7-2009

A hybrid genetic-greedy approach to the skills management problem.

Daniel Grieshaber 1986-
University of Louisville

A HYBRID GENETIC-GREEDY APPROACH TO THE SKILLS MANAGEMENT PROBLEM

By

Daniel Grieshaber
B.S., University of Louisville, 2008

A Thesis
Submitted to the Faculty of the
University of Louisville
J. B. Speed School of Engineering
in Partial Fulfillment of the Requirements
for the Professional Degree

MASTER OF ENGINEERING

Department of Computer Engineering and Computer Science

July 2009

A HYBRID GENETIC-GREEDY APPROACH TO THE SKILLS MANAGEMENT PROBLEM

Submitted by:_____
Daniel Grieshaber

A Thesis Approved on

_____
(Date)

by the Following Reading and Examination Committee:

_____
Dr. C. Tim Hardin, Thesis Director

_____
Dr. Gail DePuy

_____
Dr. Ming Ouyang

ACKNOWLEDGEMENTS

I would like to acknowledge Thesis Director Dr. Tim Hardin for posing the initial challenge: applying a genetic approach to the skills management problem. I also recognize all other students of Dr. Hardin's Artificial Intelligence class who provided ideas during brainstorming sessions. Finally, I appreciate the previous work done by Dr. Gail DePuy on the skills management problem which provided a basis for the work in this thesis.

ABSTRACT

The Naval Surface Warfare Center wishes to create a task assignment schedule with a minimal training cost for workers to raise their skills to the required levels. As the number of workers, skills, and tasks increase, the problem quickly becomes too large to solve through brute force. Already several greedy heuristics have been produced, though their performance degrades for larger data sets.

As Genetic Algorithms (GA) are effective for large combinatorial problems, their application to the task assignment problem may prove successful. The innovation in applying a GA to this problem is the utilization of existing greedy heuristics in the crossover operator. As the population begins to converge in the GA, the greedy algorithm benefits by having fewer tasks to assign. Likewise, the GA benefits from the addition of the greedy heuristic by increasing the likelihood of good valid solutions within the population.

To explore the effectiveness of the proposed method, several different crossover operators are defined. The first method is purely random to act as a control, as the only improvements will be due to the genetic algorithm. The second method provides a basic heuristic to improve upon the random crossover operator, while still primarily stochastic and therefore relying on the GA for convergent behavior. The final two techniques

incorporate existing greedy heuristics.

The four crossover operators are tested against several data sets of varying sizes to ascertain their relative performance. Crossover methods are compared based on the best score found over all runs. In addition, the evolution and convergence of populations for the different operators are examined, offering further insight into their performance.

The combination of a greedy heuristic and genetic algorithm proves to be an effective method for approaching the task assignment problem. This method compares favorably to existing techniques, as well as a purely genetic approach. While the greedy-genetic approach suffers some shortcomings, the success of the combined algorithm warrants further development of this methodology.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. BACKGROUND

As discussed in [3], the Crane Division, Naval Surface Warfare Center (NSWC) employs a large workforce to acquire and support a variety of electronic warfare devices and systems. In general NSWC wishes to retain its current workforce, so when making bids for work the cost to train current employees must be considered. While several greedy algorithms have been developed for minimizing this training cost [5], as the size of the problem increases, these methods prove inadequate. Genetic algorithms (GA) are a general technique used to solve large combinatorial problems, such as minimizing the cost of the NSWC workforce training schedule. This thesis work focuses on the implementation and analysis of a genetic algorithm that incorporates preexisting greedy algorithms to produce higher quality solutions to the workforce scheduling problem.

A. Problem Description

The NSWC scheduling problem is based around workers, tasks, and the skills possessed or required by each. For each skill, the competency of each worker is assessed. Likewise, the skill levels required by each task to complete it are also determined. When a worker is assigned a task, that worker must have an equal or greater skill level than

required by the task for each skill. If a worker is not qualified to complete the task assigned to it, the worker must undergo training, with an associated training cost. The goal is to assign all tasks such that this training cost is minimized. A formal description of the NSWC task assignment problem was originally developed by DePuy *et al* [3] and is presented here in Figure 1. The total training cost that is being minimized is listed as equation 1.

Parameters
{j} = set of skills needed to perform task j
S_{ik} = worker i's skill level for skill k
R_{jk} = required skill level for task j's skill k
T_j = length (# hrs) of task j
A_i = capacity (# hrs) of worker i
C_{klm} = cost associated with raising a worker's skill level on skill k from level l to level m
E_{klm} = time required (# hrs) to raise a worker's skill level on skill k from level l to level m

Decision Variables
X_{ij} = 1 if worker i assigned to task j
$Z_{ikS_{ik}m}$ = 1 if worker i receives training on skill k to raise skill level from S_{ik} to m
N_{ik} = 1 if worker i does not need further training in skill k

Objective Function

Minimize Training Cost Minimize $\sum_i \sum_k \sum_m C_{kS_{ik}m} Z_{ikS_{ik}m}$ (1)

Constraints

Determine Needed Training $S_{ik} N_{ik} + \sum_{m>S_{ik}}^{5} m Z_{ikS_{ik}m} \geq R_{jk} X_{ij} \quad \forall i,j,k \in \{j\}$ (2)

$N_{ik} + \sum_{m>S_{ik}}^{5} Z_{ikS_{ik}m} = 1 \quad \forall i,k$ (3)

All tasks assigned $\sum_i X_{ij} = 1 \quad \forall j$ (4)

Worker Capacity $\sum_j T_j X_{ij} + \sum_k \sum_m E_{kS_{ik}m} Z_{ikS_{ik}m} \leq A_i \quad \forall i$ (5)

Binary Variables $X_{ij} \in \{0,1\}, \; Z_{ikS_{ik}m} \in \{0,1\}, \; N_{ik} \in \{0,1\} \quad \forall i,j,k,m$ (6)

FIGURE 1 - Formal Description of the Skills Management Problem

There are several constraints for the task assignment problem. First, NSWC wishes to retain its current workforce, thus requiring each worker to be assigned at least one task. The second constraint limits how much time is available for a worker to train and perform tasks, denoted by a worker's capacity. Note that the time needed to train for a task is subtracted from a worker's total capacity, but is not relevant to the total training cost being minimized. The final constraint requires all tasks to be assigned to a worker.

B. Existing Techniques

1. Meta-RaPS Greedy Algorithm

As discussed by DePuy *et al.* in [2], Meta-RaPS (Meta-heuristic for Randomized Priority Search) is a high-level, stochastic technique used to improve greedy algorithms for combinatorial problems. By randomly allowing some less-than-optimal decisions in the execution of an algorithm, this heuristic helps avoid local optima and enables better solutions to be found. The Meta-RaPS technique has since been applied to the NSWC task assignment problem, where a greedy assignment algorithm is enhanced by the meta-heuristic [5]. The details of this algorithm are discussed below.

Since NSWC wishes to retain all current employees, the Meta-RaPS greedy algorithm first ensures each worker is assigned at least one task. During this first phase, the algorithm identifies the least skilled worker and assigns to it the least difficult task. A worker's skill is determined by summing the total training cost for all tasks, while the difficulty of a task is the total cost for all workers to train for that task. Until all workers

have a task, the worker with the maximum total training cost is assigned the task with the minimum training cost. The Meta-RaPS heuristic alters this phase of the greedy algorithm by enabling a more skilled worker to be assigned its minimum cost task. This is done through the use of an available list: any worker/task pairing whose training costs are within a certain range of the next greedy assignment are added to the list. A worker and task are then selected at random from the available list.

```
Calculate training cost for each worker over all tasks (total_worker_cost)
While (there are unassigned workers) {
        Find the worker with maximum total_worker_cost (max_cost_worker)
        Find task with minimum training cost for max_cost_worker (min_cost_task)
        P = Rand(1, 100)
        If (P <= %priority) {
                Assign min_cost_task to max_cost_worker
        }
        Else {
                Form available list of worker-task pairs such that the worker's
                total_training_cost is within %restriction of the max_cost_worker's cost
                and the cost of the associated minimum cost task is within %restriction of
                min_cost_task
                Randomly select a worker-task pair from the available list for the next
                assignment
        }
        Update skill set and capacity for worker based on requirements for the assigned
        task
        Update total_worker_cost
        Update total training cost for the solution
}
```

FIGURE 2 - Pseudocode for Meta-RaPS Greedy Algorithm Phase 1

The second phase of the Meta-RaPS greedy algorithm assigns the remaining tasks. In phase 2, the hardest task, *i.e.* the task with the greatest total training cost, is assigned to the worker that needs the least training for that task. As in phase 1, these greedy assignments are subject to randomization by Meta-RaPS. Again, this involves the

creation of an available list containing worker/task pairs within a percentage restriction of the next greedy assignment. By running many iterations of the Meta-RaPS algorithm, the randomizing elements provides basic search behavior, locating better solutions than the purely greedy algorithm [5].

```
Calculate training cost for each task over all workers (total_task_cost)
While (there are unassigned tasks) {
    Find task with maximum total_task_cost (max_cost_task)
    Find worker with minimum training cost and sufficient capacity for
    max_cost_worker (min_cost_worker)
    P = Rand(1, 100)
    If (P <= %priority) {
        Assign max_cost_task to min_cost_worker
    }
    Else {
        Form available list of worker-task pairs such that the task's
        total_task_cost is within %restriction of the max_cost_task's cost and the
        cost of the associated minimum cost worker is within %restriction of
        min_cost_worker
        Randomly select a worker-task pair from the available list for the next
        assignment
    }
    Update skill set and capacity for worker based on requirements for the assigned
    task
    Update total_worker_cost
    Update total training cost for the solution
}
```

FIGURE 3 - Pseudocode for Meta-RaPS Greedy Algorithm Phase 2

2. Meta-RaPS Regret Algorithm

Like the Meta-RaPS greedy algorithm, the modified Regret algorithm [7] incorporates randomized elements to provide search behavior. It is also a two phase algorithm, first ensuring all workers are assigned at least one task, then assigning all remaining tasks. The innovation of the algorithm is the concept of "regret": the difference in cost for a task being assigned to the worker with the minimal training cost for that task and the worker with the third lowest cost. Having a low regret factor, a task may be deferred assignment for several iterations without negatively impacting the overall score. However a high regret task would significantly degrade the solution if not assigned quickly. The overall solution may therefore be optimized by giving priority to tasks with higher regret factors. This is the motivation for the Regret algorithm.

The first phase of the algorithm ensures all workers are assigned one task, and begins by forming a list of possible tasks to assign. The tasks that have a minimal training cost over all unassigned workers are considered for the list, with the size equal to the number of unassigned workers, as well as any tasks whose cost is within *%restriction*. This list is randomly culled so that there is a task for each unassigned worker. The tasks are then ordered by regret, so that the highest regret tasks are assigned to their lowest cost workers first. As in the previous algorithm, this greedy selection is augmented by the Meta-RaPS heuristic: some iterations the highest regret task is not assigned. Instead a list of tasks within *%restriction* of the highest regret task are randomly sampled for the next assignment. This can be seen in the pseudocode for the first phase below.

```
n = number of unassigned workers
Calculate training cost for each task over all workers (total_task_cost)
Order tasks from smallest to largest total_task_cost
Form available list of tasks within %restriction of n^th smallest total_task_cost
Randomly choose n tasks from available task list (phase1_task_list)
While (there are unassigned workers) {
        For Each (task in phase1_task_list) {
                Find 3 smallest smallest cost unassigned workers with sufficient capacity
                for task
                Calculate regret as the difference between the training cost for the smallest
                cost worker  and the third smallest cost worker
        }
        Find task with maximum regret (max_regret_task)
        P = Rand(1, 100)
        If (P <= %priority) {
                Assign max_regret_task to its minimum cost worker
        }
        Else {
                Form available list of tasks with a regret within %restriction of
                max_regret_task
                Randomly select task from available list and assign to its minimum cost
                worker
        }
        Update skill set and capacity for worker based on requirements for the assigned
        task
        Update worker_task_costs
        Update total training cost for the solution
}
```

FIGURE 4 - Pseudocode for Meta-RaPS Regret Algorithm Phase 1

After assigning all workers a task, the algorithm enters the second phase. All remaining tasks are ordered by regret, with the highest regret task being assigned first. Like phase 1, this task is assigned its lowest cost worker. Again, this selection is subject to randomization by Meta-RaPS. The details of the second phase of the regret algorithm are presented as pseudocode below.

```
While (there are unassigned tasks) {
        For Each (unassigned task) {
                Find 3 smallest smallest cost unassigned workers with sufficient capacity for
                task
                Calculate regret as the difference between the training cost for the smallest
                cost worker  and the third smallest cost worker
        }
        Find task with maximum regret (max_regret_task)
        P = Rand(1, 100)
        If (P <= %priority) {
                 Assign max_regret_task to its minimum cost worker
        }
        Else {
                Form available list of tasks with a regret within %restriction of
                max_regret_task
                Randomly select task from available list and assign to its minimum cost
                worker
        }
        Update skill set and capacity for worker based on requirements for the assigned task
        Update worker_task_costs
        Update total training cost for the solution
}
```

FIGURE 5 - Pseudocode for Meta-RaPS Regret Algorithm Phase 2

C. Genetic Algorithms

Genetic Algorithms are a category of stochastic search techniques that emulate biological evolution [1]. Possible solutions are abstracted as members of a population,

where individuals compete for reproduction as well as survival to future generations. The probability of these events is determined by an individual's fitness: fitter individuals are favored for reproduction and are more likely remain in the population. This selective pressure tends the population toward better solutions, while the stochastic component counteracts the tendency toward local optima.

Once created, the population within a GA is refined through several basic steps. First pairs of individuals are selected for reproduction. While higher fitness is favored, stochastic selection techniques allow less fit individuals to reproduce, thus promoting diversity in the population and avoiding convergence on local optima. Second, children are produced through crossovers, *i.e.* components from each parent are recombined to form novel solutions. Additionally some implementations introduce random mutations at this stage to promote population diversity. Finally, the children are introduced into the population, and the least fit individuals are removed to maintain a constant population size. These steps are repeated until some termination condition is met (*e.g.* predefined running time or limited number of generations). Pseudocode for this generic genetic algorithm is listed below.

```
Create initial population
While (termination conditions are not met) {
        Select pairs of individuals for reproduction (parents)
        Cross parents to produce new solutions (children)
        Add children to the population
        Remove excess individuals from the population
}
```

FIGURE 6 - Pseudocode for a Generic Genetic Algorithm

D. Motivation

Though current methods perform well for smaller data sets, performance degrades as the number of workers, skills, and tasks increase. Indeed even a small increase in problem size causes an exponential increase in the search space. This is typical for problems that fall into the NP-complete category, *i.e.* problems that cannot be solved in polynomial time. However, genetic algorithms are known to perform well with NP-hard problems, assuming the problem can easily be abstracted into the GA framework [6].

Additionally, the current greedy algorithms rely on many iterations to produce good solutions. As the problem size increases, so does the time for each iteration, limiting the number of solutions that can be produced. In contrast, genetic algorithms produce new solutions by simply recombining elements from two existing individuals. Since not all of the tasks are reassigned when two parents are crossed, the time to create the new

solution is significantly less. By producing more possible solutions, a genetic algorithm could search more of the problem space.

An obstacle for using a purely genetic approach is the small ratio of valid to invalid solutions. There are many combinations of workers and tasks that violate the constraints of the problem. Without a heuristic to provide some guidance, a genetic algorithm may run without finding a single viable solution. In contrast, the Meta-RaPS and Regret algorithms ensure viable, if not optimal, solutions.

To benefit from the advantages of both possible approaches, a greedy algorithm is incorporated into the framework of a genetic algorithm as the crossover operator. This technique hopes to combine the reliable and effective local search of the current greedy algorithms with the robust global search capabilities of a genetic algorithm. The problem of inviable solutions in the genetic population is removed by using the greedy algorithm to initialize the population and create new individuals. Likewise, solutions created using crossovers inherit some of their assignments, reducing the computations required by the greedy algorithm.

II. HYBRID GENETIC-GREEDY ALGORITHM

As mentioned in the previous section, genetic algorithms are comprised of three basic steps: (1) selecting individuals from the population to reproduce, (2) generating new solutions through crossover and mutation operations, and (3) removing excess individuals from the population. The specific implementations for each of these components is discussed in detail below.

A. High Low Fit Selection

Several different parent selection methods were explored during the initial development of the genetic algorithm, including common techniques such as roulette-wheel and tournament selection. These methods proved inadequate, as population diversity collapsed quickly, perturbing the search behavior of the algorithm. Maintaining population diversity became a primary motivator for electing a selection method. As mentioned in [1], the HighLowFit selection method preserves population diversity over successive generations better than other common techniques. For this reason, HighLowFit is used to select parents.

The HighLowFit algorithm maintains population diversity by ensuring one of the parents has a relatively low level of fitness. This is accomplished by first ordering the

population based on fitness, *i.e.* training cost. The sorted population is then partitioned into two groups, representing individuals with high and low fitness levels. The separation point between these groups is defined using a percentage value, which can be varied to alter the selection pressure of the algorithm. A parent is then chosen at random, uniformly, from each partition. The simplicity of this algorithm ensures fast execution as an added benefit to the superior performance compared to other selection methods.

The division point, as already mention, affects the performance of the HighLowFit selection method. Lower values reduces the number of highly fit individuals, increasing the frequency that these solutions are selected for reproduction. This causes the population to converge, as genes from fit individuals are represented more. So while convergence is improved by lowering the partition percentage, it is done so at the cost of population diversity. Therefore the high-low division point must be carefully selected to balance convergent behavior and preservation of population diversity. Experimentation found that a division point of 15% worked best to strike this balance for the task assignment problem.

B. Greedy Algorithms as Crossover Operators

Crossover operations recombine genes from the selected parents to create new solutions. By inheriting genes, children benefit from the collective advancement of the population. For the task assignment problem, a worker-task pairing is considered a gene, as it is the most basic component of a solution. The crossover operation begins by first

comparing parent solutions to find worker-task pairings occurring in both. These genes are preserved in the child, providing the basis for a new solution. A greedy algorithm can then be used to complete the remaining task assignments.

Previously developed algorithms for the task assignment problem rely on many iterations to take advantage of the randomizing elements in finding better solutions. Each iteration of the algorithm makes every assignment and does not benefit from information learned in previous iterations. By incorporating these algorithms in a crossover operator, inherited genes allow some "memory" of previous generations. These genes act as fixed assignments, meaning each crossover operation does not require all tasks to be assigned as in each iteration of the original algorithms. In fact, as the population converges, the number of new assignments made during crossover operations decreases as successful worker-task pairings begin to dominate the population.

While this crossover method reduces the number of new assignments per iteration, and therefore reduces execution time, it can easily become stuck at local optima. Successful worker-task pairings quickly spread through the population causing population diversity to collapse. This behavior is counteracted through the use of a mutation operator. A mutation simply removes some inherited genes from a child, allowing the greedy algorithm to reassign those tasks. A sufficient mutation rate ensures genes cannot completely dominate the population, promoting diversity.

The proposed crossover operator is flexible in that any greedy algorithm may be used to fill in missing task assignments. Indeed, the algorithm need not even be greedy. Exploring the use of different algorithms within the crossover operator is the primary

focus of this thesis. Does a genetic approach to the task assignment problem benefit from using existing greedy algorithms, or do stochastic methods provide better results? Four different assignment algorithms are considered for this analysis and their details are provided below.

C. Crossover Algorithms Analyzed

1. Random

To provide a baseline for comparing the other methods, a purely random algorithm is implemented. Like all the algorithms used, the random algorithm consists of two phases: the first phase ensures each worker is assigned at least one task, while the second phase assigns all the remaining tasks. Without a two phase process, the performance of the algorithm is greatly inhibited as most solutions produced are invalid. In both phases, an unassigned worker (phase 1) or an unassigned task (phase 2) is chosen uniformly at random. An accompanying task/worker is then randomly selected, allowing for several attempts to find a matching where the worker's capacity is not exceeded. As this is the fastest of the four crossover methods, even allowing for a very large number of attempts does not negatively impact running time.

This method may produce invalid solutions, so the genetic algorithm must compensate by applying a penalty for any workers exceeding their capacity. Since no heuristic is used by this assignment algorithm, any improvements to solutions are the

result of the genetic algorithm, thus providing a control to measure the effectiveness of the other methods.

2. Roulette Wheel

The second algorithm incorporates a simple heuristic to improve upon the purely random method. During the first phase, a worker is randomly selected as before. But instead of randomly selecting a task, a list is created with all tasks that will not exceed the worker's capacity. A roulette wheel based on training cost is then used to select a task. This favors lower cost assignments, but allows for suboptimal assignments to be made, possibly leading to a better overall solution. Similarly, after randomly selecting an unassigned task in the second phase, workers with sufficient capacity are ordered based on training cost for the task, and one is chosen with a roulette wheel. If the available list is empty in either phase (*i.e.* no task can be assigned without exceeding the worker's capacity), a worker or task is selected uniformly at random. Use of the roulette-wheel provides a better heuristic for making assignments than the purely random approach and is therefore expected to have better performance.

3. Meta-RaPS Regret

The regret algorithm is the first of the greedy algorithms adapted for use within the crossover operator, which required several changes to the original algorithm. First, the regret calculation is altered. Instead of always using the third best assignment, a percentage value is translated into an index into the available worker list. The regret can

then be calculated by finding the difference between this worker and the lowest cost worker. Using a percentage value allows the algorithm to adapt as the number of fixed assignments changes during the evolution of the population. Preliminary tests found that this method, using a percentage value of 50%, worked better within the GA than the original regret calculation.

The second change to the regret algorithm is how the tasks are initially ordered. The original algorithm sorts unassigned tasks based on the total cost for all workers to train for each task. The implementation for the crossover operator only considers the least cost worker for this ordering. This change does not reduce the performance of the algorithm within the GA, but reduces the complexity and execution time.

TABLE I

PARAMETERS OF META-RAPS REGRET ALGORITHM

	Phase 1	Phase 2
%priority	50%	70%
%restriction	30%	70%

4. Meta-RaPS Greedy

The second greedy algorithm adapted for use in the crossover operator is the Meta-RaPS algorithm. In the first phase of the original Meta-RaPS, workers are ordered based on the total training cost over all tasks. Like the regret algorithm, only the maximum training cost is used to order the workers. Similarly, during the second phase, tasks are ordered by the maximum cost worker, rather than summing the cost of all

workers. As in the regret algorithm, these changes were not found to reduce the quality of solutions produced by the GA, but improved the running time. The *%priority* and *%restriction* parameters used for each phase within the crossover operator are included below.

TABLE II

PARAMETERS FOR META-RAPS GREEDY ALGORITHM

	Phase 1	Phase 2
%priority	75%	95%
%restriction	58%	25%

D. Culling the Population

The final component of the genetic algorithm removes excess individuals to maintain a constant population size across generations. A common technique for doing so is discussed in [1]: after adding the newly created solutions, the population is sorted based on fitness. By simply discarding the tail (*i.e.* the least fit individuals), the population size is maintained. Applied to the task assignment problem, this method applies too much selection pressure for the fittest individuals and population diversity quickly suffers. To reduce this selection pressure, not all individuals are removed from the end of the sorted population. A percentage of all removals are done at uniformly at random. This change allows some unfit solutions to remain, preserving diversity, without substantially increasing execution speed. Note that this culling methodology is not unlike the Meta-RaPS heuristic: a purely greedy removal technique is subject to randomization

to improve performance, though during this randomization, no priority is given to lower cost solutions.

E. Testing Proposed Crossover Operators

The four crossover methods are tested against several data sets of varying sizes, as indicated by the number of workers, skills, and tasks. This is reflected by the allowable running time for each data sets, as the smallest data set is stopped after only 15 minutes while the largest data set continues for 10 hours. Additionally, since smaller data sets require less run time, several iterations are completed to better ascertain the relative performance of the different methods. The stochastic nature of these algorithms leads to varying performance. These iterations help to eliminate this variability. While time did not permit multiple iterations for the largest data sets, differences in performance for the algorithms tested become more pronounced as the problem size increases. Therefore reasonable conclusions can be reached for these larger data sets without the benefit of multiple runs. The size, run time, and number of iterations for the data sets tested are presented in the table below.

TABLE III

DATA SETS TESTED

Data Set	Number of Workers	Number of Skills	Number of Tasks	Run Time (hours)	Number of Runs
1	9	11	13	0.25	26
2	11	13	44	2	8
3	30	20	100	5	5
4	50	50	100	6	4
5	50	50	220	8	1
6	100	40	400	10	1

III. IMPLEMENTATION

The algorithms discussed in this thesis were implemented using the Java programming language. The Eclipse integrated development environment provided a multi-platform tool for building and testing the Java program.

A. Input Data

The task assignment Java program provides a command line interface that accepts a single argument: the name of the execution configuration file. This file contains an "execution set". Included are the directory to store output files, the directory and file names containing data sets, run times for each data set, and the crossover operators to be tested. This enables multiple runs to be initialized and then left to run overnight. As most data sets were run for several hours, the run times represent hours of execution time. However, these values are read as floating point numbers, enabling shorter run times for small data sets. The crossover operator names are read as strings, which are then used to dynamically create class instances using reflection. The file format for data sets is discussed below. The name of the crossover operators, as well as the data set, are used to create unique file names for all runs.

The first three lines of an input data file identify the number of workers, skills,

and tasks for the data set. This is followed by two skill level matrices: each row of the first matrix corresponds to a worker while the second matrix lists the requisite skill levels to complete each task. These matrices are followed by the time required for each task, then the total capacity for each worker. The final two matrices specify the cost and time to train to increase each skill for all skill levels. All data is delimited by whitespace making input data files easily human-readable, while remaining simple to parse using regular expressions.

The input data files are read in by the `ProblemSet` class. By storing the problem data in a single static class, the memory required is minimized. The matrices are stored in arrays which enable access to problem data in constant time. Interfacing with the `ProblemSet` class is done primarily by the `Worker` and `Task` classes, which are responsible for calculating training cost, remaining worker capacity, and so on. The impetus for using a global data store is clear considering the number of workers and tasks for larger data sets. If each contained all necessary data, the memory requirement would grow quickly as the number of tasks and workers increases.

B. Greedy Scheduler Interface

The four crossover operators inherit from a base class, `GreedyScheduler` (so named for the original algorithms which scheduled tasks in a greedy manner). This base class provides two benefits. The first is the inclusion of common functionality used by all the crossover techniques. For example, before beginning any of the two-phase assignment algorithms, all unassigned workers and tasks must first be identified. A

method for doing so is provided by the base class. Another example is the need to copy the list of workers passed in so that the original is unmodified. Again a method is supplied to do so.

The second benefit to the use of the `GreedyScheduler` base class is to provide a common interface for use inside the genetic algorithm. All subclasses must implement the `schedule()` method, which accepts a list of workers and returns a complete solution. The workers may already have some tasks assigned to them, thus acting as fixed assignments. This enables the scheduling algorithms to "fill out" the remaining assignments after a child solution inherits some worker-task pairs from its parents. A `getName()` method is also required, enabling different crossover techniques to be identified dynamically in filenames, *etc*. By providing a common interface, use of a base class enables the crossover technique to easily be changed.

IV. RESULTS

A. Comparison of Solution Quality

After testing each crossover technique against all data sets, the best solutions found are recorded and presented in the table below. As hypothesized, the random crossover method has the worst performance overall, while the roulette wheel method had the second worst performance. These results indicate that the task assignment problem greatly benefits from a problem-centric heuristic. Of the two pre-existing techniques, the Meta-RaPS algorithm performed best as a crossover operator. Note that for the largest data set, the regret algorithm could not be run due to memory limitations of the Java Virtual Machine.

TABLE IV

BEST SOLUTION FOUND FOR EACH CROSSOVER METHOD
(*KNOWN OPTIMAL SOLUTION)

Data Set	Random GA	Roulette Wheel GA	Meta-RaPS Regret GA	Meta-RaPS Greedy GA
1	552	555	551*	551*
2	1411	1387	1268	1262
3	6712	5173	3416	2825
4	23784	22260	20533	18217
5	33529	25499	19448	16755
6	52226	35167	–	19041

While the random crossover technique generally performed poorly compared to other methods, it did manage to surpass the roulette wheel algorithm on the smallest data set. In fact, the best solution found by the random method is only one more than the optimal solution. Analyzing the mean and standard deviation for the best solutions from all runs further highlights the unusually good performance of the random crossover method on the smallest data set. These results are presented in the table below.

TABLE V

DISTRIBUTION OF TRAINING COSTS AFTER 26 RUNS ON DATA SET 1

	Mean	Standard Deviation
Random GA	554.35	1.70
Roulette Wheel GA	560.65	5.40
Meta-RaPS Regret GA	560.85	7.52
Meta-RaPS Greedy GA	551.00	0.00

Notice that the mean score for the random method is better than both the roulette

wheel and regret algorithms. Additionally, the random method had a much lower variability in solution quality compared to those other techniques. The reason for this may be due to the small problem size: using a purely random crossover technique allows for a much broader search of the solution space. This would explain why the initially reasonable performance quickly degrades with increasing problem size.

Also of note is the performance of the two pre-existing algorithms as crossover methods. While both found the optimal solution, their overall performance is hardly comparable. The regret algorithm had the worst average performance overall, as well as the largest variability in solution quality. In contrast, the Meta-RaPS algorithm located the optimal solution every run, giving it the best average performance and lowest variability.

Although the regret algorithm performed inconsistently for the smallest data set, as the problem size increases, so does the relative performance of the algorithm. The distribution of results for the second data set bears this out, as is evident in the table below. Already the average relative performance of the four crossover methods begin to differentiate, a trend that continues with increasing problem sizes.

TABLE VI

DISTRIBUTION OF TRAINING COSTS AFTER 8 RUNS ON DATA SET 2

	Mean	**Standard Deviation**
Random GA	1457.86	29.55
Roulette Wheel GA	1439.50	48.77
Meta-RaPS Regret GA	1305.13	25.96
Meta-RaPS Greedy GA	1273.50	8.50

The Meta-RaPS crossover operator consistently found better solutions than any other method. To provide a broader view of the algorithm's performance, the results for the original Meta-RaPS greedy algorithm are provided below [5]. Again, the Meta-RaPS crossover operator provides smaller training costs for all data sets. Clearly the combination of a genetic algorithm with a greedy heuristic proves more successful than either individually.

TABLE VII

COMPARISON OF META-RAPS GREEDY AND GENETIC ALGORITHHMS

Data Set	**MR Greedy**	**MR Greedy GA**
1	558	551
2	1462	1262
3	3202	2825
4	19436	18217
5	18799	16755
6	20510	19041

B. Comparison of Convergence

The previous section analyzed the relative performance of the four crossover methods by comparing the best solutions found for each data set. The performance of these algorithms may also be compared based on the convergence of their population. Observing the evolution of a population's fitness for each method provides further insight into the performance of these algorithms. Tracking the best, worst, and median solutions

in the population over time provides a visual indication of the distribution of population fitness.

1. Random

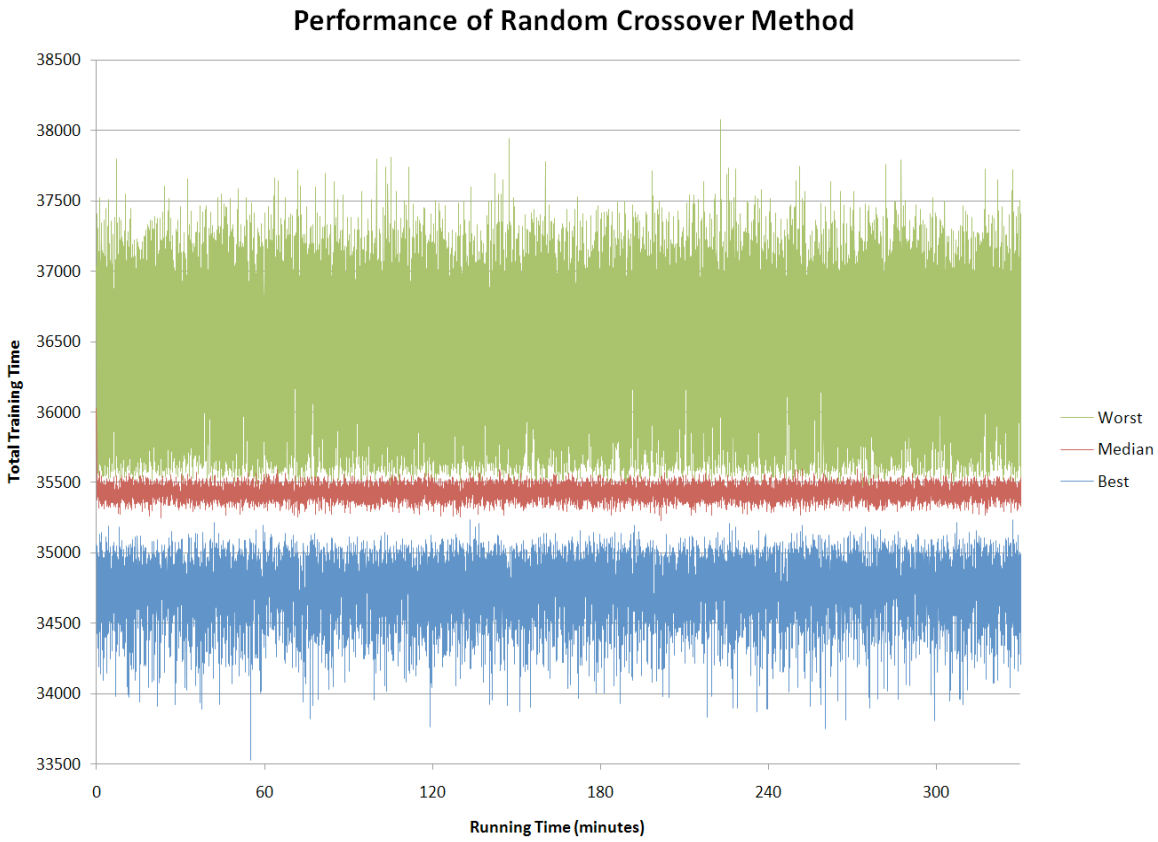

FIGURE 7 - Performance of Random Crossover Operator on Data Set 5

The first crossover method considered is the random algorithm, as seen in the preceding graph. Compared to the other methods discussed below, the random algorithm stands out by failing to exhibit any convergent behavior. The worst, median, and best solutions in the population stay well differentiated during the execution of the genetic algorithm, a likely cause for the lackluster performance of this crossover method. These

results are somewhat unexpected, as the genetic algorithm exerts some selection pressure, though the population never begins to converge. This is further indication that a purely genetic approach to the task assignment problem is inadequate for producing high quality solutions.

2. Roulette Wheel

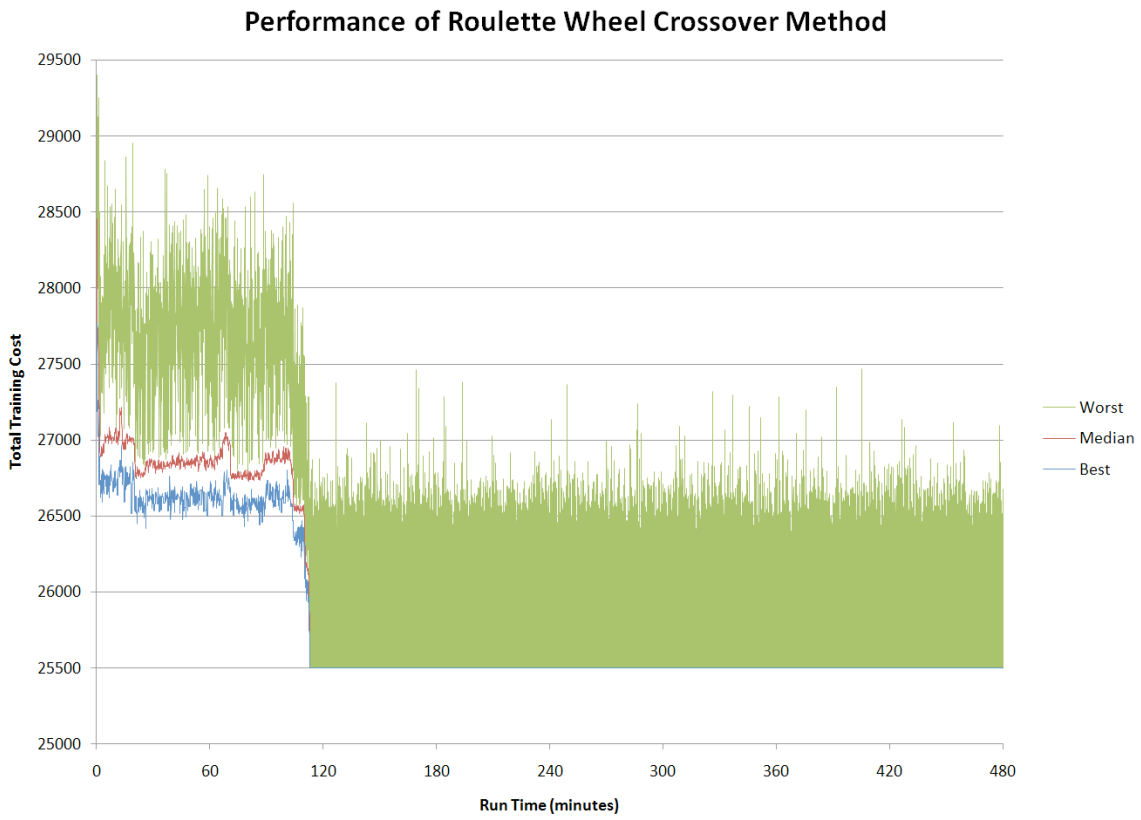

FIGURE 8 - Performance of Roulette Wheel Crossover Operator on Data Set 5

The roulette wheel crossover method is analyzed next. At first the best, worst, and median solutions remain well differentiated like the random method. However, once the fitness of the best solution crosses a certain threshold after approximately 100 minutes,

the population rapidly converges. After roughly 2 hours of run time, the diversity of the population collapses as the majority of individuals are duplications of the best solution. The worst solution in the population continues to fluctuate past this point, but it is not enough to encourage further diversity. Collapse of population diversity is the primary shortcoming of the techniques tested: once this occurs, the algorithms have difficulty finding better solutions, and often become permanently stuck at the local optima. Indeed, for the roulette wheel crossover method, no improvements are made to the best solution after the collapse of diversity.

3. Meta-RaPS Regret

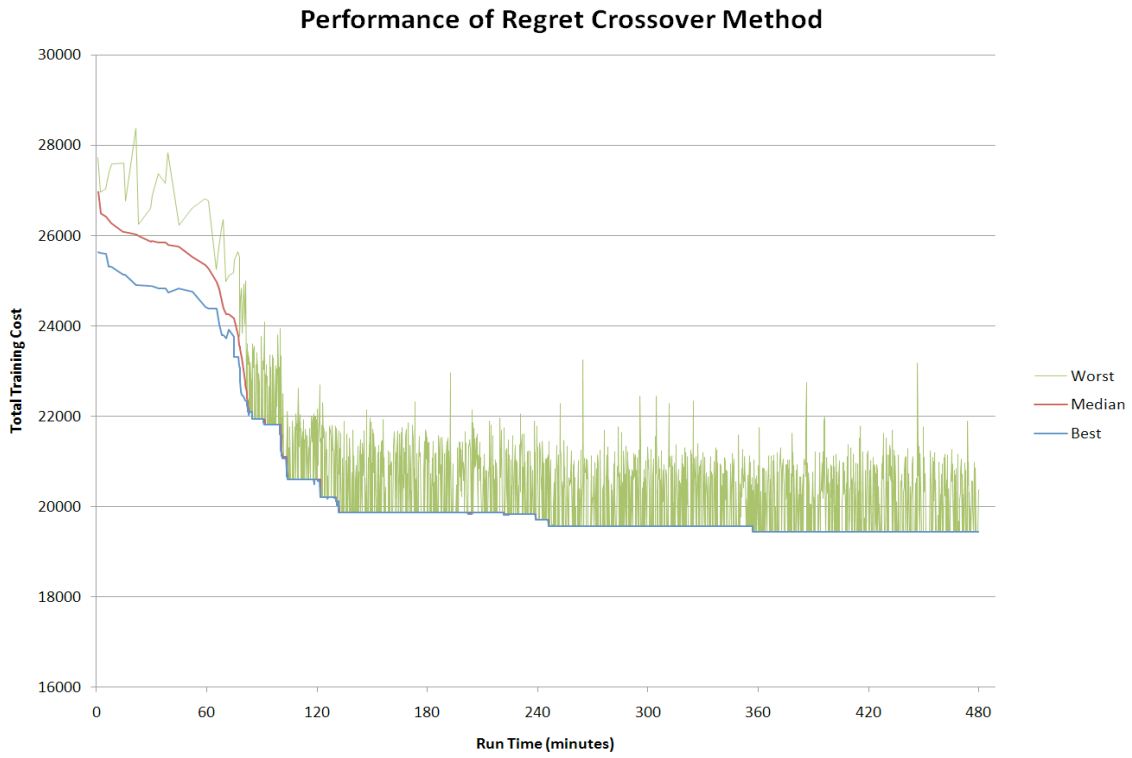

FIGURE 9 - Performance of Meta-RaPS Regret Crossover Operator on Data Set 5

The convergent behavior of the regret algorithm is now examined. In contrast to the previous crossover methods, the regret algorithm exhibits convergent behavior from the beginning. The best and median cost solutions do not fluctuate, but have a clear downward trend as the population tends toward better solutions. Like the roulette wheel method, the population diversity collapses less than two hours after initialization. However, the regret algorithm continues to make some improvements to solution quality without the benefit of a diverse population. This is likely the result of having a problem-centric heuristic in the crossover operator, which is capable of search behavior relatively independent of the genetic algorithm.

4. Meta-RaPS Greedy

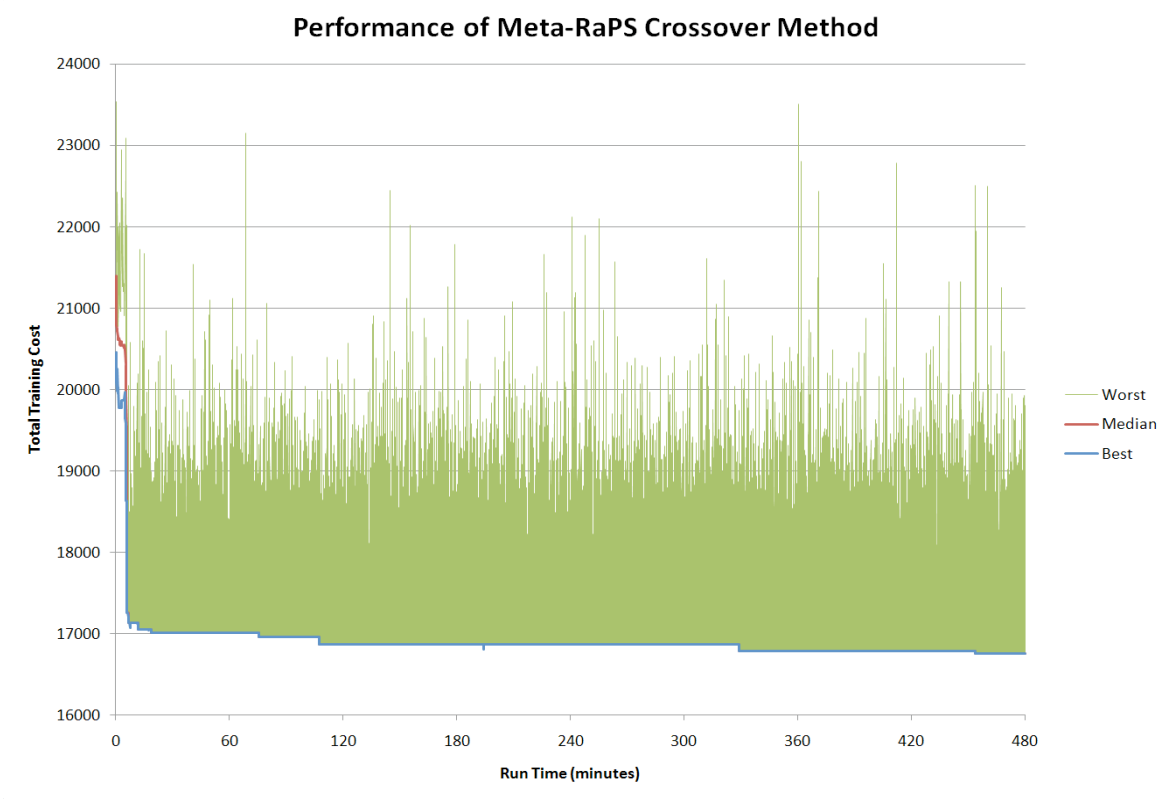

FIGURE 10 - Performance of Meta-RaPS Greedy Algorithm on Data Set 5

Finally, the Meta-RaPS crossover method is analyzed. As with the regret algorithm, Meta-RaPS causes the population to begin converging immediately, though this convergence is much more rapid than any other method. While the roulette wheel and regret algorithms took over an hour to decimate population diversity, the Meta-RaPS algorithm reaches this collapse after only five minutes of run time, as seen in the truncated graph below. Even so, the algorithm continues to find better solutions, with the last best solution found seven and half hours after beginning. Again this is likely the result of utilizing a problem-centric heuristic in the crossover algorithm.

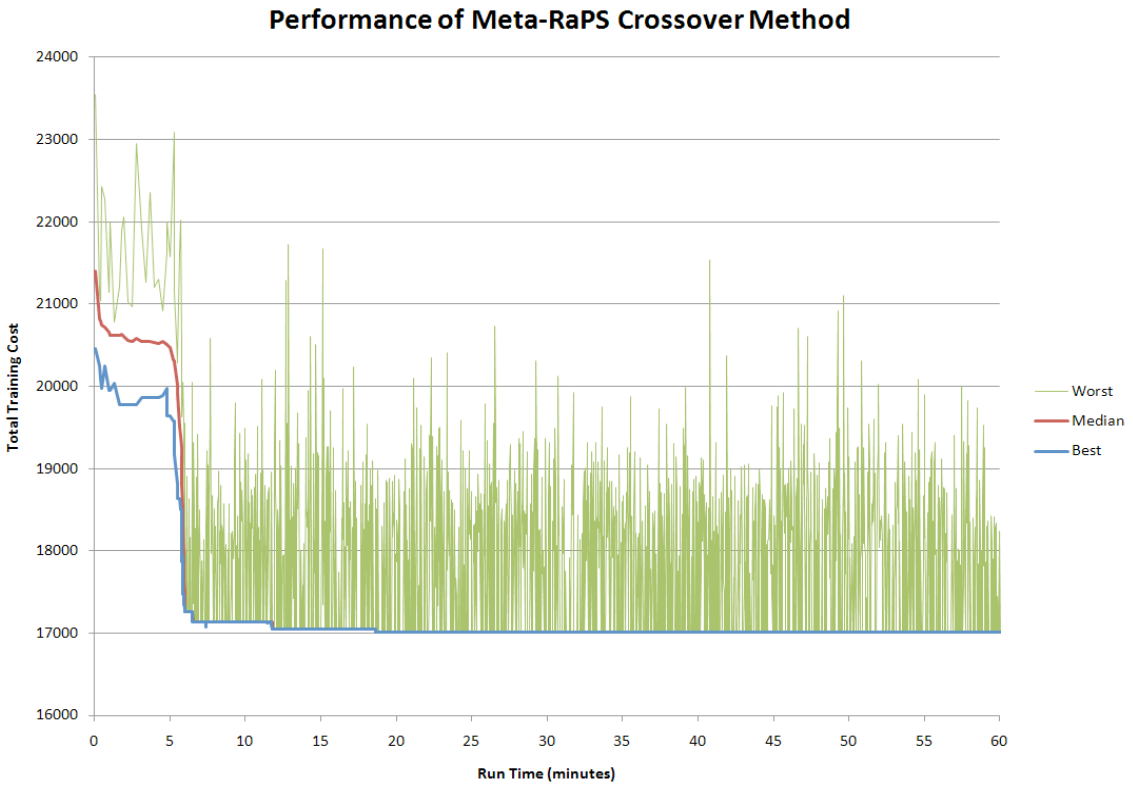

FIGURE 11 - Performance of Meta-RaPS Greedy Crossover Operator for First Hour

V. CONCLUSIONS

The proposed combination of a greedy heuristic with a genetic algorithm led to the development of several crossover operators. A purely random method acts as a control, providing a basis of comparison. The roulette wheel technique incorporates a limited stochastic heuristic, while the final two methods utilize the Meta-RaPS and Regret greedy algorithms.

Comparison of these crossover operators proved that the combination of a greedy heuristic and genetic algorithm provides better solutions than merely a genetic approach. Additionally, the top-performing Meta-RaPS genetic algorithm consistently produced lower training costs than the original Meta-RaPS greedy algorithm. These results indicate that the combination of a greedy heuristic and genetic algorithm is a better approach than either technique used individually.

In addition to comparing solution quality, the convergent behavior of each crossover operator is analyzed. The best, median, and worst cost solutions within the population are graphed over time. The random operator lacked any convergent behavior, but maintained population diversity. The roulette wheel did not initially exhibit convergence. Once started, though, the population quickly succumbed to collapse of diversity, halting further improvements to the best solution. The greedy heuristics began

converging immediately and also suffered from diversity collapse. However, the greedy methods continued to improve solution quality despite little diversity in the population.

The collapse in population diversity is the primary shortcoming of the combined algorithm. After the collapse of population diversity, improvements upon the best solution are greatly perturbed. Further investigation into maintaining this diversity may lead to better performance with a more robust search that is less likely to become stuck at local optima.

Another shortcoming of this implementation is the use of static values for the *%priority* and *%restriction* values. The setting of these parameters greatly affects the performance of the Meta-RaPS crossover operator, with the optimal values dependent on the problem size. Therefore it may be beneficial to set these values dynamically, based on the number of fixed assignments. Stricter parameters could be used to initialize the population of the genetic algorithm, ensuring reasonably good starting solutions, while looser parameters would allow for more search behavior once the population had sufficiently converged. Defining the *%priority* and *%restriction* values as a function of the population diversity or number of inherited genes would be a possible avenue of further research.

REFERENCES

1. Ali, E.E.E., 2006. "A Proposed Genetic Algorithm Selection Method." King Saud University, ccis. http://docs.ksu.edu.sa/PDF/Articles44/Article440965.pdf. Last Accessed 26 January 2009.

2. DePuy, G.W., G.E. Whitehouse, and R.J. Moraga, 2002. "Using The Meta-Raps Approach to Solve Combinatorial Problems," CD-ROM *Proceedings of the 2002 Industrial Engineering Research Conference*, May 19-21, Orlando, Florida, 6 pages.

3. DePuy G.W., J.S. Usher, B. Arterburn, R. Walker, and M. Fredrick, 2006. "Workforce training schedule for logistics skills," CD-ROM *Proceedings of the 2006 Industrial Engineering Research Conference*, May 20-24, Orlando, Florida, 6 pages.

4. DePuy, G.W., D. Grieshaber, and C.T. Hardin, 2009. "Skills Management Assignment Problem with Dynamic Costs," *Proceedings of the 2009 Industrial Engineering Research Conference*, May 30-June 3, 2009, Miami, Florida, 6 pages.

5. Jackson, E., G.W. DePuy, and G.W. Evans, 2008. "Logistics Skills Management Heuristics," CD-ROM *Proceedings of the 2008 Industrial Engineering Research Conference*, May 17-21, 2008, Vancouver, British Columbia, Canada, 6 pages.

6. K.A. De Jong, W.M. Spears. "Using Genetic Algorithms to Solve NP-Complete Problems." *Proceedings of the 3rd International Conference on Genetic Algorithms.* June 4-7, 1989, Morgan Kaufmann, Publishers.

7. S. Martello, P. Toth (1981c). An algorithm for the generalized assignment problem. In J.P. Brans (ed.), *Operational Research '81*, North-Holland, Amsterdam, 589-603.

CURRICULUM VITAE

Daniel Grieshaber

Date of Birth	February 8, 1986
Place of Birth	Kingston, NY
Undergraduate Study	University of Louisville B.S. in Computer Engineering and Computer Science 2004 – 2008
Graduate Study	University of Louisville M. Eng. in Computer Engineering and Computer Science 2008 – 2009
Experience	Course Lecturer, University of Louisville (May 2009 – July 2009) Teaching Assistant, University of Louisville (January 2009 – April 2009) Junior Software Developer Co-op Stonestreet One – Louisville, KY and Carlsbad CA (January 2005 – August 2007)