Multi-objective evolutionary algorithms for a class of sequencing problems in manufacturing environments*

Carlo Meloni, David Naso, and Biagio Turchiano
Dipartimento di Elettrotecnica ed Elettronica
Politecnico di Bari, Bari, Italy
Phone: +39-080-5963 851 Fax: +39-080-5963 410
E-mail: {meloni, naso, turchiano}@deemail.poliba.it

Abstract - *This paper describes a multi-objective evolutionary algorithm for a typical serial production problem, in which two or more consecutive departments must schedule their internal work, each taking into account the requirements of the other departments. There are various single-objective heuristics to deal with this problem, while the multi-objective formulation calls for innovative approaches. To this aim, we devise a novel evolutionary algorithm, and compare it with two other state-of-art genetic optimizers used in similar contexts. The results obtained on both small-size problems with known Pareto-sets, and larger problems derived from industrial production of furniture confirm the effectiveness of the proposed approach.*

Keywords: Multi-Objective Evolutionary Algorithms, Scheduling, Sequencing, Manufacturing systems

1 Introduction

The set-up coordination of processing operations is a typical problems of multi-stage serial manufacturing or supply chains [1]. In such environments, usually two or more consecutive departments have to schedule their internal work, each taking into account the requirements of the other stages. The models and instances considered in this paper derive from the industrial environment described in [1,9]. Namely, in different departments of the plant, items are grouped according to different attributes. In the first department, parts are grouped according to their first characteristic; and in the subsequent departments parts are grouped according to their second and subsequent attributes. Often, the unavailability of interstage buffering between two consecutive departments forces the processing sequences in the two stages to be coordinated. In this case, the two departments must follow the same processing sequence, leading to a considerably complex optimization problem with partially conflicting objectives, since each department aims to minimize its own total changeover cost. Clearly, also the global cost, i.e. the sum of the costs of each department, must be accounted in optimization process.

Some efficient heuristic algorithms are available in literature to solve the sequencing problem considering only a single objective, e.g. the sum of all costs, or the maximum of the costs of each department [1,9], while the multi-objective formulation is an extremely complex problem calling for innovative approaches. In particular, as for many manufacturing problems [3] the nature of the problem seems to be particularly suitable for Evolutionary Computation (EC) tools.

Evolutionary Algorithms (EA) are a class of heuristic search techniques inspired to the principles of survival-of-the-fittest in natural evolution and genetics. In single-objective optimization problems, the search is guided by a scalar merit function describing the fitness of the specific solution with respect to problem objectives and constraints. In recent years, interesting and successful extensions of EC methods to multi-objective optimization problems have been developed. In these Multi-Objective EA (MOEA) the search goals are not expressed or aggregated in a single scalar index of quality, but rather considered separately so that each particular solution is associated to a vector of values expressing the satisfaction of each of the different objective functions.

Technical literature describes many efficient MOEA searching for the Pareto Set [2,4,5,8], and some comparative analyses [11] have also recently been performed. We overview the recent research in the considered context in the next section. In this paper, we describe a new evolutionary algorithm specifically devised for the considered multi-objective problem. After describing the main assumptions of our problem formulation in Section 3, we illustrate the main peculiarities of our algorithm in Section 4. To evaluate the effectiveness of the proposed EA, we compare it with two other MOEA selected among the state-of-the art of this research area. Section 5 summarizes the results of the comparison on small size problems with known Pareto Set, and realistic size problems derived from industrial data. Final remarks in Section 6 conclude the paper.

* 0-7803-7952-7/03/$17.00 © 2003 IEEE.

2 Related literature

The purpose of MOEA is to optimize simultaneously a set of two or more objectives (see e.g. [4] for an introduction). With the exception of aggregative approaches that artificially combine multiple indices in a single fitness, all MOEA work iteratively searching in a population of solutions the set of non-dominated individuals. Ranking solutions with multi-objective fitness to assign higher probabilities of reproduction to better solutions is not as straightforward as in the case of scalar fitness (see [2,4,5,8], to mention some recent contributions).

Recent literature on MOEA focuses on specific strategies to cope with some critical limitations of this class of optimizers. It must be remarked that only few recently proposed MOEA can be easily applied to our sequencing problem. In particular, distance-based niching techniques that are frequently used in MOEA to continuously preserve the population diversity cannot be employed in our context due to the lack of a proper metric to effectively measure the distance between two solutions of our problem. Nevertheless, some of the most recent evolutionary approaches for multi-objective optimization can be easily employed in our context. For instance, the Pareto Archived Evolution Strategy (PAES) [8] proposes a simple (1+1) evolutionary strategies that effectively overcomes the difficulty of maintaining a population of non-dominated individuals. Namely, this approach maintains an archive of the non-dominated solutions progressively found during the search. The size of the archive is limited removing individuals according to the degree of crowding of their neighborhood. The high computational complexity of non-dominated sorting is also tackled in [2] with an improved version of the Non Dominated Sorting GA (NSGA-II). The algorithm uses an improved multi-objective sorting method that needs a significantly smaller number of comparisons ($O(NM^2)$) with respect of the $O(NM^3)$ of conventional nondominated sorting, where M is the number of objectives, and N the population size). NSGA-II also uses a crowding strategy always selecting the solutions in the less crowded areas of the currently-known Pareto front. Another interesting approach in the context of multi-objective scheduling is the Multi-Objective GA with Local Search (MOGALS) algorithms described in [5]. The authors propose a technique to maintain a population of non-dominated solutions using a randomized ranking. Namely, in a problem with n objectives, at each time two individuals must be compared, n random weights are extracted, and the weighted aggregation of the n fitness values of each individual are used for ranking. The randomization of the aggregation weights produces a selective pressure with an always changing direction, and consequently the population is progressively lead toward the Pareto set. The authors also equip their EA with a local search algorithm that cyclically explores a number of neighborhood solutions (obtained with random shifts of a job in other positions of the sequence) to achieve local improvements. The contribution of local search to the overall convergence is enhanced in subsequent versions of the algorithm [6] replacing the random mechanism selecting the direction of local search with an heuristic criterion.

The algorithm proposed in the next sections shares some basic ideas with all the mentioned approaches. In particular, since in [2] the NSGA-II provided higher performances than PAES in a wide range of typical benchmarks for multi-objective optimization, the NSGA-II and the MOGALS are used as a terms of comparisons for our research.

3 General assumptions

We focus on the coordination issues between the first two stages of the furniture production chain introduced in Section 1, and therefore consider only two attributes, called hereinafter *shape* and *color*. In both departments, a changeover occurs when the attribute of a new part changes. If a part must be cut having a different shape from the previous one, cutting machinery must be reconfigured. Similarly, when a new color is used, the painting station must be cleaned in order to eliminate the residuals of the previous color. In both cases costs are incurred in terms of time and manpower. Other important issues make the sequencing problem an extremely difficult task. For example, setups have different costs. In the painting department, set-up operation takes less time to switch from a lighter to a darker color than vice-versa. In some cases, there are precedence constraints that limit the set of feasible sequences in the two departments. Hence, the total number of changeovers can be considered in these cases a meaningful index of performance.

This paper addresses the problem in its basic version, i.e. when all the setup operations in each department and across departments have the same cost, and changes of the processing sequence between the two stages are not allowed. The sequencing optimization is formulated as a multi-objective problem, in which an optimal tradeoff between local and global indices of performance has to be found. Each item to be produced is characterized by its own shape and color. All the items having the same shape and color form a single batch. In the first (second) department, a changeover is paid when the new batch has a different shape (color) from the previous one. Otherwise, no changeover is incurred. Note that since we want to minimize the number of changeovers, the actual cardinality of each batch is of no interest at all. Each given sequence of the batches results in a cost of changeovers to be paid by each department. Thus, the three objectives that should be simultaneously optimized are minimization of the total cost of changeovers in the two departments, minimization of the maximum cost of changeovers paid by either department, and minimization of the overall frequency of setups. While the first objective corresponds to the maximization of overall utility, the second captures

more realistically the need to balance the changeover costs between the two departments. The third one is clearly related to technical and organizational constraints. Note that the last objective function requires information about the number of parts contained in each batch.

The problem we consider can be formulated as follows. Let A be a set of batches to be produced. The batches must be processed by two departments of the plant, called D_S and D_C, in the same order. Each batch is characterized by two attributes, say shape and color. Let S and C denote the sets of all possible shapes and colors respectively. We denote the shapes as s_i, $i=1,\ldots,|S|$ and the colors as c_j, $j=1,\ldots,|C|$. Each batch is therefore defined by a pair (s_i,c_j). If batch (s_i,c_j) is processed immediately after batch (s_h,c_k), a changeover is paid in department D_S if $s_h \neq s_i$, and a changeover is paid in department D_C if $c_k \neq c_j$. We can represent the input of the problem as a list B of batches. The problem is to sequence the batches in a profitable way from the viewpoint of the number of changeovers. This means that we must find an ordering σ of the elements of B. If two consecutive batches $B_1=(s_i,c_j)$ and $B_2=(s_h,c_k)$ in σ have no attribute in common, this means that both departments have to pay one changeover when switching from batch B_1 to batch B_2. We refer to this as a global changeover. On the other hand, if $i=h$ $(j=k)$, only department D_C (D_S) pays a changeover. This is called local changeover. For a given sequence σ, we can therefore easily compute the number of changeovers incurred by each department, call them $N_S(\sigma)$ and $N_C(\sigma)$ respectively. In fact, let δ_{ih} be equal to 1 if $i \neq h$ and 0 otherwise, and let $s(\sigma(q))$ denote the shape of the q-th batch in the sequence σ, and let $c(\sigma(q))$ indicate its color.

Hence, the expression of N_S and N_C is the following:

$$N_S(\sigma) = \sum_{q=1}^{|B|-1} \delta_{s(\sigma(q)),s(\sigma(q+1))} \; ; \; N_C(\sigma) = \sum_{q=1}^{|B|-1} \delta_{c(\sigma(q)),c(\sigma(q+1))} \; .$$

Then the objective functions used in our preliminary campaign of computational experiments can be formulated as follows:

a) minimize $N_S(\sigma)$;
b) minimize $N_C(\sigma)$;
c) minimize $max\{N_S(\sigma),N_C(\sigma)\}$;
d) minimize $(N_S(\sigma)+N_C(\sigma))$.

4 The proposed EA

For brevity, we will refer to our algorithm as EFGA (Elitist Front GA). Similarly to PAES algorithm, EFGA stores the non-dominated solutions found during the iterations of the GA in a distinct archive. Our algorithm works on four sets of solutions, called *Pop(i)*, *Pool(i)*, *Front(i)* and *Seeds*, where i is the iteration index.

- The set *Seeds* simply encompasses the a-priori known initial solutions, which can be profitably used to speed-up the initial exploration. Typically, some good seeds can be easily obtained with efficient heuristics for the single-objective case. The algorithm can also startup with an empty *Seeds* set if such information is not available.
- Similarly to all the EAs, *Pop* represents the set of p individuals progressively evolved by the search strategy.
- The *Pool* is another set of variable size that is primarily used for offspring selection with a mechanism that will be described later.
- The *Front* is the set of variable size containing the currently known Pareto-set.

At the beginning of the search, all the sets (with the possible exception of *Seeds*) are empty. If non-empty, *Seeds* is copied into *Pop(0)*. The rest of *Pop(0)* is generated randomly, and the fitness of each individual is evaluated. Then, nondominated solutions are determined, and copied into *Front(0)* and *Pool(0)*. Dominated solutions can also enter the *Pool(0)* if they lie inside a pre-specified neighborhood band around *Front*(0), as depicted in Fig.1 (more details will be discussed later on). Once the *Pool(0)* is completed, *Pop(1)* is created in the following way: two solutions in *Pool(0)* are selected randomly for crossover, and the two resulting individuals are placed into *Pop(1)*. This operation is repeated n_{cro} times. Subsequently, an individual from either *Pool(0)* or *Pop(1)* is selected randomly for mutation, and the resulting individual is placed in the population. This operation is repeated n_{mut} times, so that $2n_{cro}+n_{mut}=p$. After the recombination phase, the fitness of each solution on *Pop(1)* in the population is evaluated. Initially, *Pool(1)* and *Front(1)* inherit the individuals in *Pool(0) and Front(0)*, respectively. Then, individuals in *Pop(1)* are compared with those in *Front(1)*, to incrementally update the two sets as follows. For each individual in *Pop:*

- If the individual dominates other individuals in the *Front*, the dominated solutions are discarded from *Front*, and the new individual is inserted in both *Pool* and *Front*.
- If the individual is dominated by some elements of *Front*, but still lies inside a predefined neighborhood of some dominant solution, it is inserted only in *Pool*.
- If the individual has exactly the same fitness of a solution in *Front*, a local search procedure is applied.

From this point on, the algorithm can be iterated until a predefined stopping condition.

A significant peculiarity of our approach is to replace the nondominated sorting used in most MOEA to select the mating pool, with a threshold condition testing the distance of a solution from the current front. In other words, only the dominated solutions laying inside the band delimited by the current front and its projection at a predefined distance *dist* are allowed to enter the mating *Pool*. For simplicity and due to the similarity between the multiple objectives considered in our problem, we choose the same distance for all the objectives, even though different choices are also possible to emphasize the selective pressure in specific directions. The parameter *dist* is cyclically modified during the evolutionary search.

In particular, *dist* is changed to $dist\text{-}\Delta_{dist}$ at every n_{dist} iterations. When *dist* becomes zero, it is reset to a large value $dist_{max}$. Also in this case, the reduction mechanism is purely heuristic, but able to provide significant improvements according to our empirical observations. In particular, this cyclical enlargement-restriction of the threshold condition for the selection provides a satisfactory tradeoff between selective pressure and population diversity. However, since the number of elements in the *Pool* set may occasionally become excessively high, at every n_{pool} iterations, the *Pool* is emptied and reinitialized with the solutions in the current *Front*.

As previously mentioned, at every time two or more (distinct) individuals with the same fitness are found in the *Pool*, a local search procedure is initialized. Similarly to many crowding indices conventionally used in MOEA (see e.g. [2]), this condition aims to find a single improved solution that dominates (and hence excludes from the *Pool*) the two or more individuals sharing the same fitness vector. We must also remark that the proposed triggering condition is explicitly tailored for the considered problem (based on integer-valued fitness vectors), but can be easily generalized by substituting the exact matching with a less specific condition (e.g. when a norm of the difference of the fitness vectors of two individuals is lower than a threshold).

To describe the local search mechanism, let us suppose that s_1 and s_2 indicate two solutions in the *Pool* with the same fitness. Let us also suppose that these solutions are found at iteration *k*. The first stage of the local search consists in launching another EA with a reduced population, and initialized with either s_1 or s_2 and two other solutions randomly picked from *Front*(*k*) as initial *Seeds*. Furthermore, the EA uses a subset of randomly selected individuals from *Pop(k)* as initial population, and works exactly in the same way as the main EA until a predefined maximum number of fitness evaluations is exceeded. In other words, the first stage of the local search is performed using the same main EA but with a reduced population and running for a limited number of iterations. This mechanism is inspired by recent research confirming on the effectiveness of microgenetic algorithms (i.e. GA with small populations and few iterations) [7] as local hillclimber. At the end of the local EA, the resulting (local) *Front* is used to update the *Front(k)* of the main algorithm. If two or more solutions with identical fitness are still found at this point (e.g. s_1' and s_2'), the second stage of the local search is executed. In this case, a single-objective is selected randomly and a problem-specific, single-objective heuristic is applied. This heuristic compares s_1' and s_2' and searches for the largest common subsequence of jobs shared between the two solutions. Then, the heuristic constructs a new solution by adding the remaining jobs on either the left-hand or the right-hand side of the shared subsequence with a heuristic procedure that guarantees a good satisfaction of the selected single objective [9].

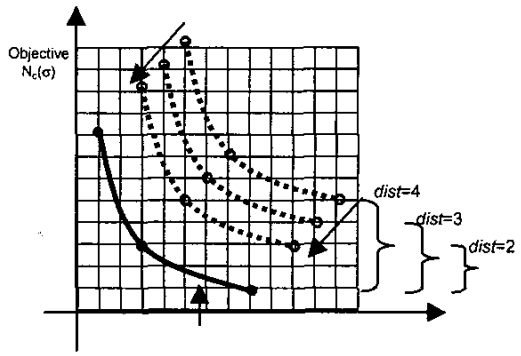

Figure 1. A graphical interpretation of *Front* and *dist* on the Cartesian plane of the two objectives N_S and N_C

5 Performance comparison

This section describes a preliminary performance comparison of the proposed EFGA with two other multi-objective optimizers derived from contemporary state-of-art of MOEA, namely NSGA-II [2], and MOGALS [5,6]. The comparison is based on two multi-objective sequencing problems. The first one is a small-size problem, consisting in the optimization of a sequence of only 9 jobs, whose shape and color attributes are summarized in Table I. In this case the true Pareto front is determined by exhaustive search on the 9!=362880 possible solutions. The free parameters of EFGA, NSGA2 and MOGALS are summarized in Table I. In particular, the parameters for NSGA2 and MOGALS were set to the suggested values proposed in the respective references [2,5,6]. All the compared algorithms must simultaneously optimize the three objectives a,b and d described in Section 3. Each algorithm is tested on runs with different stopping conditions, respectively 5000, 10000, 20000, and 40000 fitness function evaluations for both NSGA-II and MOGALS. Each algorithm is launched 10 times to determine the average results summarized in Table II. It is evident in the table that the stopping conditions for the EFGA are slightly different (1250, 2500, 5000, 10000) since this algorithm is always able to find all the solution on the true Pareto front within 10000 fitness evaluations. The obtained results clearly confirm that EFGA is much faster than both MOGALS and NSGA-II in evolving and maintaining a set of non-dominated solutions that progressively approach the true Pareto set.

Table I: *Shapes* (S_i) and *Colors* (C_j) for the first sequencing problem

	C_1	C_2	C_3	C_4	C_5
S_1	1	2	3		
S_2			4	5	
S_3		6	7		8
S_4	9				

Table II. Comparison of the three algorithms on the first case study

MOGALS		test1	test2	test3	test4	test5	test6	test7	test8	test9	test10	*Average*
5000	Found F.	4	6	7	5	4	4	6	7	7	5	*5,5*
	True F.	4	1	1	1	2	0	1	1	2	3	*1,6*
10000	NTrue F.	6	6	5	7	7	6	4	6	6	5	*5,8*
	True F.	4	6	5	7	5	6	4	3	6	3	*4,9*
20000	Found F.	7	5	6	6	6	7	5	5	8	6	*6,1*
	True F.	7	4	5	2	5	5	5	4	5	6	*4,8*
40000	Found F.	8	6	5	8	9	7	6	8	8	8	*7,3*
	True F.	8	4	4	8	9	7	5	8	8	8	*6,9*

NSGA II		test1	test2	test3	test4	test5	test6	test7	test8	test9	test10	*Average*
5000	Found F.	3	2	3	3	6	5	5	2	1	3	*3,3*
	True F.	3	2	2	3	4	4	1	2	0	2	*2,3*
10000	Found F.	3	3	3	5	5	6	5	3	8	4	*4,5*
	True F.	3	2	2	4	2	6	4	3	5	3	*3,4*
20000	Found F.	3	5	5	6	10	6	6	5	7	5	*5,8*
	True F.	3	4	5	6	8	6	6	4	7	5	*5,4*
40000	Found F.	6	8	8	9	5	8	5	5	8	9	*7,1*
	True F.	5	8	8	6	5	8	3	5	8	9	*6,5*

EFGA		test1	test2	test3	test4	test5	test6	test7	test8	test9	test10	*Average*
1250	Found F.	8	7	7	7	5	7	6	7	6	9	*6,9*
	True F.	4	2	1	4	2	4	4	5	5	2	*3,3*
2500	Found F.	6	9	5	9	7	8	8	8	7	8	*7,5*
	True F.	6	9	4	9	7	8	8	8	5	8	*7,2*
5000	Found F.	9	9	8	9	9	8	9	9	9	9	*8,8*
	True F.	9	9	8	9	9	8	9	9	8	9	*8,7*
10000	Found F.	9	9	9	9	9	9	9	9	9	9	*9*
	True F.	9	9	9	9	9	9	9	9	9	9	*9*

Legend: Found F. = number of solutions in the *Front* at the end of the run.
True F. = number of solutions belonging to the true Pareto Front

It is interesting to note that in this small-size case the EFGA is able to find more than 80% of the Pareto front already in 2500 function calls, whereas the two terms of comparison need to explore ten times more solutions before reaching a comparable result.

Table III: *Shapes* (S_i) and *Colors* (C_j) for the second sequencing problem

	C_1	C_2	C_3	C_4	C_5	C_6
S_1	1	2	3			
S_2			4	5		
S_3		6	7		8	
S_4	9		10	11	12	
S_5	13			14		
S_6			15	16	17	
S_7		18	19		20	
S_8			21	22	23	
S_9	24					25
S_{10}			26			

The second comparison is based on a sequence of 26 batches of product derived from industrial production data. In this case, the search space is significantly larger ($26!=4.03*10^{26}$) and the optimal Pareto front is not a priori known. In this case, we choose to evaluate the three algorithms using two recently proposed comparative metrics for MOEA. The first metric is the *average coverage* defined in [11] as follows. Given two sets of solutions X', $X'' \subseteq X$, the function C maps the ordered pair (X',X'') in a number in [0,1] as follows:

$$C(X',X'') = \frac{\left|\{\mathbf{a}'' \in X''; \exists \mathbf{a}' \in X' : \mathbf{a}' \succ \mathbf{a}''\}\right|}{\left|\{X''\}\right|}$$

where the notation $\mathbf{a} \succ \mathbf{b}$ indicates that **a** *covers* **b**, i.e.

$$\mathbf{a} \succ \mathbf{b} \Leftrightarrow \mathbf{a} \succ \mathbf{b} \text{ or } f(\mathbf{a}) = f(\mathbf{b})$$

and $\mathbf{a} \succ \mathbf{b}$ indicates that **a** dominates **b**. In other words, a solution **a** covers a solution **b** if and only if either **a** dominates **b** or **a** and **b** have the same fitness vector. Therefore, the overall coverage obtained with function C provides a clear indication of the relative quality of the compared solution sets. Table IV reports the relative coverage of the three algorithm. The first value in each cell indicates the average coverage on all the possible pairs of solution sets obtained on ten distinct runs of each algorithm.

Table IV. Relative coverage of the three algorithms in the second case study.

	MOGALS	NSGA II	EFGA
MOGALS	-	0,2160 [0,4754]	0,0613 [0,0886]
NSGA II	0,6075 [0,5306]	-	0,1410 [0,0506]
EFGA	0.9319 [1]	0.9188 [1]	-

The value in square brackets indicates the coverage of the *overall solution sets* obtained as the union of the ten solutions sets for each algorithm. The values clearly indicate that, while MOGALS and NSGA-II exhibit comparable performances, the proposed EFGA always converges in a better final front. Another metric often referred in the literature is the Spacing (S). We consider the definition reported in [10].

$$S = \sqrt{\frac{1}{n-1}\sum_{i+1}^{n}(\bar{d} - d_i)^2}$$

where

$$d_i = \min_j \left(\left|f_1^i(x) - f_1^j(x)\right| + \left|f_2^i(x) - f_2^j(x)\right| + \left|f_3^i(x) - f_3^j(x)\right| \right)$$

with $\bar{d} = average\ (d_i)$, i,j=1,…,n; and n is the number of points in the known Pareto-front. The metric S indicates a measure of the spread of points throughout the known Pareto-front. In particular, the lower is the value for this metric, the more uniformly spaced will be the points in the front. A value of zero for S indicates that all members of the front are equidistantly spaced. Table V reports the average (on ten runs of the algorithm) number of solutions belonging in the final front and the relative average Spacing S. Similarly to table IV, the values in square brackets in table V indicate the spacing index computed on the overall solution sets.

Table V. Number of solution in the final front (N) and Spacing indices (S) for the three algorithms.

MOGALS	N	37,9	[49]
	S	1,247	[1,495]
NSGA II	N	35,5	[61]
	S	1,149	[0,963]
EFGA	N	65,8	[79]
	S	0,859	[0,523]

The algorithm NSGA II gives quite low performances in terms of search repeatability. In fact, for each test, the algorithm finds 35.5 (avg.) non-dominated solutions, while the union of the non-dominated solutions obtained in the overall solution set of the algorithm yields a front with 61 elements, i.e. about twice of the average value. Hence, the behavior of NSGA II seems to give a different non-dominated set at each run. From this point of view, the algorithms MOGALS and EFGA present quite better performances. Note that, the Spacing S increases with respect to the average value for MOGALS (1.4949 Vs 1.2466), while for EFGA there is a decrement of S (0.5234 Vs 0.8587) that indicates a good spread of the solutions on the front.

6 Conclusions

In this paper we address a sequencing problem arising in a serial production system in the furniture industry. The need of a multi-objective formulation results in a extremely complex combinatorial problem calling for innovative approach. We tackle the problem with an evolutionary method. The proposed algorithmic approach, called Elitist Front Genetic Algorithm (EFGA), is more general and suitable to be applied to a wide class of sequencing and scheduling problems. EFGA is characterized by an archive dedicated to non-dominated solutions, a particular dynamic mechanism of individuals selection for mating and a local search.

A first campaign of computational experiments indicates the novel approach as a promising multi-criteria evolutionary solver. In fact, EFGA seems perform at least at the same level (but often better) of the current state-of-the-art algorithms in terms of quality and quantity of obtained non-dominated solutions.

References

[1] Agnetis, A., P. Detti, C. Meloni, D. Pacciarelli. Set-up coordination between two stages of a supply chain. *Annals of Operations Research*, 107, 15-32, 2001.

[2] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., A fast and elitist multiobjective genetic algorithm: NSGA-II, *IEEE Trans. on Evol. Comput.*, Vol. 6, n. 2, 182 –197, 2002.

[3] C. Dimopoulos, and A. M. S. Zalzala, "Recent developments in Evolutionary computation for manufacturing optimisation: problems, solutions, and comparisons," *IEEE Trans. Evol. Comp.*, vol. 4, no. 2, pp. 93-113, 2000.

[4] C. M. Fonseca, and P. J. Fleming, "Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms-Part I: A Unified Formulation", *IEEE Transactions on Systems, Man and Cybernetics, Part A*, Vol.28, n. 1, 26-37, 1998.

[5] Ishibuchi, H., Murata, T., "A multi-objective genetic local search algorithm and its application to flowshop scheduling, *IEEE Transactions on Systems, Man and Cybernetics, Part C*, Vol.28, n. 3, 392 –403, 1998.

[6] Ishibuchi, H., Yoshida, T., Murata, T., Selection of initial solutions for local search in multiobjective genetic local search, *IEEE CEC '02, Proceedings of the 2002 Congress on Evol. Comput.*, Vol.1, pp. 950 –955, 2002.

[7] Kazarlis, S.A., Papadakis, S.E., Theocharis, J.B., Petridis, V., "Microgenetic algorithms as generalized hill-climbing operators for GA optimization", *IEEE Trans. on Evol. Comput*, Vol.5, n. 3, pp. 204-217, 2001.

[8] J. Knowles and D. Corne, "The Pareto archived evolution strategy: A new baseline algorithm for multiobjective optimization", *IEEE CEC 99, Proceedings of the 1999 Cong on Ev. Comp.*, Vol.1, 98-105,1999.

[9] Meloni, C., An Evolutionary Algorithm for the Sequence Coordination in Furniture Production, *Lecture Notes of Computer Science*, vol. 2264, pp. 91-106, 2001.

[10] D.A. Van Veldhuizen, G.B. Lamont, "On Measuring multiobjective evolutionary algorithm performance", *IEEE CEC00, Proc. of Congr Evol. Comp.*, pp. 204-211, 2000.

[11] Zitzler, E., Deb, K., and Thiele, L. "Comparison of multiobjective evolutionary algorithms: Empirical results", *Evol. Comput.*, vol. 8, no.2, 173-195, 2000.