A Novel Priority Rule Heuristic: Learning from Justification

Frits de Nijs and **Tomas Klos**
Algorithmics Group, Delft University of Technology
Delft, The Netherlands

Abstract

The Resource Constrained Project Scheduling Problem consists of finding start times for precedence-constrained activities which compete over renewable resources, with the goal to produce the shortest schedule. The method of Justification is a very popular post-processing schedule optimization technique which, although it is not clear exactly why, has been shown to work very well, even improving randomly generated schedules over those produced by advanced heuristics.

In this paper, we set out to investigate why Justification works so well, and, with this understanding, to bypass the need for Justification by computing a priori the priorities Justification implicitly employs. We perform an exploratory study to investigate the effectiveness of Justification on a novel test set which varies the RCPSP phase-transition parameters across a larger range than existing test sets. We propose several hypotheses to explain the behavior of Justification, which we test by deriving from them several predictions, and a new priority rule. We show that this rule matches the priorities used by Justification more closely than existing rules, making it outperform the most successful priority rule heuristic.

Introduction

The Resource Constrained Project Scheduling Problem (RCPSP) has many important applications, e.g. in manufacturing, maintenance, staff scheduling and school-timetabling. An instance of the problem consists of a set of precedence-constrained tasks that compete for renewable resources, and we are asked to assign starting times to these tasks (a 'schedule') that satisfy all constraints. The RCPSP is also an intractable problem, hence an enormous body of work has developed over the past decades, in which a great variety of heuristic approaches has been proposed and evaluated (see the review in (Kolisch and Hartmann 2006), for example). A procedure called 'Justification' (Valls, Ballestín, and Quintanilla 2005), or 'Forward-Backward Improvement (FBI)' (Tormos and Lova 2001) has become a standard building block of many methods. Justification is a post-processing technique, that is applied to an existing schedule: It 'shifts' all activities to the right, and then back to the left, which often leads to a schedule with lower makespan than the input schedule. The comparison of heuristics in (Kolisch and Hartmann 2006) shows that the top performing heuristics almost invariably make use of FBI. What is not clear, however, is *why* Justification works so well. Recent experimental studies of heuristics for the RCPSP (and other computational problems), unfortunately *still* "resemble track meets more than scientific endeavors", even now, almost 20 years after Hooker published his paper "Testing Heuristics: We Have it All Wrong" in the inaugural issue of the *Journal of Heuristics* (Hooker 1995, p. 33). The typical paper proposes a heuristic and compares it with other heuristics on the standard PSPLIB benchmark set, after which a paper is submitted if the results are favorable (and not submitted, or rejected otherwise). Papers that scientifically investigate a heuristic in order to understand why it works well are rare.

In (Hooker 1994), Hooker calls for a *scientific* approach to the study of algorithms, and exemplifies it in (Hooker and Vinay 1995) with a study of a branching rule for a complete solver for the Satisfiability problem. We also adopt this approach here. First we describe the RCPSP problem in detail, along with (classes of) solutions and some algorithms for obtaining them, including FBI. We continue with an exploratory study of the FBI technique, after we have argued for and introduced a new test set of instances. Our exploration leads to observations about the behavior of FBI that we want to explain. We propose hypotheses that explain the behavior we observed. Since we can not directly observe these hypotheses, we derive expectations from them which we put to the test in controlled experiments. One expectation is that a novel priority rule heuristic we designed should work well, and we find in our experiments that indeed it does.

Problem Definition

An instance of the Resource Constrained Project Scheduling problem (RCPSP) is given by the tuple $I = (A, R, P, d, u, c)$, where $A = \{a_1, \ldots, a_n\}$ and $R = \{r_1, \ldots, r_m\}$ are the sets of *activities* and *resources*, respectively, and $P \subseteq A \times A$ is a set of precedence constraints: If $(a_i, a_j) \in P$ (which we also write as $a_i \prec a_j$), then activity a_i must complete before a_j is allowed to start. The value $d(a_i) \in \mathbb{N}^+$ is a_i's duration, while $c(r_k) \in \mathbb{N}^+$ is r_k's capacity. The value $u(a_i, r_k) \in \mathbb{N}$ says how many units of resource r_k activity a_i needs (during its entire duration). As is customary, we define two special activities

a_0 and a_{n+1} (both with zero duration and zero resource usage) that signify the project start and end, respectively: $\forall a_i \in A : a_0 \prec a_i \wedge a_i \prec a_{n+1}$.

A schedule for an instance I is a function $s : A \to \mathbb{N}$ that assigns a start time to each activity (we let $s(a_0) = 0$). Given a schedule s and a timepoint t, the function i returns the set of activities *in progress* in s at time t: $i(s,t) \mapsto \{\, a_i \mid s(a_i) \leq t < s(a_i) + d(a_i)\}$. A schedule is a *solution* if it is feasible, i.e. if it meets the precedence and resource constraints:

$$s(a_i) + d(a_i) \leq s(a_j) \qquad \forall (a_i \prec a_j) \in P \qquad (1)$$

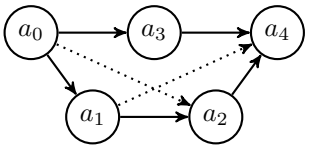
$$\sum_{a_i \in i(s,t)} u(a_i, r_k) \leq c(r_k) \qquad \forall t \in \mathbb{N}, \forall r_k \in R \qquad (2)$$

The RCPSP asks for a solution that minimizes the project makespan $s(a_{n+1})$, and finding one is an NP-hard problem (Blazewicz, Lenstra, and Rinnooy-Kan 1983).

The Earliest Start schedule s_{EST} is the shortest possible schedule that satisfies only the precedence constraints (1). It can be computed in polynomial time using a breadth first search, and it is interesting because $s_{\text{EST}}(a_{n+1})$ is a lower bound on the minimum makespan. An analogous notion is the Latest Start schedule s_{LST}. Letting $s_{\text{LST}}(a_{n+1}) = s_{\text{EST}}(a_{n+1})$, schedule s_{LST} is computed by recursively starting each directly preceding task as late as possible, taking into account only the precedence constraints (1).

Visualizing Instances and Solutions

Figure 1 presents an example RCPSP instance with 3 activities and 2 resources. Solutions to the RCPSP are often pre-



	d \ c	r_1	r_2
		4	3
a_1	1	3	0
a_2	1	2	2
a_3	2	2	1

Figure 1: An example RCPSP instance.

sented using Gantt charts (Wilson 2003)—one for each resource in R. Figure 2 presents the earliest start solution s_{EST} in the two Gantt charts on the left. (For clarity, we extend the Gantt charts with all non-redundant precedence constraints.) As the Gantt chart for r_1 shows, s_{EST} is not resource feasible: activities a_1 and a_3 together overuse resource r_1. Starting a_3 one time unit later gives the optimal makespan of 3.

Classes of solutions

(Sprecher, Kolisch, and Drexl 1995) identifies a hierarchy of classes of solutions:

$$\text{NDS} \subseteq \text{AS} \subseteq \text{SAS} \subseteq \text{FS},$$

which are the sets of Non-Delay, Active, Semi-Active, and Feasible Schedules, respectively. In an Active schedule, every activity is started as soon as possible while satisfying

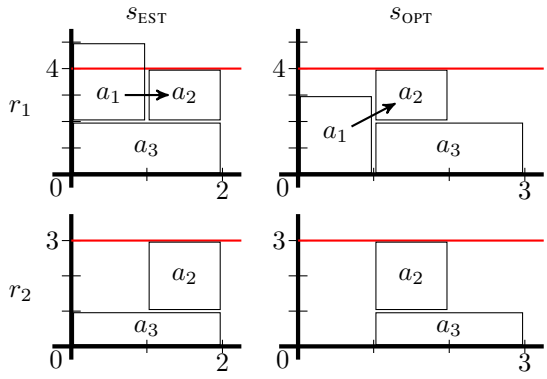


Figure 2: Two schedules for the instance in figure 1.

both the precedence and resource constraints. Non-Delay Schedules are schedules in which no activity could resource and precedence feasibly start any sooner even if we relax the constraint that activities may not be preempted. The set of Active Schedules always contains an optimal solution, while the set of Non-Delay Schedules may not contain an optimal solution (Kolisch 1996; Sprecher, Kolisch, and Drexl 1995). The solution s_{OPT} in figure 2 is a Non-Delay Schedule, while delaying activity a_2 by one results in a Feasible Schedule.

Algorithms

This section describes heuristic algorithms for finding solutions to instances of the RCPSP. The Serial and Parallel Schedule Generation Schemes (see (Kolisch 1996)) are greedy algorithms that incrementally construct a schedule using a Priority Rule heuristic to determine which activity to schedule next, from a set of eligible activities $D \subseteq A$ (an activity is eligible if all of its predecessors have already been scheduled).

Priority Rules

The priority rule that has been identified in (Kolisch 1996) and (Tormos and Lova 2001) to consistently result in good schedules is the minimum Latest Finish Time (LFT) heuristic. This rule computes a Latest Start Time schedule s_{LST} and then prioritizes the activity with $\arg\min_{a_i \in D} (s_{\text{LST}}(a_i) + d(a_i))$ the highest. This task can be seen as the most urgent, since it is followed by the longest critical chain of activities.

Another priority rule that has been shown to work well is the maximum Total Successors (TS) heuristic. It selects the activity in D that has the largest number of successors. An activity a_j is a successor of a_i if there exists a path $a_i \prec \ldots \prec a_j$. Just as with the LFT heuristic, this is intuitively an urgent task, since there are many activities waiting for it.

Finally, the Random Priority Rule is often used as a baseline with which to compare more informed heuristics. It simply selects a random activity from the set D.

Schedule Generation Schemes

The Serial Schedule Generation Scheme (Serial SGS) is a single pass greedy algorithm for constructing a schedule

for an RCPSP instance. The algorithm incrementally schedules a single activity in each iteration, at the earliest possible resource feasible time, without backtracking on the start times of already scheduled activities. The activity that is selected in each iteration is the activity with the highest priority among all the activities whose predecessors are already scheduled.

Instead of scheduling one activity at a time, the Parallel SGS schedules as many activities as will remain resource feasible, before advancing to the next time when a task completes, again without backtracking. The activities to be scheduled in the current timestep are iteratively considered in the order determined by the priority rule heuristic.

Both the Serial and the Parallel SGS operate in time $O(n^2m)$ where n is the number of activities and m is the number of resources.

Backwards Scheduling

Sometimes, SGSs produce better schedules when the schedule is created *backwards*. An RCPSP-instance I is first 'reversed,' creating the instance $I' = (A, R, P', d, u, c)$ in which all precedence constraints are reversed—setting $P' = \{a_j \prec a_i \mid \forall a_i \prec a_j \in P\}$—and then solved, after which a solution to the backwards instance is transformed to be valid for the original instance. Suppose that schedule s' was found for I', then s is easily constructed for I as follows:

$$s(a_i) = s'(a_0) - (s'(a_i) + d(a_i)) \quad \forall a_i \in A$$

Forward-Backward Improvement (FBI)

Many scheduling algorithms start the activities as early as possible, even when some activities have some 'slack', i.e. they *could* start later without adversely affecting the makespan. Tasks with slack thus consume resources at an earlier time than strictly necessary. By starting such a task later, such resources are freed up, which may allow more critical activities to start earlier, thus reducing the schedule's makespan. FBI (Valls, Ballestín, and Quintanilla 2005) implements this idea as '(double) justification,' by chaining two passes of the Serial SGS together, *after* a feasible schedule s has already been created. The first pass operates on the *reversed* instance by using the finish times of activities in s as priorities, which means that the activity to finish latest in s is scheduled first in the justified schedule s'. The second pass operates on the *original* instance, now using the finish times of activities in s' as priorities.

Consider the example instance in figure 3. Suppose we have the solution s_{SGS} in Figure 4 (left). Activities a_5, a_6 and a_7 have slack, since they could start up to 3 time units later. But because a_5 starts as early as possible, it prevents activity a_8 from running in parallel with a_1, which results in an inefficient schedule. Figure 4 (center) shows the solution s_{BWD} which is the result of applying the Serial SGS on the reversed instance, using the finish times of activities in s_{SGS} as priorities (the direction of time is shown from right to left). In this solution there is sufficient room to schedule a_8 on top of a_1 because the $a_5 - a_6 - a_7$ chain was pushed to the right. Finally, scheduling the original instance with finish times in s_{BWD} as priorities results in solution s_{FWD} (Figure 4, right), which is $\frac{1}{3}$ shorter than the first solution s_{SGS}, by exploiting the room created early in the schedule.

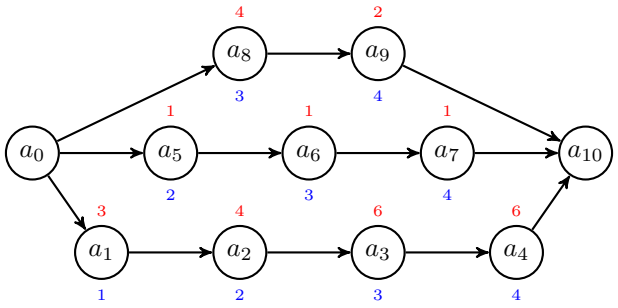


Figure 3: RCPSP instance with equal-duration activities, one resource of capacity 7. Resource usage above each activity (red), LFT below (blue).

Generating Instances

To obtain broad insight into the behavior of the Justification algorithm, we want to perform an exploratory study, in which we assess FBI's behavior on a variety of RCPSP-instances. In this section, we discuss some drawbacks of the commonly used PSPLIB instances, and argue the need for a new test set, and introduce it. The next section describes the setup and results of the exploratory study.

Phase Transition Parameters

All NP-complete problems are conjectured to possess at least one order parameter that determines the difficulty of solving an instance through its degree of constrainedness (Cheeseman, Kanefsky, and Taylor 1991). Furthermore, this order parameter is conjectured to have a critical region in which really difficult problems exist. For the 3-Satisfiability problem, for example, the order parameter is the ratio of the number of clauses to the number of variables in the instance, and at a critical value of this ratio (about 4.3), about 50% of randomly generated instances are satisfiable (Mitchell, Selman, and Levesque 1992; Monasson et al. 1999). This coincides with a sharp increase in the computational effort required to established (un)satisfiability.

Many measures have been proposed to predict how hard an RCPSP instance will be to solve optimally. Evidence of a difficulty phase transition for RCPSP is presented in (Herroelen and De Reyck 1999), on two parameters called Resource Strength and Resource Constrainedness. They also demonstrate a linear difficulty decrease in Order Strength. This section provides the definitions of these three measures.

Resource Strength (RS) This metric intends to measure the size of resource conflicts. For all resources, it computes the ratio of the resource's capacity to its highest peak usage in the Earliest Start Time schedule. Let $u_{\text{PEAK}}(r_k)$ be resource r_k's peak usage in s_{EST}, and $u_{\text{MAX}}(r_k)$ the maximum usage of r_k by a single activity. Then the Resource Strength

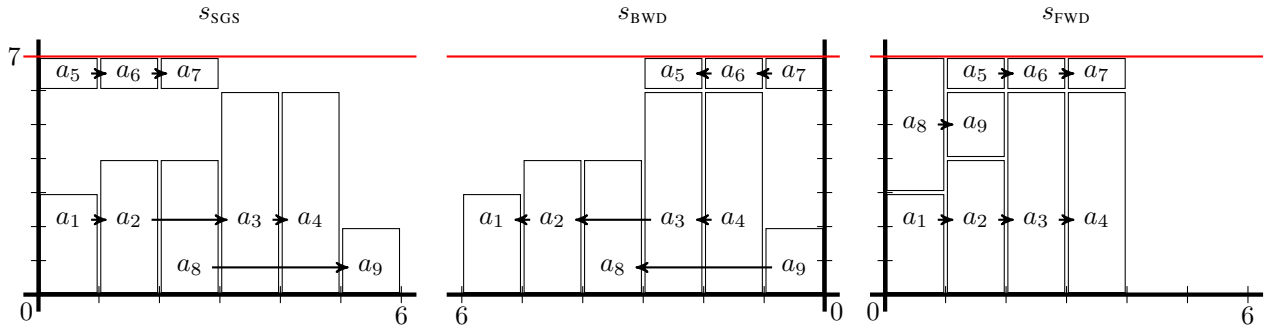


Figure 4: Left: LFT solution to the instance in Figure 3. Middle: The FBI backward pass. Right: The FBI forward pass.

is computed as follows:

$$u_{\text{PEAK}}(r_k) = \max_t \Big(\sum_{a_i \in i(s_{\text{EST}},t)} u(a_i, r_k) \Big),$$

$$u_{\text{MAX}}(r_k) = \max_{a_i} \big(u(a_i, r_k) \big),$$

$$\text{RS}_k = \frac{c(r_k) - u_{\text{MAX}}(r_k)}{u_{\text{PEAK}}(r_k) - u_{\text{MAX}}(r_k)}.$$

To compute the Resource Strength of an instance the values per resource are averaged.

Resource Constrainedness (RC) Compared to RS, this alternative resource metric aims to be a more pure measure of the resource constraints. It does not use precedence relations in its computation, while still giving an indication of the hardness of the instance. The RC measure for a resource r_k is defined as the normalized average consumption of the resource r_k, averaged over all activities that require the resource.

$$\text{RC}_k = \frac{\sum_{a_i \in A} u(a_i, r_k)}{|\{\, a_i \mid u(a_i, r_k) > 0\}| \times c(r_k)}.$$

Like Resource Strength it is averaged over all resources to obtain a measure for the complete instance.

Order Strength (OS) The Order Strength of an instance is a measure of the constrainedness of the network of precedence relations. If we call $\overline{P}$ the transitive closure of P, then the Order Strength is computed as:

$$\text{OS} = \frac{|\overline{P}|}{\frac{n^2-n}{2}}.$$

Test Set Generation

Researchers who work on the RCPSP are very familiar with the PSPLIB test sets (Kolisch, Schwindt, and Sprecher 1998), designed for the investigation of the hypothesis that small RCPSP instances cannot be hard (Kolisch, Sprecher, and Drexl 1995). This test set is extremely widely used, to the point where one can wonder whether most of the current research effort is not over-fitted to this test set.

More importantly, the PSPLIB was generated before the RCPSP phase transition parameters were studied. As such, it does not take into account the RC parameter, and instead of OS it uses Network Complexity which was shown in (De Reyck and Herroelen 1996) to have no predictive power for instance hardness.

In (Herroelen and De Reyck 1999), the authors conjecture that RS and RC are correlated such that low RS roughly corresponds to high RC. They suggest that there is only a small role for OS (and for precedence constraints in general) in determining instance difficulty. Instead, we conjecture that precedence constraints *do* play a role, and that the three measures are pairwise correlated. In particular: suppose RS is held constant, then as OS increases, fewer activities are in parallel in s_{EST} which requires higher resource usage to create peaks of the required size, corresponding to an increase in RC.

Because the PSPLIB set does not seem to cover a wide enough range of instances for the purposes of our exploratory study, we decided to generate our own test set. Based on our conjecture we generated a test set of $21,000$ instances by keeping RS fixed and varying OS between 0 and 1. The fixed values are $n = 50$, $m = 4$, RS $= 0.25$ and the probability that activity a_i uses resource r_k is 0.80. For the generation of the test set we used the RANGEN2 generator introduced in (Vanhoucke et al. 2008). There this generator was shown to produce instances with the largest precedence network diversity for a given amount of processing time. Figure 5 shows how the value for RC depends on OS for the generated instances. Because we have fixed RS, we expect to see a correlation between RC and OS, which is indeed present in the results. An increase in OS results in an increase in RC, at a Pearson correlation coefficient of 0.938.

Exploratory Study

The purpose of our exploratory study is to generate a wide range of observations on the performance of the Justification procedure. Our observations should provide us with many 'views' of the behavior of Justification, allowing us to formulate general hypotheses to explain and understand its behavior. So, for example, we will want to see what Justification does to schedules produced by the Parallel as well as the Serial SGS, even though the Parallel SGS is known to typically produce shorter schedules.

Figure 6: Mean deviation from best known solution using the LFT heuristic ('0') plus one ('1') or two ('2') justification passes.

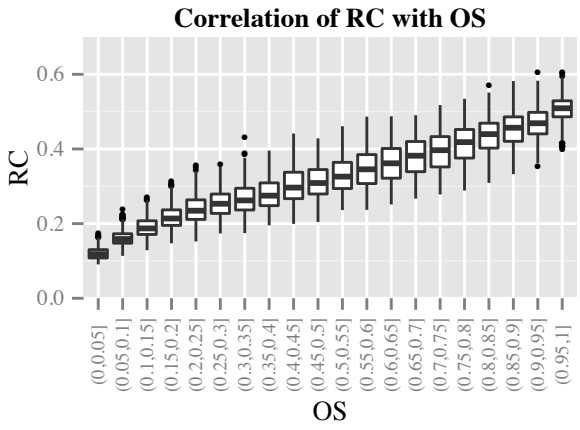


Figure 5: Correlation between OS and RC in the test set.

Experimental Design

We want to generate data showing how Justification behaves. This procedure works on an *RCPSP instance*, with a *given schedule* as input, and performs *two passes*. In these exploratory experiments, therefore, we investigate three questions: (1) How is justification affected by the characteristics of the instance? (2) How is justification affected by the choice of SGS (Serial or Parallel) used to generate the input schedule? And finally, (3) what is the interaction between the first and the second justification pass?

The setup is therefore as follows. We perform experiments on the test set described in the previous section, varying Order Strength (and by correlation Resource Constrainedness) over the range of possible values, to ensure that we can make observations regarding the first question. First, each instance in this test set is solved—using the LFT priority rule—with both the Serial (S) and the Parallel (P) SGS, allowing us to address the second question above. We use each SGS in both the forward (F) and the backward (B) direction, ensuring that any subsequent improvement obtained by Justification cannot be ascribed to the initial schedule direction being much worse than the justified direction. This provides us with four input schedules for Justification (SF, SB, PF, and PB). Then, each of these four schedules is improved with two justification passes (1 and 2). By recording the results of both justification passes we can address the third question.

We are interested in the quality of the solutions produced, measured as the percentage increase in makespan compared to the optimal makespan. Because it is not feasible to solve every instance in the test set to optimality, we use the shortest makespan we have ever found for an instance as an approximation.

Exploration Results

Figure 6 visualizes the results, using the Parallel (Serial) SGS for producing an input schedule on the left (right). Each line is the smoothed mean performance line fitted to a scatterplot of all instances (not shown), so we had six scatterplots originally. The x-value of each datapoint in these scatterplots is the OS-value of the instance, and the y-value measures the percentage by which the produced schedule s_i is longer than the best s_{BEST}, computed as $\frac{s_i - s_{\text{BEST}}}{s_{\text{BEST}}} \times 100\%$. For the values in the 'Parallel SGS' plot, the schedule used for the '0' measurement was the shortest of PF and PB, the schedule used for the '1' measurement was the shortest of PF1 and PB1, and the schedule used for the '2' measurement was the shortest of PF2 and PB2 (and similarly for the 'Serial SGS' plot). We emphasize three observations:

1. The first justification pass lowers the mean deviation by many more percentage points for the Parallel SGS, than for the Serial SGS.
2. The improvement provided by the second justification pass is much smaller than that provided by the first pass.
3. Only extreme values of Order Strength (OS > 0.975) do not show significant improvement at the 2.5% one-sided confidence interval.

Explanation

We now try to explain the above observations, by proposing two hypotheses. Because we can not observe directly whether they are true, we derive predictions from them about

what effects we expect to find in controlled experiments. If we do find those effects, this does not *prove* that a hypothesis is true, but it provides *evidence* that it is. On the other hand, if the effect does not appear, we know the hypothesis is not true (or that something went wrong in our experiments).

Analysis of Forward-Backward Improvement

As illustrated in Figure 7, the space of Active Schedules

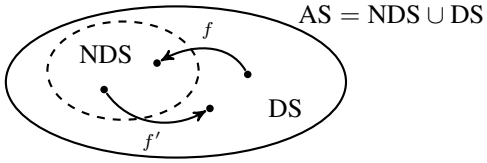


Figure 7: The solution space of Active schedules.

(AS) is divided into the set of Non-Delay Schedules (NDS) and its complement, which we call the set of Delay Schedules (DS). (We also use AS, NDS and DS as names for particular schedules in these respective sets.) Kolisch showed that the Parallel SGS produces an NDS, while the Serial SGS produces an AS, which is either a DS or an NDS (Kolisch 1996). Recall that there always exists an AS that is optimal, while it is possible that no NDS is optimal. Because FBI uses the Serial SGS, it may turn an NDS into a DS. We hypothesize that this transformation generally causes a reduction in makespan. If this is true, then the observation that FBI works better on schedules produced by the Parallel SGS is easily explained: The Serial SGS produces fewer NDSs than the Parallel SGS.

Hypothesis 1. Transforming an NDS into a similar DS results in a schedule with lower makespan, unless the NDS schedule was optimal.

To make the intuition behind the proposed hypothesis clear, we first reason from the opposite operation (f in Figure 7). Consider a function f : DS $\rightarrow$ NDS that transforms a DS s into a similar NDS, as follows. First determine for each pair of activities $(a_i, a_j) \in A^2$ if a_i is scheduled completely before, completely after or partially concurrent with a_j in s. Note that specifying this partial ordering for each pair of activities is sufficient to encode the complete schedule s. Further, identify the earliest activity a_i which can partially start at time $t < s(a_i)$. We say that this activity a_i is in the *active position*.

The proposed function f iteratively produces a new schedule s' by shifting the active position activity a_i to start at time t by setting $s'(a_j) = s(a_j)$ for $j \neq i$ and $s'(a_i) = t$, and recomputing the partial ordering. Schedule s' is infeasible due to activities a_j starting at $t' > t$ which blocked a_i from starting at t in s. To make s' feasible again, replace every partially concurrent pair (a_i, a_j) with the condition that a_i is completely scheduled before a_j. The now feasible schedule s' is similar to s (in terms of sequence) but with a_i no longer delayed. Since t was the earliest 'gap' and the first change occurs at $t' > t$, the next earliest possible gap is at $t'' >= t'$. Because start times of the active position activities considered are strictly increasing this procedure terminates, having produced an NDS schedule s'. This algorithm may increase the makespan if at some time an a_j that was part of the critical chain is pushed forward.

The inverse function f' : NDS $\rightarrow$ DS transforms an NDS s' into a similar DS. However, it is not possible to specify the inverse procedure as we have done for f for two reasons. In the first place, it is not clear which criterion we should use to select an activity a_i to *delay*, or to what time t to delay it to. Secondly, it is not clear when to terminate the procedure, since after the first iteration the objective of turning an NDS schedule into a DS schedule has been achieved. However, we conjecture that this function generally reduces the makespan of the schedule, and that the FBI algorithm performs this function on NDS schedules. We can derive a number of expectations from hypothesis 1 that can be used to test it. In the first place, we may simply look at the types of schedules before and after justification.

Expectation 1. If the application of a justification pass transforms a NDS into a DS the resulting schedule will have a lower makespan.

An alternative approach to testing the hypothesis is by altering the procedure of FBI. A defining aspect of FBI is that it uses the Serial SGS to justify the schedules. We may however try to use the Parallel SGS instead. This makes FBI produce an NDS, which means the hypothesized effect should not be present in the results, and we thus expect less improvement (expectation 2). On the other hand, if we only use the Parallel SGS for the first pass of FBI, we ensure that the second pass is always applied to an NDS. This implies that an input DS will be improved more by Parallel-Serial FBI than by Serial-Serial FBI (expectation 3).

Expectation 2. If FBI is performed with the Parallel instead of the Serial SGS, schedules will be improved less.

Expectation 3. If FBI is applied with Parallel SGS used only in the backward pass, its effect is strengthened on input Delay schedules.

Explaining the Implicit Priorities of FBI

From the exploratory study we conclude that the performance of FBI is not due to the fact that it consists of two passes. Therefore, when analyzing how FBI assigns priorities to activities, we restrict ourselves to a single justification pass. From the observation that justification works regardless of the value of OS/RC, we conclude that to explain the performance of FBI we should look for a general property of the justification algorithm.

The key insight is that justification starts by looking at a *feasible* input-schedule *from the opposite side*. Consider a solution represented in Gantt chart form. Whereas seen from the 'left' all initially feasible activities start at the same time, from the 'right' side we can differentiate between the last and the second to last activity to finish. After all it is quite unlikely that all final activities end at the same time.

Because all the schedules produced by an SGS are AS, for each activity a_i we can identify its immediate predecessors a_j. These predecessors finish exactly when a_i starts and are either related by precedences, or they use some of the resources a_i needs. Thus these predecessors are exactly those

activities that prevent a_i from starting any earlier. In any AS there is always at least one activity a_j that causes a_i to start at a later time than 0. This causality forms a *chain* of activities from 0 to $s(a_i)$.

Justification gives the highest priority to the task that finishes last (in the input schedule), which means it prioritizes the longest chain of activities. If Justification improves the schedule, it must be the case that this critical chain has become shorter. So for a schedule improved by Justification, the longest chain was delayed compared to the Earliest Start schedule s_{EST}. We call this difference between the chain length in the realized schedule and s_{EST} the *realized delay*.

Possibly, scheduling activities using priorities based on the length of their critical chain in s_{EST} plus their realized delay results in good schedules. If each activity was equally likely to be delayed compared to s_{EST}, the size of the realized delay would be a constant scale factor on the chains in s_{EST}. Then scheduling tasks by minimum LFT will in fact be equivalent to scheduling them according to their realized delay. In this case, the performance of LFT can be explained by its similarity to our hypothetical 'realized delay' priority rule.

Of course, in practice the resource usages are lopsided, and the risk of delay will differ per activity. And when there are very few precedence constraints, the resource usages almost entirely dictate the risk of delay, which, as we've seen, is a blind spot of LFT. The success of justification can be explained as follows: Every feasible schedule provides a good estimate of the risk of delay each activity has.

Hypothesis 2. Scheduling activities by prioritizing the longest expected chain of successors yields good schedules.

Instead of using a feasible schedule as an estimate, we may try to estimate a priori the risk of delay a task suffers from the resource usage peaks at its position in s_{EST}. Solving the peaks that occur in s_{EST} results in a feasible schedule, which is the basis of constraint posting solvers such as those in (Policella et al. 2007). The idea of estimating the risk of delay in s_{EST} is formalized in our new *Resource Profile Rule*.

Resource Profile Rule When a task is part of a resource peak in the s_{EST} schedule, the likelihood of delay may depend on the amount of this resource it needs. A task that needs a large amount of the resource can be delayed by a single other task, while tasks that need almost none of the resource can be scheduled alongside many other such tasks. Also, scheduling high resource consumption tasks first is likely beneficial. Therefore, assigning a higher priority to an activity that uses more of a overused resource makes sense. We estimate the delay of a task i in a EST peak on resource k as follows:

$$\text{delay}_{i,k} = \frac{\text{peak}_k - c(r_k)}{c(r_k)} \times \frac{u(a_i, r_k)}{c(r_k)}.$$

The first factor accounts for the height of the peak relative to the capacity. If the capacity is exceeded three times, it takes at least two times the average task duration to level the peak. The second factor accounts for the likelihood a task is delayed due to its use of the resource.

		NDS	DS
Parallel	NDS	19.4% of 2667	82.3% of 18333
	DS	0	0
Serial	NDS	1.2% of 2040	3.0% of 1425
	DS	17.0% of 837	60.1% of 16698

Table 1: Percentage of solutions improved out of the solutions belonging to each input-output combination.

Given an estimation of the delay a single task suffers, the length of the chain following a task is computed by determining the longest path of delays and task durations. Since the precedence graph does not contain any cycles, it suffices to compute this chain length using a recursive breadth first search.

$$\text{chain}_i = \max_{r_k \in R}(0, \text{delay}_{i,k}) + d(a_i) + \max_{(a_i \prec a_j) \in P}(\text{chain}_j).$$

The Resource Profile Rule selects the activity with the highest value for chain_i first. This priority rule should behave like FBI does, *without* first computing a feasible schedule. If hypothesis 2 is correct, we expect the following:

Expectation 4. The Resource Profile rule assigns the same priorities to activities as the Justification algorithm.

Expectation 5. The makespan of schedules produced by using the Resource Profile rule will equal those produced by FBI applied to schedules produced with LFT.

Testing

The first experiment investigates how the types of solutions affect FBI's ability to improve them (expectation 1). The second experiment tests expectation 4 by looking at the distance between the priorities assigned by justification and the rules LFT and RP. The remaining expectations are treated in an experiment on the deviation from the best known solution using the RP rule and different combinations of SGS in double justification.

Solution Types Table 1 has the type of the input schedule of the first pass of justification in the rows, and the type of output schedule in the columns. The results are presented as a percentage of schedules that was improved out of the number of schedules that belong to that input-output type combination.

We see that when justification constructs a DS (right column in the table), the output schedule is much more likely to be an improvement over the input schedule. We also see that for a large number of instances produced by the Parallel SGS, justification produces Delay schedules. This matches our expectation that justification works well on the Parallel SGS because of its ability to construct similar Delay schedules out of Non-Delay schedules. We also see that the Non-Delay schedules produced by the Serial SGS are almost never improved. The reason for this is that the Serial SGS starts producing Non-Delay schedules only when Order Strength is high. For these instances, the schedules are Non-Delay because of precedence constraints, and thus justification cannot improve them.

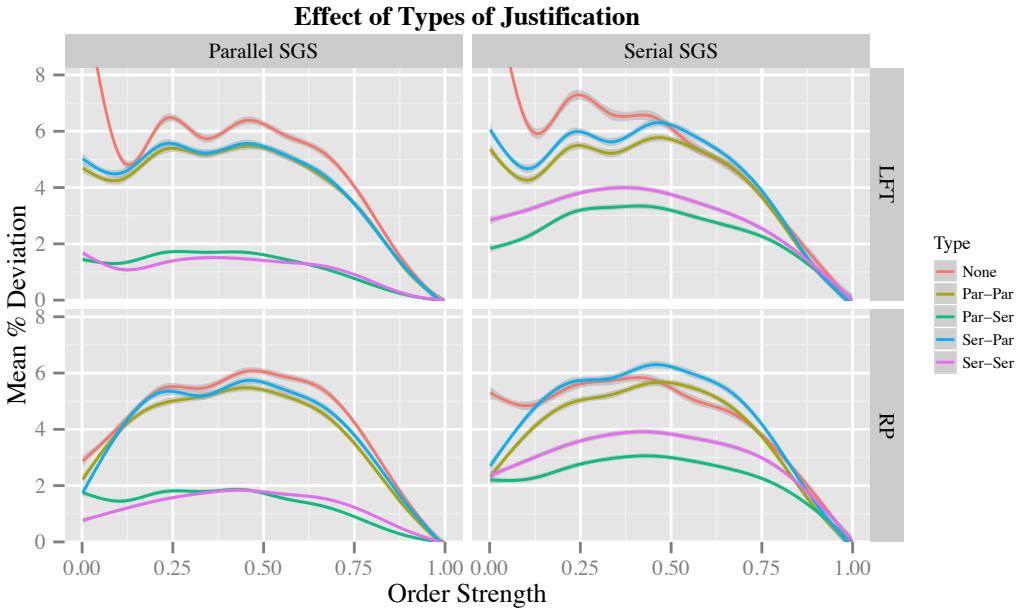


Figure 9: Effect of applying the different combinations of SGS in the justification algorithm.

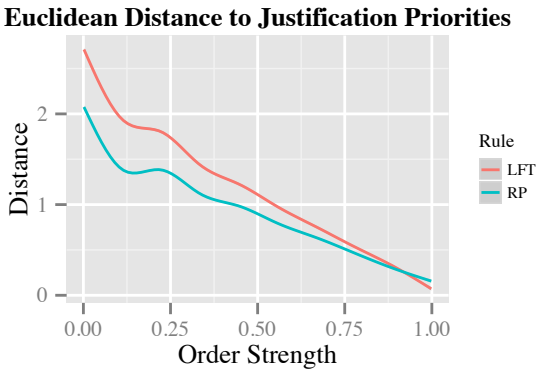


Figure 8: Euclidean distance between the normalized priorities of an FBI justification heuristic and the LFT or RP rule.

Priority Comparison We expect the RP rule to assign priorities to activities that are similar to the priorities assigned by Justification (Expectation 4). To test this we want to compare the priority assigned by RP with the priority assigned by the second pass of Justification on LFT. Because these values are not on the same scale, we normalize them such that for each rule the lowest priority task has priority value 0, while the highest has value 1. In this way a priority rule defines a vector of priorities that is independent of the SGS used. The Euclidean distance between the vectors of two priority rules measures their similarity. Figure 8 presents the smoothed mean Euclidean distance between the RP and LFT rules and the second justification pass.

As expected, the distance between priorities assigned by RP and Justification is shorter than between LFT and Justification. So the RP rule seems to assign priorities like Justification. However, there is still a lot of room for improvement in the low Order Strength region.

Modified Justification To test expectations about the way justification improves schedules through its solution types, we implemented double justification with each SGS in each of the two passes, for four different variants. Each of these four implementations was applied to the schedules obtained earlier by using the LFT and RP heuristics in combination with the SGSs. The results are in figure 9. The line 'None' in the top figures is equal to the line '0' in figure 6, while 'Ser-Ser' is equal to '2' there.

The figure shows that the similarity of RP with justification also translates into a significant improvement in mean deviation: 4.16% compared to 4.98% of LFT (combined over the SGSs). This is in line with expectation 5, although because the priorities do not match exactly, performance is less than that of justification.

On the types of solutions produced, we see that using the Parallel SGS for the final justification pass results in almost no improvement. This is what we expected (expectation 2), and means that a large part of the performance of Justification comes from its ability to generate Delay schedules. We find further confirmation of this in the way the 'Parallel-Serial' justification algorithm interacts with the solutions produced by the Serial SGS. By applying the Parallel SGS in the first pass, the second Serial pass is able to improve these schedules more than using two Serial passes, which matches expectation 3.

Conclusion

We have used an empirical approach to study the Justification post-processing algorithm and its interaction with the Schedule Generation Scheme algorithms. Our improved understanding has led to a new priority rule that mimics the principles we hypothesize Justification uses. By testing the performance of this new priority rule against the best known priority rule we not only provided evidence that our hypotheses are correct, but as a consequence also demonstrated that this new rule performs significantly better. Our research suggests that Justification is a suitable approach to bridge the gap between Non-Delay Schedules and Active Schedules, typically improving them. Perhaps more importantly than the individual results, is the result that following an empirical approach to study algorithms pays off in terms of both increased understanding *and* improved algorithms.

References

Blazewicz, J.; Lenstra, J.; and Rinnooy-Kan, A. 1983. Scheduling subject to resource constraints: classification and complexity. *Discr. Appl. Math.* 5(1).

Cheeseman, P.; Kanefsky, B.; and Taylor, W. M. 1991. Where the Really Hard Problems Are. In *Proc. IJCAI*.

De Reyck, B., and Herroelen, W. S. 1996. On the use of the complexity index as a measure of complexity in activity networks. *European J. OR* 91(2).

Herroelen, W. S., and De Reyck, B. 1999. Phase transitions in project scheduling. *J. OR Soc.* 50(2).

Hooker, J. N., and Vinay, V. 1995. Branching rules for satisfiability. *J. Automated Reasoning* 15(3).

Hooker, J. N. 1994. Needed: An empirical science of algorithms. *Operations Research* 42(2).

Hooker, J. N. 1995. Testing heuristics: We have it all wrong. *J. Heuristics* 1(1).

Kolisch, R., and Hartmann, S. 2006. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European J. OR* 174(1).

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.

Kolisch, R.; Sprecher, A.; and Drexl, A. 1995. Characterization and Generation of a General Class of Resource-constrained Project Scheduling Problems. *Man. Sci.* 41(10).

Kolisch, R. 1996. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European J. OR* 90(2).

Mitchell, D.; Selman, B.; and Levesque, H. 1992. Hard and easy distributions of SAT problems. In *Proceedings AAAI*. AAAI Press.

Monasson, R.; Zecchina, R.; Kirkpatrick, S.; Selman, B.; and Troyansky, L. 1999. Determining computational complexity from characteristic 'phase transitions'. *Nature* 400.

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

Sprecher, A.; Kolisch, R.; and Drexl, A. 1995. Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. *European J. OR* 80(1).

Tormos, P., and Lova, A. 2001. A Competitive Heuristic Solution Technique for Resource-Constrained Project Scheduling. *Annals of OR* 102(1-4):65–81.

Valls, V.; Ballestín, F.; and Quintanilla, S. 2005. Justification and RCPSP: A technique that pays. *European J. OR* 165(2).

Vanhoucke, M.; Coelho, J.; Debels, D.; Maenhout, B.; and Tavares, L. V. 2008. An evaluation of the adequacy of project network generators with systematically sampled networks. *European J. OR* 187(2).

Wilson, J. M. 2003. Gantt charts: A centenary appreciation. *European J. OR* 149(2).