A Hybrid Genetic Algorithm for the RCPSP with the Peak Crossover Operator

Vicente Valls and Francisco Ballestín
Dpto. de Estadística e Investigación Operativa, Facultad de Matemáticas, Universitat de Valencia, Dr. Moliner, 50, 46100 Burjassot, Valencia, Spain.
Vicente.Valls@uv.es, Francisco.Ballestin@uv.es

Sacramento Quintanilla
Dpto. de Economía Financiera y Matemática, Facultad de Económicas y Empresariales, Universitat de Valencia, Avda. de los Naranjos, s/n, Edificio Departamental Oriental, Valencia, Spain.
Maria.Quintanilla@uv.es.

Abstract
We propose a hybrid genetic algorithm, HGA, for the RCPSP. HGA introduces several changes in the GA paradigm based on the knowledge of the problem. The distinguishing features of our approach are the following: a new *peak crossover operator*; a specific operator for the RCPSP; and a *double justification operator*. The latter is a local improvement operator applied to each generated solution. The search is organised in two phases; and during the second phase the neighbourhood of the best solution found is searched. Computational experiments on the standard j120 ProGen-generated set show that our algorithm clearly outperforms all RCPSP state-of-the-art algorithms known to the authors.

1. Introduction

In this paper we deal with the classical resource-constrained project scheduling problem (RCPSP). As a job shop generalisation, the RCPSP is NP-hard (cf. Blazewicz [2]). Accordingly, many heuristic approaches have been proposed for RCPSP. The most competitive heuristic algorithms seem to be those of Hartmann [3,4] , Nonobe and Ibaraki [8], Merkle et al [6, 7], Möhring et al [5], Alcaraz and Maroto [9], Tormos and Lova [9], Valls et al [10, 11, 12].

It is difficult to compare these algorithms because the data published by the various authors refers to different stop rules and distinct collections of test problems. Nevertheless, all the authors present computational results based on the j120 set from the PSPLIB library, and measure the quality of their algorithms using average deviation from CPM (CPM_dev). Leaving aside the question of which computers were used, the j120 results mean the heuristic algorithms can be divided into two groups – one formed by the first seven and a second group formed by the final two.

The following data has been obtained from data published by the various authors and gives an idea of the quality of the solutions obtained by the algorithms. In the first group, when less than five minutes is employed, the CPM_dev values are between a minimum value of 35.3% and a maximum value of 36.74%. One of the algorithms (Nonobe and Ibaraki, [8]), achieves a CPM_dev value of 34.99 % while using more than 10 minutes of CPU time. Another of the algorithms (Merkle et al, [6]), achieves 33.68 % using 25 minutes of CPU time and achieves up to 32.97 % when using much more time. In the second group, the algorithm described in Valls et al [10] achieved a CPM_dev value of 34.27 % in 22.55 seconds while the algorithm HIA presented in Valls et al [11] reached a CPM_dev value of 31.58 % in 59.43 seconds.

In this paper, we propose a hybrid genetic algorithm, HGA, for the RCPSP. HGA introduces several changes in the GA paradigm. For this reason, we say that the resulting algorithm is Hybrid

Genetic Algorithm (HGA). These changes are inspired by the techniques successfully used by Valls et al [10, 11, 12].

2. Main elements of HGA

2.1. Individuals and fitness

In HGA, the individuals are activity list representations of schedules. An *activity list* is any precedence feasible permutation $\lambda = (j_1,j_2,...,j_n)$ of the activities. If $i = j_h$ we can say that activity I is in position $p(i) = h$. Given a schedule S, any activity list $\lambda = (j_1,j_2,...,j_n)$ that satisfies: $s_i < s_j$ implies $p(i) < p(j)$ – is called an *activity list (AL) representation* of S. The serial schedule generation scheme (SGS) can be used as a decoding procedure to obtain an active schedule *$S(\lambda)$* from an activity list $\lambda = (j_1,j_2,...,j_n)$ by selecting, at each stage p, the activity j_p. In the rest of the paper, we assume that the tie-breaking rule chooses the activity with the smallest activity label and we indicate the only AL representation of S with *$\lambda(S)$*. We also assume that all considered schedules are active and use S, and *$\lambda(S)$* interchangeably.

The fitness of an individual λ is given by the makespan of the associated schedule $S(\lambda)$.

2.2. The peak crossover operator

General crossover operators probabilistically recombine parts of good solutions in order to obtain new solutions. The idea behind this is that these parts have contributed to the fitness of the parent individuals, and that the recombination of these good parts may lead to better solutions. These parts are collected randomly, without any guarantee or indication as to what is their quality. The evolutionary process eliminates the bad combinations and preserves the good combinations. Nevertheless, it is possible to take advantage of the knowledge of the problem to identify and combine those parts of the good solutions that have really contributed to the level of quality. This is the purpose of the *peak crossover operator*, an operator designed specifically for RCPSP.

Broadly speaking, a peak in an activity list representation λ is a sub-list of λ – as the use of resources in the time period of the peak $S(\lambda)$ when activities are sequenced is greater than a pre-fixed proportion of the total resources available. Given two activity list representations – λ (father) y λ' (mother) – of the current population that we wish to cross, the peak operator generates two new activity lists, the son and the daughter, so that the son (daughter) inherits the peaks of the father (mother) and the mother (father) determines the position of the rest of activities to be placed ahead, and behind, each peak.

2.3. The double justification operator

Several researchers have extended the original GA approach with the use of local search strategies for improving the obtained solutions by using genetic operators. With the same objective, we use the *double justification operator* – a local improvement operator that, thanks to its simplicity, speed, and efficiency (Valls et al.[12]), can be applied to all RCPSP solutions generated by the evolutionary process. Given a schedule S, to *double justify* an activity j consists of first justifying it to the right and then to the left. .

2.4. A neighbouring population

Generally, as the number of generations increase, it becomes increasingly difficult to obtain solutions that improve on the quality of the best solution found so far – which, hopefully, is itself a high quality solution. Experience seems to show (Valls et al [10, 11]) that good candidate schedules are usually found 'fairly close' to other good schedules. Therefore, exploring the neighbourhood of the best solution found may lead to better solutions. For this reason, HGA has two phases: an initial general search phase followed by a second phase that searches in the area of the best solution obtained. Multi-pass sampling methods can be adapted to explore, in a controlled way, regions of differing depths. Specifically, the *β biased random sampling method* (Valls et al [10]) can be repeatedly applied to the best schedule obtained in order to generate a new population (*a neighbouring population*) of relatively high quality. Phase 2 begins at the best sequence obtained in Phase 1 and generates a new population of nearby schedules and initiates a process of improvement.

2.5. Initial population

We employ the regret-based biased random sampling method (cf. Hartmann [3]) together with the LFT priority rule and $\alpha = 1$ to obtain POPsize activity lists. Then each list is decoded, doubly justified, and coded again. The resulting activity list representations become members of the initial population. This method, without double justification, was used in Hartmann [3].

2.6. Mutation and selection

The mutation and selection operators are as in Hartmann [3].

3. Computational experiments

The experiments were performed on an AMD personal computer at 400 MHz. The algorithm was coded in C. We first tested HGA on the instance sets j30, j60, j90, and j120 available in PSPLIB – limiting the number of generated schedules to a maximum of 5000. The following table summarises the results. The first column indicates the instance set referred to in the results shown in each row. The second column, labelled *∑PSPLIB*, shows the sum of the best values in PSPLIB as of September 10, 2001; whereas the third column, *∑HGA*, consists of the sum of the values obtained by HGA. The average percentage deviation from the critical path makespan is reported in the fourth column, labelled *CPM_dev.* The fifth (sixth) column, *Av_dev (Max_dev),* consists of the average (maximal) percentage deviations from the best solutions in PSPLIB. The number of instances for which our algorithm obtains the best known values in PSPLIB is reported in the seventh column (*Best_sol*). It is worth noting that these known values include those found by all versions of our algorithm. The average (maximal) computational time in seconds is shown in column eight (nine), labelled *Av_CPU* (*Max_CPU*).

Table 1 Computational results for j30, j60, j90 and j120 with an upper limit of 5000 schedules.

	ΣPSPLIB	**ΣHGA**	**CPM_dev**	**Av_dev**	**Max_dev**	**Best_sol**	**Ave_CPU**	**Max_CPU**
j120	74013	75222	32.54	1.36	6.35	206	2.03	3.46
j90	45741	45974	10.46	0.40	5.48	374	0.61	2.31
j60	38368	38551	11.10	0.38	5.30	375	0.46	1.42
j30	28316	28339	13.47	0.06	3.45	462	0.31	0.66

To deepen the study of HGA we have run the algorithm on the instance set j120 while changing the maximum number of permitted schedules to 2500, 10000, 25000 and 100000. The results are shown in the following table.

Table 2 HGA computational results for j120 with different limits on the number of schedules.

	Σ	**Av_dev**	**CPM_dev**	**Best_sol**	**Ave_CPU**	**Max_CPU**
HGA2500	75584	1.77	33.18	195	1.02	1.76
HGA5000	75222	1.36	32.54	206	2.03	3.46
HGA10000	74933	1.05	32.04	226	4.01	6.97
HGA25000	74640	0.71	31.51	287	10.02	17.69
HGA100000	74319	0.37	30.95	392	39.51	69.26

4. Comparison with other heuristics

We have compared these results with those published by various authors mentioned in the introduction and despite the variety of computers used in the experiments; we can confirm that HGA clearly outperforms the others state-of-the-art algorithms known to the authors – at least for the 600 instances of the j120 set of RCPSP.

Acknowledgement.

This research was partially supported by the CICYT under contract TAP99-1123 and by the Generalitat Valenciana under contract FP 198-CB-12-303.

References

[1] Alcaraz, J. and Maroto, C. (2001), A Robust Genetic Algorithm for Resource Allocation in Project Scheduling, Annals of Operations Research 102, pp. 83-109.

[2] Blazewicz, J., Lenstra, J. K. and Rinooy Kan, A. H. G. (1983), Scheduling Subject to Resource Constraints: Classification and Complexity, Discrete Applied Mathematics 5, pp. 11-24.

[3] Hartmann, S. (1998), A Competitive Genetic Algorithm for Resource-Constrained Project Scheduling, Naval Research Logistics 45, pp. 733-750.

[4] Hartmann, S. (2000), A Self-Adapting Genetic Algorithm for Project Scheduling under Resource Constraints, Workpaper.

[5] Möhring, R. H., Schulz, A., Stork, F. and Uetz, M. (2000), Solving Project Scheduling Problems by Minimum Cut Computations, Workpaper.

[6] Merkle, D., Middendorf, M. and Schmeck, H. (1999), Ant Colony Optimization for Resource-Constrained Project Scheduling, Workpaper Institute for Applied Computer Science and Formal Description Methods, Universität of Karlsruhe, Karlsruhe, Germany.

[7] Merkle, D., Middendorf, M. and Schmeck, H. (2001), Ant Colony Optimization Techniques for the Resource-Constrained Project Scheduling Problem, Workpaper Institute for Applied Computer Science and Formal Description Methods, Universität of Karlsruhe, Karlsruhe, Germany.

[8] Nonobe, K. and Ibaraki, T. (1999), Formulation and Tabu Search Algorithm for the Resource Constrained Project Scheduling Problem (RCPSP), Essays and Surveys in Metahuristics (MIC'99).

[9] Tormos, P. and Lova, A. (2001), A Competitive Heuristic Solution Technique for Resource-Constrained Project Scheduling, Annals of Operations Research 102, pp. 65-81.

[10] Valls, V., Quintanilla, S. and Ballestín, F. (2001a), Resource-constrained Project Scheduling: A Critical Activity Reordering Heuristic. To appear in European Journal of Operational Research.

[11] Valls, V., Ballestín, F. and Quintanilla, S. (2001b), A Population Based Approach to the Resource Constrained Project Scheduling Problem. Technical Report 6-01, Departamento de Estadística e Investigación Operativa, Facultad de Matemáticas, Universitat de València, Spain.

[12] Valls, V., Ballestín, F. and Quintanilla, S. (2001c), Justification and RCPSP: a technique that pays. Technical Report 7-01, Departamento de Estadística e Investigación Operativa, Facultad de Matemáticas, Universitat de València, Spain.