Computing Explanations for the Unary Resource Constraint

Petr Vilím

Charles University
Faculty of Mathematics and Physics
Malostranské náměstí 2/25, Praha 1, Czech Republic
vilim@kti.mff.cuni.cz

Abstract. Integration of explanations into a CSP solver is a technique addressing difficult question *"why my problem has no solution"*. Moreover, explanations together with advanced search methods like directed backjumping can effectively cut off parts of the search tree and thus speed up the search.
In order to use explanations, propagation algorithms must provide some sort of reasons *(justifications)* for their actions. For binary constraints it is mostly easy. In the case of global constraints computation of factual justifications can be tricky and/or computationally expensive.
This paper shows how to effectively compute explanations for the unary resource constraint. The explanations are computed in a lazy way. The technique is experimentally demonstrated on job-shop benchmark problems. The following propagation algorithms are considered: edge-finding, not-first/not-last and detectable precedences. Speed of these filtering algorithms and speed of the explanation computation is the main interest.

1 Introduction

To show a typical usage of the unary resource constraint, let us consider the following *shop-scheduling* problem. We are given a set of machines and a set of *jobs* which must be processed. A job consists of a set of *operations*, each operation requires exclusive usage exactly one machine. Processing of an operation cannot be interrupted by any other operation. Exact processing time of each operation is known in advance. In the case of *jobshop* problem, operations in a job must be processed in a certain order. In *openshop* an order of operations in a job is arbitrary. The problem is to find a schedule with minimal completion time of all jobs, i.e. a schedule with minimal *makespan*.

Shop-scheduling problems can be modeled as a constraint satisfaction problem (CSP). In this case unary resource[1] constraints are typically used as abstractions of machines. In case of openshop, unary resource constraints are also used to model jobs.

In relation to a resource operations are called *activities*. Each activity i has following requirements:

[1] In this paper, a resource always denotes a unary resource.

– earliest possible starting time est_i
– latest possible completion time lct_i
– processing time p_i

A purpose of the resource constraint is to reduce a search space by tightening the time bounds est_i and lct_i. This process of elimination of unfeasible values is called *propagation*, an actual propagation algorithm is often called *filtering* algorithm. Interval $\langle \mathrm{est}_i, \mathrm{lct}_i \rangle$ is called a *time window* of the activity i. Thus the role of the resource constraint can be seen as a process of tightening of these time windows.

There are several filtering algorithms for unary resources, in this paper we focus on the edge-finding [7, 8], not-first/not-last [10, 9, 3] and detectable precedences [10]. Each of these algorithms filters out different inconsistent values, therefore these algorithms can be used together to achieve better pruning.

There are only a few attempts to combine explanations with unary resource constraint. The author is aware of the paper [6] where Guéret et. al. solved several open openshop problems using explanations. This result was achieved by very simple explanations for unary resource constraints. This paper focuses on computing more accurate explanations. In paper [2] explanations are used for solving dynamic schedule problems.

This paper differs from the previous work in two main aspects: explanations are computed in a lazy way and justifications are very tightly connected with filtering algorithms. This way the computation is very fast and resulting explanations are more accurate.

2 Explanations

The purpose of the explanation is to capture a *reason* why a search (sub)tree failed. Advanced search methods (directed backjumping, dynamic backtracking [5]) can exploit such information and speed up the search. The idea is to identify a reason of fail and cut off other branches of the search tree which are known to fail for the same reason.

An explanation has to describe all properties of the subproblem which leads to the infeasibility. This way the explanation can be seen as a *relaxation* of the original unfeasible subproblem. The important point is that this relaxation remains unfeasible. Our intention is to find as general relaxation (explanation) as possible. More general explanation can cover more subproblems and dismiss them as unsolvable.

Let us precisely define a specific type of explanations which is used in this paper:

Definition 1. *An fail explanation is an unfeasible CSP*[2] *which is a relaxation of the current search node. The explanation consists of:*

[2] Constraint Satisfaction Problem

1. *A subset Υ of initial constraints and search decision constraints valid in the current search node.*
2. *Conflict windows $\langle \underline{\mathrm{est}}_i,\ \underline{\mathrm{lct}}_i \rangle$ for activities. A conflict window for an activity i is a superset of the current time window: $\langle \mathrm{est}_i,\ \mathrm{lct}_i \rangle \subseteq \langle \underline{\mathrm{est}}_i,\ \underline{\mathrm{lct}}_i \rangle$. I.e. the conflict window is a relaxation of the current time window.*
   If no conflict window is given for an activity i then we consider the time window to be $\langle -\infty,\ \infty \rangle$.

The idea of this definition follows. In explanation we relax constraints which are not in the set Υ. We also relax domains by replacing time windows by conflict windows. And still the problem remains unfeasible. Conflict window $\langle -\infty,\ \infty \rangle$ is special, it says that the activity is irrelevant: can be processed at any time and yet the problem has no solution.

The explanation can be compared with state in any other search node. A problem has no solution as long as all constraints from the set Υ remain in the system and all time windows are covered by associated conflict windows.

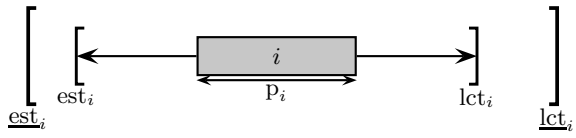

Fig. 1. Activity i, its time window and conflict window.

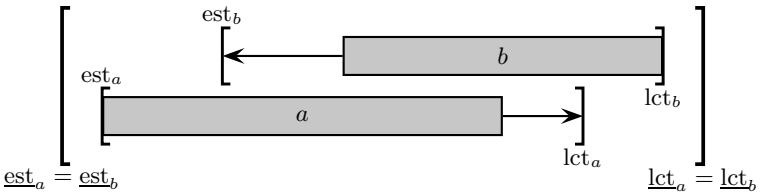

Fig. 2. Two activities a and b in conflict, their time windows and conflict windows.

2.1 Initial Explanation

When propagation comes to a dead end, an initial explanation must be computed. This initial explanation simply describe the reason of the fail which was found.

For shop-scheduling problems, the usual reason why propagation generates fail is *overloading*. Let us consider a subset $\Omega \subseteq T$ of activities on one resource. We can define processing time, earliest starting time and latest completion time

of the set Ω as:

$$\begin{aligned} \mathrm{est}_\Omega &= \min\{\mathrm{est}_i,\ i \in \Omega\} \\ \mathrm{lct}_\Omega &= \max\{\mathrm{lct}_i,\ i \in \Omega\} \\ \mathrm{p}_\Omega &= \sum\nolimits_{i \in \Omega} \mathrm{p}_i \end{aligned}$$

All activities from the set Ω must be processed during the interval $\langle \mathrm{est}_\Omega,\ \mathrm{lct}_\Omega \rangle$. However, if $\mathrm{p}_\Omega > \mathrm{lct}_\Omega - \mathrm{est}_\Omega$ then no solution exists. Empty domain for an activity i is a special case of overloading for $\Omega = \{i\}$.

The explanation for overloading consists of the unary resource constraint and conflict windows for activities $i \in \Omega$. These conflict windows can be $\langle \mathrm{est}_\Omega,\ \mathrm{lct}_\Omega \rangle$. However conflict windows can be little bit wider, as long as $\underline{\mathrm{lct}}_i - \underline{\mathrm{est}}_i \leq \mathrm{p}_\Omega - 1$. Let Δ is defined as:

$$\Delta = \mathrm{p}_\Omega - (\mathrm{lct}_\Omega - \mathrm{est}_\Omega) - 1$$

Conflict windows for $i \in \Omega$ can be set the following way:

$$\left\langle \mathrm{est}_\Omega - \left\lfloor \frac{\Delta}{2} \right\rfloor,\ \mathrm{lct}_\Omega + \left\lceil \frac{\Delta}{2} \right\rceil \right\rangle$$

2.2 Justifications

It is likely that the infeasibility of the problem cannot be simply detected by the overloading. Some propagation or even search must be done first. Initial explanation provided by overloading is just a beginning. The explanation must be refined during the way back in the search tree.

For this purpose, a *justification* must be remembered for each domain reduction. The justification captures the reason which justifies the realized reduction:

Definition 2. *Justification is a CSP which is a relaxation of the state just before the reduction. Filtering algorithm would generate exactly the same reduction for the relaxed CSP as for the original one.*

Justification consists of the propagated constraint and a set of conflict windows.

Justifications are written on the stack during constraint propagations and used for explanation (re)computation during way back in the search tree. Naturally, we are looking for as general justification as possible – more general justifications result in the more general explanation.

Let us describe more formally how justifications are used to refine explanations during way back in the backtrack:

1. Once a fail is found, an initial explanation is created.
2. One by one the reductions made by the constraint propagation are undone in the reverse order than they were originally made. For that, all realized reductions and their justifications are stored in a stack.

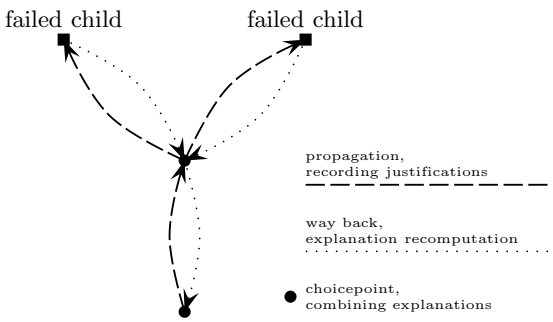

Fig. 3. Operations on the search tree

3. After undoing a particular reduction, it can happen that the explanation is not the relaxation of the current problem any more. For example $\underline{\text{est}}_i$ may become greater than est_i and thus the conflict window does not cover the time window any more. In that case, the explanation must be repaired using the justification associated with the undone reduction. It is done in the following way:
   i. $\underline{\text{est}}_i$ is set to $-\infty$.
   ii. Constraint which generated the reduction is added into the explanation: $\Upsilon := \Upsilon \cup \{c\}$.
   iii. Conflict windows from the justification are "merged" into the conflict windows of the explanation. Let $\langle \underline{\text{est}}'_k, \underline{\text{lct}}'_k \rangle$ be the conflict window for the activity k in the justification. Then the resulting conflict window for the activity k in the explanation is:

$$\langle \underline{\text{est}}_k, \underline{\text{lct}}_k \rangle := \langle \max\{\underline{\text{est}}_k, \underline{\text{est}}'_k\}, \min\{\underline{\text{lct}}_k, \underline{\text{lct}}'_k\} \rangle$$

4. In a choicepoint, the explanation is a combination of explanations from all child nodes. For example, let us suppose that the branching was done by addition of a constraint $\mathbf{a}$ in the first child node and the negation of this constraint $\neg\mathbf{a}$ in the second child node. The resulting explanation consists of:
   i. Subset of current constraints Υ:

$$\Upsilon = \left(\Upsilon^{\mathbf{a}} \setminus \{\mathbf{a}\}\right) \cup \left(\Upsilon^{\neg\mathbf{a}} \setminus \{\neg\mathbf{a}\}\right)$$

   ii. Conflict windows for activities $\langle \underline{\text{est}}_k, \underline{\text{lct}}_k \rangle$:

$$\langle \underline{\text{est}}_k, \underline{\text{lct}}_k \rangle := \langle \max\{\underline{\text{est}}^{\mathbf{a}}_k, \underline{\text{est}}^{\neg\mathbf{a}}_k\}, \min\{\underline{\text{lct}}^{\mathbf{a}}_k, \underline{\text{lct}}^{\neg\mathbf{a}}_k\} \rangle$$

Justifications are similar to explanations, however justifications are much more simple. For each particular filtering algorithm, justification can be held in a specialized data structure which exploit a particular method of filtering. Detailed descriptions of justifications for different algorithms are provided in following sections 4–7.

3 Directed Backjumping

Before going into details about justifications, let us show how to implement directed backjumping using explanations.

Consider a the choicepoint from the item 4 above. If $\mathbf{a} \notin \Upsilon^{\mathbf{a}}$ then the explanation for the first branch $\mathbf{a}$ is valid also for the second branch $\neg\mathbf{a}$. And so the second branch can be skipped because it would fail anyway. This way, explanations can speed up the search.

4 Precedence Justification

Together with resource constraint, binary precedence constraints are used to model shop-scheduling problems. A precedence constraint $i \ll j$ assures that the activity i finish before the activity j starts. Precedence constraints can be used to model ordering of operations within a job in jobshop. They are also often used as search decisions.

Let us introduce a notation convention. Whenever a reduction of a domain is made (i.e. est_i is increased or lct_i is decreased), then est_i and lct_i denote values before the reduction, est'_i and lct'_i denote values after the reduction.

Propagation of the precedence constraint $i \ll j$ is quite simple: whenever est_i is increased, the constraint propagates this change into the value est_j:

$$\mathrm{est}'_j := \max\{\mathrm{est}_j,\ \mathrm{est}_i + \mathrm{p}_i\} \tag{1}$$

Similarly, when lct_j is decreased, lct_i can be adjusted:

$$\mathrm{lct}'_i := \min\{\mathrm{lct}_i,\ \mathrm{lct}_j - \mathrm{p}_j\} \tag{2}$$

All propagation algorithms considered in this paper have two symmetric versions. One of them increase values est_i (in this case the rule (1)), the second one decrease values lct_i (the rule (2)). Since propagation algorithms and their justifications are symmetrical, we will always consider only one of these symmetric versions – the one which changes est_i.

Justification for a reduction made by the rule (1) is quite simple: precedence constraint itself and the conflict window $\langle \mathrm{est}_i,\ \infty\rangle$ for the activity i.

Now let us now focus on the usage of such justification. Explanation must be recomputed only if[3] $\mathrm{est}_j < \underline{\mathrm{est}}_j$. In that case, explanation is recomputed the following way:

i. Precedence constraint $i \ll j$ is added into the explanation.
ii. $\underline{\mathrm{est}}_i$ must be changed. According to justification it is enough to set $\underline{\mathrm{est}}_i := \mathrm{est}_i$. However, it is possible that $\underline{\mathrm{est}}_j < \mathrm{est}'_j$, i.e. to put the activity j into the conflict, it is enough to increase est_j to $\underline{\mathrm{est}}_j$. Thus it is enough to set $\underline{\mathrm{est}}_i := \underline{\mathrm{est}}_j - \mathrm{p}_i$.
iii. Conflict window of the activity j is enlarged: $\underline{\mathrm{est}}_j := -\infty$.

Recording and using one justification for a precedence constraint has both time complexity and space complexity $\mathcal{O}(1)$.

[3] Note that est_j is value before the reduction, i.e. the value after undo of this reduction.

5 Not-First/Not-Last Justifications

Filtering algorithm not-first/not-last [10, 3, 9] is based on the following rule *not-first* and its symmetric variant *not-last*. Let us consider an activity i and a set $\Omega \subset T$ such that the activity i cannot start before the set Ω. We denote such property $i \not\ll \Omega$:

$$\forall \Omega \subset T, \forall i \in (T \setminus \Omega): \quad \mathrm{lct}_\Omega - \mathrm{est}_i < \mathrm{p}_\Omega + \mathrm{p}_i \quad \Rightarrow \quad i \not\ll \Omega \tag{3}$$

If $i \not\ll \Omega$, some activity j from the set Ω must finish before the activity i can start. This allows to increase est_i:

$$i \not\ll \Omega \quad \Rightarrow \quad \mathrm{est}'_i := \max\left\{\mathrm{est}_i,\ \min\left\{\mathrm{est}_j + \mathrm{p}_j,\ j \in \Omega\right\}\right\} \tag{4}$$

A justification for such change of est_i must guarantee that if all activities remain inside conflict windows, inequality (3) remains valid and the value est'_i in the rule (4) remains the same or it is even greater.

The inequality (3) remains valid as long as lct_Ω does not increase too much. Hence for each activity j in the set Ω, the bound of the conflict window $\underline{\mathrm{lct}}_j$ must fulfill the following inequality:

$$\forall j \in \Omega: \quad \underline{\mathrm{lct}}_j < \mathrm{est}_i + \mathrm{p}_\Omega + \mathrm{p}_i$$

Similarly, as long as $\min\{\mathrm{est}_j + \mathrm{p}_j,\ j \in \Omega\}$ does not decrease, the rule (4) still justifies the reduction. Therefore:

$$\forall j \in \Omega: \quad \underline{\mathrm{est}}_j \geq \mathrm{est}'_i - \mathrm{p}_j$$

To fulfill both last inequalities, the conflict windows for activities from the set Ω are assigned in the following way:

$$\forall j \in \Omega: \quad \langle \mathrm{est}'_i - \mathrm{p}_j,\ \mathrm{est}_i + \mathrm{p}_\Omega + \mathrm{p}_i - 1 \rangle$$

Also, the conflict window $\langle \mathrm{est}_i, \infty \rangle$ must be assigned to the activity i.

Just constructed justification has time and space complexity $\mathcal{O}(n)$ because all activities from the set Ω must be enumerated into the justification. However, only some special types of sets Ω can be considered in order to find all reductions resulting from the rule not-first.

Let us consider one particular reduction according to the rule not-first (3), (4). Let Ψ be the set constructed the following way:

$$\Psi = \{j,\ j \in T\ \&\ \mathrm{est}_j + \mathrm{p}_j \geq \mathrm{est}'_i\ \&\ \mathrm{lct}_j \leq \mathrm{lct}_\Omega\ \&\ j \neq i\} \tag{5}$$

If we exchange the set Ω by the set Ψ in the rules (3) and (4), these rules would raise exactly the same change of the est_i. In fact, all not-first algorithms [10, 3, 9] consider only sets in this form. I.e. whenever a change of a est_i is made, $\Omega = \Psi$.

Thanks to this special form of the set Ω, the set Ω can be characterized only by the values est'_i and lct_Ω. Using the rule (5), the set Ω can be reconstructed

in time $\mathcal{O}(n)$. Thus the justification has size only $\mathcal{O}(1)$ and can be recorded in the time $\mathcal{O}(1)$. All three not-first/not-last algorithms [10, 3, 9] can be easily modified to record such justifications without changing their time complexities, i.e. $\mathcal{O}(n \log n)$ for [10] and $\mathcal{O}(n^2)$ for [9, 3].

Usage of each one not-first justification takes time $\mathcal{O}(n)$. One run of the not-first algorithm can generate only n changes maximum. Thus the way back "through" the not-first/not-last propagation takes $\mathcal{O}(n^2)$ maximum. In addition, a lot of justifications can be skipped because they do not interfere with the current explanation (i.e. $\mathrm{est}_i \geq \underline{\mathrm{est}}_i$).

Finally, let us consider usage of a not-first justification. Let us suppose that $\mathrm{est}_i < \underline{\mathrm{est}}_i$. I.e. we are in a situation when the current explanation is valid after the change (i.e. $\underline{\mathrm{est}}_i \leq \mathrm{est}'_i$), but not before it. Thus some repair of the current explanation is necessary. The justification captures the reason why est_i was increased to est'_i. However in order to make the current explanation valid, it may not be necessary to increase the est_i so much. It is enough to increase est_i to $\underline{\mathrm{est}}_i$. I.e. the justification can be weakened before it is merged into the current explanation. This weakening can be achieved by using $\underline{\mathrm{est}}_i$ instead of est'_i in all conflict windows.

6 Edge-Finding Justifications

Edge-finding is well known filtering algorithm for unary resource constraint. The algorithm is based on the following rules (6), (7) and their symmetric versions [3, 7]. Consider a set $\Omega \subseteq T$ and an activity $i \notin \Omega$. The activity i has to be scheduled after all activities from the set Ω if:

$$\forall \Omega \subset T,\ \forall i \in (T \setminus \Omega): \quad \min\{\mathrm{est}_\Omega,\ \mathrm{est}_i\} + \mathrm{p}_\Omega + \mathrm{p}_i > \mathrm{lct}_\Omega \quad \Rightarrow \quad \Omega \ll i \tag{6}$$

The reason follows: if the activity i is not scheduled after the set Ω then the last activity from the set Ω cannot finish before $\min\{\mathrm{est}_\Omega,\ \mathrm{est}_i\} + \mathrm{p}_\Omega + \mathrm{p}_i$, what is more than the allowed maximum lct_Ω.

Once it is known that the activity i must be scheduled after the set Ω, est_i can be adjusted:

$$\Omega \ll i \quad \Rightarrow \quad \mathrm{est}'_i := \max\{\mathrm{est}_i,\ \mathrm{ECT}_\Omega\} \tag{7}$$

Where ECT_Ω denotes a lower bound of the earliest completion time of a set Ω. ECT_Ω is defined by the following formula:

$$\mathrm{ECT}_\Omega = \max\{\mathrm{est}_{\Omega'} + \mathrm{p}_{\Omega'},\ \Omega' \subseteq \Omega\} \tag{8}$$

There are several implementations of edge-finding algorithm, [4] presents a $\mathcal{O}(n \log n)$ algorithm, another two $\mathcal{O}(n^2)$ algorithms can be found in [7, 8].

We are interested in providing justifications for reductions made by edge-finding. Naturally, a justification consists of the unary resource constraint itself and some set of conflict windows. These conflict windows have to assure, that

while the time windows of activities remain inside the conflict windows, the reduction made according to the rules (6) and (7) would be at least the same.

Lets start with the inequality (6). There are several ways how to extent time windows to conflict windows. Lets look at one of them: we allow an extension only on the right side of the time windows. Thus to satisfy the inequality (6), the conflict windows can be:

$$\begin{aligned} \forall j \in \Omega: &\quad \langle r,\ r + \mathrm{p}_\Omega + \mathrm{p}_i - 1 \rangle \\ i: &\quad \langle r,\ \infty \rangle \end{aligned} \tag{9}$$

where $r = \min\{\mathrm{est}_\Omega,\ \mathrm{est}_i\}$.

Sure, such conflict windows are not sufficient for the rule (7). This rule demands that for one particular set $\Omega' \subseteq \Omega$, the value $\mathrm{est}_{\Omega'}$ remains the same: $\mathrm{est}_{\Omega'} = \mathrm{est}'_i - \mathrm{p}_{\Omega'}$. Putting that together with previous conflict windows (9), the final conflict windows are:

$$\begin{aligned} \forall j \in \Omega': &\quad \langle \mathrm{est}'_i - \mathrm{p}_{\Omega'},\ r + \mathrm{p}_\Omega + \mathrm{p}_i - 1 \rangle \\ \forall j \in (\Omega \setminus \Omega'): &\quad \langle r,\ r + \mathrm{p}_\Omega + \mathrm{p}_i - 1 \rangle \\ i: &\quad \langle r,\ \infty \rangle \end{aligned}$$

These conflict windows are sufficient for both rules (6) and (7).

Enumeration all activities from the set Ω in the explanation would again slow down the justification generation to $\mathcal{O}(n)$. Fortunately, a trick similar to not-first justification can be used here. Let us consider one particular reduction of est_i. Let the set Ω' be such a subset of the set Ω that $\mathrm{ECT}_\Omega = \mathrm{est}_{\Omega'} + \mathrm{p}_{\Omega'}$. Note that the set Ω' must exists thanks to the definition (8) of the ECT_Ω. Further, let the sets Φ and Θ are defined the following way:

$$\begin{aligned} \Phi &= \{j,\ j \in T\ \&\ \mathrm{est}_\Omega \leq \mathrm{est}_j\ \&\ \mathrm{lct}_j \leq \mathrm{lct}_\Omega\} \\ \Theta &= \{j,\ j \in T\ \&\ \mathrm{est}_{\Omega'} \leq \mathrm{est}_j\ \&\ \mathrm{lct}_j \leq \mathrm{lct}_\Omega\} \end{aligned}$$

These sets are in the form of so called *task intervals*. We can use the set Φ in the rule (6) instead of the set Ω and the inequality stays holding. The set Θ can be used to estimate a lower bound of ECT_Φ:

$$\Theta \subseteq \Phi \quad \Rightarrow \quad \mathrm{ECT}_\Phi \geq \mathrm{est}_\Theta + \mathrm{p}_\Theta \quad \Rightarrow \quad \mathrm{est}'_i \geq \max\{\mathrm{est}_i,\ \mathrm{est}_\Theta + \mathrm{p}_\Theta\}$$

Because $\mathrm{est}_\Theta = \mathrm{est}_{\Omega'}$ and $\mathrm{p}_\Theta \geq \mathrm{p}_{\Omega'}$:

$$\mathrm{ECT}_\Omega = \mathrm{est}_{\Omega'} + \mathrm{p}_{\Omega'} \leq \mathrm{est}_\Theta + \mathrm{p}_\Theta$$

Therefore the set Ω can be replaced by the set Φ in the justification and the set Ω' can be replaced by the set Θ. In fact, edge-finding algorithms consider only sets Ω and Ω' in a form of task intervals, i.e. $\Omega = \Phi$ and $\Omega' = \Theta$.

Thus the result is very similar to the not-first justification. Instead of the enumeration of the sets Ω and Ω' in the justification, it is sufficient to record only the values est_Ω, lct_Ω and $\mathrm{est}_{\Omega'}$. Both the sets can be reconstructed from

these values within the time complexity $\mathcal{O}(n)$. The justification has size only $\mathcal{O}(1)$ and can be recorded in time $\mathcal{O}(1)$. Both edge-finding algorithm [7, 8] can be easily modified to generate justifications without changing their time complexity $\mathcal{O}(n^2)$.

Before using the justification, we can weaken it the same way as not-first justification: it is not necessary to increase est_i to est'_i to reach the infeasibility. Sufficient value of est'_i is $\underline{\mathrm{est}}_i$. However this time we must be more careful. Simple replacement of est'_i by $\underline{\mathrm{est}}_i$ in the definition of the conflict windows leads to invalid justifications. Conflict windows for the activities $j \in \Omega'$ should be:

$$\forall j \in \Omega' : \quad \langle \max\{\underline{\mathrm{est}}_i - \mathrm{p}_{\Omega'},\ r\},\ r + \mathrm{p}_\Omega + \mathrm{p}_i - 1 \rangle$$

This way the conflict window cannot run out from the conflict interval (9).

7 Justifications for Detectable Precedences

Detectable precedences is another propagation algorithm which can be used together with the edge-finding and not-first/not-last [10]. Let i and j be two different activities on the same resource. The precedence $j \ll i$ is said to be *detectable*, if the following inequality holds:

$$\mathrm{est}_i + \mathrm{p}_i > \mathrm{lct}_j - \mathrm{p}_j \tag{10}$$

Simply when the previous inequality holds then it is not possible to schedule the activity i before the activity j. The filtering algorithm builds a set Θ of all activities j, which precede the activity i according to detectable precedences:

$$\Theta = \{j,\ j \in T \ \&\ j \ll i \text{ is detectable}\}$$

The activity i cannot start until all of the activities from the set Θ finish, thus est_i can be adjusted:

$$\mathrm{est}'_i := \max\{\mathrm{est}_i,\ \mathrm{ECT}_\Theta\}$$

Let Ω' be a subset of the set Ω such that $\mathrm{ECT}_\Omega = \mathrm{est}_{\Omega'} + \mathrm{p}_{\Omega'}$. The justification has to assure two things: that $\Omega' \ll i$ and that the value $\mathrm{est}_{\Omega'} + \mathrm{p}_{\Omega'}$ does not decrease. Note that activities from the set $\Omega \setminus \Omega'$ are not included in the justification at all.

Lets start with $\mathrm{est}_{\Omega'} + \mathrm{p}_{\Omega'}$. Because this value cannot decrease, $\underline{\mathrm{est}}_j$ must fulfill the following inequality:

$$\forall j \in \Omega' : \quad \underline{\mathrm{est}}_j \geq \mathrm{est}_{\Omega'}$$

Also it has to be assured that $\Omega' \ll i$. For each $j \in \Omega'$ the precedence $j \ll i$ is detectable. And it remains detectable as long as the inequality (10) remains valid:

$$\mathrm{est}_i + \mathrm{p}_i > \mathrm{lct}_j - \mathrm{p}_j$$

To assure that, conflict windows can be set the following way:

$$i: \quad \left\langle \mathrm{est}_i - \left\lceil \frac{\Delta}{2} \right\rceil,\ \infty \right\rangle$$

$$\forall j \in \Omega': \quad \left\langle \mathrm{est}_{\Omega'},\ \mathrm{est}_i + \mathrm{p}_i + \mathrm{p}_j - \left\lceil \frac{\Delta}{2} \right\rceil - 1 \right\rangle$$

$$\text{where } \Delta = \mathrm{est}_i + \mathrm{p}_i - \max\left\{\mathrm{lct}_j - \mathrm{p}_j,\ j \in \Omega'\right\} - 1$$

Again, we do not have to enumerate all activities from the set Ω' in the explanation. The set Ω' can be easily reconstructed using value $\mathrm{est}_{\Omega'}$. The justification has space complexity $\mathcal{O}(1)$ and it can be recorded within time $\mathcal{O}(1)$. Therefore recording of justifications do not change time complexity of the filtering algorithm. Processing of each relevant justification during way back takes time $\mathcal{O}(n)$.

Like other justifications, a justification for the detectable precedences can be weakened before it is used. The idea is still the same: to reach the conflict, it is not necessary to increase est_i to est'_i, value $\underline{\mathrm{est}}_i$ is enough. However this time est'_i does not occur in the conflict window definition directly. But $\mathrm{est}_{\Omega'} = \mathrm{est}'_i - \mathrm{p}'_\Omega$. Hence conflict windows for the activities j from the set Ω' can be enlarged the following way:

$$\forall j \in \Omega': \quad \left\langle \underline{\mathrm{est}}_i - \mathrm{p}_{\Omega'},\ \mathrm{est}_i + \mathrm{p}_i + \mathrm{p}_j - \left\lceil \frac{\Delta}{2} \right\rceil - 1 \right\rangle$$

8 Experimental Results

The ideas presented in this paper were implemented in a C++ jobshop solver. Several jobshop problems of sizes 10x10 to 15x15 from the OR library [1] were used as a benchmark problems. The task is to find and prove the minimal makespan. Problems were solved using backtracking with directed backjumping based on explanations.

In order to make number of backtracks small, initial upper bound was set to the known optimal makespan. The solver has to find a solution first and then prove that there is no better solution.

The experiments shows that computation of explanation is very fast, it takes only 3–5% of the CPU time. The reason follows: propagation algorithms find a reduction only in 4–31% of runs (exact ratio depends on the filtering algorithm and an order in which the algorithms are called). From the recorded justifications, only 25–80% are really used (again, the ratio depends on the filtering algorithm).

The problems were solved twice. First time using explanations, second time without it. Tables 2 and 1 show the results. Columns CH1 and CH2 are number of choicepoints (i.e. nodes of a search tree without leaves), columns T1 and T2 shows the computation time.

Problem	Size	Makespan	CH1	CH2	CH Saving	T1	T2	T Saving
ft10	10x10	930	11478	14192	19.12%	13.112s	16.068s	18.40%
abz5	10x10	1234	5294	6445	17.86%	5.358s	6.538s	18.05%
abz6	10x10	943	3033	3807	20.33%	3.025s	4.077s	25.80%
la16	10x10	945	119	151	21.19%	0.147s	0.189s	22.22%
la17	10x10	784	26	27	3.70%	0.045s	0.045s	0.00%
la18	10x10	848	2429	2641	8.00%	2.422s	2.697s	10.20%
la19	10x10	842	14107	14989	5.89%	14.910s	16.058s	7.15%
la20	10x10	902	2623	2912	9.92%	2.812s	3.220s	12.67%
la36	15x15	1268	962	33749	97.15%	2.556s	40.214s	93.64%
la37	15x15	1397	61	61	0.00%	0.158s	0.157s	-0.64%
orb01	10x10	1059	16268	17137	5.07%	19.808s	20.927s	5.35%
orb02	10x10	888	10937	13818	20.85%	11.987s	15.348s	21.90%
orb03	10x10	1005	44152	50820	13.12%	48.051s	55.384s	13.24%
orb04	10x10	1005	1220	1319	7.50%	1.525s	1.618s	5.75%
orb05	10x10	887	2587	3312	21.89%	2.717s	3.587s	24.26%
orb06	10x10	1010	9838	10397	5.38%	11.709s	12.337s	5.09%
orb07	10x10	397	16476	20745	20.58%	16.251s	21.231s	23.46%
orb08	10x10	899	15	15	0.00%	0.039s	0.037s	-5.41%
orb09	10x10	934	491	515	4.66%	0.615s	0.641s	4.06%

Table 1. Jobshop instances: first branching strategy

Note that not all explanation computation was excluded in the second run. However, as said before, no more than 3–5% time could be saved by that.

Two different branching schemes were used to show the influence of the branching strategy to the directed backjumping:

1. The first branching strategy finds a resource with a smallest slack time. Then all yet unscheduled activities on the resource are taken and branching is done on a decision which of them will be the first. For results, see table 1.
2. The second branching strategy is taken from [11]. The resource with the relatively smallest slack is taken and branching is done by ordering two longest unordered activities on this resource. The results are in the table 2.

As can be seen, in case of quickly solvable instances ($\sim$ 1000 choicepoints) backjumping does not significantly improve the performance. However for harder instances, the savings of time and choicepoints reach 20% for the first branching strategy and 40% for the second branching strategy. The problem la36 in table 1 is quite exceptional: 97.15% of choicepoints are eliminated.

9 Conclusions and Further Work

Experimental results shows that explanations can significantly speed up the search, especially for hard problems. Also lazy computation of explanations seams to be quite effective.

In the future work we would like to explore more advanced search methods: dynamic backtracking, MAC-DBT or decision-repair.

Another technique often used to prune a search space is *shaving* [7]. For scheduling problems, shaving turned out to be quite effective. It could be interesting to combine explanations with shaving.

Problem	Size	Makespan	CH1	CH2	CH Saving	T1	T2	T Saving
ft10	10 x 10	930	5931	6246	5.05%	4.928s	5.255s	6.23%
abz5	10 x 10	1234	2188	2963	26.16%	1.500s	2.093s	28.34%
abz6	10 x 10	943	840	863	2.67%	0.618s	0.659s	6.23%
la16	10 x 10	945	1025	1231	16.74%	0.604s	0.786s	23.16%
la17	10 x 10	784	45	45	0%	0.037s	0.037s	0%
la18	10 x 10	848	828	838	1.20%	0.606s	0.626s	3.20%
la19	10 x 10	842	5088	5447	6.60%	3.783s	4.085s	7.40%
la20	10 x 10	902	1353	1369	1.17%	1.076s	1.096s	1.83%
la36	15 x 15	1268	2636	2890	8.79%	5.477s	6.123s	10.56%
la37	15 x 15	1397	2554	6398	60.09%	2.869s	6.763s	57.58%
la39	15 x 15	1233	251	276	9.06%	0.554s	0.597s	7.21%
la40	15 x 15	1222	23606	26408	10.62%	49.816s	56.019s	11.08%
orb01	10 x 10	1059	5214	5220	0.12%	4.903s	4.931s	0.57%
orb02	10 x 10	888	3200	3448	7.20%	2.339s	2.620s	10.73%
orb03	10 x 10	1005	12603	12699	0.76%	10.214s	10.422s	2.00%
orb04	10 x 10	1005	1938	1969	1.58%	1.584s	1.658s	4.47%
orb05	10 x 10	887	1625	1771	8.25%	1.134s	1.248s	9.14%
orb06	10 x 10	1010	6145	6618	7.15%	4.493s	4.867s	7.69%
orb07	10 x 10	397	2066	2190	5.67%	1.552s	1.673s	7.24%
orb08	10 x 10	899	45	45	0%	0.040s	0.041s	2.44%
orb09	10 x 10	934	532	535	0.57%	0.443s	0.451s	1.78%
orb10	10 x 10	944	146	146	0%	0.157s	0.160s	1.88%
ta04	15 x 15	1175	115525	185278	37.65%	2m 33s	4m 12s	39.42%
ta07	15 x 15	1227	763719	1290715	40.83%	20m 10s	35m 47s	43.67%
la38	15 x 15	1196	1381989	1596715	13.45%	37m 57s	45m 34s	16.71%

Table 2. Jobshop instances: second branching strategy

References

[1] OR library. URL http://mscmga.ms.ic.ac.uk/info.html.

[2] Narendra Jussien Abdallah Elkhyari, Chrstelle Guéret. Conflict-based repair techniques for solving dynamic scheduling problems. In *Principles and Prictice of Constraint Programming (CP 2002)*, pages 702–707, Ithaca, USA, 2002. Springer-Verlag.

[3] Philippe Baptiste and Claude Le Pape. Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling. In *Proceedings of the Fifteenth Workshop of the U.K. Planning Special Interest Group*, 1996.

[4] Jacques Carlier and Eric Pinson. Adjustments of head and tails for the job-shop problem. *European Journal of Operational Research*, 78:146–161, 1994.

[5] Matthew L. Ginsberg, James M. Crawford, and David W. Etherington. Dynamic backtracking, 1996. URL http://citeseer.ist.psu.edu/ginsberg96dynamic.html.

[6] Christelle Guéret, Narendra Jussien, and Christian Prins. Using intelligent backtracking to improve branch and bound methods: an application to open-shop problems. *European Journal of Operational Research*, 127(2): 344–354, 2000. ISSN 0377-2217. URL http://www.emn.fr/jussien/publications/gueret-EJOR00.pdf.

[7] Paul Martin and David B. Shmoys. A new approach to computing optimal schedules for the job-shop scheduling problem. In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Proceedings of the 5th International Conference on Integer Programming and Combinatorial Opti-*

mization, IPCO'96, pages 389–403, Vancouver, British Columbia, Canada, 1996.

[8] Claude Le Pape Philippe Baptiste and Wim Nuijten. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. Kluwer Academic Publishers, 2001.

[9] Philippe Torres and Pierre Lopez. On not-first/not-last conditions in disjunctive scheduling. *European Journal of Operational Research*, 1999.

[10] Petr Vilím. $O(n \log n)$ filtering algorithms for unary resource constraint. In *Proceedings of CP-AI-OR 2004*. Springer-Verlag, 2004.

[11] Armin Wolf. Better propagation for non-preemptive single-resource constraint problems. In *Proceedings of the ERCIM/CoLogNet workshop 2004*, 2004.