

THE NO-WAIT FLOWSHOP PROBLEM WITH SEQUENCE DEPENDENT SETUP TIMES AND RELEASE DATES

PAULO M. FRANÇA GILBERTO TIN JR. LUCIANA BURIOL

Faculdade de Engenharia Elétrica e de Computação – FEEC
Universidade Estadual de Campinas – UNICAMP
C.P. 6101 – 13083-970 – Campinas – SP – Brazil
{franca,tin,buriol@denis.fee.unicamp.br}

Abstract

Population based metaheuristics, such as hybrid genetic algorithms or memetic algorithms, play an important role in the solution of combinatorial optimization problems. The memetic algorithm presented here for the no-wait flowshop scheduling problem addresses a hierarchically organized complete ternary tree to represent the population that putted together with a recombination plan resembles a parallel processing scheme for solving combinatorial optimization problems. We propose a novel recursive local search scheme - Recursive Arc Insertion - (RAI) which is responsible for about 90% of the total processing time of the algorithm. Randomly generated instances are used to test the algorithm against other proposed methods.

Keywords: *Scheduling, Genetic Algorithms, Memetic Algorithms, Flowshop problem*

1 INTRODUCTION

The no-wait flowshop problem is characterized by scheduling jobs on machines such that jobs may not have their execution interrupted on or between machines after they have been started. A no-wait environment arises from the absence of storage capacity between operations of a job or from the processing technology (Hall & Sriskandarajah [1996]). Usually just-in-time production lines and the need to eliminate or reduce inventory costs generate the no-wait restriction. In this work, we also consider the use of time restrictions (setup times and ready times). The objective function is to minimize the makespan, the total processing time spent to process all jobs since the first operation of the first job sequenced until the last operation of the last job sequenced.

The problem consists of a set $m \geq 1$ machines and a set $n \geq 1$ jobs. Every job has exactly one operation per machine. Each job's operation must be processed on only one machine at a time and each machine may only process one operation at a time. The setup time is sequence dependent meaning that it depends on the operation being processed and the preceding operation. The setup time may overlap an operation's processing phase so that the no-wait restriction is satisfied. The first operation of a job may only start after its ready time.

A survey for no-wait flowshop problems has been conducted by Hall and Sriskandarajah [1996] where several practical applications of the problem are shown as well as several variance of the problem found in the literature. Bianco et al [1999] show the reduction of the no-wait flowshop problem, with sequence dependent setup times and ready times, to the Asymmetric Traveling Salesman Problem (ATSP) with time restrictions to visit each city. They also present a lower bound algorithm for the ATSP with time restrictions and prove that the ATSP with sequence dependent setup times is NP-hard.

Practical applications of this problem can be found in the chemical and pharmaceutical industry, in the service industry, and in the metal industry. In the metal industry, the lamination of metals such as aluminum and steel goes through a series of processes to reduce their thickness. First, the metal is heated and then, it goes through a series of operations where no-wait may occur. If the metal waits from one operation to another, it can cool and it may therefore jeopardize the process.

In a no-wait flowshop environment, every job must be processed on all machines in the same order that it is processed on the first machine. Therefore, the no-wait flowshop characterizes a special class of permutational flowshop. The no-wait restriction also causes a delay to be introduced at the start of a job's first operation so that the job may not have its execution interrupted between operations. This delay represents all the delays that the job would have to wait in later operations if its processing phase were started right after the end of the first operation of the preceding job in the permutation.

The no-wait flowshop problem can be reduced to the ATSP in polynomial time. In order to reduce the no-wait flowshop to the ATSP, a dummy job j_1 , with setup times $s_{1j}=s_{j1}=0$, processing times $p_j=0$, and ready time $r_1=0$ for $1 \leq i, j \leq n+1$ and $1 \leq k \leq m$ must be added to guarantee feasibility.

The reduced problem becomes the ATSP with ready times to visit each city (ATSP-RT). In other words, the traveling salesman may only visit a city after its ready time. The ATSP-RT becomes a complete graph $G=(V,A)$ with V being a set with $n+1$ vertices and A the set of arcs. The solution to the problem becomes a Hamiltonian route that starts from city 1, pass only once through each city, and goes back to city 1. The cost of each arc in the graph represents the incurred delay of sequencing job j after job i and it can be calculated as follows:

$$c_{ij} = \max_{1 \leq k \leq m} \left[s_{ij}^k + \sum_{h=1}^k (p_i^h - p_j^h) + p_j^k \right], \quad \forall i, j \in V \cup \{0\} \quad (i \neq j)$$

Figure 1 shows an example of a no-wait flowshop problem with $m=3$, $n=4$, and $r_i=0$ for $1 \leq i \leq n$. Note that c_{ij} represents the cost of the arc from i to j in the ATSP-RT.

Setup times and processing times tables for each machine

		Machine 1				Machine 2				Machine 3							
s_{ij}		1	2	3	4	s_{ij}		1	2	3	4	s_{ij}		1	2	3	4
1		-	0	0	0	1		-	0	0	0	1		-	0	0	0
2		0	-	2	1	2		0	-	1	3	2		0	-	3	2
3		0	2	-	3	3		0	1	-	2	3		0	2	-	1
4		0	4	1	-	4		0	3	2	-	4		0	1	3	-
p_i		0	3	4	5	p_i		0	4	5	4	p_i		0	6	3	4

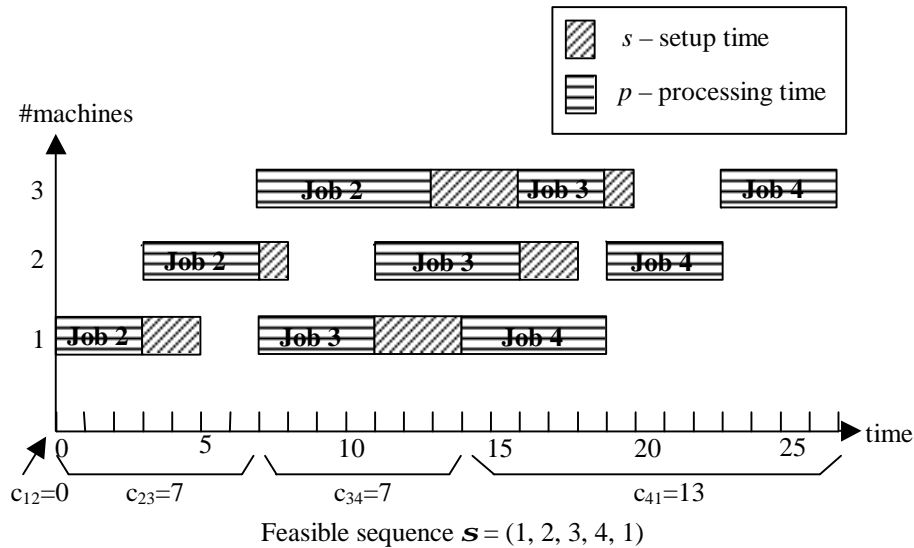


Figure 1. No-wait flowshop problem example with $m=3$ and $n=4$.

2 MEMETIC ALGORITHM APPROACH

Memetic algorithm (MA) is a metaheuristic that belongs to the class of population based algorithms. A population based algorithm uses several individuals to search the solution space of a problem. MA is similar to a genetic algorithm in terms of operators commonly used (as crossover and mutation) to explore the solution space of a problem. Differently from genetic algorithms, the MA uses a local search to improve genotypes. The term “meme” comes from R. Dawkins [1976] and it means a unit of cultural evolution. The “memes” are improved by the individual that holds them different from what happen to genes where no improvement occur through the individual that holds them. Like genes that goes through recombination and mutation, memes also participate in the evolutionary process and therefore replicates itself in the population.

2.1 Population Structure

The population has been organized hierarchically by the objective function value of its individuals as a complete ternary tree of 13 nodes (Figure 2).

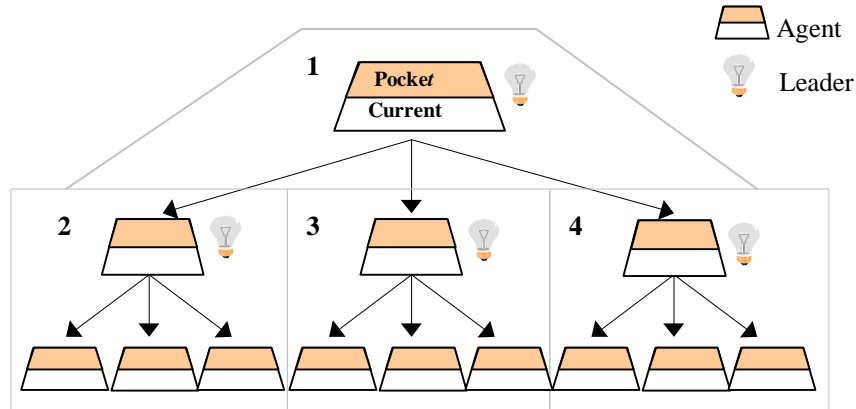


Figure 2. Population organized as a complete ternary tree of 13 agents.

Each node of the tree represents an agent that contains a *pocket* solution and a *current* solution (Moscato and Norman [1992]). The *pocket* solution acts like a long term memory that keeps track of the best individual, according to its objective function value, in the agent. The *current* solution represents the offspring resulted from a recombination process that may involve *current* and *pocket* solutions from other agents. The only solution exposed to mutation is the current solution of an agent.

The population, organized as a ternary tree, results in four sub-populations that can be compared to islands of knowledge about the problem. These islands have agents that compete and cooperate together to solve the problem in a process that resembles a parallel processing system or an ant colony organization. Moreover, the *current* solution of an agent exchanges position with the *pocket* solution every time it has a better objective function value. The population is also organized in such a way that after every generation of the algorithm, the *pocket* solution of every leader is the one with the best objective function value among its supporters and the *pocket* solutions of the supporters remain ordered in no-decreasing order of its objective function value. All the solutions that comprise the population are randomly generated except one that is generated through a nearest neighbor algorithm.

2.2 Recombination Methods

The crossover operators explore the search space of a problem by combining features from different individuals. During the crossover, solutions exchange partial genes/memes to generate an offspring. We studied four crossover operators: Distance Preserving Crossover (DPX), Strategic Arc Crossover (SAX), Ordered Crossover (OX) and Partially

Matched Crossover (PMX). Based on computational experiments we selected the PMX as the best one.

The PMX, differently from the other crossovers, tends to preserve the absolute position of every job. It first selects a fragment of the chromosome from one of the parents and copies it to the offspring. Like the OX crossover, it sequentially fills the offspring with the genes from the other parent, but when it encounters a gene that has already been copied, instead of skipping to the next gene like the OX crossover, it emulates a pair-wise position exchange procedure in the parent and copies the gene that is not present in the offspring (Goldberg [1989]). For instance, consider parent A = < 0 1 3 2 4 5 6 > and parent B = < 5 6 3 0 2 4 1 >. Now, suppose that parent A has a segment < 3 2 4 > chosen to be copied to the offspring. Then, the offspring would be filled with genes of parent B and it would result in < 5 6 3 2 4 0 1 >.

2.3 Cultural Evolution

Every new individual generated through initialization, recombination, or mutation goes through a learning process as to improve its fitness. In general the individual matures when it finds the best solution in its neighborhood, then it is ready for a new generation. For this learning process, we propose a new recursive local search called *Recursive Arc Insertion (RAI)*. Also, for the ATSP-RT problem, the cost of an arc is not simply the cost found in the adjacency cost matrix, but it instead receives the influence of the ready time. This influence of the ready time is only used to calculate the movement of the local search operators or for the nearest neighbor algorithm used for population initialization or for some of the crossovers described. Since the movement value does not reflect the real cost, the object function value must be recalculated before a movement is accepted.

The modified distance can be described as follow:

If s_i , arrival time at city i , is known then

$$d_{ij} = \max(c_{ij}, r_j - s_i), \quad 0 \leq i, j \leq n$$

else

$$d_{ij} = \max(c_{ij}, r_j - r_i), \quad 0 \leq i, j \leq n.$$

The variable s_i represents the time when the salesman arrives at city i . The operators that use this new distance d_{ij} with the ready time influence tends to favor sequencing cities with similar ready times or cities which ready time is closer to the arrival time s_i of the salesman at a previous city.

Recursive Arc Insertion (RAI)

This local search method recursively inserts arcs in points considered critical in the solution. It was first introduced by Buriol et al. [1999] for the ATSP. In this local search, the basic movement is a 3-Opt movement. During the process, a critical point will be connected to one of the best 5 predecessor or successor neighbors from all the vertexes

of the problem. Let's suppose that an arc was inserted between a critical point i and a successor neighbor j , then the algorithm will search for an arc to remove between the successor neighbor and the critical point, and finally completing the 3-Opt movement, the succeeding arc of i and the preceding arc of j is also removed and then the disconnected graph is restored by inserting two new arcs (Figure 3). The removed arc between j and i will be the one that brings the best movement cost. The movement cost is the difference between the cost of the 3 arcs leaving the solution and the 3 arcs that are inserted. After the movement has been done, five new critical points are inserted. The new critical points represent the vertexes of each arc inserted or removed from the solution but the vertex that has already been a critical point and started this movement.

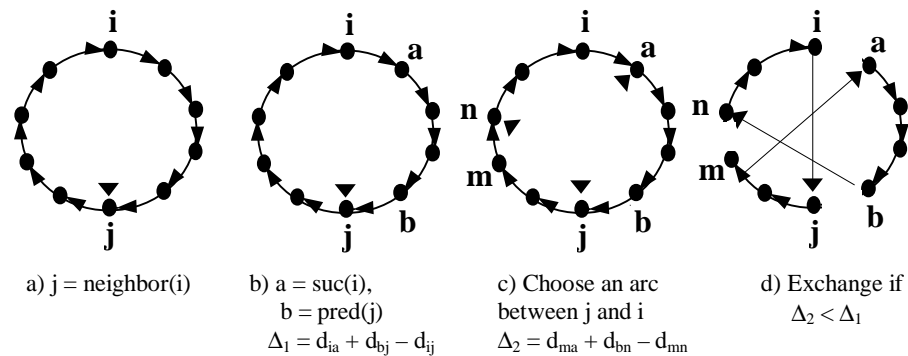


Figure 3. Local Search RAI

For the no-wait flowshop problem, a slight modification is made to this search, instead of considering the best 5 successor or predecessor neighbors from all the vertexes of the solution, it considers the 5 relative successor or predecessor neighbors to the critical point being considered. Each candidate neighbor of a critical point has a probability of being chosen. The probabilities are as follow: 40% for the first, 30% for the second, 15% for the third, 10% for the fourth, and 5% for the fifth relative neighbor (Buriol et al. [1999]).

Let $s = \langle 14 \ 12 \ 9 \ 8 \ 3 \ 4 \ 5 \ 0 \ 1 \ 2 \ 6 \ 10 \ 11 \ 13 \ 15 \rangle$ be a sequence of jobs and 0 be a critical point, then the 5 relative predecessor neighbors of 0 are (4, 3, 8, 9, 12) and the 5 relative successor neighbors of 0 are (2, 6, 10, 11, 13). Note that 5 and 1 are not considered candidate neighbors because they already are neighbors of the critical point 0. Critical points in the solution are identified during the process of crossover and mutation. For the PMX crossover the critical point is defined as the one that have its position switched for being in the crossover segment copied from the first parent.

2.4 Mutation, Diversity, and Stop Criteria

A 3-Opt based mutation is used in the MA. Its contribution to the algorithm is to introduce some noise in a solution so that it may get out from a local optimum basin of attraction and also, to bring some diversity to the population. The probability of it to occur in a solution is of 5%. This probability has been empirically tested and shown to give the best results.

The mutation basically consists in randomly selecting an arc to be inserted between two vertices (i, j) , it will result in the removing of two arcs, and an arc to be removed between j and i . Then, the disconnected graph is reconnected by the insertion of two new arcs completing then the 3-Opt movement. Diversity is maintained in the population by allowing only new solutions to the population to be inserted. The stop criterion is based on the number of generations that the algorithm may run. The number of generations is given by a logarithmic equation. This equation gives a high number of generations for smaller instances and a no much greater number for larger instances. The logarithmic equation is $13 * \log(13) * \log(n)$ where n is the number of jobs and 13 is the number of agents. The algorithm also stops when a local optimum solution is found.

3 COMPUTATIONAL RESULTS

Tests were made using randomly generated instances. The performance of our MA is evaluated through comparisons with two lower bounds, LB1 and LB2, and also with a Best Insertion Heuristic (BIH). The algorithms for both the lower bounds and the BIH heuristic were proposed by Bianco et al. [1999]. The MA run in a AMD-Duron 600MHZ with 128MB of memory. Instances were generated for $m=2, 5$, and 10 machines and $n=10, 50$, and 100 jobs. The ready times were uniformly distributed between $[1, R_{max}]$ where $R_{max}=50, 100, 200, 300, 400$, and 500. Instances with low R_{max} takes to a congested state where several jobs are released to the shop at the same time. The processing times and the setup times were chosen from a uniformly distributed range between $[5, 10]$ and $[1, 5]$, respectively. Five instances were generated for each combination of machine number, job number, and ready time interval. For each parameter combination, we show an average result and the worst value of these five instances between parentheses. We ran the MA 10 times for each instance and its best value is compared to $\max(\text{LB1}, \text{LB2})$ and to the heuristic.

Tables 3 through 5 show the results for 2, 5, and 10 machines, respectively. IniPop is the percentage error of the initial population from the lower bound, Qual.1 is the percentage deviation of the MA from the lower bound, Qual.2 is the percentage deviation of the BIH heuristic from the lower bound, Time is the average time in seconds that the algorithm takes to find the optimal solution, and CPU LS is the time percentage that the algorithm spends running the local search method.

Table 3. Computational results for the MA compared to lower bounds and to the BIH heuristic applied to randomly generated instances with 2 machines

#Mach.	#Jobs	R _{max}	IniPop (%)	#Gen.	Qual.1 (%)	Qual.2 (%)	Time (sec.)	CPU LS (%)
2	10	50	6.70	76.00	3.08(5.04)	5.23(8.74)	0.75	76.10
2	10	100	8.84	76.00	6.65(8.33)	7.78(11.67)	0.66	78.16
2	10	200	0.96	76.00	0.96(2.78)	1.41(2.78)	0.66	75.71
2	10	300	1.07	76.00	0.93(1.88)	1.05(2.50)	0.62	75.56
2	10	400	0.28	76.00	0.28(0.56)	0.28(0.56)	0.66	77.64
2	10	500	0.00	76.00	0.00(0.00)	0.00(0.00)	0.60	75.28
2	50	50	6.26	130.00	1.35(1.81)	3.45(3.65)	17.16	94.05
2	50	100	6.17	130.00	1.82(2.70)	3.71(4.28)	17.04	94.39
2	50	200	7.09	130.00	3.18(4.54)	5.91(6.63)	17.32	94.10
2	50	300	9.07	130.00	5.71(7.80)	7.56(8.45)	16.45	94.31
2	50	400	9.07	130.00	8.61(10.1)	8.85(10.21)	13.63	92.92
2	50	500	7.01	130.00	6.06(8.64)	5.57(7.00)	13.80	93.60
2	100	50	5.66	153.00	1.57(2.25)	3.26(3.50)	71.17	95.88
2	100	100	6.17	153.00	1.72(2.13)	3.54(3.90)	72.37	96.42
2	100	200	5.93	153.00	2.14(2.91)	3.72(3.82)	75.86	96.88
2	100	300	5.88	153.00	2.74(3.61)	4.53(5.01)	78.96	96.69
2	100	400	6.41	153.00	3.11(3.73)	4.79(5.01)	72.22	96.54
2	100	500	6.46	153.00	4.69(6.29)	5.42(5.71)	70.74	96.41
Average			5.60	116.41	2.96	4.29	26.00	88.23

Qual.1 = ((memetic alg. – lower bound)/lower bound)*100

Qual.2 = ((BIH – lower bound)/lower bound)*100

Table 4. Computational results for the MA compared to lower bounds and to the BIH heuristic applied to randomly generated instances with 5 machines

#Mach.	#Jobs	R_{\max}	IniPop (%)	#Gen.	Qual.1 (%)	Qual.2 (%)	Time (sec.)	CPU LS (%)
5	10	50	10.66	76.00	4.90(8.33)	5.53(8.33)	0.83	77.25
5	10	100	5.91	76.00	4.92(7.02)	5.60(8.55)	0.68	72.62
5	10	200	2.30	76.00	1.92(4.39)	2.20(4.39)	0.63	75.23
5	10	300	0.20	76.00	0.07(0.33)	0.07(0.33)	0.67	71.80
5	10	400	1.02	76.00	1.02(2.04)	1.02(2.04)	0.85	71.65
5	10	500	0.00	76.00	0.00(0.00)	0.00(0.00)	0.82	72.68
5	50	50	5.32	130.00	1.00(1.57)	2.94(3.32)	17.50	93.83
5	50	100	7.08	130.00	1.87(2.62)	3.26(4.23)	17.06	94.07
5	50	200	7.00	130.00	2.80(3.96)	4.89(6.13)	17.07	94.13
5	50	300	9.26	130.00	4.87(8.37)	7.03(9.16)	16.08	93.84
5	50	400	12.51	130.00	9.96(12.7)	10.92(13.1)	15.07	93.80
5	50	500	9.40	130.00	8.80(10.06)	8.23(9.09)	14.92	93.40
5	100	50	6.64	153.00	2.04(2.71)	3.77(4.27)	79.27	96.58
5	100	100	6.65	153.00	1.78(2.21)	3.89(4.24)	74.74	96.36
5	100	200	7.80	153.00	2.29(2.61)	4.02(4.35)	81.95	96.89
5	100	300	11.42	153.00	6.79(15.39)	8.94(17.27)	81.76	96.82
5	100	400	6.15	153.00	2.93(3.66)	4.12(4.98)	70.94	96.56
5	100	500	8.02	153.00	4.35(4.77)	5.03(5.28)	71.21	96.50
Average			6.44	116.41	3.25	4.38	27.02	87.28

Table 5. Computational results for the MA compared to lower bounds and to the BIH heuristic applied to randomly generated instances with 10 machines

#Mach.	#Jobs	R_{max}	IniPop (%)	#Gen.	Qual.1 (%)	Qual.2 (%)	Time (sec.)	CPU LS (%)
10	10	50	5.69	76.00	3.37(4.76)	4.33(7.14)	0.70	77.28
10	10	100	7.47	76.00	4.98(7.46)	6.01(8.96)	0.68	77.95
10	10	200	0.96	76.00	0.96(3.88)	1.50(3.88)	0.63	72.94
10	10	300	0.31	76.00	0.31(1.53)	0.31(1.53)	0.60	73.67
10	10	400	1.58	76.00	1.21(1.80)	2.57(3.59)	0.62	74.73
10	10	500	0.00	76.00	0.00(0.00)	0.00(0.00)	0.59	68.14
10	50	50	6.47	130.00	1.40(2.14)	3.06(3.57)	16.99	94.08
10	50	100	7.02	130.00	1.99(2.85)	3.34(4.01)	17.62	94.30
10	50	200	8.46	130.00	2.89(4.14)	4.69(5.95)	16.68	94.13
10	50	300	9.32	130.00	4.00(6.60)	5.59(7.30)	17.21	94.14
10	50	400	14.54	130.00	10.04(15.51)	10.38(14.26)	16.05	94.21
10	50	500	14.13	130.00	13.21(16.01)	12.93(14.29)	14.22	93.98
10	100	50	6.06	153.00	1.45(1.74)	2.89(3.01)	73.45	96.51
10	100	100	5.74	153.00	1.56(2.02)	3.14(3.49)	70.65	96.50
10	100	200	6.72	153.00	2.16(2.73)	3.21(3.64)	69.24	96.48
10	100	300	6.22	153.00	2.47(3.03)	3.98(4.96)	67.02	96.37
10	100	400	7.46	153.00	2.98(3.84)	4.59(4.67)	69.96	96.43
10	100	500	7.84	153.00	3.56(4.42)	4.80(5.14)	71.47	96.52
Average			6.22	116.41	2.96	4.04	25.20	87.64

As R_{max} gets higher, the instances tend to become easier because the optimal solution gets closer to a simple ordering of the ready times. This can be noted for instances with 10 jobs and 2, 5, and 10 machines with $R_{max}=(400, 500)$ where the MA and the BIH heuristic find the optimal solution.

For instances with 50 and 100 jobs, the quality of the BIH heuristic gets closer to the MA when R_{max} goes from 50 to 500. The BIH heuristic is better than the MA for instances with 50 jobs and $R_{max}=500$. This behavior of the MA can be explained by the influence of the ready times when calculating the movements in the local search method.

In general, the MA is better than the BIH heuristic in 44 instances, it finds the same result as the BIH heuristic in 7 instances, and it is worse than the heuristic in 3 instances.

4 CONCLUSIONS

We showed how to solve the NWFSP problem with time restrictions, setup times and ready times using a MA. The main contributions of our approach is the structured population organized as a ternary tree and the novel local search scheme called RAI.

The tests showed that our MA is efficient in solving instances with a congested state, (closer ready times) and presented some difficulty in solving instances with sparse ready times. Comparisons with another heuristic approach showed that the MA is capable of finding better solutions while spending greater computational effort. The MA spent most of its time in the local search method. Since the local search plays an important role in the overall process, it is clear that further study can be done in improving performance of the local search method for this problem or finding new methods with similar or better quality.

ACKNOWLEDGEMENTS

This research was partially supported by Fundação de Amparo à Pesquisa do Estado de S. Paulo - FAPESP - and Conselho de Desenvolvimento Científico e Tecnológico - CNPq -, Brazil. We thanks S. Giordani for providing us with the data of the instances, and the BIH and lower bound solutions used in the computational tests.

REFERENCES

- Bianco, L., Dell'Olmo, P., and Giordani, S. (1998). Flow shop no-wait scheduling with sequence dependent setup times and release dates, *INFOR* **37(1)**, pp. 3-19.
- Buriol, L. S., França, P., and Moscato, P. (1999). Recursive Arc Insertion: A New Local Search Embedded in a Memetic Algorithm for the Asymmetric Traveling Salesman Problem, *Journal of Heuristics*, to appear.
- Dawkins, R. (1976). *The selfish gene*, Oxford University Press, Oxford.
- Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley.
- Hall, N.G. and Sriskandarajah, C. (1996). A survey of machine scheduling problems with blocking and no-wait in process, *Operations Research* **44**, 510-525.
- Moscato, P. & Norman, M.G. (1992). A Memetic Approach for the Traveling Salesman Problem Implementation of a Computational Ecology for Combinatorial Optimization on Message-Passing Systems, in M. Valero, E. Onate, M. Jane, J. L. Larriba, B. Suarez (eds.). *Parallel Computing and Transputer Applications*, IOS Press, Amsterdam, pp. 177-186.