Generating RCPSP Instances with Known Optimal Solutions

J. Silva Coelho[1]

[1]CESUR/IST, Technical University of Lisbon, Portugal
Universidade Aberta, Portugal
e-mail: jcoelho@univ-ab.pt

Keywords: Project Scheduling, Instance Generation.

1. Introduction

Resource Constrained Project Scheduling Problem (*RCPSP*) is one of the most classical problems in Project Scheduling, and contains many other scheduling problems. It consists in scheduling a set of activities, with associated processing times and resources limited per time instant, and precedence relations between activities. The aim is to find the minimal scheduling that respects the precedence relations and resource limits (see [1]).

Several approaches for solving *RCPSP* have been purposed and compared in the past years (see [2]). To be able to make such comparison one must use the different approaches to solve the same instance set. This instance set should be representative of all instances of *RCPSP*.

For generating a representative set of instances, one must generate the precedence relations between activities (the network), and resource usage. Several algorithms have been purposed for this task (see [3-6]), and special care has been addressed to generate different instances in their network and resource characteristics. The characteristics which need to be diverse are not consensual, nevertheless a standard instance set was build *PSPLIB* (see [7]), and is used today for the purpose of testing algorithms.

The method used nowadays to generate instances is a random procedure that guaranties that a specific characteristic has a specified value, so it is possible to generate several random instances diverse in the considered characteristics. This method does not give any clue about the optimal solution. Because the *RCPSP* is a *NP-hard* problem, there is no algorithm available today to calculate the optimal solution, unless the instance is small enough or is easy enough to solve. For that reason all the optimal solutions are only known for instances of 30 activities of *PSPLIB*, allowing in this case the evaluation of algorithms of *RCPSP* relative to the optimal solution. Having optimal solutions can be useful also for the study of the complexity of the instances. Without the optimal solution any indicator of the complexity of the instance it will be inexact.

To the best of our knowledge, there is no method for *RCPSP* that allows the generation of an instance with known optimal solution, for any instance size, and applicable in any network. The purpose of this text is to present a method that use a network generator, and over the generated net build the resource information that guaranties a known optimal schedule with the desired resource indicators.

A small application, *GenRes*, was made which implements the described method based on a network based on a network in the format of *PSPLIB*, making it easy to use with existing network generators. It regenerates the resources keeping precedence relations, number of resources, and try to keep resource indicators as close as possible to those of the original instance.

In section 2 the generation method of *GenRes* is presented, in section 3 the results of some tests are given which clarify some points about the generator, and in section 4 we end with some conclusions.

2. Generation Method

The method has three phases: extra precedence relations generation, resources generation, and processing times generation. The *GenRes* parameters are an acyclic network, the number of extra precedence relations to add, *K*, and the number of saturated resources *SR*. The number of resources *R*, the desired resource factor *RF* (percentage of activities that use a resource) and resource constrainedness *RC* (percentage of utilization of resource capacity for the activities that use it), and the sum of all processing times of activities can be specified, otherwise the generator will calculate them from the input file.

In the first phase, processing times of all activities are set to 1, and the earliest start time schedule is calculated and set as actual schedule. Then, extra precedence relations are added to the network, between two activities selected at random, but only if earliest start schedule is affected and total project processing time does not become greater than some limit *L*. When *K* extra precedence relations are inserted in the network, or after a maximum number of tries without adding any extra precedence relation, the phase is marked as completed.

The second phase will assign resources to the activities, in a way that does not violate the current schedule, but that accomplish the desired *RF* and *RC*. Some of the resources should be saturated, to make sure that no better solution exists. It is possible that none of the resources is saturated, but then the optimal schedule is only an upper bound, there is no guaranty that it is optimal.

In the third phase a start instant of the current schedule is selected at random and the processing time of all activities that begin in that start instant are increased. This procedure is repeated until the total sum of processing times is achieved. Then, the extra precedence relations are discarded and the current instance is returned along with the optimal solution.

Some aspects of the method are now further detailed, namely the assignment of a resource and the calculation of time limit *L*.

For the assignment of a resource, the activities are arranged in random order, and for the first activities in that order unary assignments of the resource are done to those activities, in a way that *RF* is attained. For a resource that should be saturated, the ordered list of activities is processed and the resource is assigned to an activity only if this activity does not start at the same instant of a previous activity in the current list order, and after all instants have some activity selected, the step continues treating this resource as a normal one. This makes sure that this resource can be saturated, otherwise there could appear time instants in which the resource would be inactive.

After *RF* is achieved, the activities which consume the resource are selected at random and the resource assignment is increased. The process stops when the desired *RC* is achieved, or in the case of a saturated resource, when no more assignments are possible. Note that before assigning a resource to an activity, it is necessary to check if the resource capacity will not be excluded, in which case the assignment should not be made. After a maximum number of tries without making any resource assignment the process is set as complete.

As far as the calculation of time limit *L* is concerned, and to accomplish the *RF* and *RC* values, before the first phase the following procedure must be done: first *L* should be set to the minimum value of *RF.RC.N* (with *N* equal to the number of activities), after that, if *L* is lower than thee earliest start schedule than this value is attributed to *L*, and finally we add to *L* a random value that follows a *Geometric* distribution with probability ½, with maximum value of *N/2*. This allows the generation of different size solutions for the same *RF*, *RC* and *N* values, but keeps the value of *L* close to the minimum value.

After *L* is set, *RF* and *RC* are increased by 0.01 while *L>RF.RC.N*, so that those values will be as close as possible to the desired value. To make the original global *RF* and *RC*, distinct *RF* e *RC* are used in non saturated resources, making the average equal to the global desired parameter. This technique could unbalance the resources, so to make sure *RF* and *RC* are accomplished the number of saturated resources needs to be kept low.

Finally the earliest start schedule, considering also the extra precedence relations, is calculated and saved as an optimal schedule, and the instance without the extra precedence relations is returned.

3. Tests

In this section some tests with the generator are described to clarify some points. For this purpose, *PSPLIB* instances were used as argument of the generator, with networks ranging from 30 to 120 activities, in a total of 2040 instances. The optimal arguments of *GenRes* were not used, forcing the generator to get those values from the instances. The *K* value was set to 100, and the number of saturated resources *SR* was set to 1.

One basic question can now be clarified: is the *GenRes* capable of generating instances of all types? To answer that it is only required to verify that the resource indicators are diversified, because the morphological indicators are not controlled by the generator, and the network that is received as input is not changed. Since *RF* and *RC* are input arguments of the generator it is only necessary to make sure they are accomplished.

In figure 1 at the left is the scatter plot of original *RF* and *RC* vs the *RF* and *RC* of the generated instances, and it can be seen that for all instances the original and generated values are very close. On the right side of the same figure, are the original *RF* vs *RC* and the generated ones. The *RC* does not go until 100% since *PSPLIB* was generated using resource strength, *RS*, and not *RC*. The main reason not to use *RS* in this work is that it depends on the morphology which is not controlled by this generator.

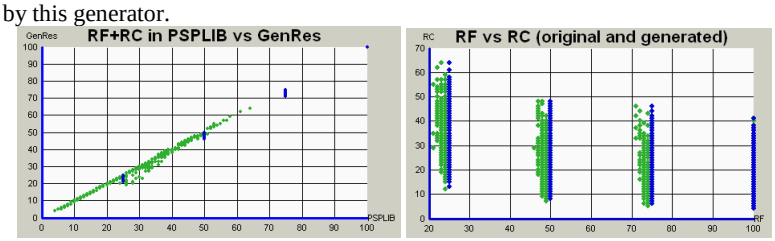

Figure 1 –Analysis of results of *GenRes* for resource indicators *RF* and *RC*

Another point is the complexity of the instances, which should span the entire range of difficulty, from easy to hard. In this work the complexity is measured using the performance of a priority rule, the latest start time, *LST*, measured relatively to the optimal value in the instances of *GenRes* and relative to the current upper and lower bound in *PSPLIB* instances.

In the left of figure 2 is a scatter plot of parallel scheduling against serial schedule using *LST* rule, relative to the upper bound and lower bound. It can be seen that if the optimal value is not known, one may be led to very different conclusions regarding the hardness of an instance, depending on whether the upper or lower bound is considered. Since parallel scheduling has worst results, the complexity measure selected is serial scheduling *LST* priority rule, relative to the optimal value for instances of *GenRes* or upper and lower bound for instances of *PSPLIB*. In the right side of figure 2 are the histograms of complexity measure for *PSPLIB*, and it can be seen that more than half of the *PSPLIB* nets are very easy, since serial *LST* priority rule returns an optimal solution. The instances generated with *GenRes* for the same networks and resource indicators, result in a more distributed instances in hardness.

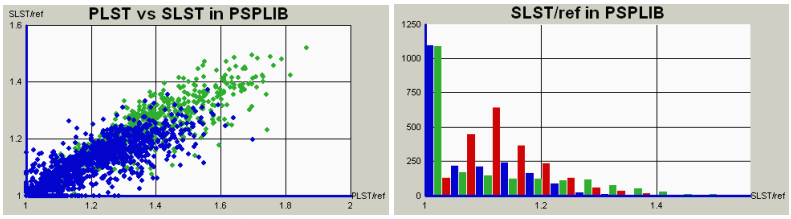

Figure 2 – Analysis of results of *PLST* and *SLST* priority rules in *PSPLIB* and *GenRes* instances

The next analysis is to study the behaviour of the generator, using different values for the number of saturated resources, *SF*. The 2040 instances of *PSPLIB* were generated again using *SF=2* and *SF=3*, instead of the original value *SF=1*. The histogram of the complexity indicator used is presented in figure 3 for all those cases. It can be seen that with increasing *SR* the instances became harder.

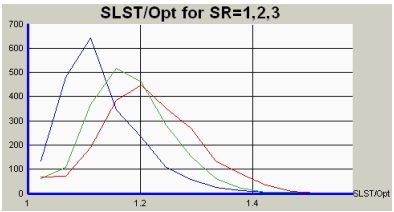

Figure 3 – Analysis of number of saturated resources *SR*

The number of extra precedence relations added, *K*, was also studied. The generation was redone for *K=32,16,8,4,2* and *1*. For *K* from 100 to 4 the histogram is more or less the same, but for *K=2* and *1*, the histogram will have more easy instances, but it still has many hard instances.

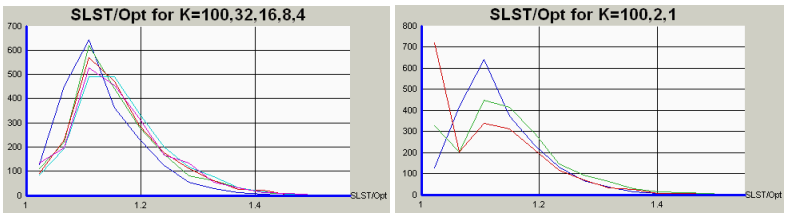

Figure 4 – Analysis of number of extra precedence relations *K*

Finally, the influence of the number of resources *R* on the complexity indicator distribution is analysed. The histograms for *R=2,4,8* and *16*, can be seen in figure 5. The conclusion that can be drawn from the scatter plot in figure 5 is that the number of instances of average difficulty decreases and that the complexity of harder instances increases. The scatter plot also show that an instance that is hard with *R=8* it will probably be hard with *R=16*, and the same for easy instances, so the hardness of the instances cannot be explained only by the number of resources.

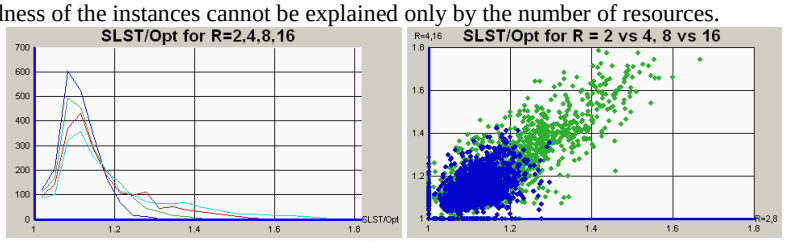

Figure 5 – Analysis of number of resources *R*

4. Conclusions

A generator, *GenRes*, based on an existing network, and producing the resources with a known optimal solution was described. The results presented are based on the networks and resource indicators from the *PSPLIB* instances, and the *RF*/*RC* indicators of the generated networks are only slightly different from the original ones.

The instances generated by *GenRes* have a more equally distribution of complexity than *PSPLIB*, where about half of the instances are easy. The increasing in the number of saturated resources and the number of total resources, results in the increasing of the instance complexity, but the value for argument *K* appears to have no influence except for very small values of *K*.

Two major applications of this work are in the analysis of the complexity of the instances and in the comparison of algorithms. To study what makes the instances complex or simple, complexity indicators are required, and this method allows the construction of such indicators based on the optimal value, and therefore increasing their precision. To study the performance of algorithms one needs to measure the quality of the solutions, and with this generator a precise measure can be calculated relative to the optimal solution, instead of using upper or lower bounds, that can have large gaps in hard instances. The optimal value allows the measuring of the exact quality of an algorithm in an instance set, instead of making conclusions relative to the performance of other algorithms.

References

[1] Peter Brucker, Andreas Drexl, Rolf Möhring, Klaus Neumann, Erwin Pesch, "Resource-constrained project scheduling: Notation, classification, models, and methods", European Journal Of Operational Research (112)1 (1999) pp. 3-41

[2] S. Hartmann and R. Kolisch. "Experimental Evaluation of State-of-the-Art Heuristics for the Resource-Constrained Project Scheduling Problem", European Journal of Operational Research, Vol. 127 (2000) pp. 394-407

[3] Demeulemeester, E., Vanhoucke, M. and Herroelen, W., 2003, "A random network generator for activity-on-the-node networks" Journal of Scheduling, 6, 13-34.

[4] L. Valadares Tavares; J. Antunes Ferreira; J. Silva Coelho, 1999, "The risk of delay of a project in terms of the morphology of its network", European Journal of Operational Research 119, 510-537

[5] M.K. Agrawal; S.E. Elmaghraby; W.S. Herroelen, 1996, “DAGEN: A generator of testsets for project activity nets”, European Journal of Operational Research, 90, 376-382
[6] R. Kolisch, A. Sprecher, A.Drexl, 1995, “Characterization and generalization of a general class of resource-constrained project scheduling problems”, Management Science 41, 1693-1703
[7] R. Kolisch, A. Sprecher, 1996, “PSPLIB - a project scheduling problem library”, European Journal of Operational Research 96, 205-216