Below, Tableau 2 holds all the paths returned by the algorithm after completion. We verify that the path between node 8 and node 7 is $X_{8,4} \rightarrow X_{4,3} \rightarrow X_{3,7}$. Also, the cost for this path is 9 minutes (See Tableau 1).

Tableau 2

	node 8	node 11	node 13	node 7
node 1	X1,2→X2,3→X3,4→X4,8	X1,5→X5,6→X6,19→X10,11	X1,5→X5,6→X6,10→X10,14→X14,13	-
node 8	-	X8,4→X4,3→X3,7→X7,11	X8,4→X4,3→X3,7→X7,11→X11,15→X15,14→X14,13	X8,4→X4,3→X3,7
node 11	X11,7→X7,3→X3,4→X4,8	-	X11,15→X15,14→X14,13	X11,7
node 13	X13,14→X14,15→X15,11→X11,7→X7,3→X3,4→X4,8	X13,14→X14,15→X15,11	-	X13,14→X14,15→X15,11→X11,7

Now that we have Tableau 2 stored in memory, we are one step away from being finished. If we study Figure 5, we see that the abstraction has eliminated the complexity that we had in Figure 1. In Figure 5, all the nodes must be visited once. We recognize that this characteristic will allow us to solve Figure 5 as an authentic TSP problem. We will use a very similar formulation to Formulation A and B in order to solve for the final answer.

13. BUILDING TSP FOR FIGURE 5

To begin, we use the data stored in Tableau 1 to build the objective function. Each cost expressed in Tableau 1 becomes the coefficient of its respective variable. For example, the intersection of node 1 and node 8 is 14; this is translated as $14\ X_{1,8}$. Then, we build the constraints for each node. To do so, we must represent in-points with negative signs and out-points with positive signs. We also must set the source node's right hand side to 1, the destination node's right hand side to –1 and the others to 0. Then, we force every inner node (8, 11 and 13) to be visited exactly once. Although the forcing method did not work too well earlier, it will work perfectly here. The reason is that here, every node must be visited without exception; therefore the answer will be connected (with no subtours). Finally, 17) and 18) remain the same. Formulation C is shown in grey for clarity.

```
MINIMIZE
   14 X1,8   + 10 X1,11 + 13 X1,13 + 11 X8,11  + 22 X8,13 + 9 X8,7 + 11 X11,8 +
   11 X11,13 + 3 X11,7  + 22 X13,8 + 11 X13,11 + 14 X13,7
SUBJECT TO
   1)   X1,13  + X1,11  + X1,8                                  =  1   !start at node 1
   8)   X8,13  - X13,8  + X8,11  - X11,8  - X1,8  + X8,7        =  0   !pass thru node 8
   11)  X11,8  - X8,11  + X11,13 - X13,11 - X1,11 + X11,7       =  0   !pass thru node 11
   13)  X13,11 - X11,13 + X13,8  - X8,13  - X1,13 + X13,7       =  0   !pass thru node 13
   7)   -X8,7  - X11,7  - X13,7                                 = -1   !end at node 7

   20)  X11,8  + X11,13 + X11,7                                 =  1   !forcing node 11
   21)  X1,11  + X8,11  + X13,11                                =  1
   22)  X13,8  + X13,7  + X13,11                                =  1   !forcing node 13
   23)  X8,13  + X11,13 + X1,13                                 =  1
   24)  X8,11  + X8,7   + X8,13                                 =  1   !forcing node 8
   25)  X1,8   + X11,8  + X13,8                                 =  1

! upper bound 1, lower bound 0.
   17)  ∀ Xij = { 0, 1 }
   18)  ∀ Xij ≤ 1
```

FORMULATION C (TSP problem)

After computing Formulation C, we obtain $z = 43$. This represents the time in minutes that it would take the robot to complete its security run in a best-case scenario. It also returns the optimize path: $X_{1,13} = X_{13,11} = X_{11,8} = X_{8,7} = 1$ while all the other edge variables equal zero. This means that the robot will start at node 1, go to node 13, then node 11, and finish at node 7. We now have the abstract representation of the final answer; for the details, we will translate this sequence in a format that is compatible to our initial problem in Figure 1. To do so, we use the information that we have previously stored in memory, and simply do the mapping. For example, using Tableau 2, we can expand the variable $X_{1,13}$ with the data found at the intersection of node 1 and node 13. This gives $X_{1,13} = X_{1,5} \rightarrow X_{5,6} \rightarrow X_{6,10} \rightarrow X_{10,14} \rightarrow X_{14,13}$. The translations are expressed in Tableau 3.

Tableau 3

High Level	Figure 1 Level
X1,13	X1,5→X5,6→X6,10→X10,14→X14,13
X13,11	X13,14→X14,15→X15,11
X11,8	X11,7→X7,3→X3,4→X4,8
X8,7	X8,4→X4,3→X3,7→X7,11

We then concatenate the expansions together (top to bottom), and obtain the fully connected tour.

X1,5→X5,6→X6,10→X10,14→X14,13 ⇒ X13,14→X14,15→X15,11 ⇒ X11,7→X7,3→X3,4→X4,8 ⇒ X8,4→X4,3→X3,7→X7,11

The red arrows trace the tour in Figure 6. Remark that we are passing by node 7, which is our destination, but we cannot stop right away, since node 8 still has to be visited. After returning from node 8, we come back and stop at node 7. Verify the answer.

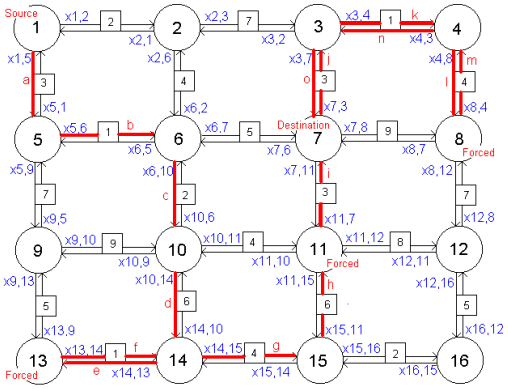
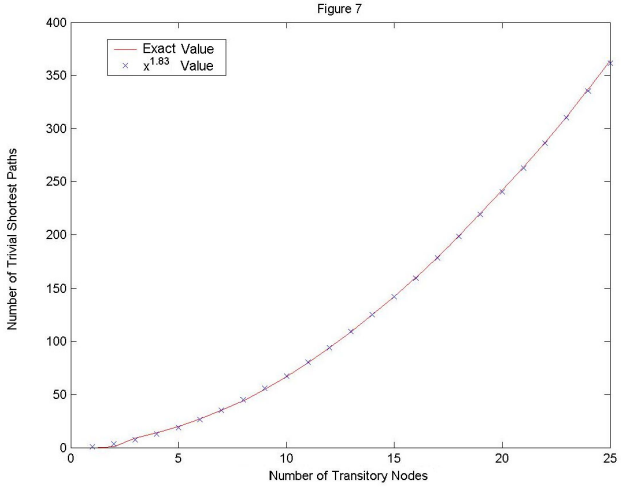


figure 6

14. DISCUSSION AND CONCLUSION

We saw that the simplex algorithm successfully solved our problem, but we ran the algorithm 9 + 1 times. This solution involves overheads, roughly 9 + 1 + 1 additional function calls. The first 9 overhead calls build the formulations and feed the simplex algorithm for each trivial shortest path computation. The tenth call builds the TSP formulation. The eleventh call translates the final answer by concatenating the shortest paths according to the sequence returned by the TSP. These steps may be broken into further parts. Fortunately, Formulation B has a similar mathematical structure for each trivial shortest path computation. If sensitivity analysis is not used, so that we must compute from scratch each trivial shortest path, then at most 4 right hand side values[8] need to be modified between each run. If used, sensitivity analysis[9] requires more groundwork between each run, but the algorithm will return the solution faster since there are fewer variables (i.e., no artificial variables)[10]. Finally, the tenth call builds a composite formulation problem.

Figure 7



Although Formulation C will always be built dynamically[11], the size of the network is significantly reduced in comparison to the original problem (Figure 4 versus Figure 1). Luckily, that fact works for us, and the impact of this call is acceptable. Finally, the eleventh call does not have much significance, as it runs in linear time and the size of the problem is minuscule.

Nevertheless, these overheads routines will cost extra computational time. For example, as the number of mandatory transitory nodes increase (let this number be x, $0 \le x \le 25$), the amount of trivial shortest paths to compute roughly approximates the curve $x^{1.83}$. Figure 7 plots two curves; the exact curve interpolated in red, and the approximate curve expressed by a discrete set of blue x's. We only need to consider a maximum of 25 nodes, since it is unlikely that a robot will have more than 25 transitory nodes to visit in a single mission during the scenarios that we are considering.

Let us evaluate roughly what a mission with 25 mandatory transitory nodes implies in computational cost. $25^{1.83}$ is approximately 362[12]. So, about 362 shortest paths, 1 TSP (1 source node + 25 transitory nodes + 1 destination node) and (~362 + 1 + 1) function calls (overheads), must be computed to obtain the final answer. Even if we know that the ratio computation_time/travelling_time will

8 Formulation A,B: the b_i in constraints 1-16

9 Or sensitivity analysis revisited. The simplex method revisited is a systematic procedure for implementing the steps of the simplex method in a smaller array, thus saving storage. This method is especially useful when n is significantly larger than m. More about this:[8].

10 One may ask whether the "double" application of the simplex algorithm on the 'smaller' problem (the dual method followed by the simplex method) is quicker than the "single" application of the simplex algorithm on the 'larger' problem (with artificial variables).

11 As opposed to Formulation A or B that can be built statically

12 Note that the accurate value is 364, 362 is the $x^{1.83}$ approximation

be very small, this estimate is impressive. That is why, when I started this project, my initial goal was to find a model that could express a security run mission in a unique formulation. I wanted to express the forced nodes, the broken edges and the source/destination nodes within a single problem, and then run the simplex algorithm only once in order to solve it. This would have saved overheads. I realized after putting forth the effort that such a model was very difficult to design. It is true that the MTZ formulation offers a run-once solution to traditional TSP without subtour, but it also prevents **connected subtours** to arise. If you look at figure 6, you see that the solution presents two connected subtours. I'm referring to the visit: node 14 to node 13, and node 13 to node 14 as the first connected subtour, and the visit: node 7 to node 3 to node 4 to node 8, and node 8 to node 4 to node 3 to node 7 as the second connected subtour. Because the method presented in this article allows these connected subtours to exist, we receive for final answer z equals 43 minutes. If we had considered the MTZ formulation, we would observe that the final solution presents no subtours (which is what we want) but also that it has no connected subtours (which we may need), and because of this, z would equal 57 minutes, this is 14 minutes more. I have also questioned myself if neural networks could not be another possible method to consider, but then what happens when an edge becomes impractical? What kind of training set is required to guaranty the same type of results we obtained here?

Finally, we discussed in section 2 that software's design homogeneity was a motivation to implement the simplex algorithm. We must foresee that our robot will certainly have additional problems in optimizing something other than shortest path planning. Since the development of the simplex algorithm, many people have contributed to the growth of LP by developing its mathematical theory, devising efficient computational methods and codes, and exploring new algorithms and new applications [8]. Nowadays, optimization is a concern in every field, and LP offers solutions for a wide range of problems. We spend our life trying to optimize what surrounds us: our incomes, our quality of life, minimizing our energy while maximizing our gains, etc. Consequently, an "intelligent" AMR should not spin a wheel or wink a sensor without first optimizing its action!

15. ACKNOWLEDGEMENT

I would like to thank my supervisor, Dr Randall Pyke, for his support and generosity. His dedication towards his work and students is an inspiration for my pursuit of a teaching career.

16. BIBLIOGRAPHY

[1] Alex Ferworn, "Class notes CPS607 Autonomous Mobile Robotic", Ryerson University, 2003.
[2] B.Muller, J Reinhardt, "Neural Networks", An introduction, Springer-Verlag Berlin Heidelberg, 1990.
[3] Harvey J.Miller and Shih-Lung Shaw, "Geographic Information Systems for Transportation", Principles and Application, Oxford University Press. 2001
[4] Howard Anton, "Elementary Linear Algebra", (7th ed.), John Wiley & Sons, Inc.,1994.
[5] Kenneth H.Rosen, "Discrete Mathematics and Its Application", McGraw Hill, (5th ed.), 2003
[6] Marilyn McCord Nelson, W.T. Illingworth, "A practical Guide to Neural Nets", Addison-Wesley Publishing Company, 1994.
[7] Masao Iri, "Network Flow Transportation and Scheduling", Theory and Algorithms, Academic Press, 1969.
[8] Mokhtar S Bazaraa, John J.Jarvis and Hanif D.Sherali,"Linear Programming and Network Flows", (2nd ed.), Wiley, John and Sons, Incorporated, January first, 1990.
[9] Randall Pyke, "Class notes MTH503 Operation Research", Ryerson University, 2003.
[10] Ravindra K. Ahuja, Thomas L.Magnanti and James B. Orlin, "Network Flows", Theory, Algorithm and Application, Prentice Hall, 1993.
[11] Robert S Garfinkel and George L.Nemhauser, "Integer Programming", John Wiley & Sons, Inc., 1972.
[12] Robin R.Murphy, "Introduction to AI Robotics", Massachusetts Institute of Technology, 2000.
[13] Thomas A. Standish, "Data Structure, Algorithms & Software Principles in C", Addison-Wesley Publishing Company, 1995.
[14] Wayne L. Winston, "Operations Research", Application and Algorithms, ITP,(2nd ed.), 1994.
[15] Gàbor Pataki, Teaching Integer Programming Formulation Using the Traveling Salesman Problem, Society for industrial and Applied Mathematics, 2003