Snow-Plow Route Planning using AI Search

T.M. Rao, Sandeep Mitra
Department of Computer Science
The College at Brockport, State University of New York
Brockport, NY, USA
{trao, smitra}@brockport.edu

James Zollweg
Department of Earth Sciences
The College at Brockport, State University of New York
Brockport, NY, USA
jzollweg@brockport.edu

Abstract— **Snow plowing comprises a major portion of the total municipal budget in many communities. In these times of severe budget stress, it is vital to find ways to perform this essential service in an efficient manner. Optimizing the routes travelled by plows is one way to reduce costs. This problem, however, can be shown to be an NP-Hard problem at its core, with several additional complications. In this paper, we demonstrate how this problem can be formulated as a state-space search problem, and how one can employ AI techniques such as A-Star (A*) search to compute optimal or close-to-optimal routes. Specifically, we have developed A*-SnowPlowRouter (A*SPR), a Java software application that uses a "roads data set" of a town and generates efficient route plans. This data set is generally obtained from the municipal Geographical Information System (GIS). Our procedure transforms this data into a directed graph representation, augmented with road priority data. We then employ the A* technique to generate routes which minimize travel distance, avoid U-Turns, and reach higher-priority roads before lower-priority roads. The route plans computed by A*SPR are then exported into a format that can be displayed on the GIS. We have experimented with road networks in several local municipalities, and obtained routes that are significantly better than the ones currently used.**

Keywords-Snowplow Route Planning, A* Algorithm.

I. INTRODUCTION

Snow plowing is an essential service provided by local municipalities in a significant portion of the USA. It is important to use the available manpower and equipment in the most cost-efficient manner possible. The goal of route development is to minimize the total distance covered by a plow (minimizing fuel consumption, driver time and equipment wear and tear), clear higher priority roads earlier, and minimize the number of time-consuming and accident-prone U-turns. The problem, in essence, is a difficult optimization problem in which we seek to minimize the total travel distance, the number of U-turns and the number of lower-priority road segments plowed before higher-priority segments (known as priority misplacement). Due to the complexity of the problem, most current routes used, especially in small communities, are almost always the result of years of "experiential evolution." The resulting routes are usually non-optimal and rarely up-graded, even as the community changes. Some attempts (RouteSmart [10], JOpt [11], ArcLogistics [12], etc.) have been made to develop software packages to assist in route planning. However, these tools generally have several short-comings, including high setup costs, steep learning curves, limited choices of optimization parameters, and the need to make a large investment in database preparation, which make them unappealing to many small public works departments. The goal of our work is to create an inexpensive, effective, and easy-to-use software package that will assist the planners in computing optimal or close-to optimal route plans. Our software accepts road network data from a Geographic Information System (most municipalities have this data in place) and produces a link-by-link route plan for the truck drivers to follow. The software allows the planners to experiment with various parameters such as priority values of the roads, and the penalties for incurring a u-turn, repeating a road segment, and misplacing priorities. With these features, the planners can create routes which optimize according to widely varied objectives. Results from our experiments so far are encouraging. For example, we generated route maps for the Town of Parma (Western New York State) with as much as a 12% reduction in travel distance with no increase in U-turns or plowing priority violations.

This paper is organized as follows: We begin by providing an overview of the snowplow routing problem and how the street network to be plowed is translated into a graph representation. We then present all of the definitions needed to proceed with our theoretical development of the solution methodology. At this time, we also make several key observations that aid in the methodology to follow. As a prelude to our own theoretical development, we discuss and compare our approaches with a number of other related problems and techniques. We then proceed with the development of network pre-processing, an important part of our solution. Following this, we proceed into the core of our solution, the A* search. We then present a brief discussion of the role of our application in the real-world and successes we have had in local municipalities. Finally, we provide some conclusions and plans for future work.

II. PROBLEM FORMULATION

The road network of a small town is represented by a directed planar graph. Each vertex of this graph represents an intersection and each edge a road segment. Edges may be bi-directional or unidirectional, representing two-way and one-way streets respectively. Associated with each edge are its length and priority values. We assume that the graph has no 'self-loops' – i.e., there are no edges that begin and end at the same vertex. A start vertex (corresponding to the 'depot') must be specified – routes begin and end at this vertex. With this representation, our problem reduces to the computation of a *complete, closed walk* in the graph. Besides beginning and ending at the start vertex, and covering every edge at least once, this walk should minimize the number of u-turns, repeated edges, and the number of times a lower priority edge is traversed before a higher priority one.

III. DEFINITIONS AND OBSERVATIONS

A *walk* W in a graph is a sequence of vertices W = $\{v_0, v_1, v_2, \dots, v_k\}$ where each (v_i, v_{i+1}) is an edge. A walk is *closed* if the first and the last vertices are the same – otherwise, it is *open*. A *complete walk* starts at a vertex, covers each edge at least once, and returns to the same vertex. An *Eulerian circuit* is a complete walk that has no repeated edges.

A *u-turn* in a walk is a subsequence $\{v_i, v_{i+1}, v_{i+2}\}$ where v_i and v_{i+2} are the same vertex. Edges are assigned numerical *priority values*, with a lower value indicating a higher priority. The *priority sequence* of a walk W is the sequence P = $\{p_0, p_1, p_2, \dots p_{k-1}\}$ where p_i is the priority value of the edge (v_i,v_{i+1}). A *priority misplacement* occurs in a priority sequence P when $p_i > p_j$ for some $j > i$. The 'severity' of this misplacement is the difference: $(p_i\text{-}p_j)$. For every edge, (v_i, v_{i+1}) with priority p_i, its *priority misplacement index* is the sum of severities of misplacements of edges in the partial walk $\{v_0, \dots, v_i\}$ that have a higher priority value (i.e., lower priority) than this edge. The *priority misplacement index* of a walk W is the sum of the priority misplacement indices of all edges in W. The values *u(W)*, *r(W), m(W)* and *d(W)* represent the number of u-turns, the number of repeats, the priority misplacement index and the total distance covered in a walk W. Observe that if *r(W)* is zero, then *d(W)* is minimized.

Our problem now can be stated as: *Given a directed graph representation as described above, compute a complete closed walk W that minimizes u(W), r(W) , and m(W).*

A graph is *strictly bi-directional* if for every edge (v_i, v_j), there also exists the edge (v_j, v_i). An example is seen in Fig. 1. *In a strictly bi-directional graph, a complete walk always exists*. This can easily be proved by observing that in a strictly bi-directional graph, the in-degree of every vertex is the same as its out-degree. Hence, the existence of an Eulerian circuit, which is a complete walk, is assured.

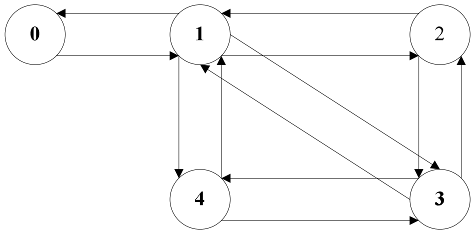

Figure 1 Strictly bi-directional Graph

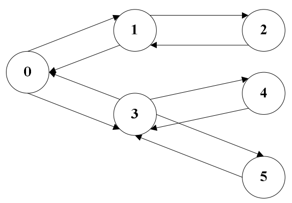

Figure 2 Strictly bi-directional Acyclic Graph

If a graph is strictly bi-directional and *acyclic* (i.e., it becomes a tree if each pair of directed edges is replaced by a single undirected edge), then there exists a complete walk with no repeated edges, with number of u-turns equaling *N* or *N*-1, where *N* is the number of leaf nodes in the graph.

A constructive proof of this can easily be given. This is illustrated in Fig. 2. Vertex 0 is the start node. It has 3 leaf nodes. The walk 0-1-3-1-0-2-4-2-5-2-0 is a complete walk with no repeated edges and 3 u-turns (i.e. 1-2-1, 3-4-3 and 3-5-3)

Finally, a *bi-connected* graph is a directed graph in which for any pair of distinct vertices $\{v_i, v_j\}$ there exist two distinct paths that connect v_i and v_j that have no vertices in common, other than v_i and v_j.

IV. RELATED WORK

Our problem is related to the Chinese Postman Problem (CPP) [3] defined as the "Problem of finding the least cost circuit on (a graph) G which traverses at least once each element (of the edge set) E." Let $E_1, E_2, \dots E_p$ be a partition of E into disjoint subsets, such that their union is the set E. Let '<' be a precedence relation between the edges that specifies if $E_i < E_j$ then every edge in E_i must be traversed for the first time before any edge in E_j. This relation is a 'linear precedence relation if $E_1 < E_2 < \dots < E_p$. Dror *et al* [3] consider the cases in which the relation < is a 'linear precedence relation', and have shown that a feasible solution exists if and only if all the edge sets E_1, $E_1 \cup E_2$, …$(E_1 \cup E_2 \cup \dots E_p)$ are connected. They also prove that the CPP is an NP-hard problem in general (i.e., a problem for which there is no known technique to compute a solution in polynomial time) when < is only a 'general precedence relation' and not a linear precedence relation. Our problem is different from CPP in the sense that we cannot assume the connectivity condition on edge sets exists and have the additional constraints of minimizing the numbers of u-turns, repeated edges and priority misplacements. Ghiani and Improta [5] study the problem in which a linear precedence relation exists – called the Hierarchical Chinese Postman Problem (HCPP) – and propose a lower-complexity algorithm for circuit computation. Kazemi *et al* [6] have studied the problem of computing a minimum cost traversal of a road network that visits the entire road network. This problem is called the Optimal Traversal Problem (OTP). They consider costs of edges as well as that of making a turn. U-turns cost the most (80), followed by left turns (60), and then right turns (10). They propose two approaches to solve this problem. The first approach is a 'greedy' algorithm that uses local information around each vertex to compute solutions that are not necessarily optimal. The second approach transforms the problem into an equivalent Asymmetric Travelling Seller's Problem (ATSP) and applies the 'Patched Cycle Cover Algorithm'. The authors report that this approach computes a near-optimal solution. Our problem differs from this approach in the sense that we also consider priorities assigned to road segments. Eiselt *et al* [4] present a general survey of arc routing problems. They consider several versions of the problem such as undirected CPP, directed CPP and mixed CPP. However, there is no mention of any software implementation or experimental results. Cabral et al [1] have proposed an algorithm to solve the HCPP by transforming it into a Rural Postman Problem (RPP). In a rural postman problem the edges are classified as 'required' or 'non-required', and only the required edges need be covered. These authors report an implementation of their algorithm and present experimental results on a real-world graph. Thimbleby [9] presents a small Java program to solve the directed CPP. While most papers consider the case with only one postman traversing the graph, Osterhues *et al* [8] consider the *k*-CPP problem, where there are *k* postmen who start and return to the base. The objective is not to minimize the total distance covered by all postmen, but to minimize the maximum distance covered by any postman. Campbell *et al* [2] study a more general problem of 'snow disposal site location and sector assignment' and apply it in the context of an urban snow removal problem for the City of Montreal.

The problem that we are trying to solve is different from those considered by these authors in the sense that we seek to optimize on three variables. Our flexible software application (A*SPR) uses an AI search algorithm to compute the solution. Our solution allows the route planners to control the quality of the solution by simply changing the parameters in a configuration file.

V. PRE-PROCESSING THE GRAPH

The time taken to compute the complete walks increases exponentially with the size of the graph. In order to manage this complexity it is advantageous to preprocess the graph and divide it into smaller components. We have taken the following approach:

- Identify all sub-graphs that are in the form of a (bi-directional) tree and record the vertex at which they are attached to the rest of the graph. These sub-graphs are then removed from the original graph.
- Decompose the remaining graph into bi-connected components which are joined by bi-directional paths of length >= 0.

Computing a complete walk in a bi-directional tree is trivial, as explained above (see Fig. 2). For each bi-connected component, a complete walk is computed using the approach described in Section VI. All these individual walks can then be 'sewn together' to form a complete walk W_c for the full graph using the following techniques:

- Let W_i and W_j be two complete walks computed for bi-connected components B_i and B_j respectively. Assume that B_i and B_j are connected by a bridge (u, v) where u is a node in B_i and v is a node in B_j. W_i and W_j can be combined into a complete walk W_k (initially empty) using the following steps:
    - Identify any location l in W_i where the node u is present.
    - Take the prefix of W_i up to location l. Append this prefix to W_k.
    - Append the edge (u, v) to W_k.
    - Reconfigure W_j in a manner such that v is the start (and end) node of the walk. Append this re-configured walk to W_k.
    - Append the edge (v, u) to W_k.
    - Obtain the suffix of walk W_i from the location l till the end node. Append this suffix to W_k.

  It can be proved that the number of u-turns in W_k, $u(W_k) <= u(W_i) + u(W_j)$. This process is repeated until all bi-connected components connected by bridges are included in the walk W_c.
- The complete walk computed for a tree sub-graph can be inserted at any of the locations where the root node of the tree is present in the walk W_c computed in the step above. A location that ensures that the priority misplacement count of the resulting walk is minimized must be chosen.

We assume that the input graph has been pre-processed and we are dealing with a single bi-connected component. Further, we also assume that the graph is a simple graph and not a multi-graph; implying that there are no multiple parallel edges between the same source and destination nodes. Thus, our current implementation does not support roads with multiple lanes. Extension of our software application A*SPR to handle multiple lanes is a part of our planned future work.

VI. A* SEARCH SOLUTION

A* search [7] proved to be a viable technique in tackling this problem. A* belongs to the family of state-space search techniques, requiring the modeling of a problem-specific state, defining a set of operators, start and goal states and devising cost and heuristic functions to guide the search. A state in this problem simply encapsulates a walk (partial or complete) represented as a sequence of vertex labels: v_0-v_1- ... v_k. We assume that the depot is located at vertex $v_0 = 0$. Thus, the *start state* has a walk that just includes one node v_0 in it. A state that encapsulates any complete walk is a goal state. Operators *go(v)*, where v is any vertex in the graph, are defined as follows: If there is an edge from v_k (the last vertex in the sequence) to v, then the operator *go(v)* is valid for this state, otherwise it is not. The state change that results from the application of a valid operator *go(v)* simply concatenates *'–v'* to the walk in the current state. Fig. 3 shows the first 3 levels of the search tree for the graph shown in Fig. 1.

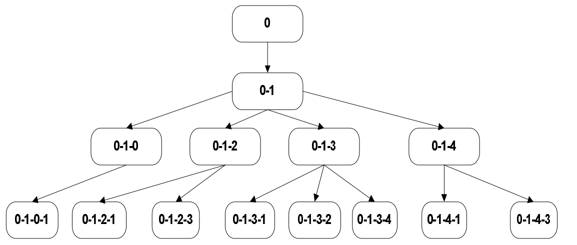

Figure 3 Build-up of a search tree (3 levels)

The potential enormity of such a search tree requires techniques like A* to guide its exploration by identifying the 'most promising' state at each step of the search. The efficacy of the search process is principally influenced by the techniques used to identify the promising states. The promise of a state *f(W)* is computed as the sum of two values: *g(W) + h(W)*, where *W* is the walk contained in the state. 'g' is the known cost of the walk *W*, and 'h' (the heuristic) is an estimate of the cost of the remaining walk that might complete this walk. A* chooses the state with the least value of *f(W)* as the 'most promising' state.

A. The Cost Function 'g'

The cost function 'g' represents the "goodness value" of a state, which is what we are trying to optimize. In this problem, since the goal is to optimize the three factors *r(W), u(W)* and *m(W)*, the cost function must include all of these. After a number of experiments, we have observed that the following function is the most effective in obtaining a good route:

$$g(W) = (dist(W) + u(W) * UP + r(W) * RP + m(W) * MP) / length(W) \quad (1)$$

where *UP*, *RP, and MP* are the user-selected penalties for u-turns, repeated edges and priority misplacements respectively; *dist(W)* is the total distance covered by the walk and *length(W)* is the number of edges in the walk. The presence of *length(W)* in the denominator encourages the search process to prefer longer walks over shorter ones.

We have experimented with a variety of definitions for the cost function. For example, the numerator alone in (1) above can be used as the cost function. However, our experience shows that with this function, the search takes many more iterations to obtain a reasonable solution than if (1) above was used. As indicated in Section VIII, our application, A* SPR, allows us to easily 'plug in' new cost and heuristic functions and test their efficacy.

B. The Heuristic Function 'h'

The heuristic function 'h' is expected to expedite the search towards the goal state. The following function has worked well in our experiments:

$$h(W) = \sum (dist(e) + priorityMisplacement(e)) \quad (2)$$

The summation is over all the edges *e* in the full graph *G* that are *not* in *W*. The notation *dist(e)* is the length of edge *e*, and, as described in Section III, *priorityMisplacement(e)* is calculated as the sum of the misplacement severities *(priority(e) – priority(x))* for each edge *x* in *W* such that *priority(x)* is numerically lower than *priority(e)* (Note: in our implementation, 1 is the highest priority).

C. Termination Condition

Our goal is to compute a complete walk that is optimal over all criteria. Continuing the search process until all complete walks are computed would enable the selection of the optimal walk. However, the exponential complexity of this process makes this impractical. Consequently, our search process terminates when it finds a complete walk. At this point the 'h' value is zero and the 'g' value is the minimum found so far. By varying the penalty parameters, we can produce walks that are better in one or the other criteria.

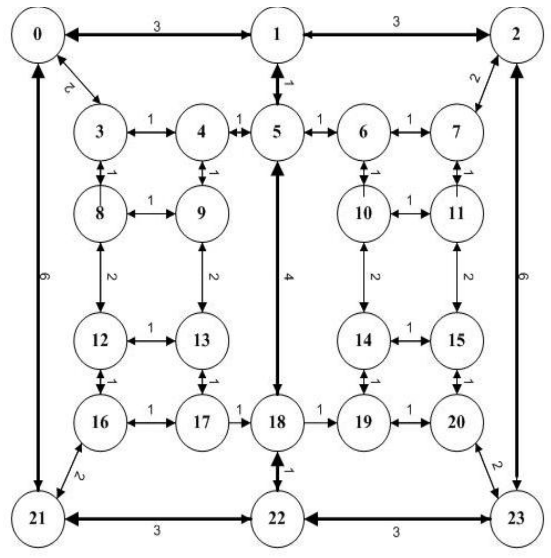
Small Town Graph (Thick lines: high priority, Numbers on line: length)

Figure 4 Small-Town Experimental Graph

Using A*SPR, we have experimented with a large number of graphs and penalty parameters. Our experience shows that small values will produce quick results but have large numbers of repeats, u-turns and priority misplacements. Larger penalties take longer processing times, but can produce better results. Results from a small-town experimental graph (Fig. 4) are shown in Table I.

For every edge (i, j) in the above graph, the length and priority of (i, j) is the same as that of its opposite edge (j, i). There are only two priority levels: high (thick lines) and low (thin lines). In Table I, *UP*, *RP* and *MP* are the parameters selected by the user, d(W) is the total distance covered by the walk, and u(W), r(W), m(W) and "eTime" are the number of u-turns, repeated segments, misplacement index, and the elapsed time in seconds in the solution found for that selection of parameters.

Our experiments were conducted on a PC running Windows XP, and with a 2.33 GHz Intel processor.

TABLE I. EXPERIMENTAL RESULTS ON SMALL-TOWN GRAPH

	UP	**RP**	**MP**	**d(W)**	**u(W)**	**r(W)**	**m(W)**	**eTime (secs)**
1	1	1	1	156	20	15	0	7
2	10	10	1	150	14	14	12	52
3	10	10	10	152	18	12	6	16
4	20	20	1	150	14	14	12	154
5	20	50	5	142	13	6	12	14
6	30	50	10	142	12	6	0	146
7	100	100	1	136	7	2	0	21
8	110	100	1	138	6	4	0	91
9	150	120	1	134	7	2	0	1239
10	200	100	1	134	7	2	0	1929
11	200	20	1	152	3	16	0	11
12	1	200	1	132	14	0	6	13

Space considerations limit us from providing all the solutions generated. However, we note that we can obtain solutions of different qualities by varying the parameters. Observe that the graph has 24 vertices and 74 edges. The minimum total distance covered by all the edges is 132. Keeping the parameters low produces quick solutions, but not of good quality. The quick solution produced in Row 7 of Table I had seven u-turns, only two repeated edges (9-13) and (13-9) each of length 2, and no priority misplacements. The solution obtained in line 10, took a longer time, but produced a better solution in the sense that the repeated edges were shorter. We show the solution in line 8 had only 6 u-turns and 4 repeated edges: [0-1-2-23-22-18-5-1-0-21-22-23-2-1-5-18-22-21-0-3-8-12-16-17-18-19-20-15-11-10-14-19-18-17-16-12-8-9-13-12-13-9-8-3-4-5-6-10-11-15-20-23-20-19-14-15-14-10-6-7-11-7-2-7-6-5-4-9-13-17-16-21-16-17-13-9-4-3-0].

The repeated edges are: 16-17, 17-16, 9-13 and 13-9 each of length 1. The solution on line 9: [0-1-2-23-22-18-5-1-0-21-22-23-2-1-5-18-22-21-0-3-8-12-16-17-18-19-20-15-11-10-14-19-18-17-16-12-8-9-4-5-6-10-11-15-20-23-20-19-14-15-14-10-6-7-11-7-2-7-6-5-4-3-4-9-13-12-13-17-16-21-16-17-13-9-8-3-0] has 7 u-turns, but only two repeats 16-17 and 17-16. We observe that by keeping UP very low, and RP very high, we can get the repeats down to zero (line 12).

It should be noted that the system computes routes assuming that there is only one plow. However, even small towns have five or six plows. An optimal allocation of the roads to plows is a different problem, and is not addressed by the A*SPR application. As such there is no upper bound on the number of vertices or edges in the input graph. A*SPR has successfully computed routes in graphs up to 25 vertices and 80 edges in reasonable time.

VII. REAL-WORLD APPLICATION

There are a number of commercial software packages that are designed to help in various route planning tasks. These include RouteSmart [10], JOpt [11], ArcLogistics [12], and many others. These packages, however, are used by a relatively small number of highway and public works departments. The lack of use of any computer solution to plow routing is especially notable in small municipalities. There are a number of reasons for the relatively low rates of adoption of current commercial software solutions by these towns and counties. These include the high cost of obtaining the software, steep learning curves, the need for extensive/complex data preparation that are usually beyond the abilities of local departments, the difficulty in adapting a general purpose routing software to the snowplow problem, and the inability to sufficiently customize the software for the specific need of each department. Our software application, A*SPR, has the potential to be a usable, effective, and inexpensive alternative for small towns.

We have used our software to compute optimal paths for the small Town of Parma in Western New York. The town currently has six plows and the route plans are manually computed – the result of several years of experience. The data for the town graph was obtained from a Geographical Information System and translated into a text file formatted to suit the input requirements of our software. Some amount of pre-processing of the graph was done manually: we identified *spurs* (a spur is a dead-end road) and removed them from the graph. Each spur creates a U-turn, which can negatively affect the progress of the software execution. As part of post-processing, A*SPR inserts the spurs back. We partitioned the rest of the graph into several bi-connected sub-graphs. We obtained either 4, 5 or 6 sub-graphs, depending on the number of plows to be used. At this point, these partitions were created manually for the experiment, based on some ad hoc criteria.

TABLE II. EXPERIMENTAL RESULTS ON REAL TOWN

Scenario	Distance (meters)	u(W)	r(W)
Current (6 rtes)	345153	45	2
A* (6 rtes)	341999	31	0
A* (5 rtes)	302978	18	0
A* (4 rtes)	293212	20	0

We computed solutions separately for each partition using A*SPR. The software application created paths that were better than the experience-driven manually-computed ones. For initial trials, all of the road segments were considered high-priority and hence priority misplacement was not an issue. We summarize the results in Table II. Observe that the total distance covered differs in various partitions even when there are no repeated edges. This is because of the roads that are shared between partitions and the need to travel to routes that do not start at the depot. The best solutions obtained produce a 12% improvement in travel distance with no decrease in quality of service.

VIII. THE FLEXIBLE ARCHITECTURE OF A*SPR

The A*SPR application has been architected in a manner that makes it highly configurable by the user. As stated earlier, the user can easily vary the penalties for repeats, u-turns and priority misplacements by editing a configuration file. A*SPR also allows the user to configure the number of expansions the search algorithm will run for. If the application does not find a complete walk within this limit, then it provides a *partial solution*.

This solution is essentially a walk of the smallest cost (g-value) among all the longest walks that the application has considered so far. The user can then use any prefix of this walk as a “head-start”. Normally, A*SPR is configured with the label of the start node. However, the user can take advantage of the partial solution by substituting the start node value with any prefix of the partial solution. A*SPR ensures that every walk it considers during the search always has this partial walk as a prefix. Using this approach, it has been possible to enable A*SPR to compute better quality solutions incrementally, with each step executing for a reasonable number of expansions.

The high configurability of A*SPR just does not end with the ability to provide data values. By making extensive internal use of the Java Reflection API, A*SPR provides the user with the ability to experiment with different cost and heuristic functions. While an end-user is unlikely to make use of these features, programmers may consider using A*SPR as a *framework* which can then be extended with different cost functions and heuristics they want to experiment with, in their quest to find the best A* search technique that can solve this problem. A*SPR, therefore, is an experimental “test bed” for any researchers who may be interested in using it to seamlessly “plug in” their own cost computation and heuristic functions for the snow plow routing problem. The flexibility that the A*SPR architecture provides has enabled us to readily consider other problems related to snow plow routing. For example, we are also working on similar software for snow clearing from sidewalks. This problem is very similar to the snow plow routing problem – the principal difference is that although all side-walks are two-way, they only need to plowed in one direction for the segment to be considered covered. We modify a few key behaviors in A*SPR. For example, we change the check for the termination condition to treat any edge (u, v) as covered when either (u. v) or (v, u) are present in the walk.

IX. CONCLUSIONS AND FUTURE WORK

Our aim in this project was to develop a software application that uses just the basic information about the roads of a town, including the length and priorities of the road segments. This information is easily extractable from standard GIS data. Our application, A*SPR, attempts to compute a suitable plowing route for each of the segments of a town that is to be covered by a single snow plow. It uses the A* search algorithm to accomplish this task.

We plan to use the A*SPR application and experiment with different graphs of towns and villages. We will also be extensively researching the sidewalk snow clearing problem.

REFERENCES

[1] Cabral C., Gendreau M., Ghiani, G., Laporte, G. *Solving the Hierarchical Chinese Postman Problem as a rural postman Problem,* European Journal of Operational Research, 155 (2004) 44-50

[2] Campbell, J., Langevin, A., *Operations Management for Urban Snow Removal and Disposal,* Transpn. Res. –A, Col. 29A, No. 5, 359-370, 1995.

[3] Dror, M., Stern, H., Trudeau, P., *Postman Tour on a Graph with Precedence Relation on Arcs,* Networks, Vol. 17, (1987) 283-294.

[4] Eiselt, H., Gendreau, M., Laporte, G., *Arc Routing Problems, Part I: The Chinese Postman Problem,* Operations Research, Vol. 43, No. 2, 1995, 231-242.

[5] Ghiani, G., Improta, G., *An Algorithm for the Hierarchical Chinese Postman Problem,* Operations Research Letters, Vol. 26, (2000) 27-32.

[6] Kazemi, L., Shahabi, C., Sharifzhdeh, M., Vincent, L., *Optimal Traversal Planning in Road Networks with Navigational Constraints,* Proc. 15th Annual Symp. On GIS, Seattle WA, 2007, pp 1-8

[7] Luger, G. F. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving.* 6th Edition, Addison-Wesley, 2008.

[8] Osterhues, A., Mariak, F., *On variants of the k-Chinese Postman Problem,* Operations Research und Wirtschaftinformatik, No. 30, 2005, http://www.wiso.tu-dortmund.de/wiso/or/Medienpool/publikationen/dispap30.pdf accessed 12-15-10.

[9] Thimbley, H. *The directed Chinese Postman Problem,* www.cs.swan.ac.uk/~csharold/cpp/SPAEcpp.pdf, obtained 12-15-10.

[10] RouteSmart, http://www.routesmart.com/, accessed 1-11-11

[11] JOpt.net-Vehicle Routing Software Library, http://www.dna-evolutions.com/joptnet.html, accessed 1-11-11

[12] ArcLogistics – Routing Software, http://www.esri.com/software/arclogistics/index.html, accessed 1-11-11