A MULTIRECOMBINATIVE EVOLUTIONARY APPROACH TO SOLVE THE PARALLEL TASK SCHEDULING PROBLEM

ESQUIVEL S.C., GATICA C. R., GALLARD R.H.

Proyecto UNSL-338403[1]
Departamento de Informática
Universidad Nacional de San Luis (UNSL)
Ejército de los Andes 950 - Local 106
5700 - San Luis, Argentina.
E-mail: {esquivel, crgatica, rgallard}@unsl.edu.ar
Phone: + 54 2652 420823
Fax : +54 2652 430224

Abstract

Allocation of the components (tasks) of a parallel program to processors in a multiprocessor or a multicomputer system take full advantage of the computational power provided by the system. Evolutionary approaches has been used in the past to implement efficiently this type of scheduling. Those approaches showed their advantages when contrasted against conventional approaches and different chromosome representations were proposed. Latest improvements in evolutionary computation include multirecombinative variants allowing multiplicity of crossovers on the selected couple of parents. Multiple crossovers per couple (MCPC) exploits good parents' features in the creation of offspring. Performance enhancements were clearly demonstrated in single and multicriteria optimization under this approach.

This paper shows three algorithms to solve the problem of allocating a number of non-identical related tasks in a multiprocessor or multicomputer system. The model assumes that the system consists of a number of identical processors and only one task may execute on a processor at a time. All schedules and tasks are non-preemptive. This involves the assignment of partially ordered tasks onto the system architecture processing components. Two evolutionary algorithms using a direct representation, are contrasted with the well-known Graham's [12] list scheduling algorithm (LSA). The first one makes use of the conventional single crossover per couple (SCPC) approach while the second, following current trends in evolutionary computation, uses (MCPC) a multirecombinated approach. Chromosome structure, genetic operators, experiments and results are discussed.

Key words: Parallel task allocation, Evolutionary algorithm, multirecombination, List Scheduling Algorithm, Optimization.

---

[1] The Research Group is supported by the Universidad Nacional de San Luis and the ANPCYT (National Agency to Promote Science and Technology).

1. INTRODUCTION

A parallel program is a collection of tasks, some of which must be completed before than others begin. The precedence relationships between tasks are commonly delineated in a directed acyclic graph known as the *task graph*. Nodes in the graph identify tasks and their duration and arcs represent the precedence relationship. Factors, such as number of processors, number of tasks and task precedence make harder to determine a good assignment. The problem to find an schedule on $m > 2$ processors of equal capacity, that minimizes the whole processing time of independent tasks has been shown as belonging to the NP-complete class [13].

In a deterministic model, the execution time for each task and the precedence relations between them are known in advance. This information is depicted in a directed graph, usually known as the task graph. In Fig. 4 we have six task graphs with the corresponding duration and their precedence relations. A schedule is an allocation of tasks to processors which can be depicted by a Gantt chart, to see Fig.1. In a Gantt chart, the initiation and ending times for each task running on the available processors are indicated and the makespan (total execution time of the parallel program) of the schedule can be easily derived. By simple observation of the Gantt chart the completion time of the last task abandoning the system, called the makespan, the processors utilization, the speed up and other performance measures can be easiy determined. Connected with the makespan, an optimal schedule is one such that the total execution time is minimized. Other performance variables, such as individual processor utilization or evenness of load distribution can be considered. As we can see some simple scheduling problems can be solved to optimality in polynomial time while others can be computationally intractable.

As we are interested in the scheduling of arbitrary tasks graphs onto a reasonable number of processors we would be content with polynomial time scheduling algorithms that provide good solutions even though optimal ones can not be guaranteed.

In the following sections we discuss conventional and evolutionary algorithms to solve this problem on a set of selected instances.

2. THE LIST SCHEDULING ALGORITHM (LSA)

For a given list of tasks ordered by priority, it is possible to assign tasks to processors by always assigning each available processor to the first unassigned task on the list whose predecessor tasks have already finished execution.

Let be:

$T=\{T_1,....,Tn\}$ a set of tasks,

$\mu: T \rightarrow (0, \infty)$ a function which associates an execution time to each task,

$\preceq$ a partial order in T and

L a priority list of tasks in T.

Each time a processors is idle, it immediately removes from L the first ready task; that is, an unscheduled task whose ancestors under $\preceq$ have all completed execution. In the case that two or more processors attempt to execute the same task, the one with lowest identifier succeed and the remaining processors look for another adequate task. Using this heuristic, contrary to the intuition, some anomalies can happen. For example, increasing the number of processors, decreasing the execution times of one or more tasks, or eliminating some of the precedence constraints can actually increase the makespan. We are seeking for heuristics which are free from these anomalies.

3. USING EVOLUTIONARY ALGORITHMS TO PROVIDE NEAR-OPTIMAL SOLUTIONS

The task allocation problem has been investigated by many researchers [3], [7], [9], [10], [11], [14], [15]. Several heuristics methods has been proposed, such as mincut-based heuristics, orthogonal recursive bisection, simulated annealing, genetic algorithms and neural networks. From the representation perspective many evolutionary computation approaches to the *general scheduling problem* exists. With respect to solution representation these methods can be roughly categorized as *indirect* and *direct* representations [1]. In the case of indirect representation of solutions the algorithm works on a population of encoded solutions. Because the representation does not directly provides a schedule, a schedule builder is necessary to transform a chromosome into a schedule, validate and evaluate it. The schedule builder guarantees the feasibility of a solution and its work depends on the amount of information included in the representation. In direct representation [2], a complete and feasible schedule is an individual of the evolving population. The only method that performs the search is the evolutionary algorithm because the represented information comprises the whole search space.

We devised different evolutionary computation approaches to task scheduling. First we addressed two different representation schemes: direct and indirect. Further work addresses the question of attempting to improve performance by means of different recombination and mating approaches. This is the subject of this contribution

3.1. DIRECT REPRESENTATION OF SOLUTIONS

Here we propose to use a schedule as a chromosome. Suppose we have two different schedules, (a) and (b) (Fig. 1), represented by the following Gantt charts.

P_2			T_3	T_4		T_6				
P_1	T_1		T_2			T_5			T_7	T_8
Ts	1	2	3	4	5	6	7	8	9	10

Schedule (a)

P_2			T_3		T_5						T_8
P_1	T_1		T_2		T_4		T_6			T_7	
Ts	1	2	3	4	5	6	7	8	9	10	11	12

Schedule (b)

Fig. 1. Feasible schedules represented by Gantt charts.

The precedence relation described in the associated task graph can be properly represented in the corresponding precedence matrix A, where element a_{ij} is set to 1 if task i precedes task j and it is set to 0 otherwise. A gene in the chromosome is the following four-tuple:

$$<task\_id, proc\_id, init\_time, end\_time>$$

where,
task_id, identifies the task to be allocated.
proc_id, identifies the processor where the task will be allocated.

init_time, is the commencing time of the *task_id* in *proc_id.*
end_time, is the termination time of the *task_id* in *proc_id.*

With this structure the list of the corresponding predecessors tasks is easily retrieved by entering the column of A indexed by the *task_id* value. The corresponding chromosomes C_a and C_b for schedules (a) and (b) are:

C_a :	1,1,0,2	2,1,2,5	3,2,2,3	4,2,3,5	5,1,5,8	6,2,5,8	7,1,8,9	8,1,9,10
C_b:	1,1,0,2	2,1,2,5	3,2,2,3	4,1,5,7	5,2,5,8	6,1,7,10	7,1,10,11	8,2,11,12

Fig 2. Two chromosomes specifying schedules corresponding to Gantt charts of figure 1

The use of conventional crossovers such as one-point crossover should generate invalid offspring. chromosomes. To avoid this problem repair algorithms attempt to build up a valid solution from an invalid one. This approach is embedded in the *knowledge-augmented crossover* operator proposed by Bruns. Here a *collision* occurs if an operation (task processing) inherited from one of the parents cannot be scheduled in the specified time interval on the assigned processor. In this case the processor assignment is unchanged and it is delayed into the future until the processor is available. We adopted an ASAP crossover approach similar to the Brun's proposal but modified because delays are avoided, moving the assignment to the earliest possible time, by random selection of one available processor at the ready time of the unassigned task. In this way no processor will remain idle if a task is available to be executed and the precedence constraints are satisfied. The available processor is selected as to minimize assignment changes in the second parent part of the offspring. In our example if we decide to apply this operator after the fifth position this decision provides only one alternative and would give us the following chromosomes and their corresponding schedules, which differs from their parents only in the assignments of tasks T_7 and T_8.

$C_{a'}$:	1,1,0,2	2,1,2,5	3,2,2,3	4,2,3,5	5,1,5,8	6,2,5,8	7,1,8,9	8,2,9,10
$C_{b'}$:	1,1,0,2	2,1,2,5	3,2,2,3	4,1,5,7	5,2,5,8	6,1,7,10	7,1,10,11	8,1,11,12

P_2			T_3	T_4		T_6				T_8
P_1	T_1		T_2			T_5			T_7	
ts	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*

Schedule (a')

P_2			T_3			T_5					
P_1	T_1		T_2			T_4		T_6		T_7	T_8
ts	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	11	12

Schedule (b')

Fig 3. Feasible offspring schedules for the model task graph (ASAP).

A similar operator was conceived for mutation. If the chromosome undergoes mutation then a search is done, from left to right, until one gene is modified either by choosing an alternative free processor or

by moving the assignment to the earliest possible time. This would imply modifying subsequent genes of the chromosome to create a valid offspring (*valid mutation*).

4. THE EVOLUTIONARY APPROACH

Previous evolutionary approaches to the parallel tasks scheduling problem considered conventional recombination method known as single crossover per couple (SCPC) [8]. New trends in Evolutionary Computation point to multirecombination. In earlier works [5], [6], we devised a different approach: to allow multiple offspring per couple, as often happens in nature. In order to deeply explore the recombination possibilities of previously found solutions, we decided to conduct several experiments in which more than one crossover operation for each mating pair was allowed. The number of children per couple was fixed or granted as a maximum number and the process of producing offspring was controlled, for each mating pair, in order not to exceed the population size. The idea of multiple children per couple was tested on a set of well-known testing functions (De Jong functions F_1, F_2 and F_3 [4], Schaffer F_6 [17] and other functions). Best quality results were obtained allowing between 2 and 4 crossovers per couple. These effects were a consequence of a greater exploitation of the recombination of good, previously found, solutions.

5. EXPERIMENTS AND RESULTS

Current experiments implemented evolutionary algorithms with *direct* representation of chromosomes, randomized initial population of size fixed at 50 individuals. Twenty series of ten runs each were performed on 10 testing cases, using elitism. ASAP crossover under SCPC and MCPC, and valid mutation were used. In the case of MCPC, tests with 2, 3 and 4 crossovers were run and after the multiple crossover operation the best created child was selected for insertion in the next generation. The maximum number of generations was fixed at 100, but a stop criterion was used to accept convergence when after 20 consecutive generations, mean population fitness values differing in $\varepsilon \leq 0.001$ were obtained. Probabilities for crossover and mutation were fixed at conventional values of 0.65 and 0.001, respectively. The testing cases corresponded to:

Case 1: Task graph G1 (7 tasks and 3 processors)
Case 2: Task graph G2 (9 tasks and 3 processors)
Case 3: Task graph G2 (9 tasks and 4 processors)
Case 4: Task graph G3 (9 tasks and 3 processors, decreasing task's duration)
Case 5: Task graph G4 (9 tasks and 3 processors, eliminating precedence constraints)
Case 6: Task graph G5 (10 tasks and 2 processors)
Case 7: Randomly generated task graph G6 (13 tasks and 3 processors)
Case 8: Randomly generated task graph (25 tasks and 5 processors)
Case 9: Randomly generated task graph (50 tasks and 5 processors)
Case 10: Randomly generated task graph (25 tasks and 5 processors)

In figure 4 shows the task graphs associated with the first 6 cases. The task graphs associated with the remaining cases are too complex to be inserted in this presentation but are available for any request.

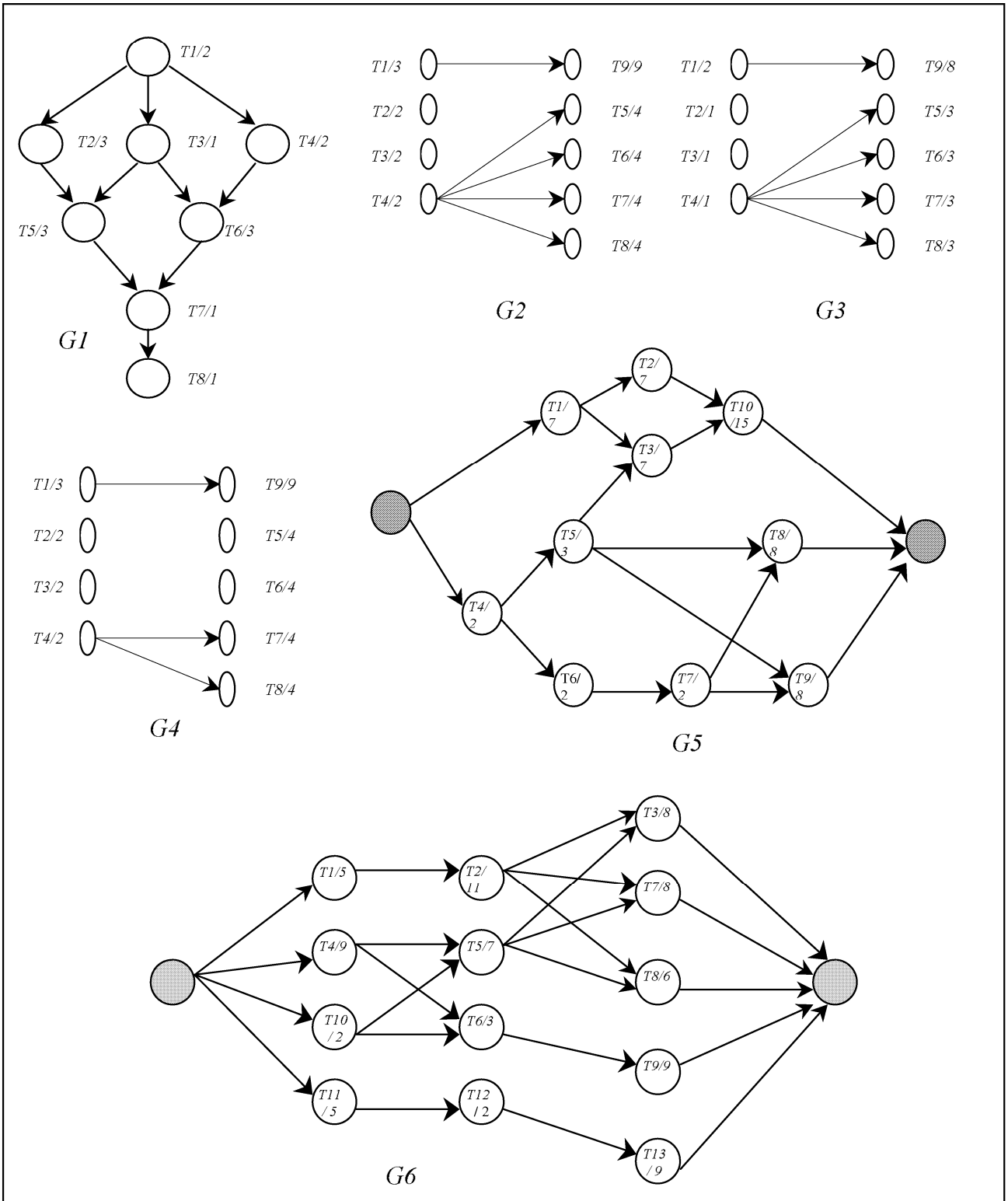

Fig. 4. Some of the tasks graphs used for testing

Opposite to other scheduling problems as Flow shop or Job shop, after an intensive search in the literature we could find few benchmarks.
The first 6 cases were extracted from the literature [12], [16] and they have known optima. Cases 7 to 10 were generated by random assignment of relations and tasks duration. Their optimum values are unknown. Nevertheless for these cases initial trials were run to determine the best quasi-optimal solution under the whole set of contrasted heuristics. This value will be referred in what follows as an estimated optimal value (the best known value assumed as optimal). Diverse performance variables were considered to contrast the algorithms. To measure *versatility* of the algorithms we used:

Alt: Number of alternative solutions. It is the mean number of distinct alternative solutions found by the algorithm in one run including optimum and non-optimum solutions.

Opt: Number of optimal solutions. It is the mean number of distinct optimum or quasi-optimum solutions found by the algorithm in one run.

Topt: Total number of optima. It is the total number of distinct optimal solutions found by the algorithm throughout all the runs.

To measure the *quality* of solutions provided by the algorithm we used:

Ebest = (Abs(*opt_val* - best value)/*opt_val*)100

It is the percentile error of the best found individual in one run when compared with the known, or estimated, optimum value *opt_val*. It gives us a measure of how far are we from that *opt_val*.

ME : Mean Ebest. It is the mean value of Ebest, throughout all runs.

BE : Best Ebest. It is the best Ebest found throughout all runs.

Best M : It is the objective (makespan) value of the best found individual in one run.

MBM : Mean Best Makespan. It is the mean value of Best MS.

GBM : Global Best Makespan. It is the best makespan found throghout all runs (opt_val).

Case	Alt			Opt			Topt		
	LSA	SCPC	MCPC	LSA	SCPC	MCPC	LSA	SCPC	MCPC
1	1	20.9	20.5	1	20.9	20.5	1	194	181
2	1	5.5	3.3	1	1.1	2.1	1	11	21
3	1	5.3	5.9	-	5.3	5.9	-	53	59
4	1	3.1	4.4	-	2.9	4.2	-	29	76
5	1	3.2	2.8	-	1.2	1.4	-	12	14
6	1	8.3	5.8	-	0.1	0.4	-	1	4
7	1	9.9	7.2	-	0	0.1	-	0	1
8	1	19.6	20.4	-	0	0.1	-	0	1
9	1	30.1	20.0	1	0	0	1	0	0
10	1	30.7	13.6	-	0.1	0	-	1	0
Average	1	**13.66**	10.39	0.3	3.16	**3.47**	0.3	30.1	**35.7**

Table 1. Versatility of evolutionary approaches versus LSA

Regarding *versatility*, average results in table 1 shows that as expected SCPC provides a greater number of alternative solutions (Alt) due to its intrinsic genetic diversity. Regarding the mean number of optimal solutions found in a run (Opt) MCPC provides a slight superior average behaviour than SCPC, and also found a greater number of distinct optimal solutions in the whole experiment (Topt).

Regarding *quality* of results table 2 shows that MCPC outperforms SCPC in each considered performance variable (ME, BE, MBM, GBM) and the average global best makespan throughout all instances of the problem (136.1) is the nearest to the average optimal value (132.6) provided by any of the contrasted heuristics. In only one case LSA outperformed the evolutionary approaches, but

providing only one solution. These facts indicate that the multirecombinated evolutionary algorithm is a good alternative to find good quality of results.

Case	ME		BE		MBM		GBM			Opt Value
	SCPC	MCPC	SCPC	MCPC	SCPC	MCPC	LSA	SCPC	MCPC	
1	0.0	0.0	0.0	0.0	9	9	9	9	9	9
2	2.5	3.33	0.0	0.0	12.3	12.4	12	12	12	12
3	0.0	0.0	0.0	0.0	12	12	13	12	12	12
4	0.0	0.0	0.0	0.0	10	10	15	10	10	10
5	2.5	4.16	0.0	0.0	12.3	12.5	16	12	12	12
6	8.06	4.51	0.0	0.0	33.5	32.4	38	31	31	31
7	9.66	9.33	6.66	0.0	32.9	32.8	33	32	30	30
8	10.55	11.61	6.47	0.0	341.6	344.9	375	329	309	309
9	8.27	6.81	4.06	5.07	639.9	631.3	591	615	621	591
10	13.7	8.80	0.0	1.61	352.5	337.3	412	310	315	310
Average	5.52	**4.85**	1.72	**0.67**	145.6	**143.46**	151.4	137.2	**136.1**	132.6

Table 2. Evolutionary approaches versus LSA. Quality of solutions provided.

A more detailed analysis on each run detected about the versatile that in most of the cases alternative solutions do not include, or include a low percentage, of non-optimal alternative solutions. That means that the final population is composed of many replicas of the optimal solutions due to a loss of diversity. This fact stagnates the search and further improvements are difficult to obtain. To avoid this behaviour it would be necessary to continue experimentation with different parameter settings.

6. CONCLUSIONS

The allocation of a number of parallel tasks in parallel supporting environments, multiprocessors or multicomputers, is a difficult and important issue in computer systems. In this paper we approached allocation attempting to minimize makespan. Other performance variables such as individual processor utilization or evenness of load distribution can be considered. As we are interested in scheduling of arbitrary tasks graphs onto a reasonable number of processors, in many cases we would be content with polynomial time scheduling algorithms that provide good solutions even though optimal ones can not be guaranteed. The list scheduling algorithm (LSA) satisfies this requirement.

Two variants of evolutionary algorithms were undertaken SCPC and MCPC with direct representation, to contrast their behaviour with the LSA. Under this representation the above described ASAP crossover approach showed its effectiveness. Preliminary results on the selected test suite lead to the following conclusions. Evolutionary algorithms are versatile: supplying not a single, but a set of optimal solutions, providing fault tolerance when system dynamics must be considered. Moreover, they are free of the LSA anomalies. Regarding quality of results evolutionary algorithms provide in most cases better results and show a better average behaviour. When we compare their performance it is clear that those approaches including multirecombination behave better than the conventional ones. The algorithm does not guarantee finding optimal solutions for any arbitrary task graph, but show a better approach to the problem and the possibility of improvements through enhanced evolutionary algorithms. Further research is necessary to investigate potentials and limitations of evolutionary approaches to the parallel task scheduling problem.

7. ACKNOWLEDGEMENTS

We acknowledge the cooperation of the project group for providing new ideas and constructive criticisms. Also to the Universidad Nacional de San Luis, and the ANPCYT from which we receive continuous support.

8. REFERENCES

[1] Bagchi S., Uckum S., Miyabe Y., Kawamura K. – *Exploring problem-specific recombination operators for job shop scheduling*- Proceedings of the Fourth International Conference on Genetic Algorithms, pp 10-17, 1991.

[2] Bruns R. – *Direct chomosome representation and advanced genetic operators for production scheduling*. Proceedings of the Fifth International Conference on Genetic Algorithms, pp 352-359, 1993.

[3] Cena M.,Crespo M., Gallard R., - *Transparent Remote Execution in LAHNOS by Means of a Neural Network Device*- ACM Press, Operating Systems Review , Vol. 29, Nr. 1, pp 17-28, January 1995.

[4] De Jong K. A. - *Analysis of the Behavior of a Class of Genetic Adaptive Systems* - PhD Dissertation, University of Michigan, 1975.

[5] Eiben A.E., Raué P-E., and Ruttkay Zs., *Genetic algorithms with multi-parent recombination*. In Davidor, H.-P. Schwefel, and R. Männer, editors, Proceedings of the 3rd Conference on Parallel Problem Solving from Nature, number 866 in LNCS, pages 78-87. Springer-Verlag, 1994.

[6] Eiben A.E., van Kemenade C.H.M., and Kok J.N., *Orgy in the computer: Multi-parent reproduction in genetic algorithms.* In F. Moran, A. Moreno, J.J. Merelo, and P. Chacon, editors, Proceedings of the 3rd European Conference on Artificial Life, number 929 in LNAI, pages 934-945. Springer-Verlag, 1995.

[7] Ercal F.- *Heuristic approaches to Task Allocation for parallel Computing*- Doctoral Dissertation, Ohio State University, 1988.

[8] Esquivel S, Gatica C, Gallard R, *"Representation Schemes in Evolutionary algorithms for Parallel Task Scheduling Problem"* Second International ICSC Symposium on Engineering of Intelligent System (EIS'2000) University of Paisley, Scotland, U.K. 29 de Junio - 02 Julio del 2000.

[9] Flower J., Otto S., Salama M. – *Optimal mapping of irregular finite element domains to parallel processors*- Caltech C3P#292b, 1987.

[10] Fox G. C. – *A review of automatic load balancing and decomposition methods for the hipercube*- In M Shultz, ed., Numerical algorithms for modern parallel computer architectures, Springer Verlag, pp 63-76, 1988.

[11] Fox G.C., Kolawa A., Williams R. – *The implementation of a dynamic load balancer* - Proc. of the 2nd Conf. on Hipercube multiprocessors, pp 114-121, 1987.

[12] Graham R. L. – *Bounds on multiprocessing anomalies and packing algorithms*.- Proceedings of the AFIPS 1972 Spring Joint Computer Conference, pp 205-217, 1972.

[13] Horowitz E. and Sahni S – *Exact and approximate algorithms for scheduling nonidentical processors* – Journal of the ACM, vol. 23, No. 2, pp 317-327, 1976.

[14] Kidwell M. – *Using genetic algorithms to schedule tasks on a bus-based system*. - Proceedings of the Fifth International Conference on Genetic Algorithms, pp 368-374, 1993.

[15] Mansour N., Fox G.C. – *A hybrid genetic algorithm for task allocation in multicomputers*. Proceedings of the Fourth International Conference on Genetic Algorithms, pp 466-473, 1991.

[16] Pinedo M., *Scheduling: Theory, Algorithms and Systems,* Prentice Hall international series in industrial and systems engineering, 1995.

[17] Schaffer J., Caruana R., Eshelman R., Das R.- *A study of control parameters affecting online performance of genetic Algorithms for function optimization* - In Third International Conference on Genetic Algorithms, pages 51 - 60, 1989.