Multi-objective Go with the Winners Algorithm: A Preliminary Study

Carlos A. Brizuela and Everardo Gutiérrez

Computer Science Department
CICESE Research Center
Km 107 Carr. Tijuana-Ensenada, Ensenada, B.C., México
{cbrizuel, egutierr}@cicese.mx
+52–646–175–0500

Abstract. This paper introduces a new algorithm to deal with multi-objective combinatorial and continuous problems. The algorithm is an extension of a previous one designed to deal with single objective combinatorial problems. The original purpose of the single objective version was to study in a rigorous way the properties the search graph of a particular problem needs to hold so that a randomized local search heuristic can find the optimum with high probability. The extension of these results to better understand multi-objective combinatorial problems seems to be a promising line of research. The work presented here is a first small step in this direction. A detailed description of the multi-objective version is presented along with preliminary experimental results on a well known combinatorial problem. The results show that the algorithm has the desired characteristics.

1 Introduction

Real world optimization problems are usually multi-objective (MO) in nature. The lack of exact methodologies to tackle these problems makes them attractive theoretically, and practically.

In the last few years many population-based strategies have been proposed to deal with these problems. These strategies seem especially well suited for MO problems where a set of solutions, instead of a single solution, is needed. Among them Genetic Algorithms [1, 9] have been the most widely used. Other approaches were proposed as an extension of their mono-objective counter part, some of these are: MO Simulated Annealing [14, 15], MO Ant Colony Optimization [16, 17], MO Particle Swarm Optimization [18], MO Evolutionary Strategy [21], MO Tabu Search [19, 20], among others.

The mainstream of research with these methods has been directed to experimental rather than theoretical issues. The reasons for this are simple: first, the analysis of these algorithms are extremely complex and the insight gained with the results are usually poor; second, the urgent need for methodologies to solve real problems is high.

On the other hand the research community in analysis of algorithms has focused in trying to find a model for the random local search heuristics. This with the objective of giving a rigorous explanation of these algorithms behavior. One such attempt has been the model of heuristics where good solutions are preferred over bad solutions in what was called the "Go with the Winner" (GWW) scheme.

The GWW algorithm have been introduced by Aldous and Vazirani [2] as an attempt to give a rigorous explanation of the behavior of some algorithms based on a non crossover "survival of the fittest" paradigm. This algorithm resembles the well known simulated annealing strategy and is restricted to work with search graph with a tree structure [2]. Dimitriou and Impagliazzo [5, 6] generalize the algorithm to deal with general search graphs. In this paper an extension of the generalized version of GWW algorithm to deal with MO optimization problems is presented.

In order to extend all theoretical results of the single objective case to the MO case, a great deal of work will be required. Our first step is to propose an extension of the algorithm and to study its performance on a *NP-hard* scheduling problem.

Scheduling appears to be one of the most challenging combinatorial MO problems. In a real scheduling problem we are interested not only in minimizing the latest completion time (makespan) but also in minimizing the total time all jobs exceed their respective due dates. These objectives are usually in conflict with each other.

The remainder of the paper is organized as follows. Section 2 presents the GWW algorithm and extends it to deal with MO problems. Section 3 states the problem to be used as a study case. Section 4 shows the experimental setup and results. Finally, section 5 presents the summary of this work.

2 The Go with the Winners Algorithm

Originally, the GWW algorithm was designed for search graphs with a tree structure and the main result is as follows [2]. If we want to increase the success probability in a randomized optimization heuristic from p (generally small) to 0.99, one way is to re-run the heuristic $O(1/p)$ times and select the best answer. In case of the GWW algorithm this number can be reduced to $log(1/p)$ runs of the algorithm by introducing some interactions between the runs. Dimitriou and Impagliazzo [5] introduced a variant of this algorithm that can be used for search graphs that are not trees. They show that for any one of a large class of distributions on search graphs their version of the GWW finds, in polynomial time, the optimum with high probability, for almost all search graphs drawn from the distribution. The most important result is that they also found a sufficient condition on the search graph structure (a combinatorial property) for the algorithm to reach the optimum with high probability. This property is known as the "local expansion property." Carson [3] uses the GWW algorithm to analyze, experimentally as well as analytically the search graph for the bisection problem. The analysis performed can give information regarding which instances will be

easy to solve by a given local search algorithm. Basically, search graphs having the local expansion property will be easy to solve by the GWW algorithm. The pseudocode for this algorithm is given below.

Algorithm 1. GWW for integer minimization [3].

> **Parameters**. P: number of particles in the population. L: number of steps in the random walk.
> **Step 1. Initialization**. Generate P random solutions. Place particles on each of these solutions. Initialize i to be the maximum objective function value in the starting population.
> **Step 2**. Until all particles are at local minima do:
> **Step 3. Restriction**. $i = i - 1$. /*decrease the threshold*/
> **Step 4. Redistribution**. Let $\mathbf{M}$ be the set of particles at solutions of cost exactly $i+1$. If $|\mathbf{M}| = P$, then exit. Otherwise, delete the particles in $\mathbf{M}$ and make $|\mathbf{M}|$ copies of randomly selected members of the remaining population.
> **Step 5. Randomization**. For every duplicate made in the previous step, execute the following random walk for L steps; select a random neighbor; if the neighbor is of cost at most i, then move to that solution, otherwise stay at the current solution. (Note that selecting a neighbor but not moving to it still counts as one of the L steps in the random walk).
> **Step 6**. Continue to stage $i = i - 1$.

The main idea with this algorithm is to improve the iterative local search procedure by the introduction of interactions between different runs of the algorithm. The interactions are given in the restriction step since i determines which solutions continue (go with the winners) and which are deleted.

The **initialization** step consists on the generation of P initial solutions. The assumption in this step is that the particles are uniformly distributed in the search graph. The **restriction** step forces the search graph to be partitioned into components with different levels, where each level represents a value of the objective function and an element of a component with no down connections represents one local optimum for the instance. In the **redistribution** stage all solutions over the current threshold level are deleted and replaced by copies of solutions under the threshold level (minimization problem). From each of these solutions, in the **randomization** step, a length L random walk is performed. This is a way to uniformly explore the subset of solutions at the current threshold level. The algorithm works on the hypothesis that the following loop invariant holds "at each iteration of the algorithm steps 3, 4 and 5 the search graph is uniformly sampled."

2.1 The Multi-objective GWW

With the same philosophy we apply these steps to the MO version of the algorithm whose pseudocode is described as follows.

Algorithm 2. MOGWW.

Parameters. P: number of particles in the population. L: number of steps in the random walk.
Step 1. **Initialization**. Generate P random solutions. Place particles on each of these solutions. Classify the solutions according to their dominance relations. Initialize i to be the worst nondominated front rank.
Step 2. Until all particles are nondominated do:
Step 3. **Restriction**. $i = i - 1$. /*decrease the threshold*/
Step 4. **Redistribution**. Let $\mathbf{M}$ be the set of particles at solutions in a nondominated front of rank $i+1$. If $|\mathbf{M}| = P$, then exit. Otherwise, delete the particles in $\mathbf{M}$ and make $|\mathbf{M}|$ copies of randomly selected members of the remaining population in higher remain fronts.
Step 5. **Randomization**. For every duplicate made in the previous step, execute the following multi-objective random walk for L steps; select a random neighbor; if the neighbor is not dominated by any solution in the front with rank i then move to it, otherwise stay at the current solution. (Note that selecting a neighbor but not moving to it still counts as one of the L steps in the random walk).
Step 6. Reclassify the solutions according to their dominance relations. Initialize i to be the worst nondominated front rank. Continue to the next stage.

This algorithm works with the same principles as Algorithm 1 does. The extension is performed by introducing a proper threshold condition and a corresponding multi-objective random walk.

The main idea regarding the threshold is as follows. At each stage the solutions are classified in nondominated fronts, here the poorest front is assigned the highest index and it is selected as the threshold, i.e. all solutions which are in that front are deleted and replaced with solutions in fronts with lower indices. An option to generalize this step can be to do the replacement with solutions coming only from a particular set of fronts (*e.g.* coming only from the best front).

Each one of the duplicates is the starting point of a length L random walk, at each step in this walk the selected neighbor is compared to solutions originally in the front adjacent to the previously deleted one. In this way it is ensured that the new solutions will not be worse than the front which is going to be the threshold at the next stage. The function of this step is to uniformly sample the search graph at a given level of solution quality. This mechanism allows the algorithm to keep a diverse set of solutions, something that is not easy to achieve, for instance in Genetic Algorithms.

An advantage for this algorithm is that it needs only two parameters L and P, where L can be determined previous to the running of the algorithm. The appropriate number of particles will depend on the search graph structure. If this is composed of many disconnected components then the required P will be large. When the graph is composed of a single component the P does not need to be large. Another point in favor of MOGWW is that it presents a 100% of nondominated solutions at the end of the run and the number of iterations is automatically determined. This number is, for a fixed L, a function of the

number of particles P. Since this multi-objective version uses the nondominance relations to define the threshold and not the objective function values it can be used also for continuous problems.

Points against the algorithm have to do with the length L of the random walk that can make the algorithm to be computationally expensive.

2.2 How to decide L: The decorrelation test

One very important condition for this algorithm to succeed is the local expansion property the search graph needs to have. In order to have an approximated measure to verify the existence of this property, Carson [3] proposed what he called the decorrelation test. The decorrelation test is proposed to understand how many steps in the random walk are needed in order to uniformly distribute the particles at each level of the graph.

The decorrelation test can be described as follows [3]: select two random vertices v and u of the search graph (solutions). From v perform a random walk of length L (parameter to be tested), and define the endpoint as v'. Compare the distance $d(v', v)$ with $d(v', u)$. For infinite random walks, v' is uncorrelated to v. Therefore, as the random walk approaches its asymptotic behavior, each distance is equally likely to be the shortest. In order to test the MOGWW with this idea, we use the relative frequency of 1000 experiments on each step as the probability of having $d(v', v) < d(v', u)$. This value representing the decorrelation probability must be near 0.5 to indicate that the algorithm pass the test, in other case the parameter L must be increased. This test gives information about the search graph structure. If the search graph is composed of a great number of weakly connected components then the L to pass the test will be large. However, if the graph have only one main component then the required L will be small.

3 A Study Case: The Permutation Flowshop Problem

The permutation flow-shop problem consists of a set $\mathbf{J}$ of n jobs that must be processed in a set of machines $\mathbf{M}$. Each job $j \in \mathbf{J}$ has $m = |\mathbf{M}|$ operations. Each operation O_{kj}, representing the k-th operation of job j, has an associated processing time t_{kj}. Each machine must finish the operation once it is started to be processed (no preemption allowed). No machine can process more than one operation at the same time. No operation can be processed by more than one machine at a time. Each job j is assigned a readiness time r_j, and due date d_j. All jobs must have the same routing through all machines. The goal is to find a permutation of jobs that minimizes a given objective function (since the order of machines is irrelevant).

In order to understand the objective functions we want to optimize we need to set up some notation first. Let us denote the starting time of operation O_{kj} by s_{kj}, its completion time by c_{kj}. Define $\mathbf{K}_m$ as the set $\{1, 2, \cdots, m\}$. With this notation a feasible solution holds the following conditions:

$$s_{kj} > r_j \quad \forall k \in \mathbf{K}_m, j \in \mathbf{J} \ , \tag{1}$$

$$s_{kj} + t_{kj} \leq s_{(k+1)j} \forall k \in \mathbf{K}_{m-1}, j \in \mathbf{J} \ . \tag{2}$$

All pairs of operations O_{kj} and O_{ri} processed on the same machine must satisfy:

$$\begin{gathered} s_{kj} + t_{kj} \leq s_{ri} \quad \text{or} \\ s_{ri} + t_{ri} \leq s_{kj} \text{ for each machine in } \mathbf{M}, k \neq r \text{ or } j \neq i \quad . \end{gathered} \tag{3}$$

Now we are in position of defining the objective functions. First we consider the makespan, which is the completion time of the latest job, i.e.

$$f_1 = \max_j \{s_{mj} + t_{mj}\} \ . \tag{4}$$

The mean flow time, representing the average value of the time during which the jobs remain in the shop, is the second objective.

$$f_2 = \bar{fl} = (1/n) \sum_{j=1}^{n} fl_j \ , \tag{5}$$

where $fl_j = s_{mj} + t_{mj} - r_j$, i.e. the time job j spends in the shop after it is released. The third objective is the mean tardiness, i.e.

$$f_3 = \bar{T} = (1/n) \sum_{j=1}^{n} T_j \ , \tag{6}$$

where $T_j = \max\{0, L_j\}$, and $L_j = s_{mj} + t_{mj} - d_j$.
Thus, we have the following MO problem:

$$\begin{gathered} \min(f_1, f_2, f_3) \\ \text{subject to } (1) - (3) \ . \end{gathered} \tag{7}$$

The application of GA's to MO scheduling problems has been rather scarce. Interesting ideas are presented in [10], [11], and [12].

In [11] the scheduling of identical parallel machines, considering as objective functions the maximum flow time among machines and a non-linear function of the tardiness and earliness of jobs, is presented. In [12] a natural extension of NSGA [13] is presented and applied to flow-shop and job-shop scheduling problems. Another, totally different approach is that presented by Isibuchi and Murata [10]. They use a local search strategy after the genetic operations without considering non-dominance properties of solutions. Their method is applied to the MO flow-shop problem.

4 Experimental setup and results

The experiments have two main objectives. The first one is related to the performance study of the algorithm when different random walk lengths are used. The second one has to do with the study of the relative performance when comparing the MOGWW with a well known genetic algorithm.

The neighborhood for this particular problem is defined by the insert operator described in [7]. The distance measure used in the decorrelation test is based on the minimum number of times this operator needs to be applied to one solution to reach the other.

Before any experiment, aimed at measuring the algorithm performance, is carried out the decorrelation test is applied for the analyzed instance. Two different lengths were tried $L = 35$ and $L = 75$, the obtained results for the mean decorrelation probability were a 0.678 (2.26% standard deviation) and 0.522 (3.08% standard deviation) respectively. As the reference value is 0.5 we can see that $L = 35$ fails to pass the test and $L = 75$ success in the test.

Once we know that the length to use is $L = 75$, we need to study the performance of this algorithm. Previous to this, some performance measures are defined.

4.1 Performance measures

The performance measures are related to the dominance relations between nondominated fronts generated by the studied algorithms. The measures can be defined as follows [7]: m^1 counts the number of times a dominance relation can not be establish between a solution given by algorithm **A** and the set of solutions given by algorithm **B**, m^2 counts the number of solutions given by algorithm **B** dominate solutions generated by algorithm **A**, m^3 counts the number of times solutions given by algorithm **A** dominate solutions given by algorithm **B**, finally m^4 counts the number of times solution has copies of itself in the other nondominated set.

4.2 Random walk length and Performance comparison

The set of instances used here are the same as those described in a recent work on the same problem [7], and which are available on the web [1]. Ten instances of 75-jobs and 20-machines were used to perform the experiments. For each instance the number of particles is set to $P = 100$ and the decorrelation length L is fixed to 75. The length was set by using the decorrelation test proposed by Carson [3].

For each studied instance 50 runs are performed taking the union of all the resulting nondominated solutions. Figures 1, 2 and 3 show the projections of the solutions obtained by two different decorrelation lengths, one which does not pass the decorrelation test ($L = 35$), and another which does ($L = 75$).

[1] *http://www.cicese.mx/~cbrizuel/MOGWW/instances.html*

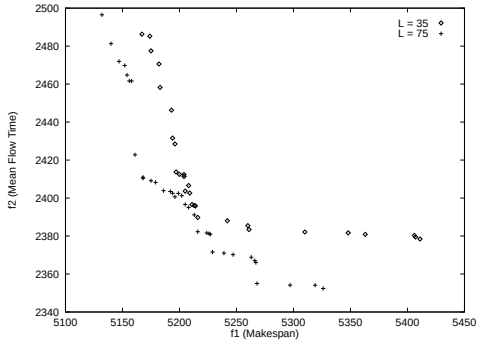

Fig. 1. Projection of Makespan and Mean Flow Time for the union of all nondominated solutions of 50 runs with $L = 35$ and $L = 75$ (Instance 6, $P = 100$).

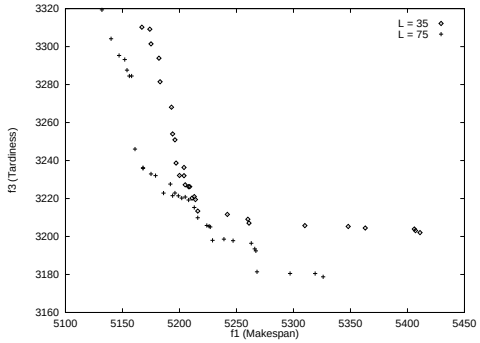

Fig. 2. Projection of Makespan and Tardiness for the union of all nondominated solutions of 50 runs with $L = 35$ and $L = 75$ (Instance 6, $P = 100$).

Table 1 shows the dominance relations between these two sets of solutions. The obtained results with the length which pass the decorrelation test dominate 20% of the solutions obtained with the length which does not pass the test ($L = 35$). Equivalent results were obtained for the other instances (not shown here). This result tell us about the importance of using the appropriate length L in order to have a good uniformity, and therefore a good sampling performance.

4.3 GA vs. MOGWW

The set of nondominated solutions obtained with length ($L = 75$) which pass the decorrelation test for each analyzed instance is compared to the results obtained in a previous work [7]. Figures 4, 5 and 6 show the projections of the nondominated fronts obtained by the MOGWW and the one reported in [7]. The parameters used by the GA are as follows: population size is 100, number of generations is 2000, crossover rate is 1.0 and mutation rate is 0.1. Table 2 shows

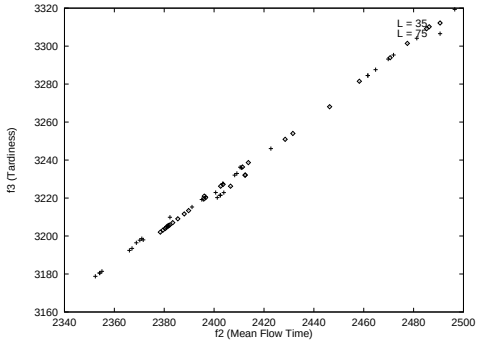

Fig. 3. Projection of Mean Flow Time and Tardiness for the union of all nondominated solutions of 50 runs with $L = 35$ and $L = 75$ (Instance 6, $P = 100$).

Table 1. Dominance relations between the union of all nondominated solutions of 50 runs with $L = 35$ and $L = 75$ (Instance 6, $P = 100$)

L	m^4	m^3	m^2	m^1
35	0.00%	0.00%	20.00%	80.00%
75	0.00%	20.00%	0.00%	80.00%

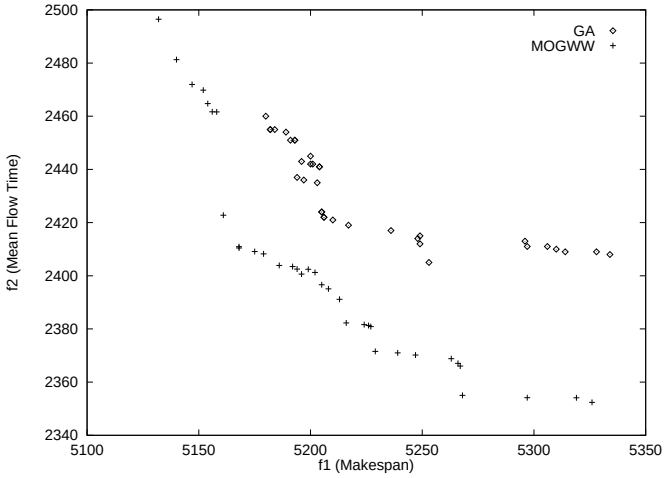

Fig. 4. Projection of Makespan and Mean Flow Time for the union of all nondominated solutions of 50 runs of GA and MOGWW (Instance 6).

the nondominance relations between these two sets of solutions. The results obtained by the MOGWW algorithm clearly dominate to the solutions obtained by the GA algorithm.

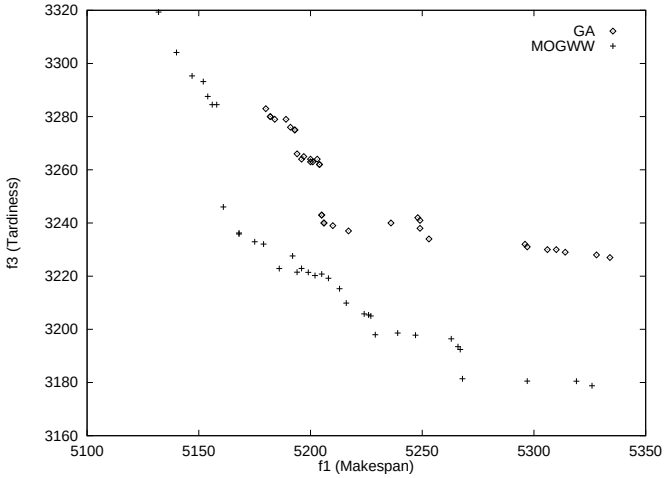

Fig. 5. Projection of Makespan and Tardiness for the union of all nondominated solutions of 50 runs of GA and MOGWW (Instance 6).

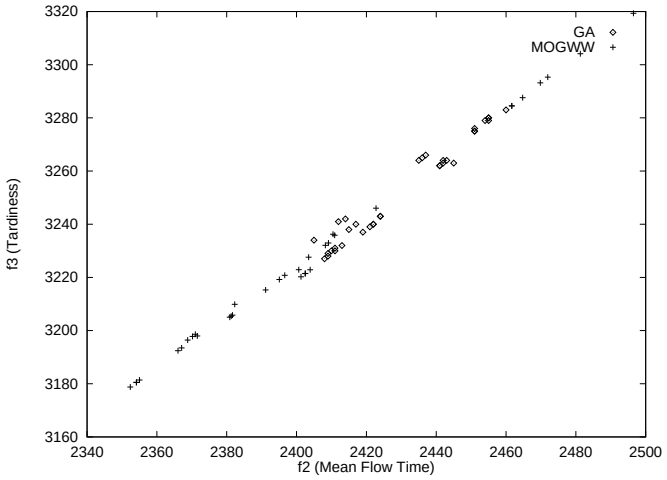

Fig. 6. Projection of Mean Flow Time and Tardiness for the union of all nondominated solutions of 50 runs of GA and MOGWW (Instance 6).

Figures 7, 8 and 9 show the projections of the solutions obtained by one run of the MOGWW and one of the GA [7], both selected randomly.Table 3 shows the nondominance relations between these two sets of solutions. The result shows a very tight case where more than 90% of the solutions are non comparable.

Table 2. Dominance relations between the union of all nondominated solutions of 50 runs of GA and MOGWW (Instance 6)

Algorithm	m^4	m^3	m^2	m^1
GA	0.00%	0.00%	36.57%	63.43%
MOGWW	0.00%	36.57%	0.00%	63.43%

Table 3. Dominance relations between nondominated solutions of a single run of GA and a single run of MOGWW (Instance 6)

Algorithm	m^4	m^3	m^2	m^1
GA	0.00%	2.36%	5.26%	92.38%
MOGWW	0.00%	5.26%	2.36%	92.38%

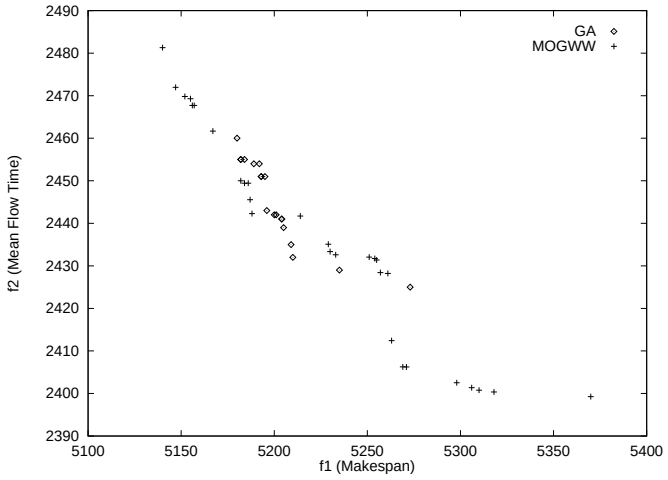

Fig. 7. Projection of Makespan and Mean Flow Time for the nondominated solutions of one run of GA and MOGWW (Instance 6).

Even though the fronts are similar with respect to the nondominance relations, they are different in the front dispersion given by the algorithms. In this sense MOGWW has a bigger front dispersion than the GA. This is a result of the good diversity maintenance mechanism of the MOGWW.

Table 4 shows the dominance relations between the union of all nondominated solutions obtained by 50 runs of the MOGWW and the union of all nondominated solutions obtained by 50 runs reported in [7]. The results for all instances

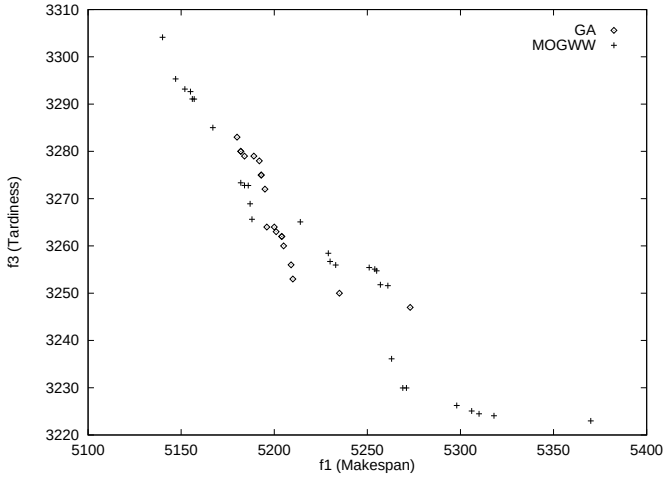

Fig. 8. Projection of Makespan and Tardiness for the nondominated solutions of one run of GA and MOGWW (Instance 6).

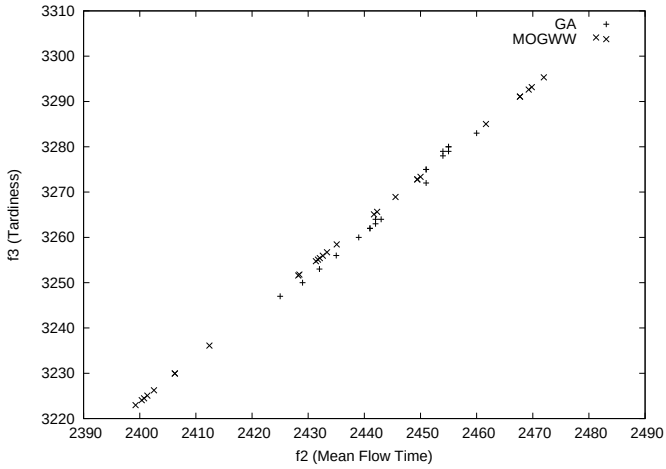

Fig. 9. Projection of Mean Flow Time and Tardiness for the nondominated solutions of one run of GA and MOGWW (Instance 6).

Table 4. Dominance relations between the union of all nondominated solutions of 50 runs of GA and MOGWW (10 instances)

	Algorithm	m^4	m^3	m^2	m^1
Instance 0	GA	0.00%	0.00%	19.23%	80.77%
	MOGWW	0.00%	19.23%	0.00%	80.77%
Instance 1	GA	0.00%	0.00%	43.69%	56.31%
	MOGWW	0.00%	43.69%	0.00%	56.31%
Instance 2	GA	0.00%	0.00%	30.62%	69.38%
	MOGWW	0.00%	30.62%	0.00%	69.38%
Instance 3	GA	0.00%	0.00%	16.74%	83.26%
	MOGWW	0.00%	16.74%	0.00%	83.26%
Instance 4	GA	0.00%	0.00%	23.12%	76.88%
	MOGWW	0.00%	23.12%	0.00%	76.88%
Instance 5	GA	0.00%	0.00%	32.27%	67.73%
	MOGWW	0.00%	32.27%	0.00%	67.73%
Instance 7	GA	0.00%	0.00%	32.80%	67.20%
	MOGWW	0.00%	32.80%	0.00%	67.20%
Instance 8	GA	0.00%	0.00%	24.34%	75.66%
	MOGWW	0.00%	24.34%	0.00%	75.66%
Instance 9	GA	0.00%	0.00%	59.10%	40.90%
	MOGWW	0.00%	59.10%	0.00%	40.90%

Table 5. Number of iterations and evaluations

Algorithm Name	Mean Number of Iterations	Standard Deviation (Iterations)	Mean Number of Evaluations	Standard Deviation (Evaluations)
GA	2000.00	0.00%	200000.00	0.00%
MOGWW	1028.26	10.19%	715776.00	10.83%

including those in Table 2 show a clear superiority of the proposed MOGWW over the GA. The algorithm used in [7] is probably not the best up to date GA for the problem, but the results are competitive. Similar results, not shown here, were obtained by comparing the MOGWW with the well-known NSGA-II [4] for the same set of instances [8].

Table 5 shows the mean number of iterations and evaluations and their standard deviations for the MOGWW and the GA. This table shows that the proposed MOGWW needs more computing time than the GA, therefore we may increase the population size of the GA and perhaps obtain better results. However, the main point here is to show that the algorithm works similar to what its single objective version does. On the other side an interesting characteristic is that the algorithm needs only two parameters, one of which can be easily tuned previous to the run of the algorithm.

5 Conclusions

A new algorithm for continuous and combinatorial multi-objective optimization problems have been proposed. The algorithm is based on the "Go with the winner" paradigm introduced to model randomized local search. A detailed description of the algorithm was presented along with the ideas behind each step. The new algorithm is tested with a classical scheduling problem, the permutation flow shop. Preliminary results show that the algorithm works following the principles of its single objective version.

Future research is planned to extend the motivating theoretical results obtained with the algorithm in single objective problems to the multi-objective cases. Additionally future research comparing the performance of the proposed algorithm against a multi-objective multi-start local search with a replacement criteria based on pareto-dominance relations would be interesting.

Acknowledgments

The authors would like to the thank the anonymous referees for their valuable comments and interesting ideas for future research.

References

1. Coello Coello, C. A., Van Veldhuizen, D. A., and Lamont, G. B. *Evolutionary Algorithms for Solving Multi-objective Problems.* Kluwer Academic Publishers, New York (2002)
2. Aldous, D., Vazirani, U. "Go with the winners algorithms". In Proceedings of the *35th IEEE Symposium on Foundations of Computer Science* pp. 492-501 (1994)
3. Carson, T. Empirical and analitic approaches to understanding local search heuristics. PhD Thesis, University of California. San Diego, California (2001)
4. Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. "A fast elitist non-dominated sorting algorithm for multi-objective optimization: NSGA-II". In *Parallel Probem Solving from Nature – PPSN VI*, Berlin. Springer (Marc Shoenauer, et al., ed.), pp. 849–858 (2000).
5. Dimitriou, A, Impagliazzo, R. "Towards a rigorous analysis of local optimization algorithms". In Proceedings of the *25th ACM Symposium on the Theory of Computing* (1996)
6. Dimitriou, A, Impagliazzo, R. "Go-with-the-winner algorithms for graph bisection". In Proceedings of the *9th Annual SIAM Symposium on Discrete Algorithms* pp. 510-520 (1998)
7. C. A. Brizuela and R. Aceves, "Experimental genetic operators analysis for the multi-objective permutation flowshop." In Fonseca C., Fleming P., Zitzler E., Deb K. and Thiele L. (Editors) *Evolutionary Multi-criterion Optimization*, LNCS Vol. 2632, pp. 578 - 592, (2003).
8. R. Aceves and C. A. Brizuela, "Análisis Experimental de Operadores Genéticos en NSGA-II para un Problema de Calendarización Multi-objetivo." In Botello S., Hernández A. and Coello C. (Eds.) *Congreso Mexicano de Computación Evolutiva*, pp. 55 - 66, (2003).

9. Deb, K. *Multi-Objective Optimization using Evolutionary Algorithms* John Wiley & Sons (2001)
10. Isibuchi, H. and Murata, T. "Multi-objective Genetic Local Search Algorithm". Proceedings of the 1996 *International Conference on Evolutionary Computation*, pp:119-124, (1996).
11. Tamaki, H., and Nishino, E. A "Genetic Algorithm approach to multi-objective scheduling problems with regular and non-regular objective functions". Proceedings of *IFAC LSS'98*, pp:289-294 (1998).
12. Bagchi, T. P. *Multiobjective Scheduling by Genetic Algorithms.* Kluwer Academic Publishers (1999).
13. Srinivas, N. and Deb, K. Multi-Objective function optimization using non-dominated sorting genetic algorithms. *Evolutionary Computation*, 2(3), pp:221-248 (1995).
14. Serafini, P. "Simulated Annealing for Multiple Objective Optimization Problems". In Tzeng, G., Wang, H., Wen, U., and Yu, P. (editors) *Proceedings of the Tenth International Conference on Multiple Criteria Decision Making: Expand and Enrich the Domains of Thinking and Application*, volume 1 pp. 283-292 (1994)
15. Thompson, M. "Application of Multi Objective Evolutionary Algorithms to Analogue Filter Tuning". In Zitzler, E., Deb, K., Thiele, L., Coello Coello, C. A., and Corne, D., (editors) *First International Conference on Evolutionary Multi-Criterion Optimization.* LNCS No. 1993 pp. 546-559 (2001)
16. Mariano, C. E., Morales, E. "MOAQ an Ant-Q Algorithm for Multiple Objective Optimization Problems". In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., (editors) *Genetic and Evolutionary Computing COnference (GECCO 99)*, volume 1, pp. 894-901 (1999)
17. Gambardella, L. M., Taillard, E., Agazzi, G. "MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows". In Corne, D., Dorigo, M., and Glover, F., (editors) *New Ideas in Optimization.* McGraw Hill pp. 63-76 (1999).
18. Kennedy, J., Eberhart, R. C. "Particle Swarm Optimization". In Proceedings of the 1995 *IEEE International Conference on Neural Networks*, pp. 1942-1948 (1995)
19. Hansen, M. P. "Tabu Search in Multiobjective Optimisation: MOTS". In Proceedings of the *13th International Conference on Multiple Criteria Decision Making (MCDM'97)* (1997)
20. Gandibleux, X., Mezdaoui, N., and Fréville, A. "A Tabu Search Procedure to Solve Combinatorial Optimisation Problems". In Caballero, R., Ruiz, F., and Steuer, R. E., (editors) *Advances in Multiple Objectives and Goal Programming*, volume 455 of Lecture Notes in Economics and Mathematical Systems, pp. 291-300 (1997)
21. Binh, T. T., and Korn, U. "An evolution strategy for the multiobjective optimization". In The *Second International Conference on Genetic Algorithms* (Mendel 96) pp. 23-28 (1996)