6-2007

A modified greedy algorithm for the task assignment problem.

Allison M. Douglas
University of Louisville

A MODIFIED GREEDY ALGORITHM FOR THE TASK ASSIGNMENT PROBLEM

By

Allison M. Douglas
B.S., University of Louisville, 2006

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 Industrial Engineering

June 2007

A MODIFIED GREEDY ALGORITHM FOR THE TASK ASSIGNMENT PROBLEM

Submitted by: _____
Allison M. Douglas

_____
(Date)

By the Following Reading and Examination Committee:

_____
Dr. Gail W. Depuy, Thesis Director

_____
Dr. John S. Usher

_____
Dr. Eric C Rouchka

ACKNOWLEDGEMENTS

Thank you to Marla Fredrick, Barbara Strahley and Randy Walker at Crane Division, Naval Surface Warfare Center (NSWC) for their assistance providing real-world information regarding the task allocation problem and the sample data sets used for analysis in this study.

I would also like to acknowledge the members of my thesis committee, Dr. J. S. Usher and Dr. E. C. Rouchka for their contribution. A special thank you to my thesis advisor, Dr. G. W. Depuy for her guidance.

ABSTRACT

Assigning workers to tasks in an efficient and cost effective manner is a problem that nearly every company faces. This task assignment problem can be very time consuming to solve optimally. This difficulty increases as problem size increases. Most companies are large enough that it isn't feasible to find an optimal assignment; therefore a good heuristic method is needed. This project involved creating a new heuristic to solve this problem by combining the Greedy Algorithm with the Meta-RaPS method. The Greedy Algorithm is a near-sighted assignment procedure that chooses the best assignment at each step until a full solution is found. Although the Greedy Algorithm finds a good solution for small to medium sized problems, introducing randomness using the meta-heuristic Meta-RaPS results in a better solution. The new heuristic runs 5000 iterations and reports the best solution. The final Excel® VBA program solves a small sized problem in less than one minute, and is within 10% of the optimal solution, making it a good alternative to time consuming manual assignments. Although larger, more realistic problems will take longer to solve, good solutions will be available in a fraction of the time compared to solving them optimally.

TABLE OF CONTENTS

APPROVAL PAGE ii
ACKNOWLEDGEMENTS iii
TABLE OF CONTENTS v
LIST OF TABLES vi
LIST OF FIGURES vii
I. INTRODUCTION 8
II. PROBLEM DESCRIPTION 10
III. BACKGROUND 15
IV. WORKERSKILLS ASSIGNMENT PROCEDURE 17
V. RESULTS 24
VI. CONCLUSIONS AND RECOMMENDATIONS 32
REFERENCES 35
APPENDIX A – TEST DATASET 1 36
APPENDIX B – TEST DATASET 2 39
APPENDIX C – TEST DATASET 3 42
APPENDIX D – CODE FOR CREATING INPUT SHEET 45
APPENDIX E – SCREENSHOTS OF INPUT SHEET 48
APPENDIX F – CODE FOR MODIFIED GREEDY ALGORITHM 51
APPENDIX G – SCREENSHOTS OF RESULTS AND OUTPUT SHEETS 63
VITA 65

LIST OF TABLES

TABLE 1 25
TABLE 2 32
TABLE 3 33

LIST OF FIGURES

FIGURE 1 - Example of the Task Assignment Problem 11
FIGURE 2 - Pseudocode for the General Greedy Algorithm 17
FIGURE 3 - Problem Specific Pseudocode for Greedy Algorithm 18
FIGURE 4 - Pseudocode for one iteration of basic Meta-RaPS procedure 21
FIGURE 5 - Problem Specific Flowchart for Greedy Algorithm 22
FIGURE 6 - Pseudocode for Modified Greedy Algorithm with Meta-RaPS 23
FIGURE 7 - Effect of Percent Priority and Percent Restriction on Best Total Cost 28
FIGURE 8 - Effect of Percent Priority and Percent Restriction on Average Total Cost 29
FIGURE 9- Effect of Percent Priority and Percent Restriction on Standard Deviation 29
FIGURE 10 - 2D Representations of Figures 7, 8 and 9 30

I. INTRODUCTION

The problem of assigning workers to tasks based on worker skill competencies and task skill requirements is one that nearly every company faces. Whether the company wants to efficiently assign workers to tasks for ongoing production or for a series of smaller projects, having a good method for making these assignments in a manner that minimizes cost is extremely important.

It is likely that if management is not using a consistent method to determine worker to task assignments, it will not be able to develop a low-cost assignment or even a feasible assignment at all. The assignment chosen by management may create a situation where the work cannot be completed by the deadline if careful attention is not paid to the time required to train each worker as well as worker capacities. Additionally, if the project is on a tight budget, a bad assignment can put the cost-effectiveness of the entire project into jeopardy. As the problem size grows larger, these negative effects are exacerbated. It is therefore obvious that a consistent method for worker to task allocations is needed.

Software tools are currently available to help companies make better worker to task assignments. These tools, however, do not incorporate means to deal with situations where further training of employees is necessary in order to complete a task. For cases like this, it must be determined which workers to train in which tasks in order to develop the lowest total training cost for the assignment. Both time to train and cost to train must be incorporated. To accomplish this, Depuy *et al.* 2006 developed a math model that includes these two variables when determining optimum worker to task assignments. That model is discussed in detail in Section II, Problem Description.

It is typically more cost effective for companies to train their current employees to meet task competency requirements as opposed to firing workers with inadequate skill competencies and hiring those with more skills. Therefore, the motivation for this project is to change the current workforce to meet the project requirements. This will allow companies to begin planning for the future instead of simply making assignments for the present.

In addition, the results show management which skills to hold training sessions for, and how many employees need that training. Anticipating future training needs and developing employees to meet their personal career goals will be easier. Management can fit employees into training sessions that are already in place to meet employee career development interests.

II. PROBLEM DESCRIPTION

The following terms and definitions are necessary in order to best understand this problem.

- Project – a combination of a few or many tasks that results in the final product or service (for example assembling the frame of an automobile or producing the automobile in its entirety)
- Task – one specific job to be completed by an employee (such as welding two pieces of metal together)
- Skill – a competency requirement in order to complete the job (for instance welding)
- Skill Level – the level of competency of a certain skill held by a worker or required by a task (such as novice, proficient, or expert in welding)

When there is a gap between a worker's current skill level and the required skill level, additional training is necessary. Figure 1 illustrates those skills gaps for an example task assignment problem. As mentioned earlier, Depuy *et al.* 2006 developed a math model that finds the optimal assignment for the Crane Division, Naval Surface Warfare Center (NSWC). This model assumes that once trained in a specific skill, the worker does not need to be retrained in order to complete a different task requiring that skill. The objective function is to minimize the total training cost.

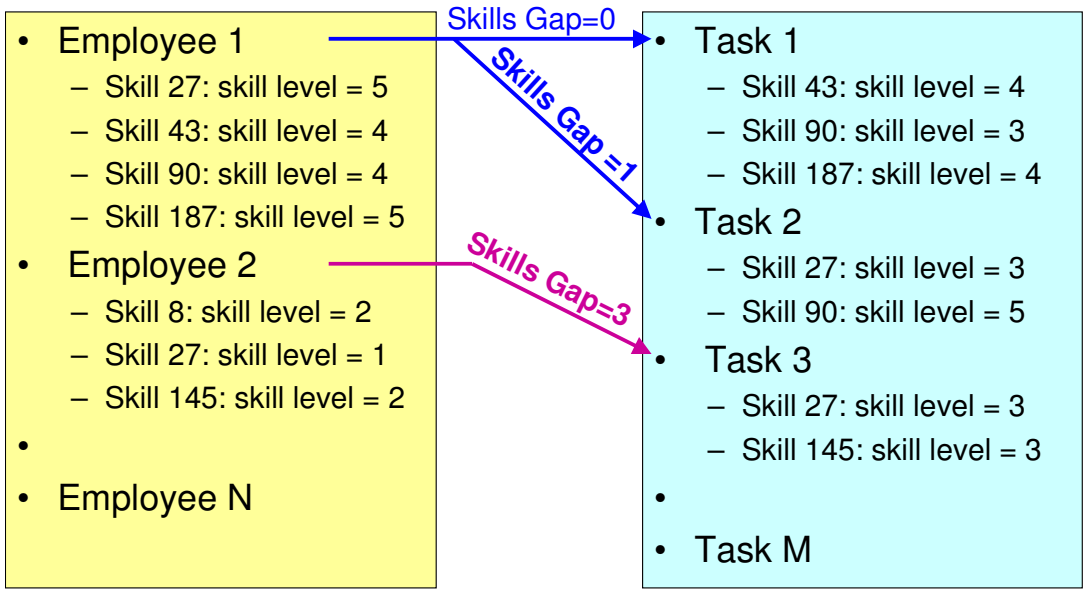


FIGURE 1 - Example of the Task Assignment Problem

The Depuy *et al.* 2006 model is as follows:

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:

$$X_{ij} = \begin{cases} 1, & \text{worker } i \text{ is assigned to task } j \\ 0, & \text{otherwise} \end{cases}$$

$$Z_{ikS_{ik}m} = \begin{cases} 1, & \text{worker } i \text{ receives training on skill } k \text{ to raise skill level from } Sik \text{ to } m \\ 0, & \text{otherwise} \end{cases}$$

$$N_{ik} = \begin{cases} 1, & \text{worker } i \text{ does not need further training in skill } k \\ 0, & \text{otherwise} \end{cases}$$

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

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

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

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

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

Equation 1 is the objective function minimizing the total training cost. Additional training required in order for a worker to be competent enough to complete a particular task is calculated in constraints 2 and 3. All of skills of the task are included when calculating the training needs for the worker. The variable N_{ik} represents when a worker has met the skill level requirement that the task requires and therefore does not need additional training. Next, the model ensures that every task is assigned, but only to one worker (constraints 4). Finally, the total workload for a worker, including training time and task time, must be within the capacity of the worker. This is ensured through constraints 5.

Although this model can solve small problems in an acceptable amount of time, as the problem size increases, run-time also increases to an unsatisfactory level. For example, solving a 9 worker, 13 task and 11 skill problem optimally required 18 hours (see dataset in Appendix A). By utilizing a heuristic, a solution can be found in a reasonable timeframe, but the benefit of an optimal solution must be sacrificed. As an extension of the work completed by Depuy *et al.*, this project focuses on developing a heuristic that will produce a good solution, although likely to be suboptimal, in a reasonable amount of time. The Greedy Algorithm meets the needs of this problem by finding a good solution quickly. For the 9 worker, 13 task and 11 skill problem mentioned above, the Greedy Algorithm finds a solution in under 1 minute.

The downside of the Greedy Algorithm is that it is deterministic. In theory, heuristics such as the Greedy Algorithm have the potential to find the optimal answer, but it is likely that they will be trapped in a local minimum. Modifying the algorithm to produce more than one assignment would allow for the best assignment from a group of possibilities to be chosen, thereby increasing the likelihood that the global optimum will be found. Multiple techniques are available to force the Greedy Algorithm to produce more than one result. These include Genetic Algorithms, Simulated Annealing, Tabu Search and Neural Networks. Each of are discussed in Section III, Background.

This method also forces the assignment of at least one task to each employee, even if that means paying a worker to receive additional training when a more skilled worker has the capacity to complete that task; ensuring that even those employees with the least training and experience will have an opportunity to receive additional training and gain more work experience. In addition, more skilled workers are typically those

who are older and have been with the company longer, and are therefore closer to retirement. If companies only assign tasks to highly skilled workers, they will eventually run into problems when those employees retire. To prepare for the future, companies must have a plan to train newer employees so they will be prepared when skilled workers retire.

III. BACKGROUND

The task assignment problem has been approached from various perspectives. Several researchers have investigated academic exam and proctor scheduling , while others have explored the task assignment problem as it relates to the non-academic work-world, such as telephone operators and construction work.

A. Academic Applications

Scheduling final examinations is a problem that universities face each term. Carter, LaPorte and Chinneck (1994) developed EXAMINE, a PC based scheduling system for exams which allows all examinations to take place in a limited time period, without conflicts, while satisfying room availability constraints. The aim of the authors was to develop a heuristic algorithm that was robust, flexible, quick and user-friendly. The algorithm progressively assigns examinations to periods while optimizing the objective function. Once a feasible schedule is created, the algorithm runs a post optimization phase.

Assigning proctors to the final examinations was approached by Awad and Chinneck (2000). Due to proctor training, preferences, and other constraints, finding a good feasible solution can be problematic. To replace time consuming manual assignments, a computer based system was developed. Assignments are based on a combination of problem-specific heuristics and a genetic-algorithm structure. The authors used Microsoft Access® and Visual Basic® to create an interface and database system for making the assignments.

B. Non-academic Applications

In 1997, Thompson developed a process for assigning telephone operators to shifts at New Brunswick Telephone Company. The specialized shift assignment heuristic (SSAH) assigns shifts to employees based on seniority until a full feasible schedule is created. Then an improvement procedure tests all two-way shift swaps between employee pairs, and makes changes when a more cost effective schedule is found. The author utilized spreadsheet macros incorporated with a stand-alone procedure to create an easy to use PC based technique.

The problem of assigning managers to construction projects at Heery International was approached using a spreadsheet optimization technique (LeBlanc *et al.*, 2000). This method is effective for problems up to 114 projects. This method is easy to modify as new projects come and new managers are hired, and as projects are completed and managers resign. Although this research was specific to assigning managers to construction projects, it is applicable for assigning managers to projects in any organization.

IV. WORKERSKILLS ASSIGNMENT PROCEDURE

A. Greedy Algorithm

A Greedy Algorithm essentially makes the best, near-sighted decision at each stage of the problem in hopes of finding a good solution. In this case, the algorithm will choose the lowest cost worker to task allocation as the first assignment, then choose the next lowest cost worker to task assignment, and so on until all tasks have been assigned. After each assignment, the worker skill set is updated based on any training that he or she may have received (See Figure 2). It is possible that choosing these local minimums will result in the global minimum training cost, but it is more likely that this method alone will not be optimal.

```
Find training cost for each worker to complete each task
Do Until all tasks assigned
        Find worker to task assignment with lowest training cost
        Assign task to worker
        Update worker skill set based on assignment
Loop
Calculate and print total training cost
```

FIGURE 2 - Pseudocode for the General Greedy Algorithm

The Greedy Algorithm shown in Figure 2 does not include the requirement that at least one task is assigned to each worker and that management is changing the current workforce to meet project needs instead of hiring new workers with more skill competencies. This is accomplished through two loops in the program. The first loop is a slight modification of the Greedy Algorithm that will eliminate a worker from the list of available workers once he or she has been assigned a task. After updating all of the

worker skill sets based on their first task assignment, the second loop will assign the rest of the tasks to the workers based solely on minimum cost. See Figure 3 for these modifications.

```
Find training cost for each worker to complete each task
Do Until each worker is assigned one task
        Calculate the sum training cost if worker completes all tasks
        Find worker with maximum sum training cost
        Find lowest cost task for this worker
        Assign task to worker
        Remove worker from available worker list
Loop
Update worker skill sets based on assignments
Do Until all tasks assigned
        Calculate the task sum cost for each task if all workers complete the task
        Find task with maximum sum cost
        Find lowest cost worker for this task
        Assign worker to task
        Remove task from available worker list
Loop
Update worker skill levels based on training received
Calculate and print total training cost
```

FIGURE 3 - Problem Specific Pseudocode for Greedy Algorithm

The theory behind finding the sum training cost in Figure 3 is that a worker who has a higher sum training cost is likely to require more training on average than a worker with a lower sum training cost. Likewise, the worker with the lowest sum training cost is likely to have more tasks where he or she requires little or no training. That person is therefore likely to be more flexible regarding which task should be assigned to them while still maintaining a very low cost. The worker with the highest sum training cost is likely to have very few or no tasks with low training costs. Therefore, that worker is assigned his or her lowest cost task in order to minimize the training cost for that worker. Other workers with lower sum training costs (i.e. more flexibility with assignments) can

then be assigned the tasks that remain. Based on this theory, the algorithm finds the worker with the maximum sum training cost, and then finds the task with the lowest training cost for that worker. Then the skill levels for that worker are updated based on any training he or she may have received. The algorithm repeats this procedure until all workers have at least one task. The second loop in the program operates with the same theory as just described. The task with the maximum sum cost is selected, and then the worker with the lowest training cost for that task is chosen.

B. Modified Greedy Algorithm

As stated earlier, heuristics have the chance to find the optimal answer, but can get trapped in a local optimal solution. Introducing randomness is a common method of dealing with this problem. The Greedy Algorithm alone will find a good answer, but randomizing parts of the algorithm will ensure that multiple answers are possible. Modifying the algorithm such that it sometimes accepts an assignment that temporarily worsens the objective function will succeed in leaving the local optimum and possibly find the global optimal solution. Other modern heuristics, called meta-heuristics, like Genetic Algorithms, Simulated Annealing, Tabu Search and Neural Networks do just that. Another meta-heuristic, Meta-RaPS, developed by Depuy and Whitehouse (2000) is the chosen method for this problem because it is easy to understand and implement while realizing good results.

Meta-RaPS, Meta-heuristic for Randomized Priority Search, was developed as a part of research on applying a modified COMSOAL (Computer Method of Sequencing Operations for Assembly Lines) approach to several combinatorial problems. Originally an approach to the assembly line balancing problem (Arcus, 1966), the theory behind

COMSOAL can also be applied to other problems. Through modifications, COMSOAL has evolved into Meta-RaPS. With Meta-RaPS, Depuy and Whitehouse were able to preserve the underlying idea of COMSOAL, but their modification is noticeably different in practice. Therefore, their approach was presented as Meta-RaPS in 2000.

Other meta-heuristics utilize some device to avoid local minima, and Meta-RaPS is no different. By incorporating an element of randomness, Meta-RaPS is able to modify construction heuristics, and avoid local minima. Using priority rules in a randomized fashion, Meta-RaPS creates a different solution at each iteration and after a number of iterations, Meta-RaPS reports the best solution.

Construction heuristics develop solutions by building up elements with the best priority values to form the final solution. Meta-RaPS modifies this method by sometimes forcing the construction heuristic to choose an element that does not have the best priority value. Three user-defined parameters are used by Meta-RaPS to introduce randomness: percent priority, percent restriction, and percent improvement. Choosing parameters for this model is described in Section V, Results.

The percent priority parameter chooses how often the best priority element is chosen and added to the solution. The rest of the time, the percent restriction parameter is used to choose the next element added to the solution. Percent restriction decides how close to the best priority value the next element needs to be. All values within the percent restriction of the best priority value will be included in the group of available elements. The next element is randomly chosen from the group of available elements. This technique of using percent priority and percent restriction to choose the next element is performed for all elements until a final solution is found (Figure 4). The percent

improvement parameter is used to determine when to run an improvement heuristic. If the solution for an iteration is within the percent improvement of the best unimproved solution so far, an improvement heuristic (neighborhood search) is run.

```
Do Until feasible solution generated
        Find training cost for each feasible worker to task assignment
        Find lowest training cost
        P = RND(1,100)
        If P<= %priority Then
                Add assignment with lowest training cost to solution
        Else
                Form 'available list' of all assignments whose priority values are within
                        %restriction of lowest cost assignment
                Randomly choose assignment from available list and add to solution
        End If
End Until
Calculate and Print solution value
```

FIGURE 4 - Pseudocode for one iteration of basic Meta-RaPS procedure

There are four locations where the Meta-RaPS procedure can be inserted into the general Greedy Algorithm. Two of those locations are in the first loop that assigns each worker one task, and the other two locations are in the final loop assigning all of the remaining tasks. The flowchart in Figure 5 is a representation of the pseudocode from Figure 2. The emphasized boxes are the four locations where Meta-RaPS can be implemented.

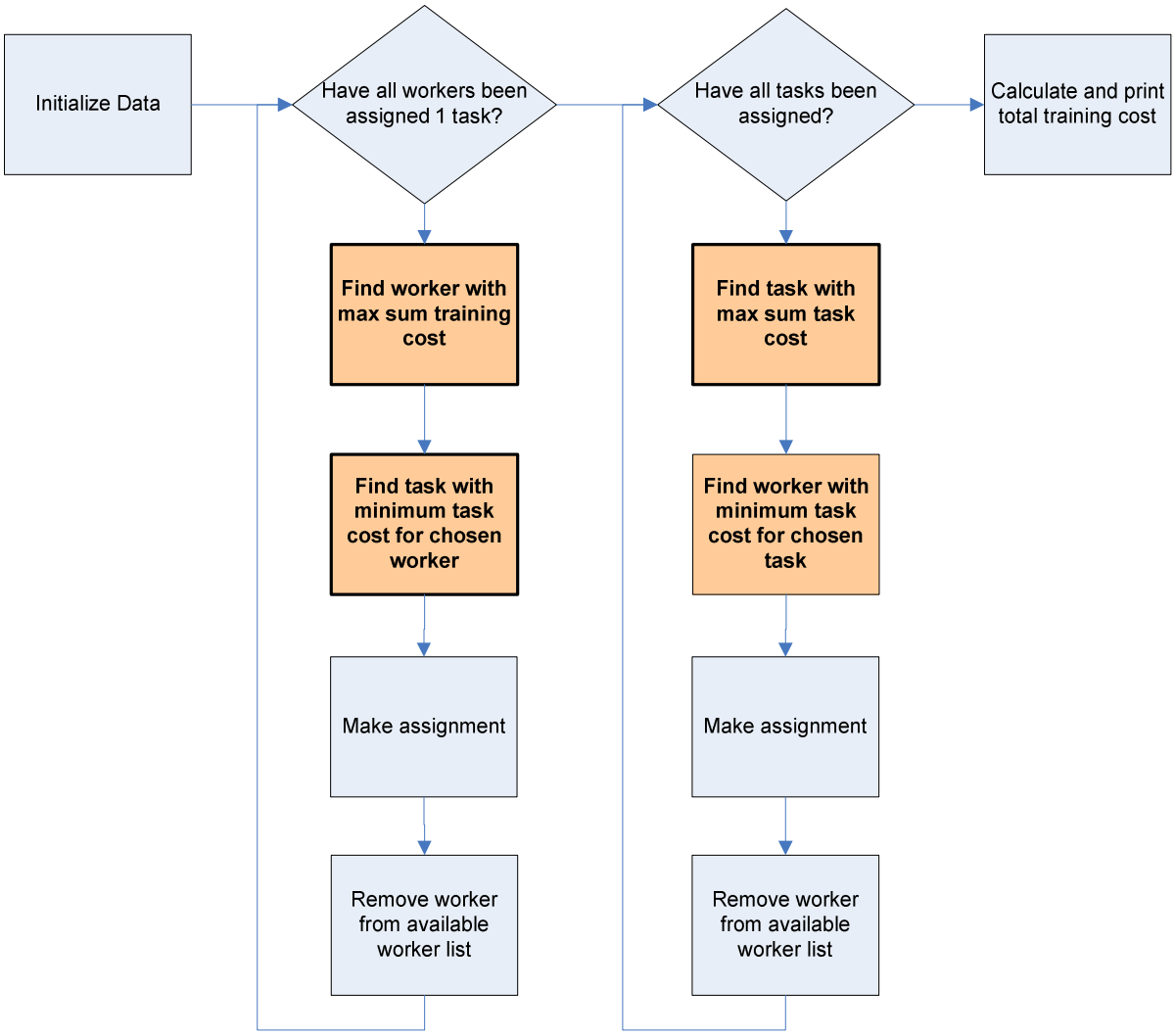


FIGURE 5 - Problem Specific Flowchart for Greedy Algorithm

Meta-RaPS has been implemented in all four of the above locations for this problem in order to maximize the ability of the heuristic to find the best answer possible. The pseudocode for this implementation is given in Figure 6. Instead of including all three parameters in this problem, only percent priority and percent restriction were used. Coding an improvement algorithm for this problem can be a future project.

```
Do Until each worker is assigned one task
                Find feasible worker with max sum training cost across all tasks
```

```
                P = RND(1,100)
                If P<= %priority Then
                        Choose worker with lowest sum training cost
                Else
                        Form 'available list' of workers whose sum training cost is within
                                %restriction of lowest sum training cost
                        Randomly choose worker from 'available list'
                End If
                Find feasible task with lowest training cost
                P = RND(1,100)
                If P<= %priority Then
                        Choose worker to task assignment with max sum cost
                Else
                        Form 'available list' of tasks whose training cost is within
                                %restriction of lowest training cost
                        Randomly choose worker to task assignment from 'available list'
                End If
                Add assignment to solution
                Remove worker from available worker list
Loop
Do Until all tasks assigned
        Find lowest cost feasible worker to task assignment for each worker
        Find overall lowest cost assignment
        P = RND(1,100)
        If P<= %priority Then
                Add assignment with overall lowest cost to solution
        Else
                Form 'available list' of all feasible assignments whose cost are within
                        %restriction of lowest cost assignment
                Randomly choose assignment from 'available list' and add to solution
        End If
End Until
Calculate and Print solution value
```

FIGURE 6 - Pseudocode for Modified Greedy Algorithm with Meta-RaPS

V. RESULTS

Microsoft Excel® VBA 2007 was used to program this modified Greedy Algorithm (see Appendix B). Other programming languages could be more efficient solving this problem, but Excel® VBA is more useful in the corporate world. One benefit is that no new programs such as Lingo® or another stand-alone program have to be purchased and installed in order to run the analysis. Excel® VBA is also very useful for developing functional outputs specific to a company's precise needs, and is easily compatible with other Office® programs such as Project® and Access®.

As stated earlier, the parameters used by Meta-RaPS are user-defined, and must be determined. If desired, this heuristic can mimic both the math model and traditional Greedy Algorithm by making the parameters specific values. A percent priority values of 0 and percent restriction values of 100 for each Meta-RaPS instance will find the optimal assignment. Percent priority values of 100 will mimic the traditional Greedy Algorithm.

At each use of Meta-RaPS, the percent priority and percent restriction values can be different. For simplicity and ease of use, however, each parameter is held constant for each application. The traditional method for choosing the values of these parameters has been trial and error, and therefore is the method used for choosing the parameters for this problem.

A sample dataset with 9 workers, 13 tasks and 11 skills was used to test this heuristic (see Appendix B). Other sample datasets (see Appendices A and C) are used to confirm the results from this sample dataset, and will be discussed in Section VI, Conclusions and Recommendations. In order to ensure that the solution values for the test dataset were accurate, 5000 iterations were run and the best solution was chosen.

This best total cost as well as the average total cost and standard deviation over all iterations are given in the Table 1 for each percent priority, percent restriction pair. It is important to ensure that these values are optimized when choosing the percent priority and percent restriction to be used for future tests.

TABLE 1

SUMMARY OF PERCENT PRIORITY AND PERCENT RESTRICTION ANALYSIS

Percent Priority	Percent Restriction	Best Total Cost	Average Total Cost	Standard Deviation
10	10	394	440	23
10	20	386	438	24
10	30	386	449	28
10	40	386	467	33
10	50	386	485	35
10	60	395	502	38
10	70	394	515	43
10	80	383	515	44
10	90	395	523	44
20	10	394	442	23
20	20	386	439	24
20	30	386	447	28
20	40	386	464	33
20	50	386	479	35
20	60	387	493	39
20	70	390	501	44
20	80	380	504	44
20	90	390	512	45
30	10	394	444	22
30	20	386	439	24
30	30	386	446	28
30	40	386	461	33
30	50	386	475	35
30	60	389	486	37
30	70	390	495	42

30	80	380	494	43
30	90	380	501	44
40	10	394	445	22
40	20	386	440	24
40	30	386	444	27
40	40	386	457	31
40	50	386	468	35
40	60	386	477	37
40	70	386	482	41
40	80	380	484	42
40	90	380	490	42
50	10	394	449	21
50	20	386	443	23
50	30	386	443	26
50	40	386	454	30
50	50	386	463	32
50	60	388	469	35
50	70	387	471	38
50	80	387	474	39
50	90	388	479	40
60	10	394	452	19
60	20	386	446	23
60	30	386	444	25
60	40	386	453	29
60	50	388	460	31
60	60	388	466	32
60	70	388	464	35
60	80	387	467	37
60	90	388	471	37
70	10	394	454	17
70	20	386	450	21
70	30	386	446	24
70	40	386	453	26
70	50	386	457	29
70	60	388	461	29
70	70	388	458	31
70	80	388	462	33

70	90	388	465	34
80	10	394	457	14
80	20	388	453	19
80	30	386	449	22
80	40	394	455	24
80	50	394	455	25
80	60	394	458	26
80	70	387	453	29
80	80	390	457	30
80	90	390	459	29
90	10	394	460	11
90	20	388	458	14
90	30	388	455	17
90	40	388	457	18
90	50	388	458	20
90	60	394	459	20
90	70	390	455	23
90	80	388	457	23
90	90	390	458	24

The charts in Figures 7 through 10 were created in Matlab® 7.4 to illustrate the effect of percent priority and percent restriction on the response variables above. Figures 7 through 9 are 3D maps, and Figure 10 shows a 2D illustration of Figures 7 through 9. It is difficult to determine what the best parameter values are from the Best Total Cost graphs, but percent priority values of 20, 40 and 40 look good as do percent restriction values of 80 and 90. The Average Total Cost graphs are more interesting. There is a trend in the percent priority that indicates that as the percent priority increases, percent restriction has a lower effect on the response. This is evident in the fact that the range decreases on the percent priority graph for that response in Figure 10. Also, there is an remarkable trend in the percent restriction graph for the Average Total Cost in Figure 10. It appears that the graph flip-flops at a percent restriction value of 30. Finally, the

Standard Deviation graphs indicate that higher values of percent priority and lower values of percent restriction create solutions with less deviation.

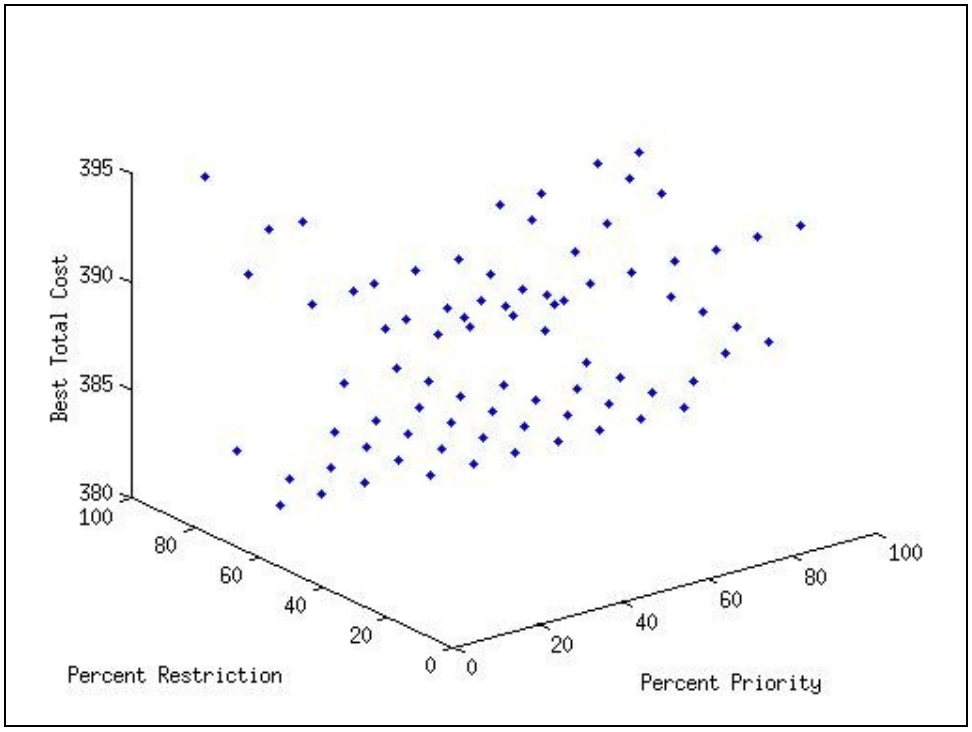


FIGURE 7 - Effect of Percent Priority and Percent Restriction on Best Total Cost

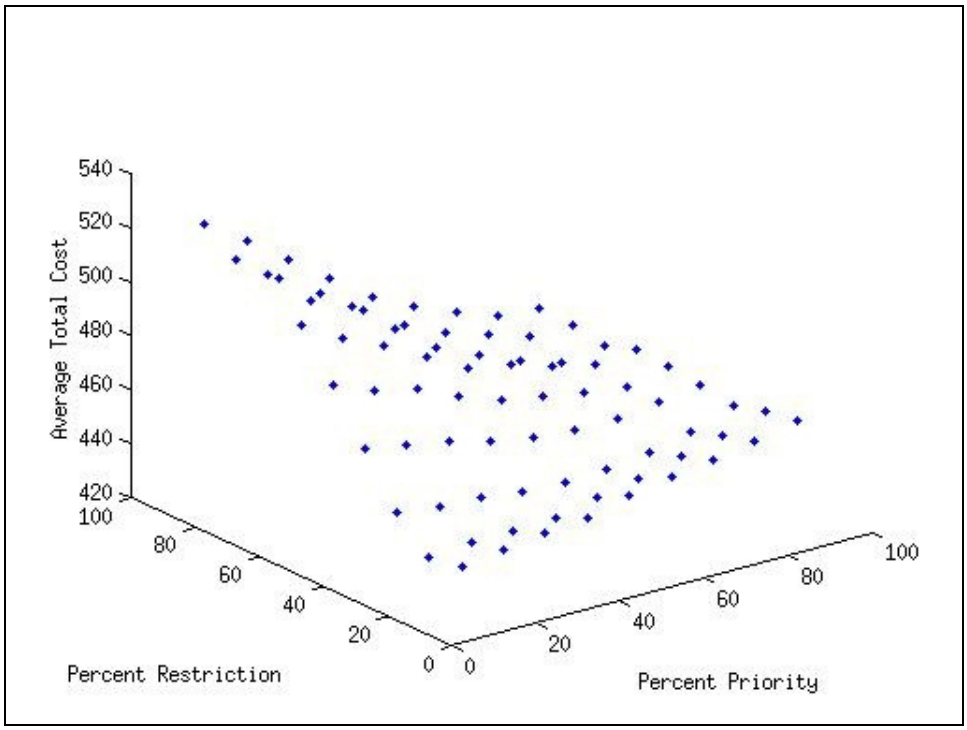


FIGURE 8 - Effect of Percent Priority and Percent Restriction on Average Total Cost

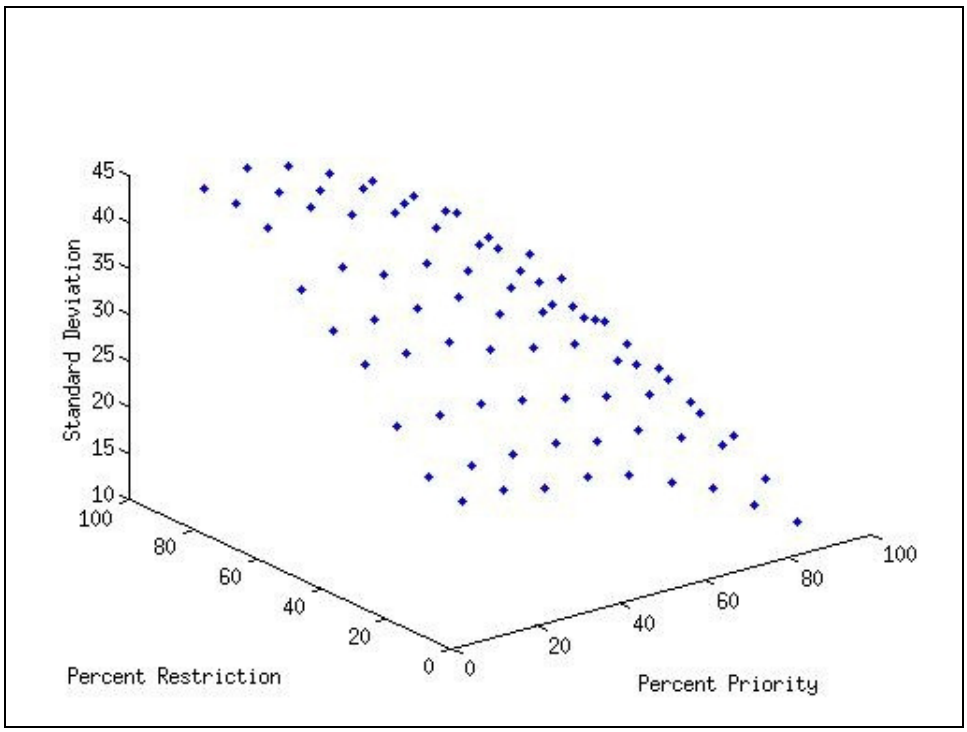


FIGURE 9- Effect of Percent Priority and Percent Restriction on Standard Deviation

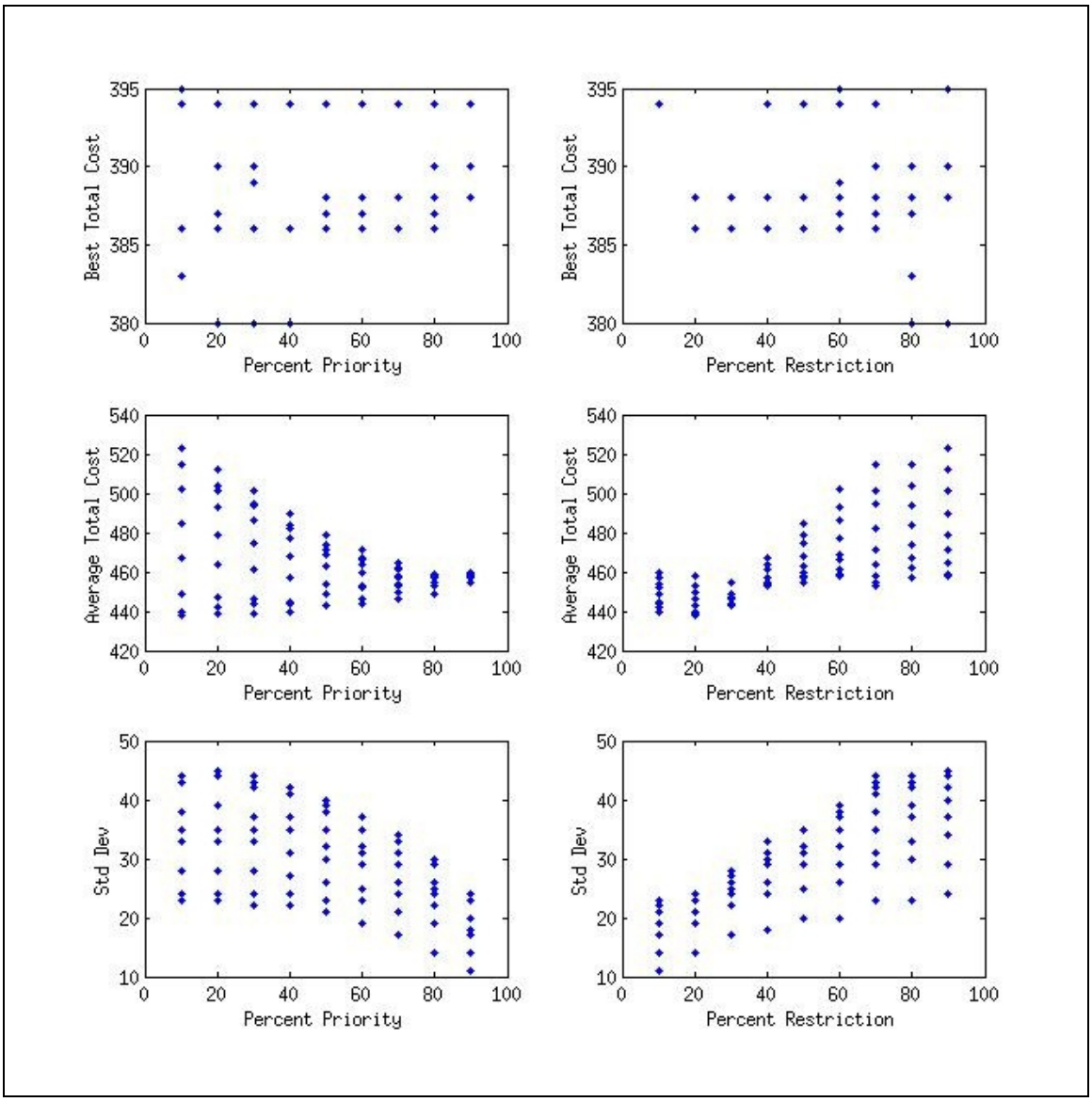


FIGURE 10 - 2D Representations of Figures 7, 8 and 9

All of this information aids in the understanding of the problem, but it still isn't clear which values are the best. Since best total cost is the value that is of most interest, it is expected that a percent priority value of 30 and a percent restriction value of 80 will create the best results. This can be confirmed using the other 3 sample datasets in Appendices A and C. These results are presented in Section VI, Conclusions and Recommendations.

The Excel® VBA program runs using two macros. The first macro is used to create the input sheet (Appendix D). The user inputs the number of workers, skills and tasks into three message boxes, and the macro creates space for the user to input all of the relevant information. Please refer to Appendix E for screenshots of the input sheet. The second macro runs the heuristic and reports the solution in the Results and Output sheets (Appendices F and G). The user can then use the solution information to plan training sessions for the employees.

VI. CONCLUSIONS AND RECOMMENDATIONS

The modified Greedy Algorithm works well for this problem. The test results for the sample data are shown in Table 2. For the parameter values chosen, the best solution is the same as the optimal solution. The average solution is 30% greater than optimal. This can be combated by ensuring that there are enough iterations to get the lowest value possible. For this dataset, 5000 iterations appears to be adequate.

TABLE 2

OPTIMAL AND HEURISTIC RESULTS FOR TEST DATASET

Optimal Solution		380
Modified Greedy Algorithm	Best Solution	380
	Average Solution	494
	Standard Deviation	43

Table 3 below shows the results for two other datasets (see Appendices A and C). The best solution for the first dataset is within 2% of optimal, and the second dataset is within 4% of optimal. The average solution for the datasets are within 30% and 48%, respectively. Although this isn't ideal, with 5000 iterations a very good solution is found. These results confirm that the selected parameter values from the previous test dataset work for other datasets as well, and prove it to be a fitting substitute for solving this problem optimally.

TABLE 3

RESULTS FOR OTHER DATASETS

Appendix A	Optimal Solution	393
	Best Heuristic Solution	400
	Average Heuristic Solution	511
	Heuristic Standard Deviation	38
Appendix C	Optimal Solution	297
	Best Heuristic Solution	309
	Average Heuristic Solution	440
	Heuristic Standard Deviation	53

Although these sample datasets prove that the heuristic works well, real-world data can be used to further analyze the effect of different parameter settings. In addition to incorporating actual data, running multiple replications and running a factorial analysis using Minitab® or another statistical software package will aid in choosing proper parameter values. A factorial analysis can optimize the percent priority and percent restriction based on all three responses (best solution, average solution, and standard deviation). To do this, multiple replicates of the heuristic can be run using real-world data to create a full factorial experimental design. This analysis is essential for making this heuristic more marketable to companies. As more data is collected for the future analysis, the interesting trends shown in Figures 7 through 10 should be revisited to determine their cause.

As mentioned earlier, it is typical to include an improvement algorithm as a part of the Meta-RaPS procedure, but it was not included at this time. Coding an improvement algorithm is a possible future endeavor, but would require future research into the best way to do this. In addition, it is not guaranteed that it will improve the solution much more than the current solution without increasing the amount of time.

Finally, the basic heuristic in Excel® VBA has been shown to be good for developing the solution, but does not present that solution in a format that is extremely functional for specific company use. The next step is to create company-specific user-friendly reports. This way it will be more appealing and easy to integrate into current employee training systems.

This new heuristic has many benefits, but also some limitations. Being able to solve problems in a fraction of the time as the optimal algorithm makes this heuristic a good option for companies needing quick solutions. Also, since the heuristic is able to find a solution within 5% of optimal (with 5000 iterations), it is a good alternative for companies who are concerned about being as close to optimal as possible without while sacrificing hours finding a solution. The major limitation lies in the lack of testing. Since no real-world data was available, proving the effectiveness of the algorithm in the corporate world is difficult. Once more testing is completed using actual data and more user-friendly reports are available, this heuristic will be extremely useful for companies wishing to find a quick and reliable method for assigning workers to tasks.

REFERENCES

Arcus, A. 1966. COMSOAL: A Computer method of sequencing operations for assembly lines. *International Journal of Production Research* 4:259-277.

Awad, R. and Chinneck, J. 2000. Proctor assignment at Carlton University. *Interfaces* 28:2:58-71.

Carter, M., Laporte, G., Chinneck, J. 1994. A general examination scheduling system. *Interfaces* 24:3:109-120.

DePuy, G., Whitehouse, G., 2000. Applying the COMSOAL computer heuristic to the constrained resource allocation problem, Computers and Industrial Engineering, 38, 413-422.

DePuy, G., Whitehouse, G., 2001. A simple and effective heuristic for the multiple resource allocation problem, International Journal of Production Research, 39 (14), 3275-3287.

Depuy, G., Moraga, R., Whitehouse, G. 2003. Meta-RaPS: A simple and effective approach for solving the traveling salesman problem. *Transportation Research Part E*. 212.

Depuy, G., Usher, J., and Arterburn, B. 2006. Workforce training schedule for logistics skills. CD-ROM Proceedings of the 2006 Industrial Engineerng Research Conference, May 20-24, Orlando, Florida.

LeBlanc, L., Randels, K., and Swan, T. K. 2000. Heery International's spreadsheet optimization model for assigning managers to construction projects. *Interfaces* 30:6:95-106.

Thompson, G. M. 1997. Assigning telephone operators to shifts at New Brunswick Telephone Company. *Interfaces* 27:4:1-11.

APPENDIX A

TEST DATASET 1

Number of workers	9
Number of skills	11
Number of tasks	13

Worker Skill Matrix											
	Skill 1	Skill 2	Skill 3	Skill 4	Skill 5	Skill 6	Skill 7	Skill 8	Skill 9	Skill 10	Skill 11
Worker 1	1	2	5	1	2	4	5	3	5	2	3
Worker 2	2	5	5	1	4	2	4	4	4	5	1
Worker 3	2	2	1	2	2	2	3	5	4	2	1
Worker 4	3	4	4	3	5	3	1	4	1	2	3
Worker 5	5	2	2	5	5	4	2	5	3	5	3
Worker 6	4	1	4	1	5	3	4	2	3	4	4
Worker 7	3	4	4	3	4	1	2	3	5	5	1
Worker 8	4	2	1	2	1	2	4	5	1	2	4
Worker 9	3	3	5	1	3	4	3	5	4	3	2

The above matrix shows the current skill levels of each worker for each skill type.

Task Skill Matrix											
	Skill 1	Skill 2	Skill 3	Skill 4	Skill 5	Skill 6	Skill 7	Skill 8	Skill 9	Skill 10	Skill 11
Task 1	4	5	3	5	3	2	5	4	3	4	5
Task 2	2	2	2	5	4	3	1	3	1	3	1
Task 3	3	4	3	4	2	5	2	3	5	4	3
Task 4	2	4	2	2	5	3	5	2	4	5	2
Task 5	5	2	5	4	5	3	1	4	5	5	4
Task 6	5	2	3	3	4	3	2	4	3	2	2
Task 7	2	1	5	5	1	5	4	4	2	1	5
Task 8	2	4	5	3	1	2	5	3	3	2	4
Task 9	2	2	3	4	1	1	3	5	1	4	4
Task 10	3	2	4	2	1	3	4	4	4	4	2
Task 11	1	2	1	5	1	5	2	1	1	3	1
Task 12	5	4	4	2	2	1	1	5	1	2	3
Task 13	1	5	2	3	1	2	5	5	2	1	1

The above matrix shows the required skill levels for each task and skill type.

	Task Time
Task 1	4
Task 2	4
Task 3	4
Task 4	4
Task 5	4
Task 6	4
Task 7	4
Task 8	4
Task 9	4
Task 10	4
Task 11	4
Task 12	4
Task 13	4

	Worker Capacity
Worker 1	25
Worker 2	25
Worker 3	25
Worker 4	25
Worker 5	25
Worker 6	25
Worker 7	25
Worker 8	25
Worker 9	25

Cost to Train Matrix					
	Train to Skill Level 1	Train to Skill Level 2	Train to Skill Level 3	Train to Skill Level 4	Train to Skill Level 5
Skill 1	0	1	3	7	15
Skill 2	0	1	3	7	15
Skill 3	0	1	3	7	15
Skill 4	0	1	3	7	15
Skill 5	0	1	3	7	15
Skill 6	0	1	3	7	15
Skill 7	0	1	3	7	15
Skill 8	0	1	3	7	15
Skill 9	0	1	3	7	15
Skill 10	0	1	3	7	15
Skill 11	0	1	3	7	15

The above matrix shows the cost to train a worker up to a skill level from the level immediately preceding it for each skill type. This sample model assumes that the cost to train up to the higher skill levels is not liner. In other words, it costs more to train a worker from a skill level of 4 to 5 than from a skill level of 1 to 2.

Time to Train Matrix					
	Train to Skill Level 1	Train to Skill Level 2	Train to Skill Level 3	Train to Skill Level 4	Train to Skill Level 5
Skill 1	0	1	1	1	1
Skill 2	0	1	1	1	1
Skill 3	0	1	1	1	1
Skill 4	0	1	1	1	1
Skill 5	0	1	1	1	1
Skill 6	0	1	1	1	1
Skill 7	0	1	1	1	1
Skill 8	0	1	1	1	1
Skill 9	0	1	1	1	1
Skill 10	0	1	1	1	1
Skill 11	0	1	1	1	1

The above matrix shows the time to train a worker up to a skill level from the level immediately preceding it for each skill type.

APPENDIX B

TEST DATASET 2

Number of workers	9
Number of skills	11
Number of tasks	13

Worker Skill Matrix											
	Skill 1	Skill 2	Skill 3	Skill 4	Skill 5	Skill 6	Skill 7	Skill 8	Skill 9	Skill 10	Skill 11
Worker 1	2	2	3	4	4	1	5	3	4	2	1
Worker 2	5	2	1	3	2	5	1	4	2	5	2
Worker 3	4	2	3	2	4	2	2	1	3	4	4
Worker 4	1	2	5	5	1	2	5	2	3	3	2
Worker 5	2	2	2	2	1	3	2	3	4	4	2
Worker 6	1	2	1	2	5	2	2	2	3	3	3
Worker 7	4	1	3	2	4	4	2	2	1	2	2
Worker 8	2	5	2	5	3	4	4	5	2	2	3
Worker 9	2	2	2	4	4	1	1	5	2	3	3

The above matrix shows the current skill levels of each worker for each skill type.

Task Skill Matrix											
	Skill 1	Skill 2	Skill 3	Skill 4	Skill 5	Skill 6	Skill 7	Skill 8	Skill 9	Skill 10	Skill 11
Task 1	2	3	5	2	2	2	3	5	3	1	3
Task 2	1	4	2	4	4	1	3	2	5	3	1
Task 3	2	4	3	5	4	2	5	4	2	3	2
Task 4	1	1	3	4	5	5	1	1	2	1	4
Task 5	3	4	3	4	1	5	3	3	1	1	1
Task 6	5	1	3	1	4	5	1	4	2	1	1
Task 7	1	2	2	5	3	1	2	2	2	1	2
Task 8	5	2	5	5	4	5	5	5	3	2	3
Task 9	4	2	4	2	3	1	1	3	1	4	5
Task 10	4	5	5	4	5	2	5	3	1	1	2
Task 11	1	4	5	4	3	4	2	3	1	4	1
Task 12	1	3	1	1	4	3	2	1	1	5	2
Task 13	1	2	2	5	4	1	3	1	2	2	5

The above matrix shows the required skill levels for each task and skill type.

	Task Time
Task 1	4
Task 2	4
Task 3	4
Task 4	4
Task 5	4
Task 6	4
Task 7	4
Task 8	4
Task 9	4
Task 10	4
Task 11	4
Task 12	4
Task 13	4

	Worker Capacity
Worker 1	25
Worker 2	25
Worker 3	25
Worker 4	25
Worker 5	25
Worker 6	25
Worker 7	25
Worker 8	25
Worker 9	25

Cost to Train Matrix					
	Train to Skill Level 1	Train to Skill Level 2	Train to Skill Level 3	Train to Skill Level 4	Train to Skill Level 5
Skill 1	0	1	3	7	15
Skill 2	0	1	3	7	15
Skill 3	0	1	3	7	15
Skill 4	0	1	3	7	15
Skill 5	0	1	3	7	15
Skill 6	0	1	3	7	15
Skill 7	0	1	3	7	15
Skill 8	0	1	3	7	15
Skill 9	0	1	3	7	15
Skill 10	0	1	3	7	15
Skill 11	0	1	3	7	15

The above matrix shows the cost to train a worker up to a skill level from the level immediately preceding it for each skill type. This sample model assumes that the cost to train up to the higher skill levels is not liner. In other words, it costs more to train a worker from a skill level of 4 to 5 than from a skill level of 1 to 2.

Time to Train Matrix					
	Train to Skill Level 1	Train to Skill Level 2	Train to Skill Level 3	Train to Skill Level 4	Train to Skill Level 5
Skill 1	0	1	1	1	1
Skill 2	0	1	1	1	1
Skill 3	0	1	1	1	1
Skill 4	0	1	1	1	1
Skill 5	0	1	1	1	1
Skill 6	0	1	1	1	1
Skill 7	0	1	1	1	1
Skill 8	0	1	1	1	1
Skill 9	0	1	1	1	1
Skill 10	0	1	1	1	1
Skill 11	0	1	1	1	1

The above matrix shows the time to train a worker up to a skill level from the level immediately preceding it for each skill type.

APPENDIX C

TEST DATASET 3

Number of workers	9
Number of skills	11
Number of tasks	13

Worker Skill Matrix											
	Skill 1	Skill 2	Skill 3	Skill 4	Skill 5	Skill 6	Skill 7	Skill 8	Skill 9	Skill 10	Skill 11
Worker 1	5	3	3	4	1	5	5	2	4	4	4
Worker 2	2	5	3	3	1	3	1	1	5	5	2
Worker 3	1	5	5	4	5	5	4	4	5	1	5
Worker 4	2	3	1	1	2	4	4	3	4	1	3
Worker 5	3	3	2	1	1	4	5	2	3	2	3
Worker 6	2	1	5	5	4	2	4	5	4	1	1
Worker 7	5	4	1	3	4	3	3	3	4	4	1
Worker 8	2	2	5	4	2	2	5	5	3	4	4
Worker 9	3	2	5	5	1	3	1	3	2	1	5

The above matrix shows the current skill levels of each worker for each skill type.

Task Skill Matrix											
	Skill 1	Skill 2	Skill 3	Skill 4	Skill 5	Skill 6	Skill 7	Skill 8	Skill 9	Skill 10	Skill 11
Task 1	5	2	3	3	4	4	5	5	2	3	5
Task 2	2	2	4	3	5	5	3	2	4	5	2
Task 3	1	4	3	4	3	3	5	3	2	3	2
Task 4	1	1	5	2	1	3	1	5	1	1	3
Task 5	3	3	4	5	1	4	2	5	4	3	3
Task 6	1	3	5	1	3	3	5	1	4	1	3
Task 7	5	1	1	1	5	5	2	4	4	3	4
Task 8	3	2	1	1	2	2	2	1	2	4	5
Task 9	2	5	5	1	2	2	1	4	4	3	5
Task 10	5	5	2	5	4	5	3	5	4	3	5
Task 11	4	1	1	1	2	2	3	3	5	2	5
Task 12	1	3	1	4	5	5	1	2	5	5	5
Task 13	3	1	1	4	4	3	1	2	4	4	1

The above matrix shows the required skill levels for each task and skill type.

	Task Time
Task 1	4
Task 2	4
Task 3	4
Task 4	4
Task 5	4
Task 6	4
Task 7	4
Task 8	4
Task 9	4
Task 10	4
Task 11	4
Task 12	4
Task 13	4

	Worker Capacity
Worker 1	25
Worker 2	25
Worker 3	25
Worker 4	25
Worker 5	25
Worker 6	25
Worker 7	25
Worker 8	25
Worker 9	25

Cost to Train Matrix					
	Train to Skill Level 1	Train to Skill Level 2	Train to Skill Level 3	Train to Skill Level 4	Train to Skill Level 5
Skill 1	0	1	3	7	15
Skill 2	0	1	3	7	15
Skill 3	0	1	3	7	15
Skill 4	0	1	3	7	15
Skill 5	0	1	3	7	15
Skill 6	0	1	3	7	15
Skill 7	0	1	3	7	15
Skill 8	0	1	3	7	15
Skill 9	0	1	3	7	15
Skill 10	0	1	3	7	15
Skill 11	0	1	3	7	15

The above matrix shows the cost to train a worker up to a skill level from the level immediately preceding it for each skill type. This sample model assumes that the cost to train up to the higher skill levels is not liner. In other words, it costs more to train a worker from a skill level of 4 to 5 than from a skill level of 1 to 2.

Time to Train Matrix					
	Train to Skill Level 1	Train to Skill Level 2	Train to Skill Level 3	Train to Skill Level 4	Train to Skill Level 5
Skill 1	0	1	1	1	1
Skill 2	0	1	1	1	1
Skill 3	0	1	1	1	1
Skill 4	0	1	1	1	1
Skill 5	0	1	1	1	1
Skill 6	0	1	1	1	1
Skill 7	0	1	1	1	1
Skill 8	0	1	1	1	1
Skill 9	0	1	1	1	1
Skill 10	0	1	1	1	1
Skill 11	0	1	1	1	1

The above matrix shows the time to train a worker up to a skill level from the level immediately preceding it for each skill type.

APPENDIX D

CODE FOR CREATING INPUT SHEET

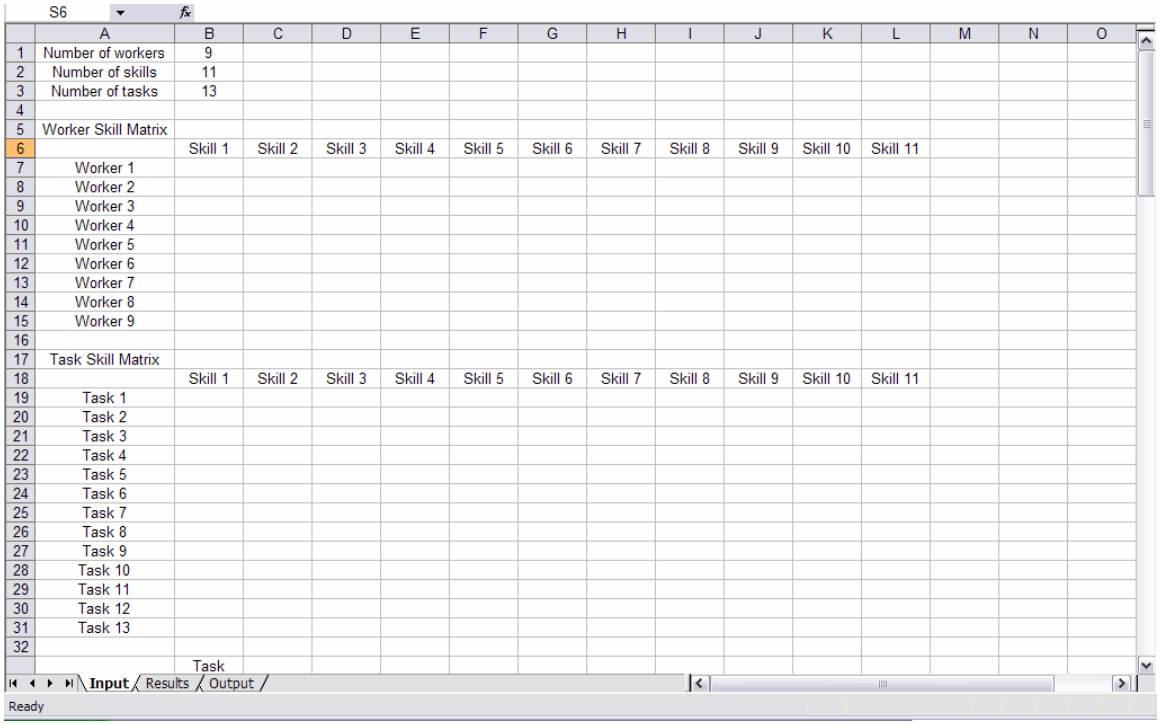
```
Sub inputs()

Dim numworkers As Single
Dim numskills As Single
Dim numtasks As Single

Sheets("Input").Select

numworkers = Application.InputBox("Input number of workers", "")
numskills = Application.InputBox("Input number of skills", "")
numtasks = Application.InputBox("Input number of tasks", "")

'Insert values
ActiveSheet.Cells(1, 1).Value = "Number of workers"
ActiveSheet.Cells(2, 1).Value = "Number of skills"
ActiveSheet.Cells(3, 1).Value = "Number of tasks"
ActiveSheet.Cells(1, 2).Value = numworkers
ActiveSheet.Cells(2, 2).Value = numskills
ActiveSheet.Cells(3, 2).Value = numtasks

'Create Worker Skill Matrix
For i = 1 To numworkers
   For k = 1 To numskills
      ActiveSheet.Cells(5, 1).Value = "Worker Skill Matrix"
      ActiveSheet.Cells(i + 6, 1).Value = "Worker " & i
      ActiveSheet.Cells(6, k + 1).Value = "Skill " & k
   Next k
Next i

'Create Task Skill Matrix
For j = 1 To numtasks
   For k = 1 To numskills
      ActiveSheet.Cells(8 + numworkers, 1).Value = "Task Skill Matrix"
      ActiveSheet.Cells(9 + numworkers + j, 1).Value = "Task " & j
      ActiveSheet.Cells(9 + numworkers, k + 1).Value = "Skill " & k
   Next k
Next j

'Create Task Time Matrix
For j = 1 To numtasks
   ActiveSheet.Cells(11 + numtasks + numworkers, 2).Value = "Task Time"
   ActiveSheet.Cells(11 + numtasks + numworkers + j, 1).Value = "Task " & j
```

```
Next j

'Create Worker Capacity Matrix
For i = 1 To numworkers
   ActiveSheet.Cells(13 + 2 * numtasks + numworkers, 2).Value = "Worker Capacity"
   ActiveSheet.Cells(13 + 2 * numtasks + numworkers + i, 1).Value = "Worker " & i
Next i

'Create Training Cost Matrix
For i = 1 To numskills
   For j = 1 To 5
      ActiveSheet.Cells(15 + 2 * numtasks + 2 * numworkers, 1).Value = "Cost to Train
Matrix"
      ActiveSheet.Cells(16 + 2 * numtasks + 2 * numworkers + i, 1).Value = "Skill " & i
      ActiveSheet.Cells(16 + 2 * numtasks + 2 * numworkers, 1 + j).Value = "Train to
Skill Level " & j
   Next j
Next i

'Create Training Time Matrix
For i = 1 To numskills
   For j = 1 To 5
      ActiveSheet.Cells(18 + 2 * numtasks + 2 * numworkers + numskills, 1).Value =
"Time to Train Matrix"
      ActiveSheet.Cells(19 + 2 * numtasks + 2 * numworkers + numskills + i, 1).Value =
"Skill " & i
      ActiveSheet.Cells(19 + 2 * numtasks + 2 * numworkers + numskills, 1 + j).Value =
"Train to Skill Level " & j
   Next j
Next i

'Create Skill Name Matrix
For i = 1 To numskills
   ActiveSheet.Cells(19 + 2 * numtasks + 2 * numworkers + numskills + numskills + 2,
1) = "Skill"
   ActiveSheet.Cells(19 + 2 * numtasks + 2 * numworkers + numskills + numskills + 2,
2) = "Skill Name"
   ActiveSheet.Cells(19 + 2 * numtasks + 2 * numworkers + numskills + numskills + 2 +
i, 1) = i
Next i

'Create Worker Name Matrix
For i = 1 To numworkers
   ActiveSheet.Cells(19 + 2 * numtasks + 2 * numworkers + numskills + numskills + 2 +
numskills + 2, 1) = "Worker "
```

```
    ActiveSheet.Cells(19 + 2 * numtasks + 2 * numworkers + numskills + numskills + 2 +
numskills + 2, 2) = "Worker Name"
    ActiveSheet.Cells(19 + 2 * numtasks + 2 * numworkers + numskills + numskills + 2 +
numskills + 2 + i, 1) = i
Next i

'Create Task Name Matrix
For i = 1 To numtasks
    ActiveSheet.Cells(19 + 2 * numtasks + 2 * numworkers + numskills + numskills + 2 +
numskills + 2 + numworkers + 2, 1) = "Task"
    ActiveSheet.Cells(19 + 2 * numtasks + 2 * numworkers + numskills + numskills + 2 +
numskills + 2 + numworkers + 2, 2) = "Task Name"
    ActiveSheet.Cells(19 + 2 * numtasks + 2 * numworkers + numskills + numskills + 2 +
numskills + 2 + numworkers + 2 + i, 1) = i
Next i

End Sub
```

APPENDIX E

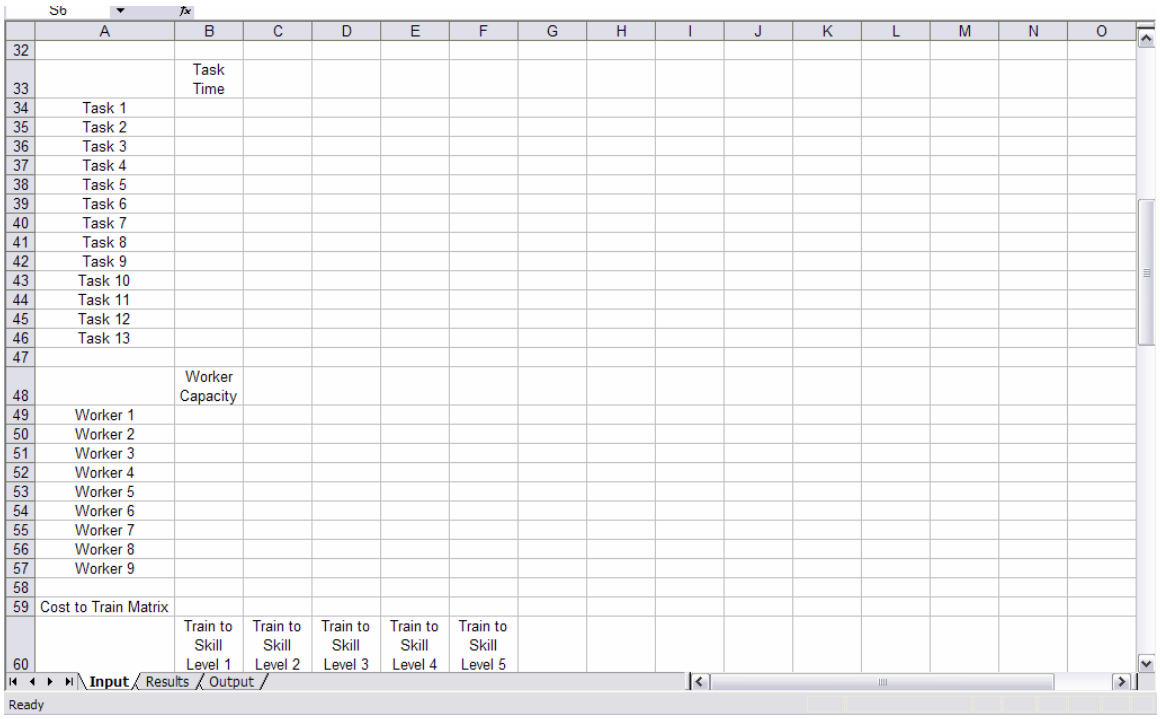
SCREENSHOTS OF INPUT SHEET



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Number of workers	9													
2	Number of skills	11													
3	Number of tasks	13													
4															
5	Worker Skill Matrix														
6		Skill 1	Skill 2	Skill 3	Skill 4	Skill 5	Skill 6	Skill 7	Skill 8	Skill 9	Skill 10	Skill 11			
7	Worker 1														
8	Worker 2														
9	Worker 3														
10	Worker 4														
11	Worker 5														
12	Worker 6														
13	Worker 7														
14	Worker 8														
15	Worker 9														
16															
17	Task Skill Matrix														
18		Skill 1	Skill 2	Skill 3	Skill 4	Skill 5	Skill 6	Skill 7	Skill 8	Skill 9	Skill 10	Skill 11			
19	Task 1														
20	Task 2														
21	Task 3														
22	Task 4														
23	Task 5														
24	Task 6														
25	Task 7														
26	Task 8														
27	Task 9														
28	Task 10														
29	Task 11														
30	Task 12														
31	Task 13														
32															
		Task													

Input / Results / Output

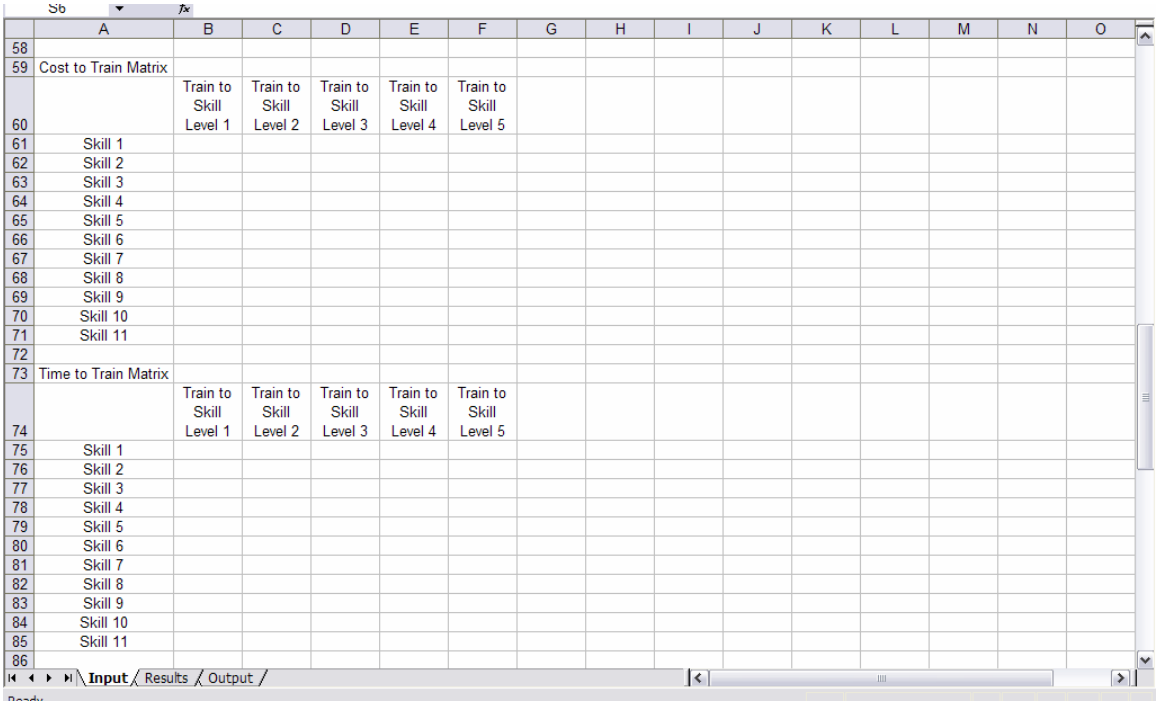
Ready



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
32															
33		Task Time													
34	Task 1														
35	Task 2														
36	Task 3														
37	Task 4														
38	Task 5														
39	Task 6														
40	Task 7														
41	Task 8														
42	Task 9														
43	Task 10														
44	Task 11														
45	Task 12														
46	Task 13														
47															
48		Worker Capacity													
49	Worker 1														
50	Worker 2														
51	Worker 3														
52	Worker 4														
53	Worker 5														
54	Worker 6														
55	Worker 7														
56	Worker 8														
57	Worker 9														
58															
59	Cost to Train Matrix														
60		Train to Skill Level 1	Train to Skill Level 2	Train to Skill Level 3	Train to Skill Level 4	Train to Skill Level 5									

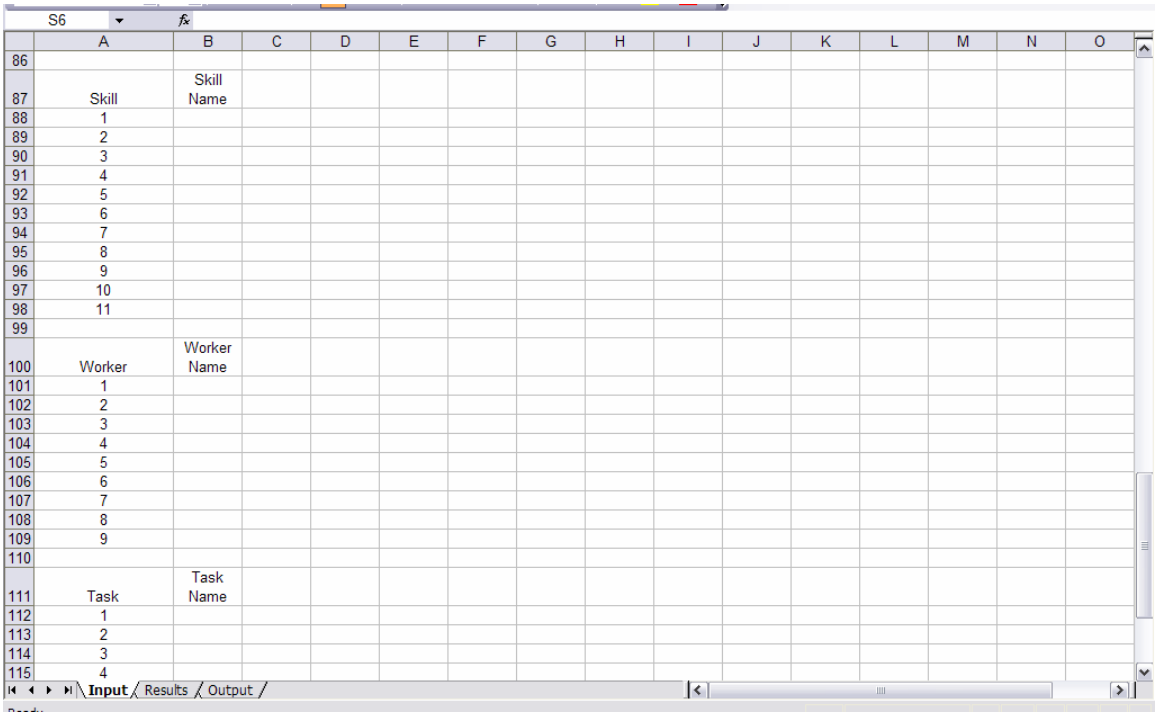
Input / Results / Output

Ready



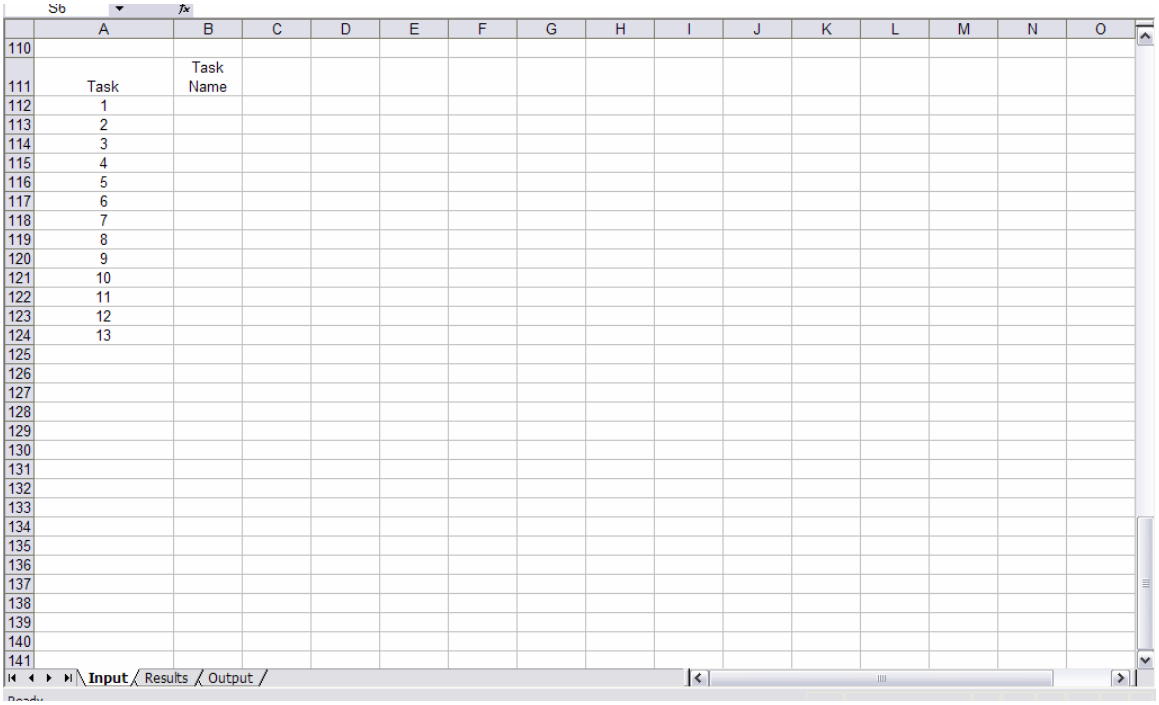
	A	B	C	D	E	F
58						
59	Cost to Train Matrix					
60		Train to Skill Level 1	Train to Skill Level 2	Train to Skill Level 3	Train to Skill Level 4	Train to Skill Level 5
61	Skill 1					
62	Skill 2					
63	Skill 3					
64	Skill 4					
65	Skill 5					
66	Skill 6					
67	Skill 7					
68	Skill 8					
69	Skill 9					
70	Skill 10					
71	Skill 11					
72						
73	Time to Train Matrix					
74		Train to Skill Level 1	Train to Skill Level 2	Train to Skill Level 3	Train to Skill Level 4	Train to Skill Level 5
75	Skill 1					
76	Skill 2					
77	Skill 3					
78	Skill 4					
79	Skill 5					
80	Skill 6					
81	Skill 7					
82	Skill 8					
83	Skill 9					
84	Skill 10					
85	Skill 11					
86						

Input / Results / Output /



	A	B	C	D	E	F
86						
87	Skill	Skill Name				
88	1					
89	2					
90	3					
91	4					
92	5					
93	6					
94	7					
95	8					
96	9					
97	10					
98	11					
99						
100	Worker	Worker Name				
101	1					
102	2					
103	3					
104	4					
105	5					
106	6					
107	7					
108	8					
109	9					
110						
111	Task	Task Name				
112	1					
113	2					
114	3					
115	4					

Input / Results / Output /



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
110															
111	Task	Task Name													
112	1														
113	2														
114	3														
115	4														
116	5														
117	6														
118	7														
119	8														
120	9														
121	10														
122	11														
123	12														
124	13														
125															
126															
127															
128															
129															
130															
131															
132															
133															
134															
135															
136															
137															
138															
139															
140															
141															

Input / Results / Output /

APPENDIX F

CODE FOR MODIFIED GREEDY ALGORITHM

```
Public Sub heuristic()

Dim workerskill() As Single, oworkerskill() As Single
Dim taskskill() As Single, otaskskill() As Single
Dim tasktime() As Single, otasktime() As Single
Dim workercapacity() As Single, oworkercapacity() As Single
Dim traincost() As Single, otraincost() As Single
Dim traintime() As Single, otraintime() As Single
Dim workerassign() As Single
Dim workertaskcost() As Single, oworkertaskcost() As Single
Dim workertasktime() As Single, oworkertasktime() As Single
Dim taskassigned() As Single
Dim tcost() As Single
Dim ttime() As Single
Dim available() As Single
Dim bestworkerassign() As Single
Dim numworkers As Single, numskills As Single, numtasks As Single
Dim totaltaskcost() As Single
Dim totalworkercost() As Single
Dim workerphase1() As Single
Dim cellrow As Single


perprior = 20
perrestrict = 50
numiter = 5000

phase1_on = 1 'this can be used as a switch to turn phase 1 on or off


Sheets("Input").Select
numworkers = ActiveSheet.Cells(1, 2).Value
numskills = ActiveSheet.Cells(2, 2).Value
numtasks = ActiveSheet.Cells(3, 2).Value

'initialize arrays
ReDim workerskill(0 To numworkers + 1, 0 To numskills + 1) As Single
ReDim oworkerskill(0 To numworkers + 1, 0 To numskills + 1) As Single
ReDim taskskill(0 To numtasks + 1, 0 To numskills + 1) As Single
ReDim otaskskill(0 To numtasks + 1, 0 To numskills + 1) As Single
ReDim tasktime(0 To numtasks + 1) As Single
ReDim otasktime(0 To numtasks + 1) As Single
```

```
ReDim workercapacity(0 To numworkers + 1) As Single
ReDim oworkercapacity(0 To numworkers + 1) As Single
ReDim traincost(0 To numskills + 1, 0 To 5, 0 To 5) As Single
ReDim otraincost(0 To numskills + 1, 0 To 5, 0 To 5) As Single
ReDim traintime(0 To numskills + 1, 0 To 5, 0 To 5) As Single
ReDim otraintime(0 To numskills + 1, 0 To 5, 0 To 5) As Single
ReDim workerassign(0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim bestworkerassign(0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim workertaskcost(0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim oworkertaskcost(0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim workertasktime(0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim oworkertasktime(0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim taskassigned(0 To numtasks + 1) As Single
ReDim tcost(0 To numskills + 1, 0 To 5) As Single
ReDim ttime(0 To numskills + 1, 0 To 5) As Single
ReDim available(0 To numworkers * numtasks + 1, 0 To 3) As Single
ReDim totaltaskcost(0 To numtasks + 1) As Single
ReDim totalworkercost(0 To numworkers + 1) As Single
ReDim workerphase1(0 To numworkers + 1) As Single



For b = 0 To numworkers + 1
   workercapacity(b) = 0
   oworkercapacity(b) = 0
   For k = 0 To numskills + 1
      workerskill(b, k) = 0
      oworkerskill(b, k) = 0
   Next k
Next b

For b = 0 To numworkers * numtasks + 1
   For k = 0 To 3
      available(b, k) = 0
   Next k
Next b


For b = 0 To numtasks + 1
   tasktime(b) = 0
   otasktime(b) = 0
   taskassigned(b) = 0
   totaltaskcost(b) = 0
   For k = 0 To numskills + 1
      taskskill(b, k) = 0
      otaskskill(b, k) = 0
```

```
   Next k
Next b

For i = 0 To numskills + 1
   For j = 0 To 5
      tcost(i, j) = 0
      ttime(i, j) = 0
      For k = 0 To 5
         traincost(i, j, k) = 0
         traintime(i, j, k) = 0
         otraincost(i, j, k) = 0
         otraintime(i, j, k) = 0
      Next k
   Next j
Next i

For b = 1 To numworkers
   totalworkercost(b) = 0
   For k = 1 To numtasks
      workerassign(b, k) = 0
      workertaskcost(b, k) = 0
      oworkertaskcost(b, k) = 0
   Next k
Next b


'read in data from file
For b = 1 To numworkers
   For k = 1 To numskills
      oworkerskill(b, k) = ActiveSheet.Cells(6 + b, 1 + k)
   Next k
Next b

For b = 1 To numtasks
   For k = 1 To numskills
      otaskskill(b, k) = ActiveSheet.Cells(6 + numworkers + 3 + b, 1 + k)
   Next k
Next b

For b = 1 To numtasks
   otasktime(b) = ActiveSheet.Cells(6 + numworkers + 3 + numtasks + 2 + b, 2)
Next b

For b = 1 To numworkers
   oworkercapacity(b) = ActiveSheet.Cells(6 + numworkers + 3 + numtasks + 2 +
numtasks + 2 + b, 2)
```

```
Next b

For i = 1 To numskills
   For j = 1 To 5
      tcost(i, j) = ActiveSheet.Cells(6 + numworkers + 3 + numtasks + 2 + numtasks + 2 +
         numworkers + 3 + i, 1 + j)
   Next j
Next i

For i = 1 To numskills
   For j = 1 To 5
      ttime(i, j) = ActiveSheet.Cells(6 + numworkers + 3 + numtasks + 2 + numtasks + 2
         + numworkers + 3 + numskills + 3 + i, 1 + j)
   Next j
Next i

For i = 1 To numskills
   For j = 1 To 5
      For k = 1 To 5
         If j < k And k > 1 Then
            otraincost(i, j, k) = otraincost(i, j, k - 1) + tcost(i, k)
         End If
      Next k
   Next j
Next i

For i = 1 To numskills
   For j = 1 To 5
      For k = 1 To 5
         If j < k And k > 1 Then
            otraintime(i, j, k) = otraintime(i, j, k - 1) + ttime(i, k)
         End If
      Next k
   Next j
Next i


'find task cost and training time for each worker for each task
For i = 1 To numworkers
   For j = 1 To numtasks
      oworkertasktime(i, j) = otasktime(j)
      For k = 1 To numskills
         If oworkerskill(i, k) < otaskskill(j, k) And otaskskill(j, k) > 1 Then
            oworkertaskcost(i, j) = oworkertaskcost(i, j) + otraincost(k, oworkerskill(i, k),
                  otaskskill(j, k))
```

```
            oworkertasktime(i, j) = oworkertasktime(i, j) + otraintime(k, oworkerskill(i, k),
                otaskskill(j, k))
        End If
      Next k
   Next j
Next i

For j = 1 To numtasks
   For i = 1 To numworkers
      totaltaskcost(j) = totaltaskcost(j) + oworkertaskcost(i, j)
   Next i
Next j


For i = 1 To numworkers
   For j = 1 To numtasks
      totalworkercost(i) = totalworkercost(i) + oworkertaskcost(i, j)
   Next j
Next i


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Sheets("Output").Select

bestsolution = 999999999

For r = 1 To numiter

   'copy original data into matrices
   For b = 1 To numworkers
      workercapacity(b) = oworkercapacity(b)
      For k = 1 To numskills
         workerskill(b, k) = oworkerskill(b, k)
      Next k
   Next b

   For b = 1 To numtasks
      tasktime(b) = otasktime(b)
      taskassigned(b) = 0
      For k = 1 To numskills
         taskskill(b, k) = otaskskill(b, k)
      Next k
   Next b

   For i = 1 To numskills
```

```
    For j = 1 To 5
        For k = 1 To 5
            traincost(i, j, k) = otraincost(i, j, k)
            traintime(i, j, k) = otraintime(i, j, k)
        Next k
    Next j
Next i

For b = 1 To numworkers
    For k = 1 To numtasks
        workerassign(b, k) = 0
        workertaskcost(b, k) = oworkertaskcost(b, k)
        workertasktime(b, k) = oworkertasktime(b, k)
    Next k
Next b

For b = 1 To numworkers
    workerphase1(b) = 0
Next b


totalcost = 0
numtaskassigned = 0

If phase1_on = 1 Then  'this can be used as a switch to turn phase 1 on or off

    'start phase 1 - each worker assigned 1 task
    Do While numtaskassigned < numworkers
    'find lowest skilled worker - worker with the highest totalcost
    'make sure they are not already assigned
        maxcost = 0
        For i = 1 To numworkers
            If workerphase1(i) = 0 And totalworkercost(i) > maxcost Then
                maxcost = totalworkercost(i)
                maxcostworker = i
            End If
        Next i

        'find lowest cost task for maxcost worker - make sure task not already assigned
        'make sure worker has enough capacity
        mincost = 99999999
        For j = 1 To numtasks
            If taskassigned(j) = 0 And workertasktime(maxcostworker, j) <=
                  workercapacity(maxcostworker) And workertaskcost(maxcostworker, j) <
                  mincost Then
                mincost = workertaskcost(maxcostworker, j)
```

```
        mincosttask = j
    End If
Next j


Randomize
priorrnd = Round(((100 - 1) * Rnd) + 1)

    If priorrnd <= perprior Then
        'assign maxcostworker to mincost task
        totalcost = totalcost + workertaskcost(maxcostworker, mincosttask)
        numtaskassigned = numtaskassigned + 1
        workerassign(maxcostworker, mincosttask) = 1
        taskassigned(mincosttask) = 1
        workerphase1(maxcostworker) = 1
        workercapacity(maxcostworker) = workercapacity(maxcostworker) -
            workertasktime(maxcostworker, mincosttask)
        assignedworker = maxcostworker
        assignedtask = mincosttask
    End If

    If priorrnd > perprior Then
            'form available list and choose assigned task from available list
            numonlist = 0
            For j = 1 To numtasks
                If taskassigned(j) = 0 And workertasktime(maxcostworker, j) <=
                        workercapacity(maxcostworker) And
                        workertaskcost(maxcostworker, j) < mincost * (1 + (perrestrict /
                        100)) Then
                        numonlist = numonlist + 1
                        available(numonlist, 1) = maxcostworker
                        available(numonlist, 2) = j
                End If
            Next j

        Randomize
        restrictrnd = Round(((numonlist - 1) * Rnd) + 1)
        assignedworker = available(restrictrnd, 1)
        assignedtask = available(restrictrnd, 2)

        totalcost = totalcost + workertaskcost(assignedworker, assignedtask)
        numtaskassigned = numtaskassigned + 1
        workerassign(assignedworker, assignedtask) = 1
        taskassigned(assignedtask) = 1
        workerphase1(assignedworker) = 1
```

```
            workercapacity(assignedworker) = workercapacity(assignedworker) -
                workertasktime(assignedworker, assignedtask)
        End If


        'update workerskills for assignedworker based on training received for
                assignedtask
        For k = 1 To numskills
            If workerskill(assignedworker, k) < taskskill(assignedtask, k) Then
                workerskill(assignedworker, k) = taskskill(assignedtask, k)
            End If
        Next k

        'update workertaskcost and workertasktime for assignedworker
        For j = 1 To numtasks
            If taskassigned(j) = 0 Then
                workertaskcost(assignedworker, j) = 0
                workertasktime(assignedworker, j) = otasktime(j)
                For k = 1 To numskills
                    If workerskill(assignedworker, k) < taskskill(j, k) And taskskill(j, k) > 1
                            Then
                        workertaskcost(assignedworker, j) = workertaskcost(assignedworker,
                            j) + traincost(k, workerskill(assignedworker, k), taskskill(j, k))
                        workertasktime(assignedworker, j) =
                            workertasktime(assignedworker, j) + traintime(k,
                            workerskill(assignedworker, k), taskskill(j, k))
                    End If
                Next k
            End If
        Next j

    Loop

  End If  'If phase1_on = 1

'end of phase 1 switch

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''


  'start phase 2 - assign remaining tasks
  Do While numtaskassigned < numtasks 'repeat until all tasks assigned
    'find highest cost task - make sure it is not already assigned
    maxcost = -55
    For j = 1 To numtasks
      If taskassigned(j) = 0 And totaltaskcost(j) > maxcost Then
```

```
        maxcost = totaltaskcost(j)
        maxcosttask = j
    End If
Next j

'find lowest cost worker for highest cost task - make sure worker has enough
    capacity
mincost = 9999999
mincostworker = 0
For i = 1 To numworkers
    If workertasktime(i, maxcosttask) <= workercapacity(i) And workertaskcost(i,
            maxcosttask) < mincost Then
        mincost = workertaskcost(i, maxcosttask)
        mincostworker = i
    End If
Next i


Randomize
priorrnd = Round(((100 - 1) * Rnd) + 1)

If mincostworker > 0 Then
    If priorrnd <= perprior Then
        'assign mincostworker to maxcost task
        totalcost = totalcost + workertaskcost(mincostworker, maxcosttask)
        numtaskassigned = numtaskassigned + 1
        workerassign(mincostworker, maxcosttask) = 1
        taskassigned(maxcosttask) = 1
        workercapacity(mincostworker) = workercapacity(mincostworker) -
            workertasktime(mincostworker, maxcosttask)
        assignedworker = mincostworker
        assignedtask = maxcosttask
    End If

    If priorrnd > perprior Then
        'form available list and choose assigned worker from available list
         numonlist = 0
         For j = 1 To numtasks
            If totaltaskcost(j) >= maxcost * (1 - (perrestrict / 100)) And taskassigned(j) =
                0 Then
                 For i = 1 To numworkers
                    If workertaskcost(i, j) <= mincost * (1 + (perrestrict / 100)) And
                        workertasktime(i, j) <= workercapacity(i) Then
                        numonlist = numonlist + 1
                        available(numonlist, 1) = i
                        available(numonlist, 2) = j
```

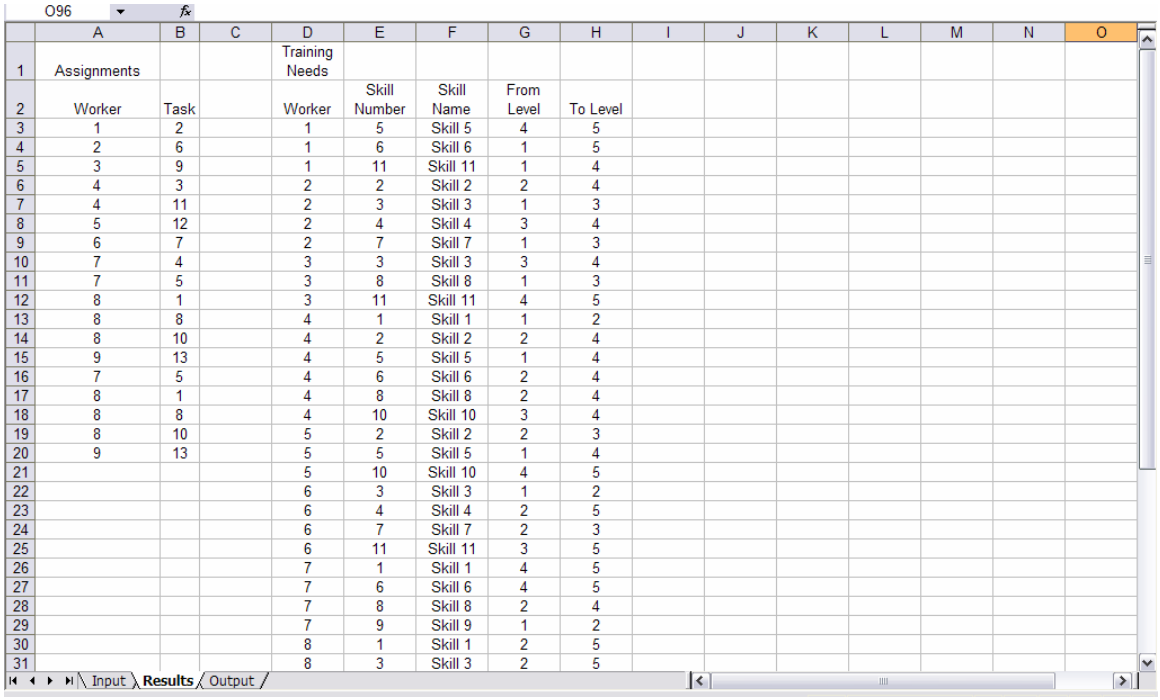
```
            End If
          Next i
        End If
      Next j

      Randomize
      restrictrnd = Round(((numonlist - 1) * Rnd) + 1)
      assignedworker = available(restrictrnd, 1)
      assignedtask = available(restrictrnd, 2)

      totalcost = totalcost + workertaskcost(assignedworker, assignedtask)
      numtaskassigned = numtaskassigned + 1
      workerassign(assignedworker, assignedtask) = 1
      taskassigned(assignedtask) = 1
      workercapacity(assignedworker) = workercapacity(assignedworker) -
          workertasktime(assignedworker, assignedtask)
   End If


   'update workerskills for assignedworker based on training received for
         assignedtask
   For k = 1 To numskills
      If workerskill(assignedworker, k) < taskskill(assignedtask, k) Then
        workerskill(assignedworker, k) = taskskill(assignedtask, k)
      End If
   Next k

   'update workertaskcost and workertasktime for assignedworker
   For j = 1 To numtasks
      If taskassigned(j) = 0 Then
        workertaskcost(assignedworker, j) = 0
        workertasktime(assignedworker, j) = otasktime(j)
        For k = 1 To numskills
          If workerskill(assignedworker, k) < taskskill(j, k) And taskskill(j, k) > 1
          Then
            workertaskcost(assignedworker, j) = workertaskcost(assignedworker, j)
                + traincost(k, workerskill(assignedworker, k), taskskill(j, k))
            workertasktime(assignedworker, j) = workertasktime(assignedworker, j)
                + traintime(k, workerskill(assignedworker, k), taskskill(j, k))
          End If
        Next k
      End If
   Next j

End If
```

```
        If mincostworker = 0 Then
            MsgBox ("No feasible solution.  Not enough worker capacity")
            totalcost = 99999999
            numtaskassigned = numtasks + 1
        End If

    Loop


    'print assignment of tasks to workers
    If totalcost < bestsolution Then
        bestsolution = totalcost
        For i = 1 To numworkers
            For j = 1 To numtasks
                bestworkerassign(i, j) = workerassign(i, j)
            Next j
        Next i
    End If

Next r

    ActiveSheet.Cells(1, 1) = "%Priority"
    ActiveSheet.Cells(1, 2) = perprior
    ActiveSheet.Cells(2, 1) = "%Restriction"
    ActiveSheet.Cells(2, 2) = perrestrict
    ActiveSheet.Cells(3, 1) = "Number of Iterations"
    ActiveSheet.Cells(3, 2) = numiter
    ActiveSheet.Cells(4, 1) = "Best Solution Cost"
    ActiveSheet.Cells(4, 2) = bestsolution
    ActiveSheet.Cells(6, 1) = "Worker to Task Assignments"
    ActiveSheet.Cells(7, 1) = "Worker"
    ActiveSheet.Cells(7, 2) = "Task"

    cellrow = 8
    For i = 1 To numworkers
        For j = 1 To numtasks
            If bestworkerassign(i, j) = 1 Then
                ActiveSheet.Cells(cellrow, 1) = i
                ActiveSheet.Cells(cellrow, 2) = j
                cellrow = cellrow + 1
            End If
        Next j
    Next i

    Sheets("Results").Select
```

```
    ActiveSheet.Cells(1, 1) = "Assignments"
    ActiveSheet.Cells(2, 1) = "Worker"
    ActiveSheet.Cells(2, 2) = "Task"

    cellrow = 3
    For i = 1 To numworkers
       For j = 1 To numtasks
          If bestworkerassign(i, j) = 1 Then
             ActiveSheet.Cells(cellrow, 1) = i
             ActiveSheet.Cells(cellrow, 2) = j
             cellrow = cellrow + 1
          End If
       Next j
    Next i

    ActiveSheet.Cells(1, 4) = "Training Needs"
    ActiveSheet.Cells(2, 4) = "Worker"
    ActiveSheet.Cells(2, 5) = "Skill Number"
    ActiveSheet.Cells(2, 6) = "Skill Name"
    ActiveSheet.Cells(2, 7) = "From Level"
    ActiveSheet.Cells(2, 8) = "To Level"

    cellrow = 3
    For i = 1 To numworkers
       For k = 1 To numskills
          If oworkerskill(i, k) < workerskill(i, k) Then
             ActiveSheet.Cells(cellrow, 4) = i
             ActiveSheet.Cells(cellrow, 5) = k
             ActiveSheet.Cells(cellrow, 6) = Sheets("Input").Cells(19 + 2 * numtasks + 2 *
                   numworkers + numskills + numskills + 2 + k, 2)
             ActiveSheet.Cells(cellrow, 7) = oworkerskill(i, k)
             ActiveSheet.Cells(cellrow, 8) = workerskill(i, k)
             cellrow = cellrow + 1
          End If
       Next k
    Next i

End Sub
```

APPENDIX G

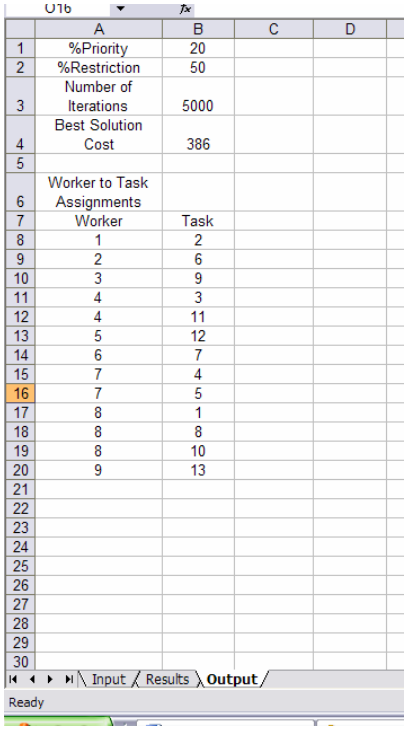
SCREENSHOTS OF RESULTS AND OUTPUT SHEETS



	A	B	C	D	E	F	G	H
1	Assignments			Training Needs				
2	Worker	Task		Worker	Skill Number	Skill Name	From Level	To Level
3	1	2		1	5	Skill 5	4	5
4	2	6		1	6	Skill 6	1	5
5	3	9		1	11	Skill 11	1	4
6	4	3		2	2	Skill 2	2	4
7	4	11		2	3	Skill 3	1	3
8	5	12		2	4	Skill 4	3	4
9	6	7		2	7	Skill 7	1	3
10	7	4		3	3	Skill 3	3	4
11	7	5		3	8	Skill 8	1	3
12	8	1		3	11	Skill 11	4	5
13	8	8		4	1	Skill 1	1	2
14	8	10		4	2	Skill 2	2	4
15	9	13		4	5	Skill 5	1	4
16	7	5		4	6	Skill 6	2	4
17	8	1		4	8	Skill 8	2	4
18	8	8		4	10	Skill 10	3	4
19	8	10		5	2	Skill 2	2	3
20	9	13		5	5	Skill 5	1	4
21				5	10	Skill 10	4	5
22				6	3	Skill 3	1	2
23				6	4	Skill 4	2	5
24				6	7	Skill 7	2	3
25				6	11	Skill 11	3	5
26				7	1	Skill 1	4	5
27				7	6	Skill 6	4	5
28				7	8	Skill 8	2	4
29				7	9	Skill 9	1	2
30				8	1	Skill 1	2	5
31				8	3	Skill 3	2	5

Input | Results | Output

The Results sheet reports the assignments and training needs associated with the best solution. Each worker to task assignment is listed. The training for each skill is also listed for each worker. This sheet aids the company in determining which skills require training sessions.



	A	B	C	D
1	%Priority	20		
2	%Restriction	50		
3	Number of Iterations	5000		
4	Best Solution Cost	386		
5				
6	Worker to Task Assignments			
7	Worker	Task		
8	1	2		
9	2	6		
10	3	9		
11	4	3		
12	4	11		
13	5	12		
14	6	7		
15	7	4		
16	7	5		
17	8	1		
18	8	8		
19	8	10		
20	9	13		
21				
22				
23				
24				
25				
26				
27				
28				
29				
30				

Input | Results | Output

Ready

The Output sheet includes the percent priority and percent restriction values as well as the number of iterations. It also reports the best solution cost and assignments for the best solution.

VITA

Allison Michelle Douglas

amdoug01@louisville.edu

EDUCATION

M.Eng. Industrial Engineering — August 2007
University of Louisville, Louisville, KY

B.S. Industrial Engineering — August 2006
University of Louisville, Louisville, KY — With High Honors

HONORS AND AWARDS

Provost-Hallmark Scholarship, University of Louisville — 2002 to present
W.S. Speed Award, University of Louisville — Spring 2007

EXTRACURRICULAR ACTIVITIES

Speed School Student Council — Fall 2002 to present
Student Government Association — Summer 2003 to present
Institute of Industrial Engineers — Fall 2006 to present
Society of Women Engineers — Fall 2006
Tau Beta Sigma Honorary Music Sorority — Fall 2002 to Fall 2005
Society of Physics Students — Fall 2002 to Fall 2003

PERSONAL ACTIVITIES

Big Brothers Big Sisters, Louisville, KY — 2004 to present
U of L Marching Band, Pep Band and Community Band — 2002-2005