FORT STEWART SHORTEST PATH ANALYSIS OF DEBRIS CLEANUP OPTIONS

by

MICHAEL ANTHONY WALLISER

(Under the Direction of Pete Bettinger)

ABSTRACT

Several heuristic algorithms are developed to find the fastest set of routes in order for a fleet of cleanup crews to clear hurricane-related debris from the road network at Fort Stewart near Savannah, Georgia. The road network is divided into priority levels such that each level must be completely cleared before work can begin on the next level. The problem is similar to the Hierarchical Chinese Postman Problem and the Capacitated Arc Routing Problem. It presents a unique challenge, however, in that each road must be cleared before it can be used for transport. Rule-based heuristics, adaptations of local beam search, and genetic algorithms are tested in various combinations with each other to build solutions. Results are evaluated based on the time required to clear the entire network of debris. The best solutions are generated by using a combination of all three of the aforementioned heuristics.

INDEX WORDS: Arc routing, Hierarchical Chinese Postman Problem, Capacitated Arc Routing Problem, Path finding, Search heuristics, Local beam search, Genetic algorithm

FORT STEWART SHORTEST PATH ANALYSIS OF DEBRIS CLEANUP OPTIONS

by

MICHAEL ANTHONY WALLISER

BS, Georgia Institute of Technology, 2005

MEd, University of Georgia, 2009

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2012

FORT STEWART SHORTEST PATH ANALYSIS OF DEBRIS CLEANUP OPTIONS

by

MICHAEL ANTHONY WALLISER

Major Professor:	Pete Bettinger
Committee:	Walter D. Potter
	Charles Cross

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
May 2012

ACKNOWLEDGEMENTS

Above all, I would like to thank Pete Bettinger for his guidance and assistance in completing this thesis. His ready availability and willingness to help made the project possible. I would also like to thank Don Potter and Charles Cross for serving on my committee. Special thanks go to Shawn Baker for going out of his way to help me with data analysis. Finally, I want to express my gratitude to everyone at the Institute for Artificial Intelligence who has helped me in any way, whether they know it or not.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

Emergency management is an area that often gets overlooked when it comes to the advancement of society, yet will always present relevant operational problems. For man-made disasters such as nuclear accidents and airplane crashes, it is obviously desirable to prevent the incident from occurring in the first place. In those cases, a reaction plan is a secondary concern to a prevention plan. Natural disasters, on the other hand, are completely unpreventable, so it is crucial to plan for them to happen and be prepared when they strike.

The Federal Emergency Management Agency (FEMA) defines emergency management as "the process of preparing for, mitigating, responding to and recovering from an emergency" [1]. Being well prepared for a disaster can enhance the efforts in any or all of the other three areas mentioned. According to the American Planning Association [2], it is "absolutely vital that planning research seek out the answers, identify the best practices, and make clear how they relate to the unique circumstances of each new community that experiences a disaster." The problem addressed in this thesis particularly involves preparing for the response and recovery aspects of emergency management.

1.1 The problem

The problem was posed by the United States Army and concerns the management of roads at Fort Stewart, which is located near Savannah, Georgia. The installation's proximity to the coast of the southeastern U.S. makes it vulnerable to hurricane-force winds. Such winds

have the potential to fell trees, covering the roads with debris and thus making them impassable. The overarching task, then, is to develop a set of routes for a fleet of road-clearing crews to follow during the cleanup effort. This is a single-objective combinatorial optimization problem, with the objective being to minimize the time required to clear *all* the roads of debris. The roads are divided into different priority levels, and the crews are assumed to operate with the constraint that all roads of a given priority must be cleared before beginning on any roads of a lower priority. There are many other ways to address the road prioritization, some of which may be better in practice, but the aforementioned constraint is what was specified by the Army. Because this is a hard constraint, it is necessary for the set of roads of any priority level, in union with all the roads of higher priority levels, to form a continuous network. This concern was considered in the assignment of a priority to each section of road.

In this particular type of problem, the number of crews utilized must be known prior to the development of a plan, and it is assumed that at least that many systems of equipment are staged at a known position prior to direct impact of the wind. These systems of equipment are assumed to be used throughout the recovery effort, with no more added and none assumed to break down. Some other assumptions were made in order to make the problem tractable. They include:

- A priority list of roads has been developed prior to the development of the plan.
- Once a road is clear, it stays clear. No new debris is deposited afterward.
- All roads can be cleared using the same equipment and processes available to each crew.
- The crews will initially leave from a pre-designated area (depot).
- The rate of speed on cleared roads is known, as is the rate of road-clearing operations. These speeds are not necessarily consistent among all roads.

- Crews change travel speeds instantaneously when moving between different speed zones.
- All crews work at the same pace under similar conditions.
- Crews will work non-stop until all the roads have been cleared.
- There is no other traffic on the road that will impede the recovery process.
- The amount of tree-related debris falling onto roads (the demand) is constant.
- Having two crews in the same place at the same time will double the speed of clearing. A third crew will provide no extra value.

1.2 Generalizing the problem

This problem combines elements of the Hierarchical Chinese Postman Problem (HCPP) and the Capacitated Arc Routing Problem (CARP). In the traditional Chinese Postman Problem (CPP), the goal is to minimize the time (or travel distance) required to traverse every arc in a given network and return to the starting point, representative of a postman delivering to every address in the network and then returning to the post office. The HCPP adds the stipulation that some roads have a higher priority than others, and therefore an order of precedence must be followed.

In the CARP, a fleet of vehicles sets out from an initial location and "must service a subset of the edges of a graph, with minimum total cost and such that the load assigned to each vehicle does not exceed its [known] capacity" [3]. An example of a CARP would be the routing of school buses, where all the relevant roads must be traversed by at least one bus, but the load of each bus is limited by its carrying capacity.

Like in the HCPP, every arc must be traversed, and the priority constraints are strictly enforced. Returning to the starting point, however, is not a concern, as the ultimate goal is

reached as soon as all the roads have been cleared. Like in the CARP, there are multiple vehicles, but in this problem, the vehicles have an unlimited service capacity.

There are several aspects of this problem, however, that are not part of the HCPP or CARP. Each road requires service on the first traversal, but after being serviced, the demand load is removed from the cost of traveling the road. In the traditional versions of these problems, each arc must *eventually* be serviced but can also be traversed before being serviced. The utility of teamwork is also factored into this problem, as two crews working together can clear a road twice as fast, though any more than two is a waste of resources.

Having multiple vehicles, as is considered in the CARP, makes this problem immensely more complex than the HCPP. The path chosen by one crew can have a great impact on how much work another crew can accomplish. Consider the portion of the road network depicted in Figure 1.1, and assume five working crews. Roads 18 and 16 provide the only avenue to reach the rest of the network, so until those two have been cleared, progress is limited to the area shown. Furthermore, that sequence will take much longer to clear than the rest of the area shown. A single crew would want to clear the entire area near the depot first and then proceed to Road 18, so it would not have to waste time returning to the area. However, with a fleet of five crews, clearing the area near the depot first would then lead to three crews just "sitting around" while two worked on Road 18 and then Road 16. Rather, it would be advisable for two crews to begin on Road 18 immediately. Since that task will take a relatively long time, the other three crews can clear the area near the depot in the meantime. Ultimately, which road to clear next depends on a combination of which roads have been cleared, where the other crews are, where they plan to go next, and the specific characteristics of the road (such as speed limit and length).

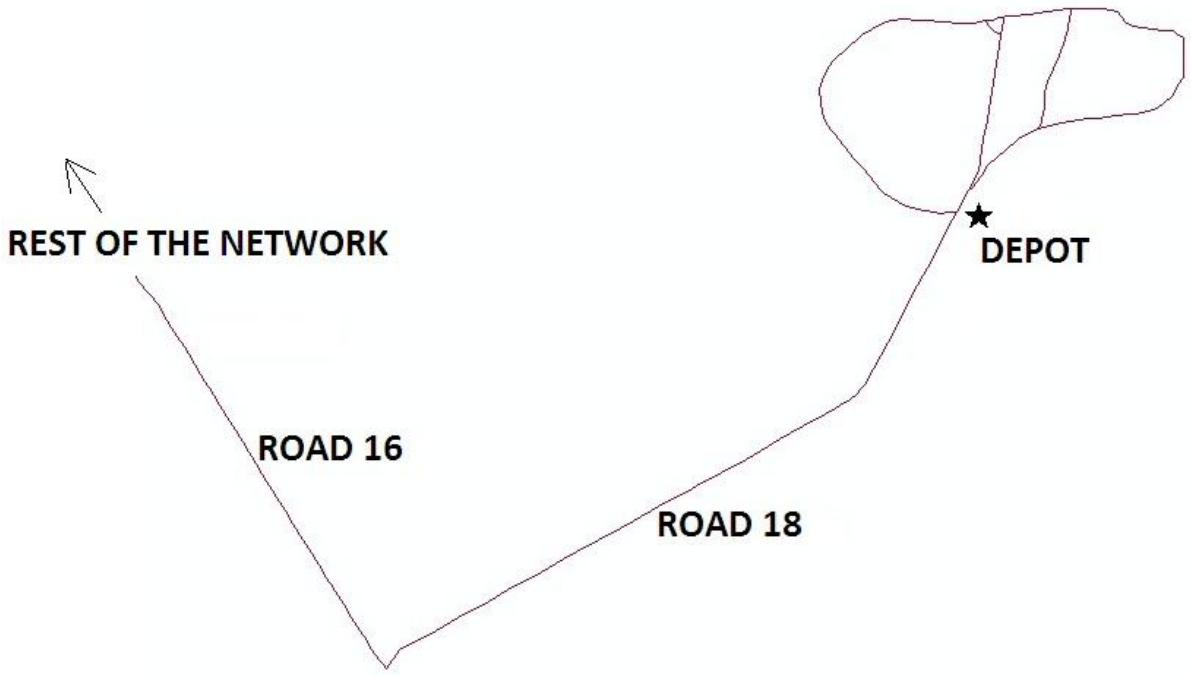

Figure 1.1: View of the road network near the starting point.

The HCPP is NP-hard [4], as is the CARP [5]. It follows that all algorithms that might be developed for problems such as these can be considered heuristics [6]. Most practical problems have their own nuances that make them different from other problems that fit into the HCPP and/or CARP categories, so most available documentation is on problems with varying degrees of *similarity* to this one, but some crucial difference as well.

1.3 A similar type of problem

A category of problems very similar to this one is the Snow Plow Problem (SPP). In the SPP, a fleet of snow plows must leave a depot and clear snow from a network of roads, much

like how the fleet in this problem must clear debris from the roads. Different versions of the SPP make different assumptions, and in some cases the hierarchical component is even included. A general feature of the SPP that differs from the Fort Stewart problem, though, is that snow plows can drive over roads that have not been serviced, while the cleaning crews at Fort Stewart must clear the path in order to make it usable.

One heuristic that has been used to address the SPP and other types of CARP problems is the *cluster-first, route-second* approach [6][7][3]. In this method, the network is partitioned into a set of sub-networks (clusters) of similar workloads, each of which is assigned to a vehicle. Then, each cluster is solved as a single-vehicle arc-routing problem. The corresponding single-vehicle problems can be approached in a number of ways, such as by transforming them into Rural Postman Problems [8].

This *cluster-first, route-second* method would presumably not lend itself well to the Fort Stewart problem, based on several problem-specific considerations. The requirement that roads be cleared on the first traversal, along with the fact that all crews start at the same location, would greatly augment the challenge of assigning clusters. Additionally, keeping the vehicles in separate clusters precludes the possibility of two crews teaming up on a section of road.

Another approach to the SPP involves constructing an initial set of feasible routes to be traversed, and then developing methods to attempt to improve these routes by means of different *destroy/repair* operators [9]. This method, however, is also much more appropriate when applied to local neighborhoods being serviced by a single vehicle. When considering the interdependence of the service vehicles, both in the Boolean terms of completing the goal and in the quantitative terms of how long it will take to traverse each arc, designing the repair operator

becomes a substantial problem in itself. Mei et al. [10] have developed a global repair operator for the CARP, but this operator assumes the ability to traverse a road before servicing it.

Amponsah and Salhi [11] developed a "look-ahead" strategy for a constructive heuristic pertaining to garbage collection, another type of CARP. While the Fort Stewart problem has the single objective of minimizing time, this particular garbage collection problem has the added objective of minimizing inconvenience due to smell from the garbage. A maximum load per vehicle is also a consideration in the garbage collection problem, and this limitation plays a strong role in the "look-ahead" algorithm.

A very basic approach which could be handled with little to no computational power is a rule-based heuristic, where a vehicle chooses its path based on a predetermined set of rules. A sufficiently descriptive set of rules could make for a deterministic path [12][13][14][15]. With the size of the network in this problem and all the considerations such as loops, dead-ends, etc., it would be challenging to find *any* feasible solution using a deterministic set of rules. Previous work on the Fort Stewart problem was performed by Bettinger et al. [16] using deterministic rules based on Dijkstra's algorithm, a shortest-path algorithm described in the Methods section. Their work was incomplete and especially encountered problems dealing with loops.

As an alternative to deterministic rules, the rules could be used simply to narrow down the possible choices at each intersection, ultimately leaving some decisions to chance. Such a set of rules would provide general guidelines on how the vehicle should choose its path. The quality of a solution developed in this way is thus left to chance, but such solutions could be improved.

In approaching the Extended CARP (which is the CARP with a few extensions that do not make it any more like the Fort Stewart problem), Xu et al. [17] use a combination of several classical heuristics that are integrated into a single standard heuristic (a genetic algorithm). The

classical heuristics are used to develop a series of possible solutions, which become the initial population of the genetic algorithm.

In a genetic algorithm [18], a population of solutions is evolved over a number of generations, mimicking the process of evolution via natural selection. Members of the population are evaluated according to their "fitness" or quality, and the better solutions are more likely to be selected to "breed" with other solutions or simply move unchanged into the next generation. The "breeding" process happens through crossover, where corresponding parts of two solutions are switched, thus creating two new solutions. Occasionally, a "mutation" happens, where a small part of a solution changes, much like how mutations occur in real life. When a population goes through the process of selection, crossover, and mutation, a new generation is created. As the evolution progresses, better solutions are formed, and eventually (whether due to convergence or reaching some time/generation limit) a solution is settled upon.

This research uses the aforementioned idea of integrating classical heuristics with a genetic algorithm to develop a solution to the Fort Stewart road debris clearing problem. This represents an application of previous research to a new problem, and involves enhancements that are necessary to address this particular land management issue.

CHAPTER 2

METHODS

This research involves the integration of classical heuristics with a genetic algorithm in order to develop a solution to a road clearance problem. The first classical heuristic involves a small set of rules to provide some guidelines for the crews. Two separate sets of rules were tested, and they are described below. The second heuristic was derived from the idea of a search heuristic called *local beam search*, though it does not follow this heuristic exactly.

2.1 Heuristic search process

2.1.1 Rule-driven decisions

Two different sets of rules were tested, plus a *random-choice* method. A crew would invoke the designated set of rules any time it reached an intersection or dead-end and needed to choose its next direction. The aim of each set of rules was to preclude the crews from choosing paths that would lead to time wasted, while still allowing for the potential to explore paths that may not seem like the best *immediate* choice but may prove to be better in the long run. Both sets of rules utilize the concept of Zero Marginal Worker Effectiveness (ZMWE), which stems from the stipulation in the problem that two crews working together on a stretch of road can clear debris at twice the speed of one crew, but any additional crews beyond two provide no added value. Thus, a section of road with two or more crews already working on it is said to be at ZMWE.

Solutions were generated from these rules by way of simulation. The crews are identified by number, and they start out from the depot at the same time. As they move through the map, they keep track of their path, and the simulation keeps track of which roads (arcs) have been cleared and which have not. It is important to note that when multiple crews are choosing their next directions, they choose in order of their ID number, as opposed to simultaneously.

2.1.1.1 Rule method 1

Rule method 1 works as follows. If there is only one arc from which to choose, then the crew will choose it. Otherwise, a list is made of all the arcs that connect to the crew's current intersection or dead-end (i.e., node) and do not violate the priority constraint. If the crew is arriving from an arc that had already been cleared, this most recent arc is removed from the list of possibilities. Using the remaining list:

1. If there are any arcs that have not been cleared and are not at ZMWE, choose randomly from among those.
2. Otherwise, if there are any arcs that have been cleared, choose randomly from among those.
3. Otherwise, choose a random arc.

The general approach represented by these rules is to clear a road whenever one is found that has not been cleared, to avoid making U-turns when driving on cleared roads, and to avoid roads that are already at capacity with workers. In some cases, these guidelines will determine with certainty which arc to choose next, while in other cases, they may only narrow down the list of options. This decision method is represented as a flow chart in Figure 2.1.

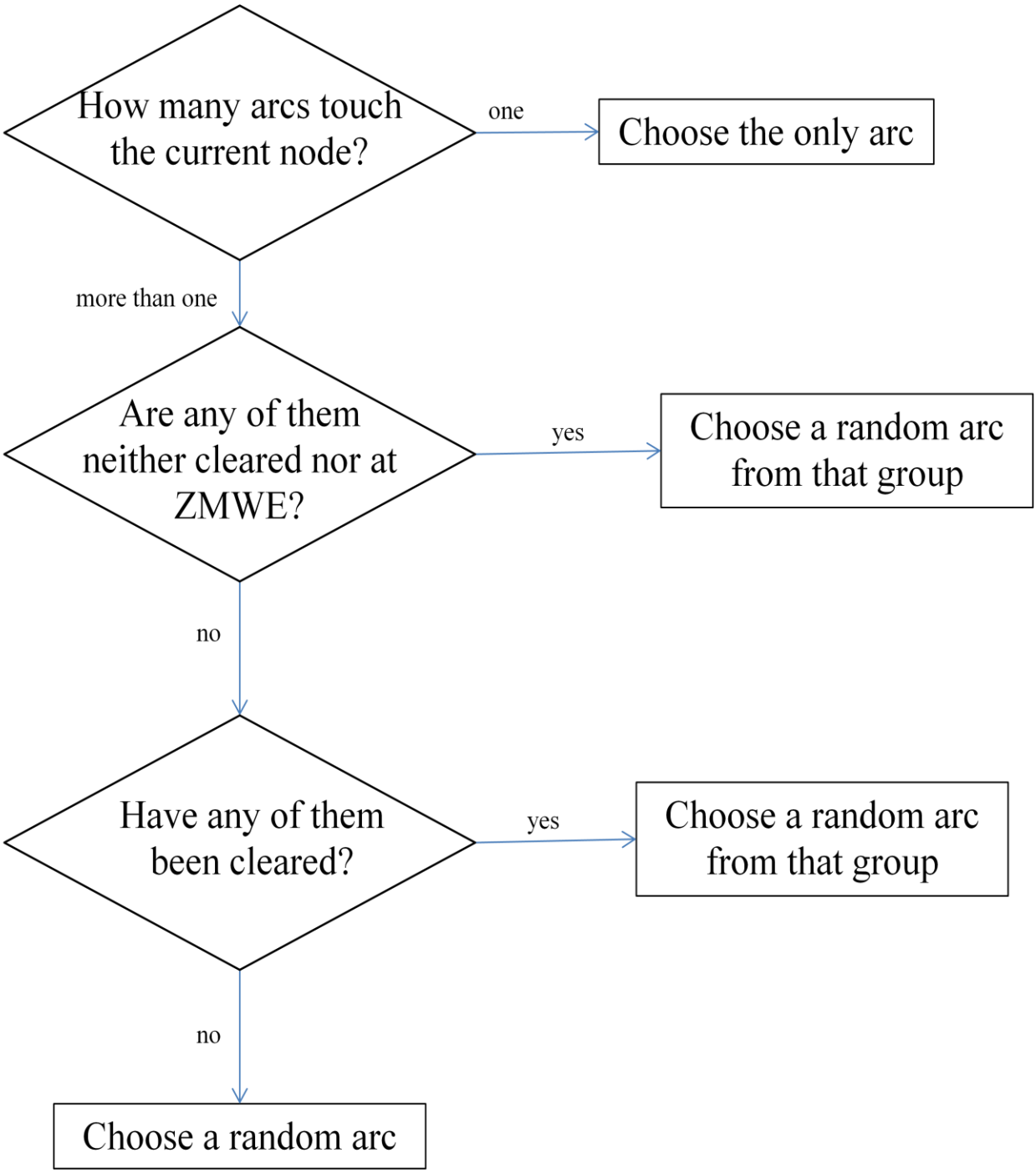

Figure 2.1: Flow chart representing rule method 1 for choosing the next direction at an intersection or dead-end.

2.1.1.2 Rule method 2

This method takes into account *dead-end paths*, which must be properly defined in the context of this problem. A dead-end path is a connected set of arcs (road segments) that, if traveled in one direction, will only ever offer one choice of direction in addition to turning around; eventually, the path reaches a node where turning around is the only option. It is important to note, however, that for this purpose, the number of choices at a node includes roads up to and including the current priority level as well as those of the next priority level.

The reasoning behind this extra inclusion is that at any given time, the next priority level might become accessible. Without including that priority level in the dead-end definition, there would be cases where the crews would be prevented from being in the best possible position to begin work on it. Consider, for instance, a case where the crews are currently working on priority level 2. Thus, they are allowed to travel on roads of priority 1 or 2. Figure 2.2 shows the map of the network consisting only of priority levels 1, 2, or 3.

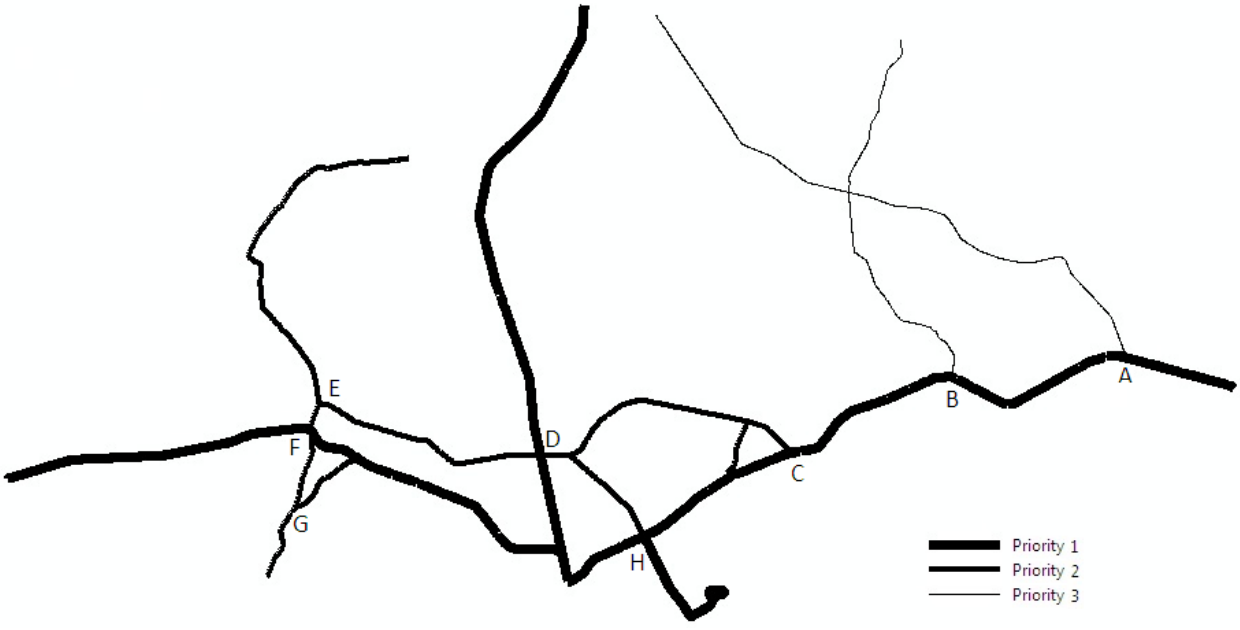

Figure 2.2: Fort Stewart network of roads of priority levels 1, 2, and 3.

Ideally, upon completing priority level 2, there will be crews near points A and B, ready to begin clearing priority level 3. Thus, crews are allowed to go east of point C before priority level 2 is cleared. However, since the section of road east of point A has been cleared, and there is no need to revisit it to get to roads of priority 3, it is labeled a dead-end path. The sections of road extending north from points D and E, west from point F, and south from point G are labeled dead-end paths for the same reason. Since the section of road southeast of point H ends in a loop, it avoids classification as a dead-end path. It is of note that a road's status as a dead-end path is dependent on which priority level is currently being cleared.

Rule method 2 attempts to minimize time wasted on dead-end paths, and it operates as follows. Starting with the list of all the arcs that connect to the crew's current node and do not violate the priority constraint:

1. If the crew is on a dead-end path that does not need help (i.e. either the dead-end path has been cleared or an arc on the dead-end path is at ZMWE), then move away from the end of the dead-end and back toward the rest of the road network.
2. Until only one arc remains (which may already be the case), eliminate roads in the following order. If multiple roads fall under the same guideline, they are removed one at a time in random order:
    a. If the crew is coming from an arc that had already been cleared, eliminate that arc.
    b. Eliminate any arc that is on a dead-end path that has already been cleared.
    c. Eliminate any arc that is at ZMWE.
    d. Eliminate any arc that has already been cleared.
    e. Eliminate the remaining arcs.

These rules take into consideration the same issues as Rule Method 1, with the additional provision of avoiding needlessly traveling down dead-end paths. Figure 2.3 shows a flow chart representing rule method 2.

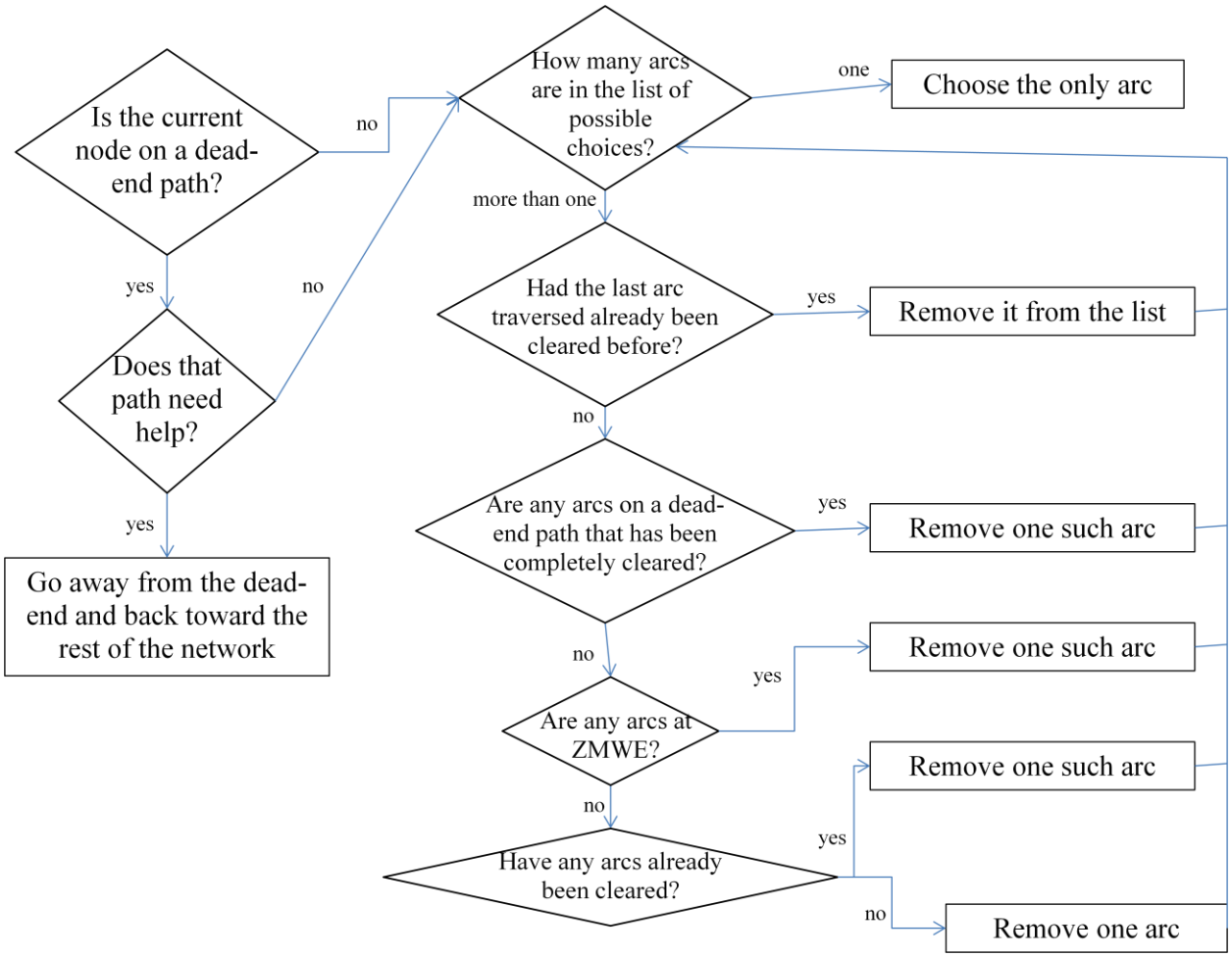

Figure 2.3: Flow chart representing rule method 2 for choosing the next direction at an intersection or dead-end.

2.1.1.3 Random-choice method

This method involves choosing randomly from all arcs that have an endpoint at the crew's current node and do not violate the priority constraint.

There are advantages and disadvantages to each method. Rule method 1 has a very simple set of rules that prevents some inefficient behaviors. However, there are some scenarios where those rules are too simplistic and actually prevent the best choice from being made. For instance, it might be better in some cases to move past some blocked roads in favor of clearing some other roads further down the line. Rule method 2 adds dead-end paths to the consideration, presumably preventing even more inefficient behaviors. Again, though, there may be some circumstances where the rules backfire. For example, upon the completion of priority level 2, being east of point A (as labeled in Figure 2.2) would actually put a crew in good position to start on priority level 3. More rules also mean a higher computational load. The random-choice method allows the crews to take wildly inefficient routes, but each decision requires less of a computational load than the rule-based methods. Due to the large number of decisions that must be made in this problem, the probability of finding a better solution with the random-choice method than with either rule method is extremely low.

2.1.2 Local beam search

Local beam search occurs in a search tree as follows [19]: "It begins with *k* randomly generated states. At each step, all the successors of all *k* states are generated. If any one is a goal, the algorithm halts. Otherwise, it selects the *k* best successors from the complete list and repeats." The idea behind this method, as it relates to the Fort Stewart problem, is that the best possible complete solution was very likely one of the best solutions when evaluated at any point during the process. Two different ways were implemented of evaluating a solution before it was completed. These implementations are discussed below.

Building a search tree for this problem would be a complex and confusing task. A solution to the problem represents a set containing one path for each crew. The crews, however, make their decisions upon reaching intersections or dead ends, which is not generally at the same time for all crews. Thus, one node in the search tree might represent a decision made by one crew, while another node represents a decision made by another crew. Yet another node may represent the combination of decisions made by two or more crews, if they happened to reach the ends of their respective arcs at the same time. In order to know which crew will be making the next decision, the algorithm has to know how much time each crew will require to reach the end if its current arc. To calculate this time requires knowledge of (a) the length of the arc, (b) the position of the crew on the arc, (c) whether the arc has already been cleared, (d) how many other crews are on the arc (if it has not been cleared), and (e) the maximum speed on the arc (if it has been cleared). Much of this information is dependent on the paths taken up to the current state, so the algorithm would either have to store this information in memory as it moves down the tree, or else recalculate it at every search node. The relevant information was kept in memory by creating each path through the search tree as a simulation of the road-clearing process. The idea of local beam search was adapted to fit the simulation.

Two adaptations of local beam search, *constant time beam search* and *constant progress beam search,* were developed and tested in the experiment. In the former, a set of s simulations (where s will be called the *exploration factor*) are run for a predetermined amount of simulated time. Upon reaching this time threshold, each simulation is evaluated based on its progress, as determined by the length of road that has been cleared. The k best simulations (where k will be called the *filter size*) are selected for advancement, while the rest are discarded. Each surviving simulation makes $(s/k - 1)$ copies of itself, returning the total number of simulations in progress

to *s*. Each of these simulations is then run, continuing from where it left off, for the same predetermined amount of time. They are then evaluated based on their progress and either discarded or selected for advancement. Those that are selected for advancement again replicate themselves to fill out the set, and the process continues until a solution is reached.

Constant progress beam search works in much the same way, but the roles of time and progress are reversed. Each simulation runs until reaching a predetermined amount of road cleared. The simulations then are evaluated and compared based on the amount of simulated time required to reach that progress threshold. Again, the *k* best are selected for advancement, and they replicate themselves to return the set of simulations to its original size. These simulations continue from where they left off, with each one running until it again reaches the progress threshold. The process repeats as such until a solution is reached.

An offshoot of constant progress beam search uses priority levels as the progress benchmarks. The progress is not constant, since each priority level contains a different amount of road, but the divergence of solutions does still occur at a predetermined threshold. This adaptation will be called *level-based beam search*.

2.1.3 Genetic algorithm

Applying the genetic algorithm presents the previously-noted challenge of creating a repair operator. When new paths are created from crossover, infeasible solutions often arise because the new paths have disconnects. Crossover and mutation could also create problems adhering to the priority constraints. Employing an appropriate representation is a key to avoiding these problems or, at the very least, allowing for a reasonably simple repair operator to fix them.

One straightforward genetic representation of a solution would be for each "step" to represent a gene, where a step is simply one arc on the path of one crew. The solution would then look like one long strand divided into sub-strands representing the paths of each crew, as illustrated by the two solutions in Figure 2.4 (which assume 4 crews).

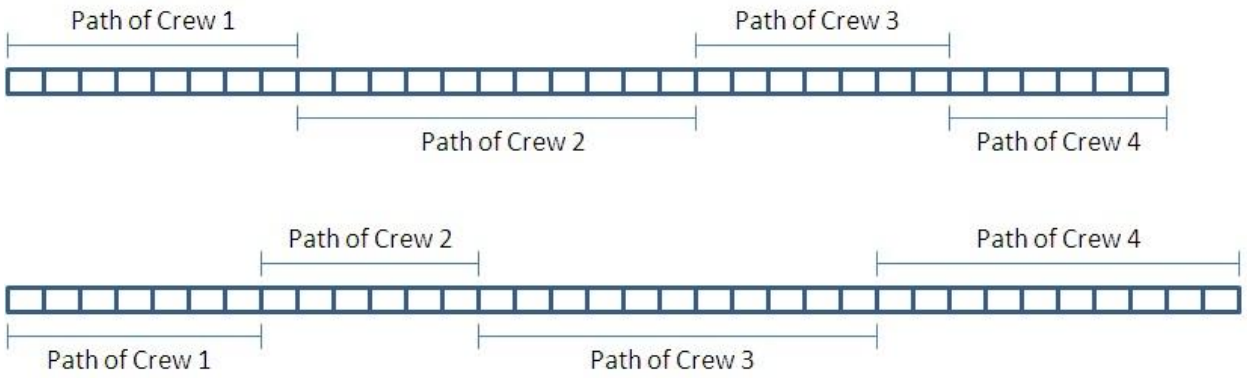

Figure 2.4: Model of one possible genetic representation of two solutions. It assumes four crews, and each gene represents one arc on the path of one crew.

An advantage to this representation is that a gene represents the most basic component of a solution (an arc), so the entire search space is attainable through the evolutionary process. This representation also presents many significant problems, though. Because solutions are of different lengths, it must be checked that any crossover point actually *exists* in both solutions. Moreover, the crossover point should be located in corresponding sub-strands in order to keep the solution properly divided into separate paths for each crew. It may be necessary for two parent solutions to have crossover points at different indices, but that is acceptable since solutions will differ in their lengths anyway.

Supposing that the crossover leads to a disconnected path, a sub-path would need to be inserted into the child solutions to make them feasible. This concept is illustrated in Figure 2.5.

Creating these "filler" paths presents a separate challenge. It might be that the path can be traveled by using only cleared roads. In this case, Dijkstra's algorithm (described below) can be used to find the shortest path. However, there is also a chance that to get from one point to the next requires clearing new roads. When this happens, the problem of finding the best path is integrated into the larger problem. The quickest path might involve the least amount of road-clearing, but then again, it might be better to take a little longer and do *more* clearing along the way. The proper balance can only be determined by assessing the resulting quality of the larger solution.

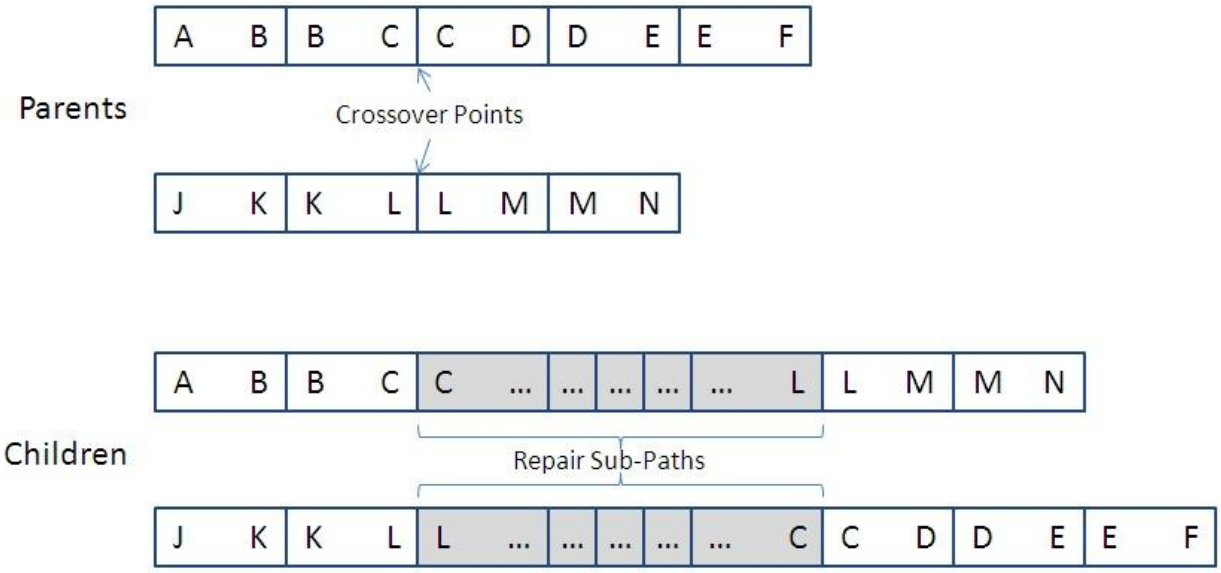

Figure 2.5: Conceptual illustration of a repair operator to make child solutions feasible after crossover.

The need for this type of repair could be eliminated by stipulating that the crossover points in the two parents must correspond to the same network node. Whether this stipulation is

included or not, however, some other issues are raised in the new solutions. Since parts of a complete solution are eliminated, the new solution may not cover the entire map, or it may lead some crews to travel on lower priority roads before the higher priority roads are cleared. More repair operators would be needed to handle these problems. Additionally, since the traversal time of an arc varies depending on whether it has already been cleared, and the "cleared" status of an arc when a crew traverses it is so intricately woven into the rest of the solution, calculating the quality (time) of a new solution would require running a simulation of the entire solution.

Most of these concerns can be alleviated by dividing the solution into priority levels, and letting each gene represent the set of paths taken by all the crews in the course of clearing one priority level. The completion of a priority level represents a specific benchmark in progress, so the roads that have been cleared after completing a given priority level in one solution are the exact same roads that have been cleared after completing that same priority level in a different solution. Thus, in crossover, the only repair operator necessary is the one illustrated in Figure 2.5, which provides a sub-path to connect two other paths. It must be implemented for each crew, but since the repair sub-paths can consist entirely of cleared roads, the shortest paths can be determined by Dijkstra's algorithm.

Dijkstra's algorithm [20] is described below. The "time" of a given node refers to the time it takes to travel from the starting node to the given node, assuming all the roads in use have been cleared. Nodes for which a time has not been determined are considered unsolved. Figure 2.6 provides a flow chart representation.

1. All nodes are initially marked as unsolved except the starting node, which is marked as solved with a time of 0.

2. For all unsolved nodes that are adjacent to a solved node, calculate the "candidate time," which is the time to its adjacent solved node plus the extra time to get to the unsolved node. If an unsolved node is adjacent to more than one solved node, the smallest candidate time is used.
3. Choose the node with the smallest candidate time and mark it as solved with that time. Keep track of the corresponding path to that node from the initial node.
4. Repeat steps 2 and 3 until the destination node has been solved.

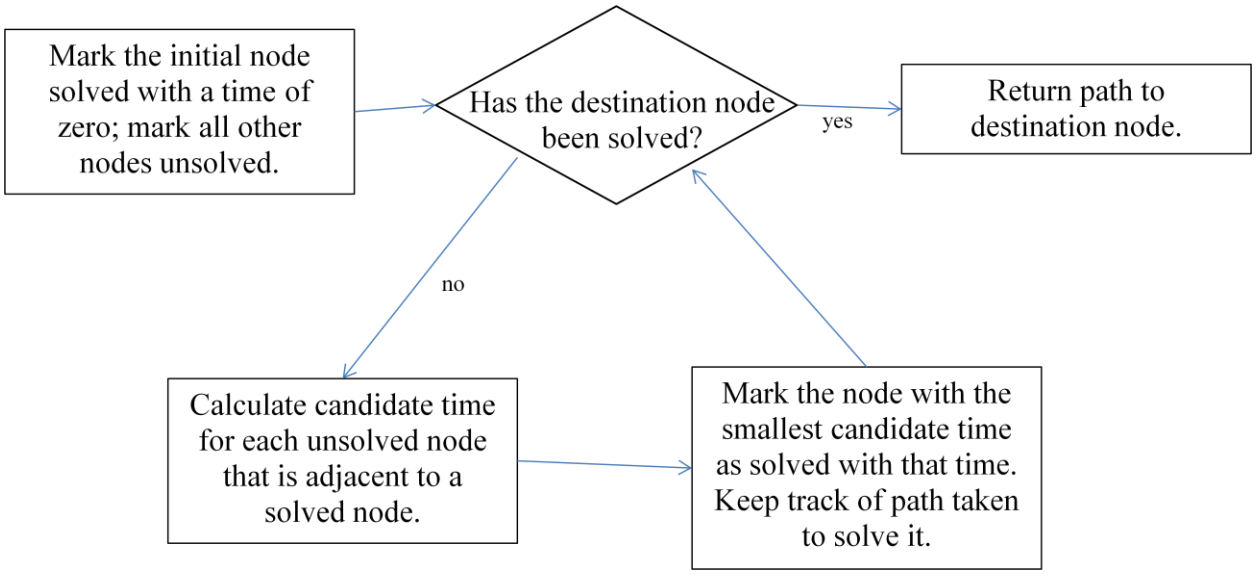

Figure 2.6: Flow chart representing Dijkstra's shortest-path algorithm adapted for minimizing time.

A solution in this representation is thus made up of priority levels, which represent the genes, and "transfers." A transfer is a set of sub-paths (one for each crew) that takes the crews from their respective locations upon finishing one priority level to their locations specified at the

beginning of the next. If *n* refers to the number of crews, then there are up to *n!* possible transfers between any two priority levels. Since the commencement of work on the next priority level assumes all crews are in their proper starting place, a transfer is measured by the *longest* (in terms of time) of the paths of the individual crews. The fastest transfer according to this evaluation method is chosen. It is noteworthy that given a set of starting and ending points, this algorithm will always determine the fastest possible transfer.

Since no clearing progress is made in the transfers, they are not included in the evolutionary process, except in the *evaluation* of a solution. Crossover produces a new combination of genes, which are then connected by generating the appropriate transfers. The time to clear each priority level is added together with the time to perform each transfer to yield the total time for the solution.

Mutation is thus performed in this representation by changing one priority level. First, an auxiliary solution is generated up to the beginning of the priority level in question. That priority level can then be generated any number of times before the best version of it is inserted into the selected solution as a mutation. Taking the best of several repetitions increases the likelihood that the mutated gene will be of high quality. In all the tests run in this experiment, five candidates are generated, and the best of these replaces the gene chosen for mutation, even if the replacement is worse. The mutated genes are created according to the same heuristic being used for the rest of the solution.

This representation also creates an option for a type of offshoot of elitism. It is likely that the best times for individual levels come from different members of the population (after all, that is the purpose of crossover). One way to generate elite solutions is to take the best of each level from the whole population (i.e., the best gene in each respective position) and put them together

to create a solution. There is a good chance that this solution will be the best generated by the whole process, but it is also certainly possible that the transfers required to make this solution work are quite long, inflating the overall time required.

2.2 Case study layout

The Fort Stewart road network is represented in this problem in terms of arcs and nodes, where each arc is a segment of road and each node is either an intersection or a dead-end. The network consists of 808 nodes and 1,126 arcs. In total, these 1,126 arcs cover 4,567,656 feet of road. The breakdown of roads by priority level is shown in Table 2.1, along with the percentage of the total road length contained in each priority level. Figure 2.7 also illustrates the road network of Fort Stewart, divided into the different priority levels. Road lengths are rounded to the nearest foot, and percentages are rounded to the nearest 0.1%. As is evident from the table, priority level 7 is by far the most substantial, while priority level 6 is comparatively almost non-existent. However, all priority levels provide ample opportunities to *waste* time due to poor planning, so they all must be considered important.

Table 2.1: Characteristics of each of the nine priority levels of the road network.

Priority level	Number of road segments (arcs)	Length of road (feet)	Percentage of total length
1	104	298,409	6.5
2	62	189,019	4.1
3	38	144,214	3.2
4	37	131,502	2.9
5	77	243,216	5.3
6	1	5,044	0.1
7	666	2,970,654	65.0
8	29	138,906	3.0
9	112	446,692	9.8

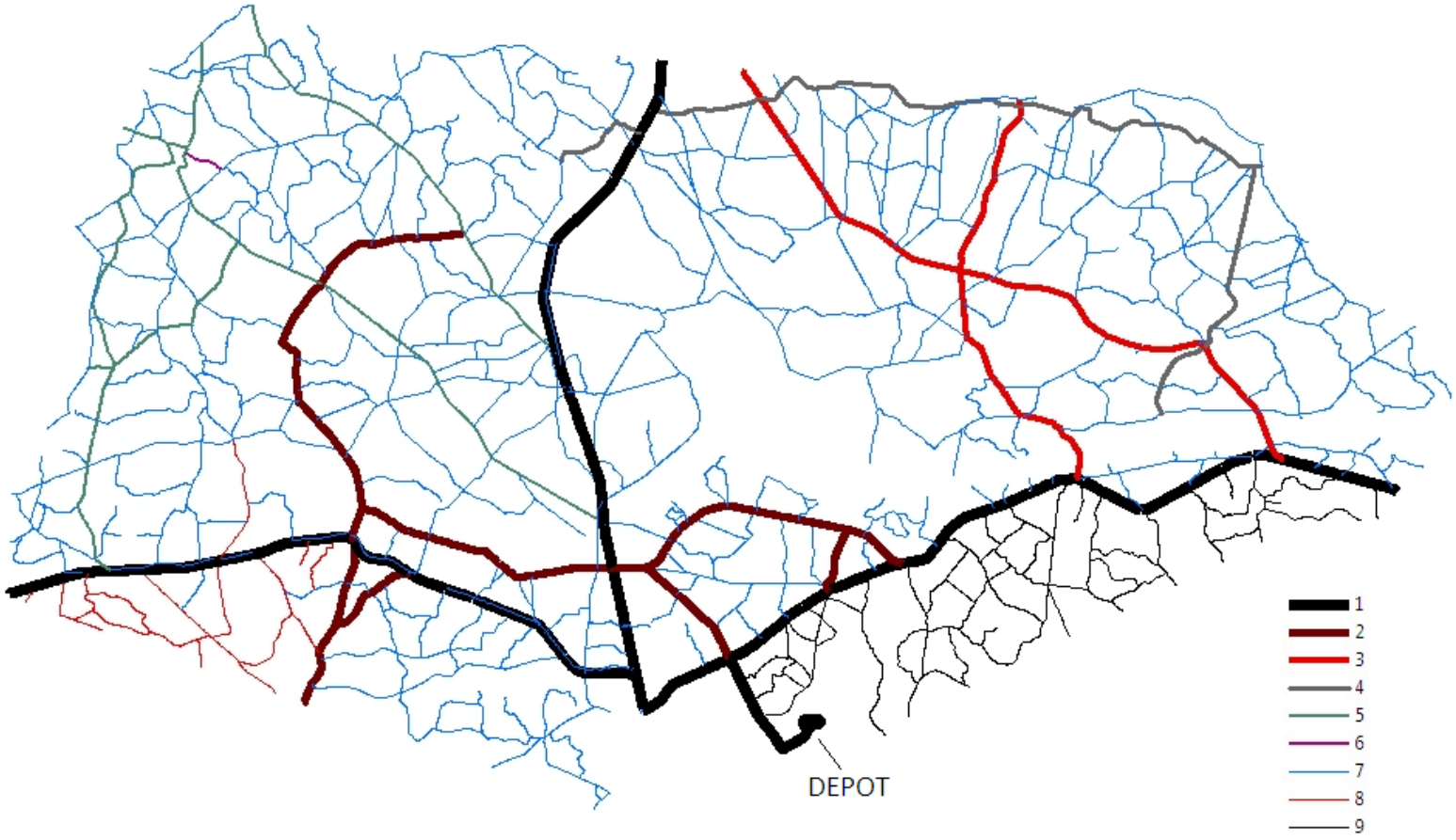

Figure 2.7: Road network at Fort Stewart divided into priority levels.

A theoretical lower limit clearing time can be determined by assuming that each crew spends the entire time clearing, with no traversing of cleared roads and never having more than two crews on the same arc. The theoretical lower limit would thus be the total number of feet of road divided by the product of the number of crews and the clearing speed of each crew. For this particular network, such a limit could never be attainable due at the very least to dead-ends, loops, and bottlenecks such as the one depicted in Figure 1.1.

A single crew was assumed to be able to clear roads at a rate of half a foot per second. Thus, two crews working together could clear at one foot per second. Under post-hurricane conditions, on cleared roads, the crews were assumed to be able to travel 44 feet per second (30 mph) on state highways and 22 feet per second (15 mph) on other roads. State highways constitute most of priority level 1, and are also mostly limited to that priority level. Figure 2.8 shows the road network divided into state highways and other roads.

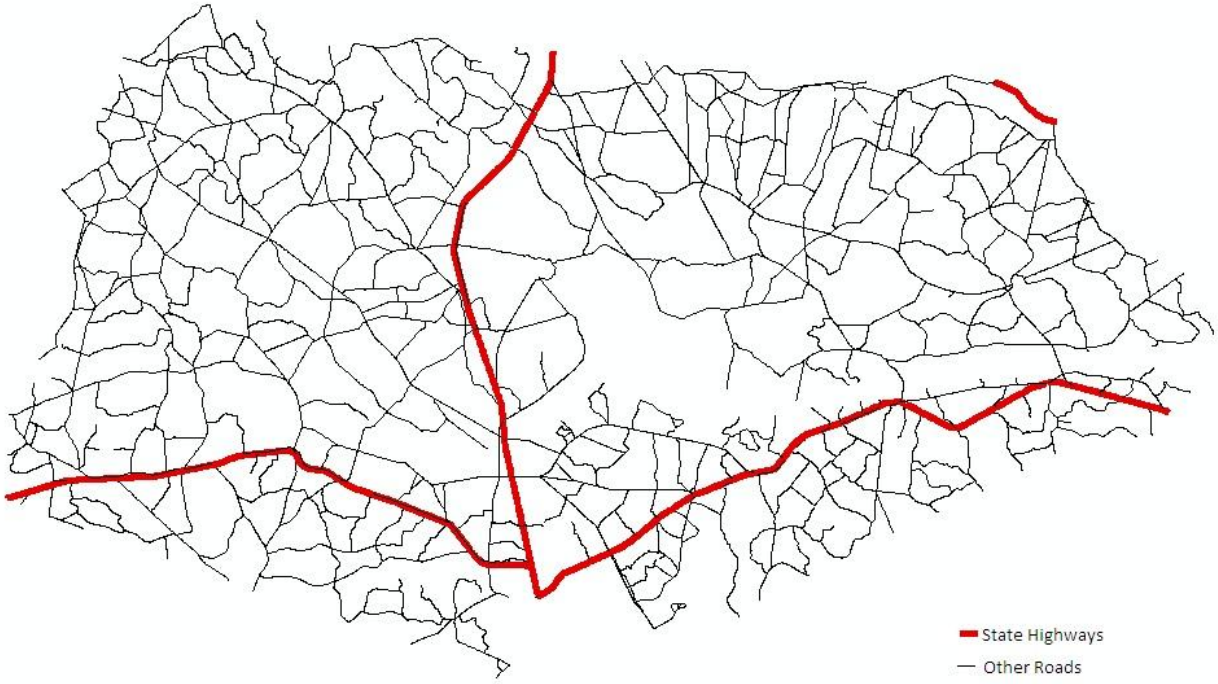

Figure 2.8: Fort Stewart road network with state highways marked.

2.3 Experiment

In running a simulation, progress could only be made in discrete steps, which were time-based. In one time step, each crew would travel along whatever arc it was on, clearing that arc if necessary. Its speed was determined by considering (a) whether it was clearing debris, (b) whether there was another crew helping it, and (c) whether the road was a state highway. For the sake of simplicity of operation, when a crew reached the end of an arc during a time step, it would have to wait until the next time step to commence traveling on the next arc. The amount of time allocated to each time step was treated as a variable in the experiments.

All of the tests that were run were evaluated according to the quality of the best solution found, the average quality of solution found, and the computation required, which was measured in both time and a quantity called *crew-steps*. Each crew-step represented the computation required to handle one crew in one time step.

The algorithms were written in the C# programming language. The experiments were run on three different computers. CPU1 used an Intel® Pentium® 4, 2.60 GHz processor with 2.00 GB of RAM. CPU2 used an Intel® Xeon™, 3.60 GHz processor with 10.00 GB of RAM. CPU3 used an Intel® Xeon®, 2.66 GHz processor with 8.00 GB of RAM.

The first test compared the rule-based heuristics. The number of crews was variable, as was the size of the time step. Table 2.2 shows the values tested for each variable. Each possible combination of variables (i.e., each of the 18 elements in the Cartesian product of the three columns) was tested 20 times using CPU1.

Table 2.2: Variables in implementing rule-based heuristics. Each possible combination of one variable from each column was used to run 20 simulations to compare the rule-based heuristics.

Rule set	Number of crews	Size of time step (seconds)
1	3	10
2	5	25
	8	100

The next test integrated a basic implementation of level-based beam search with the rule-based methods. The filter size in these tests was held at 1, while the exploration factor was adjusted. In fact, the previous test could be seen as a special case of this test with the exploration factor set to 1. Table 2.3 shows the values tested for each variable. Each possible combination of variables from the four columns was tested 50 times using CPU1.

Table 2.3: Variables in implementing level-based beam search. Each combination of variables was used to run 50 trials to compare the rule-based heuristics in conjunction with level-based beam search with a filter size of 1.

Rule set	Exploration factor	Number of crews	Size of time step (seconds)
1	3	3	10
2	10	5	25
		8	100

The two types of adapted local beam search were then tested, with each making use of both rule-based heuristics as well as the random-choice method. With new variables being introduced, the number of crews in these tests was held constant at 5, and the time step was held constant at 25 seconds. The exploration factor was also held constant at 20. In the constant time

beam search, the variables were the rule set, the time threshold, and the filter size. Table 2.4 shows the values tested for each variable, and each possible combination consisting of one variable from each column was tested 20 times. Experiments using rule set 1 and the random-choice method were run on CPU2, with the exception of the combination of the random-choice rule set, 5,000 second time threshold and filter size 2, which was run on CPU3. Experiments using rule set 2 were run on CPU1. In the constant progress beam search, the variables were the rule set, the progress threshold, and the filter size. Table 2.5 shows the values tested for each variable, and each possible combination consisting of one variable from each column was tested 20 times. Experiments using rule set 1 were run on CPU3, while experiments using rule set 2 and the random-choice method were run on CPU1. Two exceptions are the combinations involving the random-choice method, a progress threshold of 8,000 feet cleared, and filter sizes of 5 and 2, which were run on CPU3.

Table 2.4: Variables in implementing constant time beam search. Each possible combination of was used to run 20 trials. The number of crews was set to 5, the time step was set to 25 seconds, and the exploration factor was set to 20.

Rule set	Time threshold (seconds)	Filter size
1	5,000	10
2	10,000	5
Random-choice	25,000	2
	250,000	

Table 2.5: Variables in implementing constant progress beam search. Each possible combination of variables was used to run 20 trials. The number of crews was set to 5, the time step was set to 25 seconds, and the exploration factor was set to 20.

Rule set	Progress threshold (feet cleared)	Filter size
1	8,000	10
2	20,000	5
Random-choice	50,000	2
	250,000	

The genetic algorithm also introduces more variables, so some of the parameters that were varied in previous tests will be held constant in testing the genetic algorithm. The populations, each consisting of 30 solutions, were created using level-based beam search in conjunction with the first rule-based heuristic. The number of crews was set to 5, with an exploration factor of 3, a filter size of 1, and a time step of 25 seconds. Two-point crossover was used, and the crossover probability was held constant at 0.8. Each run was limited to 50 generations, but would terminate if 15 consecutive generations failed to improve on the best solution found. In each generation, two "elite" solutions were created by using the best and second-best, respectively, of each individual priority level. The selection method and mutation probability were varied, and their values/types are shown in Table 2.6. Each possible combination of values was tested 5 times using CPU1.

The last experiment tested whether the genetic algorithm could be used to improve upon the solutions developed by the local beam search adaptations. The initial population of each run of the genetic algorithm consisted of 30 solutions, which were developed using constant time beam search in conjunction with rule-set 2. The number of crews was held constant at 5, and the

Table 2.6: Variables in implementing the genetic algorithm. Each possible combination of variables was used to run 5 trials. Each run assumed 5 crews, two-point crossover with a probability of 0.8, a convergence criterion of 15 consecutive generations without improvement, and a population of 30 solutions created from level-based beam search in conjunction with rule-set 1, an exploration factor of 3, a filter size of 1, and a time-step of 25 seconds.

Selection method	Mutation probability
Ranked roulette	0.01
Tournament (size 2)	0.2

constant time beam search used a time step of 25 seconds, a time-threshold of 25,000 seconds, an exploration factor of 20, and a filter size of 2. The genetic algorithm used ranked roulette selection, two-point crossover with a probability of 0.8, and a mutation probability of 0.2. The convergence criterion was again set to 15 consecutive generations without improvement, but the number of generations was not capped. Two "elite" solutions were created in each generation in the same way as described above. The experiment was run on CPU2.

CHAPTER 3

RESULTS

Table 3.1 shows the results from using rule method 1 while varying the number of crews and the size of the time step. Recall that the time step is the amount of simulated time that passes each time the state is updated. Each combination of parameters was tested 20 times on CPU1. This method can be thought of as level-based beam search with the exploration factor set to 1. For each given number of crews, the differences in solutions generated by using the three different time steps were not statistically significant.

Table 3.2 shows the results from using rule method 2 while varying the number of crews and the size of the time step. Each combination of parameters was tested 20 times on CPU1. With three crews, the difference in average solution from using time steps of 10 seconds and 25 seconds are not statistically significant, but the differences between these and the solutions found from using a time step of 100 seconds are statistically significant ($p = 0.05$). With five crews, the solutions from using a time step of 10 seconds are not significantly different than those from using a time step of either 25 seconds or 100 seconds, but the results from those two experiments are statistically significantly different from each other. With eight crews, none of the three experiments produced solutions that were statistically significantly different from any of the others.

When comparing rule method 1 to rule method 2 based on each given parameter combination, the differences in solutions are statistically significant for all parameter

combinations involving three or five crews ($p = 0.05$). However, the differences are not statistically significant for any of the combinations involving eight crews.

Table 3.3 shows the results from using rule method 1 in conjunction with level-based beam search while varying the number of crews, the exploration factor, and the size of the time step. The filter size was held constant at 1, and each parameter combination was run 50 times on CPU1. For all the solutions from using three crews and a time step of either 10 seconds or 25 seconds, the differences are not statistically significant ($p = 0.05$). The solutions found from the two experiments using three crews and a time step of 100 seconds are not significantly different from each other, but are significantly different from those using three crews and smaller time steps. Of the combinations involving five crews, the two that used a time step of 10 seconds both differed significantly from the one that used a time step of 100 seconds and an exploration factor of 10. Solutions found from the one using a time step of 10 seconds and an exploration factor of 10 also differed significantly from those found by the one using a time step of 100 seconds and an exploration factor of 3. Otherwise, none of the combinations involving five crews produced solutions statistically significantly different than any of the others ($p = 0.05$). All six combinations involving eight crews produced results that were not significantly different than any of the others.

Table 3.4 shows the results from using rule method 2 in conjunction with level-based beam search while varying the number of crews, the exploration factor, and the size of the time step. The filter size was held constant at 1, and each parameter combination was run 50 times on CPU1. Of the solutions found using three crews, those that used a time step of 100 seconds and an exploration factor of 10 were not statistically significantly different ($p = 0.05$) from those that used a time step of 10 seconds or 25 seconds and an exploration factor of 3. The combinations

using an exploration factor of 10 and time steps of 10 seconds and 25 seconds did not differ significantly from each other. Of the combinations involving five crews, none produced solutions that were statistically significantly different than any of the others ($p = 0.05$), with the exception of the one that used a time step of 100 seconds and an exploration factor of 3. Its solutions were significantly different from those found using all the other combinations except for a time step of 100 seconds and an exploration factor of 10. Of the combinations involving eight crews, the solutions from the two combinations that used a time step of 10 seconds and the one that used a time step of 10 seconds and an exploration factor of 10 were not statistically significant ($p = 0.05$). The other three combinations also produced results that were not statistically significantly different from each other.

In the results from level-based beam search, when comparing rule method 1 to rule method 2 with corresponding combinations of all the other parameters, most of the differences are statistically significantly different ($p = 0.05$). The exceptions are when three crews are used with an exploration factor of 10 (for all three time steps) and when eight crews are used with an exploration factor of 3 (for time steps of 10 seconds and 25 seconds).

Table 3.5 shows the results from using constant time beam search in conjunction with rule methods 1 and 2. The number of crews was set to 5, the time step was held constant at 25 seconds, and the exploration factor was set to 20. The time threshold and filter size were varied. Each parameter combination was tested 20 times, but different computers were used. Table 3.6 shows the computers used for each test as well as the data regarding time and crew steps. A comparison of the solutions found from constant time beam search in conjunction with either rule method determined that any differences between any solutions produced by different parameter combinations using a given rule method were not statistically significant ($p = 0.05$),

nor were any solutions produced by the different rule methods with a fixed parameter combination.

Table 3.7 shows the results from using constant time beam search in conjunction with the random-choice method. The number of crews was set to 5, the time step was held constant at 25 seconds, and the exploration factor was set to 20. The time threshold and filter size were varied. Each parameter combination was tested 20 times, but different computers were used. Table 3.8 shows the data regarding time and crew steps. Each of these tests was run on CPU2. Using any given parameter combination, the solutions found from using constant time beam search in conjunction with the random-choice method were significantly different than those found from using constant time beam search in conjunction with either rule method ($p = 0.05$). Further, the solutions found using combinations that included a time threshold of 250,000 seconds and the random-choice method were significantly different from those using smaller time thresholds, but not from each other.

Table 3.9 shows the results from using constant progress beam search in conjunction with rule methods 1 and 2. The number of crews was set to 5, the time step was held constant at 25 seconds, and the exploration factor was set to 20. The progress threshold and filter size were varied. Each parameter combination was tested 20 times, but different computers were used. Table 3.10 shows the computers used for each test as well as the data regarding time and crew steps. A comparison of the solutions found from constant progress beam search in conjunction with either rule method determined that any differences between any solutions produced by different parameter combinations using a given rule method were not statistically significant ($p = 0.05$), nor were any solutions produced by the different rule methods with a fixed parameter combination.

Table 3.11 shows the results from using constant progress beam search in conjunction with the random-choice method. The number of crews was set to 5, the time step was held constant at 25 seconds, and the exploration factor was set to 20. The progress threshold and filter size were varied. Each parameter combination was tested 20 times, but different computers were used. Table 3.12 shows the computers used for each test as well as the data regarding time and crew steps. All of the differences between solutions found using constant progress beam search with the random-choice method and those found using constant progress beam search with either rule method and corresponding parameter combinations were found to be statistically significantly different ($p = 0.05$). The solutions found with the random-choice method and progress thresholds of 8,000 feet or 20,000 feet were not significantly different from each other. Those found using the random-choice method and progress thresholds of 50,000 feet were not significantly different from each other, and those found using progress thresholds of 250,000 feet were not significantly different from each other.

Table 3.13 shows the results from using genetic algorithms whose populations (of size 30) were generated using level-based beam search in conjunction with rule method 1. The number of crews in each was set to 5, and the level-based beam search used an exploration factor of 3, a filter size of 1, and a time step of 25 seconds. The genetic algorithm employed two-point crossover with a crossover probability of 0.8. The number of generations was limited to 50, though the algorithm would also terminate after 15 consecutive generations without improvement. Two "elite" solutions were constructed in each generation, with the first consisting of the best gene in the population at each priority level, and the second consisting of the second-best gene in the population at each priority level. These solutions were passed unchanged to the next generation. The selection method and mutation probability were variables

in these experiments, and each combination of these variables was tested 5 times. The tests were run on CPU1. Table 3.14 shows the data regarding time and crew steps.

Table 15 shows the results from the five trials of the genetic algorithm with each initial population generated from constant time beam search in conjunction with rule method 2. Data from each individual trial are displayed in addition to summary statistics.

Table 3.1: Results from using rule method 1. Based on 20 trials per parameter combination.

Time step (s)	# of crews	Average solution (s)	Standard deviation of solutions (s)	Best solution (s)	Average run time (s)	Average number of crew steps	Standard deviation of crew steps
10	3	4,617,501	480,063	3,887,820	<5	1,385,250	144,019
	5	2,831,315	146,328	2,602,190	<5	1,415,658	73,164
	8	1,977,288	97,556	1,781,460	<5	1,581,830	78,045
25	3	4,647,610	398,826	4,024,175	<5	557,713	47,859
	5	2,886,934	171,787	2,629,300	<5	577,387	34,357
	8	1,983,801	119,601	1,760,750	<5	634,816	38,272
100	3	4,923,270	526,293	3,979,400	<5	147,698	15,789
	5	3,148,355	353,377	2,721,400	<5	157,418	17,669
	8	2,166,305	170,137	1,859,500	<5	173,304	13,611

Table 3.2: Results from using rule method 2. Based on 20 trials per parameter combination.

Time step (s)	# of crews	Average solution (s)	Standard deviation of solutions (s)	Best solution (s)	Average run time (s)	Average number of crew steps	Standard deviation of crew steps
10	3	5,258,382	808,591	4,457,330	<10	1,577,515	242,577
	5	3,385,911	509,019	2,742,080	<10	1,692,956	254,509
	8	2,106,146	296,460	1,797,440	<10	1,684,917	237,168
25	3	5,215,060	719,753	3,893,400	<10	625,807	86,370
	5	3,321,668	336,093	2,844,525	<5	664,334	67,219
	8	2,113,124	210,497	1,797,725	<5	676,200	67,359
100	3	6,707,120	832,337	4,914,100	<5	201,214	24,970
	5	3,756,160	552,981	3,167,100	<5	187,808	27,649
	8	2,408,025	319,819	1,915,200	<5	192,641	25,586

Table 3.3: Results from using rule method 1 in conjunction with level based beam search.

Based on 50 trials per parameter combination.

Time step (s)	# of crews	Exploration factor	Average solution (s)	St. dev. of solutions (s)	Best solution (s)	Average run time (s)	Average # of crew steps	St. dev. of crew steps
10	3	3	4,223,658	349,100	3,716,210	10	4,087,332	214,779
		10	4,418,399	470,262	3,636,070	36	13,745,045	391,699
	5	3	2,658,209	142,060	2,416,650	8	4,375,814	214,023
		10	2,540,345	197,609	2,343,050	29	14,566,652	365,034
	8	3	1,896,921	97,682	1,701,390	8	4,745,624	168,835
		10	1,831,264	53,283	1,718,980	28	15,864,677	528,065
25	3	3	4,368,644	398,690	3,836,300	7	1,665,727	95,848
		10	4,399,270	388,524	3,584,775	20	5,548,413	165,516
	5	3	2,687,074	176,879	2,458,200	6	1,777,150	101,804
		10	2,663,444	323,661	2,351,050	19	5,949,181	143,652
	8	3	1,899,610	106,124	1,702,125	6	1,919,905	93,548
		10	1,835,306	40,391	1,758,475	17	6,395,572	218,548
100	3	3	4,801,304	532,662	3,972,400	5	451,124	25,593
		10	4,751,722	510,802	3,844,300	15	1,482,782	48,704
	5	3	2,894,426	302,371	2,482,800	5	474,306	24,756
		10	2,930,802	411,817	2,482,300	14	1,584,744	52,319
	8	3	1,986,042	90,256	1,842,900	5	519,208	26,696
		10	1,898,090	54,512	1,800,200	14	1,727,880	64,829

Table 3.4: Results from using rule method 2 in conjunction with level based beam search.

Based on 50 trials per parameter combination.

Time step (s)	# of crews	Exploration factor	Average solution (s)	St. dev. of solutions (s)	Best solution (s)	Average run time (s)	Average # of crew steps	St. dev. of crew steps
10	3	3	4,737,307	420,531	3,795,850	14	4,630,995	512,089
		10	4,413,871	142,292	4,193,250	48	15,573,762	724,465
	5	3	3,215,361	572,323	2,504,960	13	4,846,117	437,749
		10	3,316,731	262,421	2,589,030	42	16,319,731	845,895
	8	3	2,144,775	440,612	1,686,110	12	5,133,404	418,606
		10	2,124,572	378,124	1,737,200	52	16,927,300	730,512
25	3	3	5,018,978	575,533	4,370,375	11	1,980,847	193,176
		10	4,443,611	149,650	4,161,650	32	6,735,208	406,464
	5	3	3,335,400	603,623	2,565,325	10	2,021,152	213,989
		10	3,310,740	214,182	3,104,425	29	6,756,745	324,137
	8	3	2,115,122	340,874	1,789,575	10	2,093,518	123,264
		10	2,445,863	452,898	1,789,400	31	7,108,601	329,779
100	3	3	5,353,086	490,937	4,690,700	8	581,959	57,018
		10	4,776,444	285,572	4,402,900	25	1,946,498	123,985
	5	3	3,677,938	486,471	2,669,700	7	553,230	45,677
		10	3,463,860	182,104	3,184,700	24	1,865,395	101,159
	8	3	2,469,070	548,448	1,864,800	10	583,393	50,623
		10	2,594,268	437,537	1,894,500	31	1,906,076	105,191

Table 3.5: Results from using constant time beam search combined with each rule method. Based on 20 trials per parameter combination. The number of crews was set to 5, the time-step was set to 25 seconds, and the exploration factor was set to 20.

Rule set	Filter size	Time threshold (s)	Average solution (s)	St. dev. of solutions (s)	Best solution (s)
1	10	5,000	2,152,179	26,278	2,082,425
	10	10,000	2,160,673	36,874	2,083,125
	10	25,000	2,230,170	33,585	2,185,075
	10	250,000	2,375,721	45,276	2,296,625
	5	5,000	2,177,849	52,397	2,111,975
	5	10,000	2,165,461	33,316	2,117,825
	5	25,000	2,201,094	30,885	2,138,025
	5	250,000	2,410,270	46,471	2,326,200
	2	5,000	2,176,278	48,992	2,081,625
	2	10,000	2,190,018	32,781	2,127,400
	2	25,000	2,194,614	30,745	2,145,075
	2	250,000	2,469,931	60,692	2,327,075
2	10	5,000	2,122,199	35,847	2,054,050
	10	10,000	2,142,664	34,149	2,090,425
	10	25,000	2,224,960	38,403	2,174,525
	10	250,000	2,422,214	53,309	2,345,975
	5	5,000	2,131,370	39,405	2,073,600
	5	10,000	2,130,249	30,341	2,084,875
	5	25,000	2,181,531	36,069	2,134,625
	5	250,000	2,436,770	54,025	2,336,875
	2	5,000	2,131,694	26,503	2,085,075
	2	10,000	2,137,634	31,102	2,073,250
	2	25,000	2,176,515	33,627	2,131,625
	2	250,000	2,529,059	79,443	2,388,500

Table 3.6: Computational demand from using constant time beam search in conjunction with each rule method. Based on 20 trials per parameter combination. The number of crews was set to 5, the time-step was set to 25 seconds, and the exploration factor was set to 20.

Rule set	Filter size	Time threshold (s)	CPU	Average runtime (min)	Average # of crew steps	St. dev. of crew steps
1	10	5,000	2	27	8,682,253	105,783
	10	10,000	2	15	8,795,858	149,364
	10	25,000	2	7	9,305,253	139,171
	10	250,000	2	3	10,955,791	109,867
	5	5,000	2	39	8,740,709	211,487
	5	10,000	2	22	8,738,162	122,641
	5	25,000	2	11	9,018,073	129,506
	5	250,000	2	3	10,839,950	142,771
	2	5,000	2	48	8,716,396	196,798
	2	10,000	2	18	8,793,514	133,663
	2	25,000	2	11	8,882,171	127,866
	2	250,000	2	3	10,703,584	171,111
2	10	5,000	1	11	8,555,500	141,445
	10	10,000	1	6	8,748,515	129,481
	10	25,000	1	3	9,316,057	157,002
	10	250,000	1	1	11,645,280	224,182
	5	5,000	1	17	8,553,788	159,475
	5	10,000	1	8	8,594,993	123,647
	5	25,000	1	4	8,966,542	158,685
	5	250,000	1	1	11,084,270	135,383
	2	5,000	1	19	8,541,527	103,782
	2	10,000	1	9	8,591,981	123,550
	2	25,000	1	4	8,838,490	143,454
	2	250,000	1	2	11,012,883	237,459

Table 3.7: Results from using constant time beam search in conjunction with the random-choice method. Data summaries are based on 20 trials per parameter combination. The number of crews was set to 5, the time-step was set to 25 seconds, and the exploration factor was set to 20.

Filter size	Time threshold (s)	Average solution (s)	St. dev. of solutions (s)	Best solution (s)
10	5,000	2,805,446	124,052	2,613,375
10	10,000	2,942,794	83,076	2,773,375
10	25,000	3,161,538	69,681	3,036,575
10	250,000	3,829,554	173,139	3,521,625
5	5,000	2,790,815	118,654	2,569,625
5	10,000	2,855,869	115,785	2,701,475
5	25,000	3,023,398	66,612	2,888,450
5	250,000	3,787,784	132,109	3,543,475
2	5,000	2,893,381	187,639	2,604,025
2	10,000	2,853,494	116,493	2,634,550
2	25,000	3,003,114	104,060	2,812,525
2	250,000	3,878,983	185,567	3,542,950

Table 3.8: Computational demand from using constant time beam search combined with the random-choice method. Data summaries are based on 20 trials per parameter combination. The number of crews was set to 5, the time-step was set to 25 seconds, and the exploration factor was set to 20.

Filter size	Time threshold (s)	Average runtime (min)	Average # of crew steps	St. dev. of crew steps
10	5,000	34	11,379,659	494,042
10	10,000	20	12,106,524	336,980
10	25,000	10	13,461,895	309,989
10	250,000	4	20,188,753	639,769
5	5,000	49	11,219,546	475,257
5	10,000	25	11,574,149	463,870
5	25,000	13	12,540,603	287,925
5	250,000	4	18,666,686	490,992
2	5,000	14	11,599,072	751,142
2	10,000	32	11,481,813	467,363
2	25,000	15	12,261,513	427,233
2	250,000	5	17,541,932	730,368

Table 3.9: Results from using constant progress beam search in conjunction with each rule method. Data summaries are based on 20 trials per parameter combination. The number of crews was set to 5, the time-step was set to 25 seconds, and the exploration factor was set to 20.

Rule set	Filter size	Progress threshold (ft)	Average solution (s)	St. dev. of solutions (s)	Best solution (s)
1	10	8,000	2,177,008	31,807	2,135,650
	10	20,000	2,197,910	24,541	2,147,475
	10	50,000	2,262,259	32,873	2,205,775
	10	250,000	2,349,321	47,769	2,268,425
	5	8,000	2,174,166	55,242	2,102,050
	5	20,000	2,182,175	20,137	2,137,450
	5	50,000	2,213,440	40,937	2,148,625
	5	250,000	2,358,744	48,645	2,287,300
	2	8,000	2,162,754	35,956	2,093,575
	2	20,000	2,166,685	29,922	2,114,050
	2	50,000	2,219,840	33,835	2,160,600
	2	250,000	2,338,019	31,118	2,280,900
2	10	8,000	2,143,019	53,166	2,087,625
	10	20,000	2,161,286	27,992	2,098,400
	10	50,000	2,220,895	35,619	2,166,350
	10	250,000	2,371,960	56,079	2,285,425
	5	8,000	2,138,854	40,454	2,054,400
	5	20,000	2,150,559	35,961	2,072,700
	5	50,000	2,184,671	34,790	2,131,775
	5	250,000	2,330,259	50,723	2,252,950
	2	8,000	2,153,744	39,196	2,094,575
	2	20,000	2,176,575	44,291	2,078,700
	2	50,000	2,191,504	22,002	2,156,825
	2	250,000	2,326,290	71,122	2,224,125

Table 3.10: Computational demand from using constant progress beam search combined with each rule method. Based on 20 trials per parameter combination. The number of crews was set to 5, the time-step was set to 25 seconds, and the exploration factor was set to 20.

Rule set	Filter size	Progress threshold (ft)	CPU	Average runtime (min)	Average # of crew steps	St. dev. of crew steps
1	10	8,000	3	4	10,477,275	267,786
	10	20,000	3	2	10,971,533	252,599
	10	50,000	3	1	11,456,088	246,086
	10	250,000	3	<1	11,913,588	306,685
	5	8,000	3	7	10,180,739	404,542
	5	20,000	3	3	10,882,165	278,555
	5	50,000	3	<1	11,128,743	372,091
	5	250,000	3	<1	11,977,559	305,350
	2	8,000	3	8	9,742,283	370,433
	2	20,000	3	4	10,447,563	310,927
	2	50,000	3	1	11,025,659	281,872
	2	250,000	3	<1	11,892,460	275,618
2	10	8,000	1	14	11,124,889	807,238
	10	20,000	1	7	11,539,684	563,865
	10	50,000	1	4	12,199,614	566,662
	10	250,000	1	2	13,057,073	448,327
	5	8,000	1	21	10,574,935	817,322
	5	20,000	1	10	11,002,044	439,823
	5	50,000	1	5	11,823,058	702,745
	5	250,000	1	2	13,126,021	457,687
	2	8,000	1	25	10,188,894	655,950
	2	20,000	1	11	11,038,479	749,431
	2	50,000	1	6	11,681,186	502,235
	2	250,000	1	2	13,196,334	730,473

Table 3.11: Results from using constant progress beam search combined with the random-choice method. Data summaries are based on 20 trials per parameter combination. The number of crews was set to 5, the time-step was set to 25 seconds, and the exploration factor was set to 20.

Filter size	Progress threshold (ft)	Average solution (s)	St. dev. Of solutions (s)	Best solution (s)
10	8,000	2,921,944	125,833	2,747,125
10	20,000	3,132,581	84,849	2,980,200
10	50,000	3,400,223	98,807	3,218,200
10	250,000	3,881,554	164,765	3,634,525
5	8,000	2,822,818	123,997	2,644,775
5	20,000	3,018,069	131,022	2,871,950
5	50,000	3,267,364	119,096	3,063,725
5	250,000	3,843,765	183,498	3,526,725
2	8,000	2,823,255	132,029	2,557,475
2	20,000	2,954,930	124,748	2,721,800
2	50,000	3,264,565	104,757	3,135,300
2	250,000	3,794,609	134,550	3,566,625

Table 3.12: Computational demand from using constant progress beam search combined with the random-choice method. Data summaries are based on 20 trials per parameter combination. The number of crews was set to 5, the time-step was set to 25 seconds, and the exploration factor was set to 20.

Filter size	Progress threshold (ft)	CPU	Average runtime (min)	Average # of crew steps	St. dev. of crew steps
10	8,000	1	15	21,501,879	1,357,692
10	20,000	1	7	24,058,856	1,260,759
10	50,000	1	4	25,853,471	738,374
10	250,000	1	2	27,206,019	882,159
5	8,000	3	7	19,925,818	1,645,200
5	20,000	1	10	23,032,021	1,214,480
5	50,000	1	5	25,098,046	1,143,694
5	250,000	1	3	27,260,201	1,197,047
2	8,000	3	8	19,909,736	2,152,342
2	20,000	1	11	22,300,013	1,698,992
2	50,000	1	6	24,832,531	1,367,439
2	250,000	1	3	27,299,862	1,361,670

Table 3.13: Results from running the genetic algorithm five times with each combination of variables, using level based beam search to generate the initial population.

Selection method	Mutation probability	Average best solution (s)	St. dev. of best solution (s)	Average generation finding best solution	St. dev. of generation finding best solution
Ranked roulette	0.2	2,235,308	21,422	36.6	19.1
	0.01	2,272,907	22,760	2.2	2.2
Tournament (size 2)	0.2	2,230,524	16,739	34.4	17.2
	0.01	2,310,353	34,284	3.2	2.9

Table 3.14: Computational demand from running the genetic algorithm with the initial population generated using level-based beam search. Data summaries are based on five trials per parameter combination.

Selection method	Mutation probability	Average number of crew steps	St. dev. of number of crew steps	Average runtime (h)
Ranked roulette	0.2	180,284,640	37,588,791	4.94
	0.01	55,765,522	1,431,437	1.58
Tournament (size 2)	0.2	177,830,344	38,439,168	4.79
	0.01	56,939,024	2,635,052	1.78

Table 3.15: Results from running the genetic algorithm with the initial population generated using constant time beam search combined with rule method 2.

Trial	Best solution (s)	Generation finding best solution	Number of crew steps	Runtime (h)
1	2,017,988	139	4,727,301,185	22
2	2,025,208	76	2,939,538,120	14
3	2,043,346	38	1,664,340,780	8
4	2,028,271	87	3,270,284,440	15
5	2,036,025	29	1,681,842,845	7
Average	2,030,168	74	2,856,661,474	13
Standard Deviation	9,805	44	1,272,701,150	6

CHAPTER 4

DISCUSSION

The problem addressed in these experiments was very complex, so a number of assumptions were made for simplification. Some of these assumptions will be revisited in this section, and the adaptability of the algorithms to scenarios where these assumptions do not hold will be addressed. An analysis of results will determine the comparative effectiveness of the heuristics tested. Positive and negative aspects of the heuristics will be discussed, along with possible future directions of the problem.

4.1 Revisiting assumptions

Many of the assumptions were simply made to specify the problem to be solved. One such assumption was that the amount of tree-related debris falling onto roads was constant. In real life, an even distribution is unlikely. The algorithms developed, however, do not depend on this assumption. A quick aerial survey could determine the demand distribution, and the status of each road can be set before running the algorithm. Any roads that are unaffected by the winds can be marked "cleared" and set to priority level 1, so they can be used at any time. The rate at which crews clear the roads can also be set to depend on the road without affecting the operation of the algorithm, so roads with light coverage can be assumed to take less time.

In addition to depending on the demand of the particular road, the rate at which a crew clears debris can be dependent on the crew. The assumption that all crews work at the same pace is not necessary for the algorithm to function, nor is the assumed pace of half a foot per second,

so each crew in the simulation can be assigned different capabilities. A related assumption is that a maximum of two crews can work effectively on the same road. This number can change, and a formula can also be developed to accommodate the idea of multiple crews with different levels of capability working on the same road. All these changes can be smoothly incorporated into the algorithms.

The speeds at which the crews travel on cleared roads can also be much more variable than what was assumed in this problem. The assumption was that all crews could travel 30 miles per hour on state highways and 15 miles per hour on other roads, once they have been cleared. As with the clearing speeds, these rates can be adjusted for each individual road and also be dependent on the crew. Since the road conditions can be expected to improve in the days after the hurricane, these assumed speeds can even change over the course of the operation.

The assumption that all crews leave initially from a single depot is also not crucial to the operation of the algorithm. The crews can start from anywhere, whether they are in scattered locations or clustered together. However, due to protocols developed by the Department of Defense, the initial staging area needs to be clear of surrounding hazards, thus the initial staging area at Fort Stewart is an airfield. Certain initial placements may cause conflicts with the priority constraints, though, which could be handled by assigning top priority to avenues that connect a crew's initial location to the priority 1 network.

Additionally, all the crews might not be available immediately or throughout the entirety of the process, as was assumed. In a real case of hurricane damage, most people will be allowed attend to their immediate personal issues first before attending to road management issues. This may cause uneven crew availability throughout the response and recovery stages of a hurricane. Crews can be added or subtracted to the fleet without affecting the algorithm. However, the

times and locations at which they are added or subtracted would have to be assumed prior to developing a solution.

There are also some assumptions that, if disregarded, might require large changes to the algorithm. One is that all crews work constantly until the entire network is clear. In reality, breaks will be necessary, and they may involve returning to the depot or to some other location, such as a barracks. Certainly, interruptions in the availability of fuel and supplies may cause interruptions in debris-clearing progress. These types of issues are difficult to predict in a chaotic environment. However, in order to develop solutions to this problem, the algorithm would have to be adjusted to include these necessary extra routes at the appropriate times.

The hard priority constraint greatly simplified the problem, but in a real-life scenario, strictly following the constraint would likely hinder the debris cleanup effort. As a priority level nears completion, it can often be the case that not all the crews can clear debris, since there may not be enough accessible roads needing to be cleared. At this point, the excess crews should start on the next priority level, even though the current one is not quite finished. There may also be cases where clearing a low-priority road opens up an avenue that enables the crews to more quickly clear the high priority roads. However, allowing for these exceptions opens up a drastically larger search space, making it harder to find a good solution.

4.2 Analysis of results

These experiments considered three real-life scenarios involving clearing debris from the roads at Fort Stewart: one made use of three crews, one made use of five crews, and one made use of eight crews. A theoretical lower bound of the clearing time can be calculated for each scenario by assuming all crews are constantly clearing roads, without ever driving back over any

roads that have already been cleared or having more crews in one place than can effectively contribute to the job. This lower bound is the result of dividing the total amount of road by the product of the number of crews and the clearing speed of each crew. The theoretical lower limits of each scenario were calculated and are shown in Table 4.1. Solutions can be scaled down by dividing by the corresponding theoretical limit, producing an *inefficiency index*. This index will allow for easier comparison of solutions.

Table 4.1: Theoretical lower limits for three, five, and eight crews. These are based on the assumption that all crews constantly clear debris until all roads have been cleared.

Number of crews	Theoretical lower limit (s)
3	3,045,104
5	1,827,062
8	1,141,914

For obvious reasons, increasing the number of crews decreases the amount of time required to clear the network. Generally, however, an increase in the number of crews led to an increase in the inefficiency index. Figures 4.1 and 4.2 show the trends produced by rule methods 1 and 2, respectively, when varying the number of crews. The figures show the average inefficiency indices produced by the various parameter combinations of each rule method operating as the sole heuristic or in conjunction with level-based beam search.

The decrease in efficiency with more crews likely has mostly to do with the limit on the number of crews that can work effectively in one place. In a bottleneck like the one shown in Figure 1.1, any number of crews beyond two is rendered useless until the network branches out. Thus, a higher number of crews leads to a higher percentage of unutilized workers. Similar

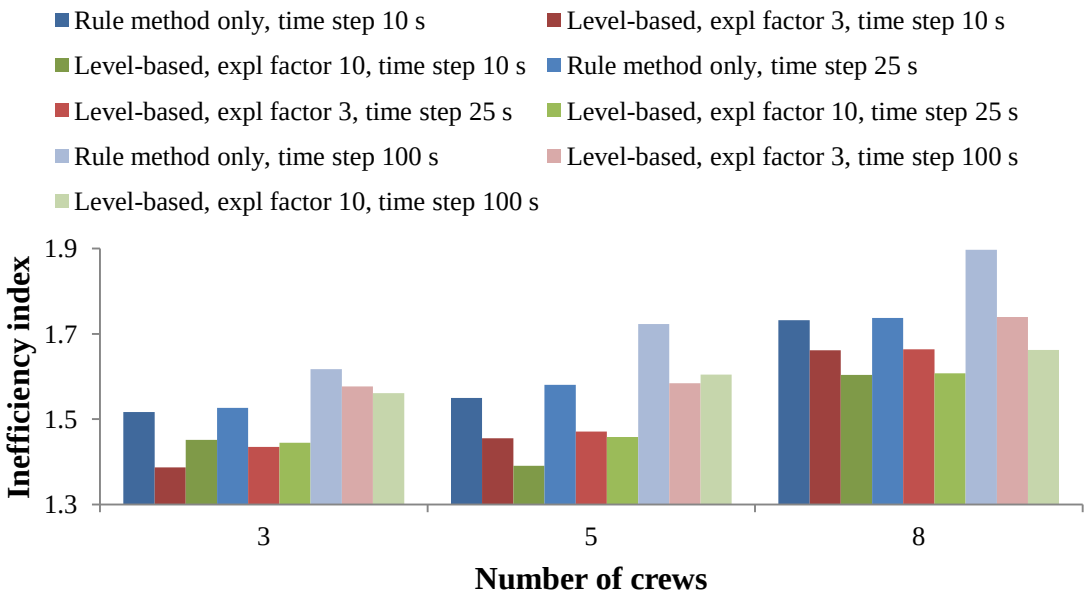

Figure 4.1: Effect of number of crews on inefficiency index when using rule method 1.

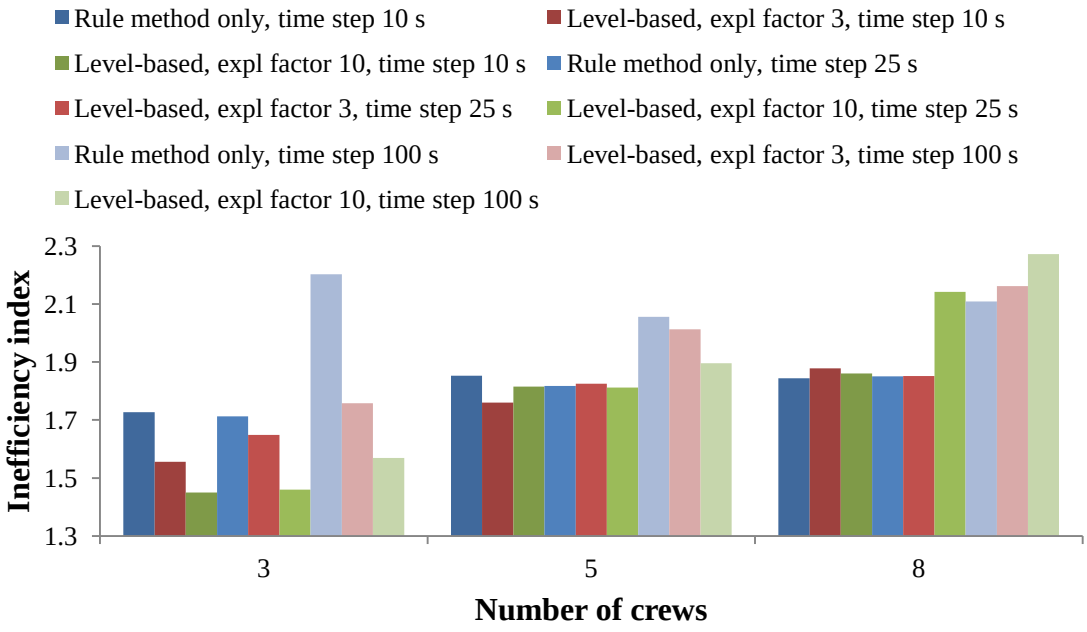

Figure 4.2: Effect of number of crews on inefficiency index when using rule method 2.

situations can arise when a priority level is nearing completion, as there may be more crews than available places for them to work.

Whether the rule-based methods performed in isolation or in conjunction with level-based beam search, rule method 1 generally resulted in better solutions than rule method 2. Figure 4.3 shows the average inefficiency index generated by each set of experiments across all the parameter combinations tested. These results were surprising because rule method 2 was intended to eliminate some inefficient behaviors generated by rule method 1.

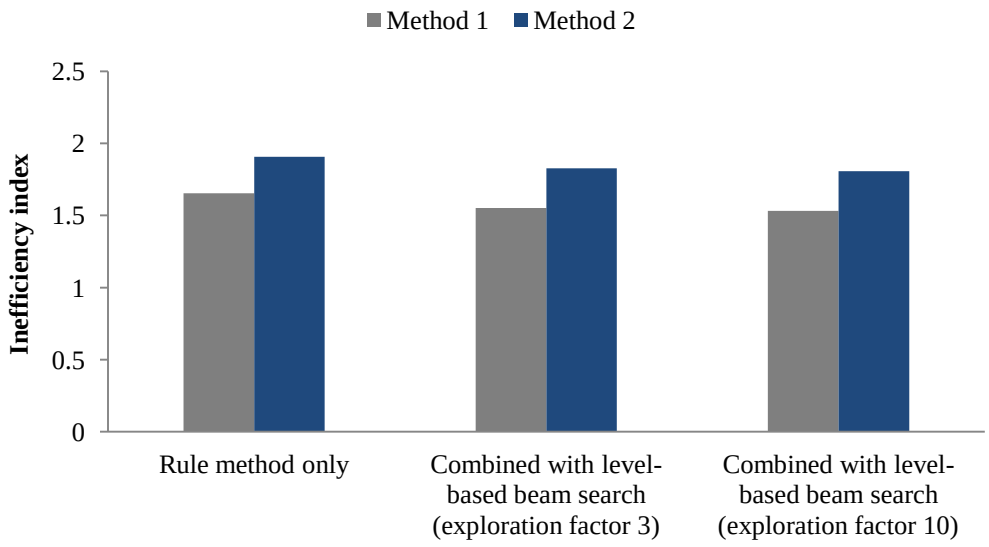

Figure 4.3: Comparison of rule methods 1 and 2.

To allow time to test other parameter variations, subsequent heuristics were only tested using a time step of 25 seconds and an assumption of 5 crews. Constant time beam search and constant progress beam search both produced drastically better results than the previous experiments. Using rule method 1 in conjunction with level-based beam search with an

exploration factor of 3, the average solution with 5 crews and a 25-second time step had an inefficiency index of 1.47. Increasing the exploration factor to 10 slightly improved the results, producing an average index of 1.46. The average solution using rule method 1 and constant time beam search (across all parameter combinations) had an inefficiency index of 1.23. When only considering solutions produced using a time threshold of 25,000 seconds or less, the index drops to 1.19. When using rule method 2, this number sinks to 1.18. Constant progress beam search produces nearly identical improvements using parameter combinations involving a progress threshold of 20,000 feet or less.

In both constant time beam search and constant progress beam search, changing the filter size had very little effect on the results. There was a noticeable trend, however, resulting from changes in the respective threshold. Figure 4.4 shows the average solution resulting from each time threshold, over all filter sizes, for constant time beam search in conjunction with rule methods 1 and 2. Figure 4.5 shows the average solution resulting from each progress threshold, over all filter sizes, for constant progress beam search in conjunction with rule methods 1 and 2.

Smaller time and progress thresholds produce better results because they evaluate the solution more often in the development process. However, implementing smaller time and progress thresholds also leads to longer runtimes due to the extra computation necessary. The right balance is dependent on how much time and computing power is available. There is also likely a lower limit to this trend, since the lowest possible thresholds are merely greedy searches.

While rule method 1 produced better results than rule method 2 using the earlier methods, both constant time beam search and constant progress beam search performed slightly better with rule method 2. Interestingly, as the time and progress thresholds became very large, and thus the process became more like the earlier methods, rule method 2 lost its advantage.

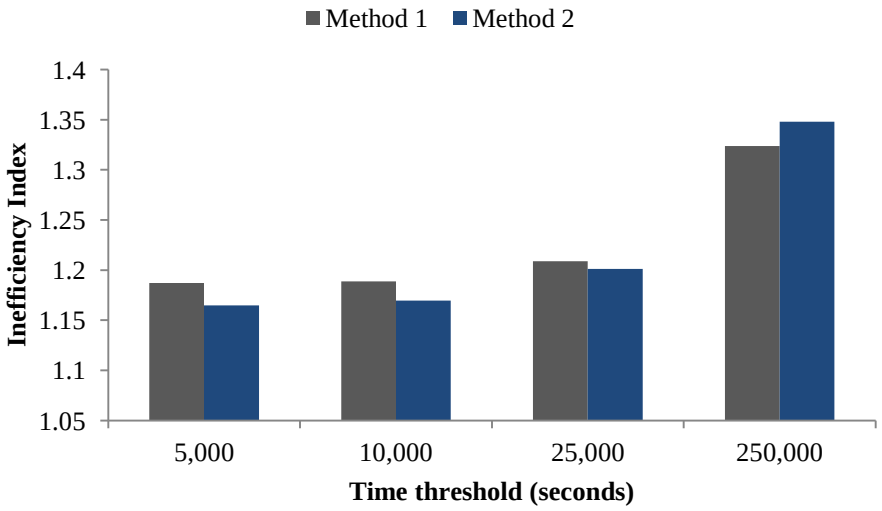

Figure 4.4: Average solutions using constant time beam search in conjunction with each rule method at various time thresholds. Note that the interval on the horizontal axis is not constant.

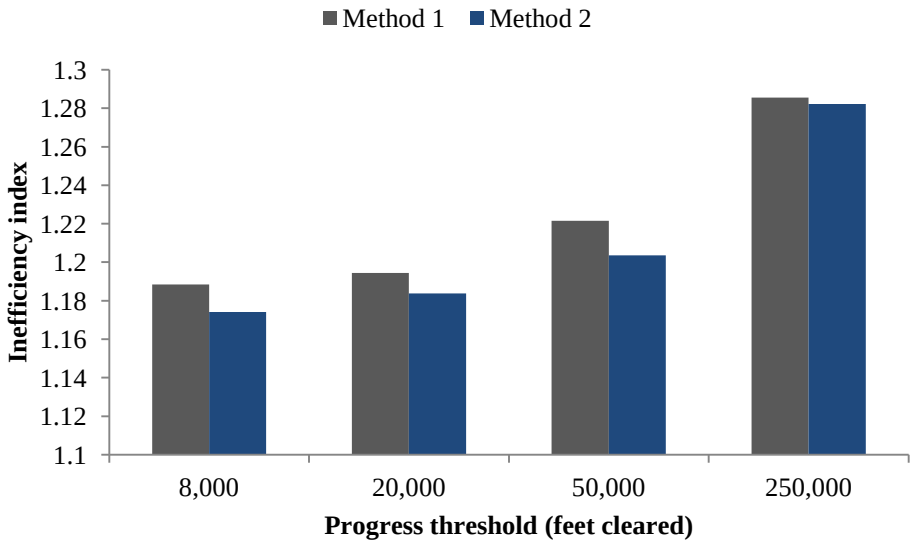

Figure 4.5: Average solutions using constant progress beam search in conjunction with each rule method at various progress thresholds. Note that the interval on the horizontal axis is not constant.

Unsurprisingly, the adaptations of local beam search did not perform as well in conjunction with the random-choice method as with the rule-based methods. Even with a time threshold of 5,000 seconds, constant time beam search produced an average inefficiency index of 1.55 with the random-choice method. With a progress threshold of 8,000 feet, constant progress beam search produced an average inefficiency index of 1.56 with the random-choice method. Additionally, while the method for choosing a path was simpler, producing a solution generally took more time using the random-choice method because the poor solutions required longer paths and thus more choices to be made.

In running the genetic algorithm with a population generated from level-based beam search, the selection method had little effect on the quality of solutions found. A higher mutation probability, however, enabled the algorithm to run longer before becoming stagnant, thus producing better solutions. The solutions found from using a mutation probability of 0.2 had an average inefficiency index of 1.22, while those found from using a mutation probability of 0.01 had an average index of 1.25. These solutions were improvements from level-based beam search, but were not as good as constant time or constant progress beam search. Four of the five genetic algorithms run with a mutation probability of 0.2 terminated because of the 50-generation limit rather than the stagnation criterion, so it is possible they would have found better solutions than they did. However, solutions generated using this mutation probability took an average of nearly five hours to find. Those generated using a mutation probability of 0.01 took an average of over an hour and a half. Better solutions could be found using constant time or constant progress beam search in minutes.

The best solutions were produced by the combination of all three types of heuristics: a rule set, a local beam search adaptation, and a genetic algorithm. The only such combination

tested used rule method 2, constant time beam search with a time threshold of 25,000 seconds, and a genetic algorithm with a mutation probability of 0.2. Each of the five runs produced a better solution than the best solution found by any other algorithm. Four of the five solutions found had an inefficiency index of 1.11, while the best had an inefficiency index of 1.10. The best found by any other combination of heuristics had an inefficiency index of 1.12 and was a product of constant time beam search in conjunction with rule method 2. This three-heuristic combination took an average of 13 hours to find a solution, though, so again, its value depends on the time and computing power available.

4.3 Analysis of the algorithms and implementations

The adaptations of local beam search, in conjunction with the rule-based methods, were responsible for most of the development of the best solutions. There was little difference between the solutions found from constant time beam search and those found from constant progress beam search, and within those heuristics, little difference between the solutions found employing rule method 1 and those found employing rule method 2. The only factor that made a noticeable difference in the performance of these heuristics was the size of the respective time or progress threshold. Even those parameters were not reported to provide significantly different results ($p = 0.05$), but the trend was apparent in Figures 4.4 and 4.5. The statistical significance testing may also have been less robust than it could have been since the sample sizes were limited to 20 solutions generated by each parameter combination.

After all its extended efforts, the genetic algorithm did ultimately prove to be of some value by improving upon the solutions found by the other heuristics. One source of the labor encountered by the genetic algorithm was the repair operator. While Dijkstra's algorithm is

simple to implement, it can require large amounts of computational effort in large networks. Since each transfer required n^2 executions of Dijkstra's algorithm (where n is the number of crews), crossover was an expensive operation. It may be beneficial to try to find or develop a faster repair operator.

One possible way to improve these algorithms is to develop different, and perhaps more complex, rule sets. The ones applied to this problem were intended to be equally applicable to any connected network. Some specific aspects of the Fort Stewart network could possibly be used to improve this algorithm's performance in this particular network. For instance, many of the priority levels in the Fort Stewart network are localized to particular areas of the map due to particular aspects of the training mission of the army. Nothing in either rule method directs a crew to roads that need to be cleared unless the crew happens to drive past such a road. That kind of direction could save significant amounts of time spent searching for roads to service.

The magnitude of the Fort Stewart problem makes the use of mathematical methods or integer programming an unrealistic means of finding solutions. The simulation representation used in this experiment not only can handle a problem this large, but also is adaptable to changes in the assumptions about the problem or the landscape. Another significant aspect of this approach is that it has no problem figuring out how to handle loops or dead ends; a straightforward Dijkstra's algorithm would need computational caveats to address these. There are, of course, elements of a real scenario that are not accounted for in the simulation algorithm.

The simulations used in the experiment are discrete representations of a continuous process, and the size of the time step can be thought of as the "resolution" of the simulation. When a crew reaches the end of an arc in a simulation, it remains at that point until the end of the time step, whereas in real life, it would immediately choose its next arc and continue its task.

This “lost” time upon reaching the end of an arc is a source of error, and the potential for this error increases with the size of the time step. Note also that this type of error can only serve to overestimate the time required to clear the road network. Thus, the same solution executed with two different-sized time steps will generally yield a better result with the smaller time step.

However, the amount of computation necessary to run the simulation is directly related to how many times the computer must update the status of the crews. Thus, smaller time steps lead to more computational effort. Finding the right balance to generate an accurate, high-quality solution in a reasonable amount of time depends on the priorities of the user.

Another concept that was not represented in the algorithm was that of turning around in the middle of an arc. It may be better in some situations for two crews to team up on the first half of a section of road, but then have one of the crews turn around and clear debris elsewhere while the other crew finishes the task at hand. In this implementation, however, crews only make decisions at the nodes, and the arcs are only designated as fully cleared or not cleared. A related oversight in the implementation involves the ability for more than two crews to work on an arc at the same time. If two crews clear debris from one end, and two crews clear debris from the other, they would not impede each other’s progress. This implementation, however, only recognizes the number of crews on the arc in determining whether an additional crew adds value.

Several minor adjustments were made to the data as well and should be noted. The priority levels of five arcs were changed in order to make the problem feasible, as these arcs created disconnected networks. Additionally, the ending node of one arc was changed to match the map image. The arc had originally been represented as being completely disconnected from the rest of the network.

4.4 Future directions

The next step in this problem would be to test the effects of changing some of the assumptions mentioned in section 4.1. While the algorithm as it stands can function under many different sets of assumptions, it may be that the rule sets are not very appropriate when the assumptions change. If the priority constraints are softened, the rule sets would need a major overhaul, and the objective function would also have to be adjusted. Currently, the objective function consists solely of the time it takes to finish clearing the network. An alternate objective function could be composed of the elapsed time upon clearing each priority level, perhaps with different weights associated with each. In a real recovery scenario, this layered objective function with soft priority constraints would probably provide a more appropriate solution.

Another issue to consider is that whatever assumptions are made are not likely to hold entirely true. Since the crews may take breaks each night (if not replaced with other workers), it can be expected that the algorithm would be re-run each night upon updating the state of the roads and probably adjusting some assumptions. This approach would prevent errors in assumptions from propagating through the entire recovery process.

The best solution found in this experiment, assuming five cleanup crews, still required about three and a half weeks of nonstop debris removal. If the crews were to work twelve-hour days, seven days a week, it would take about seven weeks. After several weeks, it may be that the focus shifts from clearing the roads to salvaging lumber. At this point, the objective becomes something entirely different. Rather than minimize the amount of time required to clear the roads, the objective may be to maximize the value of the recovered logs, since most of the main arteries throughout the installation would be open, and thus the speed at which the roads are cleared may not be as critical. In this case, the demand load would change as well, since it takes

longer to buck and trim limbs and load the wood onto log trucks than it would to simply clear a tree from the road.

4.5 Validation of results

Solutions to this problem had never been developed previously, so there are no past results against which to validate these. Different combinations of heuristics were validated against each other, and the heuristics that were made of combinations of other heuristics generally produced better results than the isolated heuristics. This is considered a self-validation process (Level 2) [21], where best case performance, average case performance, and variation in the results were examined, along with tests of the sensitivity of heuristic parameters. In cases where exact solutions to problems are unavailable, self-validation is an adequate way of assessing solutions from a new heuristic applied to a new problem. However, all of the solutions were also validated against a theoretical optimum, which was unattainable due at the very least to bottlenecks, dead ends, and loops. In the future, it may be possible to validate the algorithm against a small, theoretical problem that can be solved with integer or mixed-integer programming.

CHAPTER 5

CONCLUSION

The Fort Stewart road-clearing problem presented a challenge unique from other Hierarchical Chinese Postman Problems and Capacitated Arc Routing Problems due to the stipulation that roads must be serviced on their first traversal. Having multiple crews was an added challenge, and the limit on the number of crews that could effectively work in the same location provided yet another challenge on top of that. The size of the network, the road prioritization, and all the factors that determine how fast the crews can traverse the arcs, make it practically impossible to use mathematical methods to solve this problem. Thus, heuristic methods were developed to try to find the best solution possible.

Two basic rule sets were conceived to provide general guidelines for the cleanup crews in choosing their routes. The idea of local beam search was then adapted in three ways to fit this problem. The rule sets and the local beam search adaptations were tested independently and provided some baseline results. When the heuristics were combined, they produced much better results in a reasonable amount of time.

A representation was developed to enable the use of a genetic algorithm to develop solutions as well. Level-based beam search, which was the weakest but simplest (and fastest) of the three local beam search adaptations, was combined with one of the rule sets to generate the initial population of the genetic algorithm. From that initial population, the genetic algorithm was able to produce solutions almost as good as those from the local beam search adaptations, but with far more computational effort.

Finally, a stronger local beam search adaptation was combined with a rule set to generate the initial population of the genetic algorithm. Though the process took many hours, the resulting solutions were better than anything else that had been found. Each heuristic, when combined with the others, provided an enhancement.

REFERENCES

[1] United States Federal Emergency Management Agency. (1993). *Emergency management guide for business and industry: A step-by-step approach to emergency planning, response and recovery for companies of all sizes*. Washington: United States Federal Emergency Management Agency.

[2] American Planning Association. (2012). *Planning For Post-Disaster Recovery: Next Generation*. Chicago: American Planning Association. Retrieved April 6, 2012, from http://www.planning.org/research/postdisaster/

[3] Benavent, E., Campos, V., Corberán, & Mota, E. (1990). The capacitated arc routing problem. A heuristic algorithm. *Qüestiió*, *14*(1,2,3), 107-122.

[4] Dror, M., Stern, H., & Trudeau, P. (1987). Postman tour on a graph with precedence relations on arcs. *Networks, 17,* 283-294.

[5] Golden, B. L., & Wong, R. T. (1981). Capacitated arc routing problems. *Networks, 11,* 305-315.

[6] Perrier, N., Langevin, A., & Amaya, C. A. (2008). Vehicle routing for urban snow plowing operations. *Transportation Science, 42*(1), 44-56.

[7] Monroy, I. M., Amaya, C. A., & Langevin, A. (in press). The periodic capacitated arc routing problem with irregular services. *Discrete Applied Mathematics*.

[8] Cabral, E. A., Gendreau, M., Ghiani, G., Laporte, G. (2004). Solving the hierarchical Chinese postman problem as a rural postman problem. *European Journal of Operational Research, 155,* 44-50.

[9] Salazar-Aguilar, M. A., Langevin, A., Laporte, G. (2012). Synchronized arc routing for snow plowing operations. *Computers & Operations Research, 39,* 1432-1440.

[10] Mei, Y., Tang, K., & Yao, X. (2009). A global repair operator for capacitated arc routing problem. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics, 39*(3), 723-734.

[11] Amponsah, S. K., & Salhi, S. (2004). The investigation of a class of capacitated arc routing problems: The collection of garbage in developing countries. *Waste Management, 24,* 711-721.

[12] Atkins, J. E., Dierckman, J. S., & O'Bryant, K. (1990). A real snow job. *The UMAP Journal, 11*(3), 231-239.

[13] Chernak, R., Kustiner, L. E., & Phillips, L. (1990). The snowplow problem. *The UMAP Journal, 11*(3), 241-250.

[14] Robinson, J. D., Ogawa, L. S., & Frickenstein, S. G. (1990). The two-snowplow routing problem. *The UMAP Journal, 11*(3), 251-259.

[15] Hartman, C., Hogenson, K., & Miller, J. L. (1990). Plower power. *The UMAP Journal, 11*(3), 261-272.

[16] Bettinger, P., Merry, K. L., & Hepinstall-Cymerman, J. (2010). Fort Stewart timber salvage and recovery study, and modeling of potential windthrow and storm surges associated with hurricanes, final report. Athens, GA: Warnell School of Forestry and Natural Resources, University of Georgia.

[17] Xu, H., Zhang, C., Tan, Y., & Lu, J. (2011). An improved evolutionary approach to the extended capacitated arc routing problem. *Expert Systems with Applications, 38*(4), 4637-4641.

[18] Goldberg, D. E. (1994). Genetic and evolutionary algorithms come of age. *Communications of the ACM, 37*(3), 113-119.

[19] Russell, S. J., & Norvig, P. (2010). *Artificial intelligence: A modern approach* (3rd ed.). Upper Saddle River: Prentice Hall.

[20] Smith, D. K. (1982). *Network optimisation practice: A computational guide.* Chichester: Ellis Horwood Limited.

[21] Bettinger, P., Sessions, J., & Boston, K. (2009). A review of the status and use of validation procedures for heuristics used in forest planning. *Mathematical and Computational Forestry & Natural-Resource Sciences, 1*, 26-37.