Ann Oper Res (2011) 184: 179–207
DOI 10.1007/s10479-010-0697-y

The weighted GRAMMAR constraint

George Katsirelos · Nina Narodytska · Toby Walsh

Published online: 5 February 2010

Abstract We introduce the WEIGHTEDGRAMMAR constraint and propose propagation algorithms based on the CYK parser and the *Earley* parser. We show that the traces of these algorithms can be encoded as a weighted negation normal form (WNNF), a generalization of NNF that allows nodes to carry weights. Based on this connection, we prove the correctness and complexity of these algorithms. Specifically, these algorithms enforce domain consistency on the WEIGHTEDGRAMMAR constraint in time $O(n^3)$. Further, we propose that the WNNF constraint can be decomposed into a set of primitive arithmetic constraint without hindering propagation.

Keywords Constraint programming · Global constraints · Context free grammars · Negation normal form

1 Introduction

Specialized knowledge compilation languages like decomposable negation normal form and ordered binary decision diagrams have proved highly useful to represent a wide range of reasoning problems (Darwiche and Marquis 2002). For example, Quimper and Walsh (2007) demonstrated how the global GRAMMAR constraint can be compiled into a decomposable negation normal form, and showed that this is an effective way to propagate the GRAMMAR constraint. Representations like this have a number of nice properties. For instance, they can themselves be readily compiled into conjunctive normal form and given to a fast SAT solver. Here we consider a new knowledge compilation language, weighted negation normal form, and demonstrate that it is useful for studying the global GRAMMAR constraint. Weighted

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council. This paper is an extension of work previously published in Katsirelos et al. (5th international conference, CPAIOR 5015: 323–327, 2008).

G. Katsirelos · N. Narodytska (✉) · T. Walsh
University of New South Wales and NICTA, Sydney, Australia
e-mail: n.naroditskaya@gmail.com

negation normal form is a special case of Fargier and Marquis' recently proposed valued negation normal form (Fargier and Marquis 2007).

The global GRAMMAR constraint restricts the sequence of values assigned to its variables to form a string from a language defined by a given grammar. The advantage of the GRAMMAR constraint compared to the REGULAR constraint (Pesant 2004), which follows the same approach, is that it allows for an exponentially more succinct representation of some constraints, while still admitting an efficient propagation algorithm. The GRAMMAR constraint has proved useful in modelling a wide range of scheduling, rostering and sequencing problems. Here we introduce the weighted form of the global GRAMMAR constraint. The addition of weights improves on the unweighted GRAMMAR constraint in the following ways: first, it allows us to model over-constrained problems and problems with preferences. For instance, we can express the soft GRAMMAR constraint with Hamming and edit distance violation measures in terms of a WEIGHTEDGRAMMAR constraint. Second, for a certain class of objective functions that includes linear functions, it allows us to propagate the conjunction of a GRAMMAR constraint and the objective function, thus significantly reducing the size of the search space that is explored. Third, some constraints can be expressed even more succinctly in terms of a WEIGHTEDGRAMMAR constraints compared to the unweighted case, allowing for more efficient propagation. For example, the EDITDISTANCE global constraint can be expressed in terms of the weighted GRAMMAR constraint (Katsirelos et al. 2009). Such encodings open the possibility of using well known formal language theory algorithms to construct and propagate conjunctions of other global constraints with the EDITDISTANCE constraint. An example that is explored in Katsirelos et al. (2009) is the conjunction of an EDITDISTANCE constraint and a REGULAR constraint.

We make several contributions here. First, we introduce the WEIGHTEDGRAMMAR constraint and propose two propagation algorithms for it based on the CYK and Earley parsers. Second, we introduce weighted negation normal form (WNNF), a representation language that extends NNF with the addition of weights. We show that the trace of the actions of the CYK- and Earley-based propagators for the WEIGHTEDGRAMMAR constraint can be explained in terms of constructing and reasoning about a suitable formula expressed in S-WDNNF, a tractable subclass of WNNF. We propose a propagator for constraints expressed in S-WDNNF, a decomposition of this propagator into a set of arithmetic constraints and investigate the model of counting the models of such formulae. Finally, we experimentally show that the weighted GRAMMAR constraint is efficient for solving real-world shift scheduling problems.

2 Background

A constraint satisfaction problem (CSP) consists of a set of variables $\mathbf{X}$, each with a finite domain of values, and a set of constraints. The domain of the variable X is denoted $D(X)$. A constraint C is defined over a set of variables $scope(C) \subseteq \mathbf{X}$ and it specifies allowed combinations of values from the domains of the variables $scope(C)$. Each allowed combination of values for the variables $scope(C)$ is called a solution of C. A solution of a CSP is an assignment of a value to each variable that is also a solution of all its constraints. Backtracking search solvers construct partial assignments, enforcing a local consistency to prune the domains of the variables so that values which cannot appear in any extension of the partial to a solution are removed. We consider one of the most common local consistencies: domain consistency (DC). A value $X_i = a$ is DC for a constraint C if there exists a solution of C

that assigns $X_i = a$. This solution is called a *support* of $X_i = a$ in C. A constraint C is domain consistent if every value in the domain of a variable in $scope(C)$ is DC. A CSP is DC if each of its constraints is DC.

A context-free grammar is a tuple $G = \langle \Sigma, T, S, P \rangle$, where Σ is a set of *terminal* symbols called the *alphabet* of G, T is a set of *non-terminal* symbols, S is a unique starting symbol and P is a set of productions. A production is a rule of the form $A \to \alpha$ where A is a non-terminal and α is a sequence of terminal and non-terminal symbols. A string in Σ^* is *generated* or *accepted* by G starting with the sequence $s = \langle S \rangle$ and non deterministically generating s' by replacing any non-terminal A in s by the right hand side of any production $A \to \alpha$ until s' contains only terminal symbols.

A context free grammar is in Chomsky normal form is all productions are of the form $A \to BC$ where B and C are non terminals or $A \to a$ where a is a terminal. Any context free grammar can be converted to one that is in Chomsky normal form with at most a linear increase in its size. A context free grammar is *regular* if all productions are of the form $A \to a$ or $A \to bB$ where a and b are terminals and B is a non terminal. Regular grammars are a strict subset of context free grammars. A context free language $L(G)$ is the language of all strings that are generated by the context free grammar G. We will consider global constraints which are specified in terms of a context free grammar. Such a constraint C considers the variables in $scope(C)$ as a string of length n. The solutions of C are all the strings of length n that are accepted by the grammar (Quimper and Walsh 2006; Sellmann 2006).

Example 1 Consider the following context-free grammar in Chomsky normal form, G:

$$S \to AB \qquad A \to AA \mid a \qquad B \to BB \mid b$$

Suppose X_1, X_2 and $X_3 \in \{a, b\}$. Then enforcing DC on $\textsc{Grammar}(G, [X_1, X_2, X_3])$ prunes b from X_1 and a from X_3 as the only supports are the sequences aab and abb.

The Regular constraint (Pesant 2004) is a special case of the Grammar constraint. This accepts just those assignments which come from a regular language. A regular language can alternatively be specified in terms of a finite automaton. Note that since we fix grammars to accept strings of a specific length, it is possible to convert any context free grammar to a regular grammar. However, this may entail an exponential blowup in the size of the grammar. Similarly, any non-deterministic finite automaton can be converted to a deterministic finite automaton, but this step may increase the size of the automaton exponentially.

3 Weighted NNF

In this paper, we will study propagation algorithms for the WeightedGrammar constraint based on two parsers: the CYK parser and the *Earley* parser. We will show that both of these propagation algorithms can be explained on the basis of an AND/OR graph that is implicitly generated by these parsers. Therefore, we first introduce *weighted* NNF, a special case of the valued NNF (VNNF) framework (Fargier and Marquis 2007) for the representation context $\langle \mathbb{Z} \cup \{\infty\}, \leq, \{+, \min\} \rangle$. WNNF is especially useful in analyzing the CYK and *Earley* parsers.

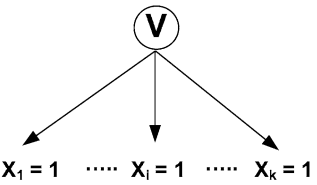

Fig. 1 An encoding of the function $(X_1 \vee X_2 \vee \cdots \vee X_k)$ using NNF

To introduce WNNF we recall first the definition of NNF (Darwiche and Marquis 2002):

Definition 1 NNF An NNF N is a rooted, directed acyclic graph (DAG) where each leaf node is labeled with TRUE, FALSE, or a literal l of a propositional variable x, and each internal node is labeled with AND-NODE or OR-NODE and can have arbitrarily many children.

An NNF N represents a Boolean formula over the variables that appear in the leaves of N. A *model* of N is an instantiation of the variables in N that evaluates the corresponding Boolean formula to TRUE.

Example 2 Figure 1 shows an NNF N for the function $f = (X_1 \vee X_2 \vee \cdots \vee X_k)$. The leaves of the NNF N are positive literals of the propositional variables $X_1, X_2, \ldots, X_k$.

We use the same notation for a graph N and the formula that it represents. We denote a subgraph (and subformula) rooted at a node T as N_T. In addition, we may attach an additional label to each node and refer to specific nodes by their labels.

In general, reasoning in an unrestricted NNF is intractable. However, there exist restrictions of this language that make specific types of queries tractable.

Definition 2 (Decomposable NNF) A DNNF N is an NNF with the decomposability property, i.e., if T is an AND-NODE of N, then for every pair T_1, T_2 of its children nodes, $vars(N_{T_1}) \cap vars(N_{T_2}) = \emptyset$.

Definition 3 (Deterministic NNF) A D-NNF N is an NNF with the determinism property, i.e., if T is an OR-NODE of N, then every pair T_1, T_2 of its children are mutually inconsistent, $\not\models N_{T_1} \wedge N_{T_2}$.

Definition 4 (Smooth NNF) A smooth NNF N is an NNF with the smoothness property, i.e., if T is an OR node of N, then for every pair T_1, T_2 of its children $vars(N_{T_1}) = vars(N_{T_2})$.

The significance of decomposability is that it makes satisfiability queries on NNF tractable, while determinism makes model counting tractable (Darwiche and Marquis 2002). In general, it is not possible to convert an arbitrary NNF into a decomposable NNF without an exponential blowup in the size of the formula (Darwiche and Marquis 2002). It is possible, however, to convert any NNF into a smooth NNF with at most a linear increase in the size of the formula.

These three properties are orthogonal, therefore we can talk about NNF with any combination of them. In particular, we are interested in decomposable smooth NNF (S-DNNF) and deterministic S-DNNF (SD-DNNF).

Next we introduce the weighted NNF that is an extension of smooth NNF.

Definition 5 (Weighted Smooth NNF) A weighted smooth NNF $N(z)$ is a smooth NNF where each node T has a weight $w[T]$. Let I be an instantiation of the variables in $N(z)$. The weight $w_I[T]$ of a node T of $N(z)$ given I is

$$w_I[T] = \begin{cases} w[l] & \text{if } T \text{ is a literal } l \text{ and } l \in I \\ z+1 & \text{if } T \text{ is a literal } l \text{ and } l \notin I \\ \min_{T_i \in \mathrm{CHD}(T)}(w_I[T_i]) + w[T] & \text{if } T \text{ is an OR node} \\ \sum_{T_i \in \mathrm{CHD}(T)} w_I[T_i] + w[T] & \text{if } T \text{ is an AND node} \end{cases}$$

The weight of N given I, which is denoted $w_I[N]$, is the weight of the root node T_{root} given I: $w_I[N] = w_I[T_{root}]$. An instantiation I is a model of $N(z)$ if and only if $w_I[N]$ less than or equal to the cost variable z.

A WNNF describes a propositional formula, in the same way that NNF does. In fact, given any $N(z)$, any of its models is also a model of the unweighted NNF N, but the converse does not necessarily hold.

Note that the definition of WNNF requires that the underlying NNF is smooth. This is because a non-smooth underlying NNF leads to problematic behaviour. Consider, for example, the case where we simply assign weights to literals, using the equivalent underlying NNFs $N_1 \equiv x_1 \vee x_2$ and $N_2 \equiv (X_1 = 1 \wedge (X_2 = 1 \vee X_2 = 0)) \vee (X_2 = 1 \wedge (X_1 = 1 \vee X_1 = 0))$, with N_1 being non-smooth and N_2 smooth. Let the weights assigned to the literals be $w(X_1 = 1) = 1, w(X_1 = 0) = 0, w(X_2 = 1) = w(X_2 = 0) = 10$. In this case, the WNNF based on N_1 evaluates the instantiation $X_1 = 1, X_2 = 1$ as having weight 1 at the root, while the WNNF based on N_2 evaluates the same instantiation as having weight 11 at the root. Requiring that the underlying NNF of WNNF is smooth avoids these issues.

Example 3 The expressiveness of s-WDNNF allows us to build succinct representations of some Boolean functions. For instance, we can use s-WDNNF to construct a compact representation of the conjunction of an arbitrary Boolean formula represented as s-DNNF and the SUM constraint. The $\mathrm{SUM}(X_1, \ldots, X_k, c)$ constraint ensures that the weighted sum of variables X_i is at most c, $\sum_{i=1}^{k} a_i X_i \leq c$.

Consider again the Boolean function $f = (X_1 \vee X_2 \vee \cdots \vee X_k)$ from Example 2. Suppose we want to build a conjunction of f and the SUM constraint. We preform the construction of s-WDNNF into two steps. First, we smooth NNF (Fig. 1) for the formula f and obtain s-DNNF that is shown at Fig. 2. Secondly, we set the weight to zero for all negative literals $(X_i = 0)$, $i = 1, 2, \ldots, k$ and to a_i for all positive literals $(X_i = 1)$, $i = 1, 2, \ldots, k$. We set the cost variable z to c, so the weight of each model of $N(c)$ is less than or equal to c. Figure 2 shows the s-WDNNF $N(c)$ that encodes $f(X_1, \ldots, X_k) \wedge \mathrm{SUM}(X_1, \ldots, X_k, c)$.

Note that the best known way to encode the SUM constraint in a DNNF requires size $O(\sum_{i \in 1 \ldots k} a_i k)$ (Eén and Sörensson 2006), which is exponential in the number of bits needed to represent each a_i, while it is in general intractable to construct the conjunction of two formulas in DNNF.

Similarly to the unweighted case, we can consider deterministic and decomposable s-WNNF. We show here that the tractability results for these restricted forms of NNF also hold for the same restrictions of WNNF.

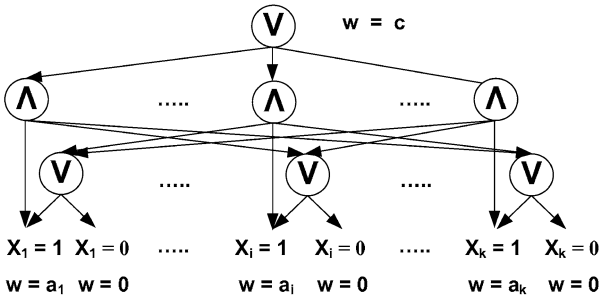

Fig. 2 An encoding of a conjunction $X_1, \ldots, X_k$ and $\sum_{i=1}^{k} a_i X_i \leq c$ using s-WDNNF

We present first an algorithm to test satisfiability of a s-WDNNF, which is a specialization of the results in Fargier and Marquis (2007). In addition, we present an algorithm to detect inconsistent literals, i.e., literals that may not appear in any model of a s-WDNNF. Both of these algorithms run in $O(|N(z)|)$ time, where the size $|N(z)|$ of a s-WDNNF $N(z)$ is *the number of edges* in the *DAG*.

Lemma 1 *Let $N(z)$ be a weighted smooth* DNNF *and P be a set of literal inconsistent with $N(z)$. The algorithm* s-WDNNF-*lowerbound returns a function l such that for each node T in $N(z)$, l gives the minimum weight of the models of the subformula N_T rooted at T. This algorithm runs in time $O(|N|)$.*

Proof We prove this by induction on the height h of the subformula N_T.

Base. In this case, the formula N_T consists of a single literal x only. Then N_T has a single model with weight $w[x]$ when $x \notin P$. If $x \in P$, N_T has no models, thus its lower bound is $z + 1$.

Step. Assume the hypothesis holds for height $h - 1$. Let the root T of the subformula N_T be an OR node and $T_c \in \mathrm{CHD}(T)$ be a child of T. By the inductive hypothesis $l[T_c]$ is correct. Any model of N_{T_c} is a model of N_T. By the smoothness property, $vars(N_T) = vars(N_{T_c})$, $\forall T_c \in \mathrm{CHD}(T)$. Thus the minimum weights computed for each N_{T_c} take every variable of N_T into account. Therefore, we can choose the model of a child with the minimum weight, so $l[T] = \min_{T_c \in \mathrm{CHD}(T)}\{l[T_c]\} + w[T]$ is the minimum weight of any model of N_T.

Let T be an AND node. By the decomposability property, the children T_c of T are such that N_{T_c}, $T_c \in \mathrm{CHD}(T)$ have no variables in common. Therefore, we can choose any model from each child and concatenate them to get a model of T. The weight of the model of T will be $w[T]$ plus the sum of the weights of the models of its children. By the inductive hypothesis $l[T_c]$, $T_c \in \mathrm{CHD}(T)$ is correct, therefore the minimum weight of a model of N_T will be the sum of the minimum weights of the models of all N_{T_c}, $l[T] = \sum_{T_c \in \mathrm{CHD}(T)}(l[T_c]) + w[T]$ is correct.

Complexity. Every node of N is added to the priority queue $queue_{\uparrow}$ exactly once, as it is popped from the queue only after each of its children has been examined already. We use the priority queue without duplicates. The body of the loop at lines 10–15 runs in $|\mathrm{CHD}(T)|$ time for each node T it examines, so the time complexity of this is in $\sum_{T \in N(z)}(|\mathrm{CHD}(T)|) = O(|N|)$. The loop at lines 16–17 examines each node T exactly $|\mathrm{CHD}(T)|$ times. Therefore,

the time complexity of this is also in $\sum_{T \in N(z)}(|\text{CHD}(T)|) = O(|N|)$ and the total complexity of the algorithm is in $O(|N|)$. □

Lemma 2 *Let $N(z)$ be a weighted smooth* DNNF. *The algorithm* S-WDNNF-*upperbound returns a function u such that for each node T in $N(z)$, u gives the maximum weight of a model of the subformula N_T rooted at T may have in order to be part of a model of $N(z)$. This algorithm runs in time $O(|N|)$.*

Proof We show this by induction on the depth d of the node T.

Base. In this case T is the root node and the hypothesis holds trivially.

Step. Assume the hypothesis holds for every node of depth $d-1$. Let T be a node at depth d. Let $PRT(T)$ be the parents of T. By the inductive hypothesis, $u[T_p]$ is correct for each $T_p \in PRT(T)$. By the decomposability property, a model of N_T can be extended to a model of N_{T_p} for any T_p. However, not all models of N_T can be extended to a model of N_{T_p} that obeys $u[T_p]$. Let $u_{T_p}[T]$ be the upper bound on the weight of the models of N_T such that they are also models of N_{T_p}. A model of N_T with weight w can be extended to a model of the entire formula $N(z)$ if there exists at least one parent T_p of T such that $w \leq u_{T_p}[T]$, therefore $u[T] = \max(u_{T_p}[T])$.

Now consider a specific parent T_p of T. Let T_p be an OR node. Then, any model of N_T with weight w is also a model of N_{T_p} with weight $w + w[T_p]$. Thus $u_{T_p}[T] = u[T_p] - w[T_p]$. Let T_p be an AND node. Then, any model of N_T with weight r_T is combined with models of other children of T_p due to the decomposability property. This gives a model of N_{T_p} with weight $r_T + \sum_{T_{sb} \in \text{CHD}(T_p)\setminus\{T\}} r_{T_{sb}} + w[T_p]$. Therefore $u_{T_p}[T] = \max(u[T_p] - \sum_{T_{sb} \in \text{CHD}(T_p)\setminus\{T\}} r_{T_{sb}}) - w[T_p]$. Since $r_{T_{sb}} \geq l[T_{sb}]$ for all siblings, which by lemma 1 are correct, we have $u_{T_p}[T] = u[T_p] - \sum_{T_{sb} \in \text{CHD}(T_p)\setminus\{T\}} l[T_{sb}] - w[T_p]$.

Complexity. The algorithm performs a breadth first search over the nodes of N, and performs |CHD| operations for each of the nodes. We use a FIFO queue without duplicates. Therefore its total time complexity is in $O(|N|)$. □

Theorem 1 *Let $N(z)$ be a weighted smooth* DNNF. *Then, algorithm* S-WDNNF-*consistent correctly computes all the literals of $N(z)$ that may appear in a model of $N(z)$, given that the literals in P are set to false, in time $O(|N|)$.*

Proof The function u that is returned by S-WDNNF-upperbound is correct by Lemma 2, therefore $u[x] \geq w[x]$ if and only if there exists a model of $N(z)$ such that the literal x is true in this model. □

Example 4 Consider the conjunction of the Boolean function f and the SUM constraint from Example 3 where k equals 3 and all weights are unit. Suppose that the variables X_2 and X_3 is assigned to 1 and the cost variable z is equal to 2. In this case, an initial set of inconsistent literals with $N(2)$ is $P = \{(X_2 = 0, X_3 = 0)\}$. Figure 3(a) shows the result of a run of Algorithm 1. Figure 3(b) shows the result of an execution of Algorithm 2. The number in the subscript of a node or a leaf represents the value l while the number in the superscript represents u. For instance, the value two in the subscript of the root shows that the minimum weight of the model of $N(2)$ is two. The value two in the superscript of the root equals z. Note that $u < l$ for the literal $X_1 = 1$. This means that X_1 has to be assigned to the value 0.

Algorithm 1 Computing the lower bound for each node of a weighted s-DNNF

procedure s-WDNNF-LOWERBOUND($N(z)$, P)
  $queue_{\uparrow} = \emptyset$ ▷ Priority queue without duplicates inversely sorted by the depth of the node
  for each leaf x in N **do**
    if $x \in P$ **then**
      $l[x] = z + 1$
    else
      $l[x] = w[x]$
    for each parent T of x **do**
      push(T, $queue_{\uparrow}$)
  while $queue_{\uparrow} \neq \emptyset$ **do**
    $T = pop(queue_{\uparrow})$
    if T is $\vee$-node **then**
      $l[T] = \min_{T_c \in \mathrm{CHD}(T)}\{l[T_c]\} + w[T]$
    else if T is $\wedge$-node **then**
      $l[T] = \sum_{T_c \in \mathrm{CHD}(T)}(l[T_c]) + w[T]$
    for each parent T' of T **do**
      push(T', $queue_{\uparrow}$)
  return l

Algorithm 2 Computing the upper bound for each node of a weighted s-DNNF

procedure s-WDNNF-UPPERBOUND($N(z), l$)
  for each node T in N **do**
    $u[T] = -1$
  $u[T_{root}] = z$
  $queue_{\downarrow} = \{T_{root}\}$ ▷ FIFO queue without duplicates
  while $queue_{\downarrow} \neq \emptyset$ **do**
    $T = pop(queue_{\downarrow})$
    for each T_c in $\mathrm{CHD}(T)$ **do**
      if T is $\vee$-node **then**
        $u[T_c] = \max(u[T_c], u[T] - w[T])$
      else if T is $\wedge$-node **then**
        $u[T_c] = \max\Big(u[T_c], u[T] - \sum_{T_{sb} \in \{\mathrm{CHD}(T) \setminus \{T_c\}\}} l[T_{sb}] - w[T]\Big)$
      push(T_c, $queue_{\downarrow}$)
  return u

Algorithm 3 Computing the consistent literals of a weighted s-DNNF

procedure s-WDNNF-CONSISTENT($N(z)$, P)
  l = s-WDNNF-lowerbound($N(z)$, P)
  u = s-WDNNF-upperbound($N(z), l$)
  return $\{x | x$ is a literal in N where $u[x] \geq w[x]\}$

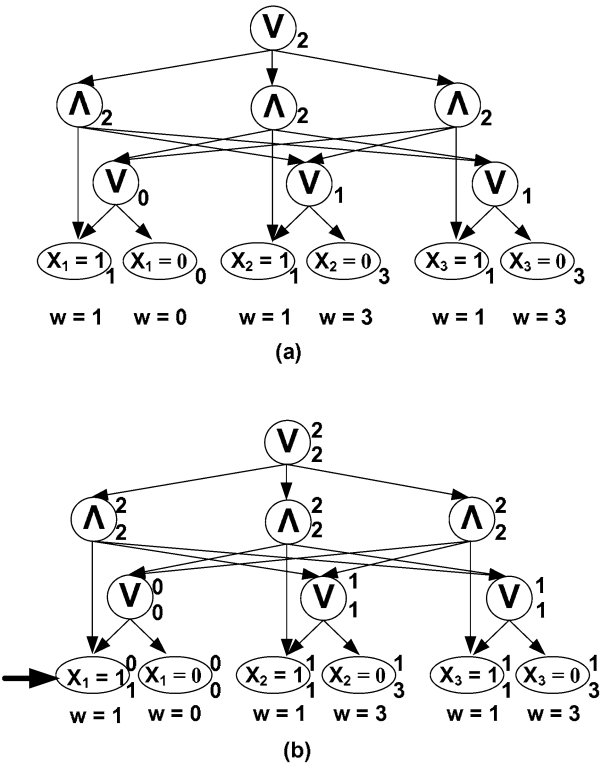

Fig. 3 The s-WDNNF graph produced by Algorithm 1 (**a**) and Algorithm 2 (**b**) on Example 4

3.1 The s-WDNNF constraint

In this section we introduce the s-WDNNF constraint. In general, any constraint can be expressed as a propositional formula, using an appropriate mapping of the multi-valued X variables to a set of propositional variables. Here, we use the direct encoding (Walsh 2000), which creates a propositional variable for each assignment $X_i = a$. We overload the notation $X_i = a$ to also denote the corresponding propositional variable, while for the negative literal we write $X_i \neq a \equiv \neg X_i = a$. We will make it clear from the context whether we mean an assignment to the CSP variable X_i or a propositional literal.

Definition 6 Let $N(z)$ be a s-WDNNF over the propositional variables $X_i = a$, $i = 1, \ldots, n$, $a \in D(X_i)$. The s-WDNNF$(N(z), [X_1, \ldots, X_n])$ constraint holds if and only if the direct encoding of an assignment X into the variables $X_i = a$ is a model of the $N(z)$.

It is easy to see that Algorithm 3 can be used to enforce DC on the s-WDNNF constraint. If a literal $X_i = a$ is not in the set P returned by Algorithm 3 then the value a can be pruned from the domain of variable X_i.

3.2 Decomposition of a s-WDNNF constraint

As an alternative to the filtering algorithm s-WDNNF-consistent, we propose a simple decomposition with which we can also detect inconsistent literals. A decomposition has several advantages. First, it is easy to add to any constraint solver. Second, decomposition

gives an efficient incremental propagator. Third, decomposition opens the door to advanced techniques like nogood learning and watched literals. The idea of the decomposition is to introduce arithmetic constraints to compute l and u.

For each node T, two integer variables are introduced to compute $l[T]$ and $u[T]$.

For each T we post a constraint to compute $l[T]$ from the lower bounds of its children $\mathrm{CHD}(T)$:

$$l(T) = \sum_{T_c \in \mathrm{CHD}(T)} l(T_c) + W[T] \quad \text{if } T \text{ is an AND-NODE}, \tag{1}$$

$$l(T) = \min_{T_c \in \mathrm{CHD}(T)} \{l(T_c)\} \quad \text{if } T \text{ is an OR-NODE} \tag{2}$$

For each node T, we post a constraint to compute $u[T]$ from the upper bounds of its parents $\mathrm{PRT}(T)$ and its siblings:

$$u(T) = \max\left\{ \max_{T_p \in \mathrm{PRT}_O(T)} \{u(T_p) - W[T_p]\}, \max_{T_p \in \mathrm{PRT}_A(T)} \left\{ u(T_p) - \sum_{T_{sb} \in \mathrm{CHD}(T_p) - \{T\}} l(T_{sb}) - W[T_p] \right\} \right\} \tag{3}$$

Finally, we introduce constraints to prune X_i. For each leaf T of the s-WDNNF that corresponds to the literal $X_i = a$, we introduce:

$$a \in D(X_i) \Rightarrow W[T] \leq l(T) \leq z, \tag{4}$$

$$a \notin D(X_i) \Leftrightarrow l(T) > z, \tag{5}$$

$$W[T] > u(T) \Rightarrow a \notin D(X_i) \tag{6}$$

As the maximal weight of a model is less than or equal to z we post:

$$u(T_{\mathrm{root}}) \leq z \tag{7}$$

Enforcing bounds consistency on constraints 1–7 will set the lower bounds of variables $l(T)$ to the minimal weight of a model of the formula rooted at T and the upper bounds of variables $u(T)$ to the maximum weight that a model of a formula rooted at T may have so that it can be extended to a model of the s-WDNNF.

Bounds propagation on this decomposition enforces domain consistency on the s-WDNNF constraint. If we invoke constraints in the decomposition in the same order as we compute functions l and u by Algorithms 1–2 then we set lower bounds of variables $l(T)$ to $l[T]$ and upper bounds to $u[T]$. This takes $O(|N(z)|)$ time.

We can speed up propagation by recognizing when constraints are entailed. If $l(T) > u(T)$ holds for a node T then constraints (1) and (2), (3) are entailed. To model entailment we augment each of these constraints in such a way that if $l(T) > u(T)$ holds then the corresponding constraints are not invoked by the solver.

3.3 Counting in s-WDNNF

The equivalent of the class NP for counting problems is $\#P$. This is the class of counting problems for which a solution can be recognized in polynomial time. This class is at least

as hard as NP, because an exact non-zero count of the solutions of a problem implies that at least one solution exists. For example, $\#3SAT$, the counting version of $3SAT$, is in $\#P$. Surprisingly, the counting version of many tractable decision problems is $\#P$-complete, e.g. $\#2SAT$ or $\#Horn-SAT$.

It is easy to see that model counting for NNF is $\#P$-complete, as any propositional formula can be encoded into an unrestricted NNF. As VNNF is a generalization of NNF, this result also holds for VNNF (Fargier and Marquis 2007). In Darwiche and Marquis (2002), it was shown that counting is also NP-hard for DNNF, but polynomial for SD-DNNF.

In this section, we make the intractability result for DNNF (and therefore its generalization VDNNF) more precise by showing that it is actually a $\#P$-complete problem.

We recall first that given a non-deterministic finite automaton(NFA), counting the number of strings of a specific length that it accepts is $\#P$-complete (Gore et al. 1997). We now show that the strings of length n accepted by an NFA $\mathcal{A}$ can be encoded in a DNNF of size $O(n|\mathcal{A}|)$.

Lemma 3 *There exists a function D, such that given an NFA $\mathcal{A}$ and an integer n, $D(\mathcal{A}, n)$ is a* DNNF *and there exists a bijection between models of $D(\mathcal{A}, n)$ and strings of length n that are accepted by $\mathcal{A}$.*

Proof Let $\mathcal{A}'$ be an NFA without ε-transitions that is equivalent to $\mathcal{A}$. This transformation is possible with at most a polynomial increase in the size of the NFA (Hopcroft and Ullman 1990). Following the results of Pesant (2004) and Quimper and Walsh (2006), we can construct an *unfolded* NFA $\mathcal{A}'_n$ that accepts exactly the strings of length n that are accepted by $\mathcal{A}'$. $\mathcal{A}'_n$ is a directed acyclic graph such that $|\mathcal{A}'_n| = O(n|\mathcal{A}'|)$. Moreover, $\mathcal{A}'_n$ is a *layered* automaton: the states are partitioned into $n+1$ layers, such that the first layer contains the starting state s and the $(i+1)$th layer contains all states that are reachable after processing i symbols of any string.

We now transform $\mathcal{A}'_n$ to a DNNF D. For each symbol a in Σ and position i we introduce a variable with literals $X_i = a$ and $X_i \neq a$. For each non-accepting state $s \in \mathcal{A}'_n$ we create an OR-NODE $n(s)$ in D, while each accepting state is mapped to the TRUE node. For every transition from state s_i^k to s_j^{k+1} on symbol a, we create an AND-NODE which is a child of the node s_i^k and has children $n(s_j^{k+1})$ and $X_k = a$, $X_k \neq b$ for all $b \neq a$. The result is a DNNF rooted at node $n(s)$.

It is easy to see that the models of D are exactly the strings of length n that are accepted by $\mathcal{A}$. □

By Lemma 3 and the result that counting the number of strings a specific length accepted by NFA is $\#P$-complete we get

Corollary 1 #DNNF *is $\#P$-complete.*

4 The weighted GRAMMAR constraint

In this section we introduce a weighted version of the GRAMMAR constraint. The motivation behind WEIGHTEDGRAMMAR is threefold: first, it allows us to model over-constrained problems and problems with preferences; second, it allows us to propagate the conjunction

of a GRAMMAR and the objective function of an optimization problem; third, by mapping WEIGHTEDGRAMMAR onto WNNF, it allows for more succinct representations of other constraints.

Definition 7 The WEIGHTEDGRAMMAR$(G, W, z, [X_1, \ldots, X_n])$ constraint, where G is a GRAMMAR, W is a function that maps productions of G to weights, z is a cost variable and $X_1, \ldots, X_n$ are decision variables, holds if and only if an assignment X forms a string belonging to the grammar G and the minimal weight of a derivation of X less than or equal to z. The weight of a derivation is the sum of the production weights used in the derivation.

We can further refine the granularity of the weights by making W a function with *three* arguments $W(P, i, j)$, so that it gives the weight of using the production P to produce the substring starting at position i with length j. This is similar to the conditional productions used in Quimper and Walsh (2007) and the cost definition used in the COSTREGULAR constraint (Demassey et al. 2006). In accordance with these previous uses, we call these *conditional weights*. In fact, we will use conditional weights in our experimental setup. For simplicity, however, we use the simpler definition of unconditional weights in the rest of this section.

Example 5 Consider again the context-free grammar G from Example 1. Suppose that the weights of all productions except $B \rightarrow b$ are zero and the weight of the production $B \rightarrow b$ is one. Let the domains of the variables X be $D(X_1) = \{a\}$, $D(X_2) = \{a, b\}$, $D(X_3) = \{b\}$ and z be smaller than or equal to 1. Then enforcing DC on WEIGHTEDGRAMMAR$(G, W, 1, [X_1, X_2, X_3])$ prunes b from X_2 as the only support is the sequence aab. The sequence abb that is a possible solution in Example 1 is not a valid solution, because its minimum weight derivation is 2.

Using conditional weights, we can see that we can use the WEIGHTEDGRAMMAR constraint to propagate the conjunction of a GRAMMAR constraint and a linear objective function. Indeed, consider a constraint $C \equiv$ GRAMMAR$(G, [X_1, \ldots, X_n])$ and an objective function $f = \sum_{i=1,\ldots,n, j=1,\ldots,d} w_{ij}(X_i = j)$, where $(X_i = j)$ is a propositional literal. Further suppose that G has productions of the form $T_a \rightarrow a$ for each terminal a (this holds for all grammars in Chomsky Normal Form). We can then use the constraint WEIGHTEDGRAMMAR$(G, W, z, [X_1, \ldots, X_n])$, where $W(T_a, i, 1) = w_{ia}$ and $W(P, i, j) = 0$ for all other productions P. This constraint propagates the conjunction of C and f over the variables $X_1, \ldots, X_n$. We can then replace the objective function f with $f' = z$ to exploit this stronger propagation.

4.1 Propagator based on the CYK parser

First we give a propagator for the WEIGHTEDGRAMMAR constraint based on a natural extension of the CYK parser to probabilistic CFGs (Ney 1991) combined with the DC filtering algorithm for the GRAMMAR constraint (Quimper and Walsh 2006). This algorithm implicitly constructs an S-WDNNF. Therefore, it is easy to recast it in terms of weight propagation in an S-WDNNF and prove its correctness based on the results of the previous section.

Algorithm 4 The weighted CYK propagator

```
1: procedure WCYK-ALG(G, W, z, [X_1, ..., X_n])
2:     for i = 1 to n do                                   ▷ Construct a dynamic programming table V
3:         V[i, 1] = {A | A → a ∈ G, a ∈ D(X_i)}
4:     for j = 2 to n do
5:         for i = 1 to n − j + 1 do
6:             V[i, j] = ∅;
7:             for k = 1 to j − 1 do
8:                 V[i, j] = V[i, j] ∪ {A | A → BC ∈ G, B ∈ V[i, k], C ∈ V[i + k, j − k]}
9:     for j = 1 to n do                                   ▷ Compute lower bounds
10:        for i = 1 to n − j + 1 do
11:            for each A ∈ G do
12:                l[i, j, A] = z + 1; u[i, j, A] = −1;
13:    for i = 1 to n do
14:        for A ∈ V[i, 1] s.t A → a ∈ G, a ∈ D(X_i) do
15:            l[i, 1, A] = min{l[i, 1, A], W[A → a]};
16:    for j = 2 to n do
17:        for i = 1 to n − j + 1 do
18:            V[i, j] = ∅;
19:            for k = 1 to j − 1 do
20:                for each A → BC ∈ G s.t. B ∈ V[i, k], C ∈ V[i + k, j − k] do
21:                    l[i, j, A] = min{l[i, j, A], W[A → BC] + l[i, k, B] + l[i + k, j − k, C]};
22:    if S ∉ V[1, n] ∨ l[1, n, S] > z then
23:        return "Unsatisfiable";
24:    mark (1, n, S);
25:    u[1, n, S] = z;                                     ▷ Compute upper bounds
26:    for j = n downto 2 do
27:        for i = 1 to n − j + 1 do
28:            for A such that (i, j, A) is marked do
29:                for k = 1 to j − 1 do
30:                    for each A → BC ∈ G s.t. B ∈ V[i, k], C ∈ V[i + k, j − k] do
31:                        if W[A → BC] + l[i, k, B] + l[i + k, j − k, C] > u[i, j, A] then
32:                            continue;
33:                        mark (i, k, B); mark (i + k, j − k, C);
34:                        u[i, k, B] = max{u[i, k, B], u[i, j, A] − l[i + k, j − k, C] − W[A → BC]};
35:                        u[i + k, j − k, C] = max{u[i + k, j − k, C], u[i, j, A] − l[i, k, B] − W[A → BC]};
36:    for i = 1 to n do
37:        D(X_i) = {a ∈ D(X_i) | A → a ∈ G, (i, 1, A) is marked and W[A → a] ≤ u[i, 1, A]};
38:    return "Satisfiable";
```

We assume that G is in Chomsky normal form and has a single starting symbol S. The algorithm works in two stages. In the first, it constructs a dynamic programing table $V[i, j]$ where an element A of $V[i, j]$ is a potential non-terminal that generates a substring $[X_i, \ldots, X_{i+j-1}]$. It then computes a lower bound $l[i, j, A]$ on the minimal weight of a derivation from A. In the second stage, it moves from $V[1, n]$ to the bottom of table V. For an element A of $V[i, j]$, it computes an upper bound $u[i, j, A]$ on the maximal weight of a derivation from A of a substring $[X_i, \ldots, X_{i+j-1}]$. that can be extended to a string from the grammar. It marks the element A if and only if $l[i, j, A] \leq u[i, j, A]$.

We present the pseudo-code in Algorithm 4. Lines 9–12 initialize l and u. Lines 2–21 compute the first stage, whilst lines 24–35 compute the second stage. Finally, we prune inconsistent values from domains of variables X in lines 36–37. Theorem 2 proves that Algorithm 4 enforces domain consistency on the WEIGHTEDGRAMMAR$(G, W, z, [X_1, \ldots, X_n])$ constraint in time $O(n^3|G|)$.

Example 6 Consider the context-free grammar G from Example 5. The dynamic programming table produced by Algorithm 4 is presented in Fig. 4. The number in the subscript of a non terminal represents the value l and the number in the superscript of a non terminal represents the value u. For instance, the value in the subscript of B_1^1 in the right bottom cell

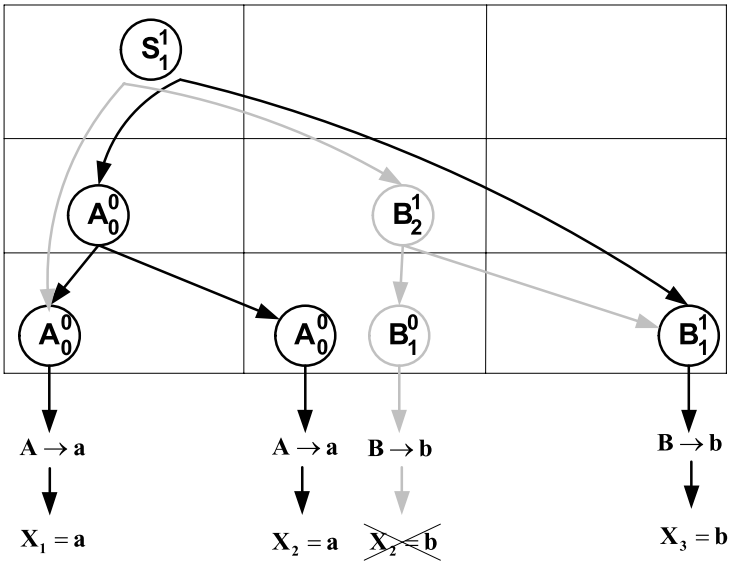

Fig. 4 Dynamic programming table produced by Algorithm 4. *Gray lines* show pruning caused by the weighted GRAMMAR constraint compared to the non-weighted GRAMMAR constraint

of the table shows that the minimum weight of the derivation from B of length one starting at position 3 is one.

Gray lines show unmarked non terminals. This also shows the pruning caused by the weighted GRAMMAR constraint compared to the non-weighted GRAMMAR constraint. Outgoing arcs from non terminals correspond to different derivations from this non terminal. For instance, the non terminal S_1^1 in the top left cell of the table has two possible derivations in the non weighted case and a single derivation in the weighted case.

To see the correspondence between Algorithm 4 and weight propagation in an s-WDNNF, we use the construction of an AND/OR graph for the GRAMMAR constraint proposed by Quimper and Walsh (2006), given the dynamic programming table V[1] obtained by Algorithm 4. We modify this construction slightly in order to construct a s-WDNNF rather than an AND/OR graph and thus make it applicable to the weighted case.

Consider the constraint $C = \text{WEIGHTEDGRAMMAR}(G, W, z, [X_1, \ldots, X_n])$. We construct a s-WDNNF $N(z)$. We create in N leaf nodes labeled with $X_i = a$ and $X_i \neq a$ and weight 0 for each $i \in [1, n], a \in D(X_i)$. For each non-terminal A, $A \in V[i, j]$, $i, j = 1, \ldots, O(n)$ we create an OR node labeled with $n(i, j, A)$ with weight 0. For each production $A \to BC$, $A \in V[i, j]$, $B \in V[i, k]$, $C \in V[i + k, j - k]$, we create an *AND* node labeled with $n(i, j, k, A \to BC)$ and weight $W[A \to BC]$. For each production of the form $A \to a$, $A \in V[i, 0]$, we create the AND-NODE $n(i, 0, 0, A \to a)$ with weight $W[A \to a]$.

Each OR-node $n(i, j, A)$ is a parent of the *AND*-nodes $n(i, j, k, A \to BC)$, $k = 1, \ldots, j - 1$. Each *AND*-node $n(i, j, k, A \to BC)$ is a parent of the OR-nodes $n(i, k, B)$ and $n(i + k, j - k, C)$. Each AND-NODE $n(i, 0, 0, A \to a)$ is a parent of the nodes labeled with $X_i = a$, $X_i \neq b$ for all $b \neq a$. Clearly, if C is satisfiable then $S \in V[1, n]$ and therefore

[1]Note that we ignore whether a specific entry in the table is marked or not.

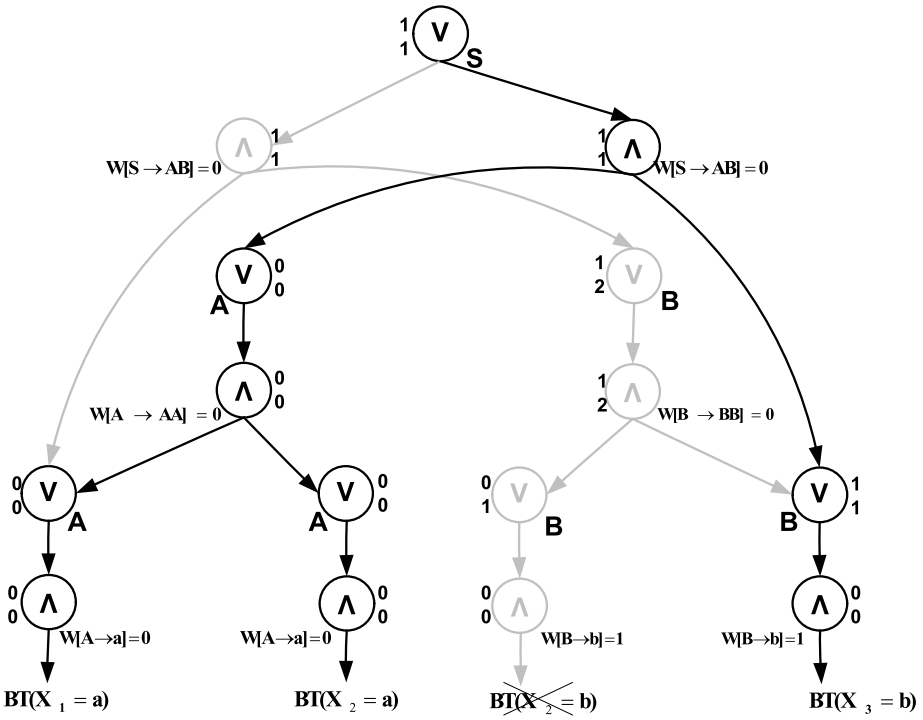

Fig. 5 S-WDNNF graph that corresponds to the dynamic programming table in Fig. 4. *Gray lines* show pruning caused by the weighted GRAMMAR constraint compared to the non-weighted GRAMMAR constraint

we have created a node $n(1, n, S)$. In this case, the resulting graph is a WNNF with root $n(1, n, S)$. We call this graph $WCYK_{NNF}(G, W, z, [X_1, \ldots, X_n])$.

Example 7 Consider the context-free grammar G from Example 5. The graph $WCYK_{NNF}$ is presented in Fig. 5. OR-nodes are labeled with corresponding non terminals from the table. As before, gray lines show pruning caused by the weighted GRAMMAR constraint compared to the non-weighted GRAMMAR constraint. We denote $BT(X_i = a)$ an AND-NODE with $O(d)$ children: $X_i = a$, $X_i \neq b, b \in D(X_i) \setminus \{a\}$.

Each node T in the graph is annotated with two numbers and the weight of this node. The number in the subscript of a node shows the value $l[T]$ computed by Algorithm 4. The number in the superscript of a node shows the value $u[T]$ computed by Algorithm 4. If the weight of T equals 0 we omit it to reduce clutter.

We show first that the construction does indeed create a smooth and decomposable WNNF, so that the results of the previous section apply.

Lemma 4 *The graph* $WCYK_{NNF}(G, W, z, [X_1, \ldots, X_n])$ *is an* S-WDNNF *if* WEIGHTEDGRAMMAR$(G, W, z, [X_1, \ldots, X_n])$ *is satisfiable.*

Proof Determinism. Consider any AND-NODE $n(i, j, k, A \to BC)$ of the graph. By construction, the children of this node are the OR-NODES $n(i, k, B)$ and $n(i + k, j - k, C)$.

But the leaves that are reachable from $n(i, k, B)$ are literals that correspond to the CSP variables $X_i \dots X_{i+k}$, while the leaves that are reachable from $n(i+k, j-k, C)$ are literals that that correspond to the CSP variables $X_{i+k+1} \dots X_{i+j}$. Therefore, $vars(n(i, k, B)) \cap vars(n(i+k, j-k, C)) = \emptyset$ as required.

Smoothness. By construction, the children of each OR-NODE $n(i, j, A)$ are the AND-NODES $n(i, j, k, A \to BC)$ for each production $A \to BC$ in G. But each of these nodes can reach exactly the leaves that correspond to the CSP variables $X_i \dots X_{i+j}$. Moreover, if a leaf $X_i = a$ is reachable, so are all the leaves $X_i \neq b$, $b \neq a$. Therefore, each of the AND-NODES that are the children of $n(i, j, A)$ can reach all of the literals $X_l = d$ for $l \in [i, i+j]$, $d \in D(X_l)$, as required. □

We show next that the above construction is a compilation of exactly the solutions of the WEIGHTEDGRAMMAR constraint.

Lemma 5 *There exists a weight-preserving bijection between models of the* S-WDNNF $N(z) = WCYK_{NNF}(G, W, z, [X_1, \dots, X_n])$ *and solutions of the constraint* $C = \text{WEIGHTEDGRAMMAR}(G, W, z, [X_1, \dots, X_n])$ *if* C *is satisfiable.*

Proof We construct first a reversible function f from instantiations of $N(z)$ to solutions of C and then show that it preserves solutions, so that it is a bijection between models of $N(z)$ and solutions of C.

Let I be a model of $N(z)$. Then we construct an instantiation $f(I)$ of the variables $X_1 \dots X_n$. Note first that each literal $X_i = a$ in N always appears as a child of an AND-NODE with siblings $X_i \neq b$ for all $b \neq a$. Thus, in any model of $N(z)$, exactly one of $X_i = a$ can be true and we can map the assignment to these variables to a unique assignment to the variable X_i for all i. Similarly, from each assignment to the variables $X_1 \dots X_n$, we construct an instantiation $f^{-1}(\mathbf{X})$ of the variables of $N(z)$ such that if $X_i = a$ then the literal $X_i = a$ is true and all the literals $X_i = b$, $b \neq a$ are false.

We show first that $f(I)$ is a solution of the constraint $\text{GRAMMAR}(G, [X_1, \dots, X_n])$ if and only if I is a model of the unweighted NNF N.

By the correctness of the CYK parser, a non-terminal symbol A is in a cell $V[i, j]$ if and only if the substring $i \dots i+j$ of $\mathbf{X} = f(I)$ can be generated from A. We show that the OR-NODE $T = n(i, j, A) \in N$ is true given I if and only if A is in the cell $V[i, j]$.

We show this by induction on the height h of the WNNF N_T.

Base. In this case, $h = 1$ and the WNNF N_T consists of a single literal $X_i = a$. Then, the node $n(i, 0, A \to a)$ is true if and only if $A \in V[i, 1]$.

Step. Assume the inductive hypothesis holds for $h-1$. Then, let T be a node, such that the height of N_T is h. If h is even then T has to be an AND-NODE and the hypothesis holds trivially. If h is odd, then T has to be an OR-NODE $n(i, j, A)$. Assume T is true. Then, at least one of its children $n(i, j, k, A \to BC)$ is true. But $n(i, j, k, A \to BC)$ is true if the OR-NODES $n(i, k, B)$ and $n(i+k, j-k, C)$ are true, which by the inductive hypothesis means that $B \in V[i, k]$ and $C \in V[i+k, j-k]$. This in turn means that CYK places A in $V[i, j]$.

Conversely, assume $A \in V[i, j]$. Then, there exists a production $A \to BC$ and k such that $B \in V[i, i+k]$ and $C \in V[i+k, j-k]$. But this means by the inductive hypothesis that the OR-NODES $n(i, i+k, B)$ and $n(i+k, j-k, C)$ are true and therefore the AND-NODE $n(i, j, k, A \to BC)$ is also true, which suffices to make the OR-NODE $T = n(i, j, A)$ true.

Because the computation of lower bounds by the WCYK algorithm is identical to the computation of weights in a WNNF, we can show that $f(I)$ is a solution of the constraint WEIGHTEDGRAMMAR$(G, W, z, [X_1, \ldots, X_n])$ if and only if I is a model of $N(z)$. The proof is analogous to the proof in the unweighted case. □

Theorem 2 *WCYK-alg enforces DC on the* WEIGHTEDGRAMMAR$(G, W, z, [X_1, \ldots, X_n])$ *constraint in time* $O(n^3|G|)$.

Proof Algorithm 4 implicitly performs the S-WDNNF-consistent algorithm on $WCYK_{NNF}(G, W, z, [X_1, \ldots, X_n])$. Specifically, lines 9–21 of WCYK-ALG correspond to the algorithm S-WDNNF-lowerbound, lines 24–35 of WCYK-ALG correspond to the algorithm S-WDNNF-upperbound and lines 36–37 of WCYK-ALG correspond to line 4 of the algorithm S-WDNNF-consistent.

Complexity. The time complexity of the algorithm is dominated by lines 24–35 and is therefore in $O(|G|n^3)$. □

4.2 Propagator based on the Earley parser

We now give a propagator for the WEIGHTEDGRAMMAR constraint based on the Earley parser (Earley 1970). The Earley parser is a dynamic programming based top-down parsing algorithm for arbitrary context free grammars. It is often preferred over the CYK algorithm because for many classes of grammars it has provably better performance. With regular or linear grammars, it runs in linear time. With unambiguous grammars, it runs in quadratic time. With grammars of bounded ambiguity, it runs in time that is greater than quadratic but less than cubic.

We first show a natural extension of the Earley parser to a propagator for the WEIGHTEDGRAMMAR constraint. It has already been shown that a propagator for the GRAMMAR constraint can be built as an extension of the Earley parser, so we simply modify the existing propagator to handle weights. Somewhat surprisingly, we show that the Earley propagator also implicitly constructs a S-WDNNF during the parsing phase. We make this construction explicit and then use the resulting S-WDNNF to perform pruning.

The pseudocode for this propagator is shown in Algorithm 5. The parser works by populating tables $C[j]$ of states of the form $(s \to \alpha \bullet v, i)$, where α is a sequence of terminals and non-terminals,. The form $s \to \alpha \bullet v$ is called a "dotted production". If a state $(s \to \alpha \bullet v, i)$ is in the table $C[j]$, it means that the substring $i \ldots j$ can be parsed using α. The basic operations performed by this algorithm are the same as for the Earley parser: scanning of terminals, prediction of productions of non-terminals and completion of productions. The procedure starts by predicting the production $s \to \bullet u$ and completes successfully when no state can be added to any of the tables and $C[n]$ contains $s \to u \bullet$.

In order to perform propagation, we concurrently construct a S-WDNNF and compute the function l which gives for each node T the minimum weight of any model of the WNNF N_T. To simplify the presentation, we do this by using the quaternary function N which constructs a WNNF node T (first argument) and simultaneously sets its lower bound $l[T]$ (second argument), its upper bound $u[T]$ (third argument) and its weight $w[T]$ (fourth argument). For each prediction operation, we construct a TRUE node, while for each scanning or completion we construct an AND-NODE. Each node is associated with the state that generated it, so if there already exists an AND-NODE for a specific state, it means we have generated the same state in two different ways, thus in the ADD procedure (shown in Algorithm 6), we construct an OR-NODE that is the parent of the previous nodes for this

Algorithm 5 The weighted *Earley* propagator

```
1: procedure EARLEY-ALG(G, W, z, [X_1, ..., X_n])
2:     for j = 1 to n do
3:         C[i] = {}
4:     push(queue, {s → •u}, 0, N(TRUE, 0, −1, 0))
5:     for i = 1 to n do
6:         for state ∈ C[i] do
7:             push(state, queue)
8:         while queue ≠ ∅ do
9:             (r, j, S) = pop (queue)
10:            add ((r, j, S), C[i])
11:            if r = (u → v•) then                                   ▷ Completion
12:                for each (w → ... • u ..., k, T) ∈ C[j] do
13:                    c = l[S] + l[T] + W[u → v]
14:                    add ((w → ... u • ..., k, N(S ∧ T, c, −1, W[u → v])), queue)
15:            else if (i ≤ n) and r = (u → ... • v ...) and v ∈ D(X_i) then   ▷ Scanning
16:                add ((u → ... v • ..., j, N(S ∧ BT({X_i = v}), l[S], −1, 0)), C[i + 1])
17:            else if r = (u → ... • v ...) and v is non terminal then        ▷ Prediction
18:                for each v → w ∈ G such that (v → •w, i, ∅) ∉ C[i] ∪ queue do
19:                    push ((v → •w, i, N(TRUE, 0, −1, 0)), queue)
20:        if C[i] = 0 then
21:            return "Unsatisfiable"
22:    if (s → u•, 0, S) ∈ C[n] then
23:        if l[S] > z then
24:            return "Unsatisfiable"
25:        else
26:            u = S-WDNNF-upperbound(S(z), l)
27:            for i = 1 to n do
28:                D(X_i) = {a | u[X_i = a] ≥ 0};
29:    else
30:        return "Unsatisfiable"
```

Algorithm 6 add $((a, k, S), q)$

```
1: procedure ADD((a, k, S), q)
2:     if ∃(a, k, S') ∈ q then
3:         q = replace((a, k, S'), (a, k, N(S ∨ S', min{l[S'], l[S]}, −1, 0)), q);
4:     else
5:         push((a, b, S, w), q)
```

state. The weight of every OR-NODE is 0, while the weight of an AND-NODE is set to the weight of the production that generated it. The lower bound of each node is set according to the rules of Definition 5, so this computation correctly computes the minimum weight of any solution of the WEIGHTEDGRAMMAR constraint. Finally, in order to ensure that the resulting WNNF is smooth (see Lemma 6), the leaf node $X_i = a$ is always generated in conjunction with the leaves $X_i \neq b$, $b \neq a$. We denote this construction by $BT(\{X_i = a\})$.

Note that we require a single unweighted starting production $s \rightarrow u$. This is not a significant limitation as we can insure this by renaming the starting symbol S of any grammar G to S' and introducing the production $S \rightarrow S'$ with weight 0. The new grammar G' satisfies this condition and is bigger than G by exactly one production.

Example 8 Consider the context-free grammar G from Example 5. The S-WDNNF graph produced by Algorithm 5 is presented in Fig. 6.

Each node T in the graph is annotated with two numbers, the weight of this node and the pair $(v \rightarrow \alpha \bullet \ldots, i)$, where i is the index of $C[i]$ where the production $v \rightarrow \alpha \bullet \ldots$ was generated. The number in the subscript of a node represents the value l, the number in

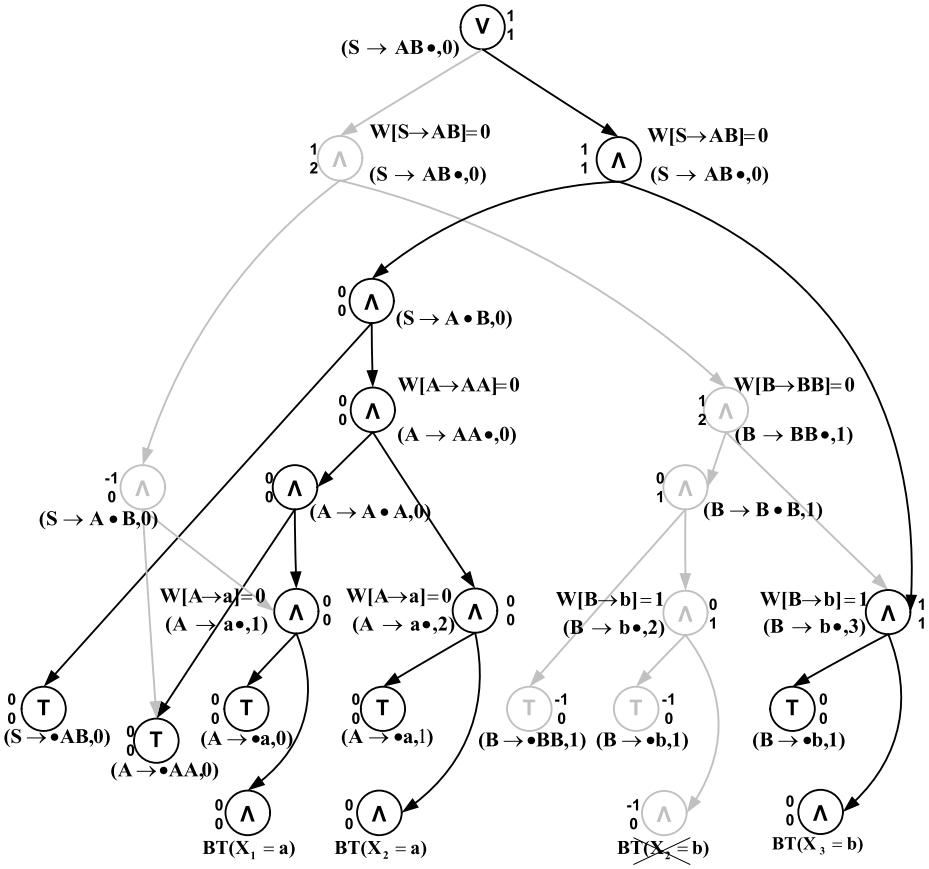

Fig. 6 The S-WDNNF graph produced by Algorithm 5 on Example 5. *Gray lines* show pruning caused by the weighted GRAMMAR constraint compared to the non-weighted GRAMMAR constraint

the superscript of a node represents the value u. If T is not labeled with the weight then $w[T] = 0$.

Again, gray lines show pruning caused by the weighted GRAMMAR constraint compared to the non-weighted GRAMMAR constraint.

The structure S produced at line 23 of Algorithm 5 is a WNNF, which we call *Earley NNF*$(G, W, z, [X_1, \ldots, X_n])$. We show first that it is actually a S-WDNNF.

Lemma 6 *The graph* $N(z) = Earley_{NNF}(G, W, z, [X_1, \ldots, X_n])$ *is a* S-WDNNF.

Proof Determinism. Consider any AND-NODE $Q \in N(z)$. A conjunction can be created in either line 14 or line 16 of Algorithm 5.

In the former case, Q is a conjunction of nodes S and T. But S is the set of literals that are supported by the production $w \to \ldots \bullet u \ldots$ at position j, starting at position k. This means that the literals that can be reached from S are all the literals of the CSP variables $X_k \ldots X_{j-1}$. Similarly, from T we can reach all the literals of the CSP variables $X_j \ldots X_i$.

But these sets are distinct, therefore the AND-NODE $S \wedge T$ satisfies the decomposability property.

The latter case can be seen to be an instance of the first case in which we advance the dot in the production by using a terminal rather than a non-terminal. Since a terminal can support a single position only, the decomposability property is satisfied.

Smoothness. Consider any OR-NODE $Q \in N(z)$. Disjunctions can only be created in Algorithm 6, called from either line 14 or line 16 of Algorithm 5. By the same argument as for decomposability, we see that both sets S and S' support variables $X_k \dots X_i$. Therefore the smoothness property is satisfied. □

Lemma 7 *There exists a weight-preserving bijection between models of the* S-WDNNF $N(z) = Earley_{NNF}(G, W, z, [X_1, \dots, X_n])$ *and solutions of the constraint* $C =$ WEIGHTEDGRAMMAR$(G, W, z, [X_1, \dots, X_n])$ *if* C *is satisfiable.*

Proof We use the reversible function f that we introduced in Lemma 5 to map instantiations of $N(z)$ to assignments to the variables of C. We show that it preserves solutions.

By the correctness of the *Earley* parser, the state $(v \to \alpha \bullet \dots, i, S)$, where α is a sequence of terminals and non-terminals, is generated in step j if and only if a state of the form $(v \to \bullet\alpha \dots, i)$ is generated and the substring $i \dots j$ can be parsed by α. We generate an AND-NODE each time that a state is generated and connect them with an OR-NODE if the state is generated more than once. Suppose that these nodes are labeled with the pair $(v \to \alpha \bullet \dots, i)$. We show that a node T labeled with $(v \to \alpha \bullet \dots, i)$ is true given I if and only if the production $v \to \alpha \dots$ can parse the substring $i \dots j$ of $\mathbf{X} = f(I)$ using α.

We show this by induction on the height h of the formula N_T rooted at T.[2] Note first that any node T such that N_T has height 1 can only be a node generated by prediction, if it has height 2 it can only be generated by scanning and if it has greater height then it can be generated by scanning or completion. However, scanning is a special case of completion and so we treat them uniformly for $h > 2$.

Base. In this case, $h = 2$ and T can only be a node labeled with $(v \to w \bullet \dots, i)$, where w is a terminal a. This node is generated by scanning, and has children $BT(\{X_i = a\})$ and $(v \to \bullet w \dots, i - 1)$, which is a node generated by prediction and is thus true. Therefore we only need to show that the state $(v \to w \bullet \dots, i)$ is generated if and only if the node $BT(\{X_i = a\})$ is true. But $X_i = a$, and this holds.

Step. Let the hypothesis hold for height $h - 1$ and let T be an node such that the height of N_T is h. Let the label of T be $(v \to \alpha\ w \bullet \dots, i)$. Assume T is an OR-NODE. Then it has the same label as any of its children. Since the children are at level $h - 1$, this means that the state $(v \to \alpha\ w \bullet \dots, i)$ is generated if and only if one of the children it true. Assume T is an AND-NODE. Then it has to be a node generated by completion, with children $(w \to q\bullet, k)$, $i < k < j$ and $(v \to \alpha \bullet w \dots, i)$. By the inductive hypothesis, each of these is true if and only if the substring $i \dots k - 1$ can be generated by α and the substring $k \dots j$ can be generated by w. But T is true if and only if both its children are true, which can be the case if and only if the substring $i \dots j$ can be generated by $\alpha\ w$. This then means that the state $(v \to \alpha\ w \bullet \dots, i)$ will be generated, as required.

Because the computation of lower bounds by the *Earley* algorithm is explicitly identical to the computation of weights in the constructed S-WDNNF, we see that $f(I)$ is a solution of the constraint WEIGHTEDGRAMMAR$(G, W, z, [X_1, \dots, X_n])$ if and only if I is a model of $N(z)$. □

[2]For simplicity, we consider the height of the formula $BT(\{X_i = a\})$ to be 1.

Theorem 3 *Algorithm 5 enforces DC on the* WEIGHTEDGRAMMAR$(G, W, z, [X_1, \ldots, X_n])$ *constraint in time* $O(n^3|G|^2)$.

Proof Correctness. This follows from Lemmas 7, which means that the lower bounds are computed correctly, and 6, which implies that the upper bounds are computed correctly.

Complexity. The time complexity of the Earley parser is $O(n^3|G|^2)$. The overhead of constructing a s-WDNNF is constant at each step, as exactly one node is generated at each step. Thus, the complexity of lines 2–21 is $O(n^3|G|^2)$. By the same argument, the number of nodes of the constructed s-WDNNFis $O(n^3|G|^2)$. Moreover, this s-WDNNFis a binary DAG, so the number of edges is also $O(n^3|G|^2)$. The cost of the rest of the algorithm is dominated by the call to s-WDNNF-upperbound, whose cost is linear in the size of the s-WDNNF, thus $O(n^3|G|^2)$. □

5 Decomposition of the weighted GRAMMAR constraint

As was shown in Sects. 4 and 4.2, these filtering algorithms generate a s-WDNNF. The size of the s-WDNNFproduced by the WCYK-alg is smaller by a factor of $O(|G|)$, therefore we use it to construct a decomposition of the WEIGHTEDGRAMMAR constraint.

Lemma 8 *The size of the* s-WDNNF$N(z) = WCYK_{NNF}(G, W, z, [X_1, \ldots, X_n])$ *is in* $O(n^3|G|)$.

Proof We show that the number of edges in $WCYK_{NNF}(G, W, z, [X_1, \ldots, X_n])$ is $O(n^3|G|)$.

Observe first that in order to enforce the smoothness property on $N(z)$ we replace each occurrence of a literal $X_i = a$ with the conjunction of it and the literals $X_i \neq b$, $b \neq a$. The total size of these subgraphs over all literals is $n|\Sigma| \leq n|G|$. This does not affect the size of the rest of the DAG.

By assumption the grammar G is in Chomsky Normal Form. Therefore, any AND-NODE in the graph has at most two children. Let P be the number of non terminals and $f(P)$ be the number of productions of the form $P \to AB$. Then the number of OR-NODESin the graph equals the number of non terminals in V. The number of children of an OR-NODEfor a non-terminal P is $O(n)f(P) = nO(|G|)$. But over all non-terminals in G, $\sum_{P \in G} f(P) = |G|$, therefore the total number of children of all OR-NODESin a cell of V over all non-terminals is $O(n)\sum_{P \in G} = O(n|G|)$. Hence, the total number of edges in the graph is $O(n^3|G|)$. □

If the constraint propagator invokes constraints in the decomposition in the same order as we compute the table V, this takes $O(n^3|G|)$ time. For simpler grammars, propagation is faster. For instance, as in the unweighted case, it takes just $O(n|G|)$ time on a regular grammar.

6 The soft GRAMMAR constraint

We can use the WEIGHTEDGRAMMAR constraint to encode a soft version of GRAMMAR constraint which is useful for modelling over-constrained problems. The soft GRAMMAR$(G, z, [X_1, \ldots, X_n])$ constraint holds if and only if the string $[X_1, \ldots, X_n]$ is at most distance z from a string in G. We consider both Hamming and edit distances. We encode the soft GRAMMAR$(G, z, [X_1, \ldots, X_n])$ constraint as a weighted

GRAMMAR$(G', W, z, [X_1, \ldots, X_n])$ constraint. For Hamming distance, for each production $A \to a \in G$, we introduce additional unit weight productions to simulate substitution:

$$\{A \to b, W[A \to b] = 1 | A \to a \in G, A \to b \notin G, b \in \Sigma\}$$

Existing productions have zero weight. For edit distance, we introduce additional productions to simulate substitution, insertion and deletion:

$$\{A \to b, W[A \to b] = 1 | A \to a \in G, A \to b \notin G, b \in \Sigma\} \cup$$
$$\{A \to \varepsilon, W[A \to \varepsilon] = 1 | A \to a \in G, a \in \Sigma\} \cup$$
$$\{A \to Aa, W[A \to Aa] = 1 | a \in \Sigma\} \cup$$
$$\{A \to aA, W[A \to aA] = 1 | a \in \Sigma\}$$

To handle ε productions we modify Algorithm 4 so loops in lines 7, 29 run from 0 to j.

7 Counting for the GRAMMAR constraint

In has been shown that counting the strings of a specific length n that are accepted by a grammar G is $\#P$-complete. This follows from the fact that NFA is a subclass of context free grammars. However, it is known that there exist restrictions of context free grammars for which counting the number of strings of a given length is tractable. For instance, counting the solutions of a $Regular_{\text{DFA}}$ constraint can be done in time $O(ndQ)$ (Zanarini and Pesant 2007). Counting for unambiguous grammars is also polynomial. Moreover, there exist polynomial time approximation algorithms even for the general case.

The connection between parsing and s-WDNNFallows us to use algorithms developed in the knowledge compilation field to perform counting. We show first that if the grammar is unambiguous, then the corresponding s-WDNNFis also deterministic, i.e., the formulas represented by the children of each OR-NODESare mutually exclusive (see Jung 2008 for more). We can then use the algorithms developed in Darwiche and Marquis (2002) to compute the number of satisfying assignments of the GRAMMAR or WEIGHTEDGRAMMAR constraint as well as the number of satisfying assignments that contain each literal.

Theorem 4 *Let D be the* DNNF*obtained from running the* CYK *algorithm for a grammar G. Then if G is unambiguous then D is deterministic.*

Proof Assume G is unambiguous but D is not deterministic. This means that there exists a model m of D and an OR-NODE $n = (i, j, A)$ such that two of children are true at the same time. Let $n_1 = (i, j, k, A \to BC)$, $n_2 = (i, j, k', A \to DE)$ be the two children that are true. Then $A \to BC$ can generate the substring at positions $i \ldots i + j$ and so can $A \to DE$. So in any parse tree that uses the production $A \to BC$ to generate the substring at positions $i \ldots i + j$, we can substitute the production $A \to DE$. This means that the string s that corresponds to m can be generated in two different ways. This would imply that G is ambiguous, a contradiction. □

Note that this result does not hold in the other direction: if D is deterministic, it may be the case that G is unambiguous for strings of length exactly n but not for longer or shorter strings.

Finally, it has been shown in Bertoni et al. (2000) that for grammars with bounded ambiguity there exists a randomized approximation algorithm that runs in $O(n^2 \log n)$ time.

8 Applications of the WEIGHTEDGRAMMAR constraint

In this section we consider several applications of the WEIGHTEDGRAMMAR constraint in modeling and solving constraint satisfaction problems.

Shift scheduling problems. Constraint programming has been successfully applied to shift scheduling (Pesant 2004; Demassey et al. 2006; Quimper and Walsh 2007). In a shift scheduling problem we need to generate an optimum schedule of shifts for a company so that two types of constraints are satisfied: labor demand constraints that make sure that the number of available workers meet labor requirements at each time point; and scheduling constraints that ensure that standard regulation rules are fulfilled. The optimization function typically expresses some combination of the total cost of production, incentives provided by the business and employees' preferences. Labor demand constraints are straightforward to model with cardinality constraints. On the other hand, real world regulation rules can often be cumbersome and difficult to express. One way to encode these rules is to represent them with a finite automaton (Pesant 2004). In practice, the resulting automaton can be very large which leads to large memory consumption and slows down search (Quimper and Walsh 2007). Alternatively, regulation rules can be represented in a succinct way using the GRAMMAR constraint (Sellmann 2006; Quimper and Walsh 2006), an approach that was shown to be effective in Quimper and Walsh (2007), Kadioglu and Sellmann (2008). The drawback of the models described above for shift scheduling problems is that there is little interaction between the problem constraints and the optimization function. The expressiveness of the WEIGHTEDGRAMMAR constraint significantly improves this situation as the following examples show.

The WEIGHTEDGRAMMAR constraint allows us model employee's preferences. Suppose there is an employee that prefers day shifts to night shifts. To model these preferences we penalize with the unit weight night shift assignments to this employee. We also put a bound k on the total number of night shift assignments by fixing the upper bound of the cost variable z. In this way we make sure that the employee will not work more than k night shifts in a schedule. In a similar way we can model an incentive to an employee. Suppose the business has to pay extra salary s for a night shift on public holidays. To model this constraint we increase the weight of the corresponding leaves the AND/OR graph by s.

The WEIGHTEDGRAMMAR constraint can also be used to model bonus payments for specific activities. Suppose the employer wants to provide incentives for employees to work longer hours. Let the grammar that represents regulation rules include the following rule: $W \rightarrow WW|W$, $W \rightarrow a$. The non-terminal W(ork) generates sequence working activities for an employee. Hence, an occurrence of a non-terminal W at the ith level of the AND/OR graph indicates that an employee performed i consecutive activities. If, for example, the salary of an employee increases by s for each activity performed after the 5th consecutive activity, we can model this by increasing by s the weight of W at the 5th level or higher in the AND/OR graph as we describe in Sect. 4.

Finally, for some forms of objective functions, we can use the WEIGHTEDGRAMMAR constraint to construct the conjunction of the optimization function and the GRAMMAR constraint without requiring an additional modeling step. This was described more extensively in Sect. 4 and is also considered in our empirical evaluation.

Over-constrained problems. A lot of real world problems are over-constrained so it is impossible to satisfy all requirements. In this case some constraints have to be relaxed in order to construct a feasible schedule. Therefore, problem constraints are usually partitioned

into hard and soft constraints. Hard constraints have to be satisfied while soft constraints accept an assignment that is close to a valid assignment by some violation measures. Some commonly used violation measures include the Hamming and edit distances. Consider again shift scheduling problems. Let the problem description include a regulation rule that requires that the schedule of an employee covers at most s activities. An employer might prefer to relax this requirement at a cost rather than hire more employees. The soft GRAMMAR constraint with a Hamming distance violation measure can be used to model the relaxed regulation rule. The soft GRAMMAR constraint generates a schedule that ensures that each employee will have to work at most k hours of overtime.

Constraint based local search. Local search performs two main operations: finds possible candidates to move to at the next step (the neighborhood) and evaluates these candidates based on a cost function to select the best one. In constraint based local search the cost function depends on the cost of each individual problem constraint.

As was pointed out (van Hoeve 2009), cost functions are closely related to violation functions for soft global constraints. The violation functions for soft global constraints can be used to compute the gradient of a constraint cost function that is used choose a variable value pair that yields the greatest decrease of the constraint violation. Therefore, the soft GRAMMAR constraint can be used to compute the gradient in the GRAMMAR constraint in constraint based local search, similarly to other constraints, e.g. ALLDIFFERENT or SEQUENCE constraints (Van Hentenryck and Michel 2005).

Soft global constraints are also used to reduce the search space of possible candidates for the next move. It was shown in Métivier et al. (2009) that using soft global constraints, like the soft REGULAR or the soft GCC constraints, significantly speeds up solving nurse scheduling problems (NSP). In a similar way the soft GRAMMAR constraint can be used to express regulation rules in NSP.

Edit distance constraint. Another interesting application of the WEIGHTEDGRAMMAR constraint was proposed in Katsirelos et al. (2009). The weighted grammar constraint was used to represent the EDITDISTANCE constraint. The EDITDISTANCE($[X_1, \ldots, X_n, Y_1, \ldots, Y_m], N$) constraint holds iff the edit distance between assignments of two sequences of variables $\mathbf{X}$ and $\mathbf{Y}$ is less than or equal to N. This constraint can be encoded as a weighted linear context free grammar. This encoding allows us to obtain automatically a domain consistency propagator for the EDITDISTANCE constraint. Moreover, it allows us to build a conjunction of the EDITDISTANCE constraint with REGULAR constraint and obtain a efficient DC propagator for the conjunction.

9 Experimental results

In this section, we evaluate the impact of the WEIGHTEDGRAMMAR constraint in practice. For this evaluation, we chose a set of shift-scheduling benchmarks (Demassey et al. 2005; Cote et al. 2007) that have been commonly used before to show the utility of GRAMMAR constraints (Quimper and Walsh 2007; Kadioglu and Sellmann 2008). We show that the weighted GRAMMAR constraint is efficient for solving these problems, because it allows propagating the conjunction of a GRAMMAR and the objective function of a shift scheduling problem.

A personal schedule is subject to various regulation rules, e.g. a full-time employee has to have a one-hour lunch. These rules are encoded into a context-free grammar augmented

with conditional productions (Quimper and Walsh 2007; Quimper and Louis-Martin 2007). A schedule for an employee has $n = 96$ 15-minute slots represented by n variables. In each slot, an employee can work on an activity (a_i), take a break (b), lunch (l) or rest (r). These rules are represented by the following grammar:

$$S \to RPR,\ f_P(i,j) \equiv 13 \le j \le 24, \quad P \to WbW,\ L \to lL|l,\ f_L(i,j) \equiv j = 4$$
$$S \to RFR,\ f_F(i,j) \equiv 30 \le j \le 38, \quad R \to rR|r, \quad W \to A_i,\ f_W(i,j) \equiv j \ge 4$$
$$A_i \to a_i A_i | a_i,\ f_A(i,j) \equiv open(i), \quad F \to PLP$$

where functions $f(i,j)$ are restrictions on productions and $open(i)$ is a function that returns 1 if the business is opened at ith slot and 0 otherwise. To model labour demand for a slot we introduce Boolean variables $x(i,j,a_k)$, equal to 1 if jth employee performs activity a_k at ith time slot. For each time slot i and activity a_k we post a constraint $\sum_{j=1}^{m} x(i,j,a_k) > d(i,a_k)$, where m is the number of employees. We break row-symmetry with a LEX constraint.

Finally, we used a static variable and value ordering as described in Quimper and Walsh (2007). We assign variables top down and from left to right. The value ordering is $r, b, l, a1, a2$. The goal is to minimize the number of slots in which employees work.

We used Gecode 2.0.1 for our experiments and ran them on an Intel Xeon 2.0 GHz with 4 Gb of RAM. In the first set of experiments, we used the WEIGHTEDGRAMMAR(G, z_j, X), $j = 1, \ldots, m$ with zero weights.

First we investigate whether the filtering algorithm for the WEIGHTEDGRAMMAR constraint introduces an overhead compared to the unweighted GRAMMAR constraint. For this purpose, we define the weight function as follows

$$W(P,i,j) = \begin{cases} 0 & \text{iff } (i,j) = 1 \\ +\infty & \text{iff } (i,j) = 0 \end{cases}$$

Our monolithic propagator gave similar results to the unweighted GRAMMAR propagator from (Quimper and Walsh 2007). Decompositions of the WEIGHTEDGRAMMAR constraint were slower than decompositions of the unweighted GRAMMAR constraint as the former uses integers instead of Booleans. Therefore, using the WEIGHTEDGRAMMAR constraint does not lead to a significant slowdown even if it does not achieve any extra pruning compared to GRAMMAR.

In the second set of experiments, we investigate whether propagating the conjunction of the GRAMMAR constraint and the objective function can achieve additional pruning. In order to do this, we augment the first model with additional constraints. We assigned weights to the productions of the grammar as described in Sect. 4. Specifically, we assigned weight 1 to activity productions, like $A_i \to a_i$. The objective function is thus reduced to $\sum_{j=1}^{m} z_j$. We recall that $\sum_{j=1}^{m} z_j$ is the number of slots in which employees worked. Results are presented in Table 1.

As can be seen from Table 1, we improved on the best solution found in the first model in 4 benchmarks and proved optimality in one. This shows that propagating the conjunction of the GRAMMAR constraints and the objective function pays off in terms of the runtime improvements. The decomposition of the WEIGHTEDGRAMMAR constraint was slightly slower than the monolithic propagator. As was shown in Sect. 3.2, the time complexity of the decomposition is the same as the complexity of the monolithic propagator if the decomposition constraints are invoked in a particular order. However, we cannot enforce this ordering in the solver. It was also pointed out that some constraints become irrelevant during the search, but they are not entailed. Therefore, they present an overhead. We eliminate this

Table 1 All benchmarks have a one-hour time limit. $|A|$ is the number of activities, m is the number of employees, *cost* shows the total number of slots in which employees worked in the best solution, *time* is the time to find the best solution, *bt* is the number of backtracks to find the best solution, *BT* is the number of backtracks in one hour, *Opt* shows if optimality is proved, *Imp* shows if a lower cost solution is found by the second model

$\|A\|$	#	m	Monolithic				Decomposition				Decomposition + entailment				Opt	Imp
			cost	time	bt	BT	cost	time	bt	BT	cost	time	bt	BT		
1	2	4	**107**	**5**	**0**	8652	**107**	7	**0**	5926	**107**	7	**0**	**11521**		
1	3	6	**148**	**7**	**1**	5917	**148**	34	**1**	1311	**148**	9	**1**	**8075**		
1	4	6	**152**	1836	**5831**	11345	**152**	**1379**	**5831**	**14815**	**152**	1590	**5831**	13287		
1	5	5	**96**	6	**0**	8753	**96**	6	**0**	2660	**96**	**3**	**0**	**45097**		
1	6	6	–	–	–	10868	**132**	3029	**11181**	13085	**132**	**2367**	**11181**	**16972**		
1	7	8	**196**	16	**16**	10811	**196**	18	**16**	6270	**196**	**15**	**16**	**10909**		
1	8	3	**82**	11	**9**	**66**	**82**	13	**9**	**66**	**82**	**5**	**9**	**66**	√	√
1	10	9	–	–	–	10871	–	–	–	9627	–	–	–	**18326**		
2	1	5	**100**	523	**1109**	7678	**100**	634	**1109**	6646	**100**	**90**	**1109**	**46137**		
2	2	10	–	–	–	**11768**	–	–	–	10725	–	–	–	6885		
2	3	6	**165**	3517	**9042**	9254	168	2702	4521	6124	**165**	**2856**	**9042**	**11450**		√
2	4	11	–	–	–	**8027**	–	–	–	6201	–	–	–	5579		
2	5	4	**92**	**37**	**118**	**12499**	**92**	59	**118**	6332	**92**	49	**118**	10329		
2	6	5	**107**	**9**	**2**	6288	**107**	22	**2**	1377	**107**	14	**2**	**7434**		
2	8	5	**126**	422	**1282**	12669	**126**	1183	**1282**	3916	**126**	**314**	**1282**	**16556**		√
2	9	3	**76**	1458	**3588**	8885	**76**	2455	**3588**	5313	**76**	**263**	**3588**	**53345**		√
2	10	8	–	–	–	3223	–	–	–	3760	–	–	–	**8827**		

redundant computation by annotating these constraints with an additional entailment test that is specific to s-WDNNF (see Sect. 3.2). If this test succeeds the solver ignores them in the remainder of the current branch. Table 1 shows that redundant computation contributes a significant overhead and the additional entailment test improves performance in most cases, making the decomposition slightly faster than the monolithic propagator.

10 Related work

Knowledge compilation languages like decomposable NNF have been extensively studied in Darwiche and Marquis (2002). Fargier and Marquis have generalized negation normal form to valued negation normal form (Fargier and Marquis 2007). This replaces the AND and OR with an arbitrary valuation structure. It can thereby represent a wider range of functions. One instance of valued negation normal form is the weighted negation normal form studied here. Binary decision diagrams (BDDs) are another special case of decomposable negation normal form. A number of generalizations of BDDs have been proposed to include weights and probabilities. For instance, weighted decision diagrams (WDDs) augment each edge with a weight function (Ossowski and Baier 2006). WDDs subsume edge-valued decision diagrams and normalized algebraic decision diagrams. They have been used for problems like probabilistic model checking.

Quimper and Walsh (2006) and Sellmann (2006) independently proposed the global Grammar constraint. Both gave a monolithic propagator based on the CYK parser. Quimper

and Walsh also gave a monolithic propagator based on the Earley chart parser. Whilst the CYK propagator takes $\Theta(n^3)$ time, the propagator based on the Earley chart parser takes just $O(n^3)$ time to run and is not restricted to grammars in Chomsky normal form.

Quimper and Walsh gave a simple AND/OR decomposition of the Grammar constraint based on the CYK parser which can be encoded into SAT (Quimper and Walsh 2007). This decomposition can easily be mapped into decomposable NNF. Unit propagation on this decomposition will prune all possible values in the same asymptotic time as the monolithic propagator. Our decomposition of the s-WDNNFconstraint (and, by extension, of the WEIGHTEDGRAMMAR constraint) can be seen as a generalization of this Boolean decomposition.

Cost based GRAMMAR constraints (CFGC) were independently proposed by Kadioglu and Sellmann (2008). The difference between this work and ours is that in Kadioglu and Sellmann (2008) weights can only be placed on individual terminals. Here, we allow weights to be placed on any production and therefore WEIGHTEDGRAMMAR is more expressive than CFGC. For instance, we can express the Soft GRAMMAR constraint with the edit distance violation measure using the weighted GRAMMAR constraint, but we can not encode this constraint with the CFGC constraint.

Demassey, Pesant and Rousseau proposed the COSTREGULAR constraint (Demassey et al. 2006) that shares a lot of advantages of the weighted GRAMMAR constraint. However, it is a well known result from formal language theory that not all context free languages can be expressed as regular languages; moreover, when restricted to fixed length strings, there exist languages that can be expressed by polynomial sized context free grammars but the smallest regular language that accepts these languages is exponential in size. One such example is the palindrome language (Sellmann 2006).

Côté, Gendron, Quimper and Rousseau have proposed a mixed-integer programming (MIP) encoding of the Grammar constraint (Cote et al. 2007). This MIP encoding has one significant difference with the previous approaches. If there is more than one parsing for a sequence, it picks one arbitrarily whilst the previous propagator keeps all. This simplifies the MIP encoding without changing the set of solutions since only one parsing is needed to show membership in a context-free grammar. Experiments on a shift scheduling problem show that such MIP encodings are highly competitive.

11 Conclusions

We have introduced a weighted form of the GRAMMAR constraint. The GRAMMAR constraint is useful to specify a wide range of scheduling, rostering and sequencing problems. It restricts the values taken by a sequence of variables to be a string from a language defined by a given grammar. The addition of weights permits us to model over-constrained problems and problems with preferences. We have proposed two propagators for the WEIGHTEDGRAMMAR constraint based on the CYK and Earley parsers. We have also proposed a decomposition of the WEIGHTEDGRAMMAR constraint into some simple arithmetic constraints. Experiments on a shift-scheduling benchmark suggest that the WEIGHTEDGRAMMAR constraint has promise for solving real-world problems. Finally, we have studied the problem of counting solutions to the WEIGHTEDGRAMMAR constraint, including some special cases like unambiguous grammars. Our analysis of these propagators and of these counting problems was greatly simplified by constructing an equivalent weighted negation normal form formula. Weighted negation normal form is a special case of valued negation normal form (Fargier and Marquis 2007). It appears to be a useful

knowledge compilation language, being both very expressive yet having several tractable fragments, especially decomposable weighted NNF. We intend to study in greater detail properties of this language, and to use it in the analysis and construction of propagators and decompositions for other global constraints.

Acknowledgements We would like to thank Louis-Martin Rousseau and Claude-Guy Quimper for providing us with the benchmark data, and Claude-Guy Quimper for his help with the experiments.

We are grateful to the anonymous referees for their comments that helped to improve the paper.

References

Bertoni, A., Goldwurm, M., & Santini, M. (2000). Random generation and approximate counting of ambiguously described combinatorial structures. In *STACS 2000, 17th annual symposium on theoretical aspects of computer science: Vol. 1770* (pp. 567–580). Berlin: Springer.

Cote, M. C., Bernard, G., Claude-Guy, Q., & Louis-Martin, R. (2007). Formal languages for integer programming modeling of shift scheduling problems. In *TR*.

Darwiche, A., & Marquis, P. (2002). A knowledge compilation map. *Journal of Artificial Intelligence Research*, *17*, 229–264.

Demassey, S., Pesant, G., & Rousseau, L. M. (2005). Constraint programming based column generation for employee timetabling. In *Second international conference, CPAIOR 2005: Vol. 3524/2005*, Prague, Czech Republic (pp. 140–154).

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

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the ACM*, *13*(2), 94–102.

Eén, N., & Sörensson, N. (2006). Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modeling and Computation* 2, 1–26.

Fargier, H., & Marquis, P. (2007). On valued negation normal form formulas. In M.M. Veloso (Ed.), *Proceedings of the 20th international joint conference on artificial intelligence (IJCAI07)*, Hyderabad, India (pp. 360–365).

Gore, V., Jerrum, M., Kannan, S., Sweedyk, Z., & Mahaney, S. (1997). A quasi-polynomial-time algorithm for sampling words from a context-free language. *Information and Computation*, *134*(1), 59–74.

Hopcroft, J. E., & Ullman, J. D. (1990). *Introduction to automata theory, languages, and computation*. Boston: Addison-Wesley, Longman.

Jung, J. C. (2008). *Value ordering based on solution counting*. PhD thesis, University Nova de Lisboa.

Kadioglu, S., & Sellmann, M. (2008). Efficient context-free grammar constraints. In: *Proceedings of the 23rd national conference on artificial intelligence* (pp. 310–316).

Katsirelos, G., Narodytska, N., & Walsh, T. (2008). The weighted cfg constraint. In *Integration of AI and OR techniques in constraint programming for combinatorial optimization problems, 5th international conference, CPAIOR 2008: Vol. 5015*, Paris, France (pp. 323–327). Berlin: Springer.

Katsirelos, G., Maneth, S., Narodytska, N., & Walsh, T. (2009). Restricted global grammar constraints. In *Proceedings of the 15th international conference on principles and practice of constraint programming, CP 2009: Vol. 5732*, Lisbon, Portugal (pp. 501–508). Berlin: Springer.

Métivier, J. P., Boizumault, P., & Loudni, S. (2009). Solving nurse rostering problems using soft global constraints. In *Proceedings of the 15th international conference on principles and practice of constraint programming, CP 2009: Vol. 5732*, Lisbon, Portugal (pp. 73–87). Berlin: Springer.

Ney, H. (1991). Dynamic programming parsing for context-free grammars in continuous speech recognition. *IEEE Transactions on Signal Processing*, *39*(2), 336–340.

Ossowski, J., & Baier, C. (2006). Symbolic reasoning with weighted and normalized decision diagrams. In: *Proceedings of the 12th symposium on the integration of symbolic computation and mechanized reasoning: Vol. 151* (pp. 39–56).

Pesant, G. (2004). A regular language membership constraint for finite sequences of variables. In M. Wallace (Ed.), *Proceedings of 10th international conference on principles and practice of constraint programming (CP'04): Vol. 3258* (pp. 482–495). Berlin: Springer.

Quimper, C. G., & Louis-Martin, R. (2007). A large neighbourhood search approach to the multi-activity shift scheduling problem. In *TR*.

Quimper, C. G., & Walsh, T. (2006). Global Grammar constraints. In F. Benhamou (Ed.), *Proceedings of the 12th international conference on principles and practice of constraint programming: Vol. 4204* (pp. 751–755). Berlin: Springer.

Quimper, C. G., & Walsh, T. (2007). Decomposing global grammar constraints. In *Proceedings of the 13th international conference on principles and practice of constraint programming, CP 2007*.

Sellmann, M. (2006). The theory of Grammar constraints. In *Proceedings of the 12th international conference on the principles and practice of constraint programming (CP06): Vol. 4204* (pp. 530–544). Berlin: Springer.

Van Hentenryck, P., & Michel, L. (2005). *Constraint-based local search*. Cambridge: The MIT Press.

van Hoeve, W. J. (2009). Soft global constraints, tutorial at principles and practice of constraint programming. In *CP 2007, 15th international conference*.

Walsh, T. (2000). Sat v csp. In R. Dechter (Ed.), *Principles and practice of constraint programming—CP 2000, 6th international conference, Singapore: Vol. 1894* (pp. 441–456). Berlin: Springer.

Zanarini, A., & Pesant, G. (2007). Solution counting algorithms for constraint-centered search heuristics. In C. Bessiere (Ed.), *Principles and practice of constraint programming—CP 2007, 13th international conference: Vol. 4741*, Providence, RI, USA. Berlin: Springer.