A Demonstration of a Simulation Tool for Planning Robust Military Village Searches

Paul Maxwell[1,3], Ryan Friese[1,2], Anthony A. Maciejewski[1], Howard Jay Siegel[1,2], Jerry Potter[1], Jay Smith[1,4]

[1]Electrical and Computer Engineering Department
[2]Computer Science Department
Colorado State University, Fort Collins, CO 80523-1373 USA
[3]United States Army
[4]DigitalGlobe, Inc., Longmont, CO 80503

Keywords: robustness, village search, resource allocation, simulation

Abstract

In the current military environment, village searches are conducted daily by a variety of search team types. Staff officers planning these resource allocation problems currently rely on experience and simple data tables to develop the plans. The Robust People, Animals, and Robots Search (RoPARS) planning tool for village searches developed at Colorado State University can assist military planners with this tedious process. The tool consists of a graphical user interface and a resource allocation engine. Its output is a mission plan that is robust against uncertainty in the battlefield environment (e.g., unit speed, temperature, enemy contact). The contributions of this paper include the RoPARS tool and its robustness concepts, mathematical models, and resource allocation heuristics.

1. INTRODUCTION

Colorado State University's Information Science and Technology Center, ISTeC (ISTeC.ColoState.edu), is organizing the PAR (People-Animals-Robots) multi-disciplinary research laboratory [8] to study how teams of people, animals, and robots can be used together in new, synergistic ways in a variety of environments, including health care, search and rescue, and military operations. This effort is part of that PAR Lab.

In the modern, fast-paced, high technology military, making decisions on how to best utilize resources to accomplish a mission with a set of specified constraints is difficult. A Cordon and Search of a village (i.e., village search) is an example of such a mission. Leaders must plan the mission, assigning assets (e.g., soldiers, robots, unmanned aerial vehicles, military working dogs) to accomplish the given task in accordance with orders from higher headquarters. Computer tools can assist these leaders in making decisions, and do so in a manner that will ensure the chosen solution is within mission constraints and is robust against uncertainty in environmental parameters. Currently, no such tools exist at the tactical or operational level to assist decision makers in their planning process and, as a result, individual experience is the only tool available. This paper introduces the Robust People, Animals, and Robots Search (RoPARS) planning tool for village searches that coordinates soldiers, military working dogs, and robots. The contributions of this paper include the RoPARS tool and its robustness concepts, mathematical models, and resource allocation heuristics. The result is a decision-making aid for military leaders to use during the mission planning phase of an operation.

The state of the art for village search planning is limited to individual experience and basic data tables from military Field Manuals ([10], [9]). The result is dramatic variation in the quality of a plan, its ability to account for uncertainty, and the resulting confidence in its success. The RoPARS simulation tool improves mission planning by incorporating: (1) mathematical models that represent people, animals, and robots (PAR) and account for uncertainty in the environment; (2) robustness metrics that assist decision makers in managing complex processes with a high degree of certainty regarding tactical mission completion; and (3) heuristics based on those models and metrics that result in near-optimal resource allocations (i.e., completing a village search by its deadline with maximum probability). The output of the tool is a recommended resource allocation for the village search mission that is expected to meet the provided constraints in a timely manner and reduce the risk for those involved in the mission.

The RoPARS tool automates many functions of the planning process thus freeing planners for other tasks. Through a graphical user interface (GUI), the tool imports Environmental Systems Research Institute (ERSI) standard shapefiles of the search area (many of which are currently available in the Urban Tactical Planner library), accepts user inputs regarding the plan (e.g., phase lines, boundary lines,

This research was supported by the National Science Foundation under grant number CNS-0905399, and by the Colorado State University George T. Abell Endowment

search team types and compositions, target buildings), creates a resource allocation using static allocation heuristics (e.g., Minimum search heuristic, genetic algorithm), evaluates the performance of the allocation using quantifiable measures, and graphically displays at a user-selected rate the resulting plan. This automated mission analysis tool uses stochastic information, is faster than human generated solutions, and is more reliable in terms of the robustness of its results than existing methods. Inevitably this will contribute to better, more informed decisions for military commanders in combat operations in the contemporary operating environment.

The remainder of the paper is organized as follows. The village search problem background is introduced in Section 2. Section 3 reviews related work in the fields of robustness and resource allocation. A description of the RoPARS tool is in Section 4. In Section 5, the simulation set-up is described and results are presented. Finally, we present our conclusions in Section 6.

2. BACKGROUND

In a military village search problem, there are one or more target buildings that require searching. The units that conduct the search can consist of numerous different forces with the primary ones being human teams of soldiers, human teams augmented with military working dogs, and human teams augmented with search robots. At the basic level, the problem is a resource allocation problem where search resources (teams) must be assigned to target buildings. This resource allocation problem in itself is computationally complex. Often this problem is made more difficult by the introduction of constraints on the solution. These constraints take the form of boundary lines (lines that demarcate allowable search areas for teams), phase lines (ground reference lines that act as synchronization barriers controlling the forward movement of the search teams), and directions of advance (limits search direction). An example village search problem is shown in Figure 1.

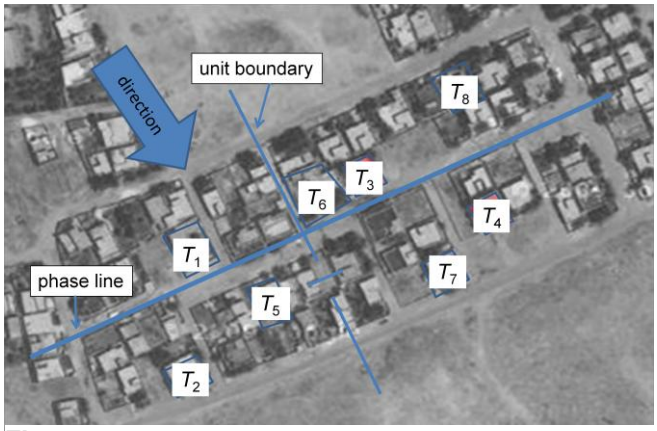

Figure 1. Example village search mission with eight target buildings (T_j), a unit boundary, a restrictive phase line, and a direction of advance.

The operating environment for these searches contains numerous uncertainties that render exact solutions impractical. These uncertainties include but are not limited to factors such as varying search times, encounters with the enemy, weather impacts on the search, and mechanical malfunctions. A resource allocation based on expected values will frequently not produce the best solution when these uncertainties are incorporated. It is therefore our goal to develop a near optimal resource allocation for the village search problem that is robust against these uncertainties.

3. RELATED WORK

At a high level of abstraction, the village search problem is similar to the multiple traveling salesmen problem (mTSP). Like the mTSP, each target building (city) must be visited once by only one search resource (salesman). To complete the comparison, the salesmen start and end at the same location. The main difference between the village search problem and mTSP is that the village search problem incorporates time spent searching at the nodes into the problem statement whereas the mTSP domain generally does not include time spent in the visited cities.

There has been extensive research into solutions for the TSP ([6], [11]) and mTSP problem [3] due to their wide applicability and complexity. Here we review only works that use genetic algorithms to find a solution.

The work in [14] models a steel rolling factory as an mTSP problem and uses a genetic algorithm to produce solutions. Similar to the village search problem, the steel rolling problem has constraints that reduce the number of valid solutions. The steel rolling problem considers time at a node instead of distance between nodes. Unlike our work though, their genetic algorithm uses discrete values instead of stochastic information and is not concerned with the robustness of the solution.

In [12], a genetic algorithm is used to produce solutions for a global satellite survey network problem. This problem domain was transformed into an mTSP problem where the salesmen are satellites and the cities are survey jobs for the satellites. The objective of the research is to find a minimal cost route between survey points using a cost matrix to define the edge cost. In this domain, the problem is restricted to deterministic values and is not concerned with robustness. Additionally, the problem does not have limiting constraints on the solution such as the boundary lines of the village search problem.

The work in [16] transforms a multiple robot mine clearing problem into an mTSP problem. In this research, multiple robots must move to multiple mines and remove those mines in a cooperative manner. The authors use a genetic algorithm to find a solution that minimizes the paths the robots traverse while removing all the mines. As with the other works surveyed, this work uses deterministic values and does not consider uncertainty in its calculations.

Additionally, the mine removal time (equivalent to the search time of target buildings) is not considered in the problem.

There is much TSP and mTSP research to provide near optimal solutions to a particular domain. The solution techniques vary in heuristic style and in choices like sequential versus parallel execution of the heuristics but none of the works surveyed address robustness.

4. RoPARS TOOL

4.1. ROBUSTNESS AND VILLAGE SEARCH

Solutions to problems are often highly valued if they are robust. In the village search domain, military leaders desire a solution to the problem that is able to withstand the effects of uncertainty. This is evidenced by the common military saying, "the plan goes out the window as soon as we cross the line of departure." A robust mission plan that can survive the effects of the unknown is valuable. However, the definition of the term "robustness" is frequently unclear.

The three robustness questions that help define robustness for a system are [2]: (1) What behavior of the system makes it robust? (2) What uncertainties is the system robust against? and (3) Quantitatively, exactly how robust is the system?

In the search of a village such as shown in Figure 1, the robustness questions can be answered differently depending on the performance objective in question. To illustrate the application of the questions to this domain, we will define the system to be robust if all the search resources (*SRs*) complete their building searches prior to the mission deadline time (*MDT*). This is a time constraint by which all teams must complete. The uncertainties against which the system is robust are variability in the ideal search rates of the search resources and variability in the ideal movement rates of the search resources. The uncertainties the RoPARS tool models are not limited to these choices. Any uncertainty that can be modeled may be incorporated into the robustness metric. To answer the third robustness question, we use the FePIA method.

The FePIA (Features, Perturbation parameters, Impact, Analysis) method [1] is a framework for measuring robustness. Using the FePIA method, the following are identified: (1) the performance features that determine if the system is robust, (2) the perturbation parameters that characterize the uncertainty, (3) the impact of the perturbation parameters on the performance features, and (4) the analysis to quantify the robustness. As these steps are followed, the robustness metric for a system is created (other examples of robust resource allocations can be found in [8] and [13]).

First, we define the search resource completion time for team i (RCT_i) to be the performance feature for this system. If there are m search resources, then there are m performance features. The perturbation parameters for step 2 are the same uncertainties outlined in the previous paragraph. The impact of the perturbation parameters on the performance feature is that they will either increase or decrease the value of RCT_i. For example, search resource i may have an estimated search rate of 0.65 m^2/s based on user input. However, the true search rate may less than that due to an unanticipated problem such as the illness of a search team member. Thus, it will take longer to search buildings and the value of RCT_i will increase.

The analysis to quantify the robustness of the system is complex and the reader is referred to [7] for a more detailed discussion. To summarize, the time to search a given building is a complicated function of factors that include the building characteristics, search team characteristics, and probability mass functions (pmfs) of the relevant uncertainties. This results in a pmf that represents the building search time. To calculate RCT_i it is necessary to convolve (assuming independence) the building search time pmfs for the buildings assigned to that team and the associated road segment traversal pmfs resulting in the search team's completion time pmf.

It is assumed that the search resources have adequate supporting elements to operate independently and therefore the search resource completion times are independent within a phase line area. If there are no phase lines constraining the search, the stochastic robustness metric, *SRM*, of [7] is defined as the product of the all the teams' probabilities of completing before the *MDT*. That is, the *SRM* is the probability that all resources (teams) will complete by the *MDT*. If phase lines exist, then the maximum of the search resources' pmfs for the phase line area is calculated. The resulting pmf is used as the initial value pmf for each search resource's *RCT* in the next phase line area. Then the *SRM* is calculated using the *RCT* values of the last phase line area.

4.2. ENVIRONMENT

The village search mission environment is defined by its boundary lines and the assignment (grouping) of search resources to specific boundary line areas. Assuming that at least one boundary line is present, a village search problem solution space has many combinations of the number of search resources to boundary line area assignments to explore. For example, if there are five search resources and two boundary line areas, then there exist four valid grouping possibilities (one *SR* on the east side, four *SRs* on the west side (1,4), two *SRs* on the east side, three *SRs* on the west side (2,3), etc.). If more boundary lines exist (i.e., two boundary lines) and five *SRs* are used then a grouping tuple may be (2,2,1).

4.3. MINIMUM SEARCH HEURISTIC

The minimum search heuristic was inspired by the original two-phase greedy heuristic in [5]. It is modified to

fit the village search domain and its constraints. It is a fast, deterministic heuristic and thus can provide valid solutions in a time constrained environment.

The minimum search heuristic is used to find a solution for one specific grouping tuple. The heuristic is deterministic and will assign the same starting building locations for each execution if unguided. To expand the search area and improve the solution, the heuristic was modified to randomly select starting building locations for each search resource. This forces the heuristic to explore other solutions. The pseudocode for the minimum search heuristic is shown in Figure 2.

```
1.  for every combination of y_i search resources assigned to boundary
    area i do
2.      for (number of trials x) do
3.          select random building starting location for each search
            resource
4.          for each phase line area j
5.              for each boundary area i
6.                  for each search resource (SR) assigned to boundary
                    area i
7.                      find minimum completion time (MCT) unassigned
                        building and associated road segment
8.                  find MCT building/SR pair from line 7
9.                  assign MCT building from line 8 to its matched SR
10.                 remove MCT building from unassigned building list
11. output allocation with the highest Stochastic Robustness Metric
```

Figure 2. Minimum search heuristic pseudocode.

4.4. VILLAGE SEARCH GENETIC ALGORITHM

The minimum search heuristic provides valid solutions but in general does not provide good quality solutions due to its limited exploration of the search space. The village search genetic algorithm (VSGA) compensates for this weakness by exploring the space more broadly.

The VSGA is a modification of a classic evolutionary genetic algorithm. Its pseudocode is shown in Figure 3. The VSGA uses the minimum search heuristic solution as a seed chromosome. The other chromosomes in the population are generated randomly (ensuring valid chromosomes). During chromosome generation, a look-up table is created that cross-references *SR* index values to *SR* absolute reference values (Figure 4a). The reason for this table will be discussed later. The VSGA uses stochastic universal sampling for the selection of the next population. In this technique, selection bias with regard to the expected reproduction rate is avoided and the next population is selected in one "spin" of the virtual roulette wheel [4]. In each generation, chromosomes are subjected (using a chosen probability) to crossover and mutation operators. Then each chromosome in the population is evaluated using the stochastic robustness metric as the fitness function. The details of the chromosome operators and the fitness function are described in subsequent paragraphs.

At its most basic level, a chromosome for the VSGA consists of multiple "strands." An example strand is shown in Figure 4b. The strand is an array composed of target building number/search resource index pairs. The strand represents target buildings, their assigned search resources, and the scheduling order of the target buildings. The building number/*SR* index pair's position in the array defines a global ordering with pairs in the leftmost array position being first in time. The strand's length is determined by the number of target buildings within its boundary line and phase line area. The VSGA uses strands due to the constraints placed on the solution by the problem domain such as boundary lines that limit crossover and mutation changes.

```
while stopping criteria not met
    1. select next population using Stochastic Universal Sampling
    2. for (number of chromosomes/2) do
        a. randomly select two chromosomes
        b. if (random < probability of crossover) do scheduling crossover
        c. if (random < probability of crossover) do matching crossover
    3. for (number of chromosomes) do
        a. if (random < probability of mutation) do scheduling mutation
        b. if (random < probability of mutation) do matching mutation
    4. for (number of chromosomes) do evaluate fitness function
```

Figure 3. Village search genetic algorithm pseudocode.

Chromosome *Y*

SR id #	Index #
1	0
2	1
4	2

(a)

Bldg Index	1	4	0	2	3
SR Index	0	1	1	0	1

(b)

Figure 4. Village search genetic algorithm chromosome components: (a) search resource look-up table, and (b) chromosome "strand."

At the next higher level, the VSGA chromosome consists of one or more strands. The number of strands in a chromosome is determined by the number of phase line areas multiplied by the number of boundary areas. In Figure 5, an example chromosome is shown with its four component strands for a two phase line area by two boundary line area village search. When the chromosome's fitness is evaluated, the strands are assembled as a whole into the chromosome and then the *SRM* is calculated.

The crossover operators in the VSGA are the scheduling and the matching crossovers. The operators function using two randomly selected parent chromosomes

from the population and they produce two child chromosomes. In a particular generation, the number of crossover operations is less than or equal to half the number of population chromosomes. Examples for these operators are shown in Figures 6 and 7. Similar to classic genetic algorithms, the crossover operators use two parent chromosomes in their operation. Additionally, the crossover operators function on the same strand within each parent chromosome. For example, a crossover operation could be performed on strand 0,0 (Figure 5) on both parent chromosome *A* and parent chromosome *B*. Crossover (and mutation) operations can be performed on more than one of the strands in a chromosome if desired. However, in this work, only one randomly selected strand per chromosome pair is operated upon in a given generation.

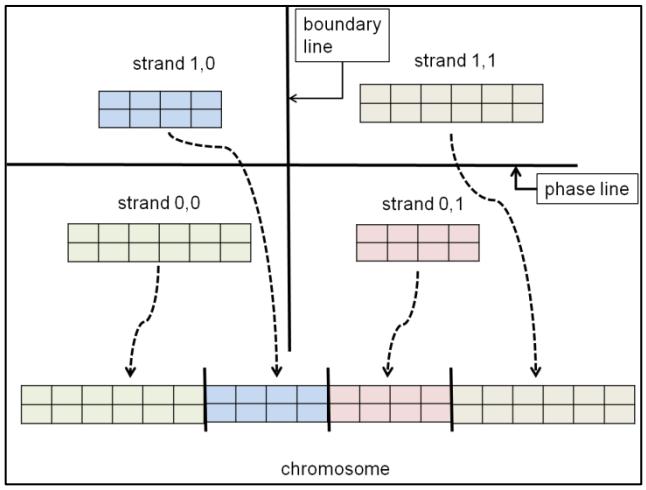

Figure 5. Village search genetic algorithm chromosome.

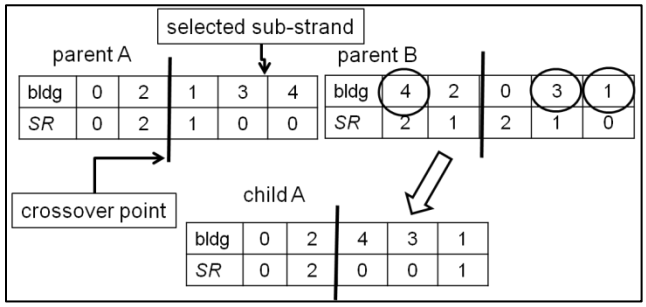

Figure 6. Village search genetic algorithm scheduling crossover example.

The scheduling crossover operation operates as follows. A single crossover point is randomly chosen and then the sub-strand to perform the crossover upon is randomly chosen. Unlike crossover operators described in [15] that function on the "right" portion of the parent chromosomes, the VSGA crossover operator chooses the "left" or "right" sub-strand of the strand for crossover. Next, the scheduling order of the target buildings in the selected sub-strand of parent chromosome *A* are re-ordered to match the scheduling order of parent chromosome *B*. The operation is performed again with the parents' roles reversed.

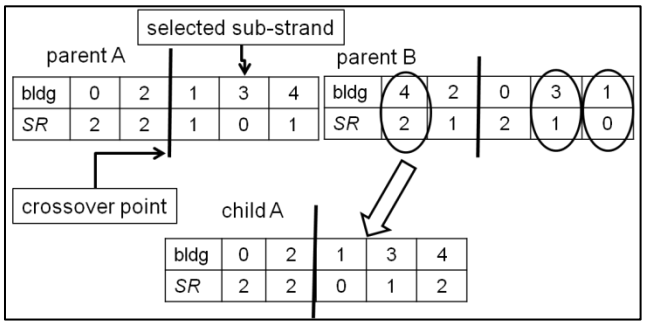

Figure 7. Village search genetic algorithm matching crossover example.

The matching crossover operates similarly to the scheduling operator. A single crossover point is randomly chosen and then a sub-strand is randomly chosen. Each target building within the chosen sub-strand of parent *A* is assigned the search resource it has in parent *B*. The operation is then repeated with the parent chromosomes reversed (see Figure 7).

Similar to the crossover operators, the mutation operators function on a strand within a chromosome. Again, the operators can be performed on more than one strand but as with the crossover operators it has been limited to one strand for the example in this paper. Examples for the scheduling and matching mutation operators are shown in Figures 8 and 9.

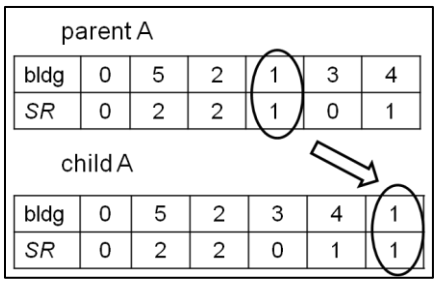

Figure 8. Village search genetic algorithm scheduling mutation example.

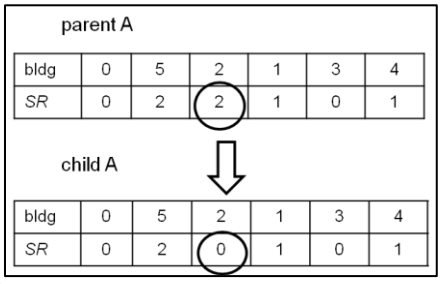

Figure 9. Village search genetic algorithm matching mutation example.

The scheduling mutation operator begins by randomly selecting a target building/search resource pair to reschedule. Next, it randomly selects a new order position in the strand. It then inserts the target building/search resource pair at the newly selected destination creating a new scheduling order for that strand.

The matching mutation operator begins by randomly selecting a target building/search resource pair to mutate. The operator then randomly selects a new search resource from the set of search resources operating within the given boundary line area. The selected search resource is then assigned to the target building creating a new matching.

All of the operators described in the preceding paragraphs operate in each generation. For each operator, a user selected probability is used (i.e., probability of crossover, probability of mutation) to randomly apply the operator on the selected chromosomes. As with all genetic algorithms, an optimal solution is not guaranteed. Additionally, its convergence rate and quality of solution is dependent upon implementation factors (e.g., population size, number of generations created, probability of crossover/mutation).

As mentioned previously, through use of a look-up table (Figure 4a), the *SR* indices are associated with absolute *SR* identification numbers. Chromosome *SR* look-up tables are inherited from parent to child during crossover and mutation operations. We use index tables so that we can search multiple combinations of search resources in a grouping without generating invalid chromosomes during crossover operations. An example of how invalid chromosomes could occur follows. In parent chromosome *A*, absolute search resources 1 and 4 are on the "eastern" side of a boundary line while resources 0 and 3 are on the "western" side (in group notation - <1,4|0,3>). In parent chromosome *B*, absolute search resources 0 and 3 are on the "eastern" side and absolute search resources 1 and 4 are on the "western" side (<0,3|1,4>). A matching crossover operation could attempt to assign search resource 3 to the "eastern" side of a child chromosome of parent *A* and *B* that has search resource 3 already assigned to the "western" side (resulting in a group <1,3|0,3>). Because a search resource can only search on one side of a boundary line, this is an illegal chromosome. With our index representation, both search resource 4 and 3 can be represented in the strand as *SR Index* 1 (group notation - <0,1|0,1>). They can then conduct crossover operations without generating invalid chromosomes. In this example, when the child chromosome *A* is assigned *SR Index* 1 during matching crossover, the valid absolute search resource number 3 is maintained from parent *A* and a valid chromosome is the result (<1,4|0,3>). Additionally, the generic representation allows us to explore the search area for a given resource assignment grouping (e.g., (3,2)) with less machine time than searching each resource assignment ordering individually.

4.5. GUI DESCRIPTION

The RoPARS GUI preprocesses the data and allows a user to visualize the layout of a specific village and manipulate search resources and constraints. The GUI allows the search area to be specified without tedious user action. The GUI handles the creation of village data files to be processed by the RoPARS resource allocation engine. After an allocation is created by the RoPARS tool, a user can review the plan by viewing an animation that shows the search plan being conducted at a user selected speed.

The GUI represents a village by reading in a pair of Environmental Systems Research Institute (ESRI) standard shapefiles. One shapefile contains information on the village's road infrastructure, while the other one contains information about the village's buildings. From these shapefiles, many important pieces of data are derived. Aside from allowing the village to be graphically represented, they also allow the GUI preprocessor to calculate a building's "center of mass" (centroid) and approximate area, as well as the lengths of the roads.

After the shapefiles have been input, the GUI allows the user to perform many actions that will define the village search plan. A user can draw boundary and phase lines on the village's representation, thus separating the village into distinct sections. The GUI assigns each section a color allowing the user to identify the buildings and roads in that section. A user can also select which buildings they need to search. The GUI uses color to show which buildings will be searched and which buildings will not be searched. The final action a user can perform is selecting search teams for the village. The user can select from three types of teams: human, military working dog, and robot. As well as being able to select the type of team, the user can edit certain aspects the team such as average search and movement rates. Figure 10 shows a model village with a boundary line, a phase line, and thirty target buildings.

After all the constraints for the search plan have been input, the GUI creates three data files and sends this information to the RoPARS resource allocation engine. These files include a road network adjacency file, a building data file, and a search resource data file. The road network adjacency file contains connection and length data for all the road segments in the village. A road segment is demarcated by two nodes. Nodes occur where a road changes direction, at road intersections, at the closest point to a target building, and every 200 meters if no other break has occurred. Every time a change is made to the village, (e.g., a new building is selected, a phase/boundary line is added) the nodes are updated. The building data file contains: information about target building locations, the road segment node connected to the buildings, and the target buildings' area. The search resource data file contains: information about the types of search resources in the village, search resource movement rates, and search resource search rates.

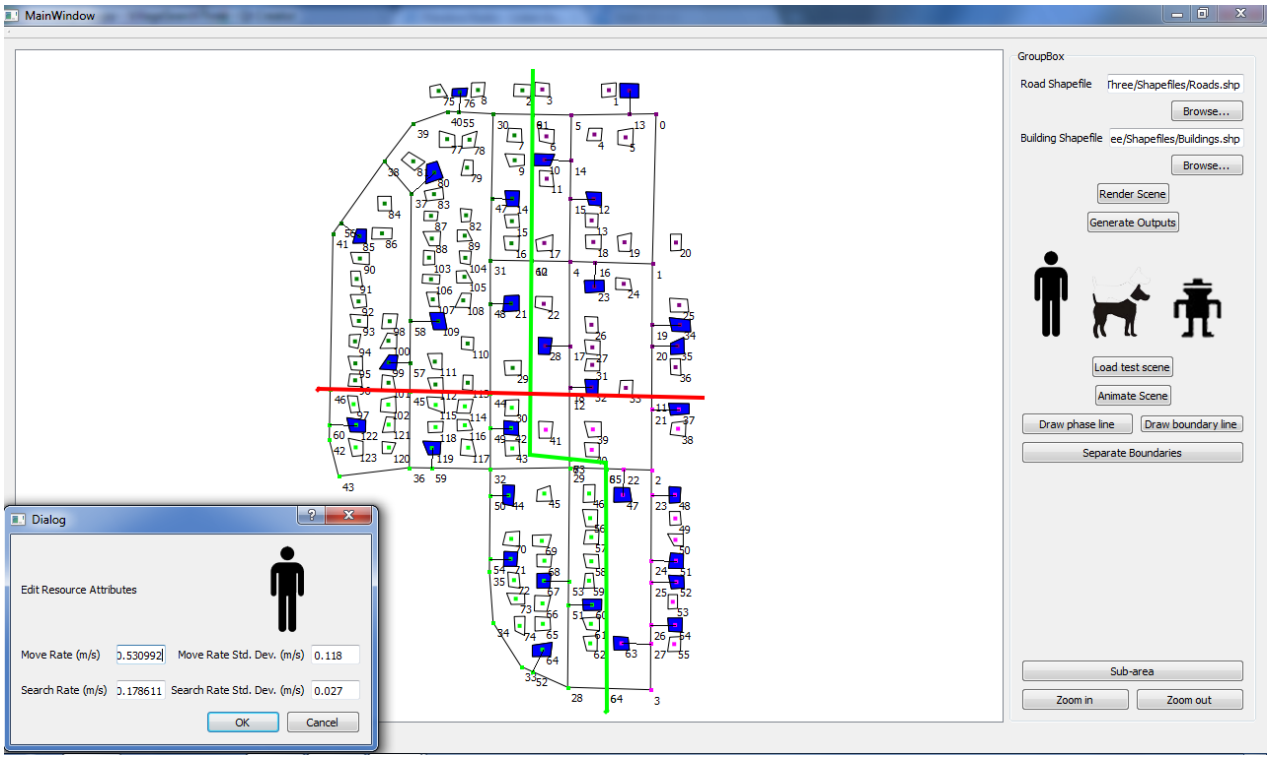

Figure 10. RoPARS tool GUI screen capture of an example village search scenario with a boundary line (green), a phase line (red), thirty target buildings (blue), and search resource input dialog box.

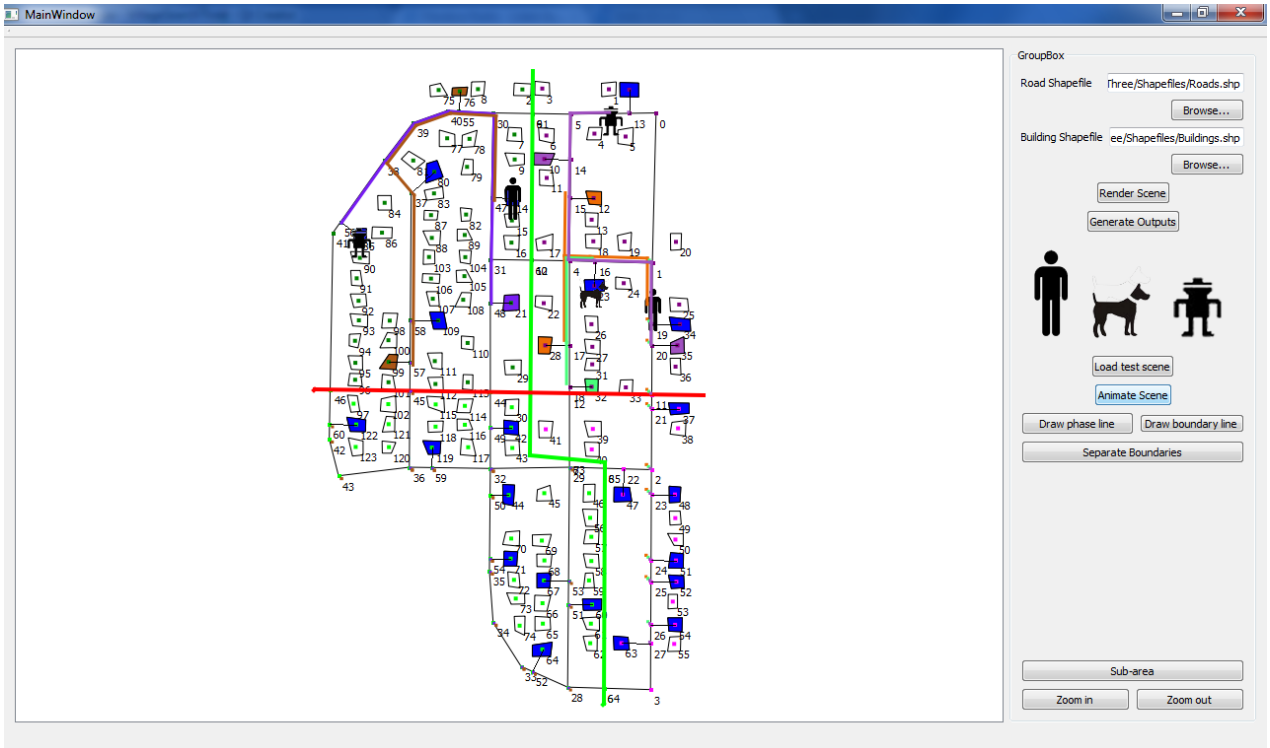

Figure 11. RoPARS tool GUI screen capture of an example resource allocation in playback mode with five search resources and their associated colored movement paths (brown, blue, purple, green, orange).

After the RoPARS resource allocation engine creates the resource allocation, a file containing the search plan is created and interpreted by the GUI. This file contains information on the specific routes individual teams will travel, the target buildings they will search, and the timing data for each team. This information allows the GUI to display an animation of the search plan (Figure 11). This animation is played at a user selected rate.

5. SIMULATION RESULTS

The results discussed here are based upon a test village search scenario using five search resources (four human teams, one military working dog team), 24 target buildings, and 64 road segment nodes. A realistic mission deadline time was chosen of 1.5 hours. Mission constraints included one phase line and one boundary line. Additionally, the *SR* grouping was three search resources on the "eastern" side of the boundary line and two search resources on the "western" side (3,2). To find the optimal solution, all possible combinations of boundary assignments should be explored. However, in this paper, only one grouping was tested as a proof of concept.

The simulation results are based on 50 trials. In each of the trials, the search resource movement rates and search rates differ from one trial to the next. The charts shown here display the data collected from these 50 trials.

The "expert" solution was created by Lieutenant Colonel Maxwell who has 18 years experience in the Army and has planned and executed village search missions. The expert solution was created by hand using only the tools currently available to military planners.

The minimum search heuristic results are shown using 50 sets of random building starting location assignments per trial. Experiments were done with different numbers of random building starting locations ranging from 1 to 50. There was a 1.5% improvement in the *SRM* between one random starting location and 50 random starting locations; however, there was only a 0.08% improvement between 20 random starting locations and 50 random starting locations. Thus, increasing the number of random building starting locations does not guarantee a proportional improvement in the quality of the solution.

Experiments were conducted using population sizes of 20, 50, and 100 chromosomes. In each experiment, the chromosomes were drawn from a file of 100 randomly generated chromosomes such that the 50 chromosomes were a subset of those in the size 100 population and the 20 were a subset of those in the size 50 population. For these experiments, the probability of crossover was set to 0.8 and the probability of mutation was set to 0.1. Additionally, the stopping conditions for the genetic algorithm were set to 1000 total generations or 200 generations with no change in the best solution. Elitism was used thus ensuring the best solution was kept in the population across generations.

On average, the quality of the solutions produced by the VSGA was better than the other approaches. The trial average of the VSGA was 1.75% better than the expert average and 3.92% better than the minimum search average. For larger village search scenarios, the length of time to calculate an expert solution will increase and its quality will probably decrease. This makes using even a simple heuristic like minimum search valuable. Additionally, it is hypothesized that the quality of the minimum search and VSGA will improve relative to the expert solution as the problem size increases.

The results are shown in Figure 12. The expert solution performed better than the minimum search allocation primarily due to the human ability to choose initial starting locations that ensure routes to subsequent target buildings are not too long. The minimum search heuristic is a greedy heuristic and does not account for the long term impact of its choices. This results in allocations that are relatively non-robust. The reader is also reminded that the quality of the expert solution will vary greatly depending on the experience and expertise of the planner.

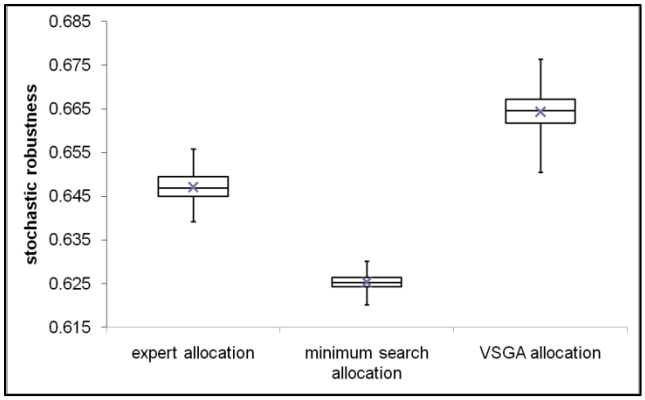

Figure 12. Stochastic robustness of the "expert", minimum search, and VSGA allocations for 50 trials.

As previously mentioned, experiments were conducted with varying population sizes for the VSGA in order to study the population size versus execution time tradeoff (Figure 13). It is clear that some improvement is gained between population size 20 and 50 (an average of 0.56%). However, there is no improvement between population sizes of 50 and 100. It is clear though that the average of three extra hours to complete the VSGA for 100 chromosomes compared to 50 chromosomes is not beneficial.

6. CONCLUSIONS

Determining resource allocations for military village search problems is a complex problem. Current solutions produced by planning officers are time consuming and of unquantifiable quality. We have presented the RoPARS tool for village search planning and two heuristics for use in its resource allocation engine. The heuristics demonstrate

that computer tools can create solutions for these problems and do so in a manner that is robust against uncertainty in the environment. This can save military planners time and resources during the planning process and improve the quality of the resulting plan.

Future work in this area includes increasing the size of the test scenarios in terms of target buildings and number of search teams. Additionally, experiments can be done on the village search genetic algorithm to determine the best number of strands to perform crossover and mutation upon per generation. Finally, additional research is needed to develop heuristics that can quickly and dynamically re-evaluate the resource allocation plan should conditions change from those used during static resource allocation.

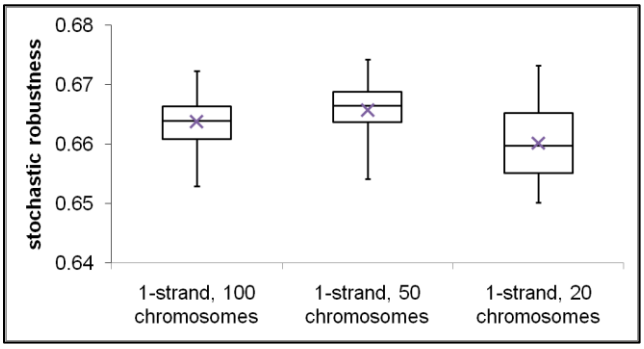

Figure 13. Village search genetic algorithm stochastic robustness for varying population sizes.

Acknowledgements. The authors would like to thank L. Briceño, Greg Pfister, and A. Al-Qa'wasmeh for their comments, and D. Young for use of his max. pmf code.

References

[1] S. Ali, A. A. Maciejewski, H. J. Siegel, and J. Kim, "Measuring the robustness of a resource allocation," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 15, No. 7, July 2004, pp. 630–641.

[2] S. Ali, A. A. Maciejewski, and H. J. Siegel, "Perspectives on robust resource allocation for heterogeneous parallel systems," in *Handbook of Parallel Computing: Models, Algorithms, and Applications*, S. Rajasekaran and J. Reif, eds., Chapman & Hall/CRC Press, Boca Raton, FL, 2008, pp. 41-1–41-30.

[3] T. Bektas, "The multiple traveling salesman problem: An overview of formulations and solution procedures," *Omega: The International J. of Management Science*, Vol. 34, 2006, pp. 209–219.

[4] T. Blickle and L. Thiele, *A comparison of selection schemes used in genetic algorithms*, Technical Report TIK-Report No. 11, Ver. 2, CEN Lab, Swiss Federal Institute of Technology, Dec. 1995, 67 pp.

[5] O. H. Ibarra and C. E. Kim. "Heuristic algorithms for scheduling independent tasks on non-identical processors," *J. of the ACM*, Apr. 1977, Vol. 24, No. 2, pp. 280–289.

[6] P. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic, "Genetic algorithms for the travelling salesman problem: A review of representations and operators," *Artificial Intelligence Review*, Vol. 13, No. 2, 1999, pp. 129–170.

[7] P. Maxwell, A. A. Maciejewski, H. J. Siegel, J. Potter, and J. Smith, "A mathematical model of robust military village searches for decision making purposes," *2009 Int'l Conf. on Information and Knowledge Engineering*, July 2009, pp. 311–316.

[8] P. Maxwell, H. J. Siegel, J. Potter, and A. A. Maciejewski, "The ISTeC People-Animals-Robots laboratory: Robust resource allocation," *2009 IEEE Int'l Workshop on Safety, Security, and Rescue Robotics*, Nov. 2009.

[9] *FM 3-31.1 Army and Marine Corps Integration in Joint Operations*, U. S. Army Training and Doctrine Command, Ft. Monroe, VA, Nov. 2001.

[10] *FM 34-8-2 Intelligence Officer's Handbook*, U.S. Army Training and Doctrine Command, Ft. Monroe, VA, May 1998.

[11] J. Potvin, "Genetic algorithms for the traveling salesman problem," *Annals of Operations Research*, Vol. 63, 1996, pp. 339–370.

[12] H. Saleh and R. Chelouah, "The design of the global navigation satellite system surveying networks using genetic algorithms," *Engineering Applications of Artificial Intelligence*, Vol. 17, 2004, pp. 111–122.

[13] V. Shestak, J. Smith, A. A. Maciejewski, and H. J. Siegel, "Stochastic robustness metric and its use for static resource allocations," *J. of Parallel and Distributed Computing*, Vol. 68, No. 8, Aug. 2008, pp. 1157–1173.

[14] L. Tang, J. Liu, A. Rong, and Z. Yang, "A multiple traveling salesman problem model for hot rolling scheduling in Shanghai Baoshan Iron & Steel Complex," *European J. of Operations Research*, Vol. 124, 2000, pp. 267–282.

[15] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *J. of Parallel and Distributed Computing*, Vol. 47, No. 1, Nov. 1997, pp. 8–22.

[16] Z. Yu, L. Jinhai, G. Gouchang, Z. Rubo, and Y. Haiyan, "An implementation of evolutionary computation for path planning of cooperative mobile robots," *4th World Congress on Intelligent Control and Automation*, June 2002, pp. 1798–1802.