An experimental comparison of distributed algorithms simulating human detailers and an extension of the Kuhn-Munkres algorithm for the Sailor Assignment Problem (SAP)

by

Jesús Alfredo Burbano Portilla

A dissertation submitted in partial fulfillment of the requirements for the degree of

Master of Sciences

Master in Systems Engineering and Computing
in The National University of Colombia
2010

Graduate Committee:
Professor Germán Hernández, PhD, Chair
Professor Dipankar Dasgupta, PhD, Co-Chair
Professor Gerardo Astaiza, MSc
Professor Yoan Pinzón, PhD
Abstract

An experimental comparison of distributed algorithms simulating human detailers and an extension of the Kuhn-Munkres algorithm for the Sailor Assignment Problem (SAP)

by

Jesús Alfredo Burbano Portilla

Co-Chairs: Germán Hernández, PhD
Dipankar Dasgupta, PhD

Variations of the Linear Assignment Problem (LAP) and variations of the Kuhn-Munkres (KM) algorithm, a solving algorithm for the LAP also known as The Hungarian method, have been applied to the Navy Enlisted Personnel Assignment process in order to solve a problem better known as the Sailor Assignment Problem (SAP). The SAP is a multi-objective optimization problem that can be reduced through the use of weight-vectors into multiple LAPs. This study focuses on the comparison of the solutions obtained by KM when it is applied to LAPs that are weighted versions of instances of SAP and the solutions obtained by simulating human detailers that currently do this assignment process manually. Two models of behavior of the human detailers: the greedy on-line and the random on-line, are studied here. The goal is to evaluate which approach is a better alternative to find competitive solutions for SAP.
Cada letra que aprendes, es una letra más que vales
— Serafín Portilla
to my Family
I want to express my gratitude and appreciation to many people who helped and supported me while I was working on this dissertation:

To my mother, who everyday encouraged me to get education.
To my father, I saw him studying all night, many times; And I want to do it too.
To Lisseth, my sister, who provided inspiration, constructive criticism, and many ideas in my research.
To Maribell, my older sister, who helped me everytime that I needed it.
To Angélica Gomez for her love and tolerance.
To Gabriela Salamanca for her comments and reviewing.
To Gomer Gonzales for his support, specially that february 28 of 2008...
Rodrigo Silva, Andres Millan, Aishuarya Kaushal, for the generosity and hospitality in my semester in Memphis
To my fellows, classmates, and professors in this 3 years at the National University of Colombia.
And especially to Professors Germán Hernández, Fernando Niño, Deon Garrett and Dipankar Dasgupta.
List of Tables

Table

4.1 Number of Weight Vectors by number of steps .......................... 19
5.1 Characteristics of the Problem instances ............................... 22
List of Figures

Figure
1.1 Problems and Methods related to SAP ................................................................. 3
2.1 “Example of a Pareto front. The boxed points represent feasible choices, and smaller values are preferred to larger ones. Point C is not on the Pareto Frontier because it is dominated by both point A and point B. Points A and B are not strictly dominated by any other, and hence do lie on the frontier.” from Wikipedia. (accessed 10 Jul 2009) Available from: http://en.wikipedia.org/wiki/Pareto_efficiency ................................................................. 7
2.2 Linear Assignment Problem. $C_{ij}$ denotes the cost of assigning agent $i$ to task $j$ ........ 8
2.3 Figure(a) represents complete bipartite graph figure (b) shows incomplete graph ........ 11
2.4 Sparse Matrix Representation ................................................................. 11
3.1 GUI for the new SAP instance generator ................................................. 14
4.1 General Architecture for Simulating the SAP .............................................. 16
5.1 Random Online non-dominated solutions and KM solutions by sets of characteristics. .......... 23
5.2 Random Online non-dominated solutions and KM solutions by 10 instances for 10 sailors, 20 jobs, and 20 preferences ($s_{10j20p20}$), along the abscissa direction are enumerated the number of the problem instance ................................................................. 23
5.3 Greedy Online non-dominated solutions and KM solutions by sets of characteristics. .......... 24
5.4 Greedy Online non-dominated solutions and KM solutions by 10 instances for 10 sailors, 20 jobs, and 20 preferences ($s_{10j20p20}$), along the abscissa direction are enumerated the number of the problem instance ................................................................. 24
5.5 Random Online non-dominated and dominated solutions by KM solutions. .............. 26
5.6 Greedy Online non-dominated and dominated solutions by KM solutions .......... 26
5.7 Random Online non-dominate and non-comparable solutions. ....................... 27
5.8 Greedy Online non-dominate and non-comparable solutions. ....................... 27
5.9 Random Online solutions percents ............................................................... 28
5.10 Greedy Online solutions percents. ............................................................... 28

vii
Chapter 1

Introduction

Matching problems, also known as assignment problems, is a wide and important class of problems in computer science and combinatorics that have also a wide range of applications. In a matching problem we are given \( n \) (agents, boys, sailors, bets, students...) to be matched with \( m \) (tasks, girls, jobs, adwords, schools...), we may have restrictions on the agents can perform the particular tasks or an associated cost or also we may have preference lists for the assignment and maybe one of maybe multiple objectives to optimize. The problem is how: to pair, to match, to assign, to make them correspond off, in an “optimal” way respecting the constrains. In practical situations we find many examples of matching problems with thousands of elements to match \[1; 2; 10; 20; 28\]. Two basic models are one-to-one and many-to-one matching models. If one agent is matched with only one task on the other side, then this is a one-to-one matching \[8\]. But if one agent is matched with more than one task on the other side, such as clustering data sets, assigning students to schools, etc., then is a many-to-one matching \[3; 24\]. Matching may require to accomplish some conditions to improve some outcomes (to minimize or to maximize a function or more, to guarantee stability, etc., by choosing the values of real or integer variables from within an allowed set).

In its most general form, an assignment problem can be seen as finding a maximum weight matching in a weighted bipartite graph and can be stated as follows \[6\]:

"There are a number of agents and a number of tasks. Any agent can be assigned to perform any task, incurring some cost that may vary depending on the agent-task assignment. It is required to perform all tasks by assigning exactly one agent to each task in such a way that the total cost of the assignment is minimized."

If the number of agents and tasks are the same and the total cost of the assignment for all tasks is equal to the sum of the costs for each agent, then the problem is called the Linear Assignment Problem (LAP) \[6\]. The LAP has been widely studied, the most efficient algorithm to solve it is the Hungarian algorithm \[18; 22\] that solves the linear assignment problem within time bounded by a polynomial expression of the number of agents \( O(n^3) \).

Minimizing or maximizing simultaneously multiple, and sometimes conflicting, objectives is important in real applications giving origin of the so called multi-objective optimization (or programming), also known as multi-criteria or multi-attribute optimization \[25; 27\]. The most wildly used technique is the Multiple Objective Linear Programming (MOLP) that often has a large number of solutions in the form of extreme efficient (minimal or maximal) points and unbounded efficient edges. The process of finding all the efficient solutions in MOLP problem or even a subset of them is very time and space consuming \[29\].

The United States Navy’s Sailor Assignment Problem (SAP) is a many objective optimization problem \[10\]. According to the Navy’s personnel policies, roughly every few years (two to four) sailors serving on
active duty are reassigned to a different job. Approximately 300,000 sailors serve in the Navy and 120,000 are reassigned each year according to [12]. Assigning U.S. Navy personnel to jobs is currently a manual process performed by enlisted detailers, close to 300 human operators with the support of an Enlisted Assignment Information System [10; 19] (EAIS) help sailors to find out an affordable job while trying at the same time to satisfy the needs and goals of the Navy.

In the current process, it is possible that detailers list the personnel according to their skills and assign the most qualified sailor to the billet, billet is the denomination that the Navy gives to job openings. Alternatively they may try to consider first the sailors’s preferences and assign to each sailor his/her preference. Assignments do not always satisfy the competing preferences of personnel and billets. Involuntary assignments may keep positions filled in the short run, but in the long run they may hurt from recruiting, readiness and retention [16].

This problem is closely related to the College (stable) Admission Problem (a special case of many-to-one [23; 26] in what “An assignment of applicants to colleges will be called unstable if there are two applicants α and β who are assigned to colleges A and B, respectively, although β prefers A to B and A prefers β to α” [8]), like the assigning of many students (instead sailors) to some schools (instead kinds of jobs), but the stability in this problem does not seem mandatory, mainly because the sailors do not know better or worse options of allocation in fact and the costs of satisfy the sailor’s needs could be very expensives [10], although stable matches are important in a voluntary labor market, the Gale-Shapley algorithm (a.k.a. Deferred Acceptance algorithm) may still favor one party depending on whether the command or sailor biased form of the algorithm is used. On the other hand, the SAP is studied as a Linear Assignment Problem; single objective instances of this problem are obtained using weight vectors to scale the natural multi-objective formulation; then the goal is to identify sailor and job matches that maximize or minimize some criterion [6], it promotes a balanced approach to meeting the preferences of both parties but does not guarantee stable matches.

In general, the Navy would like to assign the jobs to the sailors in such a way that [6]: i) the training of the sailors for the jobs assigned to them is maximized, ii) the cost of relocating the sailors to the new jobs is minimized, iii) the satisfaction of the sailors with the jobs assigned to them is maximized, iv) the satisfaction of the commanders with sailors assigned to the jobs in their units is maximized, and v) the number of unassigned sailors is minimized. Goals i, ii and iv are related to the needs of the Navy while iii and v are related to needs of the sailor. Give the nature of the problem the Navy’s needs have priority over the sailor’s needs, but, in recent years the recruiting problems have prompted a strong interest to keep sailors satisfied.

Over the past ten years different aproaches have been explored including agent based systems, multi-objective evolutionary techniques, extensions of the Gale-Shapley algorithm for the stable marriage and the extensions of the Kuhn-Munkres in order to solve multiobjective optimization problems. Figure 1.1 provides a problem clasification scheme (in ovals) and a solution methods scheme (in rectangles), the red path is the interesting path for this study and the colored rectangles are the most recent methods studied by the Intelligent Security Systems Research Laboratory (ISSRL) at The University of Memphis.

A recent study [14] shows that for problems with more that ten objectives a purely random search may perform favorably compared with an evolutionary technique.

Distributed algorithms simulating strategic behaviours of human detailers have been not studied to solve the Sailor Assignment Problem.

This work was intended to further understanding of how close are the solutions obtained by the human detailers to the "optimal pareto solutions". Intuitively the SAP instances could be divided into independent sub-problems because different job categories have almost a complete set of independent applicants, like for
example the set of applicants to "A" job will be almost independent of the applicants to "Z" job. This is the criterion to decompose the large SAP instances in smaller independent sub-instances what can be solved by the extension of Kuhn-Munkres for SAP in feasible time [5] or, as is done by the human or simulated detailers, trying to produce matches which keep the sailors and commands happy, while minimizing the total cost of implementing the match.

The simulation of human detailers in this work is only refered to the performance of assignment achieved by simulation of human decisions, neither about time of attention nor information processing.
Chapter 2

The Sailor Assignment — problem and solution attempts

The Sailor Assignment Problem (SAP) is a multi-objective optimization problem. The Aggregated Dimension SAP is a reduction achieved through the use of weight-vectors that converts SAP to a multiple (theoretically infinite) $P$-problems\(^1\) that could be solved by Kuhn Munkres algorithm (KM), but the problem of find ALL the non-dominated solutions\(^2\) in the SAP is an $NP$-problem.

In order to understand better the current approaches this chapter describes briefly the most relevant solutions attempts, the concepts of Pareto Front, Online and Offline assignment, the Linear Assignment Problem (the simplest assignment problem), and The KM, also called Hungarian algorithm, that is one of many algorithms that have been devised that solve the linear assignment problem within time bounded by a polynomial expression of the number of agents. Although it is possible to solve any of LAPs using the simplex algorithm, KM was chosen in order to take advantage of its special structure (Modified for Rectangular Matrices) and because it has been studied for the SAP at the Intelligent Security Systems Research Laboratory (ISSRL) at The University of Memphis.

Another approaches had been studied to solve the SAP, such as Deferred Acceptance (DA) proposed by Gale and Shapley (1962) that has had a profound influence on market design, by being adapted into practical matching mechanisms, and, by raising new theoretical questions, but the optimal this achieves is an “optimal” match that favors only one side of the labor market. Linear Programming, that does not provide stability like DA, was studied by Joshua H. Ho and Eng Hwee Low\(^3\) and can provide a more balanced approach, ending up with an assignment that caters to both parties and the number of unstable matches had not been examined to measure it against the benefit of higher system effectiveness.

Dasgupta and Garrett in [10] used genetic algorithms to solve single objective SAP and compare the results thus obtained with the Gale-Shapley algorithm [8]. The Gale-Shapley algorithm is a quadratic time algorithm for finding an optimal set of stable matches, i.e. marriages and produces a set of marriages that is stable and optimal with respect to men preferences after $O(n^2)$ steps. If minimizing the total permanent

---

1. A problem is assigned to the $P$ (polynomial time) class if there exists at least one algorithm to solve that problem, such that the number of steps of the algorithm is bounded by a polynomial in $n$, where $n$ is the length of the input.

2. see Pareto Front

change of station cost (PCS) is not one of the objective in SAP, the results produced by Gale-Shapley algorithm are optimal set of assignments with respect to the preferences of the sailors and commandeers and such a match may not necessarily involve all sailors. Thus, they examined an alternative genetic algorithm approaches to the SAP in order to allow outside constraints to influence the quality of the match and also investigated the ability of the genetic algorithm (GA) to generate multiple good solutions to SAP through niching techniques. The solutions produced by GA were much better with respect to coverage and fulfilling the objectives of SAP when compared to Gale-Shapley algorithm. But due to the scalarization of the objective functions (to convert multiobjective SAP to single objective problem), the impact of a change in weight vector on the solution is unpredictable thus this enforces the algorithm to run several times with different parameters settings so as to get a broad view of the solution space. Thus diversity (coverage) of the solutions is highly dependent upon the ability of the algorithm to find Pareto optimal solutions as well as the proper selection of parameters choosed by the user.

In 2007 Dasgupta and Garrett, to overcome the above problems, extended the earlier work to incorporate directly each objective into the optimization problem [9]. Hence, two well known multiobjective evolutionary algorithms (MOEAs) NSGA-II and SPEA2 were implemented to solve the multiobjective SAP but the results produced by these algorithms lacks diversity when compared to CHC a single objective genetic algorithm. Thus, a local search operator was constructed and integrated into the MOEAs in such a way as to emphasize the importance of finding diverse solutions across the Pareto set approximation.

The studies of the last two years based on NSGA-II may suffer a lack due to in its basic form, the algorithm is not well suited for the handling of a larger number of objectives (considerably larger than two or three). The main reason for this is the decreasing probability of having Pareto-dominated solutions in the initial external population. To overcome this problem, substitute distance assignment schemes was proposed by Köppen and Yoshida that can replace the crowding distance assignment, which is normally used in NSGA-II [15].

In 2009 A. Kaushal studied an extension of the earlier work [13], in order to determine the incidence of other methods when it was incorporated in MOEA i.e. informed initialized solutions included in the initial population of MOEA, this has shown additional improvements in performance. That project also investigated the effect of these informed solutions onto the MOEA while keeping the population size fixed and varying the informed solutions into the initial population of the MOEA.

The last decade have seen the development of efficient MOEAs but this are dependant of large number of iterations to be effective. These methods are often run with the goal of approximating the whole Pareto Front and most MOEAs are designed to do this on problems of arbitrary parameter space and objective space dimension. Recently researchers have shown empirically that, especially dominance ranking based, MOEAs perform poorly when the number of objectives is more than three [14].

2.1 Sailor Assignment Problem (SAP)

The U.S. Navy Personnel Department faces the problem to find a set of assignments of sailors to jobs which keeps the sailors happy and maintains fleet readiness while minimizing the cost of implementing those as-
signments but many factors go into determining a good set of assignments (objectives such as maximizing training score, minimizing permanent change of station costs, maximizing sailor and commander preferences). However, if the assignment is overly focused on costs, it may be that the sailors are unhappy with their assigned jobs. This could lead to a decrease in morale, followed by decreased retention rates and other problems [10; 16].

In fact, at any given time there is a population of sailors to be reassigned to available jobs, approximately more than 300,000 sailors serve in the Navy and 120,000 are reassigned each year [5]. If in average, there are 300 detailers and the reassign time window has 2 weeks to accomplish the task we can think that 120,000 sailors are distributed into 23 periods (56 weeks per year divided by 2) and managed by about 300 detailers, that means 18 sailors to assign in 2 weeks by detailer 

\[
\frac{120,000}{(56/2)/300},
\]

but the truth is that depending on the season, there are differing numbers of sailors to be assigned; in a time of low peak activity there are usually some hundreds of sailors to be assigned (4 in average per detailer), compared to possibly ten thousand sailors during periods of high activity (33 in average per detailer).

Formally, the SAP may be defined as:

\[
\text{Optimize } \sum_{i=1}^{N} \sum_{j=1}^{M} F_{i,j}d_{i,j}
\]

• subject to the constraints:

\[
\sum_{i=1}^{N} d_{i,j} \leq 1 \quad \forall j \in \{1,2,...M\}
\]

\[
\sum_{j=1}^{M} d_{i,j} \leq 1 \quad \forall i \in \{1,2,...N\}
\]

\[
d_{i,j} \in \{0,1\} \quad \forall i \in \{1,2,...N\} \text{ and } j \in \{1,2,...M\}
\]

where \(F_{i,j}\) denotes the fitness of assigning sailor \(i\) to job \(j\) and \(d_{i,j} = 1\) when the Sailor \(i\) is assigned to job \(j\) and 0 otherwise.

The fitness measure \(F_{i,j}\) encapsulates all information relevant to determining the desirability of the match. SAP is a problem with multiple objectives represented in an “objective vector”:

\[
\text{Optimize } F(x) = \begin{bmatrix}
\max f_1(x) \\
\min f_2(x) \\
\max f_3(x) \\
\max f_4(x)
\end{bmatrix} = \begin{bmatrix}
\max TS(x) \\
\min PCS(x) \\
\max SR(x) \\
\max CR(x)
\end{bmatrix} = \min \begin{bmatrix}
-TS(x) \\
PCS(x) \\
-SR(x) \\
-CR(x)
\end{bmatrix}
\]

The constraints on \(d_{i,j}\) ensure that at most one job will be assigned to any sailor and that no job is assigned to multiple sailors. Note that both constraints are inequalities thus allowing for the possibility that a given sailor is not assigned a job.

### 2.2 Pareto Front

Multi-objective optimization goal (unlike Single-Objective optimization that looks for a unique solution: the optimum value for single-input or multiple-input a single output problems) is obtain a unique solution that
dominates any other, or obtain a set of optimal solutions which are non-dominated among them: the Pareto Optimal Set (POS), its corresponding values in the objective space conform the Pareto Front (PF) for the problem, e.g.: If we have a minimization problem with two objectives and the three solutions \( a, b \) and \( c \): \( F(a) = \{3, 8\}, F(b) = \{5, 9\} \) and \( F(c) = \{6, 8\} \), \( a \) is in POS and PF = \( \{F(a)\} \). Formally [14]:

\[
x \prec y \iff \forall i \in 1..d, x_i \leq y_i \land \exists j \in 1..d, x_j < y_j
\]

where \( x \) and \( y \) are \( d \)-dimensional objective vectors, and

\[
\{y \in Y \mid \neg \exists x \in Y, x \prec y\}
\]

is the set called the Pareto Front.

---

**Figure 2.1** “Example of a Pareto front. The boxed points represent feasible choices, and smaller values are preferred to larger ones. Point C is not on the Pareto Frontier because it is dominated by both point A and point B. Points A and B are not strictly dominated by any other, and hence do lie on the frontier.” from Wikipedia. (accessed 10 Jul 2009) Available from: http://en.wikipedia.org/wiki/Pareto
efficiency

---

### 2.3 OnLine / OffLine assignment problem

The online version of a matching problem is that which the elements of a set, in our case jobs, are known in advance and the elements of the another set, the sailors, arrive online and must be matched immediately (if it is possible). The offline matching problem, where the elements of both sets are known, is one of the most fundamental algorithmic problems and it has played a central role in the development of the theory of classical matching algorithms[17].

The above definitions are important because currently the SAP operates like an online market (see Chapter 4), but it could be converted into a offline market where all the users states her intents to the system by listing their wishes in a preferences list, and the system notifies them about a possible assignment after the
process of look for an optimal matching having in mind all the applicants.

In the human approach: this process could be considered Online for detailers if the number of sailors and jobs is big (this could be related with How many digits can people remember) to take into account all the values of each possible assignment even if they were working with aggregated values [21]. On the other hand this problem could be Offline if the number of possible assignments is human tractable, but this varies as a human condition.

2.4 Linear Assignment Problem

In the Linear Assignment Problem (LAP) the number of agents and number of tasks are equal and every agent can perform every task but only one task can be assigned to each agent and vice versa. In most cases, there are only one goal to be optimized and usually called cost, then the problem is equivalent to the problem of finding an optimum weight vertex matching in an $n \times n$ cost-weighted complete bipartite graph, as shown in Figure 2.2.

![Figure 2.2 Linear Assignment Problem.](image)

Formally LAP can be stated as follows: given a set of agents $A = \{a_1, a_2, ..., a_n\}$ and a set with same number of tasks $T = \{t_1, t_2, ..., t_n\}$ and the cost function $C : A \times T \rightarrow \mathbb{R}$. Find a bijection (matching) $m : A \rightarrow T$ such that the cost function:

$$\sum_{a \in A} C(a, m(a))$$

is minimized or maximized. Usually the cost function is also viewed as square real-valued matrix $C$ with elements $C_{ij} = C(a_i, t_j)$.

This problem can be expressed as a integer linear program\(^4\) with the objective function

$$\sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij} x_{ij}$$

subject to the constraints

- $\sum_{j=1}^{n} x_{ij} = 1 \quad \forall j \in \{1, 2, ..., n\}$
- $\sum_{i=1}^{n} x_{ij} = 1 \quad \forall i \in \{1, 2, ..., n\}$
- $x_{ij} \in \{0, 1\} \quad \forall i, j \in \{1, 2, ..., n\}$

2.4.1 Kuhn Munkres Algorithm for Rectangular Matrices

The Kuhn-Munkres algorithm was published by H. Kuhn [18] in 1955 and later it was improved by J. Munkres in 1957 [22]. This is an algorithm used to solve the linear assignment problem (LAP) in polynomial time ($O(n^3)$).

Following is the compact description of the steps of this algorithm:

**Step 0:** Create an $n \times m$ matrix called the cost matrix in which each element represents the cost of assigning one of $n$ workers to one of $m$ jobs. Rotate the matrix so that there are at least as many columns as rows and let $k = \min(n, m)$.

**Step 1:** For each row of the matrix, find the smallest element and subtract it from every element in its row. Go to Step 2.

**Step 2:** Find a zero ($Z$) in the resulting matrix. If there is no starred zero in its row or column, star $Z$. Repeat for each element in the matrix. Go to Step 3.

**Step 3:** Cover each column containing a starred zero. If $K$ columns are covered, the starred zeros describe a complete set of unique assignments. In this case, Go to DONE, otherwise, Go to Step 4.

**Step 4:** Find a noncovered zero and prime it. If there is no starred zero in the row containing this primed zero, Go to Step 5. Otherwise, cover this row and uncover the column containing the starred zero. Continue in this manner until there are no uncovered zeros left. Save the smallest uncovered value and Go to Step 6.

**Step 5:** Construct a series of alternating primed and starred zeros as follows. Let $Z_0$ represent the uncovered primed zero found in Step 4. Let $Z_1$ denote the starred zero in the column of $Z_0$ (if any). Let $Z_2$ denote the primed zero in the row of $Z_1$ (there will always be one). Continue until the series terminates at a primed zero that has no starred zero in its column. Unstar each starred zero of the series, star each primed zero of the series, erase all primes and uncover every line in the matrix. Return to Step 3.

**Step 6:** Add the value found in Step 4 to every element of each covered row, and subtract it from every element of each uncovered column. Return to Step 4 without altering any stars, primes, or covered lines.

DONE: Assignment pairs are indicated by the positions of the starred zeros in the cost matrix. If $C(i, j)$ is a starred zero, then the element associated with row $i$ is assigned to the element associated with column $j$.

Some of these descriptions require careful interpretation. In Step 4, for example, the possible situations are, that there is a noncovered zero which get primed and if there is no starred zero in its row the program goes onto Step 5. The other possible way out of Step 4 is that there are no noncovered zeros at all, in which case the program goes to Step 6.

For our purposes is interesting that:

- The algorithm will work even when the minimum values in two or more rows are in the same column.

---


• The algorithm will work even when two or more of the rows contain the same values in the the same order.
• Optimality is guaranteed in Munkres Assignment Algorithm.
• Step 3 is an example of the greedy method. If the minimum values are all in different rows then their positions represent the minimal pairwise assignments.
• Step 5 is an example of the Augmenting Path Algorithm (from the Stable Marriage Problem theory).
• Step 6 is an example of contraint relaxation. It is ”giving up” on a particular cost and raising the constraint by the least amount possible.
• The Munkres assignment algorithm can be implemented as a sparse matrix, but it requires to ensure that the correct (optimal) assignment pairs are active in the sparse cost matrix C.
• Munkres can be extended to rectangular arrays (i.e. more jobs than workers, or more workers than jobs).

2.5 Kuhn Munkres for SAP

When classical Kuhn-Munkres algorithm is applied to SAP there were three problems faced. First, SAP is multi-objective problem whereas the Kuhn-Munkres algorithm is applicable for only single - objective assignment problem as LAP. Second, SAP may not be represented as a complete bipartite graph. The algorithm in its canonical form assumes that any sailor can perform any job. But this assumption could not be true in case of SAP since there are limited number of jobs of a particular sailor depending upon the various factors of the SAP. Finally, sometimes there are no complete matching exists in SAP whereas Kuhn-Munkres algorithm cannot be applied on such problems otherwise it will give the solutions that are even more worst in nature. These above three problems and their former solutions was discussed in [6]:

SAP is Multiobjective

As we know, the Kuhn-Munkres algorithm is used to solve only the single objective problems, and moreover the SAP is the a problem with multiple objectives like Training Score, Permanent Change of Station costs, Commandeer Score and Sailor Score. For resolving this problem, weight vectors are incorporated to make the SAP as single objective problem in such a way that the sum of the weight vectors are 1 i.e. \( w_1 + w_2 + w_3 + w_4 = 1 \) and the goal is to optimize

\[
 w_1 \times TS + w_2 \times PCS + w_3 \times SR + w_4 \times CR
\]

with \( 0 < w_i < 1 \). Even the weight space is recursively subdivided and the new single objective problem is solved for each one of those weight vectors hopping to obtain a diverse set of Pareto optimal solutions.

Sampling deficiencies of the weighted-sum approach

This method suffers from two drawbacks: First, the relationship between the vector of weights and the Pareto curve is such that a uniform spread of weight parameters rarely produces a uniform spread of points on the Pareto set. Often, all the points found are clustered in certain parts of the Pareto set with no point in the interesting “middle part” of the set, thereby providing little insight into the shape of the trade-off curve. The second drawback is that non-convex parts of the Pareto set cannot be obtained by minimizing convex combinations of the objectives (note though that non-convex Pareto sets are seldom found in actual applications)[4; 7].
Figure 2.3  Figure(a) represents complete bipartite graph figure (b) shows incomplete graph

<table>
<thead>
<tr>
<th>Row</th>
<th>Col</th>
<th>Row</th>
<th>Col</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>m</td>
<td>1</td>
<td>k</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>m</td>
<td>2</td>
<td>j</td>
</tr>
<tr>
<td>n</td>
<td>1</td>
<td>n</td>
<td>1</td>
</tr>
<tr>
<td>n</td>
<td>m</td>
<td>n</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 2.4  Sparse Matrix Representation

**SAP may not be represented as a complete bipartite graph**

As discussed above, the Kuhn-Munkres algorithm works only for the problem that can be represented as bipartite graph and having a matrix with all finite values as shown in Figure 2.3a. Whereas the graph for SAP is incomplete bipartite graph the intuitive strategy is to assign the nonexistent edges infinite cost, Figure 2.3b, thus preventing the algorithm from choosing these infeasible edges. As it is usual that there are different number of jobs than number of sailors and there are only few jobs that a sailor can perform. So its useless to store such a huge matrix for Kuhn-Munkres algorithms’ operations even this will be helpful in reducing the time complexity of the Kuhn-Munkres algorithm. For doing so sparse matrix came into picture which will contain only feasible combinations of the sailors and jobs. Thus Sparse Matrix Type in Octave (high-level language) was used to store the data. Figure 2.4 shows the resulting representation.

The values in one of these lists (Sparse Matrix) contains the relevant information like training score, permanent change of station cost, commandeer preference and sailor preference. There is one more essential list with values called as “Reduced Rating” which is obtained by the multiplication of the weight vectors by the values of each list. Thus, by storing only feasible sailor/job combinations, there are huge decrements in time and storage complexity for this Kuhn-Munkres algorithm.

**Incomplete matching**

Incomplete matching is a lack of the Classical KM algorithm, this issue is solved with the use of the Kuhn Munkres Algorithm for Rectangular Matrices because it can be applied to rectangular arrays (i.e. more jobs than workers, or more workers than jobs).
Chapter 3

Analysis and subdivision of the SAP and SAP instance generator

In order to evaluate the benefits and shortfalls of the KM Perfect Solver incorporating more realistic/representative data to model the problem of the U.S. Navy is a need to analyse the Personnel Distribution Process. This chapter describes briefly that process and explains the improvements to the new SAP instance generator.

3.1 The Personnel Distribution Process

This section will only highlight the key aspects of the personnel distribution process. A more complete (but out of date) description of the personnel distribution process is given in “Characterizing Sailor and Command Enlisted Placement and Assignment Preferences” (Butler and Molina, 2002).

Navy Personnel Command is responsible for the allocation sub-process. Before allocation can be done, the total available personnel (or personnel inventory) must first be separated into distributable inventory and non-distributable inventory. The output of the allocation sub-process is a detailed document showing the prioritized allocation of distributable inventory to the various Manning Control Authorities down to the individual commands: The Navy Manning Plan. It guides the subsequent sub-processes of placement and assignment by specifying the number and characteristics of the sailor (Rate, Rating and Navy Enlisted Classification) that each command will get by indicating if the billets are Priority 1, 2 or 3 or no priority.

The Enlisted Placement Management Center is responsible for the placement sub-process, this acts as the command advocate for the E-5 to E-9 sailors and strives to achieve that the right person, with the proper occupational skills, occupies the right billet on time. The placement officers use the Navy Manning Plan as a reference document to communicate the commands billet requirements to the detailers. They are in charge of a set of ratings and Navy Enlisted Classifications (NECs) and negotiate with detailers who are more focused on a specific rating or NEC [11].

Each month, when a detailer receives the list of jobs, there are also sailors who need to be assigned to jobs during that month. The assignment sub-process repeats every month in what is known as the requisition cycle. This process is sequential: the detailer will match sailors with the necessary skill sets to the requisitions, once a job is filled, the detailer moves on to the next job. How many jobs and sailors a detailer must deal with varies from month to month and depends on the size of the rating community. If a rating community (like
electronic technician) has 15,000 members and 19 detailers, following the math for computing the number of sailors to be assigned in 2.1, we have \(((15,000/(12))/19)\): 66 electronic technicians aprox. to be assigned per detailer\(^1\).

The time to make a match can vary from a minute or two to a couple of hours. However, it would never be a couple of hours at one time. Rather, it could be a situation where a detailer has a high priority job to fill and a very good candidate that is really interested in taking another job. The detailer would then need to sell the sailor on the idea of accepting the high priority job and the sailor might need to talk to their spouse and call the detailer back. He or she might also need a couple more phone calls from the detailer to convince them to take the job, etc.

Detailers do have a basic scoring rule that looks at the rate, rating, nec, and permanent change of station (PCS) cost for the move to the new job. Their score, or "fit", is a key determinant in deciding between job candidates.

Restarting requisition cycle: Sailors and billets not matched in the cycle are rolled over to the next cycle to be considered again.

An additional consideration is: In order to be assigned a job, sailors have to meet certain skill requirements. A sailor may already have the skills required for direct assignment or may need to attend training to obtain the prerequisite skills. It is generally preferable to choose a sailor for a billet who already possesses the necessary skills, because there are costs associated with the training class itself and with travel to and from the class. Of course, a benefit of training is that the sailor is now qualified for more jobs [19].

### 3.1.1 Analysis and subdivision of the SAP

As we can see, the subdivision of the sets of sailors is done by the own process that the Navy has been performing. The numbers of sailors and jobs handled in previous studies, between hundreds and thousands, is too big compared to the real amount. Large SAP instances take long times to be solved by KM for SAP algorithm (300 sailors with 10 preferences and 400 jobs could take 4 hours, in average, for a single one weight vector, this varies with singular properties of the instance). On the other hand for smaller instances can be solved by the KM for SAP scanning a big number of combinations of values for the weight vectors (20 sailors with 10 preferences and 30 jobs trying 10625 w.vectors take 25 minutes in average).

### 3.2 Problem instance generator

Formerly, due to unavailability of real sailors and jobs data, a problem generator was developed to allow testing on a variety of problem instances of different sizes. These simulated instances contain values for all the objectives along with the information about the jobs for each sailor that he/she can perform but the most of the values was computed with a Normal Distribution, that is not truly corresponding with the real data because some values like distance between stations can not be handled in that way, the number of preferences was also computed with the distribution parameters and do not have strict bounds.

The new SAP instance generator adds some features like bounds for preferences, different options for distributions and its parameters, and it computes the distances between stations reading a file with the coordinates of each station (using the Great Circle Distances formula), this is useful to calculate the Permanent

\(^1\)Special thanks to James Simien at the Navy Personnel Research, Studies, and Technology (NPRST/BUPERS-12)
Change of Station cost taking in mind the station in which a sailor is and the station where he wants to get a new job. A GUI was also integrated, Figure 3.1.

This generator models different possibilities of the distribution of sailors and jobs present in real situations. Each synthetically generated sailor has approximately the same number of eligible jobs and applies for roughly the same number as one could expect to occur in reality according the parameters set by the user.

The priority of the jobs was not implemented in this instance generator (even in the previous one) then the priority of jobs is the same for all of them.

For more details see *The Model*.

![Figure 3.1](media/disk/SAPFiles/DeskCodes.csv)  
![Figure 3.1](media/disk/SAPFiles/PayGrades.csv)  
![Figure 3.1](media/disk/SAPFiles/Bases3.csv)  

**Figure 3.1** GUI for the new SAP instance generator.
Chapter 4

Solutions simulating selected strategies from the distributed process done by human detailors and KM for SAP

The detailer's role is really that of sailor advocate. Currently, there are 12 month-long “windows” during a year in which a sailor is matched to a job. The matching is performed by the detailors. What the sailors are matched to is called a requisition (In our model this as being a job). The requisition is assigned a priority and contains some necessary information like required skills, paygrade, rating, and start-date. There are approximately 300 detailors that accommodate the assignment needs of approximately 330,000 active-duty personnel. Detailors are assigned by a designation - desk code. Requisitions (jobs) are distributed to the detailers by desk code. These desk codes roughly correspond to ratings (ex. Sonar tech, radar operator, air traffic controller, etc.). Sailors are also assigned by desk code to detailors. So, each detailor has a list of jobs and sailors to match that are only a fraction of the total population of each. Detailors then sequentially assign sailors to jobs with the primary considerations being paygrade, rating, skills match, and whether the sailor can arrive on time. The only budgetary aspect to this is actually a very big one - Permanent Change of Station (PCS) dollars. The detailing shop is given an annual PCS budget of several hundred million dollars to move people from one duty station to another. When the money runs out - there are no more moves.

The goal of the personnel distribution process is to get the right sailor with the right training to the right billet at the right time, or R4 as it is better known [11].

In a most general model we can think about this problem as a market: In an online exchange market a transaction happens at the same time as the user declares her intent to acquire the item. However, in an offline market, the user states her intent to the system by listing her wishes in a wishlist, and the system notifies her about a possible exchange after a process considering all the offers through offline message.

4.1 The Model

Figure 4.1 presents the General Architecture for Simulating the SAP.

We will follow the representation of the instance of SAP as a list of sailors that have their preferred jobs (its corresponding values for the TS, PCS, CR and SR in 4 matrices, a preferred job has positive values in all the matrices, if any value of a preferred job is zero it is represented with a very small decimal), this information

---

1 Special thanks to James Simien at the Navy Personnel Research, Studies, and Technology (NPRST/BUPERS-12)
will be used as it is in the Random Online Assignment for the MonteCarlo method simulating an Online market, it means: Given a permutation of the list of sailors they will be assigned sequentially to one job when it is possible, if a job is not available but the sailors has another options one of them will be randomly selected to try the match, if all the jobs preferred by the sailors are occupied this sailor will remain unmatched. A similar protocol will be follow for the Sequential Greedy method with the difference that this will try to assign the job with better aggregated value for each sailor when it is possible. A Aggregate Dimension Function Matrix is required to apply the KM and Greedy algorithms to solve the problem, this is obtained by the multiplication of evenly generated weight vectors for 4 objectives, Algorithm 1.

The changeToMinimize function converts the values of a matrix set for maximization into values set for minimization but keeping the zero values. The zero values will represent infinite (not a real value of 0) and are fundamental in order to use sparse matrix capabilities of the Octave language. This conversion is necessary because the original SAP merge maximization and minimization objectives, and to use the KM or Sequential Greedy algorithms, that implies the weighted sum to compose a unique objective function, all the objectives must be pointing to the same target. The 0.01 factor is to make the difference between a “zero” in a value of a preferred job and the representation of infinite.

The spfun(@(x)f(x),m) function compute f(x) for the non-zero values of m. This results in a sparse matrix with the same structure as m.

Every assignment method (KM, SMGZM, SEQ.GREEDY, SEQ.RANDOM) is composed by 3 subprocess. This subprocess prepare the data (preprocess(aggregationMatrix)), perform the assignment (SEQassignmentgreedy(nnam) or SEQassignmentrandom(nnam) or step1(nnam)) for KM, where nnam is a negative normalized auxiliar matrix product of the preprocess) and finally prepares the data to present it posprocess(auxAssignment, removedColslist).

The weight vectors depend on the number of steps between the lowest value and the highest value possible
Algorithm 1 evenly generated weight vectors for 4 objectives

1: procedure SAP_KM_SeqGreedy(nStep, threshold)
2: \( T S_{\text{min}} \leftarrow \text{changeToMinimize}(T S) \)
3: \( SR_{\text{min}} \leftarrow \text{changeToMinimize}(SR) \)
4: \( CR_{\text{min}} \leftarrow \text{changeToMinimize}(CR) \)
5: for \( w_1 = 0 \) to \( n\text{Step} \) do
6: \hspace{1em} for \( w_2 = 0 \) to \( n\text{Step} - w_1 \) do
7: \hspace{2em} for \( w_3 = 0 \) to \( n\text{Step} - (w_1 + w_2) \) do
8: \hspace{3em} \( w_4 \leftarrow n\text{Step} - (w_1 + w_2 + w_3) \)
9: \hspace{3em} if \((w_1 = 0) \lor (w_2 = 0) \lor (w_3 = 0) \lor (w_4 = 0)\) then
10: \hspace{4em} forces the minimum consideration of every dimension:
11: \hspace{4em} weightV \leftarrow \text{noZeroValue}(w_1, w_2, w_3, w_4, n\text{Step}, \text{threshold})
12: \hspace{3em} else
13: \hspace{4em} weightV \leftarrow [w_1 \div n\text{Step}, w_2 \div n\text{Step}, w_3 \div n\text{Step}, w_4 \div n\text{Step}]
14: \hspace{2em} end if
15: \hspace{1em} aggregationMatrix \leftarrow T S_{\text{min}} \times \text{weightV}[1] + PCS \times \text{weightV}[2] + SR_{\text{min}} \times \text{weightV}[3] + CR_{\text{min}} \times \text{weightV}[4]
16: \hspace{1em} assignmentKM \leftarrow \text{KM}_{\text{SMGZM}}(aggregationMatrix)
17: \hspace{1em} assignmentSG \leftarrow \text{SEQ}_G\text{REEDY}(aggregationMatrix)
18: end for
19: end for
20: end procedure

22: function changeToMinimize(matrix) : realMatrix
23: \hspace{1em} if any(matrix[:,] < 0) then
24: \hspace{2em} error(“All matrix elements have to be non-negative.”)
25: \hspace{1em} end if
26: \hspace{1em} maximum \leftarrow \max(\max(matrix))
27: \hspace{1em} matrix \leftarrow matrix \div (maximum + (maximum \times 0.01))
28: \hspace{1em} changedMatrix \leftarrow \text{spfun}(\emptyset(x)1 - x, \text{matrix})
29: \hspace{1em} return \text{changedMatrix}
30: end function

31: function noZeroValue(w_1, w_2, w_3, w_4, nStep, threshold) : realVector
32: \hspace{1em} w_1 \leftarrow w_1 + \text{threshold}
33: \hspace{1em} w_2 \leftarrow w_2 + \text{threshold}
34: \hspace{1em} w_3 \leftarrow w_3 + \text{threshold}
35: \hspace{1em} w_4 \leftarrow w_4 + \text{threshold}
36: \hspace{1em} nStep \leftarrow nStep + 4 \times \text{threshold}
37: \hspace{1em} noZeroWeightV \leftarrow [w_1 \div n\text{Step}, w_2 \div n\text{Step}, w_3 \div n\text{Step}, w_4 \div n\text{Step}]
38: \hspace{1em} return noZeroWeightV
39: end function


(0...1), wVectors is a simplified version of the Algorithm 1 to show the weight vectors generation, e.g.: for
the nStep 1 it will produce 4 weight vectors (the threshold .00005 is intended to replace weight vectors that
contains zeros because that vectors do not produce solutions in the pareto front in the most of the cases):

octave:2> wVectors(1, .00005)
4.9990e-05 4.9990e-05 4.9990e-05 9.9985e-01
4.9990e-05 4.9990e-05 9.9985e-01 4.9990e-05
4.9990e-05 9.9985e-01 4.9990e-05 4.9990e-05
9.9985e-01 4.9990e-05 4.9990e-05 4.9990e-05

e.g.: for the nStep 2 it will produce 10 weight vectors (the threshold .00005 is intended to replace weight
vectors that contains zeros because that vectors do not produce solutions in the pareto front in the most of the
cases):

octave:3> wVectors(2, .00005)
2.4998e-05 2.4998e-05 2.4998e-05 9.9993e-01
2.4998e-05 2.4998e-05 4.9998e-01 4.9998e-01
2.4998e-05 2.4998e-05 9.9993e-01 2.4998e-05
2.4998e-05 4.9998e-01 2.4998e-05 4.9998e-01
2.4998e-05 4.9998e-01 4.9998e-01 2.4998e-05
2.4998e-05 9.9993e-01 2.4998e-05 2.4998e-05
4.9998e-01 2.4998e-05 2.4998e-05 4.9998e-01
4.9998e-01 2.4998e-05 4.9998e-01 2.4998e-05
4.9998e-01 4.9998e-01 2.4998e-05 2.4998e-05
9.9993e-01 2.4998e-05 2.4998e-05 2.4998e-05

And so on with nStep = n and the number of weight vectors corresponding to the formula:

\[(n + 1) \times ((n + 1) + 1) \times ((n + 1) + 2)/3!\]

The number of vectors generated for the first 20 steps are tabulated in the Table 4.1.

The KM algorithm uses the Aggregate Matrix data as input to compute an optimal solution for each
weight vector. Multiple weight vectors are used with the aim of reach a representation of the Pareto Front.
The Sequential Greedy algorithm uses the Aggregate Matrix data too but the resulting assignments have no
guaranty of be an optimal solution.

Formally, the search space of the algorithms is the set of all feasible assignments:

\[\mathcal{A}^* = \{a \mid a : S \rightarrow J, \text{ a feasible injection}\}\]

an injection is a feasible assignment if:

- \(a(s_i) \in L_i\) and
- \(a(s_i) \neq a(s_j)\) for \(i \neq j\).

An assignment can be represented in the form of a vector:

\(a = (j_1, j_2, j_3, \ldots, j_n)\)
Table 4.1 Number of Weight Vectors by number of steps

<table>
<thead>
<tr>
<th>steps (nStep)</th>
<th>Number of weight vectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>35</td>
</tr>
<tr>
<td>5</td>
<td>56</td>
</tr>
<tr>
<td>6</td>
<td>84</td>
</tr>
<tr>
<td>7</td>
<td>120</td>
</tr>
<tr>
<td>8</td>
<td>165</td>
</tr>
<tr>
<td>9</td>
<td>220</td>
</tr>
<tr>
<td>10</td>
<td>286</td>
</tr>
<tr>
<td>11</td>
<td>364</td>
</tr>
<tr>
<td>12</td>
<td>455</td>
</tr>
<tr>
<td>13</td>
<td>560</td>
</tr>
<tr>
<td>14</td>
<td>680</td>
</tr>
<tr>
<td>15</td>
<td>816</td>
</tr>
<tr>
<td>16</td>
<td>969</td>
</tr>
<tr>
<td>17</td>
<td>1140</td>
</tr>
<tr>
<td>18</td>
<td>1330</td>
</tr>
<tr>
<td>19</td>
<td>1540</td>
</tr>
<tr>
<td>20</td>
<td>1771</td>
</tr>
</tbody>
</table>

where $j_i = a(s_i)$ is the job assigned to $s_i$.

Here, $\mathcal{L}_i$ represents the list of jobs that sailor $s_i$ is qualified and applied for.

4.1.1 Distribution

Due to the characteristics of the Personnel Distribution process, the distribution of the elements and sets is already done, anyway it is feasible using labels and Mathematic filters to achieve the same results. This configures many problems (at least equal to the number of detailers for a requisition cycle) that could be solved by any of the methods that we refer, it is easily simulated using multi-thread techniques to run several instances of the KM, Random or Greedy Online algorithms implemented in Octave language integrating it with Java, joPas$^2$ is an API that allows the user to create programs in Java with the advantage of implementing all the mathematical part in Octave.

4.2 Sequential Greedy Assignment

As we can see in the description about the assignment sub-process the detailers perform a sequential matching based on the arrive order of the sailors (calls) and trying to keep an idea of which could be the better assignment to him, this seems to be a difficult task but we will represent it in the most simplified way:

In our simulation a detailer has a list of jobs, he knows which sailors have applied for that job but does not know which sailor will call or arrive (or the order of the calls), then he can adopt the policy of give the job that represents a singular optimum.

---

$^2$http://jopas.sourceforge.net/index.html
Sequential Greedy assignment favors the applicants(sailors) and it acts similar to the Gale and Shapley Algorithm considering the order of arrive as preference of the proposed (girls, jobs) elements.

**Algorithm 2** Sequential Greedy Assignment (minimizing between 0 and -1)

1. `permuteRows(aggregationMatrix)`
2. `for i = 1 to nOfRows do`
3. `if nonZeroElements(aggregationMatrix[i,:]) = 0 then`
4. `assignment[i] ← 0`
5. `else`
6. `The first minimum`
7. `assignment[i] ← find(aggregationMatrix[i,:] = min(aggregationMatrix[i,:]))[1]`
8. `aggregationMatrix[:,assignment[i]] ← 0`
9. `end if`
10. `end for`

### 4.3 Montecarlo method for SAP

#### 4.3.1 Random Online Assignment

Consider the following scenario: It could be a situation where a detailer has a job to fill and a very good candidate that is really interested in taking another job AND this candidate does not change his decision. The detailer would then need to rearrange all the rest of possibles matches if he was thinking in “I love it when a plan comes together”, but not. This maybe is not so serious if only occurs few times (Anyway could be a very demanding task), but if that is not the case, it occurs several times, the assignment process could be considered a Online Random Assignment.

J. Pais in [23] says that we can assume that each individuals has preferences over the other side of the market and the prospect of being unmatched; however, they are not compelled to behave in a straightforward manner, according to these true preferences. Instead, agents are confronted with a game in which they act in what they perceive to be their own best interest.

Additionally, as a living system, human perception and judgment are subject to change when the information inputs or psychological states of the decision maker change [16; 21].

Consider this also: A sailor its very interested in any of the possibles jobs that he can perform, when he calls the detailer he is only interested in get a job does not matter which it is. If this is the logic of some of the candidates it could hurt the assignment process.

In some cases the above situations generates an increase in the number of applicants submitting only one choice [1].

For the above reasons we consider interesting and feasible the study of the Online Random Assignment as a real life situation but with the hypothesis that it is the worst method and having a reduced probability of configurate solutions in the Pareto Front.

The Algorithm 3 given a permutation of the list of sailors, it assigns sequentially the sailor to a random
job of her list when it is possible, if the job is not available but the sailors has another options one of them will be randomly selected again to try the match, if all the jobs prefered by the sailors are occupied this sailor will remain unmatched and pass to the next sailor.

**Algorithm 3 Online Random Assignment**

1: `permuteRows(Matrix)`
2: **for** `i = 1` to `nOfRows` **do**
3:     `posOfPossibles ← find(Matrix[i,:] < 0)`
4:     `Cols ← size(posOfPossibles)`
5:     **if** `Cols = 0` **then**
6:         `assignment(i) ← 0`
7:     **else**
8:         `pos ← ceil(rand * Cols)`
9:         **if** `pos = 0` **then**
10:            `pos ← 1`
11:     **end if**
12:     `sel ← Matrix[i,posOfPossibles[pos]]`
13:     `assignment[i] = find(Matrix[i,:] == sel)[1]`
14:     `Matrix[:,assignment[i]] ← 0`
15: **end if**
16: **end for**
Chapter 5

Experimentation

For 18 different sets of characteristics of the SAP, Table 5.1, were created 10 instances of the problem, considering 6 bases and with a Uniform distribution for Sailor’s Bases and Job’s Bases and random selection without replacement of Jobs for Sailors. Each instance was solved with the KM, Greedy and Random Online sequential methods using from 1 to 20 steps (for the methods that requires weight vectors), in the Online Random method the number of steps only has the effect of to give an additional chance per each vector. Thus, we obtained 60 file-solutions for each instance, (3600 per method) 10800 in total. The file-solutions contains 4, 10, 20, 35, ... 1771, potentially different assignments according to the number of steps, then there is:

\[
\sum_{n=1}^{20} (n+1) \times ((n+1) + 1) \times ((n+1) + 2)/3! = 10625 \text{ potential solutions per method per instance.}
\]

<table>
<thead>
<tr>
<th>Table 5.1</th>
<th>Characteristics of the Problem instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of Sailors</td>
<td>number of Jobs</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>
5.1 Summaries

The Figures 5.1, 5.3, 5.5, 5.6, 5.7, 5.8 shows the results for each set of characteristics, having a maximum of 106250 potential solutions per method.

The Figure 5.1 shows the non-dominated solutions (by KM) found by Random Online assignment in contrast with the solutions provided by KM. For the most of the problem instances the Random Online found a considerable number of non-dominated solutions that could suggest the KM algorithm is not reaching some regions of the Pareto Front or its number of weight vectors is not enough to find that solutions (or a dominant solution).

Eventhough, Random Online does not work uniformly over all the instances and it is possible it does not

![Figure 5.1](image1.png)  
**Figure 5.1** Random Online non-dominated solutions and KM solutions by sets of characteristics.

![Figure 5.2](image2.png)  
**Figure 5.2** Random Online non-dominated solutions and KM solutions by 10 instances for 10 sailors, 20 jobs, and 20 preferences \(s_{10j20p20}\), along the abscissa direction are enumerated the number of the problem instance.
work at all for some of them like a specific instance of a set of characteristics, i.e., instance 9 of the set s10j20p20, in Figure 5.2.

The Greedy Online algorithm presents greater number of non-dominated solutions than KM and Random Online algorithms in most of the instances, nevertheless it does not work uniformly too: for some set of instances the number of non-dominated solutions found is less than the found by Random Online and KM algorithms, i.e., sets s5j10p10, s20j10p10 and s10j20p20, shown in the Figures 5.3 and 5.4.

Figure 5.3  Greedy Online non-dominated solutions and KM solutions by sets of characteristics.

Figure 5.4  Greedy Online non-dominated solutions and KM solutions by 10 instances for 10 sailors, 20 jobs, and 20 preferences (s10j20p20), along the abscissa direction are enumerated the number of the problem instance.
KM solutions are always non-dominated and optimal but Random and Greedy Online algorithms do not guarantee non-dominated solutions. Because that the solutions of the two Online algorithms was compared by a Dominance-Comparator with the KM’s solutions. Its important to say that non-dominance is not synonymous of optimally; but the non-dominated solutions found in a well explored solution space has a high probability to be optimal. The number of dominated solutions increases with the number of preferences allowed. For the first problem instances in the Figures 5.5 and 5.6 assignments with only 1 preference by sailor was studied showing a null or very low number of dominated solutions for Random and Greedy Online solutions. But, for the last problem instances with sailor’s preferences of 10 and 20 a very high number of dominated solutios appear for both Online algorithms meanwhile the number of solutions of KM grows slowly.

Our KM algorithm is finding the solutions with the most number of assignments possibles, it is due that the unmatched places has a greater cost than fill it in the minimization process. That feature is not easily adaptable to the Random and Greedy Online algorithms because its nature online does not permit to have a priori information of a possible blocking assignments\(^1\). When two assignments has different numbers of matches to perform a comparison between them is unfair, because that when the Random or Greedy Online algorithms found solutions with numbers of matches not comparables with KM was discarted. The number of not considered solutions (not comparables to KM solutions) is represented in the Figures 5.7 and 5.8.

The percents of the number of non-dominted solutions, the number of dominated solutios and the number of non-comparable solutions reflect a performance measure of the Online methods if they were applied only \textbf{a single time} in Figures 5.9 and 5.10.

\(^1\)when a sailor with more options is assigned to the unique option of a next sailor, or the next...
Figure 5.5  Random Online non-dominated and dominated solutions by KM solutions.

Figure 5.6  Greedy Online non-dominated and dominated solutions by KM solutions.
Figure 5.7  Random Online non-dominate and non-comparable solutions.

Figure 5.8  Greedy Online non-dominate and non-comparable solutions.
Figure 5.9  Random Online solutions percents.

Figure 5.10  Greedy Online solutions percents.
Chapter 6

Conclusions

SAP does not have a clear neighborhood structure, it is difficult to trace optimal solutions since this solutions may or not be located at the neighborhood of any feasible solutions in the search space.

KM alway find an optimal solution but that solution could be the same even if it corresponds to different weight vectors even more in the case of the Pareto Front is not convex. Its diversity of solutions depends on the shape of the Pareto Front.

The simulated behaviors (random/greedy) of the detailers in this work has less than 40% of chance in sum to find non-dominated solutions, but this results are not generalizables because every instance of SAP has a characteristic shape of Pareto Front and the methods in this work are very sensitive about it. The use of distances to evaluate good (near) or bad (far) solutions to KM solutions is not possible because we can have short distances to dominated solutions and long distances to non-dominated solutions found with an Online algorithm, vice versa or both. The distance of the solutions obtained by the human detailors to the Pareto solutions depends on the shape of the Pareto Front and the order of the calls or arrives of the sailors.

In the most of the problem instances a Greedy or Random online method has found more non-dominated solutions than the KM algorithm, that suggest the Pareto Front representing the optimal solutions for the most of the instances is not convex. It is known that the KM algorithm fully solve the SAP only in the case of having a convex Pareto Front but could be necessary to try a huge number of weight vectors.

Due to the scalarization of the objective functions (to convert multiobjective SAP to single objective problem), the impact of a change in weight vector on the solution is unpredictable thus this enforces the KM algorithm to run several times with different set of weight vectors with the aim to get a broad view of the solution space but this is not possible in the most of the cases that we have studied here because the Pareto Front is not convex.

The weighted-sum approach does not affect considerably the Online Greedy algorithm like it does to the KM algorithm, the relationship between the vector of weights and the points found is different due the Online process: this points are not clustered in certain parts of the search space and the solution may contain points in the interesting middle part of the solution set, thereby providing a little more insight into the “shape” of the Pareto Front for most of the instances. Anyway in some cases the probability of reach a non-dominated solution could be so low (depending on the shape of the Pareto Front) then the Online methods do not work at all.
Bibliography


