Solving Resource-Constrained Project Scheduling Problems with Time-Windows Using Iterative Improvement Algorithms

Angelo Oddi and **Riccardo Rasconi**
Institute for Cognitive Science and Technology, CNR, Rome, Italy
{*angelo.oddi, riccardo.rasconi*}*@istc.cnr.it*

Abstract

This paper proposes an iterative improvement approach for solving the Resource Constraint Project Scheduling Problem with Time-Windows (RCPSP/max), a well-known and challenging *NP-hard* scheduling problem. The algorithm is based on Iterative Flattening Search (IFS), an effective heuristic strategy for solving multi-capacity optimization scheduling problems. Given an initial solution, IFS iteratively performs two-steps: a *relaxation-step*, that randomly removes a subset of solution constraints and a *solving-step*, that incrementally recomputes a new solution. At the end, the best solution found is returned. The main contribution of this paper is the extension to RCPSP/max of the IFS optimization procedures developed for solving scheduling problems without time-windows. An experimental evaluation performed on medium-large size and web-available benchmark sets confirms the effectiveness of the proposed procedures. In particular, we have improved the average quality w.r.t. the current bests, while discovering three new optimal solutions, thus demonstrating the general efficacy of IFS.

Introduction

This paper explores the solving capabilities of the Iterative Flattening Search (IFS) (Oddi et al. 2008) algorithm against scheduling problem instances belonging to the class of Resource Constrained Project Scheduling Problem with Time Windows (RCPSP/max). IFS represents a family of stochastic search techniques that was originally introduced in (Cesta, Oddi, and Smith 2000) as a non systematic approach to solve difficult scheduling problem instances; as demonstrated in (Bartusch, Mohring, and Radermacher 1988), RCPSP/max problems indeed belong to this category, as both the optimization and the feasibility versions of the problem are *NP-hard*. IFS is devised to iteratively use heuristics for solving makespan-minimization scheduling problems, and it has been shown to have very good scaling capabilities. The procedure basically iterates two solving steps: (1) a *relaxation step*, where a subset of solving decisions made at iteration $(i-1)$ are randomly retracted at iteration i, and (2) a *flattening step*, where a new solution is re-computed after the previous relaxation. Over the last years, the basic optimization principle behind IFS has been used to effectively solve other classes of scheduling problems, see for example (Ruiz and Stützle 2008) for sequence dependent setup times flowshop problems. However, this paper is among the very first attempts, to the best of our knowledge, to use the IFS algorithm to solve RCPSP/max instances. Tackling such instances entails an extension of the set of temporal constraints to reason upon, which increases the consistency check complexity from polynomial to *NP-hard*. The performance of the procedure will be analysed and its effectiveness empirically demonstrated.

Reference Problem and its Representation

The Resource Constrained Project Scheduling Problem (RCPSP) has been widely studied in Operations Research (OR) literature (see (Brucker et al. 1999) for a survey). The RCPSP version with *Time Windows* (RCPSP/max) is an extended formulation of the basic problem which underlies a number of scheduling applications (Neumann and Schwindt 1997) and is considered particularly difficult, due to the presence of temporal separation constraints (in particular maximum time lags) between project activities.

The RCPSP/max The RCPSP/max can be formalized in terms of the following three sets: (1) a set V of n non-preemptive activities where each activity a_i has a fixed duration d_i. Each activity has a start-time s_i and a completion-time e_i that satisfies the constraint $s_i + d_i = e_i$; (2) a set E of temporal constraints that may exist between any two activities $\langle a_i, a_j \rangle$ of the form $s_j - s_i \in [T_{ij}^{min}, T_{ij}^{max}]$, called start-to-start constraints (time lags or *generalized precedence relations* between activities); (3) a set R of renewable resources, where each resource r_k is characterized by a maximum integer capacity $c_k \geq 1$. The execution of an activity a_i requires some capacity from one or more resources. For each resource r_k the integer $rc_{i,k}$ represents the capacity required by the activity a_i. A schedule S is an assignment of values to the start-times of all activities in V ($S = (s_1, \ldots, s_n)$). A schedule is said to be *feasible* if it is both *time* and *resource-feasible*. Solving the RCPSP/max optimization problem equates to finding a feasible schedule with *minimum makespan* MK, where $MK(S) = max_{a_i \in V}\{e_i\}$.

The Scheduling Problem Representation Formalism The class of scheduling algorithms we are focusing upon in this paper is based on a representation of the basic scheduling problem as a precedence graph $G(A, E)$ where A is the set of activities (plus two fictitious activities source a_{source} and sink a_{sink}), and E is the set of precedence constraints defined among the nodes in A. A solution S is represented

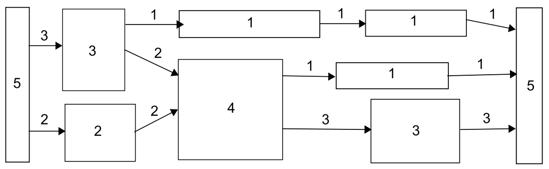

Figure 1: An example of Partial Order Schedule (POS)

as an extended graph G_S of G, characterized by an additional set of precedence constraints (or *decisions*) that are necessary to solve the original problem. The solution constraints contained in the graph G_S can be produced in different ways. Among the possible types of solution there is the so-called *Partial Order Schedule* (POS) used in one of the relaxation procedures for IFS described below. The theoretical aspects of the POS are outside the scope of this paper, for more details the reader can refer to (Policella et al. 2007). An example of POS for a single resource with capacity $c = 5$ is depicted in Figure 1. Activities are represented as rectangles and edges represent the precedence constraints. The numbers inside the rectangles represent the resource requirements and the labeling numbers on the directed edges represent the flow of resource units supplied to a generic activity a_i from its predecessors in order to satisfy the imposed resource constraint. In the figure, the activity requiring 4 units of resource receives 2 units of resource from each of its two predecessors and supplies 1 and 3 units of resource respectively to its two successors.

Iterative Flattening Search

In this section we introduce a general IFS procedure, as depicted in Figure 2. The algorithm basically alternates relaxation and flattening steps until a better solution is found or a maximal number of iterations is reached. The procedure takes two parameters as input: (1) an initial solution S; (2) a positive integer $MaxFail$ which specifies the maximum number of non-makespan improving moves that the algorithm will tolerate before terminating. After initialization (Steps 1-2), a solution is repeatedly modified within the while loop (Steps 3-10) by the application of the RELAX and FLATTEN procedures. In case a better makespan solution is found (Step 6), the new solution is stored in S_{best} and the *counter* is reset to 0. Otherwise, if no improvement is found within $MaxFail$ moves, the algorithm terminates and returns the best solution found.

The Relaxation Procedures Two relaxation strategies have been used in this work: the first strategy (IFS-CP) is based on the *critical path* analysis, while the second strategy (IFS-CH) is exclusively applied on solutions in the POS form. At each relaxation step, the IFS-CP procedure is controlled by two parameters: n_r determines the number of individual relaxation attempts performed at each relaxation cycle, and for each attempt, p_r determines the percentage of decision constraints that will be randomly removed from the critical path. On the other hand, the relaxation procedure within IFS-CH proceeds in two steps and the only single p_r parameter takes a slightly different meaning. First, some activities whose number is determined by p_r (percentage of activities to be removed) are randomly selected and eliminated from

```
IFS($S$,$MaxFail$)
1.  $S_{best} \leftarrow S$
2.  $counter \leftarrow 0$
3.  while ($counter \leq MaxFail$) do
4.      RELAX($S$)
5.      $S \leftarrow$FLATTEN($S$)
6.      if MK($S$) < MK($S_{best}$) then
7.          $S_{best} \leftarrow S$
8.          counter $\leftarrow$ 0
9.      else
10.         counter $\leftarrow$ counter + 1
11. return ($S_{best}$)
```

Figure 2: The IFS general schema

S [1]. Secondly, a new POS-form solution is recomputed from the remainig activities.

The Flattening Procedure This step aims at *levelling resource demands by posting precedence constraints*. Resource constraints are super-imposed by projecting "resource demand profiles" over time. The detected resource conflicts, once reduced to Minimal Conflict Sets (MCS, see (Laborie and Ghallab 1995) for further details), are then resolved by iteratively posting simple precedence constraints between pairs of competing activities ((Cesta, Oddi, and Smith 2002)). The procedure iteratively propagates the current temporal constraints, and then proceeds to select a resource conflict (one MCS). If no conflict exists then a solution is found. If a conflict exists that can be solved, a new precedence constraint is posted; otherwise the process fails, meaning that the temporal constraints currently posted in the solution disallow the separation of any pair of activities belonging to the currently selected MCS.

Experimental Analysis

In this section we present the results of the experiments to assess the performances of the IFS procedure against RCPSP/max instances. The empirical analysis has been organized as follows. The RCPSP/max benchmarks that have been chosen for the present investigation are taken from well known test sets available at `www.wior.uni-karlsruhe.de/LS_Neumann/Forschung/ProGenMax/rcpspmax.html`, namely the UBO-200 and the J30, both generated through the project generator ProGen/max ((Kolisch, Schwindt, and Sprecher 1998)). The J30 set is composed of 270 problem instances, and represents a rather challenging benchmark despite the relatively small size of each instance (30 activities, 5 multicapacity resources), while the UBO-200 problems are composed of 90 RCPSP/max instances each made up of 200 activities and 5 multicapacity resources. The optimal solutions for many instances of this benchmark are still unknown. In this analysis, the J30 set of RCPSP/max have been initially solved with both the precedence relaxation (IFS-CP) and chain relaxation (IFS-CH) versions of the IFS procedure. Both procedures are fed with initial solutions computed through the same algorithm used in the flattening

[1]It is worth observing that the remainig activities still represents a feasible solution to a scheduling sub-problem, which can be transformed into POS-form, in which the randomly selected activities *float* outside the solution thus re-creating *contention peaks*.

step. During this first phase it was also performed an exploration over the parameters n_r and p_r. Secondly, the same solving procedures have been used to tackle the whole set of RCPSP/max UBO-200 benchmark by using the value of the parameters which gave the best results during the first phase. The efficacy of all strategies is compared against the best known results published at the above mentioned URL.

The behavior of the proposed IFS strategies is evaluated according to the following metrics: (1) *Number of Improved Makespans*, i.e., the number of instances where the IFS algorithm succeeded in improving the makespan value w.r.t. the current best; (2) *Average Makespan Gap* Δ_{mk}: this metric returns the average percentage makespan gap between the best published results (mk_i^0) and our experimental results, for all the n instances belonging to the benchmark set which have a feasible solution; (3) *Average CPU Time*; (4) *Average* IFS *Cycles*, which returns the average number of $\{$*relaxation - flatten*$\}$ cycles performed by IFS to solve all instances.

All the proposed IFS procedures have been implemented using the JAVA libraries provided by the *Timeline-based Representation Framework*, a general modeling framework for Planning & Scheduling problem fast prototyping. The experiments have been performed on an AMD Athlon machine 2,4 Ghz, 3 GB ram, under Linux Ubuntu 8.04.

Experiments on the J30 Set In this analysis we set the $MaxFail$ parameter of the algorithm (see Figure 2) to *100* and particular attention has been dedicated to assessing the best values of (1) the number of relaxations (n_r) for the IFS-CP procedure (the value of p_r related to the critical path is set to 20) and (2) the percentage of activities to eliminate (p_r) for the IFS-CH procedure. To this aim, the J30 benchmark set has been solved seven times with the IFS-CP procedure (with $n_r = 2, 3, 4, 5, 6, 7, 8$), and seven times with the IFS-CH procedure (with $p_r = 10, 15, 20, 25, 30, 35, 40$). Since there are no assessed results in literature about solving RCPSP/max instances with IFS, it was necessary to investigate whether the best values of such parameters would somehow differ from the best values used to solve problem instances without time windows (RCPSP and MCJSSP[2]).

Table 1 shows the results of the experiments. For the IFS-CP case, the best value of the n_r parameter is around $6, 7$; this value is higher than the values *4-6* used in (Michel and Van Hentenryck 2004) for MCJSSP instances. For the IFS-CH case, the best value of the p_r parameter is around $30, 40$; again, these values are definitively higher than the figure *20* used in (Godard, Laborie, and Nuitjen 2005) to obtain the best performances when solving MCJSSP instances. The previous results clearly indicate that the presence of maximum constraints requires a deeper relaxing action in the IFS procedure. As Table 1 shows, the overall performance of IFS reveals quite good. The best performing procedure is IFS-CH(35) which gives a value $\Delta_{mk} = 9.75$, against a value of Δ_{mk} equal to 8.91 for the currently published best solutions. However, since IFS is a stochastic procedure it makes sense to run it several times (as we did in Table 1) and to take the average value of the overall best output solutions. Indeed, the best overall value for the IFS-CP procedure is 9.88, whereas the best overall value for IFS-CH is 9.02. When we consider the best values for both the procedures (last row of Table 1), we obtain the average value 8.9, which is equal to the current best results. As the table shows, the average

[2]Multiple Capacitated Job Shop Scheduling Problem.

Table 1: Results on J30 benchmark

Strategy	n_r/p_r	Impr.	Δ_{mk}	Cpu	Cycles
IFS-CP(n_r)	2	2	12.09	59.09	124.83
	3	7	11.72	58.47	122.28
	4	7	11.28	61.76	122.60
	5	6	11.64	42.53	119.77
	6	4	10.98	55.79	121.22
	7	6	10.97	57.38	118.69
	8	4	11.05	60.40	122.36
Best	-	9	9.88	-	-
IFS-CH(p_r)	10	4	11.61	40.28	141.09
	15	5	11.16	45.75	133.37
	20	7	10.49	47.82	131.27
	25	5	10.25	49.00	127.47
	30	11	9.85	52.16	128.61
	35	9	9.75	53.06	128.47
	40	6	9.91	54.45	123.90
Best	-	16	9.02	-	-
BEST	-	16	8.90	-	-

CPU time to obtain a solution ranges from 42 to 61 seconds, though the cpu time is not the major issue; in fact, the attention should rather be focused on the low number of $\{$*relaxation - flatten*$\}$ cycles necessary to converge (see below a similar observation about the CPU time related to the UBO200 instances).

Finally, it is interesting to analyse the behavior of IFS as the disturbing factor (n_r and p_r) in the relaxing step varies. As can be observed, an initial increase in both parameters is associated to an improvement in the quality of the obtained solutions; but if the increase continues, eventually the trend reverses after reaching a maximum. This behavior can be explained as follows: high values of the n_r and p_r parameters correspond to heavy disruptions of the solutions at each IFS cycle, eventually resembling the exact behavior (and the performances) of a *full* Iterative Sampling procedure (see for example (Cesta, Oddi, and Smith 2002)), which recomputes the solution from scratch at each iteration. In general, we can see an IFS procedure as an *incremental* Iterative Sampling procedure that destroys at random only a part of the current solution (or that maintains a partial memory of the previous solution), from which a new solution is recomputed. ¿From the results presented in Table 1 we can make the guess, that given the procedure IFS-CH(p_r) and considered the performance of IFS-CH(100) (i.e., full Iterative Sampling), there always exists a value $0 \leq p_r^* \leq 100$ which improves over the full iterative sampling procedure.

Experiments on the UBO200 Set Based on the experimentation performed on the J30 set, the n_r and the p_r parameters have been set to *7* and *35* in the IFS-CP and the IFS-CH case, respectively. In both cases, the $MaxFail$ parameter has been set to *400*. The experiment results are summarized in Table 2. The table is organized as follows: the first two columns, with names IFS-CP and IFS-CH, represent the performance of the proposed IFS algorithm. The last column, with name BEST, represents the results obtained merging the best performances of the previous two columns.

Among the two procedures, the better behavior exhibited by IFS-CH w.r.t. IFS-CP with the J30 set is again confirmed with the UBO200. In Table 2, the first result that catches the eye is the significant number of makespan inprovements

Table 2: Experimental results on the UBO200 set

Metrics	IFS-CP	IFS-CH	BEST
No. improved MKs (Impr.)	14	21	**27**
Avg. Makespan Gap (Δ_{mk})	1.47	0.02	**-0.44**
Avg. Cpu time (Cpu)	6867.08	13395.42	-
Avg. IFS Cycles	580.04	693.38	-
No. new optimal solutions	2	1	**3**

that has been obtained; on a total of 90 problems, IFS-CP succeeds in improving *14* instances (i.e., more than *15%*), with a correspondent value of $\Delta_{mk} = 1.47$. On the other hand, IFS-CH improves *21* instances (i.e., more than *23%*) with a value of $\Delta_{mk} = 0.02$. Globally, *27* instances have been improved with an overall $\Delta_{mk} = -0.44$, therefore improving over the best known results and finding three new optimal solutions (i.e., by equalling the solution's published lower bound). This circumstance is remarkable and leads to the following conclusion: the IFS approach can be effectively used also against the RCPSP/max benchmark. Its efficacy is mainly proved by the fact that all experimental runs have been performed with a low value of the $MaxFail$ parameter (*400*), where in general a value around the tenths of thousand is employed.

As for the algorithm convergence speed, it should however be noted that IFS-CP converges faster than IFS-CH; the explanation is simple: acting on the critical path, every search move of IFS-CP is more efficient, as it directly aims at reducing the solution's makespan. On the other hand, IFS-CH's search moves act over the whole solution; as a consequence, its convergence speed towards the smallest makespans is lower. As the table shows, the average time to obtain a solution with IFS-CP is about *6900* seconds (around *115* minutes), while the average time to obtain a solution with IFS-CH is about *13400* seconds (around *223* minutes). The reader should not be misled by the high solving time: in fact, tackling the RCPSP/max entails an increase in computational complexity, with respect to solving RCPSP instances without time windows. Nonetheless, a considerable boost might be obtained by using the fastest versions available of the All-Pair Shortest Path propagation algorithms, which is outside the scope of this paper. Again, the reader should look at the low number of $\{$*relaxation - flatten*$\}$ cycles necessary to converge to a good solution; since this figure is independent from all machine-dependent factors, it should in fact be regarded as one of the fairest efficiency measure. As shown, the average number of solving cycles is between *580* and *700* for all strategies, indeed a very low value, which definitely confirms the efficacy of the IFS approach.

Conclusions and Future Work

This work analyses the role of IFS strategies in solving Resource Constraint Project Scheduling Problem with Time-Windows (RCPSP/max). RCPSP/max represents a hard and general scheduling problem, such that even the search of a feasible solution is *NP-hard*. The performed experimental analysis has revealed that the proposed IFS strategies can be very effective in solving challenging and large-size RCPSP/max instances. Our experimental analysis demonstrates that IFS requires on average a higher degree of random disruption than in the case without time windows. In addition, the efficacy of IFS is particularly demonstrated if one considers the extremely low number of solving cycles (only a few hundreds!) that are necessary to converge to good solutions.

The results proposed in this paper have also paved the way for future research work on the definition of more effective solving procedures for large size benchmarks. As a further extension of the present work, we plan to study the effects of different flattening and relaxation procedures within the IFS loop, where key issues for improving effectiveness will be the use of efficient temporal and resource propagation algorithms as well as the definition of different strategies to retract decision constraints from the current solution.

Acknowledgments. Angelo Oddi and Riccardo Rasconi's work is partially supported by CNR under project RSTL (funds 2007), ESA (European Space Agency) under the APSI initiative and EU project ULISSE (Call "SPA.2007.2.1.01 Space Science". Contract FP7.218815).

References

Bartusch, M.; Mohring, R. H.; and Radermacher, F. J. 1988. Scheduling Project Networks with Resource Constraints and Time Windows. *Annals of Operations Research* 16:201–240.

Brucker, P.; Drexl, A.; Mohring, R.; Neumann, K.; and Pesch, E. 1999. Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods. *European Journal of Operations Research* 112(1):3–41.

Cesta, A.; Oddi, A.; and Smith, S. F. 2000. Iterative Flattening: A Scalable Method for Solving Multi-Capacity Scheduling Problems. In *AAAI/IAAI.* 17^{th} *National Conference on Artificial Intelligence*, 742–747.

Cesta, A.; Oddi, A.; and Smith, S. F. 2002. A constraint-based method for project scheduling with time windows. *J. Heuristics* 8(1):109–136.

Godard, D.; Laborie, P.; and Nuitjen, W. 2005. Randomized Large Neighborhood Search for Cumulative Scheduling. In *ICAPS-05. Proceedings of the* 15^{th} *International Conference on Automated Planning & Scheduling*, 81–89.

Kolisch, R.; Schwindt, C.; and Sprecher, A. 1998. Benchmark Instances for Project Scheduling Problems. In Weglarz, J., ed., *Handbook on Recent Advances in Project Scheduling*. Kluwer.

Laborie, P., and Ghallab, M. 1995. Planning with Sharable Resource Constraints. In *Proceedings of the* 14^{th} *Int. Joint Conference on Artificial Intelligence (IJCAI-95)*.

Michel, L., and Van Hentenryck, P. 2004. Iterative Relaxations for Iterative Flattening in Cumulative Scheduling. In *ICAPS-04. Proceedings of the* 14^{th} *International Conference on Automated Planning & Scheduling*, 200–208.

Neumann, K., and Schwindt, C. 1997. Activity-on-Node Networks with Minimal and Maximal Time Lags and Their Application to Make-to-Order Production. *Operation Research Spektrum* 19:205–217.

Oddi, A.; Cesta, A.; Policella, N.; and Smith, S. F. 2008. Combining Variants of Iterative Flattening Search. *Journal of Engineering Applications of Artificial Intelligence* 21:683–690.

Policella, N.; Cesta, A.; Oddi, A.; and Smith, S. 2007. From Precedence Constraint Posting to Partial Order Schedules. *AI Communications* 20(3):163–180.

Ruiz, R., and Stützle, T. 2008. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objective. *European Journal of Operational Research* 187(3):1143–1159.