2012 IEEE International Conference on Systems, Man, and Cybernetics
October 14-17, 2012, COEX, Seoul, Korea

Route Allocation for Multiple Snowplows Using Genetic Algorithms

T.M. Rao, Sandeep Mitra, James Zollweg
Departments of Computer Science and Earth Sciences
The College at Brockport, State University of New York
Brockport, NY, USA
{trao, smitra, jzollweg}@brockport.edu

Abstract—**The road network of a small town is represented as a directed graph where each road junction is a vertex and each road segment (which has a length and a priority value) is a directed edge. We assume that there are several plows available to service the roads. We seek to compute an optimal allocation of routes to plows. Each route begins and ends at the same (depot) vertex. The union of all plow routes must cover every edge in the graph at least once. In addition, we wish to minimize the following parameters: total distance covered by all plows (thereby minimizing the deadhead miles), amount of variation in the mileage covered by individual plows (thereby dividing the workload equitably), number of u-turns and extent of priority misplacements. In this paper, we propose a Genetic Algorithms (GA)-based solution to compute a near-optimal route allocation and the minimization of other parameters simultaneously. This algorithm is based on our GA solution to the 1-plow problem reported earlier. We have developed a Java application that implements our algorithm. Our experiments with reasonably large graphs have yielded good solutions. These solutions are especially useful in snowplow routing for small towns, as plowing costs consume significant portions of the total municipal budgets of these communities. Most of the route planning in small towns is currently done manually and routes have evolved over time by experience. In these times of severe budget stress, route allocation using our approach can help in performing this essential service in a more efficient manner.**

Keywords: Snowplow Routing, Genetic Algorithms

I. INTRODUCTION

Routing problems in graphs have many real-world applications such as computing optimal routes for package-delivery vehicles, garbage-collection trucks or snowplows. In this paper, we address the problem of computing optimal routes for a fleet of snowplows serving a small town. Our earlier work ([5, 6]) discusses two different methodologies to compute optimal routes for snowplows. In both these papers, our algorithms compute a near-optimal route for a *single* snowplow that began at the depot, covered all the road segments at least once and returned to the depot. If the town was too large to be serviced by a single plow, the town graph would be manually divided into several sub-graphs, with each sub-graph serviced by its own plow. This practice of dividing the town into manageable sub-divisions is practiced in many small towns. Physical boundaries such as rivers, canals or arterial streets are frequently used to create these sub-divisions. In such cases, no assumptions can be made about the total mileage or the density of roads in each sub-division. As a result, while optimal routes for each sub-division could be computed, overall optimality may not be achieved.

In this paper we address the problem of optimal allocation of routes for multiple plows. Let np ($np >= 1$) represent the number of available plows. The road network is represented by a directed graph G where each edge has a *length* (a function of its mileage, which influences the cost of plowing it) and a priority (we assume that the value 1 represents the highest priority). Roads with higher priority are expected to be plowed earlier in the route. Also, u-turns are expensive and need to be minimized. Furthermore, the total *length* covered by each plow should be approximately the same. The total deadhead incurred by all plows (i.e., the length of roads on which plows drive without actually plowing) must also be minimized. We assume that the graph does not have multiple edges, self-loops or *sink* vertices (i.e., every vertex has at least one incoming and one outgoing edge). We also assume that some preprocessing as in [5] has been performed and the graph we have is a single bi-connected component. The vertices are numbered *0, 1,…,* with *0* as the depot. A *route-map* for a plow is a closed walk in the town graph G, i.e. it is a sequence $W = \{v_0, v_1, \ldots v_k\}$ of vertices of G, where each $(v_i\, v_{i+1})$ is an edge, and $v_0 = v_k = 0$. Note that a walk in a graph allows both repeated vertices and edges. A subsequence of the form $\{v_i, v_j, v_i\}$ constitutes a u-turn. The distance covered by a walk, $d(W)$ is the sum of the lengths of all the edges present in W. If an edge e_1 with priority p_1 occurs in W before an edge e_2 with priority p_2 and $p_1 > p_2$ (i.e. e_1 is less important than e_2) it constitutes a priority misplacement. The severity of this misplacement is the value $(p_1 - p_2)$. For each edge e with priority p in W, the priority misplacement index of e is the sum $\sum(p - p_i)$ over all edges e_i with priority p_i in W that occur before e and $p > p_i$. The misplacement index $m(W)$ of the walk W is the sum of the priority misplacement indices of all edges in W. The number of u-turns is denoted by $u(W)$. The minimum cover mc of the graph is the sum of the lengths of all edges in it; this establishes a lower bound for a complete walk. A *route allocation* to plows is a sequence $A = \{W_0, W_1, \ldots, W_{np-1}\}$ (np = number of available plows) where each W_i is a walk (not necessarily starting and ending at the depot), and the union of all these walks covers the entire edge set of G. It is assumed that the walk W_i is serviced by plow P_i. To perform this service, plow P_i starts at the depot, takes the shortest path to the first vertex of W_i without plowing (i.e. plow raised), plows along the edges of the walk W_i and reaches its last

978-1-4673-1714-6/12/$31.00 ©2012 IEEE

vertex. From there, it takes the shortest path back to the depot with its plow raised. The total length covered without plowing is called the *deadhead* $dh(W_i)$ of walk W_i. Let $d_{max} = \max(d(W_i))$ and $d_{min} = \min(d(W_i))$ where $d(W_i)$ is the total distance covered by the walk W_i in route allocation A. We define the *cost variation* of A as $cv = d_{max} - d_{min}$. Note that internal deadheads, i.e. deadheads caused by repeated road segments, are already included in each $d(W_i)$. Our problem is to find a route allocation that minimizes the total length $c = \sum d(W_i)$, the total deadhead $dh = \sum dh(W_i)$, total number of u-turns: $u = \sum u(W_i)$, total misplacement index: $m = \sum m(W_i)$, and the cost variation cv.

We use a Genetic Algorithms (GA)-based approach to address this problem. The advantages of using genetic algorithms to solve hard problems is that they can be applied to any optimization problem, they are easy to use and produce reasonably good solutions. The disadvantages are that there is no assurance of obtaining an optimal solution quickly. Further, depending on the complexity of the representation of individuals (chromosomes) and the algorithms to implement the genetic operators, the solution computation can be very time intensive.

Our GA-based solution to this problem is implemented in a Java application named SPRN. Our solution optimizes both – the division of the graph and the finding of optimal routes for each sub-division *simultaneously*. Experiments with graphs of varied sizes and real town graphs have produced very efficient route allocations in reasonable time.

This paper is organized as follows: Section II contains a short literature survey. Section III presents a summary of the GA methodology for a single plow. In Section IV we present an adaptation of this methodology for multiple plows. Experimental results are presented in Section V. Finally, in Section VI, we provide some conclusions and plans for future work.

II. LITERATURE REVIEW

Routing problems in graphs [9, 10] have many real-world applications and have been studied extensively in literature. The most famous of these, the 'Travelling Seller's Problem [7], is to compute a minimum cost cycle that begins at a start city, visits each city exactly once and returns to the start city. The counterpart of this problem for edges, namely that of computing a minimum cost circuit that covers all the edges exactly once (Eulerian circuit) has a simple solution if all the edges have an even degree. The undirected Chinese Postman Problem (UCPP) seeks to compute a minimum cost walk that covers each undirected edge *at least once*. If all vertices have an even degree, then an Eulerian circuit exists, and since no edge is repeated, it is a solution to the UCPP [8]. Otherwise, no Eulerian circuit exists, and the complete walk can only be achieved by repeating some edges. Kramberger and Zerovnik [3] mention many variations of the CPP and explore a specific version called the Priority Constrained CPP (PCCPP). In PCCPP, some vertices have priority values. These vertices have to be visited as early as possible in the walk. A minimum-cost walk which covers each edge at least once, and visits the prioritized vertices in the order of their priority is desired. They present a polynomial-time algorithm for the PCCPP. In some problems, edges may be categorized into priority classes, with the requirement that every edge in a higher priority class must be traversed first before an edge in a lower-priority class. Kazemi, Shahadi, Sharifzadeh and Vincent [2] have studied the problem of computing a minimum cost traversal of a road network. This problem is called the Optimal Traversal Problem (OTP). They consider the costs of edges as well as that of making turns. 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. Cabral, Gendreau, Ghiani, and Laporte [1] have proposed an algorithm to solve the CPP 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. While most papers consider the case with only one postman traversing the graph, Osterhues and Mariak [4] consider the k-CPP when there are k mailpersons who start and return to the base. The objective is not to minimize the total distance covered by all mailpersons, but to minimize the maximum distance covered by any single mailperson. The problem we address in this paper is similar to the k-CPP because of the consideration of multiple plows. Toobaie and Haghani [10] take a minimal arc partitioning approach to the truck routing problem for snow plowing and salting and report a reduction of 15% in the number of trucks used and 70% in the number of deadhead miles over existing plans. Salazar-Aquilar, Langevin, and Laporte [9] have proposed a mixed integer formulation and an adaptive large neighborhood search heuristic to solve the synchronized arc routing problem where two or more lanes of a road must be plowed simultaneously by many plows. Yeun, Ismail, Omar, and Zirour [11] present an analysis of various versions of the vehicle routing problem and discuss possible solution techniques.

III. GA FOR ROUTING WITH A SINGLE PLOW

In this section, we briefly describe the GA that computes optimal routes for a single plow [6]. In the next section, we modify this algorithm to work for multiple plows. Using a GA implies that we first need to compute candidate solutions which will become *individuals* of our GA population. Therefore, we first describe a simple *greedy* algorithm to compute a single complete walk that starts and ends at vertex 0.

A. Greedy Algorithm "computeCandidateWalk"

The walk is constructed as a sequence of vertices starting from the initial vertex – i.e., {0}. At every step, the algorithm appends a new vertex v to the walk constructed so far. The

new vertex v is the 'best' of all vertices that are directly connected to the last vertex on the partially constructed walk. It is selected using only local information – i.e., by comparing vertices using the *bestVertex* algorithm described in [6]. This process repeats until all edges are covered and the walk ends at the vertex 0.

B. Post-Processing the Walk for Improvement

Our assumption that there are no *sink* vertices ensures that there is always a complete walk for the graph. However, the walks constructed by algorithm *computeCandidateWalk* can have many redundant cycles and priority misplacements. The following post-processing algorithms improve the walks by eliminating redundant cycles (which reduces the total length of the walk) and by rearranging sub-walks (which reduces the misplacement index of the walk).

C. Redundant Cycle Elimination

Let $cyc = \{v_i, \ldots, v_i\}$ be a cycle in walk W. 'Eliminating' the cycle cyc consists of replacing the subsequence $\{v_i, \ldots, v_i\}$ by the single vertex $\{v_i\}$. A cycle is redundant if W is still complete (i.e. covers all edges) after it is eliminated. The algorithm starts at the right end of the walk and looks for a redundant cycle. If such a cycle is found, it is eliminated from W. The process continues until no more redundant cycles are found. Observe that this algorithm eliminates some redundant u-turns.

D. Sub-walk Re-arrangement

Let $s_1 = \{v_i, \ldots, v_j\}$, and $s_2 = \{v_i, \ldots, v_j\}$ be sub-walks of W that begin and end at the same vertices v_i and v_j,, but have no common vertices other than v_i and v_j. Also, s_2 occurs to the right of s_1 in W. The sub-walk s_2 is *better* than s_1 if swapping the two sub-walks in W reduces the overall misplacement index. The algorithm finds pairs of such walks and swaps them if appropriate. This process is repeated for other pairs of sub-walks, and for each feasible size of the sub-walk.

E. Example

Consider the rather poor-quality walk W = 0-5-4-3-2-1-0-1-2-3-4-5-**0-2-4-5-**1-2-**0-5-**3-2-5-4-2-1-5-2-3-5-0 (dashes are added just for readability) in graph G1 (Fig. 1). This has a misplacement index of 46. Consider the subsequences s_1 = **0-2-4-5** and s_2 = **0-5.** Their priorities are $p_1 = \{2, 2, 1\}$ and $p_2 = \{1\}$ respectively and we find that s_2 is better than s_1. After swapping these two subsequences we get W = 0-5-4-3-2-1-0-1-2-3-4-5-**0-5-**1-2-**0-2-4-5-**3-2-5-4-2-1-5-2-3-5-0 which has a misplacement index of 42.

F. GA Formulation

We now present the various components of a GA solution customized for our current problem.

1) Individual and Fitness Function: To apply GA to our problem, we identify the GA individual as a complete walk that begins and ends at 0. We define an individual's (i.e., a walk W's) fitness as:

$$f(W) = D * d(W) + U * u(W) * + M * m(W)$$

where $d(W)$, $u(W)$ and $m(W)$ are the total distance of the walk, the number of u-turns and the misplacement index of W respectively, and D, U and M are user-selected weights.

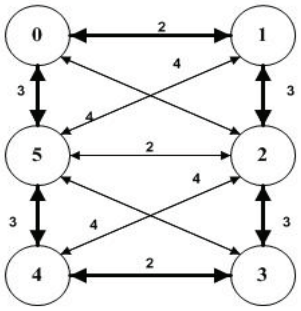

Figure 1. Graph G1

2) Initialization: We use a variation of the *computeCandidateWalk* algorithm itself to generate the initial population randomly. This algorithm, named *compute-RandomWalk*, chooses the next vertex randomly. The GA's initialization process calls this algorithm repeatedly to create an initial population.

3) Evaluation and Selection: After the initialization, each individual is evaluated using the fitness function defined above. Our selection strategy consists of sorting the population in the increasing order of their fitness and selecting a top performing (user-selected) fraction of the population.

4) Crossover: Pairs of individuals are chosen randomly from the selected population to undergo crossover. To perform crossover on two individuals W_1 and W_2, we choose a random start point i and an end point j, and identify the corresponding subsequence v_i to v_j in W_1. We find a subsequence in W_2 that begins at v_i and ends at v_j and swap the two subsequences. This process ensures that the resulting sequences are valid walks. However, they may not be complete. To complete them, another variation of our *computeCandidateWalk* algorithm, called *finishPartialWalk*, is applied to each of these two walks. This algorithm is the same as the *computeCandidateWalk* algorithm, except that instead of starting with the initial one-node walk {0}, it starts with a partial walk (with several nodes) and completes it. Thereafter, we use the redundant cycle elimination and sub-walk rearrangement algorithms to improve it.

5) Mutation: Individuals that have gone through crossover might undergo mutation as well. To perform mutation, we select a random subsequence v_i to v_j in walk W. We then partition W into three parts: prefix (0 to v_i), middle (v_{i+1} to v_{j-1}) and a suffix (v_j to end). We delete the middle and compute another sub-walk that connects v_i to v_j using another variation of our *computeCandidateWalk* algorithm, which computes a

complete walk starting with a given prefix and ending with a given suffix. Once the walk is complete, it is improved using our post-processing algorithms described in *C* and *D* above.

The population is checked for diversity and new random individuals may be added to improve it, as described in Section V. Results for this GA solution for a single plow (computed using our software package SPR1) were reported in [6].

IV. GA FOR ALLOCATION AND ROUTING OF MULTIPLE PLOWS

In this section, we present SPRN, a modified version of SPR1, which implements a GA solution for multiple plows.

A. Individual

The individual is represented by *(W, p)* where *W* is a complete closed walk starting and ending at the depot, and *p* is a specific partitioning of *W*. The partitioning is represented by the sequence $p = \{0, i_1, \ldots, i_{np}\}$ where $i_1, \ldots i_{np}$ are *indices* (indicating the location) of vertices in the walk, such that $0 < i_1 < \ldots < i_{np}$ and $i_{np} = NE$, where *NE* is the number of edges in *W* (note: the number of vertices in the walk is one more than the number of edges). This indicates that the *np* partitions are: $\{0$ to $i_1\}$, $\{i_1$ to $i_2\}$, … $\{i_{np-1}$ to $i_{np}\}$. For example: let walk *W* = {0-1-2-3-4-5-0-5-4-3-2-1-0}, *np = 4* and partition *p* = {0, 3, 5, 8, 12}. This implies that the partitioned sub-walks are: W_0 = {0-1-2-3}; W_1 = {3-4-5}; W_2 = {5-0-5-4} and W_3 = {4-3-2-1-0}. Thus, the route allocation for the four plows would be: *Route0* = {0-1-2-3, followed by a shortest path from 3 to 0}; *Route1* = {shortest path from 0 to 3, 3-4-5, shortest path from 5 to 0}; *Route2* = {shortest path from 0 to 5, 5-0-5-4, shortest path from 4 to 0}; and *Route3* = {shortest path from 0 to 4, 4-3-2-1-0}. The shortest paths from any vertex to any other vertex are computed only once by SPRN and stored before the GA computation begins.

B. Fitness Function

Let $I = (W, p)$ be an individual where *W* is partitioned into *np* sub-walks: $\{W_0, W_1, \ldots W_{np-1}\}$, as described in (A) above. We define the fitness of an individual as

$$f(I) = D * \sum d(W_i) + DH * \sum dh(W_i) + U * \sum u(W_i) + M * \sum m(W_i) + V * cv$$

where the summations are over all sub-walks, and *D*, *DH*, etc. are weights chosen by the user.

We use graph G1 (Fig. 1) to illustrate the fitness computation. Note that all edges are bi-directional where numbers indicate their length. Thick edges have priority 1, and thin edges have priority 2. This graph has: *NV* = 6 and *NE* = 22. Consider the complete closed walk: *W* = 0-1-2-3-2-1-0-5-4-3-4-5-1-5-2-4-2-0-2-5-3-5-0; and the partitioning: p = {0, 6, 12, 17, 22}. This means that the partitions are:

- W_0 = 0-1-2-3-2-1-0. This begins and ends at 0, so there are no deadheads. Total cost is 16 and deadhead = 0 + 0 = 0. $u(W_0) = 1$ and $m(W_0) = 0$.
- W_1 = 0-5-4-3-4-5-1. This begins at 0, so there is no initial deadhead. It ends in 1. So, there is a trailing deadhead = (cost of 1-0) = 2. Total cost is 17 and deadhead = 0 + 2 = 2. $u(W_1) = 1$ and $m(W_1) = 1$.
- W_2 = 1-5-2-4-2-0. This begins at 1. Leading deadhead (cost of 0-1) = 2. Since it ends in 0, trailing deadhead = 0. Total cost is 18 and deadhead = 0 + 2 = 2. $u(W_2) = 1$ and $m(W_2) = 0$.
- W_3 = 0-2-5-3-5-0. This begins and ends at 0. Therefore, total deadhead = 0. Total cost is 17. $u(W_3) = 1$ and $m(W_3) = 4$.

Observe that $d_{max} = 18$ and $d_{min} = 16$ and the cost variation *cv* = 18 – 16 = 2. Assuming that the weights *D*, *DH*, *U*, *M* and *V* are all 1, the fitness of this individual is 68 + 4 + 4+ 5 + 2 = 83.

C. Initialization

We employ the same initialization strategy used by SPR1 to generate the initial set of random walks. We now need to partition each walk in this set. We choose to do this strategically: Let *W* be the walk and the average cost *ac* = (total cost of all edges in *W*) / *np*. We just slice the walk into *np* pieces, such that they have approximately equal cost. To do this, we start from the first edge of the walk, and keep adding edges to it as long as the total cost is <= *ac*. When this happens, we start a new partition and repeat the process until all the edges are exhausted.

D. Crossover and Mutation

Only complete walks *W* (i.e., not the partitions) participate in crossover and mutation processes. These are done in exactly the same way as in SPR1. After the crossover and mutation operators have been applied, the resulting walks are partitioned strategically as described in the "Initialization" step above.

V. EXPERIMENTAL RESULTS FOR ALLOCATION AND ROUTING OF MULTIPLE PLOWS

In the following experiments using the SPRN package, we have assumed that there are four plows (*np* = 4) available. For each of the graphs below, we present a table of results for allocation and route planning. The notation for column headers in the tables below are: *a* = Route#; *b* = Leading deadhead; *c* = Walk representing the partition; *d* = Trailing deadhead; *e* = Cost of Leading Deadhead; *f* = Walk cost; *g* = Cost of Trailing Deadhead; *h* = # u-turns; *i* = Misplacement Index; and *j* = Total Cost of this Route = (*e* + *f* + *g* + *h* + *i*).

Example 1: The graph in Fig. 2 has *NV* = 21, *NE* = 56, and *mc* = 56. This graph looks like a four-leaf clover. The leaves are connected at the vertex 0. All edges in the leaf 0-1-2 … 0 have priority 1, those in 0-6-7 … 0 have priority 2, etc. All edge lengths are 1. If each of the clover leaves is treated as a

separate graph, there is a complete closed walk in each of them with no u-turns and no misplacements. Thus, if each of four plows is assigned one leaf, the whole graph is plowed with no deadheads, no u-turns and no priority misplacements. The result produced by SPRN is exactly what common sense (or manual analysis) dictates (Table 1).

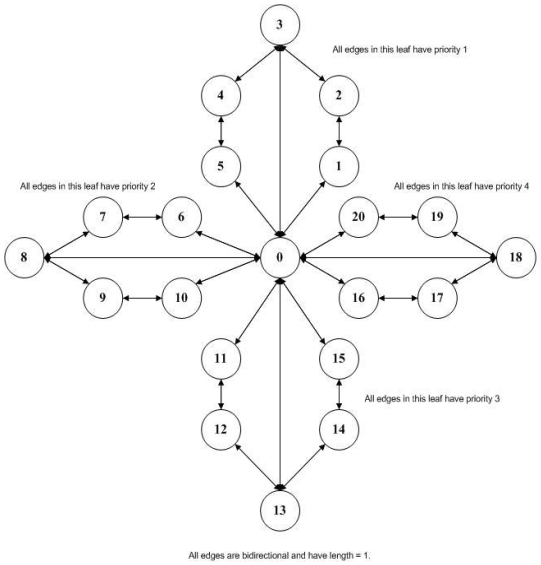

Figure 2. Graph G2 (*NV* = 21, *NE* = 56, *mc* = 56)

TABLE I. 4-PLOW SOLUTION FOR GRAPH G2

a	*b*	*c*	*c*	*e*	*F*	*g*	*h*	*i*	*j*
0	none	0-5-4-3-2-1-0-3-4-5-0-1-2-3-0	none	0	14	0	0	0	14
1	none	0-6-7-8-9-10-0-8-7-6-0-10-9-8-0	none	0	14	0	0	0	14
2	none	0-11-12-13-14-15-0-13-12-11-0-15-14-13-0	none	0	14	0	0	0	14
3	none	0-18-17-16-0-20-19-18-0-16-17-18-19-20-0	none	0	14	0	0	0	14
		Totals		0	56	0	0	0	56
		Cost Variation			0				**56**

Example 2: The graph in Fig. 3 is small, and our SPR1 package quickly computed an optimal solution with *d(W)* = 186, *u(W)* = 0, and *m(W)* = 0. Observe that the minimum cover *mc* for this graph is 186. Creating four routes for this graph is excessive, and results in a small amount of internal, leading and trailing deadheads. The 4-plow solution presented in Table 2 is still very good: internal deadhead caused by repeated edges is 192-186 = 6, and the deadhead incurred by the need to traverse the shortest paths to/from depot is 5+5 = 10, giving us a total deadhead of 16. This is less than 10% of *mc*, and it has only 2 u-turns and misplacement index = 0.

A. Experiments with other graphs

Table 3 below presents the results from our experiments with numerous other graphs (G4 - G19). The number of vertices in this set range from 11 to 71 and the edges from 31 to 218. All graphs meet the requirement that they are connected and that every vertex has at least one in-coming and one out-going edge. In fact, all graphs except G5 are completely bi-directional. While SPRN can be configured to work with any number of plows, the results reported here are based on 4 plows. The population sizes ranged from 32 to 128, and the crossover probabilities ranged from 0.8 to 0.9. Mutation probabilities ranged from 0.05 to 0.2.

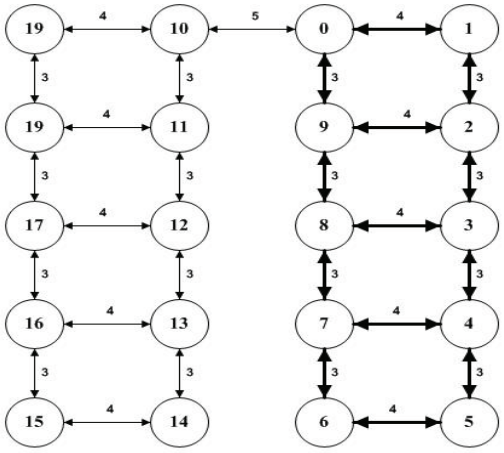

Figure 3. Graph G3 (*NV* = 20, *NE* = 54, *mc* = 186)

TABLE II. 4-PLOW SOLUTION FOR GRAPH G2

a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
0	none	0-9-8-7-6-5-4-3-2-3-8-9-2-1-0	none	0	46	0	1	0	47
1	none	0-1-2-9-8-3-4-5-6-7-4-7-8-9-0	none	0	48	0	1	0	49
2	none	0-10-19-18-17-12-13-14-15-16-13-12-11-18-19-10	10-0	0	53	5	0	0	58
3	0-10	10-11-12-17-16-15-14-13-16-17-18-11-10-0	none	5	45	0	0	0	50
		Totals		5	192	5	2	0	204
		Cost Variation			8				**212**

We have used an elitist policy in the selection step of the GA. This ensures that the all-time best performing individual is always a part of the current population. To promote diversity, we have used a replenish step. This step scans the population for duplicate individuals. If one is found, it is replaced by a randomly generated individual. Most experiments were run with maximum allowed iterations set at 300. The weights for fitness calculations were all set to 1. This amounts to treating each u-turn and each priority misplacement equivalent to driving one extra mile.

Notation used in Table 3: *NV* = number of vertices, *NE* = number of edges, *mc* = minimum cover, *td* = total distance covered by all plows (not including the shortest paths to/from the depot), *cv* = cost variation, *dh* = total deadhead caused by shortest paths to/from the depot, *u* = total number of u-turns,

m = misplacement index, *f* = fitness, *It* = the number of GA cycles at which the best fitness was achieved. Note that since the *mc* column of Table 3 represents the total distance covered by plowing each road exactly once, it is the best possible value that column *td* can have.

TABLE III. ROUTE ALLOCATION FOR FOUR PLOWS IN OTHER GRAPHS

Graph Characteristics				*GA Solution Characteristics*						
Id	*NV*	*NE*	*Mc*	*td*	*cv*	*dh*	*u*	*m*	*f*	*It*
G4	11	36	360	360	0	120	3	0	483	23
G5	13	31	292	292	10	120	5	120	547	11
G6	15	38	380	400	0	240	8	102	750	3
G7	15	52	520	520	0	100	0	0	630	85
G8	16	48	480	480	0	120	1	0	601	3
G9	16	54	540	560	0	120	1	0	681	59
G10	16	66	258	330	7	30	19	98	484	66
G11	18	58	580	600	0	120	2	0	722	55
G12	22	76	760	800	0	120	5	0	825	86
G13	23	76	760	840	0	80	2	0	922	30
G14	25	96	356	356	7	20	13	54	450	34
G15	25	76	760	760	0	80	2	0	842	29
G16	28	82	820	880	0	100	4	0	984	62
G17	48	154	356	376	0	42	8	0	440	133
G18	52	136	506	520	0	62	28	127	737	77
G19	71	218	2180	2240	0	160	8	0	2408	90

B. *Brief Analysis of the Results in Table 3*

Let us first consider large graphs where using multiple plows is advantageous. In graph G17, the minimum cover is 356. The total deadhead in this case is 62, computed as the sum of the *dh* value (42) and the difference between *td* and *mc* (20). This amounts to only 17.41% of the minimum cover. Cost variation and misplacement index are both 0, indicating that all the four routes had equal length and there were no priority misplacements, thus indicating a very good quality solution. Similarly, in G18, we observe that the total deadhead is 15.02% and all routes had equal length. However, the solution had 28 u-turns and a misplacement index of 127, making it less optimal in the number of u-turns. In the largest graph, G19 (*NV* = 71 and *NE* = 218), the total deadhead was 10.09%, route cost variation was 0, the number of u-turns was 8 and the misplacement index was 0. Considering the size of the graph, this is an excellent quality solution. Results for medium sized graphs G12 – G16, show that the average of the total deadheads was 15.55%, cost variations and misplacement indices were mostly 0 (except in G14) and the average number of u-turns was about 5. It is evident that the results for these graphs are reasonably good. For smaller graphs, G4 – G11, it may be better to use just one or two plows; use of four plows is perhaps overkill. Average deadhead for graphs G4 – G11 is over 30%. However, the cost variations, number of u-turns and misplacement indices are almost always 0 (except in G5 and G10). Our experimental results confirm what commonsense dictates: using multiple plows for small graphs results in large amounts of deadhead and huge misplacement indices. However, for larger graphs, the total deadheads are less than 20%, which is considered very reasonable.

VI. CONCLUSIONS AND FURTHER RESEARCH

In this paper, we have presented a GA solution to the problem of optimally allocating routes for a small fleet of snowplows. The novelty of our approach is that it treats the optimal division of a town graph, and the computation of optimal routes for each subdivision as components of the *same* problem. We have created a unique GA solution that optimizes both components simultaneously. Our application attempts to optimize on various parameters such as the total deadhead, number of u-turns and priority misplacement index. It ensures that the computed routes have approximately equal lengths. SPRN, our software application, computes near-optimal solutions to reasonably large problems in a moderate amount of time. Currently, it produces text files containing the solution. Our plan is to develop software that can export these files ultimately to a GPS system, so that the driver can receive turn-by-turn instructions while plowing. We also plan to explore the application of other evolutionary techniques such as scatter search and particle swarm optimization to this and other related routing problems.

REFERENCES

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

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

[3] T. Kramberger, and J. Žerovnik, "Priority Constrained Chinese Postman Problem," J. Logistics & Sustainable Transport, vol. 1 (1), 2007 (no page numbers given).

[4] A. Osterhues, and F. Mariak, "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 December 15, 2010.

[5] T. M. Rao, S. Mitra, and J. Zollweg, "Snow-Plow Route Planning using AI Search," Proceedings of the 2011 IEEE International Conference on Systems, Man and Cybernetics, Anchorage, AK, October 2011, pp. 2791-2796.

[6] T. M. Rao, S. Mitra, J. Zollweg, and F. Sahin, "Computing Optimal Snow Plow Route Plans using Genetic Algorithms," Proceedings of the 2011 IEEE International Conference on Systems, Man and Cybernetics, Anchorage, AK, October 2011, pp. 2785-2790.

[7] S. Russel, and P. Norvig, Artificial Intelligence: A Modern Approach (2nd Edition). Upper Saddle River, NJ: Prentice Hall, 2003.

[8] H. Thimbley "The directed Chinese Postman Problem," Software Practice and Experience, vol. 33 (11), pp. 1081-1096, 2003.

[9] M. A. Salazar-Aguilar, A. Langevin, and G. Laporte, "Synchronized arc routing for snow plowing operations," Computers and Operations Resecrch, vol. 39 (7), pp. 1432-1440, 2012.

[10] S. Toobaie, and A. Haghani, "Minimal Arc Partitioning Problem," Journal of the Transportation Research Board, No. 1882, TRB, pp. 167-175, 2004.

[11] L. C. Yeun, W. R. Ismail, K. Omar, and M. Zirour, "Vehicle Routing Problem: Models and Solutions," J. Quality Measurement and Analysis, vol. 4 (1), pp, 205-218, 2008.