EVOLUTIONARY APPROACHES WITH MULTIRECOMBINATION FOR THE PARALLEL MACHINE SCHEDULING PROBLEM*

Susana C. Esquivel, Claudia R. Gatica, Raúl H. Gallard
Proyecto UNSL-338403
Departamento de Informática
Universidad Nacional de San Luis
5700 – San Luis, Argentina
{esquivel, crgatica, rgallard}@unsl.edu.ar

Abstract

Parallel machine scheduling, also known as parallel task scheduling, involves the assignment of multiple tasks onto the system architecture's processing components (a bank of machines in parallel). Parallel machine scheduling is important from both the theoretical and practical points of view [3], [8], [12], [13], [14], [16]. From the theoretical viewpoint, it is a generalization of the single machine scheduling problem. From the practical point of view permits to take full advantage of the processing power provided by resources in parallel.

Two basic models involving m machines and n jobs are the foundations of more complex models: In the first problem the jobs are allocated according to resource availability following some allocation rule. In the second one, besides that, jobs are subject to precedence constraints. The completion time of the last job to leave the system, known as the makespan (C_{max}), is one of the most important objective functions to be minimized, because it usually implies high utilization of resources. These problems, minimizing the makespan, are known in the literature [21] as the unrestricted parallel machine scheduling ($Pm||C_{max}$) and the parallel machine scheduling with job precedence constraints ($Pm|prec|C_{max}$). Both problems are NP-hard for $2 \leq m \leq n$, and conventional heuristics have been developed to provide acceptable schedules as solutions.

Evolutionary algorithms (EAs) have also been used to solve scheduling problems, [8], [17], [19], [22], [23]. Dealing with $Pm|prec|C_{max}$ [9], we found outstanding behaviour of an earlier EA when contrasted against Graham's [15] well-known list scheduling algorithm (LSA),

This paper proposes a multirecombination scheme to solve both parallel machine scheduling problems as above delineated. For $Pm|prec|C_{max}$ the new EA proposed here was contrasted with the earlier EA and for $Pm||C_{max}$ it was contrasted against the longest processing time first (LPT) heuristic. Evidence of the improved behaviour of both evolutionary algorithms is given. Experiments and results are discussed.

1. Introduction

In what follows we deal with the problem of allocating a number of nonidentical tasks in a parallel system consisting of a number of identical processors. Only one task may be executed on a processor at a time and all schedules and tasks are nonpreemptive.

A clear example of the unrestricted parallel machine scheduling, minimizing the makespan, is giving in a multiprocessor system when n independent jobs are to be processed and the system's goal is to maximize throughput.

A clear example of the parallel machine scheduling with job precedence constraints, minimizing the makespan (and consequently the total running time), is giving in parallel computing when we try to find an optimal allocation of the parallel program components, some of which must be completed before others begin. Precedence relations between them are known in advance and depicted in a directed graph, usually known as the *task graph*.

To solve both problems by means of evolutionary approaches we propose to combine *indirect-decode* representations of chromosomes [1] and multiplicity of crossovers and parents [10]

* The research group is supported by the Universidad Nacional de San Luis and the ANPCYT (National Agency to Promote Science and Technology).

0-7695-0810-3/00 $10.00 © 2000 IEEE

2. The Pm| prec|C_{max} problem

Here we briefly describe former experiments performed to deal with this problem. As we said LSA was contrasted with an evolutionary approach using a proposed indirect-decode representation. Some aspects on implementation and results are depicted now. Further details can be found in [9].

2.1. 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 priority list whose predecessor tasks have already finished execution.
Let us denote

- $T=\{T_1,....,Tn\}$ a set of tasks,
- μ: $T \rightarrow (0, \infty)$ a function which associates an execution time to each task,
- $\leq$ a partial order in T and
- L a priority list of tasks in T.

Each time a processor is idle, it immediately removes from L the first ready task; that is, an unscheduled task whose ancestors under $\leq$ have all completed execution. In the case that two or more processors attempt to execute the same task, the one with lowest identifier succeeds 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. In his work Graham presented various examples using the same priority task list.

2.2. The evolutionary approach

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 (Bagchi et al, 1991 [1], Bruns R. 1993 [2]).

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. We decided to use an *indirect-decode* representation. Under this representation a schedule is encoded in the chromosome in a way such that the task indicated by the gene position is assigned to the processor indicated by the corresponding allele, as indicated in Fig.1.

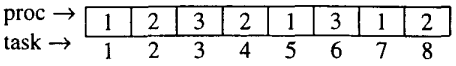

Fig. 1. Chromosome structure for the task allocation problem

The idea is to use a decoder. A *decoder* is a mapping from the space of representations, which are being evolved to the space of feasible solutions which are being evaluated [20]. Here the chromosome gives instructions to a decoder on how to build a feasible schedule. Regarding the task allocation problem, to build a feasible schedule, a decoder is instructed in the following way: "By following the priority list, traverse the chromosome and assign the corresponding task to the indicated processor as soon as the precedence relation is fulfilled". One advantage of decoders resides in their ability to create valid offspring even by means of simple conventional operators making unnecessary the use of repair algorithms or penalty functions [18]. One disadvantage is a slower evaluation of solutions.

The preliminary experiments, denoted as EA1, were restricted to a single crossover (SCPC) operation applied on couples of parents to create offspring and randomized initial population of size fixed at 50 individuals. Twenty series of ten runs each were performed on five testing cases, using elitism. 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.

Five testing cases extracted from the bibliography [15] were chosen:

1. 7 tasks, 3 processors and a precedence relationship R1.
2. 9 tasks, 3 processors and precedence relationship R2.
3. 9 tasks, 4 processors and precedence relationship R2 (incrementing the number of processors).
4. 9 tasks, 3 processors and precedence relationship R2, (but decreasing task's duration).
5. 9 tasks, 3 processors and precedence relationship R3 (eliminating precedence constraints).

The task graphs corresponding to the above mentioned precedence relationships can be seen in [9]. The following performance variables were considered to contrast EA1 versus LSA:

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

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

- **Topt** : Total number of optima. It is the mean total number of distinct optimal solutions found by the algorithm throughout ten runs.

Case	Alt		Opt		Topt	
	EA1	LSA	EA1	LSA	EA1	LSA
1	4.2	1	4.0	1	39	1
2	1.8	1	1.2	1	11	1
3	4.2	1	4.2	-	42	-
4	2.8	1	2.7	-	26	-
5	1.4	1	1.2	-	12	-

Table 1. EA1 versus LSA, comparative performance.

By observing table 1 the following comparison can be done:

- The evolutionary approach found more than a single optimal solution for any case.
- All the anomalies observed with LSA do not hold when EA is applied, because:
  - When the number of processors is increased the minimum (optimum) makespan is also found.
  - When the duration of tasks is reduced this reduction is reflected in a reduced optimum makespan.
  - When the number of precedence restrictions is reduced the optimum makespan is preserved.

Preliminary results on the selected test suite showed two important facts. Firstly, the EA provides not a single, but a set of optimal solutions, providing for fault tolerance when system dynamics must be considered. Secondly, the EA is free of the LSA anomalies. These facts do not guarantee finding optimal solutions for any arbitrary task graph, but show a better approach to the problem. As explained in the next section, attempting to improve behaviour we continue experimentation with different recombination approaches.

2.2.1. Enhancements by use of multirecombination

In his multiparent approach Eiben et al. [4], [5], [6], [7] initially used *uniform scanning crossover*. This method generates a single offspring. Each gene in the child is provided from any of the corresponding genes in the parents with equal probability.

By using a greater number n_1 of parents, offspring creation is based on a larger sample from the search space and consequently larger diversity is supplied. This can help to avoid premature convergence.

As reported in his work, it was difficult to draw conclusions on the optimal number of parents, but it was determined that a better performance is attained for 2 to 4 parents and increasing the number of parents could deteriorate the performance.

Recently we proposed *multiple crossovers on multiple parents* (MCMP) [10] which allows multiple recombination of multiple parents under uniform scanning crossover, expecting that exploitation and exploration of the problem space be adequately balanced. MCMP provides a means to exploit good features of more than two parents selected according to their fitness by repeatedly applying the selected crossover method (in this case uniform scanning). Once selected, the parents undergo crossover a number n_2 of times specified as an argument, and generates n_2 children. The preliminary experiments implemented a generational genetic algorithm with binary coded chromosomes, then n_1 parents were selected by *fitness proportional selection* to undergo multirecombination and bit flip mutation obtaining n_2 offspring. Subsequently, the fittest child was selected for insertion in the next generation.

By means of this association it is expected that the search space be efficiently exploited by the multiple application of crossovers and efficiently explored by the greater number of samples provided by the multiple parents.

Results reported can be sumarized as follows. The use of MCMP showed to be efficient in optimization of hard unimodal and multimodal testing functions. It was shown that the multiparent approach behaves better when it is associated to the multiple crossover approach on both types of functions selected for optimization. Speed of convergence, measured in number of generations, is augmented without increasing the risk of premature convergence. Consequently, the quality of results is better than the previous attained under more complex approaches [11]. Additionally, when observing the final population it was detected that all individuals are much more centred surrounding the optimum. This property is strongly detected in multimodal optimization. This is an important issue when an application requires provision of multiple alternative near-optimal solutions.

Experiments under MCMP, denoted by EA2, were run with similar parameter settings as under SCPC, but with $n_1 = 4$ and $n_2 = 2$. As shown in table 2 in most cases EA2 shows better performance than EA1 finding a greater number of alternative per run, optimal per run , and total optimal solutions.

Case	Alt		Opt		Topt	
	EA1	EA2	EA1	EA2	EA1	EA2
1	4.2	9.5	4.0	9.5	39	90
2	1.8	1.7	1.2	1.4	11	14
3	4.2	5.4	4.2	5.3	42	53
4	2.8	4.3	2.7	4.3	26	43
5	1.4	1.9	1.2	1.2	12	12

Table 1. EA2 versus EA1, comparative performance.

3. The Pm| $|C_{max}$ problem

The problem of minimizing the makespan for a set of n independent tasks in a parallel system consisting of m processors is important because it has the side effects of maximizing the throughput and balancing the load in the processors.

3.1. The longest processing time first rule (LPT)

LPT assigns at time t=0 the largest job to any available machine. After that whenever a processor is freed, the largest unscheduled job is allocated to the processor. This heuristic attempts to place the shorter jobs towards the end of the schedule in order to balance the loads.

An upper bound has been found for the ratio between the makespan computed by LPT and the optimal makespan. It is given by:

$$\frac{C_{max}(LPT)}{C_{max}(OPT)} \leq \frac{4}{3} - \frac{1}{3m}$$

Proof of the corresponding theorem can be seen in [21]. Consequently, the lower bound for the optimal makespan is given by:

$$L_b = \frac{C_{max}(LPT)}{\frac{4}{3} - \frac{1}{3m}} \leq C_{max}(OPT)$$

3.2 . Contrasting heuristics

For the $Pm|\ |C_{max}$ problem, many random testing cases were chosen and similar results were obtained. Here we report eigth of them where for n jobs and m machines, they are denoted as the following (n, m) 2-tuples;

1. (9,4)	2. (13,6)	3. (19,6)
4. (21,4)	5 (36,5)	6. (50,5)
7. (51.10)	8. (73.10)	

After many preliminary runs, the parameter settings of the MCMP evolutionary approach for this problem (EA3), were determined as follows. Number of parents $n_1 = 3$ and number of crossovers $n_2 = 2$. Randomized initial population of size fixed at 100 individuals. Twenty series of ten runs each were performed on five testing cases, using elitism. The maximum number of generations was fixed at 200, but a stop criterion was used to accept convergence when after 20 consecutive generations, mean population fitness values differing in $\varepsilon \leq 0.01$ were obtained. Probabilities for crossover and mutation were fixed at 0.8 and 0.1, respectively for instances 1 to 4 and at 0.8 and 0.0001 for instances 5 and 6. In the current experiments, alternatively the fittest and a random chosen child were selected for insertion in the next generation. This strategy was adopted to maintain population diversity.

Instance 1: 9x4

4 7 5 4 6 7 4 6 5

Instance 2: 13x6

9 10 6 11 8 6 7 10 8 11 6 9 7

Instance 3: 19x6

7 13 8 9 15 10 14 10 12
9 11 8 12 7 13 14 7 15

Instance 4: 21x4

5 6 7 8 9 13 11 10 5 11 8 12 10 13 12
9 6 14 14 5 7

Instance 5: 36x5

5 5 7 7 7 5 5 12 12 5 8 8 8
9 10 10 8 8 9 9 10 10 9 9 10
6 6 12 12 6 6 6 7 7 12 12

Instance 6: 50x5

7 6 2 7 7 7 3 3 7 1 9 3 2 1 6 3 7 6 2 1
9 7 3 8 5 8 1 5 7 9 3 4 7 1 2 4 2 8 4 7
5 9 7 7 2 9 6 1 3 5

Instance 7: 51x10

11 18 18 13 16 13 15 17 17 11 18 20 20 16
10 17 17 19 13 13 10 10 15 15 19 21 21
18 20 19 11 16 15 17 13 10 10 16 16 15
19 19 21 21 21 20 20 11 11 18 11

Instance 8: 73x10

9 18 18 9 9 20 20 20 14 14 10 11 11 22 22 16
10 10 22 14 14 14 23 23 23 15 15 16 13 17 13
12 12 17 17 12 12 19 19 19 12 13 13 13 17
20 20 21 16 16 21 21 21 21 11 11 11 17 16
22 22 23 23 15 15 15 18 10 10 18 18 19 19

Fig. 2. The testing instances

As previous experiments showed that under MCMP A*lt*, *Opt* and *Topt*, got better values, to contrast MCMP versus LPT only the following variables were considered:

- **Best**: is the best makespan value found throughout all the runs. This value has been reached by different solutions (schedules).
- **Mbest**: is the mean value, over the total number of runs, of the best makespan found in each run (average best individual).
- **Ebest** = (Abs(L_b- best value)/ L_b)100
  It is the percentile error of the average best individual when compared with the lower bound for the optimal makespan. It gives us a measure of how far are we from that lower bound. When values of Ebest are equal to 0, we can certainly conclude that the optimal solution was found. For values greater than 0 we can conclude nothing about finding the optimum.
- **Totb** : Total number of best solutions. It is the mean total number of distinct best solutions found by the algorithm throughout ten runs.

Results are shown in the following table.

Inst.	Lb	Best		Mbest	Ebest		Totb
		EA3	LPT	EA3	EA3	LPT	EA3
1	12.00	12	15	12.0	0.00	25.00	10
2	18.00	19	23	19.2	6.66	27.77	8
3	30.52	35	39	36.0	17.94	27.77	3
4	41.60	49	52	49.2	18.26	25.00	9
5	49.73	60	63	60.5	21.64	26.66	5
6	39.47	50	50	50.1	26.92	26.66	11
7	69.23	85	90	87.5	26.38	30.00	7
8	95.38	122	124	124.5	30.52	30.00	2

Table 2. MCMP versus LPT, comparative performance.

According to data in table 2, the following remarks can be done. For the smallest instance 1, MCMP is able to find the optimum makespan in each run of the series. The best makespan value (*Best*) found by MCMP is better for instances 1, 2, 3, 4, 5, 7 and 8, and equal for instance 6 than the corresponding value found by LPT. For any instance MCMP finds a number greater than 1, of alternative schedules (*Totb*) with the same best makespan. The percentile error of the average best individual (Ebest) ranges from 0.00 to 30.52 for MCMP and from 25.00 to 30.00 for LPT.

4. Conclusions

Parallel machine scheduling problems involves the allocation of a number of parallel tasks (precedence constrained or independent) in parallel supporting environments is a difficult and economically important issue in production and computer systems. In this work we approached allocation attempting to minimize the makespan. Other performance variables such as individual processor utilization or evenness of load distribution can be considered. As we are interested in scheduling an arbitrary number of tasks onto a reasonable number of machines, 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 for (Pm|prec|C_{max}) and the longest processing time first rule (LPT) does it for (Pm||C_{max}).

A variant of evolutionary algorithms MCMP, applying multirecombination was used in two versions EA2 and EA3 and run for two set of instances of the above mentioned problems. Results show an improved overall performance of the MCMP approach when contrasting EA2 against EA1 for the first problem and when contrasting EA3 versus LPT for the second problem. The effect is a better quality of results and the availability of not only one but a set of alternative solutions which permits to chose different equally valued schedules according to the systems needs.

Experiments indicate that MCMP provides a better approach when regarding quality of results. Further research is necessary to investigate potentials and limitations of this novel multirecombination approach under more complex test suites allowing to study optimal (n_1,n_2) associations that could be found by self-adaptation, and the effect of multiple crossovers on multiple parents under diverse crossover methods.

5. 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.

6. 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] 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

[5] 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.

[6] Eiben A.E. and. Bäck Th., *An empirical investigation of multi-parent recombination operators in evolution strategies.* Evolutionary Computation, 5(3):347-365, 1997.

[7] Eiben A.E. and van Kemenade C.H.M., *Diagonal crossover in genetic algorithms for numerical optimization.* Journal of Control and Cybernetics, 26(3):447-465, 1997.

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

[9] Esquivel S., Gatica C., Gallard R., *Solving the parallel task scheduling problem by means of a genetic approach.* Proceedings of the First IEEE American Network Operations and Management Symposium (LANOMS'99), pp 117-128, Rio de Janeiro, Brazil, December 1999.

[10] Esquivel S., Leiva H.,.Gallard R., *Multiplicity in genetic algorithms to face multicriteria optimization.* Proceedings of the 1999 Congress on Evolutionary Computation (IEEE). Washington DC, pp 85-90.

[11] Esquivel S., Leiva A., Gallard R.: *Couple Fitness Based Selection with Multiple Crossover per Couple in Genetic Algorithms.* Proceedings of the International Symposium on Engineering of Intelligent Systems (EIS'98), La Laguna, Tenerife, Spain Vol. 1, pp 235-241, ed. E.Alpaydin. Publishesd by ICSC Academic Press, Canada/Switzerland, February 1998.

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

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] Krause M., Nissen V., *On using penalty functions and multicriteria optimization techniques in facility layout.* Evolutionary Algorithms for Management Applications, ed J. Biethahn and V. Nissen (Berling: Springer), pp 153-166, 1995.

[19] 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.

[20] *Michalewicz Z., Genetic Algorithms + Data Structures = Evolution Programs.* Springer Verlag , Third, Extended Edition, 1996.

[21] Pinedo M, *Scheduling Theory, Algoritms, and Systems.* Prentice-Hall -1995. mpinedo@stern.nyu.edu

[22] Syswerda G., *Schedule optimization using genetic algorithms.* In L. Davis, editor, Handbook of Genetic Algorithms , chapter 21, pages 332-349. Van Nostrand Reinhold, New York, 1991.

[23] Withley D., Starkweather T., Fuquay D'A., *Scheduling Problems and Travelling Salesman: The Genetic Edge Recombination Operator.* Proceedings of the 3rd Int. Conf. on Genetic Algorithms, pp 133-140. Morgan Kaufmann Publishers, Los Altos CA, 1989.