A Comparison Of Multiobjective Evolutionary Algorithms with Informed Initialization and Kuhn-Munkres Algorithm For The Sailor Assignment Problem

Dipankar Dasgupta, German Hernandez, Deon Garrett, Pavan Kalyan Vejandla,
Aishwarya Kaushal, Ramjee Yerneni
Dept. of Computer Science, University of Memphis
Memphis, TN 38152
{ddasgupt,gjhrnndz,jdgarrtt,pvejandl,kaushal1,ryerneni}@memphis.edu

James Simien
Research Analyst
Navy Personnel Research, Studies, and Technology
Millington, TN 38055
james.simien@navy.mil

ABSTRACT

This paper examines the performance of two multiobjective evolutionary algorithms, NSGA-II and SPEA2, with informed initialization on large instances of United States Navy's Sailor Assignment Problem. The informed initialization includes in the initial population special solutions obtained by an extension of the Kuhn-Munkres algorithm. The Kuhn-Munkres algorithm, a classical algorithm that solves in $O(n^3)$ time instances of the single valued linear assignment problem, is extended here to render it applicable on single objective instances of the sailor assignment problem obtained using weight vectors to scalarize the natural multiobjective formulation. The Kuhn-Munkres extension is also used to provide a performance benchmark for comparison with the evolutionary algorithms.

Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods, and Search**]

General Terms

Algorithms,Management

1. INTRODUCTION

According to the United States Navy's personnel policies, roughly every three years sailors serving on active duty are reassigned to a different job. As a result, at any given time there exists a sizable population of sailors to be reassigned to available jobs. Currently, more than 300,000 sailors serve in the Navy and more than 120,000 are reassigned each year [6]. Sailors must be reassigned in such a way as to satisfy their individual preferences and the needs of the Navy. The Navy's goal is to identify sailor and job matches that maximizes some criterion of desirability and referred to as the Sailor Assignment Problem (SAP).

Currently the Navy employs approximately 200 "detailers" that are responsible for assigning sailors to a particular job [8]. Depending on the season, there are differing numbers of sailors to be assigned; in a time of low peak activity there usually some hundreds of sailors to be assigned, compared to possibly ten thousand sailors during periods of high activity.

In reality, the desirability of a possible assignment is not determined by a single measure since several factors in combination determine what constitutes a suitable match. The Navy would like to maximize the satisfaction of the sailors with the jobs they have been assigned, while also best utilizing the available talent at the least cost.

In previous work [4], genetic algorithms were tested on single objective versions of the sailor assignment problem and compared against results of the Gale-Shapley algorithm that solves in $O(n^2)$ time the stable marriage problem. Also, in [5], it was shown that when NSGA-II [3] or SPEA2 [11] were applied to multiobjective instances of the sailor assignment problem, the results lack diversity. In that work, adequate diversity was achieved by combining the evolutionary algorithms with a rudimentary local search operator.

In this work work we show that similar results may be obtained using only an informed initialization that includes in the initial population some reduced number of special solutions obtained by the application of the Kuhn-Munkres algorithm extension presented here.

2. LINEAR ASSIGNMENT PROBLEM

The matching or assignment problems are one the fundamental classes of combinatorial optimization problems. In its most general form, a matching or assignment problem

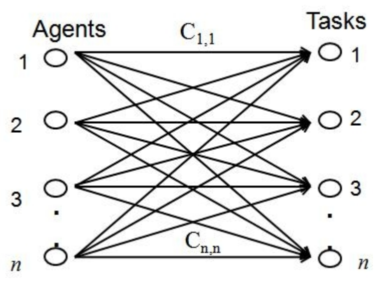


Figure 1: Linear Assignment Problem. c_{ij} **denotes the cost of assigning agent** i **to task** j**.**

can be stated as follows: a number of agents n and a number m of tasks are given, possibly with some restrictions on which agents can perform each particular task. A cost is incurred for each agent performing some task, and the goal is to perform all tasks in such a way that the total cost of the assignment is minimized.

The Linear Assignment Problem (LAP) is the simplest of the assignment problems. In the canonical LAP, the number of agents and tasks is the same, and any agent can be assigned to perform any task. LAP is thus 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 1.

Formally LAP can be formulated 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 $C : A \times T \to \mathbb{R}$. Find a bijection (matching) $m : A \to T$ 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 with the objective function

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

subject to the constraints

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

2.1 The Kuhn-Munkres Algorithm

The Kuhn-Munkres Algorithm, also known as the Hungarian algorithm, is an algorithm that solves LAP instances in polynomial time ($O(n^3)$). It was first published in 1955 by H. Kuhn [7] and later improved by J. Munkres in 1957 [9]. An extension of this algorithm for rectangular matrices was introduced by Bourgeois and Lassalle in 1971 [1]. The extension to rectangular matrices allows the algorithm to operate in situations where the numbers of agents and tasks are unequal. A compact description of the steps of this algorithm, adapted from [2], is given below, and an example is shown in Figure 2.

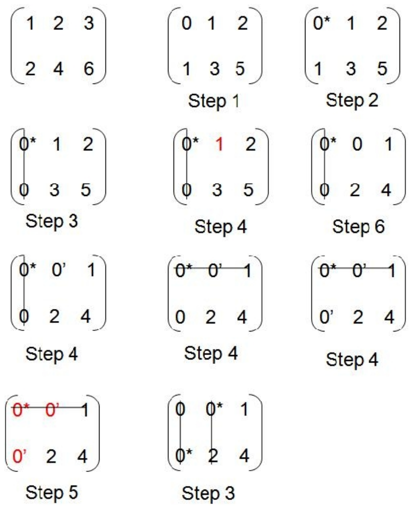


Figure 2: Kuhn-Munkres example

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

Step 2: Find a zero in the resulting matrix. If there is no starred zero in its row or column, star that zero. Repeat for each zero in the matrix.

Step 3: Cover each column containing a starred zero. If n columns are covered, the starred zeros describe a complete set of unique assignments. In this case, stop, otherwise continue with step 4.

Step 4: Find an uncovered 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. Repeat this process until there are no uncovered zeros left. After saving the smallest uncovered value go to Step 6.

Step 5: Construct a path 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. Un-star 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.

3. THE SAILOR ASSIGNMENT PROBLEM

The United States Navy's Sailor Assignment Problem (SAP) exhibits several complications compared to the canonical LAP.

- There are more tasks (jobs) than agents (sailors).

- Not every sailor can perform every possible job. Each sailor has a list of possible of jobs for which he is qualified, and from that list he chooses a subset that he wants to apply for and then ranks them according to his preferences.
- SAP is a multiobjective problem, requiring the simultaneous maximization of the satisfaction of the sailors and commanders, maximization of the training score over each assigned sailor/job combination, and minimization of the cost of relocating the sailor.

Formally SAP can be formulated as follows: Given

- $S = \{s_1, s_2, ...s_n\}$ the set of sailors(agents) ,
- $J = \{j_1, j_2, ...j_m\}$ the set of jobs, with $m > n$,
- $\mathcal{L} = \{L_1, L_2, ..., L_n\}$ the set of lists of of jobs for the sailors, with $L_i = \{j^i_1, j^i_2, ..., j^i_{k_i}\}$ the list of jobs that sailor s_i is qualified and applied for,
- $TS : S \times J \to \mathbb{R}$ for the training score, $PCS : S \times J \to \mathbb{R}$ for permanent change of cost, $SR : S \times J \to \mathbb{R}$ for the sailor ranking, $CR : S \times J \to \mathbb{R}$ for the commander ranking — the four goal functions for each sailor-job pair,

find the set of assignments or matchings $\mathcal{A}^*$ that are Pareto optimal,

$$\mathcal{A}^* = \{a \mid a : S \to J,\ a\ Pareto\ optimal\ injection\}.$$

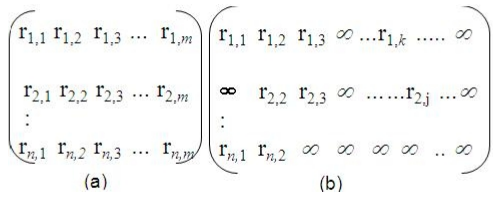
In this case the assignments are injections due to the fact that there can be unassigned sailors.

3.1 Kuhn-Munkres Algorithm Extension

The application of the Kuhn-Munkres algorithm to SAP instances thus has three problems. First, SAP cannot be represented as a complete bipartite graph. The canonical form of the algorithm assumes that any sailor may be assigned to any job. In the SAP, this assumption is not true, and the algorithm must be modified to ensure that only qualified sailors may be assigned to each job. Additionally, the sparseness of the graph can allow for improvements to the efficiency of the algorithm. Second, SAP is a multiobjective problem, whereas the Kuhn-Munkres is defined only on problems exhibiting a single objective function. Finally, the Kuhn-Munkres requires the existence of a complete match. These problems and their solutions are discussed in detailed in following section.

3.1.1 SAP cannot be represented as a complete bipartite graph

As mentioned above, the Kuhn-Munkres Algorithm will only work on a complete bipartite graph that is represented by a matrix with all finite values, as shown in Figure 3. On the other hand, the graph resulting from the SAP is an incomplete bipartite graph and the intuitive strategy is to assign the nonexistent edges infinite cost, thus preventing the algorithm from choosing these infeasible edges. As there are generally only a small number of jobs that a sailor can perform when compared to the total pool of jobs, storing the full matrix is unnecessary. A sparse matrix representation requires only those values corresponding to feasible sailor/job combinations. Figure 4 shows the resulting representation. Two lists are thus maintained – one containing



$$\begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} & \cdots & r_{1,m} \\ r_{2,1} & r_{2,2} & r_{2,3} & \cdots & r_{2,m} \\ \vdots \\ r_{n,1} & r_{n,2} & r_{n,3} & \cdots & r_{n,m} \end{pmatrix} \begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} & \infty \ldots r_{1,k} & \ldots\ldots & \infty \\ \infty & r_{2,2} & r_{2,3} & \infty \ldots\ldots r_{2,j} & \ldots & \infty \\ \vdots \\ r_{n,1} & r_{n,2} & \infty & \infty \;\; \infty \;\; \infty & .. & \infty \end{pmatrix}$$

Figure 3: Figure(a) represents complete bipartite graph figure (b) shows incomplete graph

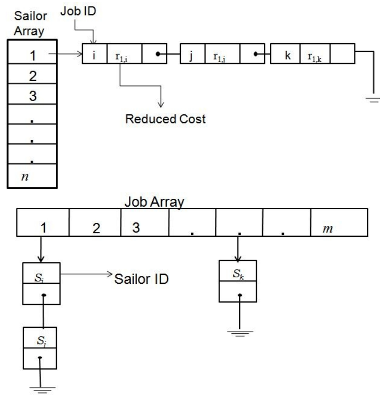


Figure 4: Row and Column Representation of Sparse Matrix

sailors and the information of jobs that he can perform and the other containing jobs and information of sailors who can perform each job.

In addition, Figure 4 demonstrates the representation in which the objective function values are stored. Each value in one of these lists contains the pertinent information regarding the utility of assigning the sailor to the given job. In particular, the values of each of the four objective functions is stored, as well as the "Reduced rating," obtained by multiplying the objective function vector by a given weight vector. By storing only the eligible sailor/job combinations, the computational and storage requirements of the algorithm may be greatly decreased.

3.1.2 SAP is Multiobjective

The Kuhn-Munkres Algorithm is defined only for a single objective function, whereas SAP requires the simultaneous consideration of multiple objectives (training score, permanent change of station cost, commander choice, and sailor choice). To resolve this incompatibility, single objective instances of SAP are obtained using weight vectors, with the resulting goal to optimize

$$w_1 \times TS \ + \ w_2 \times PCS \ + \ w_3 \times SR \ + \ w_4 \times PCS$$

with $w_i \in [0, 1]$ and $w_1 + w_2 + w_3 + w_4 = 1$. To obtain a diverse set of Pareto optimal solutions, the single objective problem is solved for each one of the weight vectors obtained via recursively subdividing the weight space, as shown in Figure 5 for two and three objective problems.

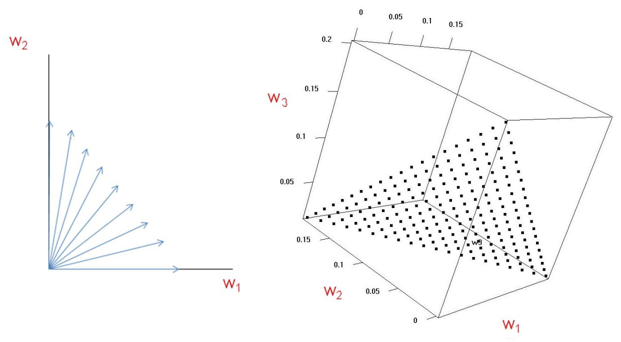


Figure 5: Weight vectors samples

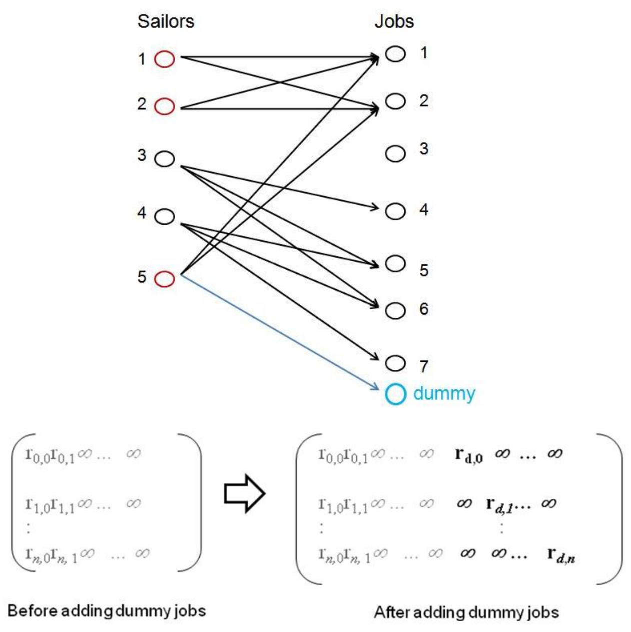


Figure 6: Addition of dummy jobs

3.1.3 Incomplete matching

In SAP instances, we can have the situation in which no feasible solution exists with all sailors successfully matched. To solve this problem, dummy jobs were added, as shown in Figure 6, for each sailor with low Training Score, high PCS cost, low Command Rank and low Sailor Rank, each guaranteed to be worse than any feasible sailor/job match, which can be chosen as a last resort. This allows the algorithm to successfully complete, even when conflicts prevent all sailors from being assigned feasible jobs.

From experimental analysis of SAP instances, it was estimated that the number of jobs that a particular sailor can perform is typically approximated by $\log m$ and that number of sailors that can perform a particular job by $\log n$. Thus this extension of the Kuhn-Munkres algorithm has to update $\log n \times \log m$ elements at most n times and accessing each element takes $logm$ jumps in the lists then time complexity is $O(n \log^2 m \log n)$.

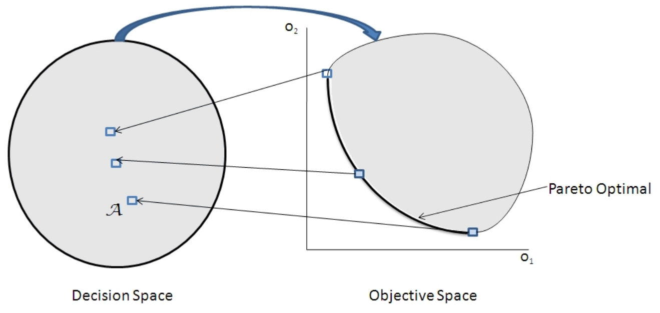


Figure 7: Informed Initialization

4. MOEA WITH INFORMED INITIALIZATION

4.1 Representation and initialization

The search space of this algorithms is the set of all feasible assignments

$$\mathcal{A}^* = \{a \mid a : S \to J,\ a\ feasible\ injection\},$$

a injection is feasible assignment if

- $a(s_i) \in \mathcal{L}_i$ and
- $a(s_i) \neq a(s_j)$ for $i \neq j$.

An assignment can be represented in the form of a vector

$$a = (j_1, j_2, j_3 \ldots.., j_n)$$

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

4.1.1 Informed Initialization

Five nondominated solutions (solutions from weight-vectors [1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,0,1], and [0.25,0.25,0.25,0.25]) were obtained from the Kuhn-Munkres algorithm and fed into the initial populations of NSGA-II. As demonstrated in Figure 7, this provides the MOEA with extreme solutions along the Pareto front, from which the evolutionary operators may work to fill in gaps along the front. The remainder of the population was initialized uniformly at random. The random initialization procedure is described in Algorithm 1.

Algorithm 1 Algorithm for Initialization of the population

```
 1: $P \leftarrow$ size of the population
 2: Let $a_k[\,]$ $\leftarrow$ an array representing the $k_{th}$ assignment(individual) in the population.
 3: Let $F[\,] \leftarrow$ an array having boolean variable which represents wheather the job is free or not
 4: for $i = 1$ to $P$ do
 5:   initialize all variable of $F[\,]$ to true.
 6:   for $j = 1$ to $n$ do
 7:     Let $J \leftarrow$ a free job in $\mathcal{L}_i$
 8:     Mark the job that is not free
 9:     if there is not then
10:       set $J$: = -1
11:     end if
12:   end for
13: end for
```

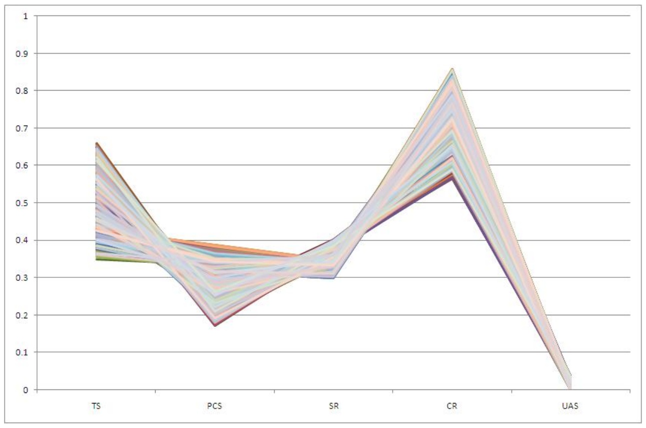


Figure 8: NSGA-II

4.2 Operators

4.2.1 Mutation:

The mutation operator first tries to assign a free job to a sailor if available, and tries to assign the current job of the sailor to an unassigned sailor this we call as *SHIFT*. If not it goes through the list of sailors that can perform the current job and tries to swap the job with some sailor which we call *SWAP*. It does this until a *SWAP* or a *SHIFT*. The procedure of mutation can be found in Algorithm 2.

Algorithm 2 Algorithm for Mutation

```
1: $S \leftarrow$ Pick a sailor at random
2: $J \leftarrow$ Job that sailor $S$ is currently performing
3: repeat
4:   if there is at least one job that a sailor $S$ can do then
5:     if $S$ is unassigned then
6:       assign one of the free jobs to $S$
7:     else
8:       mark $J$ as free
9:       {SHIFT}assign one of the free jobs to $S$
10:    end if
11:  else
12:    {SWAP} Go through the list of sailors that can perform the job $J$
13:    Collect the jobs that sailor $S$ can swap with the sailors that can perform $J$.
14:    Swap job $J$ with any other sailor who can perform $J$
15:  end if
16: until a SWAP or SHIFT or it can not perform mutation
```

4.2.2 Crossover:

Unlike [4, 5], the crossover operator used here does not require a repair operator, as it always produces an assignment that is feasible. The crossover operator picks a position at random and swaps the jobs of two parents at that position and checks for a possible conflict (repetition of a job), we call this a *CYCLE*, not to be confused with the well-known cycle crossover operator defined over permutations, sometimes denoted by CX. If a conflict does appear, the operator attempts to resolve the conflict by swapping the jobs at the conflicting position. It does this until a random number of cycles. The procedure of crossover operator is explained in Algorithm 3.

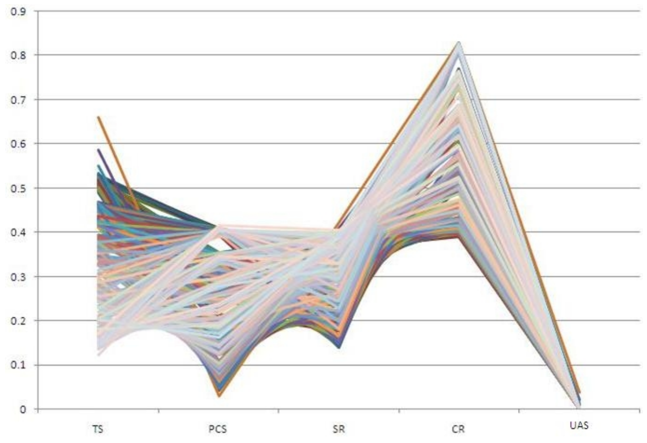


Figure 9: Multiple runs of scalarized Kuhn-Munkres.

Algorithm 3 Algorithm for Crossover

```
1: $P_1 \leftarrow$ Parent-1
2: $P_2 \leftarrow$ Parent-2
3: repeat
4:   Select a point at random
5:   Swap the jobs of $P_1$ and $P_2$ at random point
6:   Check $P_1$ and $P_2$ for repetitions
7:   if Repetitions then
8:     Swap the jobs of $P_1$ and $P_2$ to resolve the repetitions
9:   end if
10: until number of cycles
```

5. EXPERIMENTAL RESULTS AND CONCLUSIONS

The open-source software package Jmetal [10] was used for all MOEA experiments. Jmetal is a java based framework for genetic algorithms which includes several metaheuristic algorithms as well as several benchmark problems. The package was modified to include the sailor assignment problem.

Ten trials of each algorithm were performed, and the results displayed using a parallel plot method which displays each solution as a polyline connecting five points – one each along five vertical lines representing each objective. As all objectives were converted to minimization, a hypothetical perfect solution would be represented by a horizontal line along the x-axis of plots such as Figures 8, 9 and 10.

Comparing NSGA-II with the Kuhn-Munkres algorithm, the LAP method produces more diverse and much better solutions. As Kuhn-Munkres is a complete search, this performance is expected. However, the runtime required by the Kuhn-Munkres algorithm is much larger than the MOEA. To attempt to combine the fast performance of the MOEA with the superior results of the LAP algorithm, the five best solutions obtained from the Kuhn-Munkres (using the weight vectors described above) were integrated into the ini-

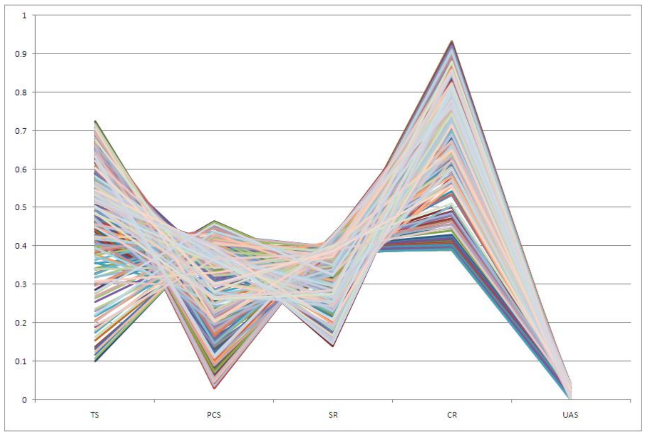


Figure 10: NSGA-II (K-M Initialization).

Table 1: Run time in hours for the included algorithms. * indicates inclusion of KM5 solutions.

Problems		Algorithms		
Sailors	Jobs	K-M(287)	NSGA-II*	SPEA2*
1000	1100	0.6686	0.48280	0.64695
2000	2100	2.4980	1.13692	1.60305
4000	4100	31.245	3.02488	4.47682
8000	10,000	383.432	8.89397	13.69670

tial population of the NSGA-II. Figure 10 shows the results, which provide essentially equivalent performance to the full Kuhn-Munkres algorithm with much lower computation time. Table 1 and Figure 11 present the required computation time for the different algorithms.

This work has shown that one may achieve substantial performance improvements in the performance of standard multiobjective evolutionary algorithms through the injection of high-quality solutions into the initial population. In the case of the Sailor Assignment Problem, a polynomial time algorithm was used to generate a small number of Pareto optimal solutions for this initial seeding, and the evolutionary algorithm started from this seed was shown to provide excellent performance in a fraction of the time required by the classical algorithm alone.

6. ACKNOWLEDGEMENTS

This project is funded under the Scientific Services Program (Task No. 07192), which is sponsored by the US Army Research Office and is managed by Battelle under Prime Contract No. W911NF-07-D-0001.The views and conclusions of this paper in no way reflect the opinions or positions of NPRST or the U.S. Government.

7. REFERENCES

[1] F. Bourgeois and J. C. Lassalle. An extension of the munkres algorithm for the assignment problem to rectangular matrices. *Communications of the ACM*, 14(12):802–804, December 1971.

[2] A. D. Doty, Iowa State University. http://www.public.iastate.edu/~ddoty/HungarianAlgorithm.html.

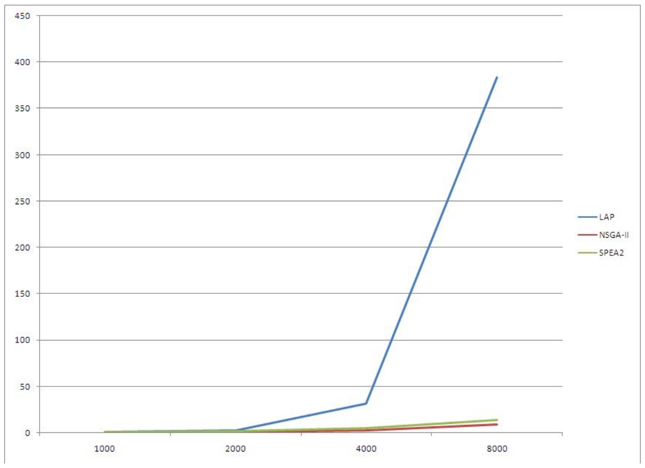


Figure 11: Comparision of Execution Times(in Hours)

[3] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[4] J. D. Garrett, J. Vannucci, R. Silva, D. Dasgupta, and J. Simien. Genetic algorithms for the sailor assignment problem. In *Proceedings of the 2005 Genetic and Evolutionary Computation Conference (GECCO-05)*. ACM press, 2005.

[5] J. D. Garrett, J. Vannucci, R. Silva, D. Dasgupta, and J. Simien. Applying hybrid multiobjective evolutionary algorithms to the sailor assignment problem. In L. Jain, V. Palade, and D. Srinivasan, editors, *Advances in Evolutionary Computing for System Design*. Springer Verlag, 2007.

[6] A. Holder. Navy personnel planning and optimal partition. *Operations research*, 53(1):77–89, January-February 2005.

[7] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2:83–97, 1955.

[8] L. McCauley and S. Franklin. A large multi-agent system for navy personnel distribution. *Connection Science*, 14(4):371–385, December 2002.

[9] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society of Industrial and Applied Mathematics*, 5(1):32–38, March 1957.

[10] Networking and S. Emerging Optimization (NEO), University of Malaga. JMetal: Metaheuristic algorithms java library. http://mallba10.lcc.uma.es/wiki/index.php/JMetal. Version 1.5, april 2008.

[11] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), Gloriastrasse 35, CH-8092 Zurich, Switzerland, May 2001.