Efficient Context-Free Grammar Constraints*

Serdar Kadioglu and Meinolf Sellmann
Brown University, PO Box 1910, Providence, RI 02912
{serdark,sello}@cs.brown.edu

Abstract

With the introduction of constraints based on finite automata a new line of research has opened where constraints are based on formal languages. Recently, constraints based on grammars higher up in the Chomsky hierarchy were introduced. We devise a time- *and* space-efficient incremental arc-consistency algorithm for context-free grammars. Particularly, we show how to filter a sequence of monotonically tightening problems in cubic time and quadratic space. Experiments on a scheduling problem show orders of magnitude improvements in time and space consumption.

Introduction

A major strength of constraint programming is its ability to provide the user with high-level constraints that capture and exploit problem structure. However, this expressiveness comes at the price that the user must be aware of the constraints that are supported by a solver. One way to overcome this problem is by providing highly expressive global constraints that can be used to model a wide variety of problems and that are associated with efficient filtering algorithms. As was found in (Pesant 2003; Beldiceanu, Carlsson & Petit 2004), a promising avenue in this direction is the introduction of constraints that are based on formal languages, which enjoy a wide range of applicability while allowing the user to focus on the desired properties of solutions rather than having to deal for herself with the problem of constraint filtering.

The first constraints in this regard were based on automata (Pesant 2003; Beldiceanu, Carlsson & Petit 2004). Especially, incremental implementations of the regular membership constraint have been shown to perform very successfully on various problems and even when used to replace custom constraints for special structures which can be expressed as regular languages. In (Quimper & Walsh 2006; Sellmann 2006), algorithms were devised which perform filtering for context-free grammar constraints in polynomial time. Now, we focus on practical aspects when dealing with context-free grammars. We give an incremental algorithm which combines low memory requirements with very efficient incremental behavior. Tests on a real-world shift-scheduling problem prove the practical importance of grammar constraints and show massive speed-ups achieved by the new algorithm. Finally, we show how context-free grammar constraints can efficiently be conjoined with linear constraints to perform cost-based filtering.

*This work was supported by the National Science Foundation through the Career: Cornflower Project (award number 0644113).

Basic Concepts

We start by reviewing some well-known definitions from the theory of formal languages and the existing algorithm for filtering context-free grammar constraints. For a full introduction, we refer the interested reader to (Hopcroft & Ullman 1979) and (Sellmann 2006). All proofs that are omitted in this paper can also be found there.

Definition 1 (Alphabet and Words)**.** *Given sets Z, Z_1, and Z_2, with Z_1Z_2 we denote the set of all* sequences *or* strings *$z = z_1z_2$ with $z_1 \in Z_1$ and $z_2 \in Z_2$, and we call Z_1Z_2 the* concatenation *of Z_1 and Z_2. Then, for all $n \in \mathbb{N}$ we denote with Z^n the set of all sequences $z = z_1z_2 \dots z_n$ with $z_i \in Z$ for all $1 \leq i \leq n$. We call z a* word *of length n, and Z is called an* alphabet *or set of* letters*. The empty word has length 0 and is denoted by ϵ. It is the only member of Z^0. We denote the set of all words over the alphabet Z by $Z^* := \bigcup_{n\in\mathbb{N}} Z^n$. In case that we wish to exclude the empty word, we write $Z^+ := \bigcup_{n\geq 1} Z^n$.*

Definition 2 (Context-Free Grammars)**.** *A* grammar *is a tuple $G = (\Sigma, N, P, S_0)$ where Σ is the alphabet, N is a finite set of* non-terminals*, $P \subseteq (N\cup\Sigma)^*N(N\cup\Sigma)^* \times (N\cup\Sigma)^*$ is the set of* productions*, and $S_0 \in N$ is the start non-terminal. We will always assume that $N \cap \Sigma = \emptyset$. Given a grammar $G = (\Sigma, N, P, S_0)$ such that $P \subseteq N \times (N \cup \Sigma)^*$, we say that the grammar G and the language L_G are* context-free*. A context-free grammar $G = (\Sigma, N, P, S_0)$ is said to be in* Chomsky Normal Form (CNF) *if and only if for all productions $(A \to \alpha) \in P$ we have that $\alpha \in \Sigma^1 \cup N^2$. Without loss of generality, we will then assume that each literal $a \in \Sigma$ is associated with exactly one unique non-literal $A_a \in N$ such that $(B \to a) \in P$ implies that $B = A_a$ and $(A_a \to b) \in P$ implies that $a = b$.*

Remark 1. *Throughout the paper, we will use the following convention: Capital letters A, B, C, D, and E denote non-terminals, lower case letters a, b, c, d, and e denote letters in Σ, Y and Z denote symbols that can either be letters or*

Algorithm 1 CFGC Filtering Algorithm

1. We run the dynamic program based on the CYK recursion equation with initial sets $S_{i1} := \{A \mid (A \to v) \in P, v \in D_i\}$.
2. We define the directed graph $Q = (V, E)$ with node set $V := \{v_{ijA} \mid A \in S_{ij}\}$ and arc set $E := E_1 \cup E_2$ with $E_1 := \{(v_{ijA}, v_{ikB}) \mid \exists\, C \in S_{i+k,j-k} : (A \to BC) \in P\}$ and $E_2 := \{(v_{ijA}, v_{i+k,j-k,C}) \mid \exists\, B \in S_{ik} : (A \to BC) \in P\}$ (see Figure 1).
3. Now, we remove all nodes and arcs from Q that cannot be reached from v_{1nS_0} and denote the resulting graph by $Q' := (V', E')$.
4. We set $S'_{ij} := \{A \mid v_{ijA} \in V'\} \subseteq S_{ij}$, and $D'_i := \{v \mid \exists\, A \in S'_{i1} : (A \to v) \in P\}$.

non-terminals, u, v, w, x, y, and z denote strings of letters, and α, β, and γ denote strings of letters and non-terminals. Moreover, productions (α, β) in P can also be written as $\alpha \to \beta$.

Definition 3 (Derivation and Language)**.** • *Given a grammar $G = (\Sigma, N, P, S_0)$, we write $\alpha\beta_1\gamma \underset{G}{\Rightarrow} \alpha\beta_2\gamma$ if and only if there exists a production $(\beta_1 \to \beta_2) \in P$. We write $\alpha_1 \overset{*}{\underset{G}{\Rightarrow}} \alpha_m$ if and only if there exists a sequence of strings $\alpha_2, \ldots, \alpha_{m-1}$ such that $\alpha_i \underset{G}{\Rightarrow} \alpha_{i+1}$ for all $1 \le i < m$. Then, we say that α_m can be* derived *from α_1.*

• *The* language given by *G is $L_G := \{w \in \Sigma^* \mid S_0 \overset{*}{\underset{G}{\Rightarrow}} w\}$.*

Context-Free Grammar Constraints

Based on the concepts above, we review the definition of context-free grammar constraints from (Sellmann 2006). The purpose of the constraint is to enforce that an assignment to an ordered sequence of variables defines a word in the given grammar.

Definition 4 (Grammar Constraint)**.** *For a grammar $G = (\Sigma, N, P, S_0)$ and variables $X_1, \ldots, X_n$ with domains $D_1, \ldots, D_n \subseteq \Sigma$, we say that $Grammar_G(X_1, \ldots, X_n)$ is true for an instantiation $X_1 \leftarrow w_1, \ldots, X_n \leftarrow w_n$ if and only if it holds that $w = w_1 \ldots w_n \in L_G \cap D_1 \times \cdots \times D_n$. We denote a given grammar constraint $Grammar_G(X_1, \ldots, X_n)$ over a context-free grammar G in CNF by $CFGC_G(X_1, \ldots, X_n)$.*

The filtering algorithm for $CFGC_G$ presented in (Sellmann 2006) is based on the parsing algorithm from Cooke, Younger, and Kasami (CYK). CYK works as follows: Given a word $w \in \Sigma^n$, let us denote the sub-sequence of letters starting at position i with length j (that is, $w_i w_{i+1} \ldots w_{i+j-1}$) by w_{ij}. Based on a grammar $G = (\Sigma, N, P, S_0)$ in CNF, CYK determines iteratively the set of all non-terminals from where we can derive w_{ij}, i.e. $S_{ij} := \{A \in N \mid A \overset{*}{\underset{G}{\Rightarrow}} w_{ij}\}$ for all $1 \le i \le n$ and $1 \le j \le n - i$. It is easy to initialize the sets S_{i1} just based on w_i and all productions $(A \to w_i) \in P$. Then, for j from 2 to n and i from 1 to $n - j + 1$, we have that

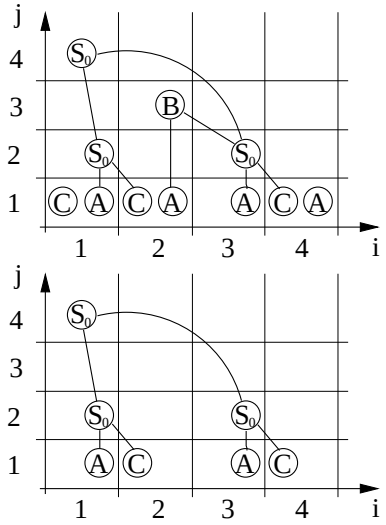

Figure 1: The upper picture shows sets S_{ij} and the lower the sets S'_{ij}. We see that the constraint filtering algorithm determines the only word in $L_G \cap D_1 \ldots D_4$ is "[][]".

$S_{ij} = \bigcup_{k=1}^{j-1} \{A \mid (A \to BC) \in P \text{ with } B \in S_{ik} \text{ and } C \in S_{i+k,j-k}\}$. Then, $w \in L_G$ if and only if $S_0 \in S_{1n}$. From the recursion equation it is simple to derive that CYK can be implemented to run in time $O(n^3|G|) = O(n^3)$ when we treat the size of the grammar as a constant.

The filtering algorithm for $CFGC_G$ that we sketch in Algorithm 1 works bottom-up by computing the sets S_{ij} for increasing j after initializing S_{i1} with all non-terminals that can produce in one step a terminal in the domains of X_i. Then, the algorithm works top-down by removing all non-terminals from each set S_{ij} which cannot be reached from $S_0 \in S_{1n}$. In (Sellmann 2006), it was shown:

Theorem 1. *Algorithm 1 achieves generalized arc-consistency for the $CFGC$ and requires time and space cubic in the number of variables.*

Example: Assume we are given the following context-free, normal-form grammar $G = (\{], [\}, \{A, B, C, S_0\}, \{S_0 \to AC, S_0 \to S_0S_0, S_0 \to BC, B \to AS_0, A \to [\ , C \to\ ]\ \}, S_0)$ that gives the language L_G of all correctly bracketed expressions (like, for example, "[[][]]" or "[][[]]"). In Figure 1, we illustrate how Algorithm 1 works when the initial domain of X_3 is $D_3 = \{[\}$ while all other domains are $D_1 = D_2 = D_4 = \{[,]\}$: First, we work bottom-up, adding non-terminals to the sets S_{ij} if they allow to generate a word in $D_i \times \cdots \times D_{i+j-1}$. Then, in the second step, we work top-down and remove all non-terminals that cannot be reached from $S_0 \in S_{1n}$.

Efficient Context-Free Grammar Filtering

Given the fact that context-free grammar filtering entails the parsing problem, there is little hope that, for general context-free grammar constraints, we can devise significantly faster filtering algorithms. However, with respect to filtering performance it is important to realize that a filtering algorithm should work quickly within a constraint propagation engine. Typically, constraint filtering needs to be conducted on a

sequence of problems, whereby each subsequent problem differs only slightly from the last. When branching decisions and other constraints tighten a given problem, state-of-the-art systems like Ilog Solver provide a filtering routine with information regarding which values were removed from which variable domains since the last call to the routine. By exploiting such condensed information, incremental filtering routines can be devised that work faster than starting each time from scratch. To analyze such incremental algorithms, it has become the custom to provide an upper bound on the total work performed by a filtering algorithm over one entire branch of the search tree (see, e.g., (Katriel et al. 2007)).

Naturally, given a sequence of s monotonically tightening problems (that is, when in each successive problem the variable domains are subsets of the previous problem), context-free grammar constraint filtering for the entire sequence takes at most $O(sn^3|G|)$ steps. Using existing ideas on efficient incremental graph updates in DAGs (see for instance (Fahle et al. 2002)), it is trivial to modify Algorithm 1 so that this time is reduced to $O(n^3|G|)$: When storing additional information on which productions support which arcs in the graph (whereby each production can support at most $2n$ arcs for each set S_{ij}), we can propagate the effects of domain values being removed at the lowest level of the graph to adjacent nodes without ever touching parts of the graph that are not removed. The total workload for the entire problem sequence can then be distributed over all $O(|G|n)$ production supports in each of $O(n^2)$ sets, which results in a time bound of $O(n^2|G|n) = O(n^3|G|)$. Alternatively, incremental filtering can also be achieved by decomposition as shown in (Quimper & Walsh 2007).

The second efficiency aspect regards the memory requirements. In Algorithm 1, they are in $\Theta(n^3|G|)$. It is again trivial to reduce these costs to $O(n^2|G|)$ simply by recomputing the sets of incident arcs rather than storing them in step 2 for step 3 of Algorithm 1. However, following this simplistic approach we only trade time for space. The incremental version of our algorithm as sketched above is based on the fact that we do not need to recompute arcs incident to a node which is achieved by storing them. So while it is trivial to achieve a space-efficient version that requires time in $\Theta(sn^3|G|)$ and space $\Theta(n^2|G|)$ *or* a time-efficient incremental variant that requires time and space in $\Theta(n^3|G|)$ (like the decomposition from (Quimper & Walsh 2007)), the challenge is to devise an algorithm that *combines* low space requirements with good incremental performance.

We will therefore modify our algorithm such that the total workload of a sequence of s monotonically tightening filtering problems is reduced to $O(n^3|G|)$, which implies that, asymptotically, an entire sequence of more and more restricted problems can be filtered with respect to context-free grammars in the same time that it takes to filter just one problem from scratch. At the same time, we will ensure that our modified algorithm will require space in $O(n^2|G|)$.

A Memory- and Time-Efficient Filtering Algorithm

In Algorithm 1, we observe that it first works bottom-up, determining from which nodes (associated with non-terminals of the grammar) we can derive a legal word. Then, it works top-down determining which non-terminal nodes can be used in a derivation that begins with the start non-terminal $S_0 \in S_{1n}$. In order to save both space and time, we will modify these two steps in that every non-terminal in each set S_{ij} will perform just enough work to determine whether its respective node will remain in the shrunken graph Q'.

To this end, in the first step that works bottom-up we will need a routine that determines whether there exists a *support from below*: That is, this routine determines whether a node v_{ijA} has any *outgoing* arcs in $E_1 \cup E_2$. To save space, the routine must perform this check without ever storing sets E_1 and E_2 explicitly, as this would require space in $\Theta(n^3|G|)$.

Analogously, in the second step that works top-down we will rely on a procedure that checks whether there exists a *support from above*: Formally, this procedure determines whether a node v_{ijA} has any *incoming* arcs in $E'_1 \cup E'_2$, again without ever storing these sets which would require too much memory.

The challenge is to avoid having to pay with time what we save in space. To this end, we need a methodology which prevents us from searching for supports (from above or below) that have been checked unsuccessfully before. Very much like the well-known arc-consistency algorithm AC-6 for binary constraint problems (Bessiere & Cordier 1993), we achieve this goal by ordering potential supports so that, when a support is lost, the search for a new support can start right after the last support, in the respective ordering.

According to the definition of E_1, E_2, E'_1, E'_2, supports of v_{ijA} (from above or below) are directly associated with productions in the given grammar G and a splitting index k. To order these supports, we cover and order the productions in G that involve non-terminal A in two lists:

- In list $Out_A := [(A \to B_1 B_2) \in P]$ we store and implicitly fix an ordering on all productions with non-terminal A on the left-hand side.
- In list $In_A := [(B_1 \to B_2 B_3) \in P \mid B_2 = A \ \vee \ B_3 = A]$ we store and implicitly fix an ordering on all productions where non-terminal A appears as non-terminal on the right-hand side.

Now, for each node v_{ijA} we store two production indices p^{Out}_{ijA} and p^{In}_{ijA}, and two splitting indices $k^{Out}_{ijA} \in \{1, \ldots, j\}$ and $k^{In}_{ijA} \in \{j, \ldots, n\}$. The intended meaning of these indices is that node v_{ijA} is currently supported from below by production $(A \to B_1 B_2) = Out_A[p^{Out}_{ijA}]$ such that $B_1 \in S_{i,k^{Out}_{ijA}}$ and $B_2 \in S_{i+k^{Out}_{ijA},j-k^{Out}_{ijA}}$ (analogously for the support from above). When node v_{ijA} has no support from below (or above), we will have $k^{Out}_{ijA} = j$ ($k^{In}_{ijA} = j$).

In Algorithm 2, we show a function that (re-)computes the support from below for a given node v_{ijA}, whereby we assume that variables p^{Out}_{ijA}, k^{Out}_{ijA}, S_{ij}, and Out_A are global and initialized outside of these functions. We see that the routine starts its search for a new support right after the last. This is correct as within a sequence of monotonically tightening problems we will never add edges to the graphs Q and Q'. Therefore, replacement supports can only be found later

Algorithm 2 Function that incrementally (re-)computes the support from below for a given node v_{ijA}.

```
void findOutSupport(i, j, A)
p^Out_ijA ← p^Out_ijA + 1
while k^Out_ijA < j do
  while p^Out_ijA ≤ |Out_A| do
    (A → B1B2) ← Out_A[p^Out_ijA]
    if (B1 ∈ S_{i k^Out_ijA}) and (B2 ∈ S_{i+k^Out_ijA, j−k^Out_ijA}) then
      return
    end if
    p^Out_ijA ← p^Out_ijA + 1
  end while
  p^Out_ijA ← 1, k^Out_ijA ← k^Out_ijA + 1
end while
```

in the respective ordering of supports. The function that recomputes supports from above works analogously.

In Algorithm 3 we present our function filterFromUpdate that re-establishes arc-consistency for the context-free grammar constraint based on the information which variables were affected and which values were removed from their domains since the last filtering call (Δ). The function starts by iterating through the domain changes, whereby each node on the lowest level adds those nodes whose current support relies on its existence to a list of affected nodes (*nodesListOut* and *nodesListIn*). This is a simple task when storing information which nodes a given node supports in lists *listOfOutSupported* and *listOfInSupported* whenever a new support is found (which we do in functions *informOutSupport* and *informInSupport* which we cannot show here for lack of space). Furthermore, by organizing the affected nodes according to the level to which they belong, we make it easy to perform the two phases (one working bottom-up, the other top-down) later, whereby a simple flag (*lost*) ensures that no node is added twice.

Algorithm 3 Filtering context-free grammars incrementally in space $O(n^2|G|)$ and amortized total time $O(n^3|G|)$ for any sequence of monotonically tightening problems.

```
bool filterFromUpdate(varSet,Δ1, ..., Δn)
for r = 1 to n do
  nodeListOut[r] ← ∅, nodeListIn[r] ← ∅
end for
for all X_i ∈ varSet and a ∈ Δ_i do
  if not (A_a ∈ S_i1) then continue end_if
  S_i1 ← S_i1 \ {A_a}
  for all (p, q, B) ∈ listOfOutSupported_{i,1,A_a} do
    if (lost^Out_pqB) then continue end_if
    lost^Out_pqB ← true
    nodeListOut[q].add(p, q, B)
  end for
  for all (p, q, B) ∈ listOfInSupported_{i,1,A_a} do
    if (lost^In_pqB) then continue end_if
    lost^In_pqB ← true
    nodeListIn[q].add(p, q, B)
  end for
end for
updateOutSupports()
if (S0 ∉ S_1n) then return false end_if
S_1n ← {S0}
updateInSupports()
return true
```

Algorithm 4 Function computing new supports from below by proceeding bottom-up.

```
void updateOutSupports(void)
for r = 2 to n do
  for all (i, j, A) ∈ nodeListOut[r] do
    (A → B1B2) ← Out_A[p^Out_ijA]
    listOfOutSupported_{i,k^Out_ijA,B1}.remove(left^Out_ijA)
    listOfOutSupported_{i+k^Out_ijA,j−k^Out_ijA,B2}.remove(right^Out_ijA)
    findOutSupport(i,j,A)
    if (k^Out_ijA < j) then
      lost^Out_ijA ← false
      informOutSupport(i, j, A)
      continue
    end if
    S_ij ← S_ij \ {A}
    for all (p, q, B) ∈ listOfOutSupported_{i,j,A} do
      if (lost^Out_pqB) then continue end_if
      lost^Out_pqB ← true
      nodeListOut[q].add(p, q, B)
    end for
    for all (p, q, B) ∈ listOfInSupported_{i,j,A} do
      if (lost^In_pqB) then continue end_if
      lost^In_pqB ← true
      nodeListIn[q].add(p, q, B)
    end for
  end for
end for
```

In Algorithm 4, we show how the phase that recomputes the supports from below proceeds: We iterate through the affected nodes bottom-up. First, for each node that has lost its support from below, because one of its supporting nodes was lost, we inform the other supporting node that it is no longer supporting the current node. Then, we try to replace the lost support from below by calling findOutSupport. Recall that the function seeks for a new support starting at the old, so that no two potential supports are investigated more than just once. Now, if we were successful in providing a new support from below (test $k^{Out}_{ijA} < j$), we call *informOutSupport* which informs the new supporting nodes that the support of the current node relies on them. Otherwise, the current node is removed and the nodes that it supports are being added to the lists of affected nodes. The update of supports from above works analogously. With the complete method as outlined in Algorithms 2–4, we can now show:

Theorem 2. *For a sequence of s monotonically tightening context-free grammar filtering problems, based on the grammar G in CNF, filtering for the entire sequence can be performed in time $O(n^3|G|)$ and space $O(n^2|G|)$.*

Proof. Note that nodes that are removed because they lost their support from above do not need to inform the nodes that they support from below. This is obviously not nec-

Benchmark			Search-Tree			Non-Incremental		Incremental		Speedup
ID	#Act.	#Workers	#Fails	#Prop's	#Nodes	Time [sec]	Mem [MB]	Time [sec]	Mem [MB]	
1_1	1	1	2	136	19	79	12	1.75	24	45
1_2	1	3	144	577	290	293	38	5.7	80	51
1_3	1	4	409	988	584	455	50	8.12	106	56
1_4	1	5	6	749	191	443	60	9.08	124	46
1_5	1	4	6	598	137	399	50	7.15	104	55
1_6	1	5	6	748	161	487	60	9.1	132	53
1_7	1	6	5311	6282	5471	1948	72	16.13	154	120
1_8	1	2	6	299	99	193	26	3.57	40	54
1_9	1	1	2	144	35	80	16	1.71	18	47
1_10	1	7	17447	19319	1769	4813	82	25.57	176	188
2_1	2	2	10	430	65	359	44	7.34	88	49
2_5	2	4	24	412	100	355	44	7.37	88	48
2_6	2	5	30	603	171	496	58	9.89	106	50
2_7	2	6	44	850	158	713	84	15.14	178	47
2_8	2	2	24	363	114	331	44	3.57	84	92
2_9	2	1	16	252	56	220	32	4.93	52	44
2_10	2	7	346	1155	562	900	132	17.97	160	50

Table 1: Shift-scheduling: We report running times on an AMD Athlon 64 X2 Dual Core Processor 3800+ for benchmarks with one and two activity types. For each worker, the corresponding grammar in CNF has 30 non-terminals and 36 productions. Column #Propagations shows how often the propagation of grammar constraints is called for. Note that this value is different from the number of choice points as constraints are usually propagated more than just once per choice point.

essary as the supported nodes must have been removed before as they could otherwise provide a valid support from above. Thus, after having conducted one bottom-up phase and one top-down in filterFromUpdate, all remaining nodes must have an active support from below and above. With the completeness proof of Algorithm 1 in (Sellmann 2006), this implies that we filter enough.

On the other hand, we also never remove a node if there still exist supports from above and below: we know that, if a support is lost, a replacement can only be found later in the chosen ordering of supports. Therefore, if functions findOutSupport or findInSupport fail to find a replacement support, then none exists. Consequently, our filtering algorithm is also sound.

Regarding space, we note that the total memory needed to store, for each of the $O(n^2|G|)$ nodes, the nodes that are supported by them is not larger than the number of nodes supporting each node (which is four) times the number of all nodes. Therefore, while an individual node may support many other nodes, for all nodes together the space required to store this information is bounded by $O(4n^2|G|) = O(n^2|G|)$. All other global arrays (like left, right, p, k, S, lost, and so on) also only require space in $O(n^2|G|)$.

Finally, it remains to analyze the total effort for a sequence of s monotonically tightening filtering problems. Given that, in each new iteration, at least one assignment is lost, we know that $s \leq |G|n$. For each filtering problem, we need to update the lowest level of nodes and then iterate twice (once bottom-up and once top-down) through all levels (even if levels should turn out to contain no affected nodes), which imposes a total workload in $O(|G|n + 2n|G|n) = O(n^2|G|)$. All other work is dominated by the total work done in all calls to functions findInSupport and findOutSupport. Since these functions are never called for any node for which it has been assessed before that it has no more support from either above or below, each time that one of these functions is called, the support pointer of some node is increased by at least one. Again we find that the total number of potential supports for an individual node could be as large as $\Theta(|G|n)$, while the number of potential supports for all nodes in each set S_{ij} is asymptotically not larger. Consequently, the total work performed is bounded by the number of sets S_{ij} times the number of potential supports for all nodes in each of them. Thus, the entire sequence of filtering problems can be handled in $O(n^2|G|n) = O(n^3|G|)$. □

Note that the above theorem states time- and space complexity for monotonic reductions without restorations only! Within a backtrack search, we need to also store lost supports for each search node leading to the current choice point so that we can backtrack quickly. Consequently, within a full backtrack search we cannot keep the claim of quadratic space requirements. However, as we will see in the next section, the amount of additional restoration information needed is fortunately limited in practice.

Experimental Results

We implemented the previously outlined incremental context-free grammar constraint propagation algorithm and compared it against its non-incremental counterpart on real-world instances of the shift-scheduling problem introduced in (Demassey, Pesant & Rousseau 2006). We chose this problem because it is the only real-world problem grammar constraints have been tested on before in (Quimper & Walsh 2007). [1] The problem is that of a retail store manager who

[1] Many thanks to L.-M. Rousseau for providing the benchmark!

needs to staff his employees such that the expected demands of workers for various activities that must be performed at all times of the day are met. The demands are given as upper and lower bounds of workers performing an activity a_i at each 15-minute time period of the day.

Labor requirements govern the feasibility of a shift for each worker: 1. A work-shift covers between 3 and 8 hours of actual work activities. 2. If a work-activity is started, it must be performed for at least one consecutive hour. 3. When switching from one work-activity to another, a break or lunch is required in between. 4. Breaks, lunches, and off-shifts do not follow each other directly. 5. Off-shifts must be assigned at the beginning and at the end of each work-shift. 6. If the actual work-time of a shift is at least 6 hours, there must be two 15-minute breaks and one one-hour lunch break. 7. If the actual work-time of a shift is less than 6 hours, then there must be exactly one 15-minute break and no lunch-break. We implemented these constraints by means of one context-free grammar constraint per worker and several global-cardinality constraints ("vertical" gcc's over each worker's shift to enforce the last two constraints, and "horizontal" gcc's for each time period and activity to meet the workforce demands) in Ilog Solver 6.4. To break symmetries between the indistinguishable workers, we introduced constraints that force the ith worker to work at most as much as the $i+$first worker.

Table 1 summarizes the results of our experiments. We see that the incremental propagation algorithm vastly outperforms its non-incremental counterpart, resulting in speed-ups of up to a factor 188 while exploring *identical* search-trees! It is quite rare to find that the efficiency of filtering techniques leads to such dramatic improvements with unchanged filtering effectiveness. These results confirm the speed-ups reported in (Quimper & Walsh 2007). We also tried to use the decomposition approach from (Quimper & Walsh 2007), but, due to the method's excessive memory requirements, on our machine with 2 GByte main memory we were only able to solve benchmarks with one activity and one worker only (1_1 and 1_9). On these instances, the decomposition approach implemented in Ilog Solver 6.4 runs about ten times slower than our approach (potentially because of swapped memory) and uses about 1.8 GBytes memory. Our method, on the other hand, requires only 24 MBytes. Finally, when comparing the memory requirements of the non-incremental and the incremental variants, we find that the additional memory needed to store restoration data is limited in practice.

Cost-Based Filtering for Context-Free Grammar Constraints

In our final technical section, we consider problems where context-free grammar constraints appear in conjunction with a linear objective function that we are trying to maximize. Assume that each potential variable assignment $X_i \leftarrow w_i$ is associated with a profit $p^i_{w_i}$, and that our objective is to find a complete assignment $X_1 \leftarrow w_1 \in D_1, \ldots, X_n \leftarrow w_n \in D_n$ such that $CFGC_G(X_1, \ldots, X_n)$ is true for that instantiation and $p(w_1 \ldots w_n) := \sum_{i=1}^{n} p^i_{w_i}$ is maximized. Once we have found a feasible instantiation that achieves profit T, we are only interested in improving solutions. Therefore, we consider the conjunction of the context-free grammar constraint $CFGC_G$ with the requirement that solutions ought to have profit greater than T. In Algorithm 5, we give an efficient algorithm that performs cost-based filtering for context-free grammar constraints.

Algorithm 5 CFGC Cost-Based Filtering Algorithm

1. For all $1 \leq i \leq n$, initialize $S_{i1} := \emptyset$. For all $1 \leq i \leq n$ and productions $(A_a \rightarrow a) \in P$ with $a \in D_i$, set $f^{i1}_{A_a} := p(a)$, and add all such A_a to S_{i1}.
2. For all $j > 1$ in increasing order, $1 \leq i \leq n$, and $A \in N$, set $f^{ij}_A := \max\{f^{ik}_B + f^{i+k,j-k}_C \mid A \underset{G}{\Rightarrow} BC, B \in S_{ik}, C \in S_{i+k,j-k}\}$, and $S_{ij} := \{A \mid f^{ij}_A > -\infty\}$.
3. If $f^{1n}_{S_0} \leq T$, then the optimization constraint is not satisfiable, and we stop.
4. Initialize $g^{1n}_{S_0} := f^{1n}_{S_0}$.
5. For all $k < n$ in decreasing order, $1 \leq i \leq n$, and $B \in N$ set $g^{ik}_B := \max\{g^{ij}_A - f^{ij}_A + f^{ik}_B + f^{i+k,j-k}_C \mid (A \rightarrow BC) \in P, A \in S_{ij}, C \in S_{i+k,j-k}\} \cup \{g^{i-j,j+k}_A - f^{i-j,j+k}_A + f^{i-j,j}_C + f^{i,k}_B \mid (A \rightarrow CB) \in P, A \in S_{i-j,j+k}, C \in S_{i-j,j}\}$.
6. For all $1 \leq i \leq n$ and $a \in D_i$ with $(A_a \rightarrow a) \in P$ and $g^{i1}_{A_a} \leq T$, remove a from D_i.

Lemma 1. *In Algorithm 5:*

1. *It holds that* $f^{ij}_A = \max\{p(w_{ij}) \mid A \underset{G}{\overset{*}{\Rightarrow}} w_{ij} \in D_i \times \cdots \times D_{i+j-1}\}$, *and* $S_{ij} = \{A \mid \exists\, w_{ij} \in D_i \times \cdots \times D_{i+j-1} : A \underset{G}{\overset{*}{\Rightarrow}} w_{ij}\}$.
2. *It holds that* $g^{ik}_B = \max\{p(w) \mid w \in L_G \cap D_1 \times \cdots \times D_n, B \underset{G}{\overset{*}{\Rightarrow}} w_{ik}, S_0 \underset{G}{\overset{*}{\Rightarrow}} w_1 \ldots w_{i-1} B w_{i+k} \ldots w_n\}$.

Proof. 1. The lemma claims that the sets S_{ij} contains all non-terminals that can derive a word supported by the domains of variables $X_i, \ldots, X_{i+j-1}$, and that f^{ij}_A reflects the value of the highest profit word $w_{ij} \in D_i \times \cdots \times D_{i+j-1}$ that can be derived from non-terminal A. To prove this claim, we induce over j. For $j = 1$, the claim holds by definition of S_{i1} and f^{i1}_A in step 1. Now assume $j > 1$ and that the claim is true for all $1 \leq k < j$. Then $\max\{p(w_{ij}) \mid A \underset{G}{\overset{*}{\Rightarrow}} w_{ij} \in D_i \times \cdots \times D_{i+j-1}\} = \max\{p(w_{ij}) \mid w_{ij} \in D_i \times \cdots \times D_{i+j-1}, A \underset{G}{\Rightarrow} BC, B \underset{G}{\overset{*}{\Rightarrow}} w_{ik}, C \underset{G}{\overset{*}{\Rightarrow}} w_{i+k,j-k}\} = \max\{p(w_{ik}) + p(w_{i+k,j-k}) \mid w_{ij} \in D_i \times \cdots \times D_{i+j-1}, A \underset{G}{\Rightarrow} BC, B \underset{G}{\overset{*}{\Rightarrow}} w_{ik}, C \underset{G}{\overset{*}{\Rightarrow}} w_{i+k,j-k}\} = \max_{(A \rightarrow BC) \in P} \max\{p(w_{ik}) \mid B \underset{G}{\overset{*}{\Rightarrow}} w_{ik} \in D_i \times \cdots \times D_{i+k-1}\} + \max\{p(w_{i+k,j-k}) \mid C \underset{G}{\overset{*}{\Rightarrow}} w_{i+k,j-k} \in D_{i+k} \times \cdots \times D_{i+j-1}\} = \max\{f^{ik}_B + f^{i+k,j-k}_C \mid A \underset{G}{\Rightarrow} BC, B \in$

$S_{ik}, C \in S_{i+k,j-k}\} = f_A^{ij}$. Then, f_A^{ij} marks the maximum over the empty set if and only if no word in accordance with the domains of $X_i, \ldots, X_{i+j-1}$ can be derived. This proves the second claim that $S_{ij} = \{A \mid f_A^{ij} > -\infty\}$ contains exactly all those non-terminals from where a word in $D_i \times \cdots \times D_{i+j-1}$ can be derived.

2. The lemma claims that the value g_A^{ij} reflects the maximum value of any word $w \in L_G \cap D_1 \times \cdots \times D_n$ in whose derivation non-terminal A can be used to produce w_{ij}. We prove this claim by induction over k, starting with $k = n$ and decreasing to $k = 1$. We only ever get past step 3 if there exists a word $w \in L_G \cap D_1 \times \cdots \times D_n$ at all. Then, for $k = n$, with the previously proven part 1 of this lemma, $\max\{p(w) \mid w \in L_G \cap D_1 \times \cdots \times D_n, S_0 \overset{*}{\underset{G}{\Rightarrow}} w_{1n}\} = f_{S_0}^{1n} = g_{S_0}^{1n}$. Now let $k < n$ and assume the claim is proven for all $k < j \leq n$. For any given $B \in N$ and $1 \leq i \leq n$, denote with $w \in L_G \cap D_1 \times \cdots \times D_n$ the word that achieves the maximum profit such that $B \overset{*}{\underset{G}{\Rightarrow}} w_{ik}$ and $S_0 \overset{*}{\underset{G}{\Rightarrow}} w_1..w_{i-1}Bw_{i+k}..w_n$. Let us assume there exist non-terminals $A, C \in N$ such that $S_0 \overset{*}{\underset{G}{\Rightarrow}} w_1..w_{i-1}Aw_{i+j}..w_n \underset{G}{\Rightarrow} w_1..w_{i-1}BCw_{i+j}..w_n \overset{*}{\underset{G}{\Rightarrow}} w_1..w_{i-1}Bw_{i+k}..w_n$ (the case where non-terminal B is introduced in the derivation by application of a production $(A \rightarrow CB) \in P$ follows analogously). Due to the fact that w achieves maximum profit for $B \in S_{ij}$, we know that $g_A^{ij} = p(w_{1,i-1}) + f_A^{ij} + p(w_{i+j,n-i-j})$. Moreover, it must hold that $f_B^{ik} = p(w_{ik})$ and $f_C^{i+k,j-k} = p(w_{i+k,j-k})$. Then, $p(w) = (p(w_{1,i-1}) + p(w_{i+j,n-i-j})) + p(w_{ik}) + p(w_{i+k,j-k}) = (g_A^{ij} - f_A^{ij}) + f_B^{ik} + f_C^{i+k,j-k} = g_B^{ik}$.

□

Theorem 3. *Algorithm 5 achieves generalized arc-consistency on the conjunction of a $CFGC$ and a linear objective function constraint. The algorithm requires cubic time and quadratic space in the number of variables.*

Proof. We show that value a is removed from D_i if and only if for all words $w = w_1 \ldots w_n \in L_G \cap (D_1 \ldots D_n)$ with $a = w_i$ it holds that $p(w) \leq T$.

$\Rightarrow$ (Soundness) Assume that value a is removed from D_i. Let $w \in L_G \cap (D_1 \ldots D_n)$ with $w_i = a$. Due to the assumption that $w \in L_G$ there must exist a derivation $S_0 \overset{*}{\underset{G}{\Rightarrow}} w_1 \ldots w_{i-1} \; A_a \; w_{i+1} \ldots w_n \underset{G}{\Rightarrow} w_1 \ldots w_{i-1}w_iw_{i+1} \ldots w_n$ for some $A_a \in N$ with $(A_a \rightarrow a) \in P$. Since a is being removed from D_i, we know that $g_{A_a}^{i1} \leq T$. According to Lemma 1, $p(w) \leq g_{A_{w_i}}^{i1} \leq T$.

$\Leftarrow$ (Completeness) Assume that for all $w = w_1 \ldots w_n \in L_G \cap (D_1 \ldots D_n)$ with $w_i = a$ it holds that $p(w) \leq T$. According to Lemma 1, this implies $g_{A_a}^{i1} \leq T$. Then, a is removed from the domain of X_i in step 6.

Regarding the time complexity, it is easy to verify that the workload is dominated by steps 2 and 5, both of which require time in $O(n^3|G|)$. The space complexity is dominated by the memorization of values f_A^{ij} and g_A^{ij}, and it is thus limited by $O(n^2|G|)$. □

Conclusions

We devised an incremental space- and time-efficient arc-consistency algorithm for context-free grammar constraints. For an entire sequence of monotonically tightening problems, we can now perform filtering in quadratic space and the same worst-case time as it takes to parse a context-free grammar by the Cooke-Younger-Kasami algorithm (CYK). We showed experimentally that the new algorithm is equally effective but massively faster in practice than its non-incremental counterpart. We also gave a new algorithm that performs cost-based filtering when a context-free grammar constraint occurs in combination with a linear objective function. This algorithm has again the same cubic worst-case complexity of CYK. Further research is needed to determine whether it is possible to devise an incremental version of this algorithm.

References

N. Beldiceanu, M. Carlsson, T. Petit. Deriving Filtering Algorithms from Constraint Checkers. *CP*, LNCS 3258, 2004.

C. Bessiere and M.O. Cordier. Arc-Consistency and Arc-Consistency again. *AAAI*, 1993.

S. Bourdais, P. Galinier, G. Pesant. HIBISCUS: A Constraint Programming Application to Staff Scheduling in Health Care. *CP*, LNCS 2833:153–167, 2003.

N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory* 2:113–124, 1956.

S. Demassey, G. Pesant, L.-M. Rousseau. A cost-regular based hybrid column generation approach. *Constraints*, 11(4):315–333, 2006.

T. Fahle, U. Junker, S.E. Karisch, N. Kohl, M. Sellmann, and B. Vaaben. Constraint programming based column generation for crew assignment. *Journal of Heuristics*, 8(1):59–81, 2002.

F. Focacci, A. Lodi, and M. Milano. Cost-Based Domain Filtering. *CP*, LNCS 1713:189–203, 1999.

J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*, Addison Wesley, 1979.

I. Katriel, M. Sellmann, E. Upfal, P. Van Hentenryck. Propagating Knapsack Constraints in Sublinear Time. *AAAI*, 2007.

G. Pesant. A Regular Language Membership Constraint for Sequences of Variables *Modelling and Reformulating Constraint Satisfaction Problems*, pp. 110–119, 2003.

C.-G. Quimper and L.-M. Rousseau. Language Based Operators for Solving Shift Scheduling Problems. *Metaheuristics*, 2007.

C.-G. Quimper and T. Walsh. Global Grammar Constraints. *CP*, LNCS 3258:751–755, 2006.

C.-G. Quimper and T. Walsh. Decomposing Global Grammar Constraints. *CP*, LNCS 4741:590–604, 2007.

M. Sellmann. The Theory of Grammar Constraints. *CP*, LNCS 4204:530–544, 2006.

M. Trick. A dynamic programming approach for consistency and propagation for knapsack constraints. *CPAIOR*, 113–124, 2001.