Project Scheduling: The Impact of Instance Structure on Heuristic Performance

Master's Thesis

Frits de Nijs

Project Scheduling: The Impact of Instance Structure on Heuristic Performance

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Frits de Nijs
born in Den Haag, the Netherlands

TUDelft
Algorithmics Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.alg.ewi.tudelft.nl

© 2013 Frits de Nijs. All rights reserved.

Project Scheduling: The Impact of Instance Structure on Heuristic Performance

Author: Frits de Nijs
Student id: 1224263
Email: f.denijs@tudelft.nl

Abstract

Many meta-heuristic approaches have been suggested for or applied to the Resource Constrained Project Scheduling Problem (RCPSP). The existence of a number of highly accessible standard benchmark sets has promoted a research focus on finding anything that improves average solution quality, without investigating what effect is responsible for the improvement, or what is responsible for holding us back.

This work focuses instead on understanding the original constructive Schedule Generation heuristics and their interaction with a well known but poorly understood post-processing step called Forward-Backward Improvement that is known to almost always improve any generated RCPSP schedule. We follow an empirical investigation methodology by first observing the effect of FBI on a large generated testset. Based on these observations we explain why FBI works by means of hypotheses on its operation. These hypotheses generate predictions that we subsequently successfully test in a second round of experiments. In the process we are able to propose a novel priority rule heuristic based on the principles of FBI. We find that this new rule outperforms the current best priority rule heuristic.

Thesis Committee:

Chair:	Prof. Dr. C. Witteveen, Faculty EEMCS, TU Delft
University supervisor:	Dr. T.B. Klos, Faculty EEMCS, TU Delft
Committee Member:	Dr.ir. F.A. Kuipers, Faculty EEMCS, TU Delft

Contents

Chapter 1

Introduction

Whenever people need to come together to produce something bigger in scope than they themselves can oversee, they would be wise to set up a project structure for it. As a consequence, examples of projects are everywhere, ranging from the grand such as the Apollo missions to the humble student homework assignments.

Any project can be characterized by a number of indivisible steps, or activities, that must be completed to complete the project. Each activity takes some time to perform and frequently activities depend on or interact with each other, explicitly or implicitly imposing a sequence. Deciding when to start work on each activity *schedules* the project, making the entire sequence of activities explicit. Scheduling the activities a project is made up of can be a daunting task, and it is thus no surprise that computer assisted scheduling is a hot research topic.

1.1 The Art of Project Scheduling

The International Standards Organization provides the following definition of a project (emphasis in original):

> "*project*
> unique process, consisting of a set of coordinated and controlled *activities* with start and finish dates, undertaken to achieve an objective conforming to specific requirements, including the constraints of time, cost and resources" [15]

The process of a project can be characterized by a life cycle containing the phases of Planning, Scheduling and Control [7]. The Planning phase consists of determining what activities should be performed, how long they will (probably) take and what resources they could need. Then, the Scheduling phase deals with determining when to start a given task and which specific resources a task may use, while ensuring the constraints of time, cost and resources are met. Finally, the Control phase must ensure that the tasks are performed according to the schedule.

In reality, the three phases of a project life cycle are not cleanly separated. Rather, in the project planning phase, planners may use scheduling to determine whether all activities can

actually be performed in the given constraints. And in the control phase, controllers cannot simply rely on the base schedule to deal with all problems. Especially with the current focus on agile development and just in time delivery, the schedule must sometimes be adjusted to ensure all constraints will be met. Thus scheduling is actually at the heart of project management.

The schedule that is created in the Scheduling phase serves a number of purposes. It is the basis of all coordination for internal and external agents involved in the project. On the other hand, it also serves as baseline for monitoring project progress.

Scheduling is a very large and active research field involving Operations Research, Artificial Intelligence and Computer Science. Gains in the field of scheduling result in algorithms for producing schedules that allow the same project to be completed in less time. Delivering a project earlier with the same amount of effort results in a raw saving in fixed costs such as rent and monthly wages. However, if the deadline of the project is fixed, shorter schedules may instead allow for more slack time along the critical chain of activities to guard against delays, or an increase in the scope of the project to add more value. The many successes and breakthroughs in this field have raised the question whether or not scheduling is a solved topic [27], however as we shall see, there is still room for improved understanding.

1.2 The Model of Project Scheduling

A popular model of the scheduling problem can be found in the Resource Constrained Project Scheduling Problem, further referred to as RCPSP. In this model activities are characterized by their duration, their use of renewable resources, and their precedence relations. The duration of activities is deterministic, and known in advance for each activity. Resources are renewable, which means that they are released back to the pool of resources when an activity completes. Finally, precedence relations specify the constraints on the order activities must be performed in.

A solution to an RCPSP instance is an assignment of start times that ensures activities are performed in the order dictated by the precedences, while never using more than the resources capacities. The finish time of the last finishing activity is called the makespan of the schedule. An optimal schedule has the minimum possible makespan. Since RCPSP is a generalization of the Job Shop problem, finding a solution that minimizes the makespan is an NP-hard problem [2].

The combination of a highly relevant topic with a highly complex problem difficulty has resulted in an explosion of solution methods. Although several exact solution methods have been proposed, research focus has mostly been on heuristic approaches due to the complexity of the problem and the real need to solve large instances. Section 1.2.1 presents a quick overview of the exact methods used to solve the RCPSP, while section 1.2.2 presents an overview of the focus of this thesis, heuristic solution methods.

1.2.1 Exact Solution Methods

Up to recently, successful exact solvers were mostly based on Branch-and-Bound concepts [6, 3, 8]. Integer Programming models have also been extensively tried and developed [24, 22], but these have been less successful, either due to exponential model size or due to lack of search space pruning that is common to the Branch-and-Bound methods.

Another powerful exact solving technique, Boolean Satisfiability (SAT) solving, has until recently not been applied to the RCPSP because, as Schutt et al. put it in [26], "... modern SAT solvers can often handle problems with millions of constraints and hundreds of thousands of variables. But problems like RCPSP are difficult to encode into SAT without breaking these implicit limits." Nevertheless, Horbach [14] used a SAT encoding with a solver modified for the RCPSP to construct an exact method that improved on the state-of-the-art, evidenced by its ability to prove optimality of over 100 until then unsolved benchmark instances.

The current best exact solver however appears to be the Lazy Finite Domain solver put forward by Schutt et al. [26]. This solver is a hybrid solver that switches between a Finite Domain and a SAT view of the problem, using advanced constraint propagation techniques unique to either type of solver to prune large sections of the search space. They are able to solve 42.7% of a popular 120 activity test set optimally under a 15 second cut-off per instance with this method.

Despite the fact that exact solvers are themselves ultimately limited in usefulness due to their exponential time complexity, advances in exact solvers have resulted in useful advances in knowledge of the RCPSP itself: Research on the quality of lower bounds is on the one hand driven by Branch-and-Bound methods since they rely on them for pruning, while on the other hand, many useful lower bounds are relaxations of Integer Programming models. And constraint propagation techniques put forward by the Finite Domain research are equally valid when used to simplify the scheduling options a heuristic solver has to choose from.

1.2.2 Heuristic Solution Methods

The original heuristic methods to solve the RCPSP are the Serial and Parallel Schedule Generation Schemes (SGS) revisited by Kolisch in [17]. These methods are basic single-pass greedy algorithms that are suited to being embedded into more advanced (meta-)heuristic methods. SGS methods themselves schedule tasks according to priorities assigned to individual tasks, and it is in the assignment of priority that heuristics are involved. In principle, the Serial SGS is capable of producing an optimal schedule, if only the priorities assigned to tasks are correct [28]. However, approximation results presented in [10] have shown that for basic priority rules the ratio between the heuristic makespan and the optimal makespan is bounded from below by the square root of the number of activities n, thus pure priority rule heuristics are limited to at least a $O(\sqrt{n})$ approximation ratio on the makespan in the worst case.

For comparing the approximation performance of more advanced heuristic methods based on the SGS algorithms, Hartmann and Kolisch [11] present an excellent survey, which

was later updated in Kolisch and Hartmann [18]. Their surveys present a unique combination of a benchmark test set, a testing methodology including a stopping condition that is not based on CPU time and a sorted table of competitors. Despite the fact that the CPU time pitfall was avoided, the survey has unfortunately created a research focus on competition and the 'algorithmic track races' pitfall identified by Hooker in [13].

Indeed, the field of RCPSP has become somewhat of a playground for meta-heuristics. It almost seems that, as soon as a natural phenomenon has been converted into a meta-heuristic, it is applied to the RCPSP. Examples from 2012/2013 include Shuffled Frog-Leaping [9], Fireflies [25] and Particle Swarms [16].

This focus on competition has missed investigating a highly interesting observation already made in Kolisch and Hartmann's updated survey [18]. The best heuristic methods almost invariably make use of a post-processing technique called Forward-Backward Improvement (FBI) or Justification [29]. Despite the simplicity of the Serial and Parallel SGS, solving with *random* priority assignments, when followed by FBI, manages to surpass many more 'advanced' heuristics. The earlier mentioned examples continue this trend: Of the three meta-heuristics only the Particle Swarms paper does not mention FBI, and indeed it is also the only method that does not produce better results than random SGS sampling with FBI. It appears as if currently FBI is the dominant factor in determining the quality of heuristic solutions. In this thesis we aim to investigate the surprising performance of the FBI method: Understanding why FBI works so well may point us into the right direction for what area of the search space is interesting to search.

1.3 Research Questions

The conceptual operation of the Forward-Backward Improvement algorithm is presented in figure 1.1. The first pass of FBI uses the finish times of activities in schedule a as priorities for the Serial SGS applied backwards relative to the scheduling direction of the original schedule. In this backwards scheduling pass the final activity is started at time 0, followed by the activities leading up to the final task and so on, resulting in a reversed schedule b for the project. If we consider the activities aligned to the left in the base schedule this operation can be seen to justify all activities to the right. The second pass of FBI performs the same operation as the first, but then applied to the schedule computed in the first pass. Thus, the second pass re-justifies the activities to the left, resulting in a logical schedule c. A justification pass cannot increase the makespan [29], and thus each application of FBI gives two opportunities to improve the base schedule.

Section 1.2.2 argued that for many heuristic algorithms, post-processing schedules with the FBI algorithm results in a significant reduction in average makespan compared to those heuristics that do not use the algorithm. Two reasons make that result quite surprising: Firstly, the FBI algorithm is essentially deterministic, using randomization only to break ties. Secondly, each invocation of the FBI algorithm performs two passes of the Serial SGS. Because Kolisch and Hartmann limit the running time of heuristics by the number of scheduling passes performed, this means that heuristics that use FBI actually use only $\frac{1}{3}$ of their scheduling budget on global search.

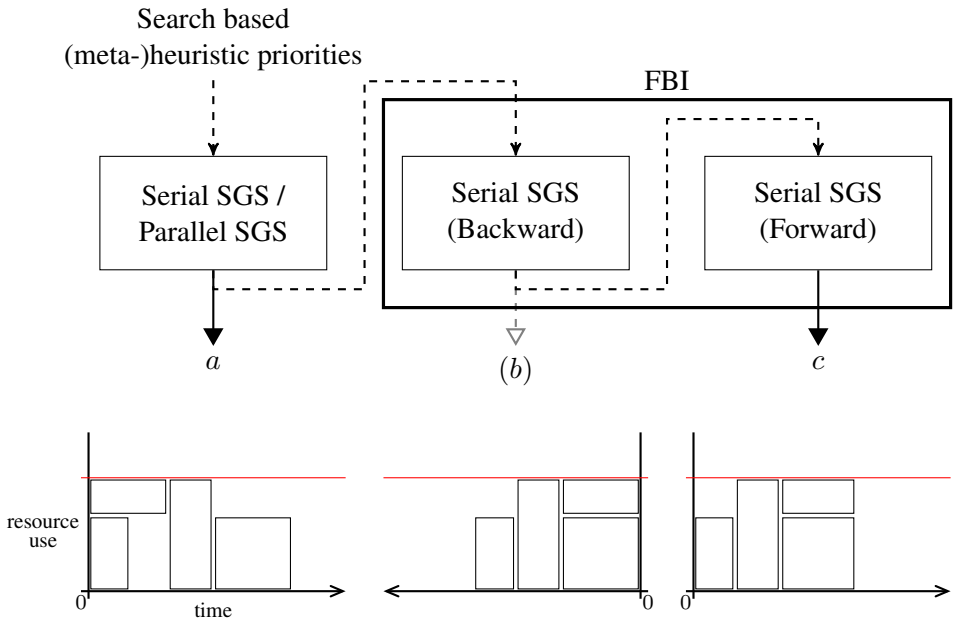


Figure 1.1: Conceptual operation of the Forward-Backward Improvement algorithm.

Since the FBI algorithm uses the Serial SGS heuristic to produce schedules the question arises if we cannot short-circuit the FBI algorithm by inputting the 'correct' priorities into the Serial SGS from the start. This would permit two immediate advantages: The full scheduling budget becomes available for search and it becomes possible to guide the global search with the 'FBI-like' priorities. This thesis will investigate this idea, formulated in the research questions 1 and 2.

Research Question 1. What underlying principle does the Forward-Backward Improvement algorithm use to assign priorities to activities?

Research Question 2. How can the principle guiding the Forward-Backward Improvement algorithm be captured as a priority rule?

Answering question 1 requires having a clear model of what FBI does to the input schedules. One prominent aspect of the algorithm is the sequential application of the two passes. An advantage of the forward pass is that the output schedule is a valid sequence for the constraints of the original instance. However it is not required to perform a second pass to correct the backward schedule b: we can simply translate the schedule by using the finish times of the tasks and setting the finish time of the last task to be the starting point. An obvious question leading to understanding the FBI algorithm is thus: how does the second pass contribute to the effectiveness of FBI? This question is formalized in research question 3.

Research Question 3. How do the backward and forward pass of the Forward-Backward Improvement algorithm interact?

Another factor that may provide insight into the workings of the FBI algorithm is the interaction of instance properties with its operation. If there exist properties of instances that make it particularly easy or difficult for FBI to improve schedules, the effect of those properties on the instance may allow us to propose hypotheses explaining why this is the case, indirectly modeling the operation of FBI. One can imagine that the degree to which activities are related by precedence constraints is correlated to the degree with which FBI can improve schedules: if tasks are unconstrained and thus 'fluid' in the time dimension, it is easy for them to hop into gaps later on in the schedule, while highly constrained instances provide almost no room for tasks to move. Research question 4 treats this aspect of the operation of FBI.

Research Question 4. How do instance properties affect the performance of the Forward-Backward Improvement algorithm?

While finding answers to research question 4 we have to ensure we include difficult instances of the RCPSP. It is known that typical instances of many NP-complete problems are actually quite easy to solve optimally [4]. Conclusions obtained about FBI only on easy instances are unlikely to hold for the difficult instances since they may be based on the inherent easiness of the instance.

All NP-complete problems are conjectured to possess at least one order parameter that controls the level of difficulty of an instance through its degree of constrainedness. Furthermore, this order parameter is conjectured to have a critical region in which really difficult problems exist [4]. When instances are generated by increasing the order parameter, phase-transition behavior is observed, resulting in instances that belong to three distinct phases: The first phase occurs for low values of the order parameter, resulting in under-constrained instances which are easy to solve. This is followed by a peak in difficulty around the critical value, where instances are highly constrained and either hard to solve, or hard to detect as unsolvable. And finally there is a region of over-constrained instances, which are easy to detect as unsolvable by enumeration of the few possible branches.

To ensure we find the right answer to research question 4 we must therefore consider instances along all three phases of the order parameter(s) of the RCPSP. This requires us to first identify which parameters of instances are order parameters of the RCPSP, which is research question 5.

Research Question 5. Which order parameters affect the difficulty of an RCPSP instance?

1.4 Thesis Outline

This chapter introduced the project scheduling domain, with its successes and challenges. Section 1.3 presented the five research questions about scheduling we intend to investigate in this thesis. This section provides an outline of the rest of the thesis, presenting what the topic of each chapter is, and where each research question is treated.

Before diving into the research questions themselves, chapter 2 provides the necessary background information for the rest of the thesis. Here the scheduling problem is defined

formally, along with the heuristic algorithms we consider in this thesis, the Serial and Parallel Schedule Generation Schemes. Further, the Forward-Backward Improvement algorithm is explained, including examples of how it is able to improve schedules. Finally, the chapter provides definitions of a number of instance properties that have been shown to control some aspects of instance difficulty.

Chapter 3 treats the subject of research question 5, controlling the difficulty of an RCPSP instance. The chapter first presents an analysis of the current known model of the interaction between instance properties and instance difficulty. We provide evidence that this model is limited because of previously unknown interactions between parameters. Thus, we propose a new model of this interaction which we test using measurements on a large generated test set in chapter 4. Finally, chapter 3 provides the methodology used for measuring the difficulty of an instance with heuristics.

The research questions 4 and 3 are the topic of chapter 4. In this chapter, FBI is applied to the schedules generated in chapter 3, resulting in improved schedules. The size of the improvement is analyzed with respect to the order parameters identified in the previous chapters to produce new insights into the FBI algorithm.

Chapter 5 subsequently uses these insights to propose hypotheses on the underlying principles FBI uses to assign priorities. This aims to answer research question 1. Given the hypotheses on the operation of FBI, the chapter proposes a new priority rule heuristic that assigns priorities to activities in the way FBI is hypothesized to assign priorities.

The final remaining research question 2 is considered in chapter 6, where the newly proposed priority rules are compared to the performance of FBI to see if they indeed manage to assign priorities like FBI as expected. Finally, chapter 7 concludes the thesis by summarizing the results and proposing recommendations on future research topics to extend this work.

Chapter 2

Concepts and Definitions

Before starting to investigate the research questions posed in chapter 1, it is required to have a formal definition of the RCPSP problem, which is provided in section 2.1. The next section defines the Serial, Parallel and Forward-Backward Improvement algorithms that will be studied in this thesis. Finally section 2.3 treats measures of RCPSP instance properties that have been used to predict the difficulty of the instance.

2.1 Resource Constrained Project Scheduling Problem (RCPSP)

An instance of the Resource Constrained Project Scheduling problem (RCPSP) is given by the tuple $I = (A, R, P, d, u, c)$. The primary elements of an RCPSP instance are the sets of *activities* A and *resources* R. The set of activities A contains n activities a_1 through a_n, while the set of resources R contains m resources r_1 through r_m.

The set of precedence constraints $P \subseteq A \times A$ consists of constraints of the form $a_i \prec a_j$ that imply activity a_i must complete before a_j is allowed to start. If the set P contains a cycle the instance is infeasible due to conflicting precedence constraints. Therefore, for feasible instances at most $|P| \leq \frac{n^2-n}{2}$ precedence constraints are imposed [20].

The function $d : A \to \mathbb{N}^+$ maps a non-zero duration to each activity, while function $u : (A \times R) \to \mathbb{N}$ maps for each activity its resource usage of a specific resource. Resources also have a positive capacity $c : R \to \mathbb{N}^+$ which specifies how much of a resource is available. If $u(a_i, r_k) > c(r_k)$ holds for any $a_i \in A, r_k \in R$ the instance is infeasible because activity a_i needs too much of resource r_k.

For convenience many authors define two special activities that signify the project start and end. Start activity a_0 occurs before all other activities, i.e. $a_0 \prec a_i \quad \forall a_i \in A$, while finish activity a_{n+1} occurs after all other activities, $a_i \prec a_{n+1} \quad \forall a_i \in A$. Both special activities have zero duration and zero resource usage to ensure they do not affect the instance in any way.

2.1.1 Solutions

A schedule for an RCPSP instance is a function $s : A \to \mathbb{N}$ that assigns to each activity a start time. Given a schedule s we can define a function i that returns the activities in progress at time t as follows:

$$i(s,t) := \{\, a_i \mid s(a_i) \leq t < s(a_i) + d(a_i)\}$$

A schedule is a solution if it is feasible, i.e. if it meets the precedence and resource constraints:

$$s(a_i) + d(a_i) \leq s(a_j) \qquad \forall (a_i \prec a_j) \in P \tag{2.1}$$

$$\sum_{a_i \in i(s,t)} u(a_i, r_k) \leq c(r_k) \qquad \forall t \in \sum_{a_i \in i(s,t)} d(a_i), \forall r_k \in R \tag{2.2}$$

Any instance that does not contain cycles, and has no single activity overusing a resource allows a solution that can be computed in polynomial time: Schedule all activities one after the other in sequence, while respecting the precedence constraints. The challenge is in finding solutions that have lower cost than the sequential schedule.

The cost of a schedule s can be measured as a function $f(s)$ of the start times. If the cost of a schedule is non-increasing for componentwise decreasing start times and the goal is minimizing the cost, function f is called a regular objective function [1]. This means f is regular if it holds that:

$$s'(a_i) \leq s(a_i) \;\; \forall a_i \in A \implies f(s') \leq f(s)$$

Typically, researchers study the minimum makespan objective function, which is a regular function. The makespan of a solution is the finish time of the last task to finish, thus:

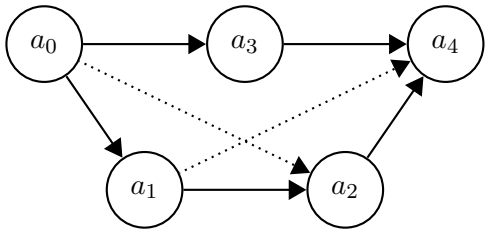
$$f_{\text{makespan}} = \max_{a_i \in A \cup \{a_0, a_{n+1}\}} (s(a_i) + d(a_i)) = s(a_{n+1})$$

Minimizing the makespan of a schedule results in a schedule which takes the shortest possible amount of time to complete, making it an attractive objective for practical purposes.

A special schedule is the Earliest Start schedule s_{EST}. This is the shortest possible schedule that satisfies only the precedence constraints (2.1). Schedule s_{EST} can be computed in polynomial time by performing a breadth first search starting with the assignment of time 0 to a_0 and recursively assigning $s(a_j) = \max_{a_i \in A} (s(a_j), s(a_i) + d(a_i)) \quad \forall (a_i \prec a_j) \in P$. It is an interesting schedule because it is the shortest possible schedule when the resource constraints are relaxed and therefore $f(s_{\text{EST}})$ is a lower bound on the lowest possible cost schedule. When the resource constraints are trivial s_{EST} is the optimal solution.

2.1.2 Visualizing Instances and Solutions

The RCPSP lends itself well to a graph representation $G(V, E)$, where the activities A are mapped on the vertices V and the precedence constraints P are represented by the edges E.



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

Figure 2.1: Example RCPSP instance of 3 activities, 2 resources.

Such a graph is called an activity-on-node (AoN) network (in contrast with the activity-on-arc (AoA) network where $A \subseteq E$). In an AoN graph a_0 and a_{n+1} function as the source and sink respectively. In this thesis, example RCPSP instances are presented using the AoN graph representation. Figure 2.1 presents an example RCPSP instance. In the figure the durations, resource capacities and usages are annotated in a table next to the network. Although there is a precedence constraint $a_0 \prec a_2$, it is not required for consistency of the graph because it is implied by transitivity: $a_0 \prec a_1 \prec a_2$. In all following examples such *redundant* precedence constraints are omitted for clarity.

The graph format represents the constraints clearly, but does not allow us to visually reason about a solution. Solutions to the RCPSP are in literature frequently presented using the Gantt chart [31], which represents each activity as a rectangle with the width set equal to the duration and the height equal to the resource usage. If the instance contains more than one resource, a Gantt chart must be created for each resource. Figure 2.2 presents the earliest start solution s_{EST} for the instance in figure 2.1 in Gantt format on the left. In this thesis, Gantt charts are extended with the non-redundant precedence constraints to prevent the need to look up the original instance.

As the left figure shows, s_{EST} is not resource feasible. By scheduling a_1 and a_3 together at time 0 the resource capacity of r_1 is exceeded by 1. Thus, in any feasible schedule tasks a_1 and a_3 must not overlap. Starting a_3 one time unit later alongside a_2 is the most efficient solution, giving the optimal makespan of 3.

2.1.3 Classes of solutions

The optimal makespan solution s_{OPT} presented in figure 2.2 is not the only optimal solution for the example of figure 2.1. After all, the start of activity a_2 could be delayed by one time-unit without consequence. Both schedules are feasible and optimal, but they belong to different classes of solutions according to Sprecher, Kolisch and Drexl [28], who propose a classification of solutions based on the classification from the Job Shop Scheduling problem. They identify a hierarchy of solutions:

Non-Delay Schedules $\subseteq$ Active Schedules $\subseteq$ Semi-Active Schedules $\subseteq$ Feasible Schedules

Using this classification scheme, the solution s_{OPT} from figure 2.2 is a Non-Delay Schedule, while delaying activity a_2 by one results in a Feasible Schedule. For our purposes, the

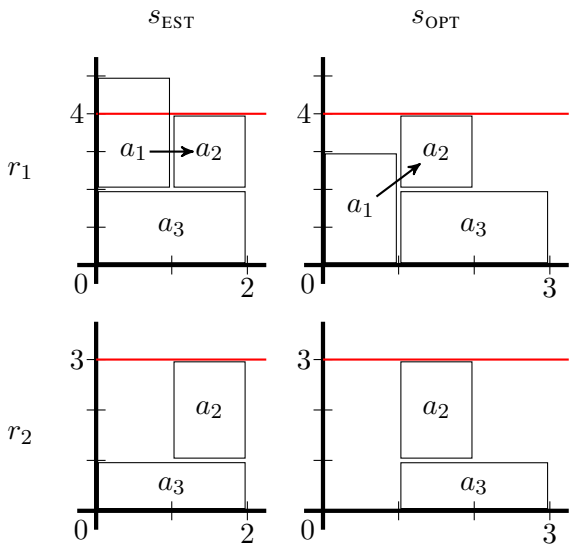


Figure 2.2: Gantt chart representations of the earliest start schedule (on the left) and an optimal solution (on the right) to the example instance from figure 2.1.

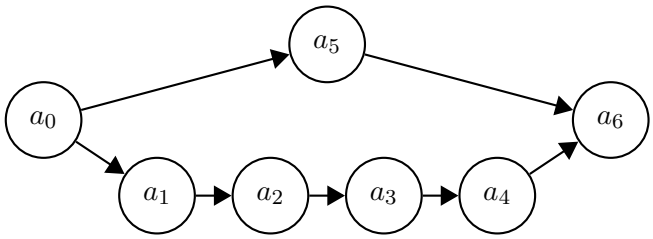
classes of Non-Delay and Active Schedules are interesting to investigate, since solutions produced by Schedule Generation Scheme heuristics are always at least Active. Further, we know that the set of Active Schedules always contains an optimal solution, while the set of Non-Delay Schedules is not guaranteed to contain an optimal solution.

Active Schedules

A schedule is called Active if none of the activities can be (resource and precedence) feasibly started at any earlier point in time. At least one makespan optimal schedule is Active. To prove this, assume that for an instance no Active schedule is optimal. Then in all optimal schedules, at least one of the activities can be feasibly started earlier. Since it is feasible to do so, we may consider the schedule where one of these activities is indeed started as early as feasible. Starting an activity earlier can only lower the makespan since $f_{\text{makespan}} = \max_{a_i} (s(a_i) + d(a_i))$. But because the schedule is optimal, the makespan is not affected. As long as the schedule is not Active we can keep feasibly starting activities earlier, until we end up with a schedule in which no activity can be feasibly started earlier. Because makespan cannot increase by feasibly starting activities earlier, the resulting Active schedule is also optimal, contradicting our assumption.

Non-Delay Schedules

Non-Delay Schedules are schedules in which not even a part of the duration of any activity can feasibly be started earlier. In the RCPSP activities cannot be interrupted or preempted, which means that once an activity starts it must remain in progress for its entire duration.



	d \ c	r_1
		2
a_1	1	1
a_2	1	2
a_3	3	1
a_4	3	1
a_5	6	1

Figure 2.3: Example RCPSP instance of 5 activities, 1 resource that does not allow an optimal Non-Delay solution.

If we relax this constraint on a schedule and still no parts of activities can start earlier it is classified as a Non-Delay schedule.

To illustrate this concept further and to prove that sometimes no makespan optimal schedule is a Non-Delay schedule, consider the instance in figure 2.3. The Gantt charts for the Earliest Start schedule and the unique optimal solution are presented in figure 2.4. Activity a_2 and a_5 are the only activities that are in conflict in the s_{EST}, resulting in overuse at time point two. Starting a_5 before a_2 is highly inefficient, resulting in an $f_{\text{makespan}} = 13$ schedule where the full resource capacity is used for only two time points. Thus, in the optimal schedule a_5 is started after a_2.

However this optimal solution is not a Non-Delay schedule. If we were allowed to pre-empt activity a_5, it could be started alongside a_1 as figure 2.4 shows in the bottom right Gantt chart. Since the optimal solution is unique and not a Non-Delay schedule, this instance does not allow an optimal Non-Delay schedule. Research has shown that instances that do not allow an optimal Non-Delay schedule are not rare. Kolisch enumerated all Non-Delay schedules on 298 instances of 30 activities and found that $41,27\%$ did not allow an optimal Non-Delay schedule [17].

An interesting observation is that if the instance is trivial, the optimal solution is a Non-Delay schedule, since no activity in s_{EST} can be started any earlier without violating the precedence constraints.

2.2 Algorithms

This section provides the heuristic algorithms we will use to find solutions to instances of the RCPSP defined in the previous section. The Serial and Parallel Schedule Generation Schemes are greedy algorithms that use a Priority Rule heuristic to determine which activity to schedule next. First the concept of a Priority Rule is treated in section 2.2.1, followed by the two greedy algorithms, Serial in section 2.2.2 and Parallel in section 2.2.3. Finally, section 2.2.4 explains the concept of reversing an instance of the RCPSP and solving it

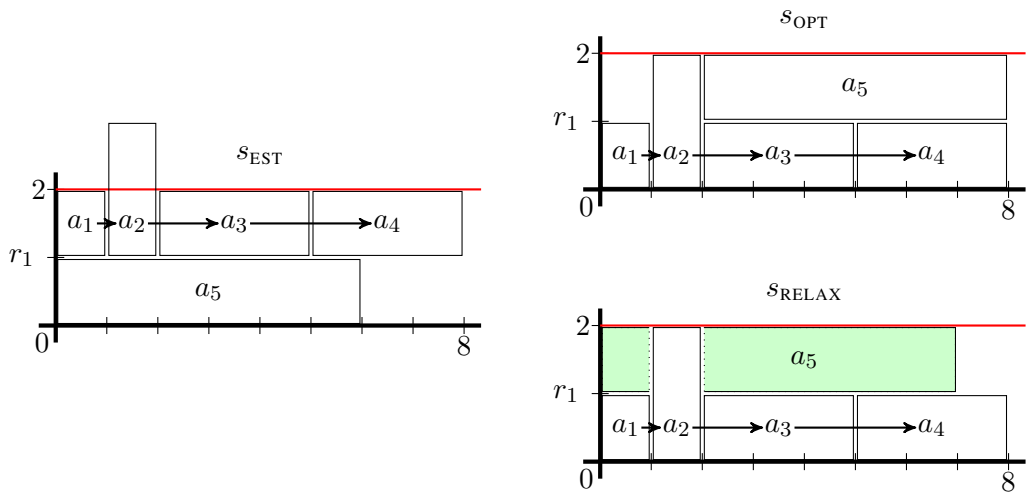


Figure 2.4: Gantt chart representations of the earliest start schedule (on the left) and the optimal solution (top right) to the example instance from figure 2.3. The bottom right Gantt chart shows an optimal solution if the no pre-emption constraint is relaxed.

backwards. This leads to the definition of the Forward-Backward Improvement algorithm in section 2.2.5.

2.2.1 Priority Rules

A Priority Rule heuristic is a function h that intends to select the best activity to schedule next from a set of eligible activities $D \subseteq A$, taking into account the properties of the instance I and the current schedule s. Priority rules that are invariant to the current schedule s are called *static* priority rules, since they can be precomputed from the instance I. The simplest static priority rules can be expressed as a *list* ordered by some property of the instance I, where the function h consists of finding the element of D that occurs earliest in the list. In this thesis we only consider such static list-based priority rules, since they work well in practice and they match what the FBI algorithm does. The following sections give the priority rules we will use in this thesis.

Random Priority Rule

The most simple 'heuristic' is to simply assign each activity a random position in the list. This means that each activity in D is equally likely to be selected upon invocation of the priority rule, resulting in a random schedule. Random sampling is sometimes used as worst-case baseline with which to compare more informed heuristics.

Latest Finish Time

The priority rule that has been identified in literature to result in the best schedules is the Latest Finish Time (LFT) heuristic. Just like the Earliest Start Time schedule s_{EST}, we may

analogously compute a Latest Start Time schedule s_{LST} by starting from the finish time of a_{n+1} in s_{EST} and recursively starting each preceding task as late as possible relative to a_{n+1} (not considering the resource constraints). The LFT heuristic selects the activity with the $\min_{a_i \in D} s_{\text{LST}}(a_i) + d(a_i)$. This task may intuitively be seen as the most urgent task, since it has the largest duration of successors.

Total Successors

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

Randomized Rules

Priority rule heuristics frequently make use of randomization. There are two reasons for this: In the first place, sometimes two or more activities may have equal priority. Then randomization is used to break ties. In the second place, using a deterministic priority rule results in undesirable worst-case behavior where pathological inputs can result in a priority rule selecting the a bad activity to schedule at each invocation. Randomization can be used to avoid this worst case behavior by ensuring there is at least a chance to select a better activity.

The randomization method that results in the best schedules is called Regret Based Biased Random Sampling [17]. This method computes for each activity $a_i \in D$ a regret value ρ_i that measures the worst case consequence that could occur from selecting a different activity $a_j \in D$. Suppose that activities are sorted on increasing values of priority function $p : A \to \mathbb{R}$. Then the regret of activity a_i is $\rho_i = \max_{a_j \in D} p(a_j) - p(a_i)$. These regrets are then used to compute a probability ψ_i of selecting activity a_i using:

$$\psi_i = \frac{(\rho_i + 1)^\alpha}{\sum_{a_j \in D} (\rho_j + 1)^\alpha}$$

The parameter α determines the level of bias. An α of 0 gives each activity equal priority, while a large α results in deterministic choices with random tie breaking.

2.2.2 Serial Schedule Generation Scheme

The Serial Schedule Generation Scheme (Serial SGS) is a single pass greedy algorithm for constructing a schedule from an RCPSP instance. The algorithm incrementally schedules a single activity at a time, at the earliest possible resource feasible time, without backtracking on the start times of already scheduled activities.

Algorithm 1 gives the pseudocode for the Serial SGS. It uses two subroutines, algorithms 3 and 2. The first subroutine ISFEASIBLE simply checks the feasibility conditions

Algorithm 1 The Serial Schedule Generation Scheme finds a solution s for instance I using priority rule heuristic h.

1: **function** SERIAL($I = (A, R, P, d, u, c), h$)
2: $s(a_i) \leftarrow$ NULL $\quad \forall a_i \in A$
3: $D \leftarrow$ DETERMINEELIGIBLE(s, I)
4: **while** $D \neq \{\}$ **do**
5: $a_i \leftarrow h(D, I, s)$
6: $t \leftarrow 0$
7: **while** ISFEASIBLE(a_i, t, s, I) $\neq$ TRUE **do** $t \leftarrow t + 1$
8: $s(a_i) \leftarrow t$
9: $D \leftarrow$ DETERMINEELIGIBLE(s, I)
10: **return** s

Algorithm 2 Algorithm for determining the eligibility of activities to be scheduled based on partial schedule s for instance I.

1: **function** DETERMINEELIGIBLE($s, I = (A, R, P, d, u, c)$)
2: $D \leftarrow \{\}$
3: **for all** $a_i \in A$ **do**
4: **if** $s(a_i) =$ NULL **then**
5: ELIGIBLE = TRUE
6: **for all** $(a_j \prec a_i) \in P$ **do**
7: **if** $s(a_j) =$ NULL **then**
8: ELIGIBLE = FALSE
9: **if** ELIGIBLE = TRUE **then**
10: $D \leftarrow D \cup a_i$
11: **return** D

Algorithm 3 Algorithm for determining the feasibility of starting activity a_i at time t_i given partial schedule s for instance I.

1: **function** ISFEASIBLE($a_i, t_i, s, I = (A, R, P, d, u, c)$)
2: **for all** $(a_j \prec a_i) \in P$ **do**
3: **if** $s(a_j) =$ NULL $\vee$ $s(a_j) + d(a_j) > t_i$ **then return** FALSE
4: **for** $t_i \leq t < t_i + d(a_i)$ **do**
5: **if** $\left(\sum_{a_j \in i(s,t)} u(a_j, r_k) \right) + u(a_i, r_k) > c(r_k)$ **then return** FALSE
6: **return** TRUE

2.1 and 2.2 for a single activity and its start time against a partial schedule. When ISFEASIBLE returns true, activity a_i can resource and precedence feasibly be started at time t_i.

Which activity is to be scheduled next by the Serial SGS is determined by two aspects: The activity must be eligible to schedule, which means that all its predecessors are already scheduled, and it must have the highest priority among the eligible activities. This activity is determined by invoking the provided priority rule heuristic h on the set of eligible activities D. The subroutine DETERMINEELIGIBLE determines set D by looking at which as of yet unscheduled activities have all of their predecessors scheduled.

The Serial SGS operates in time $O(n^2m)$ where n is the number of activities and m is the number of resources because the only time points t that are important to test are those where an activity starts or stops. Thus each iteration schedules an activity by scanning at most $2n$ possible start times. The schedules produced by Serial SGS are always at least Active schedules, which thus guarantees that there exists a heuristic h that produces an optimal schedule.

2.2.3 Parallel Schedule Generation Scheme

The Parallel SGS differs from the Serial SGS in the way it schedules activities at each incremental step. 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 (without backtracking to earlier time points or rescheduling activities). Which activities it tries to schedule first is again determined by the priority rule heuristic.

Algorithm 4 The Parallel Schedule Generation Scheme finds a solution s for instance I using priority rule heuristic h.

```
1:  function PARALLEL(I = (A, R, P, d, u, c), h)
2:      s(a_i) ← NULL   ∀a_i ∈ A
3:      t ← 0
4:      while ∃a_i  s(a_i) = NULL do
5:          D ← PARDETERMINEELIGIBLE(t, s, I)
6:          while D ≠ {} do
7:              a_i ← h(D, I, s)
8:              D ← D \ a_i
9:              if ISFEASIBLE(a_i, t, s, I) = TRUE then
10:                 s(a_i) ← t
11:         t ← t + 1
12:     return s
```

The pseudocode for the Parallel SGS is given in algorithm 4. It also uses the feasibility test from algorithm 3, however it uses a different method for determining the set of eligible activities D, found in algorithm 5. Because the Parallel SGS advances time the set D must be valid for the current time t, which means we must account for activities in progress at t in the partial schedule s.

Algorithm 5 Algorithm for determining the eligibility of activities to be scheduled at time t based on partial schedule s for instance I.

```
1:  function ParDetermineEligible(t, s, I = (A, R, P, d, u, c))
2:      D ← {}
3:      for all a_i ∈ A do
4:          if s(a_i) = NULL then
5:              ELIGIBLE = TRUE
6:              for all (a_j ≺ a_i) ∈ P do
7:                  if s(a_j) = NULL ∨ s(a_j) + d(a_j) > t then
8:                      ELIGIBLE = FALSE
9:              if ELIGIBLE = TRUE then
10:                 D ← D ∪ a_i
11:     return D
```

The Parallel SGS also operates in time $O(n^2m)$, again because the only interesting time points to consider are the start and finish times of tasks. The schedules produced by Parallel SGS are always Non-Delay schedules, which does not always contain the optimal schedule. The trade-off is that the non-delay search space is smaller, and in practice the Parallel SGS produces better schedules on average unless the number of iterations is high or the instance is easy [17].

2.2.4 Backward Scheduling

Sometimes, the SGS methods outlined in the previous sections produce better schedules when the schedule is solved *backwards*. An instance I of the RCPSP can be transformed into a backwards instance I' by reversing all precedence constraints, thus:

$$I = (A, R, P, d, u, c),\ I' = (A, R, P', d, u, c),\ P' = \{\, a_j \prec a_i \mid \forall a_i \prec a_j \in P\}$$

Solutions to the backwards instance must be transformed again to be valid for the original instance. Suppose that solution s' was found for I', then s is constructed for I as follows:

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

The example instance in figure 2.3 is presented in both the forward and the reversed direction in figure 2.5. Suppose the parallel method is applied to the forward schedule. In the first step of the algorithm $D = \{a_1, a_5\}$, and they can both be resource feasibly started at time 0. The consequence is that in any schedule produced by the parallel method $s(a_1) = s(a_5) = 0$ and a_2 can start only after a_5 finishes. Thus, the parallel method can never produce the optimal schedule on the forward instance. By contrast, in step one on the backward instance $D = \{a_4, a_5\}$ and both are resource feasible. Thus, the optimal schedule always follows.

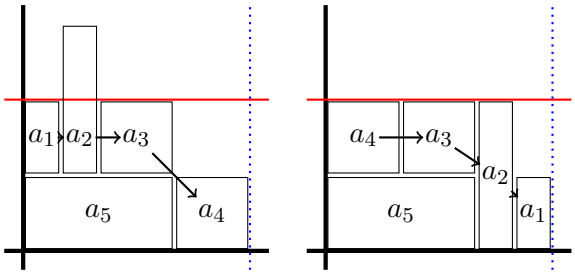


Figure 2.5: Gantt charts of s_{EST} schedule of instance 2.3 in the forward direction (left) and the backward direction (right).

2.2.5 Forward-Backward Improvement (FBI)

As explained in section 1.2.2, many heuristics use the Forward-Backward Improvement algorithm as a post-processing step to improve schedules obtained through global search. FBI is inspired by the concept of free slack [29]: In any solution produced by the Serial or Parallel SGS, activities are started as soon as possible (since they produce at least Active solutions). This implies that some activities might as well have started later without it adversely affecting the makespan. The amount of space a task has to 'start later' is called its slack. Tasks that have a large amount of slack are consuming resources at an early time in the schedule, even though they do not really need to yet. By shifting such a task forward in its slack resources are freed up, hopefully allowing other activities to start earlier, reducing the makespan of the total schedule.

The concept of backward scheduling explained in the previous section is the basic building block of the Forward-Backward Improvement (FBI) algorithm. By scheduling an instance backwards using an SGS activities are effectively started as 'late' as possible when viewed in the forward dimension. Thus scheduling a *solution* backwards should push all activities to start as 'late' as possible in their slack windows. Then rescheduling this second, backwards, solution in the forward direction resources that were freed up early can be used by other activities.

To demonstrate this concept in action, consider the example instance found in figure 2.6. Suppose that we are given a solution s_{SGS} to this instance, presented in Gantt chart format in figure 2.7. In this solution the activities a_6, a_7 and a_8 are the only activities that have some slack. All of them could have been started 3 time units later. Because a_6 starts as soon as possible, it prevents activity a_9 from running parallel with a_2 resulting in an inefficient schedule. Figure 2.7 also presents the solution s_{BWD} which is s_{SGS} scheduled backwards. In the backwards solution there is sufficient room to schedule a_9 on top of a_2 because the a_6, a_7 and a_8 chain was pushed to the right. Rescheduling s_{BWD} forwards results in solution s_{FWD}, which is 2 time units shorter than the original by exploiting the room created early in the schedule.

Algorithm 6 called JUSTIFY turns s_{SGS} into s_{BWD} and s_{BWD} into s_{FWD}. It first flips the instance backwards by reversing each precedence constraint in the network. Next the Serial SGS is applied with the justification heuristic j_s. This heuristic simply uses the fin-

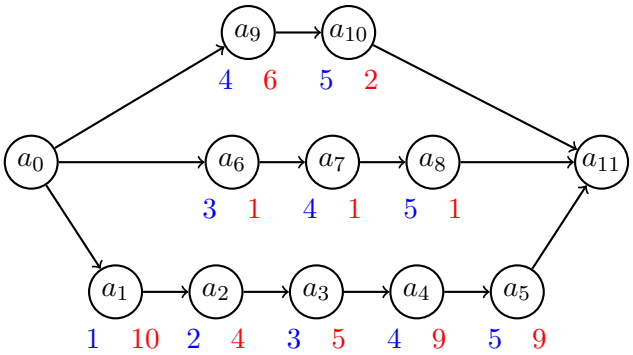


Figure 2.6: Example project network of ten tasks with equal duration and a single resource of capacity 10 (resource usage in red, latest finish time in blue).

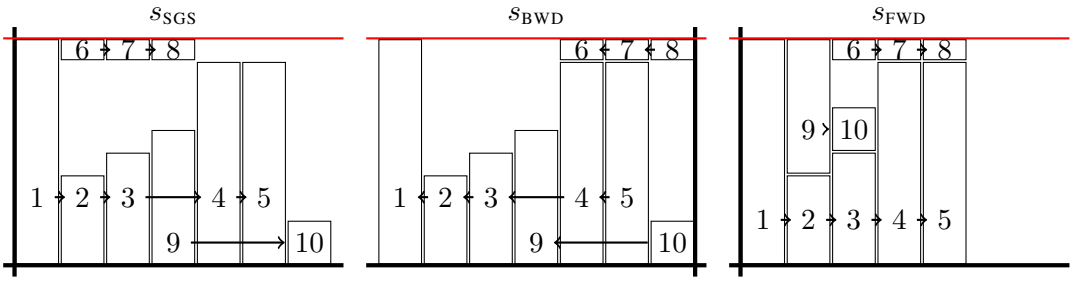


Figure 2.7: Left: Possible minimum LFT solution to 2.6. Middle: After the backward pass of FBI. Right: After the forward pass of FBI.

ish times of activities in the input schedule s as priorities, prioritizing the activity with $\max_{a_i \in D} s(a_i) + d(a_i)$. Then FBI (Algorithm 7) simply chains two justification operations together, the first applied to the input schedule and the second applied to the justified schedule. In a non-pseudocode implementation we would only reverse the schedule precedence constraints once, however because we will later consider the Justify algorithm (6) in isolation, we leave the schedule reversing steps in its pseudocode for correctness.

Algorithm 6 Algorithm for justifying a solution s to instance I into its reversed version.

1: **function** JUSTIFY($s, I = (A, R, P, d, u, c)$)
2: $P' \leftarrow \{\}$
3: **for all** $(a_i \prec a_j) \in P$ **do**
4: $P' = P' \cup (a_j \prec a_i)$
5: $I' = (A, R, P', d, u, c)$
6: **return** SERIAL(I', j_s)

Algorithm 7 The FBI algorithm for improving solution s to instance I.

```
1: function FBI(s, I = (A, R, P, d, u, c))
2:     P' ← {}
3:     for all (a_i ≺ a_j) ∈ P do
4:         P' = P' ∪ (a_j ≺ a_i)
5:     I' = (A, R, P', d, u, c)
6:     s_BWD = JUSTIFY(s, I)
7:     s_FWD = JUSTIFY(s_BWD, I')
8:     return s_FWD
```

2.3 RCPSP Instance Measures

Many measures have been proposed to predict how hard a RCPSP instance will be to solve optimally. Clearly increasing the size of the instance, the number of activities n or the number of resources m, will make the instance more difficult. However research on other NP problems has shown that given a fixed size, a secondary parameter often sharply determines how difficult the instance actually is. Randomly generated instances with parameter levels near the critical value exhibit exponential scaling of difficulty with increasing size, while those generated with values far from the critical value may exhibit only polynomial scaling. This does not mean that all instances with the critical parameter value are difficult, since fixing the secondary parameter may expose a third factor of difficulty [13].

Knowing which parameter(s) influences difficulty is crucial to be able to test algorithm performance: It is expected that an algorithm that performs well on the most difficult instances will perform well on average over all types of instances. Therefore the study of instance metrics is closely related to the study of benchmark sets and instance generators. Kolisch, Sprecher and Drexl [21] demonstrated the need for such research by showing that all earlier benchmark instances are easy to solve while simultaneously showing that even small instances ($n = 30$) produced by their generator can be very difficult to solve. Their benchmark sets and thus parameter choices are currently the de facto standard for testing algorithm performance.

This section aims to present the most successful measures of RCPSP instances, and how they have been shown to affect algorithm performance.

2.3.1 ProGen Instance Measures

The benchmark sets generated by the ProGen generator and presented in [21] are produced by controlling for the metrics (Network) Complexity, Resource Factor and Resource Strength.

Network Complexity (NC)

The Network Complexity of an instance is a measure of the amount of constrainedness originating from the precedence constraint set P. It measures the average number of non-

redundant precedence constraints per activity, *including* the special source and sink activities. A precedence constraint is redundant if it is implied by transitivity: $a_i \prec a_j$ is redundant if there exists a longer path from a_i to a_j: $a_i \prec a_x, ..., a_y \prec a_j$. Thus, Network Complexity is defined as:

$$\text{NC} = \frac{\text{\# Non-Redundant Precedences}}{|A| + 2}$$

Consider the first example instance found in figure 2.1. This instance contains 7 precedence constraints, of which 5 are non-redundant. The instance consists of 3 regular activities. Thus for that instance, NC $= \frac{5}{3+2} = 1$.

Network Complexity is not normalized in the study of Kolisch, Sprecher and Drexl [21] because the value would otherwise tend towards zero for realistic precedence networks when the number of activities increases [20]. They demonstrated that the computational effort needed to solve an instance decreases with an increase in Network Complexity.

De Reyck and Herroelen instead demonstrated that Network Complexity is strongly correlated with another metric called the Complexity Index [5]. They showed that when Complexity Index is held constant Network Complexity has no significant effect on instance hardness, while varying the Complexity Index for constant Network Complexity has a significant effect on instance hardness.

Resource Factor (RF)

Resource Factor measures how many different resources are used by the average activity. An activity uses a resource when $u(a_i, r_k) > 0$, which suggests an intuitive view of the metric is to measure it as the density of the matrix described by u. Formally Resource Factor is defined as:

$$\text{RF} = \frac{1}{nm} \sum_{i=1}^{n} \sum_{k=1}^{m} \min\left(1, u(a_i, r_k)\right)$$

Therefore an instance with a Resource Factor of 0 is trivial, since the early start solution is feasible (no resources are requested). On the other hand, if the Resource Factor is 1 all activities request all resource types resulting in maximum potential resource conflicts. Consider again the first example (figure 2.1). In this instance the Resource Factor obtains the value RF $= \frac{5}{6}$ because activity a_1 does not request resource r_2.

Resource Factor has been shown to have a strong impact in instance difficulty by Kolisch, Sprecher and Drexl [21], and this is a generally accepted result in literature.

Resource Strength (RS)

The Resource Factor alone is not sufficient to measure instance hardness due to resource constraints. Intuitively, if $c(r_k) > n$ assigning each activity a usage $u(a_i, r_k) > 0, u(a_i, r_k) \approx 1$ does not constitute a hard instance even though RF attains the maximum value.

The metric Resource Strength is intended to measure the size of resource conflicts. It computes the ratio of the resource capacity to the highest peak resource usage on a single

resource in the Earliest Start Time schedule. Consider the value u_{PEAK} defined as the peak resource usage in s_{EST} and u_{MAX} the maximum resource usage by a single activity. Then Resource Strength is computed as follows:

$$u_{\text{PEAK}}(r_k) = \max_t \Big(\sum_{a_i \in i(s_{\text{EST}}, t)} u(a_i, r_k) \Big)$$
$$u_{\text{MAX}}(r_k) = \max_{a_i} \big(u(a_i, r_k) \big)$$
$$\text{RS}_k = \frac{c(r_k) - u_{\text{MAX}}(r_k)}{u_{\text{PEAK}}(r_k) - u_{\text{MAX}}(r_k)}$$

Applied to the example instance of figure 2.1, the resource strengths are:

$$\text{RS}_1 = \frac{4-3}{5-3} = \frac{1}{2}$$
$$\text{RS}_2 = \frac{3-2}{3-2} = 1$$

To compute the Resource Strength of an instance the values per resource are averaged, thus the example has a Resource Strength of $\frac{3}{4}$. Usable values for Resource Strength range from 0 to 1, which defines a scale ranging from highly constrained to unconstrained. If the Resource Strength is 0 at least one of the activities uses all of a resource. A Resource Strength of 1 on the other hand, means that the peak usage is equal to the capacity and thus that the instance is trivial.

The measure of Resource Strength was shown to have the strongest impact on instance hardness in the evaluation of Kolisch, Sprecher and Drexl [21]. This result was confirmed by De Reyck and Herroelen [5], who further demonstrated an easy-hard-easy pattern as Resource Strength increases.

Nevertheless, Resource Strength is sometimes considered a flawed measure. The value returned by Resource Strength is dependant on the largest peak, which means that the size and frequency of smaller peaks are completely discarded. Especially as the number of activities increases, it may eventually fail to provide enough precision. At the same time, there is a weakness in precision when there is an activity with $u(a_i, r_k) = c(r_k)$. Regardless of the size and frequency of peaks, the Resource Strength will always be reported as zero, even if the instance is (almost) trivial. Finally, erroneous predictions of instance difficulty can occur when the Resource Strength has large variance over the resources.

Furthermore, a criticism of the Resource Strength measure is that it is not a pure resource measure. By computing the resource profile of the s_{EST} schedule, it implicitly includes precedence constraints in its computation. A consequence of this is that the Resource Strength of an instance depends on its schedule direction.

2.3.2 Alternative Measures

The measures of Network Complexity and Resource Strength presented in the previous section have received criticism, and some authors have proposed different measures for these concepts. This section presents the measures of Order Strength and Complexity Index

which are alternatives for Network Complexity and Resource Complexity as an alternative to Resource Strength.

Complexity Index

The Complexity Index is a precedence network measure that was shown to be a more explanatory measure of instance difficulty than Network Complexity in [5]. The measure itself is defined as the minimum number of 'node reductions' required to turn the Activity-on-the-Arc network of an RCPSP instance into a single arc. Although the measure is more predictive than Network Complexity, we will not elaborate on CI in this thesis due to its undesirable properties: It requires use of the impractical AoA representation, and it is not possible to generate instances by controlling CI [5].

Order Strength

The Order Strength of an instance is a measure of the precedence constraint network constrainedness. It computes the ratio of the number of redundant and non-redundant precedence constraints to the maximum. Given the set P we can compute the transitive closure $\overline{P}$ containing all the constraints P and all implied redundant constraints. Then the Order Strength is computed as:

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

Unlike Network Complexity, Order Strength is a normalized measure. It also ignores the precedence constraints imposed by the dummy activities. It is easier to compute and more practical to reason about than the Complexity Index, while having more explanatory power than the Network Complexity.

Resource Constrainedness

The Resource Constrainedness is an alternative to Resource Strength that 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 Resource Constrainedness is defined as the average resource usage of a resource k over those activities that use k.

$$\text{RC}_k = \frac{\frac{\sum_{a_i \in A} u(a_i, r_k)}{\sum_{a_i \in A} \max(1, u(a_i, r_k))}}{c(r_k)}$$

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

Chapter 3

Measuring and Controlling Instance Difficulty

The previous chapter presented definitions of several different RCPSP instance measures in section 2.3, along with their interaction with instance difficulty as reported in literature. This chapter aims to use this knowledge to answer the research question 5 by combining it into a model of the interactions in section 3.1.

This proposed model raises a number of questions that are treated in section 3.1.1, prompting us to propose a novel model of the instance difficulty in section 3.2. Finally, section 3.3 presents how the difficulty of an instance can be measured using heuristic solvers.

3.1 Model of Instance Difficulty

In their paper on phase transitions in project scheduling, Herroelen and De Reyck present evidence of a difficulty phase transition in Resource Strength and Resource Constrainedness [12]. They conjecture that the two measures are correlated such that low Resource Strength roughly corresponds to high Resource Constrainedness. Further, they demonstrate a linear hard-easy trend for increasing Order Strength. This model is presented in figure 3.1.

In this model the effects of the precedence constraints and resource constraints are effectively separated, with the resource constraints having the most influence on instance difficulty. This matches the observations by Kolisch, Sprecher and Drexl in their paper on the generation of difficult instances [21]. They find that Network Complexity has only marginally significant impact on instance hardness, while the effect of Resource Strength is highly significant.

Further, the model considers the measures of Resource Strength and Resource Constrainedness to be essentially the same, except that Resource Constrainedness has more precision since it will keep increasing even after an activity takes resource usage equal to the capacity (resulting in a Resource Strength of 0).

According to this model, the answer to research question 5 is that Resource Constrainedness or Resource Strength should be used as order parameter. Thus, the FBI algorithm

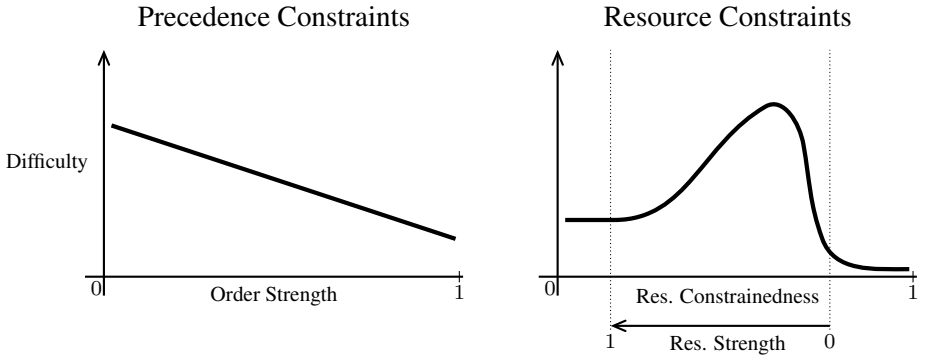


Figure 3.1: Interaction between order parameters and instance difficulty.

should be tested on instances with a low Order Strength and varying Resource Constrainedness or Strength.

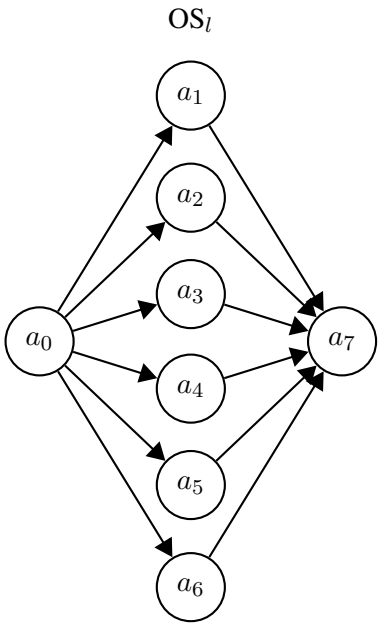
3.1.1 Questions on the Model of Instance Difficulty

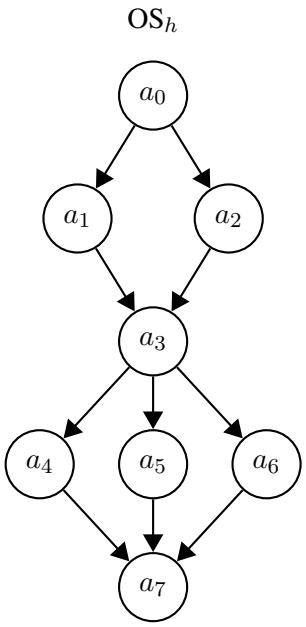
The model presented in the previous section raises a number of questions on the interactions between precedence and resource constraints. In the first place, how can Resource Strength be considered as effectively decoupled from Order Strength when Resource Strength implicitly uses the precedence constraints to compute the Earliest Start schedule s_{EST}? More fundamentally, if it is true that difficulty is linearly decreasing with Order Strength this implies that the problem would be harder without precedence constraints. Why should we not just study a planning problem without precedence constraints?

To emphasize the connection between Resource Strength and Order Strength, consider the instances in figure 3.2. Four different instances can be built up from a selection of low or high Order Strength and low or high Resource Constrainedness. The values for Order Strength are $\text{OS}_l = 0$ and $\text{OS}_h = \frac{11}{15}$, while the Resource Constrainedness options are $\text{RC}_l = \frac{1}{3}$ and $\text{RC}_h = \frac{16}{27}$. The Resource Strengths for these four instances are:

$$\begin{aligned} RS(OS_l, RC_l) &= \frac{9-3}{18-3} = \frac{2}{5} \\ RS(OS_h, RC_l) &= \frac{9-3}{9-3} = 1 \\ RS(OS_l, RC_h) &= \frac{9-8}{32-8} = \frac{1}{24} \\ RS(OS_h, RC_h) &= \frac{9-8}{12-8} = \frac{1}{4} \end{aligned}$$

Based on these values for Resource Strength, we see that Resource Strength indeed decreases as the Resource Constrainedness increases. However, we also see that the Resource Strength increases as the Order Strength increases. This suggests that there is indeed an interaction between Order Strength and Resource Strength.





		RC$_l$	RC$_h$
	d \ c	9	9
a_1	1	3	6
a_2	1	3	6
a_3	1	3	8
a_4	1	3	4
a_5	1	3	4
a_6	1	3	4

Figure 3.2: Four different RCPSP instances are obtained by selecting an Order Strength and Resource Constrainedness level, for example the instance (OS$_l$, RC$_h$) has low Order Strength and high Resource Constrainedness.

3.2 New Model of Instance Difficulty

The questions about the current model of instance difficulty prompt us to propose a different model of instance difficulty:

Hypothesis 1. The difficulty of RCPSP instances attains a peak when the precedence constraints and resource constraints are in balance.

Hypothesis 1 states in other words that as the degree of precedence constrainedness increases, the degree of resource constrainedness should increase to keep instance difficulty high. Instead of a single peak of difficulty, the hypothesis predicts a 'ridge' of difficulty in two dimensional space defined by the resource and precedence constrainedness. If we could perfectly follow this ridge, we might find a setting of resource and precedence constrainedness that is 'hardest' for its size but we will never find it by fixing just one of the two types of constraints.

Whereas Order Strength purely measures the precedence aspect and Resource Constrainedness only measures the resource aspect, the measure of Resource Strength appears to be inspired by hypothesis 1 as it is a hybrid of precedence and resource constraints. Perhaps keeping Resource Strength fixed is an adequate way to follow the difficulty ridge. If this is the case, we would expect that Resource Strength, Order Strength and Resource Constrainedness are pairwise correlated:

Expectation 1. If Order Strength is held constant, Resource Strength is decreasing for increasing Resource Constrainedness.

Expectation 2. If Resource Strength is held constant, Resource Constrainedness is increasing for increasing Order Strength.

Expectation 3. If Resource Constrainedness is held constant, Resource Strength is increasing for increasing Order Strength.

The expectation 1 is what is expressed in the model explained in section 3.1. However, the other expectations are not consistent with the existing model.

Further, if the hypothesis is true, it should be the case that instances with fixed Resource Strength are on average more difficult than instances with fixed Resource Constrainedness, since only a limited range of Order Strength will meet the peak difficulty.

Expectation 4. Holding Resource Strength constant results in more difficult instances than holding Resource Constrainedness constant.

If hypothesis 1 is true, the answer to research question 5 is that we should keep Resource Strength fixed and use Order Strength (or Resource Constrainedness) as order parameter.

3.3 Measuring Heuristic Performance

The Schedule Generation Schemes (SGS) and by extension the Forward-Backward Improvement (FBI) are single pass heuristics that take polynomial time in the order of the instance size to compute a schedule. As such, when discussing the performance of these methods, we are not interested in the CPU time or computational complexity. Kolisch already remarked in his investigation into the serial and parallel generation schemes in [17] that ”the computational effort is not influenced by any of the problem parameters.” Reporting on the CPU time would thus only serve to give an impression of the speed of the machine and efficiency of the implementation, as opposed to provide new insights into the actual qualitative performance of the algorithms.

Instead, we are interested in the quality of the solutions produced. The quality of a heuristic solution is determined by the makespan of the solution relative to the optimal makespan: The closer the schedule is to the optimal, the more useful it is. If we want to know how difficult instances of a given setting of order parameter are, we should compute the mean quality over a large number of instances with that parameter setting. By studying the mean performance, we obtain an estimate of the expected performance for that parameter setting.

In practice, it is not feasible to compute the optimal solution of a large number of instances for each parameter setting. Consider that for several of the 360 nontrivial instances of the test set of 60 activities put forward by Kolisch et al. in [21] almost 20 years ago, no optimal solution has been found. Thus for practical purposes we should use a bound on the optimal makespan.

Using a lower bound ensures that we will not overestimate the performance of the heuristics, however lower bounds have their own disadvantages. The most serious disadvantage is that they are likely biased by the parameters we intend to control. Consider the resource relaxation based lower bound, obtained by computing the makespan of s_{EST}. This bound becomes more accurate the higher Order Strength becomes: As more precedences are introduced, the number of activities in the critical chain increases increasing the makespan of s_{EST}. At the same time, fewer activities are in conflict with each other, reducing the chance of resource constraints to increase the optimal makespan above the critical chain.

Due to these issues with lower bounds, we opt to use an upper bound on the optimal makespan instead. We compare each computed schedule to the best schedule we have ever found on a given instance. This is an underestimate of the actual deviation, however in this thesis we are not interested in the absolute performance of heuristics: We only want to investigate how well FBI performs relative to the base heuristics, and how much of that improvement can be attributed to each pass.

Chapter 4

Experimental Evaluation

The previous chapter proposed a new hypothesis on the interaction of the RCPSP order parameters in the process of answering research question 5. This chapter will focus on testing the expectations from chapter 3 and, using the results of these tests, explore the performance of the FBI algorithm along the order parameters in order to answer research questions 3 and 4.

First we explain the need for our own test set in section 4.1. Then, section 4.2 tests the expectations about the interaction of the RCPSP order parameters on the new test set. Finally, section 4.3 explores the performance of FBI along the identified order parameters.

4.1 Test Set

This section describes the test set used in all subsequent experiments. Despite the existence of many test sets for the RCPSP, we opted to generate our own test set. First, we will look at the history and design of the most frequently used benchmark set, PSPLIB. Next we explain why these test sets are not sufficient for our goals, and finally we show our test set design, and how it compares to the PSPLIB instances.

4.1.1 Limits of the PSPLIB

Researchers who work on the RCPSP are very familiar with the PSPLIB, four benchmark test sets described by Kolisch, Schwindt and Sprecher in 1998 [19]. The design of these test sets originates from 1995 when Kolisch, Sprecher and Drexl [21] investigated the hypothesis that small RCPSP instances cannot be hard.

To this end they first identified three potential measures of instance hardness: Complexity, Resource Factor and Resource Strength. They subsequently constructed an instance generator that allowed to vary these measures. By applying a full factorial design they showed that instances with a high Resource Strength or low Resource Factor are trivial to solve. Further, they have shown that higher Complexity instances are somewhat easier to solve. Finally, they showed that small instances (of only 30 activities) can indeed be intractable to solve when the three factors are set to their hardest.

The public PSPLIB benchmark sets use the same full factorial design as was applied to determine the hardness of instances, however now applied to four different instance sizes (of 30, 60, 90 and 120 activities). The instances of 120 activities have different factors for Resource Strength ($(0.1, 0.2, 0.3, 0.4, 0.5)$ as opposed to $(0.2, 0.5, 0.7, 1)$ for the smaller sets, and consequently the set contains more, harder instances than the other sizes.

Due to the original purpose of Kolisch, Sprecher and Drexl to show the varying difficulty of RCPSP instances the PSPLIB contains a high variety of instances. In fact, fully one quarter of the smaller sets is trivial to solve (the instances with Resource Strength of 1). Further, due to their focus on Resource measures to describe the hardness of instances, the set contains just 30 instances per level of resource difficulty.

Since we hypothesize that precedences play just as vital a role as resources, we conclude the PSPLIB does not offer enough different levels of precedence constrainedness to use. Furthermore, the set does not aim to keep the resource constraints in balance with the precedence constraints, making it impossible to test hypothesis 1 on this set.

4.1.2 Generation of the test set

To circumvent the limits of the PSPLIB we generated our own test set. For the generation of the test set we used the RANGEN2 generator[1]. This generator was introduced by Van-Houcke et al. in [30], in which they also evaluate the performance of RANGEN2 and three other project network generators. They showed that, while all generators are able to produce a large variety of different project networks, RANGEN2 produced the most varied networks in the shortest time.

The networks generated for the new test set have the following fixed properties:

- 50 activities.
- 4 resource types.
- Resource Factor set to 0.8.
- Resource Strength set to 0.25.

In the generation of the test set we varied the paralellism inherent in the instances in 21 steps of 1000 instances, resulting in a set of 21000 instances that cover the entire spectrum of Order Strength from 0 to 1.

To show the effect of generating our own test set, we compare it with the PSPLIB testset J60, which contains 480 instances of 60 activities. Figure 4.1 presents the variance in the instances, by sorting the instances against three resource measures Resource Factor (RF), Resource Strength (RS) and Resource Constrainedness (RC). The x-axis of these figures presents the position inside the sorted lists, and due to the difference in number of instances this results in a double scale: Thus, the instances at $20000/480$ correspond to the instance in our testset with RF (or RS / RC) higher than 20000 other instances, while for Kolisch' set it is the instance with the maximum RF/RS/RC, since it contains 480 instances. We can

[1] http://www.projectmanagement.ugent.be/?q=research/RanGen

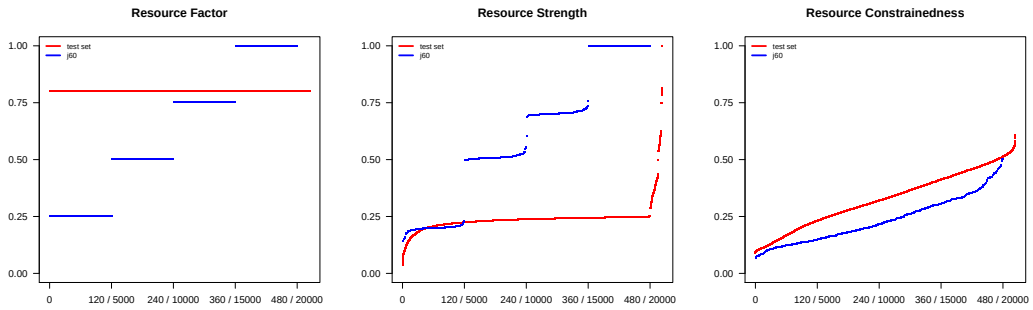


Figure 4.1: The generated test set compared to J60 on the resource measures.

clearly see the four factors of RF and RS in the PSPLIB benchmark set, while the new test set has a (nearly) uniform level for both measures.

The uncontrolled factor of Resource Constrainedness is not constant over the newly generated instances, which is part of the expectation 2 that Resource Constrainedness is correlated to Order Strength. If Resource Constrainedness was constant, it would be uncorrelated to any measure.

4.2 Testing the New Model

The test set presented in section 4.1 contains instances across the entire range of Order Strength for fixed Resource Strength. This allows us to easily test the expectation 2 that Resource Constrainedness is correlated with Order Strength. However it does not allow us to test expectation 3 that Resource Strength is correlated with Order Strength for fixed Resource Constrainedness. To this end we derive a second test set from the original which has fixed Resource Constrainedness. This is done by using the following rule (where $\mathrm{U}[x, y]$ stands for the uniform distribution from x to y):

$$\forall u(a_i, r_k) > 0 \quad u(a_i, r_k) = \frac{c(r_k)}{3} + \mathrm{U}\left[\frac{-c(r_k)}{4}, \frac{c(r_k)}{4}\right]$$

This second set thus has a mean Resource Constrainedness of $\frac{1}{3}$, which is approximately equal to the mean found in the first set. Because we only replace the resource usages when they are greater than zero, the Resource Factor is unaffected by this transformation. Similarly, the precedence constraints are unaffected and thus Order Strength stays the same. Thus, the second test set has a fixed Resource Constrainedness but is otherwise identical to the first set.

To search for the expected correlations we measure the Resource Constrainedness and Resource Strength of each instance in both sets. These values are then tested against the Order Strength using the Pearson correlation coefficient. This coefficient ranges from 0 (no correlation) to 1 or -1 (completely linear correlation). Considering the noise introduced

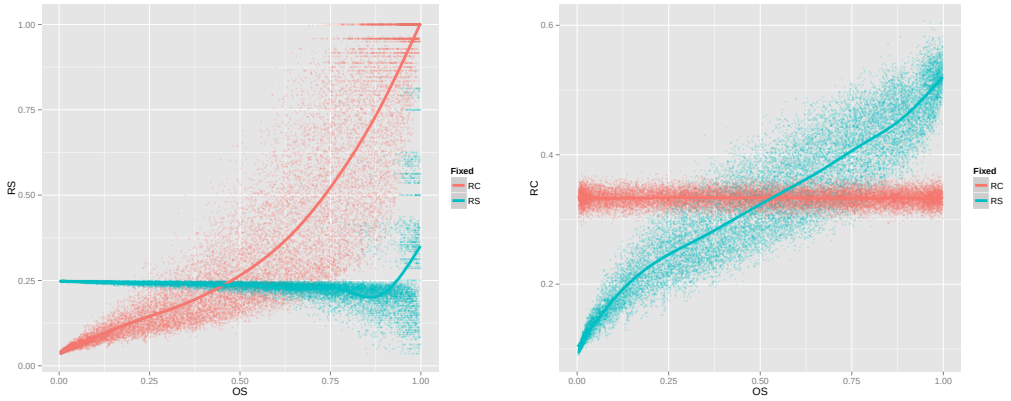


Figure 4.2: The left figure presents the correlation of Resource Strength and Order Strength. The right figure presents the same for Resource Constrainedness. The blue set has fixed Resource Strength, while the orange set has fixed Resource Constrainedness.

by randomness, we consider a value exceeding 0.75 to indicate correlation, while values exceeding 0.9 are considered strongly correlated. Figure 4.2 presents the results graphically.

We see that our expectations about the correlation of the resource parameters with Order Strength are met by the results: When Resource Constrainedness is fixed, Resource Strength is correlated to Order Strength with a Pearson correlation coefficient of 0.865, while Resource Constrainedness is correlated to Order Strength with coefficient 0.938 for fixed Resource Strength. The figures confirm that the correlation for Resource Constrainedness is stronger than that for Resource Strength which may be explained by the relatively high variance in the value for fixed Resource Constrainedness.

One expectation is not yet tested: instances with fixed Resource Strength are on average more difficult than instances with fixed Resource Constrainedness. As section 3.3 explains, when testing with heuristics the deviation from the best known solution is an approximation of the difficulty. Thus, to test this expectation we solve each instance using both SGS algorithms applied with the Latest Finish Time priority rule and compare the results with the lowest makespan solution encountered (including solutions found in later experiments). The difference in mean deviation is tested for significance using the Wilcoxon rank test, since we cannot assume the normal distribution for the deviations. Figure 4.3 presents the results graphically.

We find that mean deviation with fixed Resource Strength is significantly larger than that of fixed Resource Constrainedness. The difference is about 0.5% in the mean, which is more than 10% of the mean deviation of fixed Resource Strength (mean is 4.46%). Nevertheless, the results for this experiment are not entirely consistent. Particularly the Serial SGS seems to have difficulty with the low Order Strength fixed Resource Constrainedness instances, which skews the results somewhat.

An explanation for this behavior may be found in the way the parallel rule schedules: The Parallel SGS fills the resource to capacity before advancing the decision set D, while

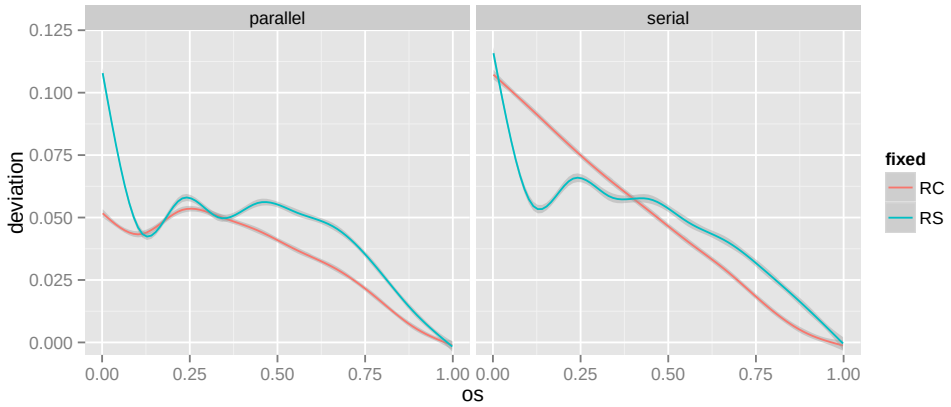


Figure 4.3: Mean deviation from best known solution using the LFT heuristic for varying Order Strength.

the Serial SGS must adhere more strictly to the priorities imposed by the LFT rule. Of course, when the Order Strength is low, there is little guidance from the LFT rule since it only considers the precedence network which is very sparse. This also explains why performance is very bad for the lowest OS instances in general.

4.3 Exploring FBI Performance

We want to explore two aspects of FBI: What is the impact of the the forwards pass (research question 3) and how is the performance of FBI affected by instance difficulty (research question 4). To do so we use the same experimental setup as in the previous section. The solutions obtained in the previous experiment are improved using the FBI algorithm. To test the effect of the forward pass, the intermediate schedule is also stored. Figure 4.4 presents the results.

Based on the results we can make four observations:

1. Solutions by the Parallel SGS are improved much more than those by the Serial SGS.
2. The largest improvement is obtained for low Order Strength.
3. Only extreme values of Order Strength (OS > 0.975) do not show significant improvement at the 2.5% one-sided confidence interval.
4. The improvement provided by the second justification pass is also significant, but much smaller than that provided by the first pass.

The first observation can be explained by looking at the solution types. The Parallel SGS only produces Non-Delay schedules. By applying FBI some of these schedules will become Active schedules which may be shorter than the best Non-Delay schedule. The second observation may somewhat be expected as a consequence of the poor solutions LFT

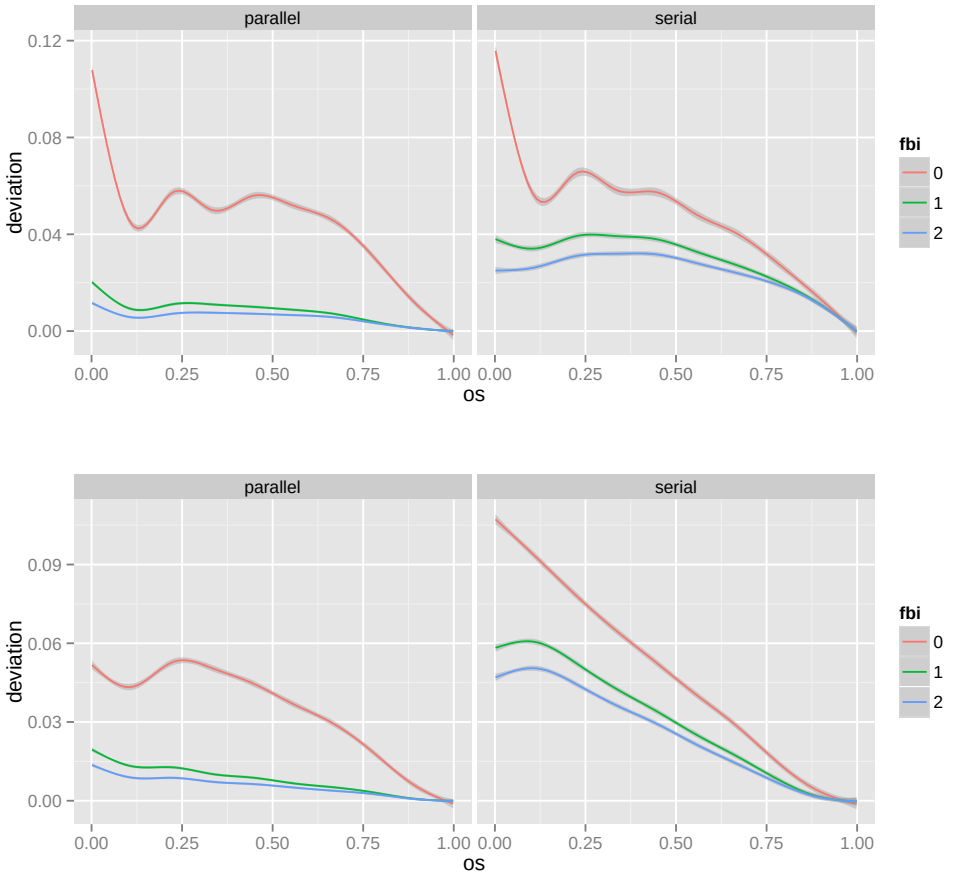


Figure 4.4: Mean deviation from best known solution using the LFT heuristic plus one or two applications of justification. Top image: RS fixed, bottom image: RC fixed.

generates for low Order Strength. The third and fourth observations are both quite surprising. One justification pass is sufficient to produce a considerable improvement in mean makespan.

Chapter 5

Analysis of Forward-Backward Improvement

The previous chapter performed exploratory experiments on the performance of FBI in section 4.3, which resulted in four observations on its interaction with the order parameter Order Strength and the two different types of SGS. This chapter aims to analyze these observations and pose hypotheses to explain them. These hypotheses lead to new expectations on the behavior of FBI, including a new priority rule heuristic that is designed to use the underlying principle of FBI. The observation that FBI performs well when applied to solutions produced by the Parallel SGS is analyzed by section 5.1, while the other observations are treated in section 5.2.

5.1 Explaining the Performance of FBI on Parallel SGS

Even though the Parallel SGS already produces better schedules than the Serial SGS on average, the improvement effect of FBI is also much stronger on the schedules produced by the Parallel SGS. Since the priority rule heuristic used is the same, only the type of SGS can be the cause of this difference. To explain the difference, therefore, we should reason about the difference in the scheduling approach used by the Parallel and the Serial SGS, and how those interact with FBI.

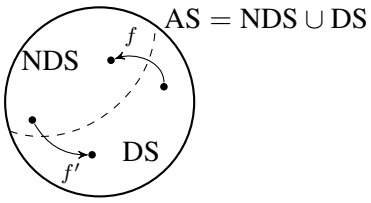


Figure 5.1: The solution space of Active Schedules is divided into Non-Delay Schedules and its complement, Delay Schedules. Functions f and f' turn solutions into a similar schedule in the complementing set.

Consider figure 5.1, the space of Active Schedules (AS) is divided into Non-Delay Schedules (NDS) and its complement, Delay Schedules (DS). We will exploit the fact that some reasoning was already done in [17], who showed that the Parallel SGS produces NDS, while the Serial SGS produces AS (thus either DS or NDS). 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, a NDS produced by the Parallel SGS may be justified into a DS. We propose that this transformation generally causes a reduction in makespan. Then the observation that FBI works better on the Parallel SGS is easily explained: The Serial SGS produces fewer NDS than the Parallel SGS.

Hypothesis 2. Transforming a NDS into a similar DS results in a shorter makespan schedule, unless the NDS schedule was optimal.

To make the intuition behind the proposed hypothesis clear, we first reason from the opposite operation. Consider a function f : DS $\rightarrow$ NDS that transforms a DS into the nearest NDS. Here we will sketch the way f works: In a DS we can identify at least one activity a_i which could have partially started at an earlier time t. We say that such an activity a_i with the lowest potential t is in the active position. Function f starts the a_i in active position at earliest time t. Because the original schedule was Active starting a_i at time t results in an overconsumption at a later time t'. This overconsumption must be resolved to again obtain a feasible schedule.

An increase in resource consumption can only be caused by the start of another activity a_j. Thus at least one a_j that started after t and finished before $s(a_i)$ in the original schedule must now be started after a_i finishes. To see that a_j must start later, consider that it was not in the active position originally and that starting a_i before a_j can only increase resource use at all times before the start of a_j. If a_j was part of the critical chain, starting a_j later increases the makespan of the schedule.

If we sequentially schedule the activity in the active position as early as possible and resolve the conflicts, then at some point no more activities are in the active position and the schedule has been transformed into a Non-Delay schedule that is still quite similar to the original schedule. 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 operation f' cannot be easily defined. After all, there is no easy way to select the task to 'delay'. However we propose that the FBI algorithm performs this function on NDS schedules. We can derive a number of expectations from hypothesis 2 that can be used to test it. In the first place, we may simply look at the type of schedule before and after justification:

Expectation 5. 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 ensures FBI produces NDS, which means the hypothesized effect will not be present in the results, and we thus expect less improvement (expectation 6). 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 a NDS. This implies that input DS will be improved more by Parallel-Serial FBI than by Serial-Serial FBI (expectation 7). These expectations will be tested in the next section.

Expectation 6. If FBI is performed with the Parallel SGS instead of the Serial SGS, schedules will be improved significantly less.

Expectation 7. If FBI is applied with Parallel SGS used only in the backward pass, its effect is strengthened on input Delay schedules.

These expectations will be tested in chapter 6.

5.2 Explaining the Implicit Priorities of FBI

The results from the exploratory study allow us to conclude that the performance of FBI is not due to the fact that it consists of two passes. Thus, when analyzing the way FBI assigns priorities to activities, we can restrict ourselves to a single justification pass. Further, we have seen that FBI produces the biggest relative improvement when Order Strength is low. However there are two reasons we do not ascribe this observation to FBI: In the first place, the priority rule used is Latest Finish Time. If the instance has low Order Strength it contains almost no precedence constraints, and this rule will have no way to differentiate between tasks. Therefore many activities will end up having equal priority. This leads to mostly random tie breaking, and thus random sampling instead of heuristic guidance. In the second place, chapter 4 presents evidence for hypothesis 1. This hypothesis implies that instances with a fixed level for Resource Strength should have roughly equal difficulty, from which we conclude that LFT is to blame for the bad performance (as opposed to FBI for its good performance relative to LFT).

The consequence is that in order to explain the performance of FBI we should look for a general property of the justification algorithm. The key insight here is that justification looks at a feasible schedule from the opposite side. 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 schedules produced by an SGS are Active, for each activity a_i in a schedule we can identify its immediate predecessors. These predecessors are exactly those activities that prevent a_i from starting any earlier. These activities need not be linked with precedence constraints. In other words, in any Active schedule there is always at least one activity the cause of the start time of any activity that does not start at time 0. Following this causality to the task that does start at time 0 gives us a *chain* of activities.

Justification prioritizes the final task to finish the highest, 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. Thus for a schedule improved by Justification the longest chain was delayed compared to the Earliest Start schedule s_{EST}. This difference between the chain length in the realized schedule and s_{EST} is the *realized delay*.

Apparently, 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 their Minimum Latest Finish Time will schedule them according to their realized delay. This would explain why LFT is such a successful 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 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 3. Scheduling activities by prioritizing the longest expected chain of successors results in good schedules.

Instead of using a feasible schedule as estimate, we may try to estimate 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 [23]. The idea of estimating the risk of delay in s_{EST} is formalized in a priority rule in section 5.2.1.

5.2.1 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. Further, we have seen in the examples that 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 may try to estimate the delay of a task i in a EST peak on resource k as follows:

$$\text{virtualdelay}_{i,k} = \frac{\max\left(0, \text{peak}_k - \text{cap}_k\right)}{\text{cap}_k} \times \frac{\text{usage}_{i,k}}{\text{cap}_k} \times \text{avg. task duration}$$

The first term 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 term accounts for the likelihood a task is delayed due to its use of the resource, while the third term simply accounts for the amount of time a task is expected to be delayed if another task is started first. If there is more than one resource (peak) involved at the earliest start time of a task, it is fairest to estimate the delay by the maximum of all delays caused by the peaks. If the task is cleared for the highest peak, it is certainly cleared for smaller peaks. Thus:

$$\text{virtualdelay}_i = \max_{r_k \in R} \left(\text{virtualdelay}_{i,k}\right)$$

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.

$$\text{chain}_i = \text{virtualdelay}_i + \text{duration}_i + \max_{(a_i \prec a_j) \in P} (\text{chain}_j)$$

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.

5.2.2 Expectations

Suppose that hypothesis 3 is correct, and that this is the reason why FBI works as well as it does. Then, considering that the Resource Profile Rule computes more accurate estimates of the risk of delay than existing priority rules such as LFT, we expect the following:

Expectation 8. The Resource Profile rule assigns the same priorities to activities as the Justification algorithm.

Expectation 9. The makespan of schedules produced by using the Resource Profile rule will equal those produced by FBI applied to schedules produced with LFT.

Chapter 6

Testing

The analysis of the functionality of FBI in chapter 5 has resulted in a number of expectations about the FBI procedure. This chapter reports on the experiments we performed to test these expectations. First, we investigate what type of schedules are improved by FBI to test expectation 5. Next, the priorities of FBI are compared with the priorities of the proposed Resource Profile Rule to test expectation 8. And finally, the remaining expectations are tested by altering the type of schedule used by FBI, and comparing its impact on the SGS and priority rules.

6.1 FBI types

To see if justification improves NDS schedules by transforming them into DS schedules, we computed the type of all schedules obtained on the RS fixed test set in the exploratory experiments. Table 6.1 presents 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 schedule that is Delay (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 schedules that are Delay. This matches our expectation that justification

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

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 only starts producing Non-Delay schedules when Order Strength is high. For these instances, the schedules are Non-Delay because of precedence constraints, and thus justification cannot improve them.

6.2 Priority Comparison

Each of the priority rules produces an ordering of activities based on their priority value, such as its finish time in the case of LFT. It is also the magnitude of difference in this priority value that affects its probability of being selected in biased random sampling. Thus, when the priorities assigned between two different rules are similar, the expectation is that their resulting schedules will be similar. To see if RP and FBI are similar we might try to compare the priorities each assign to an activity. Because these values are not on the same scale, they must be normalized. We use the following normalization which ensures the lowest priority task $p(a_i)$ has priority value 0, while the highest has value 1:

$$\frac{p(a_i) - \min\limits_{a_j \in A} p(a_j)}{\max\limits_{a_j \in A} p(a_j) - \min\limits_{a_j \in A} p(a_j)}$$

Then each priority rule heuristic defines a vector of priorities that is independent of the SGS used. The similarity between two rules on an instance is the Euclidean distance between these two normalized vectors. To test if the RP rule is similar to FBI we compute the Euclidean distance between the RP rule and the second justification pass of FBI applied to the schedules obtained with LFT. We use the second pass of justification for two reasons: First, this matches what we set out to achieve, which is to bypass (both passes of) FBI. Second, this pass of justification is performed in the same schedule direction as the RP rule. The priorities assigned in the first, backward pass would not be comparable. To give the results a reference, this distance is compared to the Euclidean distance between the LFT rule and FBI. The results are in figure 6.1.

We see that the distance between priorities assigned by RP and FBI is indeed significantly shorter than those assigned by LFT. The distance is almost linearly decreasing with Order Strength for both rules which suggests that there is still a lot of room for improvement in the low Order Strength region. One particularly interesting aspect is the way the RP compares to FBI for fixed Resource Constrainedness. Under fixed RC, the RP rule stops growing more distant at the low Order Strength, which implies that RP is able to find relatively good priorities when Resource Constrainedness is high relative to the precedence constrainedness. This is an implicit verification of the hypothesis 1 posed in chapter 3: Keeping only resource constrainedness fixed results in a peak difficulty, beyond which instances become easier again.

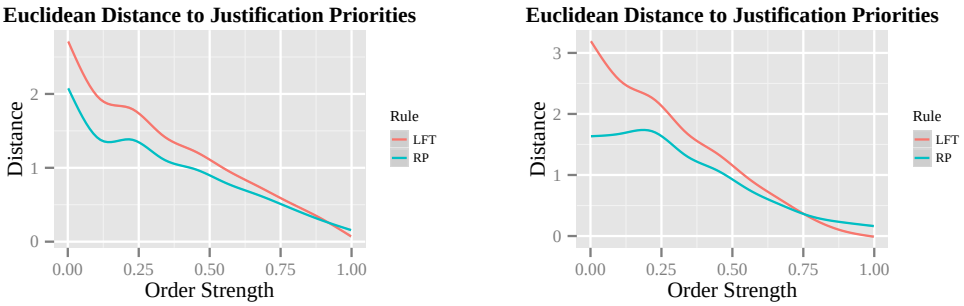


Figure 6.1: Euclidean distance between the normalized priorities of an FBI justification heuristic and the LFT or RP rule. Left figure for fixed Resource Strength, right figure for fixed Resource Constrainedness.

6.3 On the Generated Set

Finally, we want to test the expectations on the way justification improves schedules through its solution types. To this end we implemented double justification in four different modes, depending on which SGS is used in the first and second pass (Serial-Serial, Serial-Parallel and so on). Each of these four implementations was applied to the schedules obtained earlier by using the LFT and RP heuristics in combination with the SGSs on the RS fixed set. The results are in figure 6.2. The line for 'None' in the top figures is equal to the line for '0' in figure 4.4, while 'Ser-Ser' is equal to '2' in that same figure.

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 9, 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 6), 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 7.

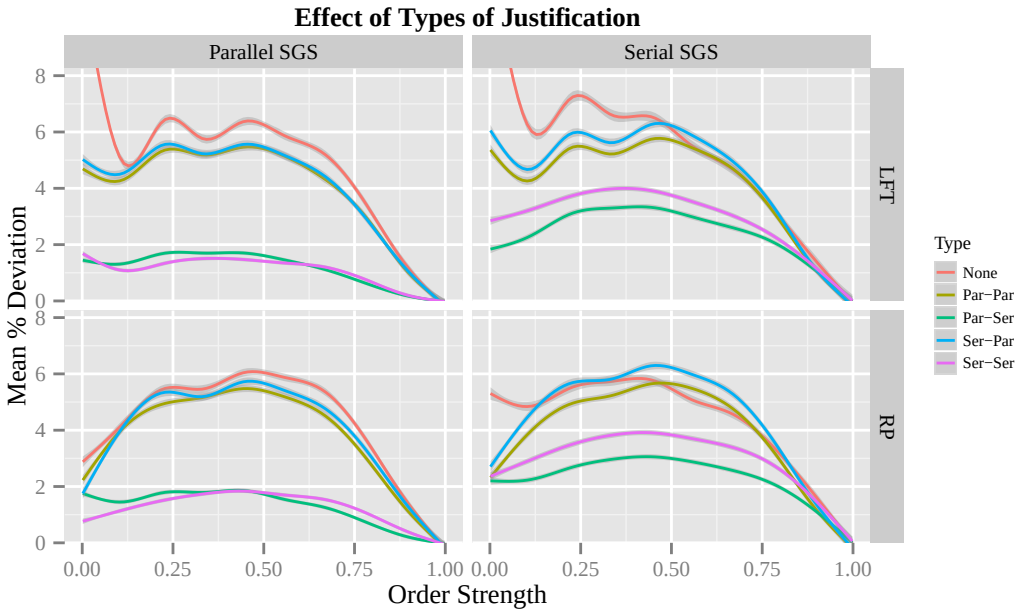


Figure 6.2: Effect of applying the different combinations of SGS in the justification algorithm.

Chapter 7

Discussion

In this thesis we empirically investigated the unknown nuances and principles behind the well-known performance of the Forward-Backward Improvement algorithm. We identified five research questions for which the answers should lead us to an increased understanding of this algorithm, and in the process produce a new priority rule capable of harnassing the performance of FBI. This chapter will discuss the answers we have found, how they may be used by others, and what future work remains.

7.1 Contributions

This section describes the main contributions of this thesis, and how they might influence the research field and industry.

On Instance Difficulty

In the process of answering research question 5 we encountered several answers in literature. Several authors identified that in particular controlling measures Resource Strength and Resource Constrainedness (both measures of resource conflictedness) resulted in phase transitions in difficulty. This lead Herroelen and De Reyck to conjecture that these two instances are correlated in [12]. From literature it would appear that the precedence constraints are unimportant for instance difficulty.

Instead, we conjecture that the precedence constrainedness is equally important to the resource constrainedness for instance difficulty. We find evidence of this conjecture in the way Order Strength is correlated with the resource measures of Resource Strength or Resource Constrainedness depending on which of those two is held constant. Further, we demonstrate that keeping Resource Strength fixed results in instances that are more difficult than those that keep Resource Constrainedness fixed. This provides evidence that keeping the resource and precedence constraints balanced results in the most difficult instances.

Understanding what makes an instance difficult is valuable for the scientific community: Currently, evaluation of algorithms for the RCPSP almost exclusively happens on the test sets put forward by Kolisch, Sprecher and Drexl. This situation risks designing algorithms tailored for the characteristics of this test set only. Furthermore, the focus on mean

performance only may result in promising algorithms to remain unpublished. If we want to generate new representative test sets, we must understand what parameters are interesting to factor on.

This result is less interesting for industry. Although understanding what parameters make a randomly generated instance difficult can give some impression of the difficulty of problems in industry, it is understood that randomly generated problems typically are not representative of real-world instances [13].

On FBI Performance

Investigating the research questions 4 and 3 allowed us to reason about the way FBI improves schedules. We found that FBI is able to improve schedules irrespective of the levels of the order parameters, and that it is able to do so without the need for both passes. This suggests that FBI operates on a general principle, answering research question 1: Scheduling activities by the longest expected chain of successors results in good schedules and a feasible schedule is a good estimate of this expected chain length.

Although it was known that FBI is able to produce good improvement in the mean, the confirmation that it works on all types of instances suggests that when solution quality is required, FBI should always be applied. This recommendation is valuable for industry where quality is typically all that counts. The principle underlying FBI gives the research field a suggestion of an interesting search direction for more successful algorithms.

On Bypassing FBI

Finally, we also designed a new priority rule based on the principle uncovered in answering research question 1. By testing the performance of this new rule we were able to present evidence that this principle is indeed behind FBI. Although the new priority rule did not allow to bypass the need for FBI post-processing, it is significantly better than the current best known priority rule LFT. This suggests that studying the principle underlying FBI is an interesting search direction for more successful algorithms.

7.2 Future Work

These are suggestions of research directions and experiments we did not pursue in this thesis, but that could nevertheless be intersting to follow.

7.2.1 Expected Mean Makespan based on Hypothesis 1

We may deduce more expectations from hypothesis 1 than we did in this thesis. In particular, the balance between resource constraints and precedence constraints may be made visible by looking at the mean optimal makespan of instances: Activity durations of randomly generated instances follow a fixed (typically uniform) distribution. The makespan of a feasible schedule is then not an arbitrary number, but it is in expectation a multiple of the expected value of the distribution used when generating instances. This is a consequence of

the critical chain consisting of a number of whole task durations. When an instance is more constrained, the optimal makespan will be higher (fewer tasks can start in parallel). Thus, we would expect that if the constraints are balanced when Resource Strength is fixed, that the mean optimal makespan would grow with increasing Order Strength. On the other hand, if Resource Constrainedness is held fixed, the mean makespan would start out horizontally for increasing Order Strength until the 'balanced constraints' tipping point, after which it would start increasing linearly with Order Strength.

7.2.2 Parallelizing the Resource Profile Rule

It is clear that the Resource Profile rule can be improved more in the low Order Strength range from the way it diverges from the priorities assigned by FBI. Another sign is in the way the quality of solutions produced by the Serial and Parallel SGS diverge at low Order Strength with the RP rule. It is possible to let the Serial SGS produce solutions that are typical of the Parallel SGS by modifying the priority rule: The longer an activity is in the decision set, the higher its priority becomes relative to the newly introduced activities. FBI also intuitively uses this idea of locality because it schedules tasks in the order of a feasible schedule. This modification may make the RP rule more accurate.

7.3 Discussion

In this thesis we attempted to follow an empirical methodology for studying the heuristic algorithms for the RCPSP problem. By designing the experiments around the expectations we derived from the hypotheses, we were able to present results that not only demonstrated that the RP rule works well, but that also strengthen our understanding why rules such as the RP rule should work better than rules like LFT. We find that following this methodology not only helps to produce focused, informed experiments, but also naturally opens up new research questions and study subjects. We hope that the reader is convinced by this thesis that it is valuable to use this methodology when researching algorithms.

Bibliography

[1] M. Bartusch, Rolf H. Möhring, and F. J. Radermacher. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16(1):199–240, December 1988. 10

[2] J Blazewicz, J.K. Lenstra, and A.H.G.Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, January 1983. 2

[3] Peter Brucker, Sigrid Knust, Arno Schoo, and Olaf Thiele. A branch and bound algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 107(2):272–288, June 1998. 3

[4] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the Really Hard Problems Are. In *IJCAI*, pages 331–337, 1991. 6

[5] Bert De Reyck and Willy S. Herroelen. On the use of the complexity index as a measure of complexity in activity networks. *European Journal of Operational Research*, 91(2):347–366, June 1996. 22, 23, 24

[6] Erik L. Demeulemeester and Willy S. Herroelen. New Benchmark Results for the Resource-Constrained Project Scheduling Problem. *Management Science*, 43(11):1485–1492, November 1997. 3

[7] Erik L. Demeulemeester and Willy S. Herroelen. *Project Scheduling: A Research Handbook*, volume 49 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, Boston, 2002. 1

[8] Ulrich Dorndorf, Erwin Pesch, and Toàn Phan-Huy. A Time-Oriented Branch-and-Bound Algorithm for Resource-Constrained Project Scheduling with Generalised Precedence Constraints. *Management Science*, 46(10):1365–1384, October 2000. 3

[9] Chen Fang and Ling Wang. An effective shuffled frog-leaping algorithm for resource-constrained project scheduling problem. *Computers & Operations Research*, 39(5):890–901, May 2012. 4

[10] Evgeny R. Gafarov, Alexander a. Lazarev, and Frank Werner. Approximability results for the resource-constrained project scheduling problem with a single type of resources. *Annals of Operations Research*, March 2012. 3

[11] Sönke Hartmann and Rainer Kolisch. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 127(2):394–407, December 2000. 3

[12] Willy S. Herroelen and Bert De Reyck. Phase transitions in project scheduling. *Journal of the Operational Research Society*, 50(2):148–156, February 1999. 25, 47

[13] J. N. Hooker. Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1(1):33–42, September 1995. 4, 21, 48

[14] Andrei Horbach. A Boolean satisfiability approach totheresource-constrained project scheduling problem. *Annals of Operations Research*, 181(1):89–107, February 2010. 3

[15] ISO. ISO 10006: Quality management systems – Guidelines for quality management in projects, 2003. 1

[16] Qiong Jia and Yoonho Seo. An improved particle swarm optimization for the resource-constrained project scheduling problem. *The International Journal of Advanced Manufacturing Technology*, January 2013. 4

[17] Rainer Kolisch. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90(2):320–333, April 1996. 3, 13, 15, 18, 28, 38

[18] Rainer Kolisch and Sönke Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 174(1):23–37, October 2006. 4

[19] Rainer Kolisch, Christoph Schwindt, and Arno Sprecher. Benchmark Instances for Project Scheduling Problems. In *Handbook on Recent Advances in Project Scheduling*, pages 197–212. 1998. 31

[20] Rainer Kolisch, Arno Sprecher, and Andreas Drexl. Characterization and generation of a general class of resource-constrained project scheduling problems. In *Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel*, number 301, 1992. 9, 22

[21] Rainer Kolisch, Arno Sprecher, and Andreas Drexl. Characterization and Generation of a General Class of Resource-constrained Project Scheduling Problems. *Management Science*, 41(10):1693–1703, 1995. 21, 22, 23, 25, 28, 31

[22] Aristide Mingozzi, Vittorio Maniezzo, Salvatore Ricciardelli, and Lucio Bianco. An Exact Algorithm for the Resource-Constrained Project Scheduling Problem Based on

a New Mathematical Formulation. *Management Science*, 44(5):714–729, May 1998. 3

[23] Nicola Policella, Amedeo Cesta, Angelo Oddi, and Stephen F. Smith. From Precedence Constraint Posting to Partial Order Schedules. *AI Communications*, 20(3):163–180, 2007. 40

[24] A Alan B Pritsker, L. J. Waiters, and Philip M Wolfe. Multiproject Scheduling with Limited Resources: A Zero-One Programming Approach. *Management Science*, 16(1):93–108, September 1969. 3

[25] Pejman Sanaei, Reza Akbari, Vahid Zeighami, and Sheida Shams. Using Firefly Algorithm to Solve Resource Constrained Project Scheduling Problem. In Jagdish Chand Bansal, Pramod Kumar Singh, Kusum Deep, Millie Pant, and Atulya K. Nagar, editors, *BIC-TA*, volume 201 of *Advances in Intelligent Systems and Computing*, pages 417–428, India, 2013. Springer India. 4

[26] Andreas Schutt, Thibaut Feydy, Peter J. Stuckey, and Mark G. Wallace. Explaining the cumulative propagator. *Constraints*, 16(3):250–282, August 2011. 3

[27] Stephen F. Smith. Is Scheduling a Solved Problem? In *MISTA*, pages 3–17, 2003. 2

[28] Arno Sprecher, Rainer Kolisch, and Andreas Drexl. Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 80(1):94–102, January 1995. 3, 11

[29] Vicente Valls, Francisco Ballestín, and Sacramento Quintanilla. Justification and RCPSP: A technique that pays. *European Journal of Operational Research*, 165(2):375–386, September 2005. 4, 19

[30] Mario Vanhoucke, José Coelho, Dieter Debels, Broos Maenhout, and Luís V. Tavares. An evaluation of the adequacy of project network generators with systematically sampled networks. *European Journal of Operational Research*, 187(2):511–524, June 2008. 32

[31] James M. Wilson. Gantt charts: A centenary appreciation. *European Journal of Operational Research*, 149(2):430–437, September 2003. 11