Towards Smaller Explanations for Detectable Precedences in the Disjunctive Constraint

Andrei Ioniță
Delft University of Technology, Netherlands

Imko Marijnissen
Delft University of Technology, Netherlands

Emir Demirović
Delft University of Technology, Netherlands

Abstract

The Disjunctive constraint is a widely used scheduling constraint that requires that no two tasks overlap in time. Although propagation for the Disjunctive has been thoroughly studied in traditional constraint programming, comparatively fewer studies have considered it in the context of lazy clause generation. In this paper, we consider two propagators for the Disjunctive, detectable precedences and edge finding, and study their combination in a lazy clause generation solver. We study explanation generation for the timeline-based variant of detectable precedences and present a method for constructing explanations that are minimal with respect to the number of tasks. We show that such minimal explanations are not necessarily preferable to existing explanation construction methods in practice. Finally, we discuss explanations for the combined propagation of detectable precedences and edge finding.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases constraint programming, lazy clause generation, disjunctive constraint, scheduling, detectable precedences, edge finding, explanations

1 Introduction

Scheduling problems involve a set of tasks that need to be scheduled while requiring certain resources. Such problems are relevant in many different fields such as industrial scheduling or transportation. A common solving methodology for scheduling problems is constraint programming, due to its ability to quickly remove infeasible schedules from the solution space.

The case where all tasks share a single resource, thus not being able to run concurrently, is usually modeled using the Disjunctive constraint. Multiple filtering methods have been specifically devised to prune the search space for this constraint, or adopted from other constraints, as is the case for Edge Finding [1], which was originally developed for the Cumulative constraint.

Algorithms for most of the existing filtering methods were introduced by Petr Vilím in his PhD thesis [10]. Vilím also introduced the Detectable Precedences rule [8], which is the main rule studied in this paper. In his PhD thesis [10], Vilím introduced an $\mathcal{O}(n \log n)$ algorithm for the Detectable Precedences rule. Fahimi et al. [3] later improved this to an $\mathcal{O}(n)$ algorithm. The latter authors also recommended to extend the algorithm by supporting explanations, which would allow the usage of lazy clause generation (LCG) solvers.

Although the Disjunctive constraint has received significant attention in traditional constraint programming, its use in lazy clause generation has received much less attention. Explanation generation for Edge Finding has been studied in several works [5, 7], partly because Edge Finding is also commonly used for the Cumulative constraint. By contrast, explanation generation for Detectable Precedences has received comparatively less attention, with Vilím's 2005 paper [9] providing one of the few existing treatments of this idea.

A related limitation is that explanation-generation methods for these filtering rules have mostly been studied separately. This contrasts with the filtering side, where Vilím reports obtaining the best results by combining several filtering rules [10]. However, the most efficient known implementations of Edge Finding and Detectable Precedences rely on different data structures, which makes such a combination non-trivial. As a result, existing explanations do not exploit the interaction between the two rules.

This paper addresses these two issues. First, we study whether explanations for Detectable Precedences can be reduced in size. Second, we make a first step towards combining Detectable Precedences and Edge Finding for the DISJUNCTIVE constraint in an LCG solver. This second direction is work in progress: in this paper, we report preliminary results on the interaction between the two rules and discuss the challenges involved in generating explanations for the combined propagation.

The paper is organized as follows. Section 2 discusses existing work and positions our work within existing literature. Section 3 reviews the DISJUNCTIVE constraint and two filtering methods: Detectable Precedences and Edge Finding. Section 4 contains the main contributions of this work. Section 5 presents the experimental results, and Section 6 concludes the paper and discusses future work.

2 Related Work

Several filtering rules have been developed for unary-resource scheduling. Edge Finding [1] reasons about tasks that must be placed before or after a set of other tasks, while Detectable Precedences [8] derives mandatory precedences from the current bounds of the tasks. Vilím proposed efficient algorithms for several such rules, including Edge Finding and Detectable Precedences, and gave an $\mathcal{O}(n \log n)$ algorithm for the latter [10]. Fahimi et al. later improved Detectable Precedences to a linear-time timeline-based algorithm [3].

Using these propagators in lazy clause generation requires explanations for each propagation. Vilím [9] proposed conflict windows to explain unary-resource propagation, but these do not directly correspond to the clausal explanations used by modern LCG solvers. More recently, van Vliet adapted this idea to explain Detectable Precedences in an LCG setting, using the last cluster computed by the timeline data structure as the explanation set, i.e., the last group of tasks in the timeline which are responsible for the propagation [6]. For a comprehensive description of the Last Cluster method, we refer the reader to Section 4.3 of [6].

Our work builds on this explanation method. Rather than only generating a valid explanation for Detectable Precedences, we study whether the last cluster can be replaced by a smaller set that preserves the same propagation strength. We also study the interaction between Detectable Precedences and Edge Finding when both are used for the DISJUNCTIVE.

3 Preliminaries

3.1 The Disjunctive constraint

The DISJUNCTIVE constraint is a scheduling constraint over a set of tasks $\mathcal{I}$. Each task $i \in \mathcal{I}$ has a start-time variable S_i and a fixed processing time p_i. The constraint enforces that no two tasks overlap in time.

For a task i, we denote by est_i and lst_i its earliest and latest possible start times, respectively. Similarly, $ect_i = est_i + p_i$ denotes its earliest completion time, and $lct_i = lst_i + p_i$

denotes its latest completion time. We can extend these definitions for sets of tasks $\Omega \subseteq \mathcal{I}$:

$$p_\Omega = \sum_{i \in \Omega} p_i \qquad est_\Omega = \min_{i \in \Omega} est_i \qquad lct_\Omega = \max_{i \in \Omega} lct_i$$

Similarly, lower bounds on the earliest completion time (ect) and latest starting time (lst) of a set can be obtained by scheduling tasks with preemption:

$$ect_\Omega = \max_{\Omega' \subseteq \Omega} (est_{\Omega'} + p_{\Omega'}) \qquad lst_\Omega = \min_{\Omega' \subseteq \Omega} (lct_{\Omega'} - p_{\Omega'})$$

Intuitively, ect_Ω is the earliest time by which all tasks in Ω can be completed, given their current earliest start times.

3.2 Filtering rules

Filtering rules strengthen the domains of the start-time variables by removing infeasible solutions. In this work, we focus on two filtering rules for the Disjunctive constraint: Edge Finding [1] and Detectable Precedences [8]. Both rules reason about sets of tasks that must be scheduled before another task, and derive stronger lower bounds on start times.

3.2.1 Edge Finding

Edge Finding detects whether a task must be scheduled after a set of other tasks. Given a task i and a set of tasks $\Omega \subseteq \mathcal{I} \setminus \{i\}$, if the tasks in $\Omega \cup \{i\}$ cannot all complete before the latest completion time of Ω, then i cannot be scheduled before all tasks in Ω. This can be expressed as:

$$ect_{\Omega \cup \{i\}} > lct_\Omega \quad \Rightarrow \quad \Omega \ll i.$$

Once this ordering is detected, the lower bound of S_i can be updated to $est_i \geq ect_\Omega$. Thus, Edge Finding identifies tasks that are forced to be placed after a given set of tasks. There exist multiple approaches implementing this rule, but for the Disjunctive constraint specifically, we are discussing Vilím's proposed algorithm that runs in $\mathcal{O}(n \log n)$ time and is based on the Θ-Λ tree [10].

3.2.2 Detectable Precedences

Detectable Precedences also derives implied precedences between tasks. For two distinct tasks i and j, the precedence $j \ll i$ is *detectable* when task i cannot be scheduled before task j, i.e. $ect_i > lst_j$.

For a task i, let $\Omega = \{\, j \in \mathcal{I} \mid ect_i > lst_j \ \wedge \ j \neq i \,\}$ denote the set of all detectable predecessors of i. Since all tasks in Ω must precede i, the earliest start time of i can be delayed up to the earliest completion time of this set:

$$est_i := \max\{est_i,\ ect_\Omega\}.$$

In this paper, we use the timeline-based algorithm for Detectable Precedences by Fahimi et al. [3]. This algorithm computes the relevant set Ω using a timeline data structure and runs in linear time. The timeline is an improvement on the previous fastest data structure used for Detectable Precedences, namely the Θ-Λ tree as introduced by Vilím [10].

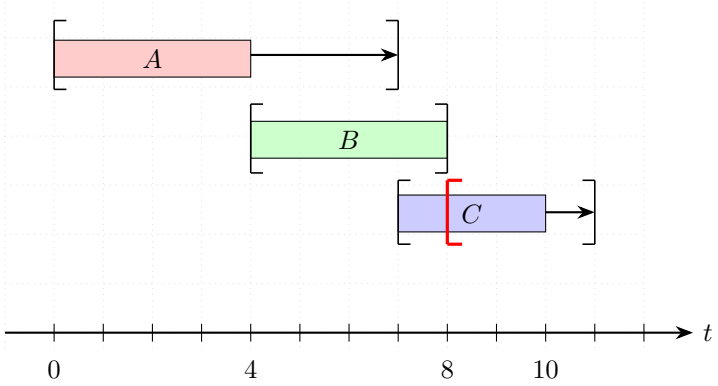

Figure 1 Tasks with intervals $A = [0, 7]$, $B = [4, 8]$, and $C = [7, 11]$.

3.3 Explanations

In lazy clause generation [4], every propagation must be justified by an explanation. An explanation is a set of predicates over the current domains that is sufficient to imply the propagated bound. For example, if a propagator derives $S_i \geq b$, then it must also provide a reason showing under which current bounds this propagation is valid. These explanations are then used during conflict analysis to derive learned nogoods.

The generality of explanations is one of the most important factors behind the efficiency of lazy clause generation. More general explanations can lead to more general learned nogoods, which are more likely to be reused later in the search. Since larger explanations depend on more predicates, they may produce learned nogoods that are too specific to the current conflict to be reused later. For this reason, reducing the size of explanations is a natural objective in lazy clause generation.

4 Methodology

In this section, we introduce a method to minimize the explanations produced by the timeline-based algorithm for the detectable precedences rule, in terms of the number of tasks in the explanation.

4.1 Minimum-Cardinality Suffix Explanations

The explanation for Detectable Precedences introduced in [6] contains the last cluster of tasks in the timeline data structure. However, this set of tasks is not guaranteed to be of a minimum cardinality among the sets that can justify the same propagation. In the context of lazy clause generation, removing unnecessary tasks from explanations can produce shorter explanations and potentially more general learned nogoods.

Figure 1 shows a simple example with three tasks. In this example, based on the Detectable Precedences rule, task C needs to be scheduled after both A and B. Hence, the lower bound of its start time can be propagated to $ect_{\{A,B\}} = 8$ (marked by the red bracket). The standard explanation would contain the set $\Omega = \{A, B\}$, which is the last (and only) cluster produced by the algorithm. However, task B alone already has earliest completion time $ect_B = est_B + p_B = 8$. Therefore, in this case, the suffix $\{B\}$ gives a shorter explanation than the full last cluster $\{A, B\}$.

In general, let Ω be the last cluster produced by the algorithm based on the timeline data structure when scheduling a task i. We are searching for the minimum-cardinality subset $S \subseteq \Omega$ such that $ect_S = ect_\Omega$. If this condition is satisfied, then S can also be used as the explanation set to explain the propagation $est_i \geq ect_\Omega$. However, there are exponentially many subsets to consider for a given Ω.

We fix an arbitrary tie-breaking order for tasks with equal earliest start time. All suffixes below are defined with respect to this fixed ordering.

▶ **Lemma 1.** *Let Ω be the last cluster of tasks, and let $X = \langle j_1, \ldots, j_m \rangle$ be the tasks of Ω sorted by nondecreasing earliest start time (est). It is sufficient to consider suffixes of X (X included) to find a minimum-cardinality subset preserving ect_Ω.*

Proof. Let $S \subseteq \Omega$ be a minimum-cardinality subset such that $ect_S = ect_\Omega$. Since S is minimum-cardinality, no proper subset of S can have earliest completion time equal to ect_Ω. Therefore, the maximum defining ect_S is attained by S itself, i.e., $ect_S = est_S + p_S$.

Let j_k be the first task in X that belongs to S. Then j_k has minimum *est* among tasks in S, and every task in S appears at position at least k in X. Therefore, $S \subseteq X_k$, where $X_k = \{j_k, \ldots, j_m\}$ is the suffix starting at j_k.

Suppose that $S \neq X_k$. Then there exists a task $j_\ell \in X_k \setminus S$. Since $est_{j_\ell} \geq est_{j_k}$, adding j_ℓ to S does not decrease the minimum start time of the set, while it increases the total processing time. Hence, $ect_{S \cup \{j_\ell\}} > ect_S = ect_\Omega$.

Since $S \cup \{j_\ell\} \subseteq \Omega$, this contradicts the definition of ect_Ω as the maximum value of $est_{\Omega'} + p_{\Omega'}$ over all subsets $\Omega' \subseteq \Omega$. Thus $S = X_k$, and therefore S is a suffix of X. ◀

Using Lemma 1, after obtaining the last cluster Ω, we sort its tasks by nondecreasing earliest start time and select a minimum-cardinality suffix $S \subseteq \Omega$ such that $ect_S = ect_\Omega$. Using the same lifting as in [6], we can generate the following explanation for the propagation $est_i \geq ect_S$:

$$S_i \geq lst_S - p_i + 1 \bigwedge_{j \in S} (S_j \geq est_S \ \wedge \ S_j \leq lst_S)\,. \tag{1}$$

Since $S \subseteq \Omega$, the resulting explanation contains at most as many task bounds as the original explanation.

Given a task i, the timeline data structure $\mathcal{I}$, and the root bounds of the list of tasks, Algorithm 1 shows how the minimal set of tasks that can explain the same propagation can be obtained. Lines 3–8 build buckets of tasks with the same earliest start time using the indices already stored in the timeline data structure. Lines 9–21 then scan these buckets in reverse order (thus iterating through *est*s in decreasing order) and select the first suffix whose earliest completion time is equal to ect_Ω. Since the scan starts from the largest earliest start time, the first such suffix is the minimum-cardinality suffix preserving ect_Ω.

This implementation avoids explicitly sorting the tasks in Ω. Instead, the timeline data structure already provides the ordering information through the t and m parameters. Building the buckets takes time linear in $|\Omega|$, and the reverse scan over the buckets is linear in the size of the timeline data structure. Therefore, selecting the minimum-cardinality suffix does not add a logarithmic sorting factor. The explanation generation remains linear in the number of tasks considered, matching the asymptotic complexity of the original explanation method.

5 Experiments

This section describes the experiments conducted. First, we compared the Last Cluster [6] method against our Minimum-Cardinality Suffix method. We observe that the explanations

Algorithm 1 GenerateMinimumCardinalitySuffix($i, \mathcal{I}, root$)

```
Ω ← timeline.GetOmega()
ect_Ω ← timeline.EarliestCompletionTime()
buckets ← empty buckets for each index in I.t
for j ∈ Ω do
    b ← I.m[j]
    buckets[b].tasks ← buckets[b].tasks ∪ {j}
    buckets[b].duration ← buckets[b].duration + p_j
end for
suffix_duration ← 0
S ← Ω
for b ← |buckets| down to 1 do
    if buckets[b].tasks = ∅ then continue
    end if
    suffix_duration ← suffix_duration + buckets[b].duration
    est_suffix ← I.t[b]
    ect_suffix ← est_suffix + suffix_duration
    if ect_suffix = ect_Ω then
        S ← ⋃_{b'≥b} buckets[b'].tasks
        break
    end if
end for
return S
```

produced by our method, despite being smaller in terms of the number of tasks, are not necessarily more general, hence they do not necessarily improve the number of conflicts or solving time. Second, we study the effect of combining Detectable Precedences with Edge Finding, and find that running Edge Finding first, followed by Detectable Precedences is more efficient in terms of most metrics than running them in the opposite order.

We implemented all algorithms in the Pumpkin solver [2]. The experiments were conducted on 34 job-shop scheduling instances selected from the MiniZinc benchmark repository.[1] All experiments were run on a machine with an 11th Gen Intel(R) Core(TM) i5-11400H 2.70GHz processor. We used a fixed search strategy with a timeout of 5 minutes for all instances. All instances were solved by all configurations without reaching the designated timeout.

We evaluate the effect of the alternative explanation strategy indirectly through solver-level metrics: solving time, number of conflicts, average learned nogood length, and average Literal Block Distance (LBD). These metrics do not measure the size of the generated propagation explanations directly; they capture whether the alternative explanations lead to smaller or more useful learned nogoods and improved search behaviour.

5.1 Comparison of Explanation Methods for Detectable Precedences

Table 1 shows the average metric values over all instances under both explanation methods. We observe that, interestingly, despite the fact that Minimum-Cardinality Suffix constructs smaller explanations by design, this does not translate to a better solving performance.

[1] https://github.com/MiniZinc/minizinc-benchmarks/tree/master/jobshop

Table 1 Average metrics for Last Cluster and Minimum-Cardinality Suffix explanations.

Metric	Last Cluster	Minimum-Cardinality Suffix
Time (s)	8.85	12.09
LBD	9.98	9.79
Conflicts	6219.32	6971.71
Nogood length	23.09	23.91

Last Cluster is on average $\approx 26.8\%$ faster than Minimum-Cardinality Suffix, and produces approximately 10.8% fewer conflicts. The average length of learned nogoods is close to equal, with a very small advantage to Last Cluster.

The only observed advantage of Minimum-Cardinality Suffix is in terms of the LBD, which is a modest 1.8% smaller than its counterpart. This suggests that the smaller explanations may sometimes lead to learned nogoods that are slightly better according to this metric. However, this improvement is not reflected in the overall solving time or number of conflicts. Figure 2 provides a more detailed instance-wise comparison, showing that neither method has a consistent advantage across all instances in terms of solving time or LBD.

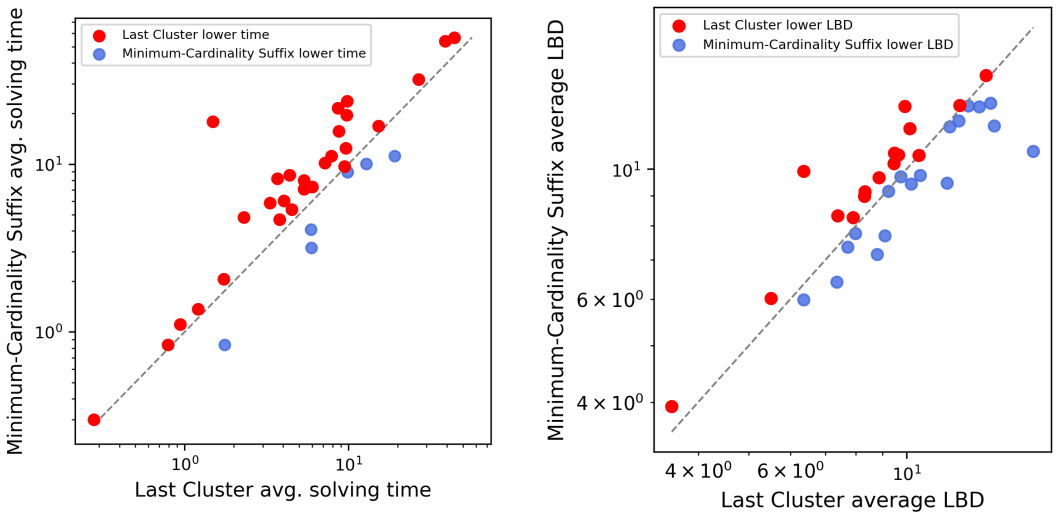

Figure 2 Per-instance comparison between the Last Cluster and Minimum-Cardinality Suffix explanation methods. Each point represents one instance. Points below the diagonal indicate instances where the Minimum-Cardinality Suffix method obtains a lower value, while points above the diagonal indicate instances where Last Cluster obtains a lower value.

5.2 Edge Finding and Detectable Precedences Combination

This behaviour might be partly explained by the fact that fewer tasks do not necessarily produce a more general explanation. For example, consider the set of tasks in Figure 1. Detectable Precedences is able to propagate $est_C \geq 8$. The explanation for this propagation produced by the Last Cluster method is $est_A \geq 0 \wedge est_A \leq 4 \wedge est_B \geq 0 \wedge est_B \leq 4 \wedge est_C \geq 2$. On the other hand, the explanation produced by the Minimum-Cardinality Suffix method is $est_B \geq 4 \wedge est_B \leq 4 \wedge est_C \geq 2$. Despite requiring one less task, the second explanation is not more general than the first due to the bounds of task B being stronger. To be more

Table 2 Average metrics for the two propagation orders.

Metric	DP → EF	EF → DP
Time (s)	16.37	10.20
LBD	9.31	9.69
Conflicts	5133.42	4628.23
Nogood length	24.18	25.46

specific, the two explanations do not subsume each other: Last Cluster contains an additional task, while Minimum-Cardinality relies on a tighter bound. This is due to the interaction with the lifting procedure described in [6]. This helps explain why the smaller explanations constructed by Minimum-Cardinality Suffix do not lead to a clear improvement in the results.

A natural next step is to study whether Edge Finding and Detectable Precedences can be combined into a single propagator for the Disjunctive constraint. However, such a combination is not straightforward, because the two most efficient implementations rely on different data structures: Edge Finding is typically implemented using a Θ-Λ tree, whereas the linear-time Detectable Precedences algorithm uses a timeline data structure. In this paper, we study a simpler form of combination in which both propagators are posted for the same constraint and are executed in different orders.

We consider two configurations: Edge Finding followed by Detectable Precedences, and Detectable Precedences followed by Edge Finding. This allows us to evaluate whether the order in which the two propagators are applied has an effect on the resulting search behaviour.

Table 2 compares the two propagation orders. Running Edge Finding before Detectable Precedences gives better average solving performance: it is approximately 37.7% faster and produces approximately 9.8% fewer conflicts. However, Detectable Precedences followed by Edge Finding obtains slightly better learned nogood metrics, with approximately 3.9% lower average LBD and 5.0% shorter learned nogoods. This suggests that applying Edge Finding first is more effective for search overall, even though the nogoods learned in the opposite order appear slightly better according to these aggregate metrics. A possible reason is that the pruning performed by Edge Finding exposes tighter bounds before Detectable Precedences is executed.

6 Conclusion

We studied explanation generation for the timeline-based Detectable Precedences algorithm in the Disjunctive constraint. We introduced Minimum-Cardinality Suffix, a method that replaces the last cluster used in existing explanations by the smallest suffix preserving the same earliest completion time. Although this produces explanations with fewer tasks by construction, the experimental results show that this does not necessarily improve solver performance. This is partly because fewer tasks do not yield more general explanations once lifting is also performed. A direct analysis of the sizes of the generated explanations would provide a more detailed evaluation, but we leave this for future work.

We also performed a preliminary study on the interaction between Detectable Precedences and Edge Finding by running propagators for both rules in both possible orders. The results suggest that applying Edge Finding first is generally more effective. Future work can include adapting the implementations of the rules so they can share more information, or developing a fully integrated propagator for the two rules, with explanations that exploit this interaction.

References

1. Jacques Carlier and Eric Pinson. Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78(2):146–161, 1994.
2. Emir Demirović, Maarten Flippo, Imko Marijnissen, Konstantin Sidorov, and Smits Jeff. Pumpkin: A lazy clause generation constraint solver in rust, 2024. *URL: https://github.com/ConSol-Lab/Pumpkin*.
3. Hamed Fahimi, Yanick Ouellet, and Claude-Guy Quimper. Linear-time filtering algorithms for the disjunctive constraint and a quadratic filtering algorithm for the cumulative not-first not-last: Linear-time filtering algorithms for the disjunctive constraint. *Constraints*, 23(3):272–293, 2018.
4. Olga Ohrimenko, Peter J Stuckey, and Michael Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, 2009.
5. Andreas Schutt, Thibaut Feydy, and Peter J Stuckey. Explaining time-table-edge-finding propagation for the cumulative resource constraint. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 234–250. Springer, 2013.
6. Matthias van Vliet. *Explaining detectable precedences for the disjunctive constraint*. PhD thesis, Delft University of Technology, 2025.
7. Radu Andrei Vasile. Evaluating the impact of explanations on the performance of an edge-finding propagator.
8. Petr Vilím. Batch processing with sequence dependent setup times: New results. In *Proceedings of the 4th Workshop of Constraint Programming for Decision and Control, CPDC'02*, 2002.
9. Petr Vilím. Computing explanations for the unary resource constraint. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 396–409. Springer, 2005.
10. Petr Vilím. Global constraints in scheduling. 2007.