Proceedings of the 6th WSEAS Int. Conf. on EVOLUTIONARY COMPUTING, Lisbon, Portugal, June 16-18, 2005 (pp15-23)

Improving on Excellence. An Evolutionary Approach.

J. P. CALDEIRA[1,3]
[1] I.P.S. – E.S.T.
R. Vale de Chaves-Estefanilha
2810 Setúbal
PORTUGAL

F. MELICIO [2, 3]
[2] I.S.E.L.
R. Conselheiro Emidio Navarro
1900 Lisboa
PORTUGAL

A. ROSA [3]
[3] LASEEB-ISR-IST
Av. Rovisco Pais, 1, TN 6.21
1049-100 Lisboa
PORTUGAL

Abstract: - In this paper, we present a new hybridization method that, under certain conditions, can be used to improve results obtained by the best existing algorithms for a particular problem. The proposed hybrid uses Evolutionary Algorithm (EA) with a population of algorithms, to simultaneously evolve problem solutions and individual algorithm parameters. As an example of this approach we describe the details of its application to the Job Shop Problem (JSP) and use an EA to enhance results obtained by one of the most successful algorithms for this problem – Nowicki and Smutnicki's "Taboo Search Algorithm with back jump" (TSAB) [12]. The new algorithm not only improved TSAB's results but also improved the best known results for several well known benchmark problems.

Key-Words: - Hybrid, Evolutionary, Taboo, Job Shop, Scheduling

1 Introduction

It has become quite common to use evolutionary hybrids especially due to their improved performance in relation to the basic evolutionary algorithm. Usually hybrids are combined with a simple Local Search Algorithm (LSA) to reduce the EA's search space to the set of local minimum. These hybrid's rely on the EA to guide the population to promising regions in this reduced search space, therefore the LSA's do not have any mechanisms to escape local minimum. Their role is limited to finding the closest local minimum as efficiently as possible. Although this kind of hybridization makes the EA's much more competitive they can not really compete with algorithms like TSAB. There are two main reasons for this difference in performance:

- Neighborhood
- Solution Evaluation

TSAB uses a very small neighborhood with a high probability of improvement. The neighborhood resulting from the application of the EA's variation operators is much larger neighborhood with a lower improvement probability. This means that the EA will need more evaluations to reach a solution of similar quality.

TSAB also evaluates every solution in a neighborhood very efficiently. Since neighbors are generated making small changes to the original solution, their value can be calculated simply by determining how a particular change will impact the value of the original solution. This means that to evaluate a neighbor, you do not start from scratch but instead take a look at what was changed between the two solutions. In an EA we can not take this approach because the solutions resulting from crossover operators differ greatly from the parents, so every evaluation has to be done from scratch.

We therefore choose a different approach to hybridization, one that will take full advantage of TSAB's speed and performance. Although TSAB is a deterministic algorithm, if it is run with a different initial solution or with different parameters, different results can be obtained. The objective of this paper is to describe an evolutionary hybrid that achieves better results that the basic multi-start strategy given equal computational resources. Our hybrid is simply a EA with a population of TSAB's with randomly generated parameters and initial solutions. All elements of the population run in parallel and from time to time, some elements are randomly chosen and cross their best solutions and algorithm parameters.

We also describe several strategies to make our hybrid more efficient allocating more computational resources to more promising elements of the population.

2 Job Shop Problem

Job Shop scheduling is a well known scheduling problem. It is NP-hard [3] and one of the most intractable combinatorial problems.

The terminology of scheduling theory derives from the processing and manufacturing industries. We shall thus talk about jobs and machines, even tough, in some cases, the objects referred to bear little resemblance to either jobs or machines. The structure of the general job-shop problem fits many different scheduling problems in many diverse fields as well as those in industry.

In the basic job-shop problem JSP [5] we have *n* Jobs $\{J_1, J_2, ..., J_n\}$ to be processed on *m* Machines $\{M_1, M_2, ..., M_m\}$. Each job is processed on each machine once and only once. The processing of a job on a machine is called an operation. Operation *Oij* represents the processing of the *i* th job on the *j* th machine. The technological constraints specify the order in which each job has to be processed on the machines. Each job has its own processing order that may have no relation to the processing order of other jobs.

Each operation *Oij* is performed in a certain amount of time – processing time *Pij*. This time is known in advance and assumed fixed. The time needed to transport a job to a machine and to set up that machine for *Oij* is considered to be included in *Pij*. We shall also assume that the machines not processing any job are always available. The time at which a Job *Ji* becomes available for processing is called its ready time ri.

The general JSP is a static and deterministic optimization problem. It is static because the number of jobs and their ready times are known and fixed. It is deterministic since processing times and remaining parameters are known and fixed.

To solve the JSP we must find a sequence in which jobs are processed on each machine that is:

- Feasible i.e. Compatible with the technological constraints
- Optimal with respect to some objective function or performance criteria.

The most widely used objective is to find feasible schedules that minimise the completion time of the total production program, normally referred to as makespan (Cmax).

Since each job is partitioned into the operations processed on each machine, a schedule for a certain problem instance consists of an operation sequence for each machine involved. These operation sequences can be permuted independently from each other. We therefore have a total of $(n!)^m$ different sequences (feasible and infeasible).

In the general JSP a number of assumptions are made. Some were mentioned explicitly above, others were implicit. A complete list of these assumptions is found in [5].

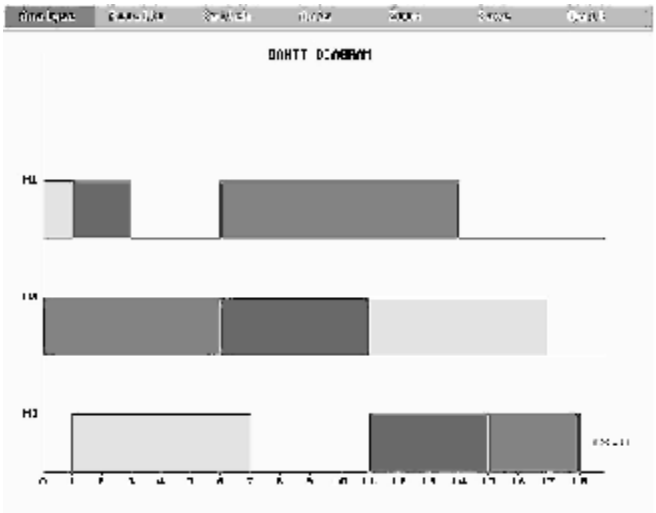

Fig. 1 – The Gantt Diagram

The diagram in Fig. 1 is a Gantt Diagram [5] of a schedule with 3 jobs and 3 machines and shows when each operation is processed on each machine.

3 The Taboo Algorithm

The Taboo Search Algorithm (TSA) was proposed by Glover [7]. A TSA tries to keep track of the visited search space to avoid revisiting the same solutions and thus improving the efficiency of search.

Taboo Search can be seen as an extension of the basic Hill Climbing Algorithm. The same basic concepts of Movement and Neighborhood apply. A "movement" is a procedure that partially changes a solution, transforming it into another, known as its neighbor.

The Neighborhood, N(Sc), is the set of solutions obtained when all possible "moves" are applied to the current solution Sc. The TSA stores visited solutions in a list. This list is known as the Taboo List, T, because in the TSA it is forbidden to return to solutions in this list. In each step, the TSA chooses the best, unforbidden neighbor to be the current solution of the next iteration. For this reason, the TSA is also known as the "Steepest Decent, Mildest Ascent". The flowchart of the basic TSA is shown in fig. 2.

Numerous authors have applied TSA to the JSP. Their approaches differ mainly in the Neighborhood Definition, the generation of the Initial Solution and in the maintenance of the Taboo List.

One of the best algorithms for solving the JSP is Nowicki and Smutnicki's Taboo Search Algorithm with back jump (TSAB) [12].

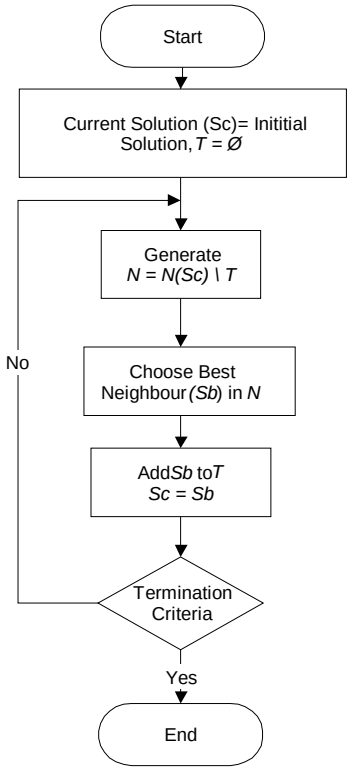

Fig. 2 – The flowchart of the basic TSA

TSAB uses an acyclic graph representation. An example of this representation for the schedule in the Gantt diagram presented previously is:

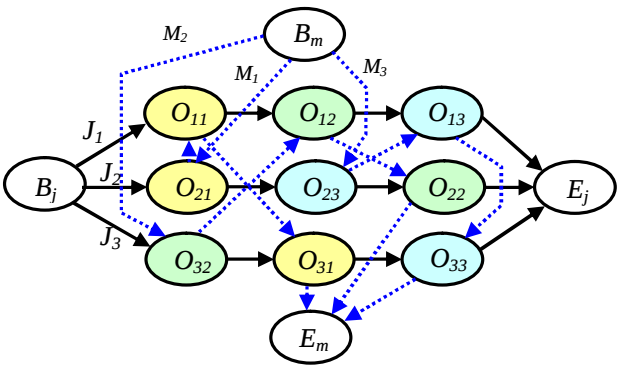

Fig. 3 – The Acyclic Graph

In fig. 3:

- Continuous arcs represent the technological constraints.
- Dotted arcs represent the order of operations on each machine
- Each arc is weighed with the processing time of the operations from which it originates.

We attribute TSAB's success mainly to two factors:

- Neighborhood
- Back jump Mechanism (long term memory)

The neighborhood used is known as N_1 [9]. It is a small neighborhood of highly correlated feasible solutions with a high probability of improvement. These characteristics make N_1 an excellent choice.

The back jump mechanism makes the TSA much more effective allowing the use of information from previous runs later on. Instead of starting with an empty taboo list and some kind of heuristic initial solution, TSAB returns to the promising points of previous runs and tries a different route.

TSAB's main parameters are:

- *BestSol* – Best Solution found at that time (Coded as an Acyclic Graph)
- MaxInter – Maximum Number of Iterations without improvement
- *MaxT* – Maximum size of the Taboo List
- *MaxL*- Maximum size of Back jump List

4 The Evolutionary Algorithm

A good implementation of an Evolutionary Algorithm for any problem implies a careful choice of representation and appropriate variation operators. This is especially true for the JSP due its technological constraints. Naive choices easily result in the production of infeasible sequences.

We chose to use one of the most successful representations for the JSP called Permutation with Repetition (PWR) initially proposed by Bierwirth [1].

In this representation, the order of operations within the permutation is interpreted as a sequence for building a schedule solution. The decoding procedure scans each permutation from left to right and uses sequence information to build a schedule consecutively. The simple permutation of operations normally leads to infeasible schedules because an operation can only be scheduled if its predecessors have already been scheduled. This problem can be avoided using permutations with repetition in which operations are represented by the identifier of the job they belong to. The solution of the JSP with n = 3 and m = 3 shown previously in fig. 1 can be represented as: [2,3,1,2,3,1,2,1,3].

The k th time job *Ji* appears in the permutation refers to the k th operation of job *Ji*.'s technological constraints. Since we make use of the technological constraints to decode permutations, no infeasible solutions will result.

This representation therefore covers all feasible sequences for any JSP instance but none of the infeasible category. There is however a large redundancy in this representation. The total amount of different permutations is: $PwR = \frac{(m.n)!}{m!^n}$ (permutation with repetition) which is much larger than the number of possible sequences $NR = n!^m$. If we take into consideration that permutation with repetition only cover feasible sequences and that these are only a small subset of the possible sequences we have an idea of the magnitude of the redundancy.
This redundancy can be seen in table 1:

N	2	3	5	10	10
M	2	3	5	10	15
NR	4	216	2.5E+10	4E+65	2.5E+98
PwR	6	1680	6.2E+14	2.4E+92	4E+141

Table 1: Comparison between representations

The variation operators consist of a multi-crossover and multi-mutator.
The multi-crossover randomly selects a crossover among Generalized Order Crossover (GOX), Generalized Position Crossover (GPX) and Uniform Crossover (GUX) [9]. All these operators produce valid permutations with repetition while preserving the relative order of the operations within both parents.
The GOX was proposed by Bierwirth [1]. We start by choosing a subsequence of the father chromosome. The operations in this subsequence are removed from the mother chromosome respecting operation indices. We insert the subsequence in the position where we found the first operation deleted from the mother chromosome.

Father	3	2	2	2	3	1	1	1	3
Index	1	1	2	3	2	1	2	3	3

Mother	~~1~~	1	3	2	~~2~~	1	~~2~~	~~3~~	3
Index	1	2	1	1	2	3	3	2	3

GOX	1	3	2	2	2	3	1	1	3
GPX	1	3	2	2	3	1	2	1	3

Fig. 4. GOX and GPX Crossover

In fig. 4, the chosen subsequence is shaded in the father chromosome. It consists of two operations of Job 2 (indices 2 and 3), one operation of Job 3 (index 2) and one operation of Job 1 (index 2). These operations are crossed out in the mother chromosome.
In GOX we conclude by inserting the subsequence in the position where the first operation was found in the mother chromosome (circled operation). In the GPX, we maintain the position of the substring in the father.
The GUX operator in a normal uniform crossover in which a child is obtained by randomly selecting genes from the parent chromosomes. Selected genes are eliminated in both parents. This procedure is repeated until both parents are empty.
The multi-mutator randomly selects a mutation operator amongst Position Based Mutator (PBM), Order Based Mutator (OBM) and Swap Based Mutator (SBM) [9]. The PBM deletes a randomly chosen gene from the chromosome and reinserts it in a randomly chosen position (fig. 5).

Parent	3	1	2	2	2	3	1	1	3
Child	1	2	2	3	2	3	1	1	3

Fig. 5 - PBM Mutator

The OBM simply swaps values of two randomly chosen genes and SBM swaps values of randomly chosen adjacent genes.
We use a multi-crossover and multi-mutator because they increase diversity and improve the EA's efficiency [4].

5 The Evolutionary-Taboo Algorithm

The comparison of the results shows that a pure EA's (without local search) such as [1], are far from being competitive with other algorithms such as TSAB [12]. Including some local search greatly improves the EA's performance and results, but even hybrid algorithms [9] are not very competitive.
Current hybridization approaches use very simple local search heuristics, leaving most of the computation time to the EA.
We propose a different hybridization model in which the EA is used more as a meta-heuristic. The use of a Meta–Heuristic will slow down the algorithm we choose to hybridize but as long as the original algorithm is fast and improved results are obtained it will still be useful and practical.
We can compensate the loss of speed taking advantage of the fact that EA's are easy to parallelize and to distribute over a local network Using distribution the time required to run a hybrid EA can be greatly reduced.
Our EA will evolve a population of independent TSAB algorithms. Each element of the population is

initialized with a random initial solution and parameters.

The TSAB parameters that are evolved are those considered to have a greater influence on the algorithm's speed and results. In TSAB these are *MaxT*, *MaxL*, *MaxIter* and *BestSol*. Since the EA operators and TSAB use different representations every time an operator is applied, the chromosomes involved must first have their solutions encoded into permutation with repetition and results have to be decoded back into acyclic graphs as represented in fig. 6.

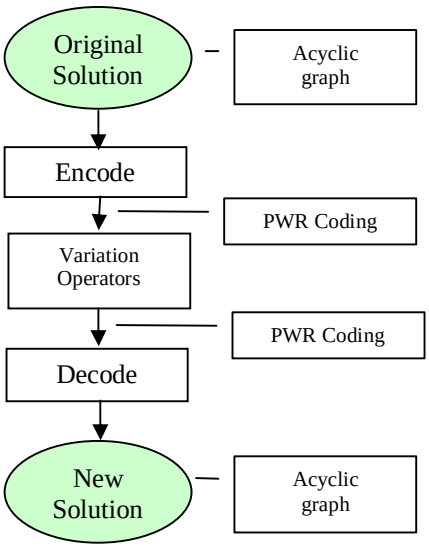

Fig. 6 – Generation of new solutions

The reduced use of the crossover and mutation operators ensures that the encoding and decoding procedures will not be computationally expensive.

We will now go into the different parts of the EA in more detail. Our hybrid algorithm is a Generational EA with a replacement percentage of 5 to 10% and the population size is 30. A tournament selector is used to select the two parents. The child resulting from the multi-crossover is then subjected to mutation according to the mutation probability.

To make new elements more competitive with the rest of the population, they are subjected to an additional number of TSAB iterations before joining the population. A tournament selector that returns the worst individual is used to select the chromosomes that will be replaced by the new individuals.

As usual, iterations of the local search procedure TSAB are performed during the evaluation step of the EA. In our algorithm, the evaluation policies are responsible for deciding who is evaluated and how many TSAB iterations are performed on each individual.

The flow chart of the basic operation of the Evolutionary Taboo Algorithm is shown in fig. 7.

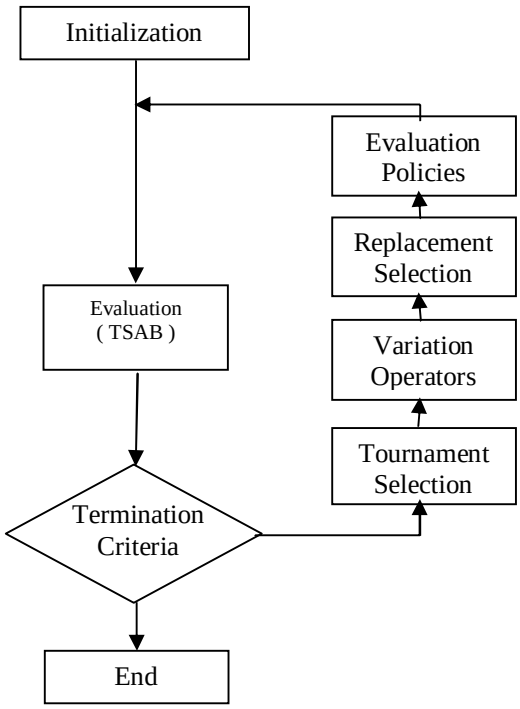

Fig. 7 – The flowchart of Evolutionary-Taboo Algorithm

Taking a closer look at some of the EA components, we have:

Chromosome: The chromosome consists of TSAB parameters (*MaxT*, *MaxL* and *MaxIter*), a solution *BestSol* (the best solution it found) and NumIter (number of iterations to be done in the next evaluation). The parameters that are changed by the variation operators are *BestSol*, *MaxT*, *MaxL* and *MaxIter*.

Initialization: The initial solution for TSAB is obtained by generating random permutations with repetition. The remaining parameters are initialized as generating Gaussian values centered on the default values of each parameter. Every chromosome has NumIter set to InitIter (Initial amount of iteration performed on each individual). Example:

MaxT = Gaussian(DefaultMaxT, MaxTVariance)

Crossover: The TSAB solutions (*BestSol*) are crossed using the crossover operators described for permutations with repetition. The average of the remaining parent parameters are used for the child and NumIter is incremented by CxIter (Number of iterations performed on the child produced).

Example: $$MaxT = \frac{MaxT_1 + MaxT_2}{2}$$

Mutator: The TSAB solutions are mutated using the mutators for permutations with repetition. The remaining parameters are also mutated using a Gaussian centered on previous parameter values, i.e.

MaxT = Gaussian(MaxT, MaxTVariance)

MaxT = Max(MinMaxT, MaxT)

Every parameter has a minimum value below which values cannot fall. This keeps parameter values in an acceptable range. Mutated chromosomes get an additional *MutIter* iterations.

Evaluator: The evaluation of a chromosome consists of performing a certain number of TSAB iterations (*NumIter*). This number is determined by the evaluation policies.

Evaluation Policies: In Evolutionary Taboo Algorithm, we found the need to apply several evaluation policies. The order in which the policies are applied is as follows:

- Only New
- Improve Best
- Random Eval
- Mutate Equals
- KillTsa

These evaluation policies will be described in more detail in the following section.

Only New: This policy marks the new elements resulting from the variation operators for evaluation.

Improve Best: This policy tries to improve the efficiency of the hybrid algorithm. To this end, the 5 individuals with the lowest makespan in which the TSAB algorithm has still not come to an end, have their iteration number increased by Num1, Num2, Num3, Num4 e Num5 respectively. In our implementation there is an exponential drop in values from Num1 to Num5, but other strategies can also be applied.

Random Eval: This policy allows any of the population elements to be iterated. Once again, it tries to attribute more computational resources to the better individuals. This is done by sorting the population by increasing makespans and then generating a random number (Rn) between zero and population size. All individuals ranking below Rn are iterated with an additional EvalIter Iterations i.e.:

NumIter = NumIter + EvalIter

Mutate Equals: This policy is responsible for eliminating duplicates that may show up in the population that would lead a repeated search of the same areas in the search space (TSAB is deterministic). Thus, to avoid wasting computational resources covering the same areas, solutions are compared two by two and when the disparity between them falls below a certain threshold (DisparThresh), one of the solutions is mutated. The Disparity Function measures the degree of similarity between solutions, according to the formula:

$$Disparity = \sqrt{\sum_i \left[Sol_1 OpPos\ (i) - Sol_2 OpPos\ (i)\right]^2}$$

$Sol_1 OpPos\ (i)$ – *position of operation i in machine sequence of solution* 1

Once again in mutated chromosomes:

NumIter = NumIter + DefaultMutIter

KillTsa: To maintain the same number of active individuals in the population, we increase the size of the population by one every time a TSAB algorithm ends its execution i.e. reaches *MaxIter* iterations without improvement and has no more states stored in its back jump list. To avoid the population growing indefinitely, we limit the number of terminated TSAB's to a maximum of MaxTSA, beyond which the inactive chromosome with the greatest makespan is mutated. Again:

NumIter = NumIter + DefaultMutIter

6 Results

As stated before the Evolutionary-Taboo algorithm will be less computationally efficient than TSAB. This should be obvious since we have a population of TSAB algorithms running in parallel. What we want to prove is that an evolutionary hybrid is better than a multi-start strategy. To compare these algorithms we performed 200 runs of the TSAB, initialized in the same way as in the hybrid (with random parameters and initial solution). Each run is performed until TSAB reaches its termination criteria. The problem instances selected are well known benchmark problems [8] on which TSAB has more difficulty in producing high quality results.

The results in table 2 show that, contrary to what authors suggested [12], a multi-start strategy resulted in a significant improvement in results (2.7% on average) and for yn1 the best know makespan was lowered to 886.

For the comparison to be fair, both algorithms must be given an equal amount of computational resources. In table 3, the hybrid was run 20 times per problem instance for 100 generations. Although the total number of generations in table 2 surpasses that of table 3 for most problem instances, results of the hybrid are normally better than those of the multi-start algorithm. The hybrid also has a lower standard deviation and average value.

We can also see that the average value of iterations per evaluation is 14585 which show that most of the computational resources are used on TSAB.

When we increased the number of generations, the results of the Evolutionary-Taboo Hybrid surpassed our greatest expectations, improving the best known makespans for many problem instances (table 4)
If we take into consideration that for some problem instances only one run was needed to improve the best know makespan, it should be relatively easy to further improve some of the results shown in table 4.

The parameters used where:

- popsize = 30 (Population size)
- repPerc = 0.08 (Replacement Percent)
- cross prob = 0.4 (Crossover Prob.)
- mut prob = 0.75 (Mutation Prob.)
- cxiter = 2500 (Crossover Iterations)
- mutiter = 1000 (Mutation Iterations)
- inititer = 2500 (Initial Iterations)
- evaliter = 500 (RandomEval Iter.)
- toursize = 2 (Tournament Size)
- numgen = 2500 (Number of Generations)
- NUM1 = 1050 (Improve Best Param.)
- NUM2 = 750
- NUM3 = 450
- NUM4 = 300
- NUM5 = 150

7 Conclusion

The evolutionary hybrid is better than a multi-start strategy. If we take into consideration that the TSAB algorithm was already one of the best algorithms for the JSP, we were pleased that the hybrid took it even further improving not only its results but also the best-known results for several problem instances.
Although the hybrid requires much more computational resources, this is not a serious problem because it can be easily distributed over several computers greatly reducing the time needed to produce high quality results..
We expect that the same methodology will work on other scheduling problems provided the local search algorithm is fast. It could be worthwhile to try a different representation for the EA. Although the PWR coding works well by itself, its high redundancy suggests that it is not ideal for use in this kind of hybrid architecture. We would also like to implement the island model to try to further improve the algorithms results and efficiency.

Problem Instance	Lower Bound	Best Known Make Span	n	m	Single TSAB			Multi-Start TSAB					
					Best Make Span	Dist to Best %	Num. Of Iterations	Best Make Span	Average Make Span	Stardard Deviation%	Average Num. of Iterations	Total Iterations	Dist to Best %
swv09	1604	1663	20	15	1793	7.8	63 373	1695	1777	1.9%	168 454	33 690 825	1.9%
swv11	2983	3005	50	10	3199	6.5	152 529	3060	3226	3.1%	231 943	46 388 569	1.8%
swv06	1591	1696	20	15	1768	4.3	55 568	1732	1789	1.9%	159 048	31 809 593	2.1%
swv12	2972	3038	50	10	3161	4.1	80 623	3078	3261	3.1%	221 932	44 386 344	1.3%
swv04	1450	1483	20	10	1541	3.9	101 195	1496	1543	1.5%	118 181	23 636 153	0.9%
yn1	826	888	20	20	921	3.7	66 965	886	905	0.8%	176 939	35 387 817	-0.2%
swv05	1421	1434	20	10	1486	3.6	24 473	1458	1528	2.2%	114 197	22 839 344	1.7%
swv13	3104	3146	50	10	3251	3.3	98 558	3184	3303	2.0%	181 759	36 351 878	1.2%
swv07	1446	1620	20	15	1669	3	121 385	1649	1716	1.5%	159 025	31 804 906	1.8%
yn2	861	909	20	20	934	2.8	72 230	909	929	0.9%	162 945	32 589 058	0.0%
swv01	1392	1418	20	10	1449	2.2	11 291	1439	1478	1.4%	108 354	21 670 761	1.5%
la29	1142	1153	20	10	1177	2.1	43 683	1164	1180	0.9%	84 540	16 908 052	1.0%
					Avg	3.94%			Avg	1.8%		Avg	1.25%

Table 2 – Results using Multi-start TSAB

Problem Instance	Lower Bound	Best Known Make Span	n	M	EvolutionaryTaboo Hybrid							
					Best Make Span	Average Make Span	Std. Dev. %	Num. of Evals	Average Num. of Iter.	Total Iterations	Iteration / evaluations	Dist ao Melhor %
swv09	1604	1663	20	15	1682	1704.7	0.8%	92	1 331 502	26 573 088	14 473	1.14%
swv11	2983	3005	50	10	3044	3104.4	0.7%	91	1 327 057	26 541 148	14 527	1.30%
swv06	1591	1696	20	15	1705	1721.4	0.5%	91	1 317 484	26 349 683	14 407	0.53%
swv12	2972	3038	50	10	3098	3154.9	0.9%	92	1 330 605	26 612 100	14 510	1.97%
swv04	1450	1483	20	10	1487	1497	0.4%	91	1 340 538	26 810 754	14 683	0.27%
yn1	826	888	20	20	889	894.15	0.3%	92	1 360 294	27 205 887	14 730	0.11%
swv05	1421	1434	20	10	1445	1454.1	0.3%	93	1 333 471	26 669 418	14 377	0.77%
swv13	3104	3146	50	10	3155	3194.8	0.9%	92	1 320 562	26 411 247	14 370	0.29%
swv07	1446	1620	20	15	1640	1664	0.8%	93	1 340 624	26 812 484	14 454	1.23%
yn2	861	909	20	20	909	913.75	0.3%	93	1 362 551	27 251 022	14 722	0.00%
swv01	1392	1418	20	10	1428	1435.2	0.3%	93	1 370 974	27 419 486	14 679	0.71%
la29	1142	1153	20	10	1162	1167.3	0.2%	92	1 386 030	27 720 597	15 098	0.78%
							0.5%				14585.8	0.76%

Table 3 – Results using Evolutionary Hybrid (100 Generations)

Prob.	LB	Best Known Make Span	n	m	#o	TSAB			EvolutionaryTaboo Hybrid			
						Best Make Span	Dist to Best %	Num. of Iter.	Best Make Span	Mean Make Span	Num. of Iterations	MakeSpan Improvement %
swv09	1604	1663	20	15	300	1793	7.82	63 373	1661	1664.4	63 769 734	0.1%
swv11	2983	3005	50	10	500	3199	6.46	152 529	2987	2998.3	75 229 511	0.6%
swv15	2885	2940	50	10	500	3121	6.16	78 023	2913	2924	295 460 198	0.9%
abz7	656	656	20	15	300	687	4.73	55 414	658	658	67 494 609	-0.3%
swv03	1369	1398	20	10	200	1459	4.36	8 473	1403	1409	118 875 055	-0.4%
swv06	1591	1696	20	15	300	1768	4.25	55 568	1695	1695	74 487 227	0.1%
swv12	2972	3038	50	10	500	3161	4.05	80 623	3023	3031.6	124 861 250	0.5%
swv04	1450	1483	20	10	200	1541	3.91	101 195	1476	1476	54 151 075	0.5%
yn1	826	888	20	20	400	921	3.72	66 965	886	887	168 178 730	0.2%
swv05	1421	1434	20	10	200	1486	3.63	24 473	1431	1432.5	97 386 922	0.2%
swv13	3104	3146	50	10	500	3251	3.34	98 558	3105	3105	275 907 453	1.3%
swv07	1446	1620	20	15	300	1669	3.02	121 385	1608	1608	270 187 563	0.7%
yn2	861	909	20	20	400	934	2.75	72 230	906	906.5	123 356 300	0.3%
swv01	1392	1418	20	10	200	1449	2.19	11 291	1417	1417.5	53 773 767	0.1%
la29	1142	1153	20	10	200	1177	2.08	43 683	1153	1157.1	21 333 804	0.0%

Table 4 – Results using Evolutionary Hybrid

References:

[1] Bierwirth C,"A Generalized Permutation Approach to Job Shop Scheduling with Genetic Algorithms ", *OR Spectrum*, Vol 17, 89-92, 1995

[2] Bierwirth C., MattFeld D., "Production Scheduling and Rescheduling with Genetic Algorithms", *Evolutionary Computation*, Spring 1999, Volume 7, Number 1

[3] Blazewicz, J., Domschke,W., and Pesch, E. (1996), "The Job-Shop scheduling problem: Conventional and new solution techniques.", *European Journal of Operations Research*, 22:25-40

[4] Eiben A., Hinterding R., Michalewicz Z., "Parameter control in Evolutionary Algorithms" Technical Report 98–07 *Rijksuniveriteit te Leiben Vakgroep Informatica*, 1997

[5] French S., "*SEQUENCING AND SCHEDULING : An Introduction to the Mathematics of the Job-Shop*", John Wiley & Sons 1986

[6] Giffler B., Tompson G. "Algorithms for solving production scheduling problems.", *Operations Research*,1960, 8:487-503

[7] Glover F. E. Taillard, and D. de Werra, "A Users Guide to Taboo Search", *Annals of Operation Research.* 41 (1993), 3-28

[8] Imperial College Management School, *OR-Library*, http://mscmga.ms.ic.ac.uk/info.html

[9] MattFeld D., "*Evolutionary Search and the Job Shop - Investigations on genetic algorithms for production scheduling* ", Spriger Verlag 1996

[10] Melicio, F., Caldeira, J.P., Rosa, A.C., ”Timetabling implementation aspects by Simulated Annealing”. *IEEE-ICSSSE'98*, Bei*Ji*ng, 1998.

[11] Michalewicz, Z., “*Genetic Algorithms + Data Structures = Evolution Programs*”. Springer-Verlag, 1994.

[12] Nowicki E., C. Smutnicki, "A fast Taboo Search Algorithm for the Job Shop Problem ", *International Jornal of Management Science* Vol. 42 Nº6, June 1996

[13] Taillard, E., “Parallel Taboo Search Techniques for the Job Shop Scheduling Problem”, *ORSA Journal on Computing*, Vol. 6, Nº2, Spring 1994