5-2011

A multi-objective decision support system for worker-task assignments and workforce training.

Brandon B. Elmes
University of Louisville

A MULTI-OBJECTIVE DECISION SUPPORT SYSTEM FOR WORKER-TASK ASSIGNMENTS AND WORKFORCE TRAINING

By

Brandon B. Elmes
B.S., University of Louisville, 2007

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

MASTER OF ENGINEERING

Department of Industrial Engineering

May 2011

A MULTI-OBJECTIVE DECISION SUPPORT SYSTEM FOR WORKER-TASK ASSIGNMENTS AND WORKFORCE TRAINING

Submitted by:_____

Brandon B. Elmes

A Thesis Approved on

_____
(Date)

by the Following Reading and Examination Committee:

_____
Gerald W. Evans, Thesis Director

_____
Gail W. DePuy, Thesis Director

_____
Rammohan K. Ragade, Committee Member

ACKNOWLEDGEMENTS

The author received considerable help from both Thesis Directors: Dr. Gerald Evans and Dr. Gail DePuy. Thank you for your time, effort, and patience. Also, thanks to Dr. Rammoham Ragade for his time as a thesis committee member.

The author acknowledges the assistance of the NSWC in Crane, IN which provided data for the mathematical model.

ABSTRACT

This paper models a realistic problem involving workforce assignment and training for a large manufacturing environment. In this particular environment, the workforce is undertrained and most assignments will result in necessary training. This problem was previously addressed as a single objective problem. This paper expands to a multi-objective formulation. This is a more accurate reflection of the problem because almost all real world problems have many objectives which can be conflicting.

The program developed in this paper is designed for use by supervisors in the production setting. A two stage program is designed where the first stage generates initial solutions by solving each objective function idependently of the others. Meta-RaPS—a modified greedy algorithm—is used to find these solutions. The user selects one of these solutions to carry into the second stage: compromise programming. The second stage uses input from the user in an iterative and intuitive fashion. This input guides the program to the solution which the user determines is the best compromise solution.

Meta-RaPS is effective at finding a good solution extremely quickly. There is an important trade-off between the quality of solutions and computational run-time which will need to be tweaked for a specific application. The compromise programming stage could benefit from coding improvements; however, it is still effective at allowing the user to guide the program towards the best compromise solution by assigning trade-off values between objectives

TABLE OF CONTENTS

NOMENCLATURE:

$\{j\}$ = 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 (hr) of task j

A_i = capacity (hr) 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 (hr) to raise a worker's skill level on skill k from level l to level m

$$P_{ik} = \begin{cases} 1 & \text{worker i would like additional training in skill k} \\ 0 & \text{otherwise} \end{cases}$$

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

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

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

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

The Naval Surface Warfare Center (NSWC) has a division located in Crane, Indiana. NSWC Crane is a shore command of the US Navy. This division is a shore command of the United States Navy under the Naval Sea Systems Command headquartered in Washington D.C. NSWC specializes in the acquisition and fleet support of electronics, ordinance products, and electronic warfare products. NSWC routinely bids on a wide variety of remanufacturing work from many different military organizations.

One of the key challenges in preparing these bids is determining how to train their current workforce to keep up with the new technology the military continues to develop. NSWC wants to maintain their current workforce without eliminating any current workers. So, if the current workforce does not have the skill set to complete the necessary tasks, training will be required.

In formulating this problem, there are a series of *tasks* which will be completed by a *worker*. Each worker has a particular *skill set* which they have obtained from previous training. In addition, each worker has a skill *level* associated with each skill. These levels range from novice to expert (1-5). So, each worker has a particular skill set defined by the skill level. Similarly, each task has a skill set required to complete the task at a desired quality. So, just as each worker has a skill set with associated skill levels, each task has a skill set with associated skill level.

The problem then becomes a task of assigning tasks to workers. It is possible that only one task is assigned to a worker; but it is also possible for more than one task to be assigned to a worker. If a worker is assigned to a task where his skill level is below the

level required for the task, then a skill gap is present (Figure 1). In this case, the worker will need to be trained to attain the necessary skill level as required by that task. This, of course, incurs a training cost.

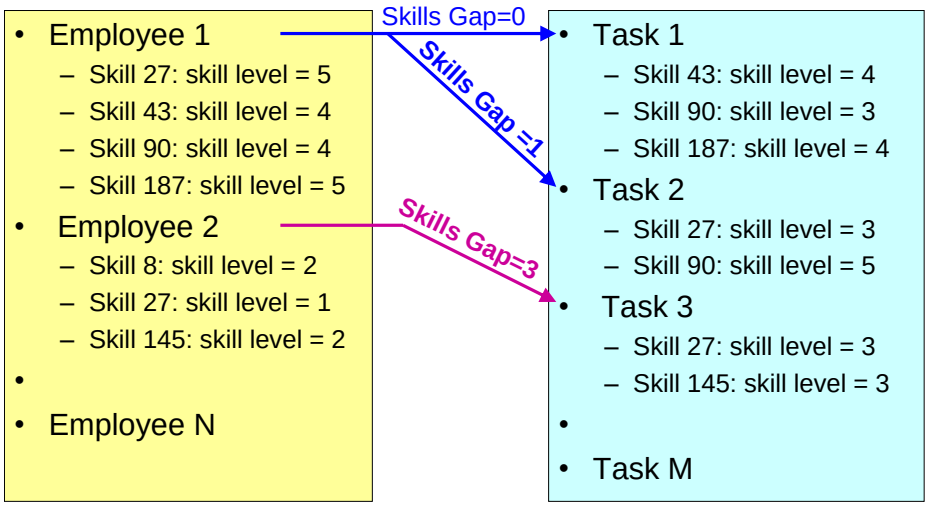

FIGURE 1 – Demonstration of Skill Levels and Skill Gaps

In Figure 1, if Employee 1 is assigned to Task 1, there is no skill gap present. This is because the employee's skill level meets or exceeds the skill level required by the task. On the other hand, if Employee 1 is assigned to Task 2, a skill gap *is* present. This is because Skill 90 requires a level of 5 whereas the employee only has a skill level of 4. Thus, there is a skill gap of 1, and the employee will require training. It is often the case that the employee requires training in more than one skill. This is exemplified by assigning Employee 2 to Task 3. A summarization of this information can also be found in the technical paper by Elmes, Evans, and DePuy (2008).

II. LITERATURE REVIEW

Worker assignment and scheduling problems are faced by virtually all companies. For some companies, it is a simple scheduling problem. For example, hospitals need to schedule enough nurses to cover all patients. The major challenge with this scheduling problem is the length of shifts, when to take breaks, etc. Many manufacturing companies have the challenge of scheduling trained workers to specific jobs. In these instances, the workers are frequently cross-trained and can be assigned to a variety of different tasks.

The general worker assignment problem has been addressed in literature many times. Mazolla and Neebe (1986) developed a branch and bound algorithm to solve the general assignment problem where each potential assignment carries a cost. Even for a simple assignment problem where the only consideration is cost, the formulation is NP-complete and is extremely difficult to solve optimally when using a large data set (over 1,000 workers and 1,000 tasks). For this reason, Mazolla and Neebe developed a branch and bound algorithm to solve the problem.

More recent articles have expanded upon this general assignment problem. For example, Nembhard and Osothsilp (2005) incorporated individual learning and forgetting rates for workers. They concluded it is beneficial to schedule tasks with lower redundancy in order to reduce forgetting. That is, by giving workers a variety of tasks (instead of performing the same tasks repeatedly), workers are less likely to forget what they have learned. Sayin and Karabati (2007) took this one step further by including skill gaps into their formulation. They used a learning curve to model how workers improve their skill levels.

Adding these layers of complexity improves the assignments generated by the model; unfortunately, it also makes it more difficult to solve optimally. Considering the size of many of these problems, heuristics are the only feasible way to generate a solution. For this particular problem, Table I below shows the number of decision variables and constraints for certain problem sizes.

TABLE I

SAMPLE PROBLEM SIZES

workers	skills	tasks	decision variables	constraints
8	6	10	1328	552
8	15	10	3200	1332
15	6	20	2640	1925
15	15	20	6150	4728

Even with a relatively small problem size, the number of decision variables and constraints grows very rapidly. When dealing with very large data sets (over 1,000 workers), the amount of decision variables and constraints make it extremely time consuming to solve optimally. Thus, a heuristic is necessary to solve the problem in a practical amount of time.

Other authors have pointed out that these heuristic methods provide very good results when compared to optimum methods. Kolisch and Hartman (2005) as well as Campbell and Diaby (2002) analyzed several different heuristic techniques and showed that proper heuristics perform very well when compared to the optimum solution. In fact, heuristics are consistently withing 5% of optimum solutions and many times within 2%.

The model developed in this paper is both multi-objective and has a very large data set (over 1,000 workers and over 1,000 tasks). Thus, a heuristic is the only feasible way to solve the problem. However, "no matter how effective a heuristic or algorithm is, there remains a need to use human judgment to find a balance between an organization's conflicting objectives" (Belton and Elder, 1996). In multi-objective formulations, this "human judgment" is incorporated through weights attached to each objective function or through a value function. Cowling *et. al.*(2006) showed that this is not a good method since "objective weights are seldom known in advance" for complex real-world problems. Furthermore, it is unreasonable to expect the user to decide these weights without a thorough knowledge of the mathematical formulation (which a supervisor will not have). As an alternative, the program developed in this paper utilizes iterative input from the user to generate a solution.

III. FORMULATION OF THE PROBLEM

A. Single Objective Function Formulation

This problem was previously solved according to one objective function: to minimize training costs (Douglas, 2006). The formulation of this problem can be found below:

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 (hr) of task j

A_i = capacity (hr) 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 (hr) to raise a worker's skill level on skill k from level l to level m

Decision Variables:

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

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

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

Objective Function:

Minimize Training Cost Minimize $Z_1 = \sum_i \sum_k \sum_m C_{kS_{ik}m} Z_{ikS_{ik}m}$ (1)

Constraints:

Determine Needed Training

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

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

All tasks assigned

$$\sum_{i} X_{ij} = 1 \qquad \forall j \quad (4)$$

All workers assigned at least one task

$$\sum_{j} X_{ij} \geq 1 \qquad \forall i \quad (5)$$

Worker Capacity

$$\sum_{j} T_j X_{ij} + \sum_{k}\sum_{m} E_{kS_{ik}m} Z_{ikS_{ik}m} \leq A_i \qquad \forall i \quad (6)$$

Binary Variables

$$X_{ij} \in \{0,1\}, \quad Z_{ikS_{ik}m} \in \{0,1\}, \quad N_{ik} \in \{0,1\} \qquad \forall i, j, k, m \quad (7)$$

The objective function (1), as previously mentioned, is to minimize the total training cost. The inequalities (2) and (3) are used to determine the total amount of training a worker will need in order to meet the skill levels required by a certain task. Equation (4) is used to make sure that all tasks have been assigned and inequality (5) ensures that each worker is assigned at least one task. Inequality (5) could be removed from the formulation if it was not a requirement to retain the current workforce. Inequality (6) is used to enforce the worker capacity constraint. That is, training time plus the time required to complete the task(s) cannot exceed the total amount of time available to the worker. Lastly, line (7) defines all decision variables to be binary.

When dealing with a problem size of thousands of workers and thousands of tasks, solving this formulation optimally is not computationally feasible. Thus, DePuy *et. al.*

(2006) developed the Meta-RaPS heuristic to solve this formulation on the large scale. Other authors such as Fowler, Wirojanagud, and Gel (2008) and Hartmann and Kolisch (2000) have shown that heuristics provide accurate results using far less computational time.

B. Extension to Multi-Objective Formulation

There are two primary reasons to expand this problem to a multi-objective formulation:

1) It may be beneficial to provide workers with more than the minimal amount of training.
2) In order to include worker preference in training decisions.

The expectation is that considering additional objective functions will improve the quality of the solution presented by the Decision Support System created to solve this worker assignment problem. Much of the information presented in this section is also summarized in the technical paper written by Elmes, Evans, and DePuy (2008).

As with most multi-objective models, it is very common for the objectives to be conflicting (Cowling *et. al.*, 2006). This is very evident with the multi-objective formulation used in this model. Specifically, minimizing total training cost and maximizing the number of skill levels gained are conflicting objectives. The solution which is preferred by the Decision Maker will most likely not be the optimal solutions to one of the objective functions. Instead, the preferred solution will be a compromise of each objective function.

1. Maximize Total Training Received

While minimizing the total training cost has obvious short-sighted benefits, it can actually create the same problem in the future that is currently being solved. The problem is that the workforce is undertrained. In addition, newer and newer technology is being implemented which magnifies the problem of an undertrained workforce. Thus, providing workers with minimal training today will lead to undertrained workers in the future as well. With that in mind, an objective function to maximize training received was included in the model:

Maximize Training Maximize $Z_2 = \sum_i \sum_k \sum_{k' \geq S_{ik}+1} (k' - S_{ik}) Z_{ikS_{ik}k'}$ (8)

The objective function above will maximize the total number of skill levels gained across all skills and all workers.

2. Maximize Worker Satisfaction

Numerous articles have been written highlighting the benefits of worker happiness. There is, obviously, the intrinsic benefit of having a happy workforce. There are also several benefits to the employer which can be seen on the bottom line. According to Sayin and Karabati (2007), "cross-training and skill improvement have been regarded as positive aspects for the self-esteem of the workers and their psychological well-being, which in turn would lead to higher productivity." Even as far back as 1919, George Bell recognized that if "we are to arrive at any real agreement between management and workers to cooperate in increasing production we must conceive of the master aim of the plant as being such production as is compatible with a real and measurable degree of

human happiness and content in the work." Furthermore, Fischer and Sousa-Poza (2009) showed that "improvements in job satisfaction over time appear to prevent workers from (further) health deterioration."

With the addition of one new parameter P_{ik} (defined below), worker preference regarding which skills they learn can be incorporated into the model. Sayin and Karabati (2007) showed that skill acquisition improves worker self-esteem. This model takes that one step further by allowing the workers to specify which skills they wish to acquire. The new parameter and formulation for this objective function can be found below:

Maximize Worker Satisfaction:

$$\text{Maximize } Z_3 = \sum_i \sum_k P_{ik} \sum_{k' \geq S_{ik}+1} (k' - S_{ik}) Z_{ikS_{ik}k'} \qquad (9)$$

$$P_{ik} = \begin{cases} 1 & \text{worker } i \text{ would like additional training in skill } k \\ 0 & \text{otherwise} \end{cases}$$

This objective function is very similar to (8), with the exception that each skill level gained is multiplied by P_{ik}. Thus, the objective function will maximize skill level gains in skills which the worker would prefer to be trained. While (8) seeks to maximize training across all workers and all skills, (9) incorporates worker preference regarding which skills they wish to gain expertise.

As an example, suppose that worker 5 currently has a training level of 2 on skill 9 (i.e. $S_{59} = 2$), but has not expressed an interest in additional training in that skill (i.e., $P_{59} = 0$); then, a solution in which worker 5 were to be trained to level 4 on skill 9 (i.e. $Z_{5924} =$

1) would add 4-2 = 2 to the second objective function Z_2 (8), but 0 to the third objective function Z_3 (9).

3. Goal of the Multi-Objective Decision Support System

Simply put, the goal of this Decision Support System (DSS) is to provide the Decision Maker (DM) with a tool that is easy to use and will incorporate the expertise of the DM in order to find the preferred solution to the worker assignment problem. While Douglas (2006) solved this problem in terms of lowest training cost, it is believed that using a multi-objective approach will present a solution which is more preferred by the DM.

IV. SOLUTION PROCESS

It is important to point out an inherent problem with any multi-objective solution. In actuality, there is no solution which optimizes all objectives simultaneously. Instead, considering multiple objectives at the same time creates a Pareto-front where we have numerous good solutions (Cowling *et. al.*, 2006). How then, do you decide which solution is the best compromise of all the objectives?

It is a long-standing industrial engineering principle that the person who knows best is the person who does the job every day. No good industrial engineer would begin the redesign of a process without talking to the worker who performs that process on a daily basis. With that in mind, the ideal way to select the best compromise solution is to use the knowledge of the person who has been doing the scheduling on a daily basis. This person is usually a supervisor and is referred to in this paper as the Decision Maker (DM).

The solution process used in the Decision Support System (DSS) developed in this paper differs from the solution process used in many Multi-Objective Optimization problems in the past. In these examples, a weight is assigned to each of the individual objective functions and then the problem is solved in one pass. The problem with this process is that the weight assigned to each individual objective function is essentially an arbitrary value selected by the Decision Maker (DM). That is, without a thorough knowledge of how the program runs one cannot assign accurate weights to each objective function. Without accurate weights, the solution provided will not accurately reflect the

DM's idea of a "good" solution. Cowling *et. al.* showed that the error associated with having inaccurate weights on the objective functions can be very costly.

The solution process used for this DSS does not use arbitrary weights assigned to each individual objective function. Instead, the expertise of the DM is used in an iterative fashion in order to guide the DSS to the best compromise solution. Since the DM will not have a detailed knowledge of the mathematical formulation behind the DSS, it is much easier to guide the program through iterative input as opposed to assigning weights to each objective function or by defining a value function.

A. Multiple Stage Decision Support System

The Decision Support System (DSS) is divided up into two primary stages (Figure 2). The first stage is to solve each of the three objective functions independent of one another. Each objective function is solved using the Meta-RaPS process developed by DePuy *et. al.* (2006) Meta-RaPS is employed because the computational time is too large to solve these objective functions optimally when using a large data set (over 1,000 workers and tasks).

The Decision Maker (DM) is presented a configurable number of solutions for each of the three objective functions—usually this is five. Solving each objective function is considered a step of the first stage. Since each objective function was solved independent of the others, the solutions presented to the DM are extremes and none of them are likely what the DM considers to be the best compromise solution. At this point, the DM selects

one of the 15 solutions presented as the preferred solution of that group. This concludes Stage 1.

Stage 2 utilizes iterative input from the DM to execute compromise programming. This allows the DM to guide the DSS towards the best compromise solution. Using the solution identified in Stage 1, the DM is then asked two questions: which objective function should be improved; and how much should the other two be compromised? The DSS utilizes Meta-RaPS again to make trade-offs of worker-task assignments. The DSS will continue to make these trade-offs in order to improve the objective function identified by the DM until the compromise threshold (also identified by the DM) has been met. At this point, the new solution is presented to the DM. If the DM feels that this is the best compromise solution, the DSS is terminated and full results are presented to the DM. If not, Stage 2 is repeated and the DM is again

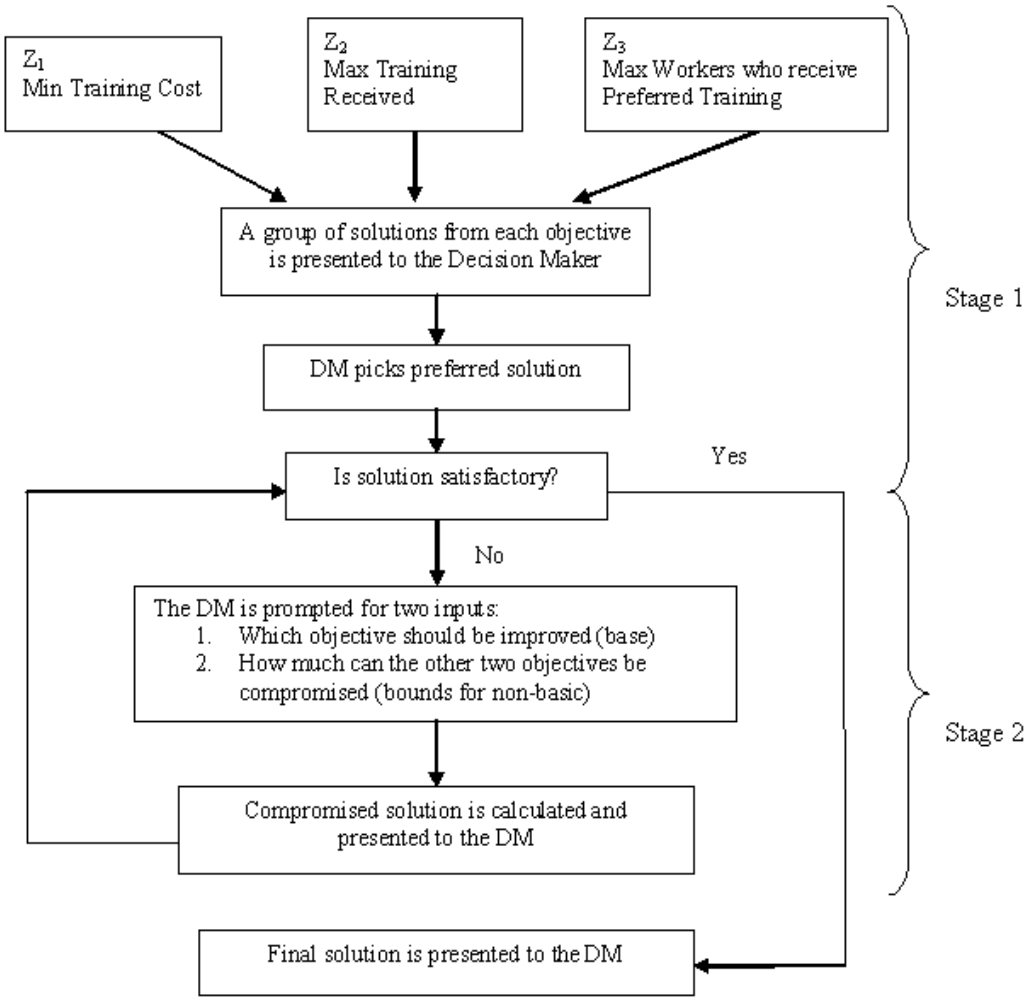

FIGURE 2 – Flowchart of Solution Process

Stage 2 is repeated until the DM has guided the DSS to the solution which he feels is the best compromise solution. This is the final solution, which is presented to the DM.

V. PSEUDO-CODE AND INPUT FROM THE USER

The previous sections outlined the motivation for solving this large worker-task assignment problem in addition to providing a flowchart for the solution process. This section describes the VBA program that the Decision Maker (DM) uses as well as the pseudo-code for the problem solving algorithm. As mentioned before, a modified version of the Meta-RaPS algorithm developed by DePuy *et. al.* is used during each step.

The program (Decision Support System, or DSS) has two different buttons on the input sheet that control the operations. The first button is the "input button" and the second is the "find assignments" button. The input button requires little explanation—it sets up the tables for the DM to input all the necessary data (worker skill levels, etc.). The "find assignments" button actually executes the algorithm. This algorithm has been explained in more detail in previous sections—Figure 2 provides a good summary.

A. Input Button

FIGURE 3 – Input Button

The first button is fairly self-explanatory: it will set up the input sheet for the DM to input values. When the user clicks on "input button," this will clear the sheet of all information. So, it is important to note that if you do not want to lose the information currently on the sheet, do not click the input button unless the data has been saved

somewhere else. Once the input button is clicked, the program will prompt the user for the following information:

- Number of workers in the problem size
- Number of tasks in the problem size
- Number of skills
- Number of solutions to store for each objective in Stage 1

With this information, the program automatically generates all the tables that need to be filled out by the user. These tables can be found below:

- Worker-Skill Matrix
- Task-Skill Matrix
- Task Time Table
- Worker Capacity Table
- Cost to Train Matrix (cost of raising a skill level for each skill)
- Time to Train Matrix (again, for each skill level of each skill)
- Worker Preference Matrix

All of these tables are extremely important and must be filled out accurately by the DM. Obviously, this is the input data for the program, and without good input data, one cannot expect a good solution. The screenshot below shows a portion of the tables the DM fills out. In this example (same example used throughout this section), there are 5 workers, 5 skills, and 5 tasks:

Worker Skill Matrix					
	Skill 1	Skill 2	Skill 3	Skill 4	Skill 5
Worker 1	2	4	4	3	4
Worker 2	3	2	3	4	3
Worker 3	3	5	4	5	4
Worker 4	4	3	2	2	2
Worker 5	3	3	3	3	2
Task Skill Matrix					
	Skill 1	Skill 2	Skill 3	Skill 4	Skill 5
Task 1	4	4	1	1	2
Task 2	1	1	4	4	1
Task 3	3	3	1	1	1
Task 4	1	2	1	4	1
Task 5	1	4	4	1	3

FIGURE 4 – Portion of Input Matrix

B. Find Assignments Button

FIGURE 5 – Find Assignments Button

This button executes a very short program which actually calls upon four larger subprograms. The subprograms, in order, are listed below:

- InitialTrainingCostSoln (Stage 1, Step 1)
- InitialSkillGapSoln (Stage 1, Step 2)
- InitialWorkerPrefSoln (Stage 1, Step 3)
- Compromise (Stage 2)

The main program calls each subprogram individually in order to complete the algorithm. The first three subprograms comprise Stage 1—Find Initial Solutions. Each subprogram runs the Meta-RaPS heuristic to quickly calculate good solutions in regards to each objective function. Each subprogram then displays these solutions on the screen

before moving on to the next subprogram. In this manner, the first three subprograms are extremely similar.

After running the three steps in stage 1, the list of initial solutions are presented to the DM. The DM chooses one of these initial solutions as the "best" from that group. However, as discussed earlier, all of the initial solutions optimize *one* of the objective functions without consideration of the others. Thus, it is unlikely the solution chosen at this point will be the final solution. This solution is simply used as a starting point moving into Stage 2.

The screenshot below continues the example setup earlier with 5 workers, 5 skills, and 5 tasks. In this example, the DM has chosen one of the solutions to Z_2 (Maximize Training). In turn, Z_1 (Minimize Training Cost) and Z_3 (Maximize Worker Satisfaction) are far from optimized.

Iteration	1
Worker	Task
1	3
2	1
3	4
4	2
5	5
Total Cost	57
Total Training Amount	11
Total Worker Preference	7

FIGURE 6 – Initial Solution (Stage 1 complete)

Unlike the first three subprograms, the final subprogram (Stage 2) utilizes iterative input from the DM. Stage 2 uses compromise programming in order to improve one of the objective functions at the sacrifice of the other two. The DM is first prompted for which objective function should be improved. Since the solution used to initialize Stage

2 was one that optimized Z_2 then the available choices are Z_1 or Z_3. The screenshot below shows this prompt:

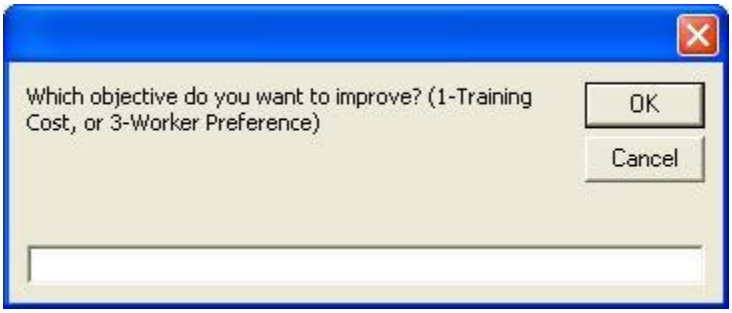

FIGURE 7 – Which Objective to Improve

After the DM specifies which objective function to improve, the DM inputs a percentage that the other two objective functions can be sacrificed. The screenshot for this choice is below:

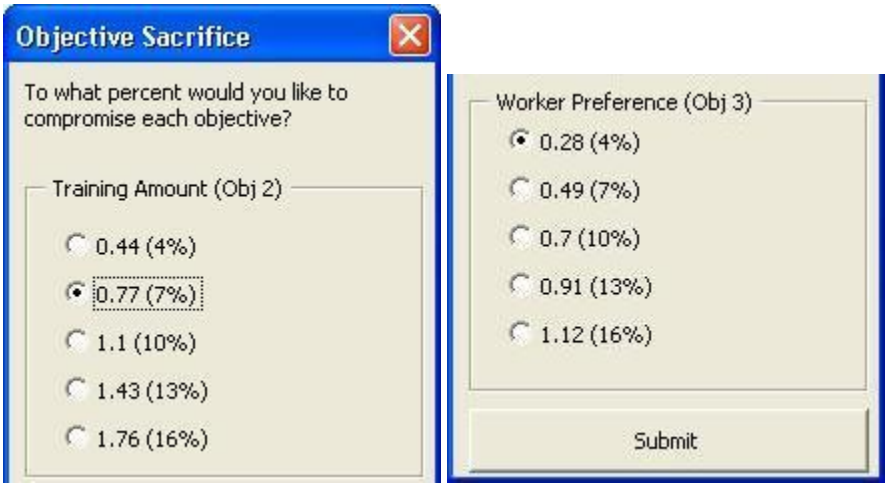

FIGURE 8 – How Much to Sacrifice Each Objective

At this point, the algorithm now has a complete solution to work with in addition to trade-off data between the different objectives provided by the DM. With this information, a series of worker-task assignments swaps are made in order to improve the base objective. The subprogram will continue to make swaps until one (or both) of the

bounds have been violated or there are no more beneficial swaps. The new solution is then presented to the DM:

Iteration	1		2
Worker	Task	Worker	Task
1	3	1	3
2	1	2	2
3	4	3	4
4	2	4	1
5	5	5	5
Total Cost	57		34
Total Training Amount	11		6
Total Worker Preference	7		5

FIGURE 9 – Solution After Compromise Programming

At this point, if the DM is satisfied with the solution, the program is terminated. If the DM is not satisfied, the program will prompt the DM for new trade-off data and repeat Stage 2. This is repeated until the DM has reached the best compromise solution.

It is important to point out again that there is no solution which will optimize all the objective functions simultaneously (Cowling *et. al.*, 2006). Generally speaking, if a multi-objective problem is solved optimally, it is through the use of weights on each objective function or through a value function. It can be very difficult for the DM to specify weights or define a value function. This is because the DM is selected for his expertise in the production setting—not his knowledge of mathematical formulations. The goal of this program is to use iterative input from the DM to guide the algorithm towards the best compromise solution. This is a much more intuitive way for the DM to use the program.

C. Pseudo-code for Decision Support System (DSS)

This section summarizes the code used for the DSS. There are four primary sections of code to analyze. One section for each of the three objective functions and the fourth section is for the compromise programming.

1. Key Variables used in Pseudo-code

Before jumping into the pseudo-code, it is beneficial to review some of the key variables used throughout the code. First of all, it is important to note the similarities between the three steps in the first stage. Take the first step for example; many of the decisions about worker-task assignments are based on the variables 'totaltaskcost' and 'totalworkercost.' These variables are calculated using the 'workertaskcost' variable, which is a large array that holds the cost associated with training any worker to any task. So, 'totaltaskcost' is simply the sum of all workertaskcost values for a specific task. Similarly, 'totalworkercost' is the sum of all workertaskcost values for a specific worker. These 'total' variables are used to determine who is the least trained worker (maxcostworker, for phase 1) and which is the most difficult task (maxcosttask, for phase 2). More detail on how the worker-task assignments are made can be found in the following section. The intention of this section is to bring attention to the similarities between the three steps in the first stage.

In Stage 1, Step 1, the program uses variables such as 'totaltaskcost,' 'workertaskcost,' etc. as outlined above. For Step 2, the program uses variables such as 'totaltasksgap' and 'workertasksgap.' Similarly, for Step 3, the program uses variables such as 'totaltaskpref' and 'workertaskpref.' Each step uses the "total" variables

associated with the objective function for making worker-task assignments; however, it does keep track of all "total" variables (i.e. the ones for the other objective functions). While the other "total" variables are not used for determining worker-task assignments, they are used to calculate the objective function values once a complete assignment has been made. Also, these values are used in Stage 2, where certain objectives are being compromised for the benefit of another.

2. Generalized Meta-RaPS Pseudo-code

The pseudo-code below shows the generalized form of the Meta-RaPS heuristic. Each of the three initial solution programs outlined below follows this generalized code:

```
Do Until max number of iterations is reached
    Do Until all tasks are assigned
        Identify the greedy assignment
        If random value <= priority percentage Then
            Make greedy assignment
        Else
            Develop a candidate list of other assignments close in value to the
            greedy assignment
            Choose an assignment from the candidate list
        End If
    Loop
    Place the solution in its ranked position compared to all other iterations
Loop
```

Sections 3-5 show how the Meta-RaPS heuristic is applied to each objective function. Meta-RaPS is actually run twice for each objective function—once to make sure each worker is assigned one task and the second time to assign all remaining tasks. If there is not a constraint to retain the current workforce, the program can jump straight to the second use of Meta-RaPS and simply assign all tasks.

3. InitialTrainingCostSoln Pseudo-code

Once the DM clicks the "find assignments" button, this is the first subprogram which is run. This subprogram uses the Meta-RaPS heuristic to determine worker-task assignments. The pseudo-code for this subprogram is below:

```
Do Until max number of iterations is reached
    Do Until each worker has a task assigned
        Identify the maxcostworker
        Identify mincosttask for maxcostworker
        If random value <= priority percentage Then
            Assign mincosttask to maxcostworker
        Else
            If worker is unassigned and totalworkercost >=
            maxcostworker*restriction percent Then
                If task is unassigned and cost to assign worker to that task
                <= mincost*(1+restriction percentage) Then
                    This worker-task assignment goes on candidate list
                End If
            End If
            Use random number to select a worker-task assignment from the
            candidate list
        End If
    Loop

    Do Until all tasks are assigned
        Identify maxcosttask
        Identify mincostworker
        If random value <= priority percentage Then
            Assign mincostworker to maxcosttask
        Else
            If task is unassigned and totaltaskcost >= maxcosttask*restriction
            percent Then
                If cost to assign worker to that task <=
                mincost*(1+restriction percentage) Then
                    This worker-task assignment goes on candidate list
                End If
            End If
            Use a random worker to select a worker-task assignment from the
            candidate list
        End If
        Update objective function values and worker skill levels
    Loop
    Eliminate the solution if it is a duplicate from a previous iteration
    Place the solution in its ranked position compared to all other iterations
Loop
```

The complete code can be found in Appendix E. The two phases for the above section of code are very similar. They both use the Meta-RaPS heuristic, which is a modified greedy algorithm. The first step is to identify the worker who has the highest training cost across all tasks. Next, the program identifies the task which can be assigned to that worker at the minimum cost. This would be the greedy assignment, and according to Meta-RaPS, this assignment is sometimes made. However, Meta-RaPS differs from a greedy algorithm because it sometimes makes an assignment from a "candidate list." The candidate list contains other worker-task assignments that are within a certain percentage of the greedy assignment. This selection is made using random numbers.

Phase 1 continues assigning tasks to workers who are currently unassigned. Once each worker has a task assignment, the program moves to phase 2, where the rest of the tasks are assigned. Phase 2 is extremely similar to phase 1 since they both employ the Meta-RaPS heuristic. The difference is that phase 1 identifies a worker for assignment and then finds a task for that worker. On the other hand, phase 2 identifies a task first, and then finds a worker to assign to that task. Once phase 2 is complete, a complete worker-task assignment has been completed. Next, the program determines how good the current solution is compared to solutions from other iterations.

The first step in determining how the current solution compares to other iterations is to check if the current solution is a duplicate solution. All duplicate solutions are thrown out and a new iteration begins. However, considering the problem size and the randomness of Meta-RaPS, there are not many duplicate solutions. If a unique solution is generated, the program determines what rank the solution is. If it is not a top ranking

solution (the number of solutions to store is a user input), then the program moves to the next iteration. If the current solution is a unique, top ranking solution, then it is saved in a best solution matrix. Once all the iterations are complete, the top ranking solutions are printed on the screen and the program moves on to the second subprogram (Stage 1, Step 2): InitialSkillGapSoln.

4. InitialSkillGapSoln Pseudo-code

The program automatically starts this sub-program after the first sub-program (InitialTrainingCostSoln; Stage 1, Step 1) is completed. The pseudo-code for this sub-program is below:

```
Do Until max number of iterations is reached
        Do Until each worker has a task assigned
                Identify the maxsgapworker
                Identify maxsgaptask for maxsgapworker
                If random value <= priority percentage Then
                        Assign maxsgaptask to maxsgapworker
                Else
                        If worker is unassigned and totalworkersgap >=
                        maxsgapworker*restriction percent Then
                                If task is unassigned and the skill gap closed >=
                                maxsgap*restriction percent Then
                                        This worker-task assignment goes on candidate list
                                End If
                        End If
                        Use random number to select a worker-task assignment from the
                        candidate list
                End If
                Update objective function values and worker skill levels
        Loop

        Do Until all tasks are assigned
                Identify maxsgaptask
                Identify maxsgapworker
                If random value <= priority percentage Then
                        Assign maxsgapworker to maxsgaptask
                Else
```

```
                    If task is unassigned and totaltasksgap >= maxsgaptask*restriction
                    percent Then
                        If skill level gained >= maxsgap*restriction percent Then
                            This worker-task assignment goes on candidate list
                        End If
                    End If
                    Use a random worker to select a worker-task assignment from the
                    candidate list
                End If
                Update objective function values and worker skill levels
            Loop
            Eliminate the solution if it is a duplicate from a previous iteration
            Place the solution in its ranked position compared to all other iterations
Loop
```

Where the first sub-program identified the least trained worker (according to training costs), and assigned that worker the most inexpensive task, this sub-program is actually *maximizing* the total number of skill levels gained. So, the greedy assignment would be the worker with the fewest total skill levels and the task which would gain that worker the most skill levels. The candidate list is comprised of other worker-task assignments that increase the worker's skill levels within a certain percentage of the greedy assignment.

This sub-program is very similar to the first one (and the last one). This sub-program has two phases which both employ Meta-RaPS. Also, this subprogram uses the same ranking system. The major difference between each sub-program is that each one solves a different objective function.

Once this sub-program is complete, it will print the top results on the screen and then automatically move on to the last sub-program in Stage 1: InitialWorkerPrefSoln.

5. InitialWorkerPrefSoln Pseudo-code

The program automatically starts this sub-program after the first sub-program (InitialTrainingCostSoln; Stage 1, Step 1) is completed. The pseudo-code for this sub-program is below:

```
Do Until max number of iterations is reached
    Do Until each worker has a task assigned
        Identify the maxprefworker
        Identify maxpreftask for maxprefworker
        If random value <= priority percentage Then
            Assign maxpreftask to maxprefworker
        Else
            If worker is unassigned and totalworkerpref >=
            maxprefworker*restriction percent Then
                If task is unassigned and the skill levels gained * worker
                preference >= maxsgap*restriction percent Then
                    This worker-task assignment goes on candidate list
                End If
            End If
            Use random number to select a worker-task assignment from the
            candidate list
        End If
        Update objective function values and worker skill levels
    Loop

    Do Until all tasks are assigned
        Identify maxpreftask
        Identify maxprefworker
        If random value <= priority percentage Then
            Assign maxprefworker to maxpreftask
        Else
            If task is unassigned and totaltaskpref >= maxpreftask*restriction
            percent Then
                If skill level gained *worker preference >=
                maxpref*restriction percent Then
                    This worker-task assignment goes on candidate list
                End If
            End If
            Use a random worker to select a worker-task assignment from the
            candidate list
        End If
        Update objective function values and worker skill levels
    Loop
    Eliminate the solution if it is a duplicate from a previous iteration
    Place the solution in its ranked position compared to all other iterations
```

Loop

This subprogram is extremely similar to the previous: InitialSkillGapSoln. The worker-task assignments are still determined using the amount of skill levels gained on a particular task; however, for this objective function, the skill levels gained is also multiplied by a binary worker preference value. Consider the following hypothetical example: if worker 1 is assigned to task 1, he will gain 3 skill levels in both skills 1 and 2. In addition, worker 1 has a preference value of 0 for skill 1 and 1 for skill 2. That is, he does not have a desire to learn skill 1, but does want to learn skill 2. In step 2, where preferences are not considered, both skills would contribute a value of 3 to the objective function if this assignment was selected. However, for step 3, the skill levels gained is multiplied by the preference value. Thus, skill 1 would contribute 0 to the objective function since the worker preference value is 0 also. On the other hand, skill 2 would still add 3 to the objective function since the preference value is 1 for that skill. Here is another situation to consider in understanding the relationship between step 2 and 3. If the worker preference matrix was all set to 1, then the objective function for step 2 and 3 would be equivalent.

The results from this subprogram are displayed on the screen for the DM to reference along with the other two subprograms. Complete worker-task assignments are printed on the screen; however, for large data sets, it is not feasible to compare the different solutions based on the individual assignments. For this reason, there is a summary at the top of the page which simply shows the three objective function values associated with each initial solution. The DM is then prompted to identify which of the initial solutions is the preferred solution. At this point, the program is using input from

the DM to help guide the algorithm towards a solution which best matches the preferences of the DM.

This concludes the first stage of the program. Next, the program automatically moves into stage 2—Compromise Programming.

6. Compromise Programming Pseudo-code

Compromise Programming makes up the second stage of the program, and is iterative in nature. At this point, the DM has identified a single solution to use in the second stage. Two pieces of information are prompted from the DM:

- Which objective function to improve
- Bounds on the other objective functions

Using this information, the program will continue to swap worker-task assignments until one of the bounds on the other objective functions have been violated. The pseudo-code for stage 2 is below:

```
Do Until usersatisfied=1
        Prompt User for which objective to improve
        Option Box used to input bounds on the other two objective functions
        For i = 1 to numworkers
        For j = 1 to numtasks
                If worker i is assigned to task j Then
                        For i2 = 1 to numworkers
                        For j2 = 1 to numworkers
                                Calculate the affect this swap has on base and store than in
                                a matrix
                        Next j2
                        Next i2
                End If
        Next j
        Next i
```

```
        Do Until bound on either non-basic objective function is violated Or no swap
        improves base
                Determine which swap will increase the base the most and make that swap
                Update worker skill levels and objective function values
        Loop
        Print the new solution on the screen
        Ask if the user is satisfied (usersatisfied=[1 for yes, 0 for no])
Loop
```

Right before the final loop, the program asks if the DM is satisfied with the current solution. If he is not, then the subprogram loops to the top and asks the DM to specify a base and bounds on non-basic objective functions once again. Hence, the DM has complete control in manipulating the solution to accurately reflect his preferences. He can continue to select the same objective function as the base to improve it more, or he can select a different base to "guide" the program in another direction. Additionally, this goal is achieved without assigning weights to each objective function at the beginning of the program. Instead, the DM's knowledge and experience is used later in the program, and in a manner that is simple for the DM. After all, it is much easier for someone inexperienced with how the program runs to specify bounds on objective functions than to assign arbitrary weights at the beginning of the program.

VI. RESULTS

This section outlines the results calculated using several different data sets. All three objective functions are analyzed, though the Minimize Training Cost is done so in less detail due to the fact that Douglas (2006) already examined using Meta-RaPS on this objective function in great detail. Similarly, this paper did not analyze the different values that could be used for percent restriction and percent priority which could be used. These values are used when the program decides whether to make a greedy assignment or to choose an assignment from the candidate list. Douglas (2006) determined that 30 and 80 for percent restriction and priority, respectively, was best suited for this assignment problem. These values were used in this data analysis.

Since Meta-RaPS executes very quickly and is quite random, it is very beneficial to use a large number of iterations in order to find a near optimal solution. Douglas (2006) recommended using 5000 iterations. However, it was determined that a much smaller value is sufficient (200 iterations). While the program is running, the user can see the top list of solutions currently found. Meta-RaPS is surprisingly effective at finding a good solution quickly. Thus, a small number of iterations is sufficient. However, it should be noted that when used in a practical setting, it would be recommended to use a larger number of iterations to guarantee a good solution. For that matter, it would be wise to tweak each of the parameter settings depending on the specific application.

Microsoft VBA for Excel was used to program the Meta-RaPS algorithm. While there are other programming languages that would have been more efficient, Excel is commonly used in practical settings. Thus, it is a very familiar way for the user to input

data into the model. In addition, since most companies already own and use Microsoft Excel, there is no need to purchase any additional software.

The optimum solutions were calculated using LINGO mathematical software. The code used can be found in Appendix E. The results for each objective function are outlined in the following subsections.

A. Minimize Training Cost Results

Table II summarizes the results for the Minimize Training Cost objective function. The full results can be found in Appendix A. Run times are in seconds unless otherwise noted.

TABLE II

MINIMIZETRAINING COST RESULTS

workers	skills	tasks	Meta-RaPS	run-time	Optimum	run-time	%diff
8	6	10	121.6	8.5	116.5	12.9	4.2%
8	15	10	532.6	8.1	513	1.75hr	3.8%
15	6	20	220.7	12.4	172	31.5hr	28.3%
15	15	20	801	11.56	-	-	-

When using a small data set, Meta-RaPS is extremely effective at finding a close to optimum solution. However, once a larger data set was tested, Meta-RaPS was on average close to 30% worse than optimum. Obviously, it would be preferred to have a solution closer to optimum. This is where raising the number of iterations would be beneficial. These trial runs were used with only 200 iterations. This is reflected in the very fast run times. Certainly, a run time of 12.4 seconds is greatly superior to a run time of over 31 hours (the 15 worker, 6 skills data set). Using a larger number of iterations

would produce results closer to optimum and still have a much shorter run time than the optimum method.

B. Maximize Skill Levels Gained Results

Table III summarizes the results for the Maximize Skill Levels Gained objective function. The full results can be found in Appendix B. Run times are in seconds unless otherwise noted.

TABLE III

MAXIMIZE SKILL LEVELS GAINED RESULTS

workers	skills	tasks	Meta-RaPS	run-time	Optimum	run-time	%diff
8	6	10	53.4	13.3	63.4	1	15.8%
8	15	10	128.9	9.0	147	7	12.3%
15	6	20	108.7	14.5	138	2	21.2%
15	15	20	233.1	13.5	294	409	20.7%

It can be seen from the table that Meta-RaPS produces solutions which are close to optimum. While a little more than 20% from optimum may seem like a significant difference, this is in part because only 200 iterations were used. A larger number of iterations would produce a solution closer to the optimum.

However, there is an important trade-off between the number of iterations to use and the run-time of the program. LINGO is able to produce optimum results for this objective function (as well as Maximize Worker Preference) much faster than the Minimize Training Cost objective function. Thus, it is not as easy to simply state that raising the iterations will solve the problem. However, on the other hand, Meta-RaPS

already produces solutions closer to optimum compared to the first objective function. So, there is not as much a need to raise the number of iterations.

C. Maximize Worker Preference Results

Table IV summarizes the results for the Maximize Worker Preference objective function. The full results can be found in Appendix C. Run times are in seconds unless otherwise noted.

TABLE IV

MAXIMIZE WORKER PREFERENCE RESULTS

workers	skills	tasks	Meta-RaPS	run-time	Optimum	run-time	%diff
8	6	10	30.6	14.6	39.2	1.1	21.9%
8	15	10	76.0	9.6	96	1	20.8%
15	6	20	66.9	14.2	98	2	31.8%
15	15	20	139.2	13.8	187	81	25.6%

This objective function is very similar in nature to the Maximize Skill Levels Gained objective function. However, you can see from the percent difference, that Meta-RaPS was not as effective with this objective function. On the other hand, Meta-RaPS still generated solutions closer to optimum than the first objective function (Minimize Total Cost). This is a promising result and indicates that Meta-Raps is an effective method for generating solutions to all three objective functions.

An interesting thing to point out with this data set is the run-time for calculating the optimum solution. When comparing the optimum solution to Meta-RaPS run time for the first objective function (total cost), the optimum solution took over 31 hours compared to only 12.4 seconds for Meta-RaPS. There is nowhere near the same disparity when we

look at the results for this objective function. At the same time, the optimum solution did take more than five times as long (81 seconds) compared to Meta-RaPS (13.8) seconds. When using an extremely large data set (over 1,000 workers), this difference in run-time will be very beneficial.

D. Compromise Programming Results

Unfortunately, there is no easy way to analyze results for the second stage of the program. The only way to determine an optimum solution would be to run the model through LINGO. However, to do that, each of the three objective functions would need to have a weight assigned to it or a value function is neccesary. The premise to this paper is to avoid using those techniques. Instead, the knowledge and expertise of the DM guides the program to the best compromise solution. In a way, this interaction does assign weights to each objective function, but it is done in a more intuitive fashion. This will help the DM more accurately weigh the importance of each objective function.

This idea is supported by Cowling *et. al.*(2006) as well as Belton and Elder (1996). They point out that using inaccurate weights can have very costly effects on the quality of the solution. Since this program is designed for a supervisor in the field to use, it can be assumed that the user will not have a detailed knowledge of the mathematical formulation and thus cannot assign accurate weights. This is by design. Instead, the user can guide the program towards the best compromise solution using the knowledge he has gained in the field.

VII. CONCLUSIONS AND RECOMMENDATIONS

The goal of this paper is to extend the Decision Support System (DSS) developed by Douglas (2006) to a multi-objective formulation. The two objective functions to maximize skill levels gained as well as to maximize worker preference were incorporated in with the original objective function to minimize total cost. Clearly, the objective function to maximize skill levels gained contradicts the original objective function to minimize cost because training costs money. However, as Cowling *et. al.*(2006) pointed out, virtually every real world multi-objective problem has "many and contradictory objectives." How then, do we determine the best solution?

It is a long-held Industrial Engineering principle that the person who knows how to do the job the best is the person who has been doing it the longest. Enter: the Decision Maker (DM). The Decision Support System (DSS) developed in this paper is designed to take advantage of the knowledge the DM possesses from working in the field for many years. Hence, the logical selection for the DM is a supervisor.

Many multi-objective formulations in the past assign a weight to each objective function in order to solve the formulation as one larger optimization problem. Cowling *et. al.*(2006) show that the cost of using inaccurate weights in this type of formulation can have "severe" detrimental effects. For this program, we assume that the DM will not be able to assign accurate weights to the objective functions because this person does not have a deep understanding of the mathematical formulation—that is not the DM's purpose. The DM is being selected because of his knowledge of the production setting

(worker competencies, creating schedules, etc.). Thus, the DM's knowledge is used in an iterative manner to arrive at the best compromise solution in the eyes of the DM.

One thing that makes this program unique is that it is designed to be used for *very* large problem sizes (thousands of workers). This presents a problem regarding solution time. Instead of solving each of the three objective functions optimally, they are solved using the Meta-RaPS heuristic developed by DePuy, *et. al.* (2003) and adapted by Douglas (2006) to be used for the assignment problem. Solving each objective function independently is stage 1 of the program. These solutions (a configurable number) are then fed into stage 2 of the program—which incorporates input from the DM.

Once a pool of initial solutions has been generated, the DM selects the one he feels is closest to the "best" solution. From there, the program utilizes compromise programming to adjust the solution as the DM sees fit. More specifically, the DM identifies which objective function should be improved and then provides bounds on the other objective functions. The program then makes trade-offs in the worker-task assignments in order to improve the identified objective function at the cost of the other. This process can be repeated as many times as necessary until the DM has a solution he is satisfied with. In this manner, the DM is essentially assigning weights to each objective function. These weights are determined through the DM's iterative input regarding trade-off values.

While this concept appears to be very effective, it has yet to be tested using real-world data. There are several other areas that provide good opportunity for future research. For example, the VBA code was written by an Industrial Engineer—not a computer programmer. As a result, the code is very inefficient, and cleaning up the code

would most likely reduce computational times. This could be done through code refactoring. Code refactoring would shorten the length of the code as well as remove unnecessary steps. A shorter run time would have huge benefits in practical applications.

Another area for improvement is in the compromise programming. Currently, when the DM is specifying how much to sacrifice the objective functions (for the benefit of another) these ranges are predefined percentages. It is possible to program this so that the DM can enter his preferred percentage instead of choosing one from the option list. Another way to get this input from the DM would be to include a slider. This way the DM could move the slider to the percentage that he prefers. This provides more flexibility while still maintaining ease of use.

One downside to the current input sheet is that it is very time consuming to enter in data. When there are over 1,000 workers and tasks, it will take the DM a very long time to input skill levels for each worker and task. There may be a way to group certain workers together (by department, by years of experience, etc.). This way, the DM could input a smaller amount of data. Similarly, there may be a way to group tasks together in order to reduce input time.

Lastly, another area for improvement is with the parameters used in the program. As mentioned during the Results section, Douglas (2006) concluded that the best values for percent restriction and percent priority were 30 and 80, respectively. These values were only tested for the first objective function (Minimize Training Cost). However, these values were also used for the two new objective functions introduced in this paper. The results could have been more accurate it different parameters were used for these

objective functions. On the other hand, this is somewhat a moot point because the parameters for all three objective functions would need to be validated for a real-world problem.

LIST OF REFERENCES

Bell, George. 1919. Production the Goal. *Annals of the American Academy of Political and Social Science.* Vol. 85, Modern Manufacturing. A Partnership of Idealism and Common Sense, 1-7

Belton, V., Elder, M.D. 1996. Exploring a Multicriteria Approach to Production Scheduling. *The Journal of the Operational Research Society.* Volume 47, Issue 1, 162-174

Campbell, G.M., Diaby, M. 2002. Development and Evaluation of an Assignment Heuristic for Allocating Cross-trained Workers. *European Journal of Operational Research.* Volume 138, Issue 1, 9-20

Cowling, P., Colledge, N., Keshav, D., Remde, S. 2006. The Trade Off between Diversity and Quality for Multi-objective Workforce Scheduling. *Lecture Notes in Computer Science,* Volume 3906/2006, 13-24

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

Douglas, Allison M. 2006. A Modified Greedy Algorithm for the Task Assignment Problem. Master of Engineering Thesis, University of Louisville.

Elmes, B.B., Evans, G.W., DePuy, G.W., 2008. A Multi-Objective Decision Support System for Workforce Training. *Proceedings of the 2008 Industrial Engineering Research Conference*

Fischer, J., Sousa-Poza, A. 2009. Does Job Satisfaction Improve the Health of Workers? New Evidence Using Panel Data and Objective Measures of Health. *Health Economics,* Volume 18, Issue 1, 71-89

Fowler, J., Wirojanagud, P., Gel, E. 2008. Heuristics for workforce planning with worker differences. *European Journal of Operational Research.* Volume 190, Issue 3, 724-740.

Hartmann, S., Kolisch, R. 2000. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research.* Volume 127, Issue 2, 394-407

Kolisch, R., Hartmann, S. 2005. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research.* Volume 174, Issue 1, 23-37

Nembhard, D.A., Osothsilp, N. 2005. Learning and Forgetting-Based Worker Selection for Tasks of Varying Complexity. *The Journal of the Operational Research Society* Volume 56, Issue 5,576-587

Mazolla, J.B., Neebe, A.W. 1986. Resource-Constrained Assignment Scheduling. *Operations Research.* Volume 34, Issue 4, 560-572

Sayin, S., Karabati, S. 2007. Assigning cross-trained workers to departments: A two-stage optimization model to maximize utility and skill improvement. *European Journal of Operational Research.* Volume 176, Issue 3, 1643-1658

APPENDIX A—Results for Minimize Total Training Cost

The table below has the full results for the Minimize Training Cost objective function. The majority of the calculations were performed on a desktop computer with an Intel Pentium 4 3.0GHz processor with 1GB of ram. Some of the run times had to be discarded because they were performed on a laptop with faulty hardware that skewed the run times.

For each trial run, the data was randomly generated. Summaries of this data can be found the Results section of the text.

Trial	Workers	Skills	Tasks	Meta-RaPS	run-time	Optimum	run-time	%diff
1	8	6	10	70	8.3	70	2	0.0%
2	8	6	10	94		87	13	8.0%
3	8	6	10	95		91	3	4.4%
4	8	6	10	161	11.4	161	21	0.0%
5	8	6	10	82		77	12	6.5%
6	8	6	10	102		94	5	8.5%
7	8	6	10	145		142		2.1%
8	8	6	10	128		128	8	0.0%
9	8	6	10	69		67	6	3.0%
10	8	6	10	171		163		4.9%
11	8	6	10	124	8.2	112	10	10.7%
12	8	6	10	153	8.4	153	18	0.0%
13	8	6	10	152		145	23	4.8%
14	8	6	10	167	7.9	161	49	3.7%
15	8	6	10	120	7.6	109	4	10.1%
16	8	6	10	112	7.6	104	6	7.7%
17	8	15	10	513	7.8	513	6377s	0.0%
18	8	15	10	514	8.2			0.2%
19	8	15	10	548	8			6.8%
20	8	15	10	545	7.9			6.2%
21	8	15	10	518	7.9			1.0%
22	8	15	10	546	7.9			6.4%

23	8	15	10	533	8			3.9%
24	8	15	10	528	10			2.9%
25	8	15	10	536	7.9			4.5%
26	8	15	10	548	7.9			6.8%
27	8	15	10	528	7.9			2.9%
28	8	15	10	529	8			3.1%
29	8	15	10	532	8			3.7%
30	8	15	10	528	7.9			2.9%
31	8	15	10	528	7.9			2.9%
32	8	15	10	548	7.9			6.8%
33	15	6	20	223	12.4	172	31.5hr	29.7%
34	15	6	20	211	11.9			22.7%
35	15	6	20	208	12			20.9%
36	15	6	20	220	12			27.9%
37	15	6	20	223	11.8			29.7%
38	15	6	20	204	11.8			18.6%
39	15	6	20	220	11.9			27.9%
40	15	6	20	237	12.1			37.8%
41	15	6	20	209	12.9			21.5%
42	15	6	20	224	11.8			30.2%
43	15	6	20	234	11.8			36.0%
44	15	6	20	232	11.8			34.9%
45	15	6	20	234	11.9			36.0%
46	15	6	20	222	13.2			29.1%
47	15	6	20	232	11.9			34.9%
48	15	6	20	206	13			19.8%
49	15	6	20	223	11.9			29.7%
50	15	6	20	224	16.7			30.2%
51	15	6	20	220	11.9			27.9%
52	15	6	20	225	12			30.8%
53	15	6	20	202	15.4			17.4%
54	15	6	20	218	11.9			26.7%
55	15	6	20	224	12			30.2%
56	15	15	20	778	11.7	N/A	N/A	
57	15	15	20	814	11.7			
58	15	15	20	778	11.4			
59	15	15	20	795	11.5			
60	15	15	20	770	11.7			
61	15	15	20	807	11.6			
62	15	15	20	820	11.4			
63	15	15	20	777	11.5			

64	15	15	20	800	11.6			
65	15	15	20	773	11.5			
66	15	15	20	821	11.5			
67	15	15	20	848	11.6			
68	15	15	20	792	11.7			
69	15	15	20	809	11.5			
70	15	15	20	813	11.5			
71	15	15	20	811	11.6			
72	15	15	20	811				

APPENDIX B—Results for Maximize Skill Levels Gained

The table below has the full results for the Maximize Skill Levels Gained objective function. The calculations were performed on a desktop computer with an Intel Pentium 4 3.0GHz processor with 1GB of ram.

Trial	Workers	Skills	Tasks	Meta-RaPS	run-time	Optimum	run-time	%diff
1	8	6	10	47	9.1	53	1	11.3%
2	8	6	10	52		61	1	14.8%
3	8	6	10	51		61	1	16.4%
4	8	6	10	53	12.5	64	1	17.2%
5	8	6	10	52		63	1	17.5%
6	8	6	10	50		56	1	10.7%
7	8	6	10	54		68	1	20.6%
8	8	6	10	62		75	1	17.3%
9	8	6	10	45		58	1	22.4%
10	8	6	10	62		70	1	11.4%
11	8	6	10	48	24.1	58	1	17.2%
12	8	6	10	53	22.3	63	1	15.9%
13	8	6	10	64		73	1	12.3%
14	8	6	10	64	8.4	78	1	17.9%
15	8	6	10	52	8.4	59	1	11.9%
16	8	6	10	45	8.4	54	1	16.7%
17	8	15	10	129	8.7	147	7	12.2%
18	8	15	10	128	9.2			12.9%
19	8	15	10	129	9.1			12.2%
20	8	15	10	128	8.9			12.9%
21	8	15	10	131	9			10.9%
22	8	15	10	128	8.9			12.9%
23	8	15	10	131	8.9			10.9%
24	8	15	10	130	10.5			11.6%
25	8	15	10	130	8.9			11.6%
26	8	15	10	128	8.9			12.9%
27	8	15	10	128	8.8			12.9%
28	8	15	10	128	8.9			12.9%
29	8	15	10	129	8.9			12.2%
30	8	15	10	130	9.2			11.6%

31	8	15	10	129	8.9			12.2%
32	8	15	10	126	8.9			14.3%
33	15	6	20	108	13.5	138	2	21.7%
34	15	6	20	110	13.8			20.3%
35	15	6	20	107	13.8			22.5%
36	15	6	20	107	13.8			22.5%
37	15	6	20	111	13.7			19.6%
38	15	6	20	108	13.9			21.7%
39	15	6	20	108	15.7			21.7%
40	15	6	20	108	13.8			21.7%
41	15	6	20	109	14.3			21.0%
42	15	6	20	109	16.1			21.0%
43	15	6	20	109	13.7			21.0%
44	15	6	20	110	13.8			20.3%
45	15	6	20	109	13.8			21.0%
46	15	6	20	111	14.3			19.6%
47	15	6	20	108	13.9			21.7%
48	15	6	20	110	13.7			20.3%
49	15	6	20	109	13.5			21.0%
50	15	6	20	109	24.8			21.0%
51	15	6	20	109	13.7			21.0%
52	15	6	20	108	13.7			21.7%
53	15	6	20	108	14.7			21.7%
54	15	6	20	108	14			21.7%
55	15	6	20	108	13.7			21.7%
56	15	15	20	235	13.7	294	409	20.1%
57	15	15	20	234	13.6			20.4%
58	15	15	20	236	13.5			19.7%
59	15	15	20	234	13.6			20.4%
60	15	15	20	233	13.7			20.7%
61	15	15	20	232	13.4			21.1%
62	15	15	20	233	13.2			20.7%
63	15	15	20	232	13.7			21.1%
64	15	15	20	232	13.6			21.1%
65	15	15	20	230	13.5			21.8%
66	15	15	20	235	13.6			20.1%
67	15	15	20	230	13.4			21.8%
68	15	15	20	234	13.4			20.4%
69	15	15	20	234	13.7			20.4%
70	15	15	20	234	13.5			20.4%
71	15	15	20	232	13.4			21.1%

72	15	15	20	232				21.1%

APPENDIX C—Results for Maximize Worker Preference

The table below has the full results for the Maximize Worker Preference objective function. The calculations were performed on a desktop computer with an Intel Pentium 4 3.0GHz processor with 1GB of ram.

Trial	Workers	Skills	Tasks	Meta-RaPS	run-time	Optimum	run-time	%diff
1	8	6	10	17	9.7	24	1	29.2%
2	8	6	10	29		33	1	12.1%
3	8	6	10	29		34	2	14.7%
4	8	6	10	37	12.9	44	1	15.9%
5	8	6	10	34		42	1	19.0%
6	8	6	10	36		43	1	16.3%
7	8	6	10	32		41	1	22.0%
8	8	6	10	36		49	1	26.5%
9	8	6	10	24		37	1	35.1%
10	8	6	10	29		42	1	31.0%
11	8	6	10	30	29.6	36	1	16.7%
12	8	6	10	30	23.8	39	1	23.1%
13	8	6	10	27		35	1	22.9%
14	8	6	10	37	8.5	48	1	22.9%
15	8	6	10	33	9	39	1	15.4%
16	8	6	10	30	9	41	1	26.8%
17	8	15	10	76	9.3	96	1	20.8%
18	8	15	10	76	9.7			20.8%
19	8	15	10	77	9.6			19.8%
20	8	15	10	76	9.6			20.8%
21	8	15	10	76	9.5			20.8%
22	8	15	10	78	9.6			18.8%
23	8	15	10	75	9.5			21.9%
24	8	15	10	74	10.7			22.9%
25	8	15	10	78	9.6			18.8%
26	8	15	10	76	9.6			20.8%
27	8	15	10	74	9.6			22.9%
28	8	15	10	76	9.5			20.8%
29	8	15	10	76	9.6			20.8%
30	8	15	10	78	9.6			18.8%

31	8	15	10	77	9.5			19.8%
32	8	15	10	73	9.5			24.0%
33	15	6	20	65	13.4	98	2	33.7%
34	15	6	20	70	13.8			28.6%
35	15	6	20	69	13.8			29.6%
36	15	6	20	68	13.7			30.6%
37	15	6	20	67	13.9			31.6%
38	15	6	20	68	13.8			30.6%
39	15	6	20	66	15.9			32.7%
40	15	6	20	68	13.9			30.6%
41	15	6	20	65	13.8			33.7%
42	15	6	20	67	15.8			31.6%
43	15	6	20	66	14			32.7%
44	15	6	20	66	13.8			32.7%
45	15	6	20	70	14.1			28.6%
46	15	6	20	65	13.8			33.7%
47	15	6	20	67	13.8			31.6%
48	15	6	20	67	13.9			31.6%
49	15	6	20	66	13.6			32.7%
50	15	6	20	66	17.3			32.7%
51	15	6	20	67	13.8			31.6%
52	15	6	20	67	14.3			31.6%
53	15	6	20	68	14			30.6%
54	15	6	20	63	14.1			35.7%
55	15	6	20	67	13.8			31.6%
56	15	15	20	144		187	81	23.0%
57	15	15	20	135	13.8			27.8%
58	15	15	20	138	13.7			26.2%
59	15	15	20	137	13.6			26.7%
60	15	15	20	142	13.6			24.1%
61	15	15	20	139	14.2			25.7%
62	15	15	20	141	13.6			24.6%
63	15	15	20	137	13.8			26.7%
64	15	15	20	141	13.7			24.6%
65	15	15	20	142	13.7			24.1%
66	15	15	20	139	14.6			25.7%
67	15	15	20	135	13.9			27.8%
68	15	15	20	141	13.9			24.6%
69	15	15	20	142	13.9			24.1%
70	15	15	20	139	13.8			25.7%
71	15	15	20	139	13.8			25.7%

72	15	15	20	135				27.8%

APPENDIX D–LINGO CODE

Below is the LINGO code used to calculate the optimum solutions in the Results section. All three objective functions are included in the code. Two of the three are commented out so that only one will run at a time. Simply changing which objective functions are commented will execute the correct objective function.

```
Model:
! Crane skills training;

Data:
     numworkers = @OLE('input 15-15-20.xls');

     numskills = @OLE('input 15-15-20.xls');

     numtasks = @OLE('input 15-15-20.xls');

enddata

Sets:
     levels/1..5/;
     workers/1..numworkers/;
     skills/1..numskills/;
     tasks/1..numtasks/;
     incremental(skills,levels):inctime,inccost;
     workerskill(workers,skills):wpref,wskill,notrai
n;
     taskskill(tasks,skills):tskill;
     workertask(workers,tasks):assigned;
     tasktime(tasks):ttime;
     workercap(workers):wcap;
     levelpairs(levels,levels);
     training(skills,levels,levels):tcost,trtime;
     moretraining(workers,skills,levels,levels):more
train;
     allcombos(workers,tasks,skills);
Endsets
```

```
Data:
wskill=  @OLE('input 15-15-20.xls');

tskill = @OLE('input 15-15-20.xls');

ttime = @OLE('input 15-15-20.xls');

wcap = @OLE('input 15-15-20.xls');

tcost = @OLE('input 15-15-20.xls');

trtime = @OLE('input 15-15-20.xls');

wpref = @OLE('input 15-15-20.xls');

Enddata


!Minimize Total Cost;
     !Min =
@SUM(moretraining(i,k,l,m)|l#EQ#wskill(i,k)  #AND#
m#GT#wskill(i,k): tcost(k,l,m)*moretrain(i,k,l,m));

!Maximize Skills Gained;
     !Max = @sum(workers(i):
               @sum(tasks(j):

     @sum(skills(k)|tskill(j,k)#GT#wskill(i,k):
(tskill(j,k)-wskill(i,k))*assigned(i,j))));

!Maximize Worker Preference;
     Max = @sum(workers(i):
               @sum(tasks(j):

     @sum(skills(k)|tskill(j,k)#GT#wskill(i,k):
wpref(i,k)*(tskill(j,k)-
wskill(i,k))*assigned(i,j))));
```

```
     @FOR(workers(i):
          @For(tasks(j):
               @For(skills(k)|tskill(j,k)#GT#0:
wskill(i,k)*notrain(i,k) +

     @Sum(levels(m)|m#GT#wskill(i,k):m*moretrain(i,k
,wskill(i,k),m)) >= tskill(j,k)*assigned(i,j))));

     @FOR(workers(i):
          @FOR(skills(k): notrain(i,k)+

     @Sum(levels(m)|m#GT#wskill(i,k):moretrain(i,k,w
skill(i,k),m)) = 1));

     @FOR(tasks(j):
          @Sum(workers(i):assigned(i,j)) = 1);

     @FOR(workers(i):
          @Sum(tasks(j):assigned(i,j)) >= 1);

     @FOR(workers(i):
          @Sum(tasks(j):ttime(j)*assigned(i,j))+

     @Sum(skills(k):@sum(levels(m):trtime(k,wskill(i
,k),m)*moretrain(i,k,wskill(i,k),m)))<=wcap(i));


     @FOR(workers(i):
          @FOR(tasks(j): @BIN(assigned(i,j))));

     @FOR(workers(i):
          @FOR(skills(k): @BIN(notrain(i,k))));

     @For(moretraining(i,k,l,m)|l#EQ#wskill(i,k):
@BIN(moretrain(i,k,l,m)));


End
```

APPENDIX E—

Code for Input Sheet

```
'This command button will setup the input sheet for the user
Public Sub InputButton_Click()

Dim numworkers As Single
Dim numskills As Single
Dim numtasks As Single
Dim numSolns As Single
Dim MinCostSwitch As Single
Dim MinSkillGapSwitch As Single
Dim MinPreferenceSwitch As Single
Dim MinOverTimeSwitch As Single

Dim TotalNumObjFn As Single

'initial values
TotalNumObjFn = 3
MinOverTimeSwitch = 0


'Clears all the sheets (except parameters)
Sheets("Input").Cells.Clear
Sheets("Results Summary").Cells.Clear


'Inputs setup values (number of workers/skills/tasks and which Objective Functions will be considered
```

```
Sheets("Input").Select
numworkers = Application.InputBox("Input number of workers", "")
numskills = Application.InputBox("Input number of skills", "")
numtasks = Application.InputBox("Input number of tasks", "")
numSolns = Application.InputBox("Input number of initial solutions to store", "")

'Insert values
ActiveSheet.Cells(1, 1).Value = "PROBLEM SIZE"
ActiveSheet.Cells(2, 1).Value = "Number of workers"
ActiveSheet.Cells(3, 1).Value = "Number of skills"
ActiveSheet.Cells(4, 1).Value = "Number of tasks"
ActiveSheet.Cells(5, 1).Value = "Numer of initial Solutions"
ActiveSheet.Cells(6, 1).Value = "OBJECTIVE FUNCTIONS"
ActiveSheet.Cells(7, 1).Value = "Min Training Cost?"
ActiveSheet.Cells(8, 1).Value = "Max Amount of Training?"
ActiveSheet.Cells(9, 1).Value = "Max Worker Preference?"
'ActiveSheet.Cells(10, 1).Value = "Min OverTime?"
ActiveSheet.Cells(2, 2).Value = numworkers
ActiveSheet.Cells(3, 2).Value = numskills
ActiveSheet.Cells(4, 2).Value = numtasks
ActiveSheet.Cells(5, 2).Value = numSolns
ActiveSheet.Cells(7, 2).Value = MinCostSwitch
ActiveSheet.Cells(8, 2).Value = MinSkillGapSwitch
ActiveSheet.Cells(9, 2).Value = MinPreferenceSwitch
'ActiveSheet.Cells(10, 2).Value = MinOverTimeSwitch

'Add some colors to these ranges
ActiveSheet.Range("A1", Cells(TotalNumObjFn + 6, 1)).Select
   With Selection.Interior
      .ColorIndex = 39
      .Pattern = xlSolid
```

```
    End With


'Create Worker Skill Matrix
For i = 1 To numworkers
    For k = 1 To numskills
        ActiveSheet.Cells(TotalNumObjFn + 8, 1).Value = "Worker Skill Matrix"
        ActiveSheet.Cells(i + TotalNumObjFn + 9, 1).Value = "Worker " & i
        If k < 6 Then ActiveSheet.Cells(TotalNumObjFn + 9, k + 1).Value = "Skill " & k
        If k >= 6 Then ActiveSheet.Cells(TotalNumObjFn + 9, k + 1).Value = k
    Next k
Next i

    'title coloring
    ActiveSheet.Cells(TotalNumObjFn + 8, 1).Select
    With Selection.Interior
        .ColorIndex = 36
        .Pattern = xlSolid
    End With
    'skill coloring
    ActiveSheet.Range(Cells(TotalNumObjFn + 9, 2), Cells(TotalNumObjFn + 9, numskills + 1)).Select
    With Selection.Interior
        .ColorIndex = 40
        .Pattern = xlSolid
    End With
    'worker coloring
    ActiveSheet.Range(Cells(TotalNumObjFn + 10, 1), Cells(TotalNumObjFn + numworkers + 9, 1)).Select
    With Selection.Interior
        .ColorIndex = 34
        .Pattern = xlSolid
    End With
```

```
'Create Task Skill Matrix
For j = 1 To numtasks
   For k = 1 To numskills
      ActiveSheet.Cells(11 + TotalNumObjFn + numworkers, 1).Value = "Task Skill Matrix"
      ActiveSheet.Cells(12 + TotalNumObjFn + numworkers + j, 1).Value = "Task " & j
      If k < 6 Then ActiveSheet.Cells(12 + TotalNumObjFn + numworkers, k + 1).Value = "Skill " & k
      If k >= 6 Then ActiveSheet.Cells(12 + TotalNumObjFn + numworkers, k + 1).Value = k
   Next k
Next j

   ActiveSheet.Range(Cells(11 + TotalNumObjFn + numworkers, 1), Cells(11 + TotalNumObjFn + numworkers, 1)).Select
   With Selection.Interior
      .ColorIndex = 36
      .Pattern = xlSolid
   End With
   ActiveSheet.Range(Cells(12 + TotalNumObjFn + numworkers, 2), Cells(12 + TotalNumObjFn + numworkers, numskills +
1)).Select
   With Selection.Interior
      .ColorIndex = 40
      .Pattern = xlSolid
   End With
   ActiveSheet.Range(Cells(13 + TotalNumObjFn + numworkers, 1), Cells(12 + TotalNumObjFn + numworkers + numtasks,
1)).Select
   With Selection.Interior
      .ColorIndex = 38
      .Pattern = xlSolid
   End With
```

```
'Create Task Time Matrix
For j = 1 To numtasks
   ActiveSheet.Cells(14 + TotalNumObjFn + numtasks + numworkers, 2).Value = "Task Time"
   ActiveSheet.Cells(14 + TotalNumObjFn + numtasks + numworkers + j, 1).Value = "Task " & j
Next j

   ActiveSheet.Range(Cells(14 + TotalNumObjFn + numworkers + numtasks, 2), Cells(14 + TotalNumObjFn + numworkers + numtasks, 2)).Select
   With Selection.Interior
      .ColorIndex = 36
      .Pattern = xlSolid
   End With
   ActiveSheet.Range(Cells(15 + TotalNumObjFn + numworkers + numtasks, 1), Cells(14 + TotalNumObjFn + numworkers + 2 * numtasks, 1)).Select
   With Selection.Interior
      .ColorIndex = 38
      .Pattern = xlSolid
   End With


'Create Worker Capacity Matrix
For i = 1 To numworkers
   ActiveSheet.Cells(16 + TotalNumObjFn + 2 * numtasks + numworkers, 2).Value = "Worker Capacity"
   ActiveSheet.Cells(16 + TotalNumObjFn + 2 * numtasks + numworkers + i, 1).Value = "Worker " & i
Next i

   ActiveSheet.Range(Cells(16 + TotalNumObjFn + numworkers + 2 * numtasks, 2), Cells(16 + TotalNumObjFn + numworkers + 2 * numtasks, 3)).Select
   With Selection.Interior
      .ColorIndex = 36
      .Pattern = xlSolid
```

```
   End With
   ActiveSheet.Range(Cells(17 + TotalNumObjFn + numworkers + 2 * numtasks, 1), Cells(16 + TotalNumObjFn + 2 * numworkers +
2 * numtasks, 1)).Select
   With Selection.Interior
      .ColorIndex = 34
      .Pattern = xlSolid
   End With


'Create Training Cost Matrix
For i = 1 To numskills
   For j = 1 To 4
      ActiveSheet.Cells(18 + TotalNumObjFn + 2 * numtasks + 2 * numworkers, 1).Value = "Cost to Train Matrix"
      ActiveSheet.Cells(19 + TotalNumObjFn + 2 * numtasks + 2 * numworkers + i, 1).Value = "Skill " & i
      ActiveSheet.Cells(19 + TotalNumObjFn + 2 * numtasks + 2 * numworkers, 1 + j).Value = "Level " & j & " to " & j + 1
   Next j
Next i

   ActiveSheet.Range(Cells(18 + TotalNumObjFn + 2 * numworkers + 2 * numtasks, 1), Cells(18 + TotalNumObjFn + 2 *
numworkers + 2 * numtasks, 1)).Select
   With Selection.Interior
      .ColorIndex = 36
      .Pattern = xlSolid
   End With
   ActiveSheet.Range(Cells(20 + TotalNumObjFn + 2 * numworkers + 2 * numtasks, 1), Cells(19 + TotalNumObjFn + 2 *
numworkers + 2 * numtasks + numskills, 1)).Select
   With Selection.Interior
      .ColorIndex = 40
      .Pattern = xlSolid
   End With
```

```
   ActiveSheet.Range(Cells(19 + TotalNumObjFn + 2 * numworkers + 2 * numtasks, 2), Cells(19 + TotalNumObjFn + 2 *
numworkers + 2 * numtasks, 5)).Select
   With Selection.Interior
      .ColorIndex = 35
      .Pattern = xlSolid
   End With


'Create Training Time Matrix
For i = 1 To numskills
   For j = 1 To 4
      ActiveSheet.Cells(21 + TotalNumObjFn + 2 * numtasks + 2 * numworkers + numskills, 1).Value = "Time to Train Matrix"
      ActiveSheet.Cells(22 + TotalNumObjFn + 2 * numtasks + 2 * numworkers + numskills + i, 1).Value = "Skill " & i
      ActiveSheet.Cells(22 + TotalNumObjFn + 2 * numtasks + 2 * numworkers + numskills, 1 + j).Value = "Level " & j & " to " & j
+ 1
   Next j
Next i

   ActiveSheet.Range(Cells(21 + TotalNumObjFn + 2 * numworkers + 2 * numtasks + numskills, 1), Cells(21 + TotalNumObjFn + 2
* numworkers + 2 * numtasks + numskills, 1)).Select
   With Selection.Interior
      .ColorIndex = 36
      .Pattern = xlSolid
   End With
   ActiveSheet.Range(Cells(23 + TotalNumObjFn + 2 * numworkers + 2 * numtasks + numskills, 1), Cells(22 + TotalNumObjFn + 2
* numworkers + 2 * numtasks + 2 * numskills, 1)).Select
   With Selection.Interior
      .ColorIndex = 40
      .Pattern = xlSolid
   End With
```

```
   ActiveSheet.Range(Cells(22 + TotalNumObjFn + 2 * numworkers + 2 * numtasks + numskills, 2), Cells(22 + TotalNumObjFn + 2
* numworkers + 2 * numtasks + numskills, 5)).Select
   With Selection.Interior
      .ColorIndex = 35
      .Pattern = xlSolid
   End With


'Create Worker Skill Preference Matrix
For i = 1 To numworkers
   For j = 1 To numskills
      ActiveSheet.Cells(24 + TotalNumObjFn + 2 * numtasks + 2 * numworkers + 2 * numskills, 1).Value = "Worker Preference
Matrix"
      ActiveSheet.Cells(25 + TotalNumObjFn + 2 * numtasks + 2 * numworkers + 2 * numskills + i, 1).Value = "Worker " & i
      If j < 6 Then ActiveSheet.Cells(25 + TotalNumObjFn + 2 * numtasks + 2 * numworkers + 2 * numskills, j + 1).Value = "Skill "
& j
      If j >= 6 Then ActiveSheet.Cells(25 + TotalNumObjFn + 2 * numtasks + 2 * numworkers + 2 * numskills, j + 1).Value = j
   Next j
Next i

   ActiveSheet.Range(Cells(24 + TotalNumObjFn + 2 * numworkers + 2 * numtasks + 2 * numskills, 1), Cells(24 + TotalNumObjFn
+ 2 * numworkers + 2 * numtasks + 2 * numskills, 1)).Select
   With Selection.Interior
      .ColorIndex = 36
      .Pattern = xlSolid
   End With
   ActiveSheet.Range(Cells(26 + TotalNumObjFn + 2 * numworkers + 2 * numtasks + 2 * numskills, 1), Cells(26 + TotalNumObjFn
+ 3 * numworkers + 2 * numtasks + 2 * numskills - 1, 1)).Select
   With Selection.Interior
      .ColorIndex = 34
      .Pattern = xlSolid
```

```
    End With
    ActiveSheet.Range(Cells(25 + TotalNumObjFn + 2 * numworkers + 2 * numtasks + 2 * numskills, 2), Cells(25 + TotalNumObjFn
+ 2 * numworkers + 2 * numtasks + 2 * numskills, numskills + 1)).Select
    With Selection.Interior
        .ColorIndex = 40
        .Pattern = xlSolid
    End With
End Sub


"There will be a sub similar to this one for each of the objectives.
"A separate procedure will simply call each of them needed


'This command button will find the initial solution for Minimum Training Cost
Public Sub InitalTrainingCostSoln_click()
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 workertaskSgap() As Single, oworkertaskSgap() As Single
Dim workertaskPref() As Single, oworkertaskPref() As Single
Dim Prefmatrix() 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, numSolns As Single
Dim totaltaskcost() As Single
Dim totalworkercost() As Single
Dim totaltaskSgap() As Single
Dim totalworkerSgap() As Single
Dim totaltaskPref() As Single
Dim totalworkerPref() As Single
Dim workerphase1() As Single
Dim cellrow As Single
Dim trainingneeds() As Single
Dim skilltrainingneeds() As Single
'everything below is new:
Dim bestCost() As Single
Dim bestSgap() As Single
Dim bestPref() As Single

Dim TotalNumObjFn As Single
TotalNumObjFn = 3


Sheets("Results Summary").Cells.Clear


Sheets("Parameters").Select
p1perprior = ActiveSheet.Cells(1, 2).Value
p1perrestrict = ActiveSheet.Cells(2, 2).Value
```

```
perprior = ActiveSheet.Cells(3, 2).Value
perrestrict = ActiveSheet.Cells(4, 2).Value
numiter = ActiveSheet.Cells(5, 2).Value

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


Sheets("Input").Select
numworkers = ActiveSheet.Cells(2, 2).Value
numskills = ActiveSheet.Cells(3, 2).Value
numtasks = ActiveSheet.Cells(4, 2).Value
numSolns = ActiveSheet.Cells(5, 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
'added a dimension to the two matrices below
ReDim workerassign(0 To 1, 0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim bestworkerassign(0 To numSolns + 1, 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 workertaskSgap(0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim oworkertaskSgap(0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim workertaskPref(0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim oworkertaskPref(0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim Prefmatrix(0 To numworkers + 1, 0 To numskills + 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 totaltaskSgap(0 To numtasks + 1) As Single
ReDim totalworkerSgap(0 To numworkers + 1) As Single
ReDim totaltaskPref(0 To numtasks + 1) As Single
ReDim totalworkerPref(0 To numworkers + 1) As Single
ReDim workerphase1(0 To numworkers + 1) As Single
ReDim trainingneeds(0 To numworkers + 1, 0 To numskills + 1) As Single
ReDim skilltrainingneeds(0 To numskills + 1, 0 To 5) As Single
'new one below:
ReDim bestCost(0 To numSolns + 20) As Single
ReDim bestSgap(0 To numSolns + 20) As Single
ReDim bestPref(0 To numSolns + 20) As Single

  starttime = Timer

For a = 0 To numSolns + 1
   bestCost(a) = 0
   bestSgap(a) = 0
   bestPref(a) = 0
```

```
    For b = 0 To numworkers + 1
        For c = 0 To numtasks + 1
            bestworkerassign(a, b, c) = 0
        Next c
    Next b
Next a


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
        trainingneeds(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
    totaltaskSgap(b) = 0
```

```
    totaltaskPref(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
        skilltrainingneeds(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
    totalworkerSgap(b) = 0
    totalworkerPref(b) = 0
    For k = 1 To numtasks
        workerassign(1, b, k) = 0
        workertaskcost(b, k) = 0
        oworkertaskcost(b, k) = 0
        workertaskSgap(b, k) = 0
        oworkertaskSgap(b, k) = 0
```

```
      workertaskPref(b, k) = 0
      oworkertaskPref(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(9 + TotalNumObjFn + b, 1 + k)
   Next k
Next b

For b = 1 To numtasks
   For k = 1 To numskills
      otaskskill(b, k) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + b, 1 + k)
   Next k
Next b

For b = 1 To numtasks
   otasktime(b) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + numtasks + 2 + b, 2)
Next b

For b = 1 To numworkers
   oworkercapacity(b) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + numtasks + 2 + numtasks + 2 + b, 2)
Next b

For i = 1 To numskills
   For j = 2 To 5
      tcost(i, j) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + numtasks + 2 + numtasks + 2 + numworkers + 3 + i, j)
   Next j
```

```
Next i

For i = 1 To numskills
   For j = 2 To 5
      ttime(i, j) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + numtasks + 2 + numtasks + 2 + numworkers + 3 +
numskills + 3 + i, j)
   Next j
Next i

For b = 1 To numworkers
   For k = 1 To numskills
      Prefmatrix(b, k) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + numtasks + 2 + numtasks + 2 + numworkers + 3
+ numskills + 3 + numskills + 3 + b, k + 1)
   Next k
Next b


'done reading in values from the excel sheet

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))
            oworkertaskSgap(i, j) = oworkertaskSgap(i, j) + (otaskskill(j, k) - oworkerskill(i, k))
            oworkertaskPref(i, j) = oworkertaskPref(i, j) + Prefmatrix(i, k) * (otaskskill(j, k) - oworkerskill(i, 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)
      totaltaskSgap(j) = totaltaskSgap(j) + oworkertaskSgap(i, j)
      totaltaskPref(j) = totaltaskPref(j) + oworkertaskPref(i, j)
   Next i
Next j
```

```
For i = 1 To numworkers
   For j = 1 To numtasks
      totalworkercost(i) = totalworkercost(i) + oworkertaskcost(i, j)
      totalworkerSgap(i) = totalworkerSgap(i) + oworkertaskSgap(i, j)
      totalworkerPref(i) = totalworkerPref(i) + oworkertaskPref(i, j)
   Next j
Next i


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
warned1 = 0
warned2 = 0

For a = 1 To numSolns + 1
   bestCost(a) = 9999999
   bestSgap(a) = -9999999
   bestPref(a) = -9999999
Next a
bestCost(0) = -9999999
bestSgap(0) = 9999999
bestPref(0) = 9999999

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(1, b, k) = 0
      workertaskcost(b, k) = oworkertaskcost(b, k)
      workertaskSgap(b, k) = oworkertaskSgap(b, k)
      workertaskPref(b, k) = oworkertaskPref(b, k)
      workertasktime(b, k) = oworkertasktime(b, k)
   Next k
Next b

For b = 1 To numworkers
```

```
    workerphase1(b) = 0
  Next b


  totalcost = 0
  totalsgap = 0
  totalpref = 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 = -55
      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
      mincosttask = 0
      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


If mincosttask = 0 Then
   If warned1 = 0 Then
      response = MsgBox("no feasible solution - not enough worker capacity", vbOKOnly, "Capacity Error")
      warned1 = 1
   End If
   'If response = vbOK Then
   '    Stop
   'End If
   totalcost = 99999999
   numtaskassigned = numtasks + 1
End If


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

   'make the best assignment
   If priorrnd <= p1perprior Then
      'assign maxcostworker to mincost task
      totalcost = totalcost + workertaskcost(maxcostworker, mincosttask)
      totalsgap = totalsgap + workertaskSgap(maxcostworker, mincosttask)
      totalpref = totalpref + workertaskPref(maxcostworker, mincosttask)
      numtaskassigned = numtaskassigned + 1
      workerassign(1, maxcostworker, mincosttask) = 1
      taskassigned(mincosttask) = 1
      workerphase1(maxcostworker) = 1
```

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

          'make a near-optimal assignment
          If priorrnd > p1perprior Then
            'form available list and choose assigned worker from available list
             numonlist = 0
             For i = 1 To numworkers
               If workerphase1(i) = 0 And totalworkercost(i) >= maxcost * (1 - (p1perrestrict / 100)) Then
                  For j = 1 To numtasks
                    If taskassigned(j) = 0 And workertaskcost(i, j) <= mincost * (1 + (p1perrestrict / 100)) And workertasktime(i, j)
<= workercapacity(i) Then
                      numonlist = numonlist + 1
                      available(numonlist, 1) = i
                      available(numonlist, 2) = j
                    End If
                 Next j
               End If
            Next i

            'determine random assignment from the candidate list created above
            Randomize
            restrictrnd = Round(((numonlist - 1) * Rnd) + 1)
            assignedworker = available(restrictrnd, 1)
            assignedtask = available(restrictrnd, 2)

            'increase each objective function
            totalcost = totalcost + workertaskcost(assignedworker, assignedtask)
            totalsgap = totalsgap + workertaskSgap(assignedworker, assignedtask)
```

```
            totalpref = totalpref + workertaskPref(assignedworker, assignedtask)
            'adjust counter variables
            numtaskassigned = numtaskassigned + 1
            workerassign(1, assignedworker, assignedtask) = 1
            taskassigned(assignedtask) = 1
            workerphase1(assignedworker) = 1
            'reduce assigned workers capacity
            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
                'zero out all the objective function values and recalculate them with new skill levels determined above
                workertaskcost(assignedworker, j) = 0
                workertaskSgap(assignedworker, j) = 0
                workertaskPref(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))
```

```
                workertaskSgap(assignedworker, j) = workertaskSgap(assignedworker, j) + (taskskill(j, k) - workerskill(assignedworker, k))
                workertaskPref(assignedworker, j) = workertaskPref(assignedworker, j) + Prefmatrix(assignedworker, k) * (taskskill(j, k) - workerskill(assignedworker, 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
  'very similar to phase 1, just making the next assignment a slightly different way
  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; update objective function values
      totalcost = totalcost + workertaskcost(mincostworker, maxcosttask)
      totalsgap = totalsgap + workertaskSgap(mincostworker, maxcosttask)
      totalpref = totalpref + workertaskPref(mincostworker, maxcosttask)
      'update counter variables
      numtaskassigned = numtaskassigned + 1
      workerassign(1, 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)
   totalsgap = totalsgap + workertaskSgap(assignedworker, assignedtask)
   totalpref = totalpref + workertaskPref(assignedworker, assignedtask)
   numtaskassigned = numtaskassigned + 1
   workerassign(1, 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
               workertaskSgap(assignedworker, j) = 0
               workertaskPref(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))
                     workertaskSgap(assignedworker, j) = workertaskSgap(assignedworker, j) + (taskskill(j, k) - workerskill(assignedworker, k))
                     workertaskPref(assignedworker, j) = workertaskPref(assignedworker, j) + Prefmatrix(assignedworker, k) * (taskskill(j, k) - workerskill(assignedworker, 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
```

```
            If warned2 = 0 Then
               response = MsgBox("no feasible solution - not enough worker capacity", vbOKOnly, "Capacity Error")
               warned2 = 1
            End If
            'If response = vbOK Then
            '    Stop
            'End If
            totalcost = 99999999
            numtaskassigned = numtasks + 1
         End If

   Loop

'eliminate duplicate solutions by setting totalcost to M
For a = 1 To numSolns
   sameassign = 0
   If totalcost = bestCost(a) Then
      For b = 1 To numworkers
         For c = 1 To numtasks
            If workerassign(1, b, c) = bestworkerassign(a, b, c) Then
               sameassign = sameassign + 1
            'try adding in: Else, exit the entire loop.
            End If
         Next c
      Next b
   End If
   If sameassign = numworkers * numtasks Then
      totalcost = 999999
   End If
Next a
```

```
'check to see if totalcost is one of the top solutions.  if so, everything behind it gets bumped
'and totalcost replaces the top solution that it beats out (a determines this)
If totalcost <= bestCost(numSolns) Then
   For a = numSolns To 1 Step -1
      'determines where in the list the new solution goes
      If totalcost <= bestCost(a) And totalcost > bestCost(a - 1) Then
         b = numSolns
         c = a
         'move the lower solutions back one rank
         Do While c < numSolns
            bestCost(b) = bestCost(b - 1)
            bestSgap(b) = bestSgap(b - 1)
            bestPref(b) = bestPref(b - 1)
            For i = 1 To numworkers
               For j = 1 To numtasks
                  bestworkerassign(b, i, j) = bestworkerassign(b - 1, i, j)
               Next j
            Next i
            c = c + 1
            b = b - 1
         Loop
         'set the new solution into its rank
         bestCost(a) = totalcost
         bestSgap(a) = totalsgap
         bestPref(a) = totalpref
         For i = 1 To numworkers
            For j = 1 To numtasks
               bestworkerassign(a, i, j) = workerassign(1, i, j)
            Next j
         Next i
      End If
```

```
    Next a
End If


    'Print results
    Sheets("Results Summary").Select
    ActiveSheet.Cells(1, 1) = "Best Solution Costs"
    ActiveSheet.Cells(2, 1) = "Training Amount"
    ActiveSheet.Cells(3, 1) = "Worker Preference"
    For a = 1 To numSolns
        ActiveSheet.Cells(1, a + 1) = bestCost(a)
    Next a
    For a = 1 To numSolns
        ActiveSheet.Cells(2, a + 1) = bestSgap(a)
    Next a
    For a = 1 To numSolns
        ActiveSheet.Cells(3, a + 1) = bestPref(a)
    Next a

Next r

endtime = Timer
totaltime = endtime - starttime
Sheets("Results Summary").Select
ActiveSheet.Cells(1, numSolns + 3) = "Run Time"
ActiveSheet.Cells(1, numSolns + 4) = totaltime

'Print to Results Summary sheet
Sheets("Results Summary").Select
    'ActiveSheet.Cells(1, 1) = "Best Solution Costs"
    'ActiveSheet.Cells(1, 2) = bestsolution(1)
```

```
ActiveSheet.Cells(6, 1) = "Solution #:"
For a = 1 To numSolns
   ActiveSheet.Cells(6, 2 * a) = a
Next a

ActiveSheet.Cells(5, 1) = "Worker to Task Assignments"
ActiveSheet.Cells(8 + numtasks, 1) = "Total Cost"
ActiveSheet.Cells(9 + numtasks, 1) = "Total Skill Levels"
ActiveSheet.Cells(10 + numtasks, 1) = "Total Preference"
b = 1
For a = 1 To numSolns * 2
   ActiveSheet.Cells(7, a) = "Worker"
   ActiveSheet.Cells(7, a + 1) = "Task"
   cellrow = 8
   For i = 1 To numworkers
      For j = 1 To numtasks
         If bestworkerassign(b, i, j) = 1 Then
            ActiveSheet.Cells(cellrow, a) = i
            ActiveSheet.Cells(cellrow, a + 1) = j
            cellrow = cellrow + 1
         End If
      Next j
   Next i
   ActiveSheet.Cells(8 + numtasks, a + 1) = bestCost(b)
   ActiveSheet.Cells(9 + numtasks, a + 1) = bestSgap(b)
   ActiveSheet.Cells(10 + numtasks, a + 1) = bestPref(b)
   b = b + 1
   a = a + 1
Next a

ActiveSheet.Range("A1").Select
```

```
With Selection.Interior
    .ColorIndex = 35
    .Pattern = xlSolid
End With
ActiveSheet.Range("A4:B4").Select
With Selection.Interior
    .ColorIndex = 36
    .Pattern = xlSolid
End With
```

```
Public Sub InitialSkillGapSoln_click()

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 workertaskSgap() As Single, oworkertaskSgap() As Single
Dim workertaskPref() As Single, oworkertaskPref() As Single
Dim Prefmatrix() 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, numSolns As Single
Dim totaltaskcost() As Single
Dim totalworkercost() As Single
Dim totaltaskSgap() As Single
Dim totalworkerSgap() As Single
Dim totaltaskPref() As Single
Dim totalworkerPref() As Single
Dim workerphase1() As Single
Dim cellrow As Single
Dim trainingneeds() As Single
Dim skilltrainingneeds() As Single
Dim bestCost() As Single
```

```
Dim bestSgap() As Single
Dim bestPref() As Single

Dim TotalNumObjFn As Single
TotalNumObjFn = 3


Sheets("Parameters").Select
p1perprior = ActiveSheet.Cells(1, 2).Value
p1perrestrict = ActiveSheet.Cells(2, 2).Value

perprior = ActiveSheet.Cells(3, 2).Value
perrestrict = ActiveSheet.Cells(4, 2).Value
numiter = ActiveSheet.Cells(5, 2).Value

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

Sheets("Input").Select
numworkers = ActiveSheet.Cells(2, 2).Value
numskills = ActiveSheet.Cells(3, 2).Value
numtasks = ActiveSheet.Cells(4, 2).Value
numSolns = ActiveSheet.Cells(5, 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 1, 0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim bestworkerassign(0 To numSolns + 1, 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 workertaskSgap(0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim oworkertaskSgap(0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim workertaskPref(0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim oworkertaskPref(0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim Prefmatrix(0 To numworkers + 1, 0 To numskills + 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 totaltaskSgap(0 To numtasks + 1) As Single
ReDim totalworkerSgap(0 To numworkers + 1) As Single
ReDim totaltaskPref(0 To numtasks + 1) As Single
ReDim totalworkerPref(0 To numworkers + 1) As Single
ReDim workerphase1(0 To numworkers + 1) As Single
ReDim trainingneeds(0 To numworkers + 1, 0 To numskills + 1) As Single
ReDim skilltrainingneeds(0 To numskills + 1, 0 To 5) As Single
ReDim bestCost(0 To numSolns + 20) As Single
```

```
ReDim bestSgap(0 To numSolns + 20) As Single
ReDim bestPref(0 To numSolns + 20) As Single

  starttime = Timer


For a = 0 To numSolns + 1
   bestCost(a) = 0
   bestSgap(a) = 0
   bestPref(a) = 0
   For b = 0 To numworkers + 1
      For c = 0 To numtasks + 1
         bestworkerassign(a, b, c) = 0
      Next c
   Next b
Next a


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
      trainingneeds(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
    totaltaskSgap(b) = 0
    totaltaskPref(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
        skilltrainingneeds(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
   totalworkerSgap(b) = 0
   totalworkerPref(b) = 0
   For k = 1 To numtasks
      workerassign(1, b, k) = 0
      workertaskcost(b, k) = 0
      oworkertaskcost(b, k) = 0
      workertaskSgap(b, k) = 0
      oworkertaskSgap(b, k) = 0
      workertaskPref(b, k) = 0
      oworkertaskPref(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(9 + TotalNumObjFn + b, 1 + k)
   Next k
Next b

For b = 1 To numtasks
   For k = 1 To numskills
      otaskskill(b, k) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + b, 1 + k)
   Next k
Next b

For b = 1 To numtasks
   otasktime(b) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + numtasks + 2 + b, 2)
Next b
```

```
For b = 1 To numworkers
   oworkercapacity(b) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + numtasks + 2 + numtasks + 2 + b, 2)
Next b

For i = 1 To numskills
   For j = 2 To 5
      tcost(i, j) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + numtasks + 2 + numtasks + 2 + numworkers + 3 + i, j)
   Next j
Next i

For i = 1 To numskills
   For j = 2 To 5
      ttime(i, j) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + numtasks + 2 + numtasks + 2 + numworkers + 3 +
numskills + 3 + i, j)
   Next j
Next i

For b = 1 To numworkers
   For k = 1 To numskills
      Prefmatrix(b, k) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + numtasks + 2 + numtasks + 2 + numworkers + 3
+ numskills + 3 + numskills + 3 + b, k + 1)
   Next k
Next b


'done reading in values from the excel sheet

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))
            oworkertaskSgap(i, j) = oworkertaskSgap(i, j) + (otaskskill(j, k) - oworkerskill(i, k))
            oworkertaskPref(i, j) = oworkertaskPref(i, j) + Prefmatrix(i, k) * (otaskskill(j, k) - oworkerskill(i, 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)
      totaltaskSgap(j) = totaltaskSgap(j) + oworkertaskSgap(i, j)
      totaltaskPref(j) = totaltaskPref(j) + oworkertaskPref(i, j)
   Next i
Next j


For i = 1 To numworkers
   For j = 1 To numtasks
      totalworkercost(i) = totalworkercost(i) + oworkertaskcost(i, j)
      totalworkerSgap(i) = totalworkerSgap(i) + oworkertaskSgap(i, j)
      totalworkerPref(i) = totalworkerPref(i) + oworkertaskPref(i, j)
   Next j
Next i


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
warned3 = 0

For a = 1 To numSolns + 1
   bestCost(a) = 999999999
   bestSgap(a) = -9999999
   bestPref(a) = -9999999
Next a
bestCost(0) = -9999999
bestSgap(0) = 999999999
bestPref(0) = 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(1, b, k) = 0
```

```
        workertaskcost(b, k) = oworkertaskcost(b, k)
        workertaskSgap(b, k) = oworkertaskSgap(b, k)
        workertaskPref(b, k) = oworkertaskPref(b, k)
        workertasktime(b, k) = oworkertasktime(b, k)
      Next k
   Next b

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


   totalcost = 0
   totalsgap = 0
   totalpref = 0
   numtaskassigned = 0

  If phase1_on = 1 Then
      Do While numtaskassigned < numworkers
         maxsgapt1 = -9999999
         For i = 1 To numworkers
            If workerphase1(i) = 0 And totalworkerSgap(i) > maxsgapt1 Then
               maxsgapt1 = totalworkerSgap(i)
               maxsgapworker = i
            End If
         Next i

         maxsgapt2 = -9999999
         For j = 1 To numtasks
            If taskassigned(j) = 0 And workertasktime(maxsgapworker, j) <= workercapacity(maxsgapworker) And
workertaskSgap(maxsgapworker, j) > maxsgapt2 Then
```

```
            maxsgapt2 = workertaskSgap(maxsgapworker, j)
            maxsgaptask = j
          End If
        Next j


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

          'make the best assignment
          If priorrnd <= p1perprior Then
            totalcost = totalcost + workertaskcost(maxsgapworker, maxsgaptask)
            totalsgap = totalsgap + workertaskSgap(maxsgapworker, maxsgaptask)
            totalpref = totalpref + workertaskPref(maxsgapworker, maxsgaptask)
            numtaskassigned = numtaskassigned + 1
            workerassign(1, maxsgapworker, maxsgaptask) = 1
            taskassigned(maxsgaptask) = 1
            workerphase1(maxsgapworker) = 1
            workercapacity(maxsgapworker) = workercapacity(maxsgapworker) - workertasktime(maxsgapworker, maxsgaptask)
            assignedworker = maxsgapworker
            assignedtask = maxsgaptask
          End If

          If priorrnd > p1perprior Then
             numonlist = 0
             For i = 1 To numworkers
               If workerphase1(i) = 0 And totalworkerSgap(i) >= maxsgapt1 * (1 - (p1perrestrict / 100)) Then
                  For j = 1 To numtasks
                    If taskassigned(j) = 0 And workertaskSgap(i, j) >= maxsgapt2 * (1 - (p1perrestrict / 100)) And workertasktime(i, j)
<= workercapacity(i) Then
                       numonlist = numonlist + 1
```

```
                available(numonlist, 1) = i
                available(numonlist, 2) = j
              End If
            Next j
          End If
        Next i

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

        totalcost = totalcost + workertaskcost(assignedworker, assignedtask)
        totalsgap = totalsgap + workertaskSgap(assignedworker, assignedtask)
        totalpref = totalpref + workertaskPref(assignedworker, assignedtask)

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

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

    For k = 1 To numskills
        If workerskill(assignedworker, k) < taskskill(assignedtask, k) Then
            workerskill(assignedworker, k) = taskskill(assignedtask, k)
        End If
    Next k

    For j = 1 To numtasks
```

```
            If taskassigned(j) = 0 Then
                workertaskcost(assignedworker, j) = 0
                workertaskSgap(assignedworker, j) = 0
                workertaskPref(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))
                        workertaskSgap(assignedworker, j) = workertaskSgap(assignedworker, j) + (taskskill(j, k) - workerskill(assignedworker, k))
                        workertaskPref(assignedworker, j) = workertaskPref(assignedworker, j) + Prefmatrix(assignedworker, k) * (taskskill(j, k) - workerskill(assignedworker, 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

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

End While numtaskassigned < numtasks 'repeat until all tasks assigned
    maxsgapt1 = -9999999
```

```
For j = 1 To numtasks
   If taskassigned(j) = 0 And totaltaskSgap(j) > maxsgapt1 Then
      maxsgapt1 = totaltaskSgap(j)
      maxsgaptask = j
   End If
Next j

maxsgapt2 = -9999999
maxsgapworker = 0
For i = 1 To numworkers
   If workertasktime(i, maxsgaptask) <= workercapacity(i) And workertaskSgap(i, maxsgaptask) > maxsgapt2 Then
      maxsgapt2 = workertaskSgap(i, maxsgaptask)
      maxsgapworker = i
   End If
Next i


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

If maxsgapworker > 0 Then
   If priorrnd <= perprior Then
      totalcost = totalcost + workertaskcost(maxsgapworker, maxsgaptask)
      totalsgap = totalsgap + workertaskSgap(maxsgapworker, maxsgaptask)
      totalpref = totalpref + workertaskPref(maxsgapworker, maxsgaptask)

      numtaskassigned = numtaskassigned + 1
      workerassign(1, maxsgapworker, maxsgaptask) = 1
      taskassigned(maxsgaptask) = 1
      workercapacity(maxsgapworker) = workercapacity(maxsgapworker) - workertasktime(maxsgapworker, maxsgaptask)
      assignedworker = maxsgapworker
```

```
   assignedtask = maxsgaptask
End If

If priorrnd > perprior Then
    numonlist = 0
    For j = 1 To numtasks
      If totaltaskSgap(j) >= maxsgapt1 * (1 - (perrestrict / 100)) And taskassigned(j) = 0 Then
         For i = 1 To numworkers
           If workertaskSgap(i, j) >= maxsgapt2 * (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)
   totalsgap = totalsgap + workertaskSgap(assignedworker, assignedtask)
   totalpref = totalpref + workertaskPref(assignedworker, assignedtask)
   numtaskassigned = numtaskassigned + 1
   workerassign(1, assignedworker, assignedtask) = 1
   taskassigned(assignedtask) = 1
   workercapacity(assignedworker) = workercapacity(assignedworker) - workertasktime(assignedworker, assignedtask)
End If
```

```
        For k = 1 To numskills
           If workerskill(assignedworker, k) < taskskill(assignedtask, k) Then
              workerskill(assignedworker, k) = taskskill(assignedtask, k)
           End If
        Next k

        For j = 1 To numtasks
           If taskassigned(j) = 0 Then
              workertaskcost(assignedworker, j) = 0
              workertaskSgap(assignedworker, j) = 0
              workertaskPref(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))
                    workertaskSgap(assignedworker, j) = workertaskSgap(assignedworker, j) + (taskskill(j, k) - workerskill(assignedworker, k))
                    workertaskPref(assignedworker, j) = workertaskPref(assignedworker, j) + Prefmatrix(assignedworker, k) * (taskskill(j, k) - workerskill(assignedworker, 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 maxsgapworker = 0 Then
```

```
            If warned3 = 0 Then
               response = MsgBox("no feasible solution - not enough worker capacity", vbOKOnly, "Capacity Error")
               warned3 = 1
            End If
            'If response = vbOK Then
            '    Stop
            'End If
         totalsgap = -999997
         numtaskassigned = numtasks + 1
      End If

   Loop

For a = 1 To numSolns
   sameassign = 0
   If totalsgap = bestSgap(a) Then
      For b = 1 To numworkers
         For c = 1 To numtasks
            If workerassign(1, b, c) = bestworkerassign(a, b, c) Then
               sameassign = sameassign + 1
            End If
         Next c
      Next b
   End If
   If sameassign = numworkers * numtasks Then
      totalsgap = -9995
   End If
Next a

If totalsgap > bestSgap(numSolns) Then
   For a = numSolns To 1 Step -1
```

```
      If totalsgap >= bestSgap(a) And totalsgap < bestSgap(a - 1) Then
         b = numSolns
         c = a
         Do While c < numSolns
            bestCost(b) = bestCost(b - 1)
            bestSgap(b) = bestSgap(b - 1)
            bestPref(b) = bestPref(b - 1)
            For i = 1 To numworkers
               For j = 1 To numtasks
                  bestworkerassign(b, i, j) = bestworkerassign(b - 1, i, j)
               Next j
            Next i
            c = c + 1
            b = b - 1
         Loop

         bestCost(a) = totalcost
         bestSgap(a) = totalsgap
         bestPref(a) = totalpref
         For i = 1 To numworkers
            For j = 1 To numtasks
               bestworkerassign(a, i, j) = workerassign(1, i, j)
            Next j
         Next i
      End If
   Next a
End If
   'Print results
   Sheets("Results Summary").Select
   ActiveSheet.Cells(1, 1) = "Best Solution Costs"
   ActiveSheet.Cells(2, 1) = "Training Amount"
```

```
    ActiveSheet.Cells(3, 1) = "Worker Preference"
    For a = 1 To numSolns
        ActiveSheet.Cells(1, a + 1) = bestCost(a)
    Next a
    For a = 1 To numSolns
        ActiveSheet.Cells(2, a + 1) = bestSgap(a)
    Next a
    For a = 1 To numSolns
        ActiveSheet.Cells(3, a + 1) = bestPref(a)
    Next a

Next r


endtime = Timer
totaltime = endtime - starttime
Sheets("Results Summary").Select
ActiveSheet.Cells(2, numSolns + 3) = "Run Time"
ActiveSheet.Cells(2, numSolns + 4) = totaltime

'Print to Results Summary sheet
    Sheets("Results Summary").Select
    'ActiveSheet.Cells(1, 1) = "Best Solution Costs"
    'ActiveSheet.Cells(1, 2) = bestsolution(1)
    ActiveSheet.Cells(12 + numtasks, 1) = "Solution #:"
    For a = 1 To numSolns
        ActiveSheet.Cells(12 + numtasks, 2 * a) = numSolns + a
    Next a

    b = 1
    For a = 1 To numSolns * 2
```

```
    ActiveSheet.Cells(13 + numtasks, a) = "Worker"
    ActiveSheet.Cells(13 + numtasks, a + 1) = "Task"
    ActiveSheet.Cells(14 + 2 * numtasks, 1) = "Total Cost"
    ActiveSheet.Cells(15 + 2 * numtasks, 1) = "Total Skill Levels"
    ActiveSheet.Cells(16 + 2 * numtasks, 1) = "Total Preference"
    cellrow = 14 + numtasks
    For i = 1 To numworkers
       For j = 1 To numtasks
          If bestworkerassign(b, i, j) = 1 Then
             ActiveSheet.Cells(cellrow, a) = i
             ActiveSheet.Cells(cellrow, a + 1) = j
             cellrow = cellrow + 1
          End If
       Next j
    Next i
    ActiveSheet.Cells(14 + 2 * numtasks, a + 1) = bestCost(b)
    ActiveSheet.Cells(15 + 2 * numtasks, a + 1) = bestSgap(b)
    ActiveSheet.Cells(16 + 2 * numtasks, a + 1) = bestPref(b)
    b = b + 1
    a = a + 1
  Next a



End Sub
```

```
Public Sub InitialWorkerPrefSoln_click()

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 workertaskSgap() As Single, oworkertaskSgap() As Single
Dim workertaskPref() As Single, oworkertaskPref() As Single
Dim Prefmatrix() 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, numSolns As Single
Dim totaltaskcost() As Single
Dim totalworkercost() As Single
Dim totaltaskSgap() As Single
Dim totalworkerSgap() As Single
Dim totaltaskPref() As Single
Dim totalworkerPref() As Single
Dim workerphase1() As Single
Dim cellrow As Single
Dim trainingneeds() As Single
Dim skilltrainingneeds() As Single
Dim bestCost() As Single
```

```
Dim bestSgap() As Single
Dim bestPref() As Single

Dim TotalNumObjFn As Single
TotalNumObjFn = 3


Sheets("Parameters").Select
p1perprior = ActiveSheet.Cells(1, 2).Value
p1perrestrict = ActiveSheet.Cells(2, 2).Value

perprior = ActiveSheet.Cells(3, 2).Value
perrestrict = ActiveSheet.Cells(4, 2).Value
numiter = ActiveSheet.Cells(5, 2).Value

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

Sheets("Input").Select
numworkers = ActiveSheet.Cells(2, 2).Value
numskills = ActiveSheet.Cells(3, 2).Value
numtasks = ActiveSheet.Cells(4, 2).Value
numSolns = ActiveSheet.Cells(5, 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 1, 0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim bestworkerassign(0 To numSolns + 1, 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 workertaskSgap(0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim oworkertaskSgap(0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim workertaskPref(0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim oworkertaskPref(0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim Prefmatrix(0 To numworkers + 1, 0 To numskills + 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 totaltaskSgap(0 To numtasks + 1) As Single
ReDim totalworkerSgap(0 To numworkers + 1) As Single
ReDim totaltaskPref(0 To numtasks + 1) As Single
ReDim totalworkerPref(0 To numworkers + 1) As Single
ReDim workerphase1(0 To numworkers + 1) As Single
ReDim trainingneeds(0 To numworkers + 1, 0 To numskills + 1) As Single
ReDim skilltrainingneeds(0 To numskills + 1, 0 To 5) As Single
ReDim bestCost(0 To numSolns + 20) As Single
```

```
ReDim bestSgap(0 To numSolns + 20) As Single
ReDim bestPref(0 To numSolns + 20) As Single

  starttime = Timer


For a = 0 To numSolns + 1
   bestCost(a) = 0
   bestSgap(a) = 0
   bestPref(a) = 0
   For b = 0 To numworkers + 1
      For c = 0 To numtasks + 1
         bestworkerassign(a, b, c) = 0
      Next c
   Next b
Next a


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
      trainingneeds(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
    totaltaskSgap(b) = 0
    totaltaskPref(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
        skilltrainingneeds(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
   totalworkerSgap(b) = 0
   totalworkerPref(b) = 0
   For k = 1 To numtasks
      workerassign(1, b, k) = 0
      workertaskcost(b, k) = 0
      oworkertaskcost(b, k) = 0
      workertaskSgap(b, k) = 0
      oworkertaskSgap(b, k) = 0
      workertaskPref(b, k) = 0
      oworkertaskPref(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(9 + TotalNumObjFn + b, 1 + k)
   Next k
Next b

For b = 1 To numtasks
   For k = 1 To numskills
      otaskskill(b, k) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + b, 1 + k)
   Next k
Next b

For b = 1 To numtasks
   otasktime(b) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + numtasks + 2 + b, 2)
Next b
```

```
For b = 1 To numworkers
   oworkercapacity(b) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + numtasks + 2 + numtasks + 2 + b, 2)
Next b

For i = 1 To numskills
   For j = 2 To 5
      tcost(i, j) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + numtasks + 2 + numtasks + 2 + numworkers + 3 + i, j)
   Next j
Next i

For i = 1 To numskills
   For j = 2 To 5
      ttime(i, j) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + numtasks + 2 + numtasks + 2 + numworkers + 3 +
numskills + 3 + i, j)
   Next j
Next i

For b = 1 To numworkers
   For k = 1 To numskills
      Prefmatrix(b, k) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + numtasks + 2 + numtasks + 2 + numworkers + 3
+ numskills + 3 + numskills + 3 + b, k + 1)
   Next k
Next b


'done reading in values from the excel sheet

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))
               oworkertaskSgap(i, j) = oworkertaskSgap(i, j) + (otaskskill(j, k) - oworkerskill(i, k))
               oworkertaskPref(i, j) = oworkertaskPref(i, j) + Prefmatrix(i, k) * (otaskskill(j, k) - oworkerskill(i, 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)
      totaltaskSgap(j) = totaltaskSgap(j) + oworkertaskSgap(i, j)
      totaltaskPref(j) = totaltaskPref(j) + oworkertaskPref(i, j)
   Next i
Next j


For i = 1 To numworkers
   For j = 1 To numtasks
      totalworkercost(i) = totalworkercost(i) + oworkertaskcost(i, j)
      totalworkerSgap(i) = totalworkerSgap(i) + oworkertaskSgap(i, j)
      totalworkerPref(i) = totalworkerPref(i) + oworkertaskPref(i, j)
   Next j
Next i


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
warned4 = 0

For a = 1 To numSolns + 1
   bestCost(a) = 999999999
   bestSgap(a) = -9999999
   bestPref(a) = -9999999
Next a
bestCost(0) = -9999999
bestSgap(0) = 999999999
bestPref(0) = 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(1, b, k) = 0
```

```
        workertaskcost(b, k) = oworkertaskcost(b, k)
        workertaskSgap(b, k) = oworkertaskSgap(b, k)
        workertaskPref(b, k) = oworkertaskPref(b, k)
        workertasktime(b, k) = oworkertasktime(b, k)
      Next k
   Next b

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


   totalcost = 0
   totalsgap = 0
   totalpref = 0
   numtaskassigned = 0

  If phase1_on = 1 Then
      Do While numtaskassigned < numworkers
         maxpreft1 = -9999999
         For i = 1 To numworkers
            If workerphase1(i) = 0 And totalworkerPref(i) > maxpreft1 Then
               maxpreft1 = totalworkerPref(i)
               maxprefworker = i
            End If
         Next i

         maxpreft2 = -9999999
         For j = 1 To numtasks
            If taskassigned(j) = 0 And workertasktime(maxprefworker, j) <= workercapacity(maxprefworker) And
workertaskPref(maxprefworker, j) > maxpreft2 Then
```

```
            maxpreft2 = workertaskPref(maxprefworker, j)
            maxpreftask = j
          End If
        Next j


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

          'make the best assignment
          If priorrnd <= p1perprior Then
            totalcost = totalcost + workertaskcost(maxprefworker, maxpreftask)
            totalsgap = totalsgap + workertaskSgap(maxprefworker, maxpreftask)
            totalpref = totalpref + workertaskPref(maxprefworker, maxpreftask)
            numtaskassigned = numtaskassigned + 1
            workerassign(1, maxprefworker, maxpreftask) = 1
            taskassigned(maxpreftask) = 1
            workerphase1(maxprefworker) = 1
            workercapacity(maxprefworker) = workercapacity(maxprefworker) - workertasktime(maxprefworker, maxpreftask)
            assignedworker = maxprefworker
            assignedtask = maxpreftask
          End If

          If priorrnd > p1perprior Then
             numonlist = 0
             For i = 1 To numworkers
               If workerphase1(i) = 0 And totalworkerPref(i) >= maxpreft1 * (1 - (p1perrestrict / 100)) Then
                  For j = 1 To numtasks
                    If taskassigned(j) = 0 And workertaskPref(i, j) >= maxpreft2 * (1 - (p1perrestrict / 100)) And workertasktime(i, j)
<= workercapacity(i) Then
                       numonlist = numonlist + 1
```

```
                available(numonlist, 1) = i
                available(numonlist, 2) = j
            End If
        Next j
    End If
Next i

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

totalcost = totalcost + workertaskcost(assignedworker, assignedtask)
totalsgap = totalsgap + workertaskSgap(assignedworker, assignedtask)
totalpref = totalpref + workertaskPref(assignedworker, assignedtask)

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

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

For k = 1 To numskills
    If workerskill(assignedworker, k) < taskskill(assignedtask, k) Then
        workerskill(assignedworker, k) = taskskill(assignedtask, k)
    End If
Next k

For j = 1 To numtasks
```

```
                If taskassigned(j) = 0 Then
                    workertaskcost(assignedworker, j) = 0
                    workertaskSgap(assignedworker, j) = 0
                    workertaskPref(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))
                            workertaskSgap(assignedworker, j) = workertaskSgap(assignedworker, j) + (taskskill(j, k) - workerskill(assignedworker, k))
                            workertaskPref(assignedworker, j) = workertaskPref(assignedworker, j) + Prefmatrix(assignedworker, k) * (taskskill(j, k) - workerskill(assignedworker, 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

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

    Do While numtaskassigned < numtasks 'repeat until all tasks assigned
        maxpreft1 = -9999999
```

```
For j = 1 To numtasks
   If taskassigned(j) = 0 And totaltaskPref(j) > maxpreft1 Then
      maxpreft1 = totaltaskPref(j)
      maxpreftask = j
   End If
Next j

maxpreft2 = -9999999
maxprefworker = 0
For i = 1 To numworkers
   If workertasktime(i, maxpreftask) <= workercapacity(i) And workertaskPref(i, maxpreftask) > maxpreft2 Then
      maxpreft2 = workertaskPref(i, maxpreftask)
      maxprefworker = i
   End If
Next i


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

If maxprefworker > 0 Then
   If priorrnd <= perprior Then
      totalcost = totalcost + workertaskcost(maxprefworker, maxpreftask)
      totalsgap = totalsgap + workertaskSgap(maxprefworker, maxpreftask)
      totalpref = totalpref + workertaskPref(maxprefworker, maxpreftask)

      numtaskassigned = numtaskassigned + 1
      workerassign(1, maxprefworker, maxpreftask) = 1
      taskassigned(maxpreftask) = 1
      workercapacity(maxprefworker) = workercapacity(maxprefworker) - workertasktime(maxprefworker, maxpreftask)
      assignedworker = maxprefworker
```

```
   assignedtask = maxpreftask
End If

If priorrnd > perprior Then
    numonlist = 0
    For j = 1 To numtasks
      If totaltaskPref(j) >= maxpreft1 * (1 - (perrestrict / 100)) And taskassigned(j) = 0 Then
         For i = 1 To numworkers
           If workertaskPref(i, j) >= maxpreft2 * (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)
   totalsgap = totalsgap + workertaskSgap(assignedworker, assignedtask)
   totalpref = totalpref + workertaskPref(assignedworker, assignedtask)
   numtaskassigned = numtaskassigned + 1
   workerassign(1, assignedworker, assignedtask) = 1
   taskassigned(assignedtask) = 1
   workercapacity(assignedworker) = workercapacity(assignedworker) - workertasktime(assignedworker, assignedtask)
End If
```

```
        For k = 1 To numskills
           If workerskill(assignedworker, k) < taskskill(assignedtask, k) Then
              workerskill(assignedworker, k) = taskskill(assignedtask, k)
           End If
        Next k

        For j = 1 To numtasks
           If taskassigned(j) = 0 Then
              workertaskcost(assignedworker, j) = 0
              workertaskSgap(assignedworker, j) = 0
              workertaskPref(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))
                    workertaskSgap(assignedworker, j) = workertaskSgap(assignedworker, j) + (taskskill(j, k) - workerskill(assignedworker, k))
                    workertaskPref(assignedworker, j) = workertaskPref(assignedworker, j) + Prefmatrix(assignedworker, k) * (taskskill(j, k) - workerskill(assignedworker, 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 maxprefworker = 0 Then
```

```
            If warned4 = 0 Then
               response = MsgBox("no feasible solution - not enough worker capacity", vbOKOnly, "Capacity Error")
               warned4 = 1
            End If
            'If response = vbOK Then
            '    Stop
            'End If
         totalpref = -999997
         numtaskassigned = numtasks + 1
      End If

   Loop

For a = 1 To numSolns
   sameassign = 0
   If totalpref = bestPref(a) Then
      For b = 1 To numworkers
         For c = 1 To numtasks
            If workerassign(1, b, c) = bestworkerassign(a, b, c) Then
               sameassign = sameassign + 1
            End If
         Next c
      Next b
   End If
   If sameassign = numworkers * numtasks Then
      totalpref = -9995
   End If
Next a

If totalsgap > bestSgap(numSolns) Then
   For a = numSolns To 1 Step -1
```

```
      If totalpref >= bestPref(a) And totalpref < bestPref(a - 1) Then
         b = numSolns
         c = a
         Do While c < numSolns
            bestCost(b) = bestCost(b - 1)
            bestSgap(b) = bestSgap(b - 1)
            bestPref(b) = bestPref(b - 1)
            For i = 1 To numworkers
               For j = 1 To numtasks
                  bestworkerassign(b, i, j) = bestworkerassign(b - 1, i, j)
               Next j
            Next i
            c = c + 1
            b = b - 1
         Loop

         bestCost(a) = totalcost
         bestSgap(a) = totalsgap
         bestPref(a) = totalpref
         For i = 1 To numworkers
            For j = 1 To numtasks
               bestworkerassign(a, i, j) = workerassign(1, i, j)
            Next j
         Next i
      End If
   Next a
End If
   'Print results
   Sheets("Results Summary").Select
   ActiveSheet.Cells(1, 1) = "Best Solution Costs"
   ActiveSheet.Cells(2, 1) = "Training Amount"
```

```
    ActiveSheet.Cells(3, 1) = "Worker Preference"
    For a = 1 To numSolns
        ActiveSheet.Cells(1, a + 1) = bestCost(a)
    Next a
    For a = 1 To numSolns
        ActiveSheet.Cells(2, a + 1) = bestSgap(a)
    Next a
    For a = 1 To numSolns
        ActiveSheet.Cells(3, a + 1) = bestPref(a)
    Next a

Next r


endtime = Timer
totaltime = endtime - starttime
Sheets("Results Summary").Select
ActiveSheet.Cells(3, numSolns + 3) = "Run Time"
ActiveSheet.Cells(3, numSolns + 4) = totaltime

'Print to Results Summary sheet
    Sheets("Results Summary").Select
    'ActiveSheet.Cells(1, 1) = "Best Solution Costs"
    'ActiveSheet.Cells(1, 2) = bestsolution(1)
    ActiveSheet.Cells(18 + 2 * numtasks, 1) = "Solution #:"
    For a = 1 To numSolns
        ActiveSheet.Cells(18 + 2 * numtasks, 2 * a) = 2 * numSolns + a
    Next a

    b = 1
    For a = 1 To numSolns * 2
```

```
      ActiveSheet.Cells(19 + 2 * numtasks, a) = "Worker"
      ActiveSheet.Cells(19 + 2 * numtasks, a + 1) = "Task"
      cellrow = 20 + 2 * numtasks
      For i = 1 To numworkers
         For j = 1 To numtasks
            If bestworkerassign(b, i, j) = 1 Then
               ActiveSheet.Cells(cellrow, a) = i
               ActiveSheet.Cells(cellrow, a + 1) = j
               cellrow = cellrow + 1
            End If
         Next j
      Next i
      ActiveSheet.Cells(20 + 3 * numtasks, a + 1) = bestCost(b)
      ActiveSheet.Cells(21 + 3 * numtasks, a + 1) = bestSgap(b)
      ActiveSheet.Cells(22 + 3 * numtasks, a + 1) = bestPref(b)
      b = b + 1
      a = a + 1
   Next a
   ActiveSheet.Cells(20 + 3 * numtasks, 1) = "Total Cost"
   ActiveSheet.Cells(21 + 3 * numtasks, 1) = "Total Skill Levels"
   ActiveSheet.Cells(22 + 3 * numtasks, 1) = "Total Preference"

End Sub
```

```
'This command button will find the initial solution for Minimum Training Cost
Public Sub Compromise_click()
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 workertaskOFVs() As Single, oworkertaskOFVs() As Single
Dim workertasktime() As Single, oworkertasktime() As Single
Dim Prefmatrix() 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, numSolns As Single
Dim totaltaskOFVs() As Single
Dim totalworkerOFVs() As Single
Dim workerphase1() As Single
Dim cellrow As Single
Dim trainingneeds() As Single
Dim skilltrainingneeds() As Single
'everything below is new:
Dim bestOFVs() As Single
Dim TotalOFVs() As Single
Dim st2workerassign() As Single
Dim Valuefn() As Single
Dim Lambda() As Single
Dim varB() As Single
```

```
Dim workerswapped() As Single
Dim baseOF() As Single
Dim tempOFVs() As Single
Dim tempwskill() As Single
Dim tempwtOFVs() As Single
Dim workerswap() As Single
Dim tempworkerassign() As Single
Dim TotalNumObjFn As Single
Dim temptaskcost() As Single
Dim temptaskSgap() As Single
Dim temptaskPref() As Single
Dim possibleswaps() As Single
Dim st2OFV() As Single
TotalNumObjFn = 3


'Sheets("Results Summary").Cells.Clear
'Sheets("Worker Training Results").Cells.Clear
'Sheets("Skill Training Results").Cells.Clear


'might need to use different parameters here
Sheets("Parameters").Select
p1perprior = ActiveSheet.Cells(1, 2).Value
p1perrestrict = ActiveSheet.Cells(2, 2).Value

perprior = ActiveSheet.Cells(3, 2).Value
perrestrict = ActiveSheet.Cells(4, 2).Value
numiter = ActiveSheet.Cells(5, 2).Value

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

```
Sheets("Input").Select
numworkers = ActiveSheet.Cells(2, 2).Value
numskills = ActiveSheet.Cells(3, 2).Value
numtasks = ActiveSheet.Cells(4, 2).Value
numSolns = ActiveSheet.Cells(5, 2).Value
maxcompromises = 50

'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
'added a dimension to the two matrices below
ReDim workerassign(0 To 1, 0 To 1, 0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim bestworkerassign(0 To TotalNumObjFn + 1, 0 To numSolns + 1, 0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim workertaskOFVs(0 To TotalNumObjFn + 1, 0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim oworkertaskOFVs(0 To TotalNumObjFn + 1, 0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim Prefmatrix(0 To numworkers + 1, 0 To numskills + 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 totaltaskOFVs(0 To TotalNumObjFn + 1, 0 To numtasks + 1) As Single
ReDim totalworkerOFVs(0 To TotalNumObjFn + 1, 0 To numworkers + 1) As Single
ReDim workerphase1(0 To numworkers + 1) As Single
ReDim trainingneeds(0 To numworkers + 1, 0 To numskills + 1) As Single
ReDim skilltrainingneeds(0 To numskills + 1, 0 To 5) As Single
'new one below:
ReDim bestOFVs(0 To TotalNumObjFn + 1, 0 To TotalNumObjFn + 1, 0 To numSolns + 1) As Single
ReDim TotalOFVs(0 To TotalNumObjFn + 1) As Single
ReDim possibleswaps(0 To numworkers + 1, 0 To numtasks + 1, 0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim st2workerassign(0 To maxcompromises + 1, 0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim Valuefn(0 To maxcompromises + 1) As Single
ReDim Lambda(0 To TotalNumObjFn + 1, 0 To maxcompromises + 1) As Single
ReDim varB(0 To TotalNumObjFn + 1, 0 To maxcompromises + 1) As Single
ReDim baseOF(0 To maxcompromises + 1) As Single
ReDim workerswapped(0 To numworkers + 1) As Single
ReDim tempOFVs(0 To TotalNumObjFn + 1) As Single
ReDim tempwskill(0 To numworkers + 1, 0 To numskills + 1) As Single
ReDim tempwtOFVs(0 To TotalNumObjFn + 1, 0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim workerswap(0 To maxcompromises + 1, 0 To numworkers + 1) As Single
ReDim tempworkerassign(0 To 1, 0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim temptaskcost(0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim temptaskSgap(0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim temptaskPref(0 To numworkers + 1, 0 To numtasks + 1) As Single
ReDim possibleswaps(0 To numworkers + 1, 0 To numtasks + 1, 0 To numworkers + 1, 0 To numtasks + 1, 0 To TotalNumObjFn +
1) As Single
ReDim st2OFV(0 To maxcompromises + 1, 0 To TotalNumObjFn + 1) As Single

starttime = Timer
```

```
'new thing below

For a = 0 To numworkers + 1
   For b = 0 To numtasks + 1
      For i = 0 To numworkers + 1
         For j = 0 To numtasks + 1
            For y = 0 To TotalNumObjFn + 1
               possibleswaps(a, b, i, j, y) = 0
            Next y
         Next j
      Next i
   Next b
Next a

For a = 0 To maxcompromises
   Valuefn(a) = 0
   baseOF(a) = 0
   For i = 1 To numworkers + 1
      For j = 1 To numtasks + 1
         st2workerassign(a, i, j) = 0
      Next j
   Next i
Next a

'read in data from file
For b = 1 To numworkers
   For k = 1 To numskills
      oworkerskill(b, k) = ActiveSheet.Cells(9 + TotalNumObjFn + b, 1 + k)
   Next k
Next b
```

```
For b = 1 To numtasks
   For k = 1 To numskills
      otaskskill(b, k) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + b, 1 + k)
   Next k
Next b

For b = 1 To numtasks
   otasktime(b) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + numtasks + 2 + b, 2)
Next b

For b = 1 To numworkers
   oworkercapacity(b) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + numtasks + 2 + numtasks + 2 + b, 2)
Next b

For i = 1 To numskills
   For j = 2 To 5
      tcost(i, j) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + numtasks + 2 + numtasks + 2 + numworkers + 3 + i, j)
   Next j
Next i

For i = 1 To numskills
   For j = 2 To 5
      ttime(i, j) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + numtasks + 2 + numtasks + 2 + numworkers + 3 +
numskills + 3 + i, j)
   Next j
Next i

For b = 1 To numworkers
   For k = 1 To numskills
```

```
        Prefmatrix(b, k) = ActiveSheet.Cells(9 + TotalNumObjFn + numworkers + 3 + numtasks + 2 + numtasks + 2 + numworkers + 3
+ numskills + 3 + numskills + 3 + b, k + 1)
    Next k
Next b


'done reading in values from the excel sheet

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


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

```
Sheets("Results Summary").Select

    For a = 1 To numSolns
        ActiveSheet.Cells(1, a + 1) = ActiveSheet.Cells(8 + numtasks, 2 * a)
    Next a
    For a = 1 To numSolns
        ActiveSheet.Cells(2, a + 1) = ActiveSheet.Cells(15 + 2 * numtasks, 2 * a)
    Next a
    For a = 1 To numSolns
        ActiveSheet.Cells(3, a + 1) = ActiveSheet.Cells(22 + 3 * numtasks, 2 * a)
    Next a


st2count = 1
prefsoln = Application.InputBox("What is the preferred solution so far?", "")
UserForm1.Label1.Caption = "To what percent would you like to compromise each objective? "
    varB(1, st2count) = 0
    varB(2, st2count) = 0
    varB(3, st2count) = 0

If prefsoln <= numSolns Then
    cellrow = 7
    cellcol = prefsoln * 2
    Do Until baseOF(st2count) = 2 Or baseOF(st2count) = 3
        baseOF(st2count) = InputBox("Which objective do you want to improve? (2-Training Amount, or 3-Worker Preference)", "")
    Loop

    totalcost = ActiveSheet.Cells(cellrow + numtasks + 1, cellcol)
    totalsgap = ActiveSheet.Cells(cellrow + numtasks + 2, cellcol)
    totalpref = ActiveSheet.Cells(cellrow + numtasks + 3, cellcol)
    If baseOF(st2count) = 2 Then
```

```
    'UserForm1.Label1.Caption = UserForm1.Label1.Caption & baseOF(st2count) & " from " & totalsgap & " to " & totalsgap * 1.1
& "(10%)"
    UserForm1.Frame1.Caption = "Total Cost (Obj 1)"
    UserForm1.Frame2.Caption = "Worker Preference (Obj 3)"
    UserForm1.OptionButton1.Caption = totalcost * 0.04 & " (4%)"
    UserForm1.OptionButton2.Caption = totalcost * 0.07 & " (7%)"
    UserForm1.OptionButton3.Caption = totalcost * 0.1 & " (10%)"
    UserForm1.OptionButton4.Caption = totalcost * 0.13 & " (13%)"
    UserForm1.OptionButton5.Caption = totalcost * 0.16 & " (16%)"
    UserForm1.OptionButton6.Caption = totalpref * 0.04 & " (4%)"
    UserForm1.OptionButton7.Caption = totalpref * 0.07 & " (7%)"
    UserForm1.OptionButton8.Caption = totalpref * 0.1 & " (10%)"
    UserForm1.OptionButton9.Caption = totalpref * 0.13 & " (13%)"
    UserForm1.OptionButton10.Caption = totalpref * 0.16 & " (16%)"
    UserForm1.Show
    If (UserForm1.OptionButton1.Value = True) Then varB(1, st2count) = 0.04
    If (UserForm1.OptionButton2.Value = True) Then varB(1, st2count) = 0.07
    If (UserForm1.OptionButton3.Value = True) Then varB(1, st2count) = 0.1
    If (UserForm1.OptionButton4.Value = True) Then varB(1, st2count) = 0.13
    If (UserForm1.OptionButton5.Value = True) Then varB(1, st2count) = 0.16
    If (UserForm1.OptionButton6.Value = True) Then varB(3, st2count) = 0.04
    If (UserForm1.OptionButton7.Value = True) Then varB(3, st2count) = 0.07
    If (UserForm1.OptionButton8.Value = True) Then varB(3, st2count) = 0.1
    If (UserForm1.OptionButton9.Value = True) Then varB(3, st2count) = 0.13
    If (UserForm1.OptionButton10.Value = True) Then varB(3, st2count) = 0.16

  End If
  If baseOF(st2count) = 3 Then
    'UserForm1.Label1.Caption = UserForm1.Label1.Caption & baseOF(st2count) & " from " & totalpref & " to " & totalpref * 1.1
& "(10%)"
    UserForm1.Frame1.Caption = "Total Cost (Obj 1)"
```

```
      UserForm1.Frame2.Caption = "Training Amount (Obj 2)"
      UserForm1.OptionButton1.Caption = totalcost * 0.04 & " (4%)"
      UserForm1.OptionButton2.Caption = totalcost * 0.07 & " (7%)"
      UserForm1.OptionButton3.Caption = totalcost * 0.1 & " (10%)"
      UserForm1.OptionButton4.Caption = totalcost * 0.13 & " (13%)"
      UserForm1.OptionButton5.Caption = totalcost * 0.16 & " (16%)"
      UserForm1.OptionButton6.Caption = totalsgap * 0.04 & " (4%)"
      UserForm1.OptionButton7.Caption = totalsgap * 0.07 & " (7%)"
      UserForm1.OptionButton8.Caption = totalsgap * 0.1 & " (10%)"
      UserForm1.OptionButton9.Caption = totalsgap * 0.13 & " (13%)"
      UserForm1.OptionButton10.Caption = totalsgap * 0.16 & " (16%)"
      UserForm1.Show
      If (UserForm1.OptionButton1.Value = True) Then varB(1, st2count) = 0.04
      If (UserForm1.OptionButton2.Value = True) Then varB(1, st2count) = 0.07
      If (UserForm1.OptionButton3.Value = True) Then varB(1, st2count) = 0.1
      If (UserForm1.OptionButton4.Value = True) Then varB(1, st2count) = 0.13
      If (UserForm1.OptionButton5.Value = True) Then varB(1, st2count) = 0.16
      If (UserForm1.OptionButton6.Value = True) Then varB(2, st2count) = 0.04
      If (UserForm1.OptionButton7.Value = True) Then varB(2, st2count) = 0.07
      If (UserForm1.OptionButton8.Value = True) Then varB(2, st2count) = 0.1
      If (UserForm1.OptionButton9.Value = True) Then varB(2, st2count) = 0.13
      If (UserForm1.OptionButton10.Value = True) Then varB(2, st2count) = 0.16
   End If

End If
If prefsoln > numSolns And prefsoln <= numSolns * 2 Then
   cellrow = 7 + numtasks + 6
   cellcol = (prefsoln - numSolns) * 2
   Do Until baseOF(st2count) = 1 Or baseOF(st2count) = 3
      baseOF(st2count) = InputBox("Which objective do you want to improve? (1-Training Cost, or 3-Worker Preference)", "")
   Loop
```

```
    totalcost = ActiveSheet.Cells(cellrow + numtasks + 1, cellcol)
    totalsgap = ActiveSheet.Cells(cellrow + numtasks + 2, cellcol)
    totalpref = ActiveSheet.Cells(cellrow + numtasks + 3, cellcol)
    If baseOF(st2count) = 1 Then
        'UserForm1.Label1.Caption = UserForm1.Label1.Caption & baseOF(st2count) & " from " & totalcost & " to " & totalcost * 0.9
& "(10%)"
        UserForm1.Frame1.Caption = "Training Amount (Obj 2)"
        UserForm1.Frame2.Caption = "Worker Preference (Obj 3)"
        UserForm1.OptionButton1.Caption = totalsgap * 0.04 & " (4%)"
        UserForm1.OptionButton2.Caption = totalsgap * 0.07 & " (7%)"
        UserForm1.OptionButton3.Caption = totalsgap * 0.1 & " (10%)"
        UserForm1.OptionButton4.Caption = totalsgap * 0.13 & " (13%)"
        UserForm1.OptionButton5.Caption = totalsgap * 0.16 & " (16%)"
        UserForm1.OptionButton6.Caption = totalpref * 0.04 & " (4%)"
        UserForm1.OptionButton7.Caption = totalpref * 0.07 & " (7%)"
        UserForm1.OptionButton8.Caption = totalpref * 0.1 & " (10%)"
        UserForm1.OptionButton9.Caption = totalpref * 0.13 & " (13%)"
        UserForm1.OptionButton10.Caption = totalpref * 0.16 & " (16%)"
        UserForm1.Show
        If (UserForm1.OptionButton1.Value = True) Then varB(2, st2count) = 0.04
        If (UserForm1.OptionButton2.Value = True) Then varB(2, st2count) = 0.07
        If (UserForm1.OptionButton3.Value = True) Then varB(2, st2count) = 0.1
        If (UserForm1.OptionButton4.Value = True) Then varB(2, st2count) = 0.13
        If (UserForm1.OptionButton5.Value = True) Then varB(2, st2count) = 0.16
        If (UserForm1.OptionButton6.Value = True) Then varB(3, st2count) = 0.04
        If (UserForm1.OptionButton7.Value = True) Then varB(3, st2count) = 0.07
        If (UserForm1.OptionButton8.Value = True) Then varB(3, st2count) = 0.1
        If (UserForm1.OptionButton9.Value = True) Then varB(3, st2count) = 0.13
        If (UserForm1.OptionButton10.Value = True) Then varB(3, st2count) = 0.16
    End If
```

```
   If baseOF(st2count) = 3 Then
      'UserForm1.Label1.Caption = UserForm1.Label1.Caption & baseOF(st2count) & " from " & totalpref & " to " & totalpref * 1.1
& "(10%)"
      UserForm1.Frame1.Caption = "Total Cost (Obj 1)"
      UserForm1.Frame2.Caption = "Training Amount (Obj 2)"
      UserForm1.OptionButton1.Caption = totalcost * 0.04 & " (4%)"
      UserForm1.OptionButton2.Caption = totalcost * 0.07 & " (7%)"
      UserForm1.OptionButton3.Caption = totalcost * 0.1 & " (10%)"
      UserForm1.OptionButton4.Caption = totalcost * 0.13 & " (13%)"
      UserForm1.OptionButton5.Caption = totalcost * 0.16 & " (16%)"
      UserForm1.OptionButton6.Caption = totalsgap * 0.04 & " (4%)"
      UserForm1.OptionButton7.Caption = totalsgap * 0.07 & " (7%)"
      UserForm1.OptionButton8.Caption = totalsgap * 0.1 & " (10%)"
      UserForm1.OptionButton9.Caption = totalsgap * 0.13 & " (13%)"
      UserForm1.OptionButton10.Caption = totalsgap * 0.16 & " (16%)"
      UserForm1.Show
      If (UserForm1.OptionButton1.Value = True) Then varB(1, st2count) = 0.04
      If (UserForm1.OptionButton2.Value = True) Then varB(1, st2count) = 0.07
      If (UserForm1.OptionButton3.Value = True) Then varB(1, st2count) = 0.1
      If (UserForm1.OptionButton4.Value = True) Then varB(1, st2count) = 0.13
      If (UserForm1.OptionButton5.Value = True) Then varB(1, st2count) = 0.16
      If (UserForm1.OptionButton6.Value = True) Then varB(2, st2count) = 0.04
      If (UserForm1.OptionButton7.Value = True) Then varB(2, st2count) = 0.07
      If (UserForm1.OptionButton8.Value = True) Then varB(2, st2count) = 0.1
      If (UserForm1.OptionButton9.Value = True) Then varB(2, st2count) = 0.13
      If (UserForm1.OptionButton10.Value = True) Then varB(2, st2count) = 0.16
   End If
End If
If prefsoln > numSolns * 2 And prefsoln <= numSolns * 3 Then
   cellrow = 7 + numtasks + 6 + numtasks + 6
   cellcol = (prefsoln - 2 * numSolns) * 2
```

```
    Do Until baseOF(st2count) = 2 Or baseOF(st2count) = 1
        baseOF(st2count) = InputBox("Which objective do you want to improve? (1-Training Cost, or 2-Training Amount)", "")
    Loop

    totalcost = ActiveSheet.Cells(cellrow + numtasks + 1, cellcol)
    totalsgap = ActiveSheet.Cells(cellrow + numtasks + 2, cellcol)
    totalpref = ActiveSheet.Cells(cellrow + numtasks + 3, cellcol)
    If baseOF(st2count) = 1 Then
        'UserForm1.Label1.Caption = UserForm1.Label1.Caption & baseOF(st2count) & " from " & totalcost & " to " & totalcost * 0.9
& "(10%)"
        UserForm1.Frame1.Caption = "Training Amount (Obj 2)"
        UserForm1.Frame2.Caption = "Worker Preference (Obj 3)"
        UserForm1.OptionButton1.Caption = totalsgap * 0.04 & " (4%)"
        UserForm1.OptionButton2.Caption = totalsgap * 0.07 & " (7%)"
        UserForm1.OptionButton3.Caption = totalsgap * 0.1 & " (10%)"
        UserForm1.OptionButton4.Caption = totalsgap * 0.13 & " (13%)"
        UserForm1.OptionButton5.Caption = totalsgap * 0.16 & " (16%)"
        UserForm1.OptionButton6.Caption = totalpref * 0.04 & " (4%)"
        UserForm1.OptionButton7.Caption = totalpref * 0.07 & " (7%)"
        UserForm1.OptionButton8.Caption = totalpref * 0.1 & " (10%)"
        UserForm1.OptionButton9.Caption = totalpref * 0.13 & " (13%)"
        UserForm1.OptionButton10.Caption = totalpref * 0.16 & " (16%)"
        UserForm1.Show
        If (UserForm1.OptionButton1.Value = True) Then varB(2, st2count) = 0.04
        If (UserForm1.OptionButton2.Value = True) Then varB(2, st2count) = 0.07
        If (UserForm1.OptionButton3.Value = True) Then varB(2, st2count) = 0.1
        If (UserForm1.OptionButton4.Value = True) Then varB(2, st2count) = 0.13
        If (UserForm1.OptionButton5.Value = True) Then varB(2, st2count) = 0.16
        If (UserForm1.OptionButton6.Value = True) Then varB(3, st2count) = 0.04
        If (UserForm1.OptionButton7.Value = True) Then varB(3, st2count) = 0.07
        If (UserForm1.OptionButton8.Value = True) Then varB(3, st2count) = 0.1
```

```
      If (UserForm1.OptionButton9.Value = True) Then varB(3, st2count) = 0.13
      If (UserForm1.OptionButton10.Value = True) Then varB(3, st2count) = 0.16
   End If
   If baseOF(st2count) = 2 Then
      'UserForm1.Label1.Caption = UserForm1.Label1.Caption & baseOF(st2count) & " from " & totalsgap & " to " & totalsgap * 1.1
& "(10%)"
      UserForm1.Frame1.Caption = "Total Cost (Obj 1)"
      UserForm1.Frame2.Caption = "Worker Preference (Obj 3)"
      UserForm1.OptionButton1.Caption = totalcost * 0.04 & " (4%)"
      UserForm1.OptionButton2.Caption = totalcost * 0.07 & " (7%)"
      UserForm1.OptionButton3.Caption = totalcost * 0.1 & " (10%)"
      UserForm1.OptionButton4.Caption = totalcost * 0.13 & " (13%)"
      UserForm1.OptionButton5.Caption = totalcost * 0.16 & " (16%)"
      UserForm1.OptionButton6.Caption = totalpref * 0.04 & " (4%)"
      UserForm1.OptionButton7.Caption = totalpref * 0.07 & " (7%)"
      UserForm1.OptionButton8.Caption = totalpref * 0.1 & " (10%)"
      UserForm1.OptionButton9.Caption = totalpref * 0.13 & " (13%)"
      UserForm1.OptionButton10.Caption = totalpref * 0.16 & " (16%)"
      UserForm1.Show
      If (UserForm1.OptionButton1.Value = True) Then varB(1, st2count) = 0.04
      If (UserForm1.OptionButton2.Value = True) Then varB(1, st2count) = 0.07
      If (UserForm1.OptionButton3.Value = True) Then varB(1, st2count) = 0.1
      If (UserForm1.OptionButton4.Value = True) Then varB(1, st2count) = 0.13
      If (UserForm1.OptionButton5.Value = True) Then varB(1, st2count) = 0.16
      If (UserForm1.OptionButton6.Value = True) Then varB(3, st2count) = 0.04
      If (UserForm1.OptionButton7.Value = True) Then varB(3, st2count) = 0.07
      If (UserForm1.OptionButton8.Value = True) Then varB(3, st2count) = 0.1
      If (UserForm1.OptionButton9.Value = True) Then varB(3, st2count) = 0.13
      If (UserForm1.OptionButton10.Value = True) Then varB(3, st2count) = 0.16
   End If
End If
```

```
For j = 1 To numtasks
   st2workerassign(st2count, ActiveSheet.Cells(cellrow + j, cellcol - 1), ActiveSheet.Cells(cellrow + j, cellcol)) = 1
Next j

st2OFV(st2count, 1) = totalcost
st2OFV(st2count, 2) = totalsgap
st2OFV(st2count, 3) = totalpref

'want to make this print on a new sheet
Sheets("Results Summary").Select
ActiveSheet.Cells.Clear

ActiveSheet.Cells(1, 1) = "Iteration"
ActiveSheet.Cells(1, 2) = st2count

cellrow = 2
For i = 1 To numworkers
For j = 1 To numtasks
   If st2workerassign(st2count, i, j) = 1 Then
      ActiveSheet.Cells(cellrow, st2count) = i
      ActiveSheet.Cells(cellrow, st2count + 1) = j
      cellrow = cellrow + 1
   End If
Next j
Next i
ActiveSheet.Cells(2 + numtasks, 1) = "Total Cost"
ActiveSheet.Cells(3 + numtasks, 1) = "Total Training Amount"
ActiveSheet.Cells(4 + numtasks, 1) = "Total Worker Preference"
ActiveSheet.Cells(2 + numtasks, 2) = st2OFV(st2count, 1)
ActiveSheet.Cells(3 + numtasks, 2) = st2OFV(st2count, 2)
```

```
ActiveSheet.Cells(4 + numtasks, 2) = st2OFV(st2count, 3)



usersatisfied = 0
Do While usersatisfied = 0

   For y = 1 To numworkers
      workerswap(st2count, y) = 0
   Next y

   For i1 = 1 To numworkers
   For j1 = 1 To numtasks
   If st2workerassign(st2count, i1, j1) = 1 Then
      For i2 = 1 To numworkers
      For j2 = 1 To numtasks
         If st2workerassign(st2count, i2, j2) = 1 And i2 > i1 Then
            'zero out temptaskcost here for all i and j
            For i3 = 1 To numworkers
            For j3 = 1 To numtasks
               tempworkerassign(1, i3, j3) = st2workerassign(st2count, i3, j3)
            Next j3
            Next i3

            tempworkerassign(1, i1, j1) = 0
            tempworkerassign(1, i2, j2) = 0
            tempworkerassign(1, i1, j2) = 1
            tempworkerassign(1, i2, j1) = 1
            'we now have the new assignment to be considered against the Value Fn
```

```
   tempcost = 0
   tempsgap = 0
   temppref = 0
   For i4 = 1 To numworkers
   For j4 = 1 To numtasks
      If tempworkerassign(1, i4, j4) = 1 Then
         For a = 1 To numworkers
         For b = 1 To numtasks
            temptaskcost(a, b) = 0
            temptaskSgap(a, b) = 0
            temptaskPref(a, b) = 0
         Next b
         Next a

         For k = 1 To numskills
            If oworkerskill(i4, k) < otaskskill(j4, k) And otaskskill(j4, k) > 1 Then
               temptaskcost(i4, j4) = temptaskcost(i4, j4) + otraincost(k, oworkerskill(i4, k), otaskskill(j4, k))
               temptaskSgap(i4, j4) = temptaskSgap(i4, j4) + (otaskskill(j4, k) - oworkerskill(i4, k))
               temptaskPref(i4, j4) = temptaskPref(i4, j4) + Prefmatrix(i4, k) * (otaskskill(j4, k) - oworkerskill(i4, k))
            End If
         Next k
         tempcost = tempcost + temptaskcost(i4, j4)
         tempsgap = tempsgap + temptaskSgap(i4, j4)
         temppref = temppref + temptaskPref(i4, j4)
      End If
   Next j4
   Next i4
   possibleswaps(i1, j1, i2, j2, 1) = tempcost - st2OFV(st2count, 1)
   possibleswaps(i1, j1, i2, j2, 2) = tempsgap - st2OFV(st2count, 2)
   possibleswaps(i1, j1, i2, j2, 3) = temppref - st2OFV(st2count, 3)
End If
```

```
      Next j2
      Next i2
    End If
    Next j1
    Next i1

    For i = 1 To numworkers
    For j = 1 To numtasks
      st2workerassign(st2count + 1, i, j) = st2workerassign(st2count, i, j)
    Next j
    Next i


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
NEWcost = st2OFV(st2count, 1)
NEWsgap = st2OFV(st2count, 2)
NEWpref = st2OFV(st2count, 3)

If baseOF(st2count) = 1 Then
Do Until NEWsgap <= st2OFV(st2count, 2) * (1 - varB(2, st2count)) Or NEWpref <= st2OFV(st2count, 3) * (1 - varB(3, st2count))
    mindeltacost = 0
    For a = 1 To numworkers
      If workerswap(st2count, a) = 0 Then
        For b = 1 To numtasks
          If st2workerassign(st2count, a, b) = 1 Then
            For i = 1 To numworkers
              If workerswap(st2count, i) = 0 Then
                For j = 1 To numtasks
                  If st2workerassign(st2count, i, j) = 1 Then
                    If possibleswaps(a, b, i, j, 1) < mindeltacost Then
                      besti1 = a
```

```
                        bestj1 = b
                        besti2 = i
                        bestj2 = j
                        mindeltacost = possibleswaps(a, b, i, j, 1)
                    End If
                End If
            Next j
        End If
    Next i
End If
Next b
End If
Next a

workerswap(st2count, besti1) = 1
workerswap(st2count, besti2) = 1

st2workerassign(st2count + 1, besti1, bestj1) = 0
st2workerassign(st2count + 1, besti2, bestj2) = 0
st2workerassign(st2count + 1, besti1, bestj2) = 1
st2workerassign(st2count + 1, besti2, bestj1) = 1

NEWsgap = NEWsgap + possibleswaps(besti1, bestj1, besti2, bestj2, 2)
NEWpref = NEWsgap + possibleswaps(besti1, bestj1, besti2, bestj2, 3)
Loop
End If
If baseOF(st2count) = 2 Then
Do Until NEWcost >= st2OFV(st2count, 1) * (1 + varB(1, st2count)) Or NEWpref <= st2OFV(st2count, 3) * (1 - varB(3, st2count))
maxdeltasgap = 0
For a = 1 To numworkers
    If workerswap(st2count, a) = 0 Then
```

```
      For b = 1 To numtasks
         If st2workerassign(st2count, a, b) = 1 Then
            For i = 1 To numworkers
               If workerswap(st2count, i) = 0 Then
                  For j = 1 To numtasks
                     If st2workerassign(st2count, i, j) = 1 Then
                        If possibleswaps(a, b, i, j, 2) > maxdeltasgap Then
                           besti1 = a
                           bestj1 = b
                           besti2 = i
                           bestj2 = j
                           maxdeltasgap = possibleswaps(a, b, i, j, 2)
                        End If
                     End If
                  Next j
               End If
            Next i
         End If
      Next b
   End If
Next a

workerswap(st2count, besti1) = 1
workerswap(st2count, besti2) = 1

st2workerassign(st2count + 1, besti1, bestj1) = 0
st2workerassign(st2count + 1, besti2, bestj2) = 0
st2workerassign(st2count + 1, besti1, bestj2) = 1
st2workerassign(st2count + 1, besti2, bestj1) = 1

NEWcost = NEWcost + possibleswaps(besti1, bestj1, besti2, bestj2, 1)
```

```
   NEWpref = NEWpref + possibleswaps(besti1, bestj1, besti2, bestj2, 3)
Loop
End If
If baseOF(st2count) = 3 Then
Do Until NEWcost >= st2OFV(st2count, 1) * (1 + varB(1, st2count)) Or NEWsgap <= st2OFV(st2count, 2) * (1 - varB(2, st2count))
   maxdeltapref = 0
   For a = 1 To numworkers
      If workerswap(st2count, a) = 0 Then
         For b = 1 To numtasks
            If st2workerassign(st2count, a, b) = 1 Then
               For i = 1 To numworkers
                  If workerswap(st2count, i) = 0 Then
                     For j = 1 To numtasks
                        If st2workerassign(st2count, i, j) = 1 Then
                           If possibleswaps(a, b, i, j, 3) > maxdeltapref Then
                              besti1 = a
                              bestj1 = b
                              besti2 = i
                              bestj2 = j
                              maxdeltapref = possibleswaps(a, b, i, j, 3)
                           End If
                        End If
                     Next j
                  End If
               Next i
            End If
         Next b
      End If
   Next a

   workerswap(st2count, besti1) = 1
```

```
    workerswap(st2count, besti2) = 1

    st2workerassign(st2count + 1, besti1, bestj1) = 0
    st2workerassign(st2count + 1, besti2, bestj2) = 0
    st2workerassign(st2count + 1, besti1, bestj2) = 1
    st2workerassign(st2count + 1, besti2, bestj1) = 1

    NEWcost = NEWcost + possibleswaps(besti1, bestj1, besti2, bestj2, 1)
    NEWsgap = NEWsgap + possibleswaps(besti1, bestj1, besti2, bestj2, 2)
Loop
End If

    st2count = st2count + 1

    st2OFV(st2count, 1) = 0
    st2OFV(st2count, 2) = 0
    st2OFV(st2count, 3) = 0
    For i4 = 1 To numworkers
    For j4 = 1 To numtasks
    If st2workerassign(st2count, i4, j4) = 1 Then

        For a = 1 To numworkers
        For b = 1 To numtasks
            temptaskcost(a, b) = 0
            temptaskSgap(a, b) = 0
            temptaskPref(a, b) = 0
        Next b
        Next a

        For k = 1 To numskills
            If oworkerskill(i4, k) < otaskskill(j4, k) And otaskskill(j4, k) > 1 Then
```

```
            temptaskcost(i4, j4) = temptaskcost(i4, j4) + otraincost(k, oworkerskill(i4, k), otaskskill(j4, k))
            temptaskSgap(i4, j4) = temptaskSgap(i4, j4) + (otaskskill(j4, k) - oworkerskill(i4, k))
            temptaskPref(i4, j4) = temptaskPref(i4, j4) + Prefmatrix(i4, k) * (otaskskill(j4, k) - oworkerskill(i4, k))
        End If
    Next k

    st2OFV(st2count, 1) = st2OFV(st2count, 1) + temptaskcost(i4, j4)
    st2OFV(st2count, 2) = st2OFV(st2count, 2) + temptaskSgap(i4, j4)
    st2OFV(st2count, 3) = st2OFV(st2count, 3) + temptaskPref(i4, j4)
End If
Next j4
Next i4

ActiveSheet.Cells(1, 2 * st2count) = st2count

cellrow = 2
For i = 1 To numworkers
For j = 1 To numtasks
If st2workerassign(st2count, i, j) = 1 Then
    ActiveSheet.Cells(cellrow, st2count * 2 - 1) = i
    ActiveSheet.Cells(cellrow, st2count * 2) = j
    cellrow = cellrow + 1
End If
Next j
Next i
ActiveSheet.Cells(2 + numtasks, st2count * 2) = st2OFV(st2count, 1)
ActiveSheet.Cells(3 + numtasks, st2count * 2) = st2OFV(st2count, 2)
ActiveSheet.Cells(4 + numtasks, st2count * 2) = st2OFV(st2count, 3)

usersatisfied = Application.InputBox("Is this a satisfactory assignment (1)? Run compromise stage again(0)", "")
```

```
    If usersatisfied = 1 Then
        MsgBox ("Done")
    End If
    If usersatisfied <> 1 Then

    Do Until baseOF(st2count) = 1 Or baseOF(st2count) = 2 Or baseOF(st2count) = 3
        baseOF(st2count) = InputBox("Which objective do you want to improve? (1-Cost, 2-Training, 3-Preference)", "")
    Loop

    UserForm1.Label1.Caption = "To what percent would you like to compromise each objective? "

    varB(1, st2count) = 0
    varB(2, st2count) = 0
    varB(3, st2count) = 0

    If baseOF(st2count) = 2 Then
        'UserForm1.Label1.Caption = UserForm1.Label1.Caption & baseOF(st2count) & " from " & st2OFV(st2count, 2) & " to " &
st2OFV(st2count, 2) * 1.1 & "(10%)"
        UserForm1.Frame1.Caption = "Total Cost (Obj 1)"
        UserForm1.Frame2.Caption = "Worker Preference (Obj 3)"
        UserForm1.OptionButton1.Caption = st2OFV(st2count, 1) * 0.04 & " (4%)"
        UserForm1.OptionButton2.Caption = st2OFV(st2count, 1) * 0.07 & " (7%)"
        UserForm1.OptionButton3.Caption = st2OFV(st2count, 1) * 0.1 & " (10%)"
        UserForm1.OptionButton4.Caption = st2OFV(st2count, 1) * 0.13 & " (13%)"
        UserForm1.OptionButton5.Caption = st2OFV(st2count, 1) * 0.16 & " (16%)"
        UserForm1.OptionButton6.Caption = st2OFV(st2count, 3) * 0.04 & " (4%)"
        UserForm1.OptionButton7.Caption = st2OFV(st2count, 3) * 0.07 & " (7%)"
        UserForm1.OptionButton8.Caption = st2OFV(st2count, 3) * 0.1 & " (10%)"
        UserForm1.OptionButton9.Caption = st2OFV(st2count, 3) * 0.13 & " (13%)"
        UserForm1.OptionButton10.Caption = st2OFV(st2count, 3) * 0.16 & " (16%)"
        UserForm1.Show
```

```
    If (UserForm1.OptionButton1.Value = True) Then varB(1, st2count) = 0.04
    If (UserForm1.OptionButton2.Value = True) Then varB(1, st2count) = 0.07
    If (UserForm1.OptionButton3.Value = True) Then varB(1, st2count) = 0.1
    If (UserForm1.OptionButton4.Value = True) Then varB(1, st2count) = 0.13
    If (UserForm1.OptionButton5.Value = True) Then varB(1, st2count) = 0.16
    If (UserForm1.OptionButton6.Value = True) Then varB(3, st2count) = 0.04
    If (UserForm1.OptionButton7.Value = True) Then varB(3, st2count) = 0.07
    If (UserForm1.OptionButton8.Value = True) Then varB(3, st2count) = 0.1
    If (UserForm1.OptionButton9.Value = True) Then varB(3, st2count) = 0.13
    If (UserForm1.OptionButton10.Value = True) Then varB(3, st2count) = 0.16
  End If

  If baseOF(st2count) = 3 Then
    'UserForm1.Label1.Caption = UserForm1.Label1.Caption & baseOF(st2count) & " from " & st2OFV(st2count, 3) & " to " &
st2OFV(st2count, 3) * 1.1 & "(10%)"
    UserForm1.Frame1.Caption = "Total Cost (Obj 1)"
    UserForm1.Frame2.Caption = "Training Amount (Obj 2)"
    UserForm1.OptionButton1.Caption = st2OFV(st2count, 1) * 0.04 & " (4%)"
    UserForm1.OptionButton2.Caption = st2OFV(st2count, 1) * 0.07 & " (7%)"
    UserForm1.OptionButton3.Caption = st2OFV(st2count, 1) * 0.1 & " (10%)"
    UserForm1.OptionButton4.Caption = st2OFV(st2count, 1) * 0.13 & " (13%)"
    UserForm1.OptionButton5.Caption = st2OFV(st2count, 1) * 0.16 & " (16%)"
    UserForm1.OptionButton1.Caption = st2OFV(st2count, 2) * 0.04 & " (4%)"
    UserForm1.OptionButton2.Caption = st2OFV(st2count, 2) * 0.07 & " (7%)"
    UserForm1.OptionButton3.Caption = st2OFV(st2count, 2) * 0.1 & " (10%)"
    UserForm1.OptionButton4.Caption = st2OFV(st2count, 2) * 0.13 & " (13%)"
    UserForm1.OptionButton5.Caption = st2OFV(st2count, 2) * 0.16 & " (16%)"
    UserForm1.Show
    If (UserForm1.OptionButton1.Value = True) Then varB(1, st2count) = 0.04
    If (UserForm1.OptionButton2.Value = True) Then varB(1, st2count) = 0.07
    If (UserForm1.OptionButton3.Value = True) Then varB(1, st2count) = 0.1
```

```
    If (UserForm1.OptionButton4.Value = True) Then varB(1, st2count) = 0.13
    If (UserForm1.OptionButton5.Value = True) Then varB(1, st2count) = 0.16
    If (UserForm1.OptionButton1.Value = True) Then varB(2, st2count) = 0.04
    If (UserForm1.OptionButton2.Value = True) Then varB(2, st2count) = 0.07
    If (UserForm1.OptionButton3.Value = True) Then varB(2, st2count) = 0.1
    If (UserForm1.OptionButton4.Value = True) Then varB(2, st2count) = 0.13
    If (UserForm1.OptionButton5.Value = True) Then varB(2, st2count) = 0.16
  End If

  If baseOF(st2count) = 1 Then
    'UserForm1.Label1.Caption = UserForm1.Label1.Caption & baseOF(st2count) & " from " & st2OFV(st2count, 1) & " to " &
st2OFV(st2count, 1) * 0.9 & "(10%)"
    UserForm1.Frame1.Caption = "Training Amount (Obj 2)"
    UserForm1.Frame2.Caption = "Worker Preference (Obj 3)"
    UserForm1.OptionButton1.Caption = st2OFV(st2count, 2) * 0.04 & " (4%)"
    UserForm1.OptionButton2.Caption = st2OFV(st2count, 2) * 0.07 & " (7%)"
    UserForm1.OptionButton3.Caption = st2OFV(st2count, 2) * 0.1 & " (10%)"
    UserForm1.OptionButton4.Caption = st2OFV(st2count, 2) * 0.13 & " (13%)"
    UserForm1.OptionButton5.Caption = st2OFV(st2count, 2) * 0.16 & " (16%)"
    UserForm1.OptionButton6.Caption = st2OFV(st2count, 3) * 0.04 & " (4%)"
    UserForm1.OptionButton7.Caption = st2OFV(st2count, 3) * 0.07 & " (7%)"
    UserForm1.OptionButton8.Caption = st2OFV(st2count, 3) * 0.1 & " (10%)"
    UserForm1.OptionButton9.Caption = st2OFV(st2count, 3) * 0.13 & " (13%)"
    UserForm1.OptionButton10.Caption = st2OFV(st2count, 3) * 0.16 & " (16%)"
    UserForm1.Show
    If (UserForm1.OptionButton1.Value = True) Then varB(2, st2count) = 0.04
    If (UserForm1.OptionButton2.Value = True) Then varB(2, st2count) = 0.07
    If (UserForm1.OptionButton3.Value = True) Then varB(2, st2count) = 0.1
    If (UserForm1.OptionButton4.Value = True) Then varB(2, st2count) = 0.13
    If (UserForm1.OptionButton5.Value = True) Then varB(2, st2count) = 0.16
    If (UserForm1.OptionButton6.Value = True) Then varB(3, st2count) = 0.04
```

```
    If (UserForm1.OptionButton7.Value = True) Then varB(3, st2count) = 0.07
    If (UserForm1.OptionButton8.Value = True) Then varB(3, st2count) = 0.1
    If (UserForm1.OptionButton9.Value = True) Then varB(3, st2count) = 0.13
    If (UserForm1.OptionButton10.Value = True) Then varB(3, st2count) = 0.16
  End If


End If
Loop 'this loop ends when usersatisfied=1

End Sub
```