The Weighted CFG Constraint

George Katsirelos, Nina Narodytska, and Toby Walsh

University of New South Wales and NICTA, Sydney, Australia

Abstract. We introduce the weighted CFG constraint and propose a propagation algorithm that enforces domain consistency in $O(n^3|G|)$ time. We show that this algorithm can be decomposed into a set of primitive arithmetic constraints without hindering propagation.

1 Introduction

One very promising method for rostering and other domains is to specify constraints via grammars or automata that accept some language. We can specify constraints in this way on, for instance, the number of consecutive night shifts or the number of days off in each 7 day period. With the REGULAR constraint [4], we specify the acceptable assignments to a sequence of variables by a deterministic finite automaton. One limitation of this approach is that the automaton may need to be large. For example, there are regular languages which can only be defined by an automaton with an exponential number of states. Researchers have therefore looked higher up the Chomsky hierarchy. In particular, the CFG constraint [8,6] permits us to specify constraints using any context-free grammar. In this paper, we consider a further generalization to the weighted CFG constraint. This can model over-constrained problems and problems with preferences.

2 The Weighted CFG Constraint

In a context-free grammar, rules have a left-hand side with just one non-terminal, and a right-hand side consisting of terminals and non-terminals. Any context-free grammar can be written in Chomsky form in which the right-hand size of a rule is just one terminal or two non-terminals. The weighted WCFG$(G, W, z, [X_1, \ldots, X_n])$ constraint holds iff 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 matrix W defines weights of productions in the grammar G. The weight of a derivation is the sum of production weights used in the derivation. The WCFG constraint is domain consistent iff for each variable, every value in its domain can be extended to an assignment satisfying the constraint.

We give a propagator for the WCFG constraint based on an extension of the CYK parser to probabilistic grammars [3]. We assume that G is in Chomsky normal form and with a single start non-terminal S. The algorithm has two stages. In the first, we construct a dynamic programing table $V[i, j]$ where an element A of $V[i, j]$ is a potential non-terminal that generates a substring $[X_i, \ldots, X_{i+j}]$. We compute a lower bound $l[i, j, A]$ on the minimal weight of a derivation from A. In the second stage, we move from $V[1, n]$ to the bottom of table V. For an element A of $V[i, j]$, we compute an

L. Perron and M. Trick (Eds.): CPAIOR 2008, LNCS 5015, pp. 323–327, 2008.

upper bound $u[i, j, A]$ on the maximal weight of a derivation from A of a substring $[X_i, \ldots, X_{i+j}]$. We mark the element A iff $l[i, j, A] \leq u[i, j, A]$. The pseudo-code is presented in Algorithm 1. Lines 2–5 initialize l and u. Lines 6–16 compute the first stage, whilst lines 20–29 compute the second stage. Finally, we prune inconsistent values in lines 30–31. Algorithm 1 enforces domain consistency in $O(|G|n^3)$ time.

Algorithm 1. The weighted CYK propagator

1: **procedure** WCYK-ALG($G, W, z, [X_1, \ldots, X_n]$)
2:   **for** $j = 1$ **to** n **do**
3:     **for** $i = 1$ **to** $n - j + 1$ **do**
4:       **for** each $A \in G$ **do**
5:         $l[i, j, A] = z + 1; u[i, j, A] = -1;$
6:   **for** $i = 1$ **to** n **do**
7:     $V[i, 1] = \{A | A \rightarrow a \in G, a \in D(X_i)\}$
8:     **for** $A \in V[i, 1]$ s.t $A \rightarrow a \in G, a \in D(X_i)$ **do**
9:       $l[i, 1, A] = \min\{l[i, 1, A], W[A \rightarrow a]\};$
10:   **for** $j = 2$ **to** n **do**
11:     **for** $i = 1$ **to** $n - j + 1$ **do**
12:       $V[i, j] = \emptyset;$
13:       **for** $k = 1$ **to** $j - 1$ **do**
14:         $V[i, j] = V[i, j] \cup \{A | A \rightarrow BC \in G, B \in V[i, k], C \in V[i + k, j - k]\}$
15:         **for** each $A \rightarrow BC \in G$ s.t. $B \in V[i, k], C \in V[i + k, j - k]$ **do**
16:           $l[i, j, A] = \min\{l[i, j, A], W[A \rightarrow BC] + l[i, k, B] + l[i + k, j - k, C]\};$
17:   **if** $S \notin V[1, n]$ **then**
18:     return 0;
19:   mark $(1, n, S)$; $u[1, n, S] = z;$
20:   **for** $j = n$ **downto** 2 **do**
21:     **for** $i = 1$ **to** $n - j + 1$ **do**
22:       **for** A such that (i, j, A) is marked **do**
23:         **for** $k = 1$ **to** $j - 1$ **do**
24:           **for** each $A \rightarrow BC \in G$ s.t. $B \in V[i, k], C \in V[i + k, j - k]$ **do**
25:             **if** $W[A \rightarrow BC] + l[i, k, B] + l[i + k, j - k, C] > u[i, j, A]$ **then**
26:               continue;
27:             mark (i, k, B); mark $(i + k, j - k, C)$;
28:             $u[i, k, B] = \max\{u[i, k, B], u[i, j, A] - l[i + k, j - k, C] - W[A \rightarrow BC]\};$
29:             $u[i + k, j - k, C] = \max\{u[i + k, j - k, C], u[i, j, A] - l[i, k, B] - W[A \rightarrow BC]\};$
30:   **for** $i = 1$ **to** n **do**
31:     $D(X_i) = \{a \in D(X_i) | A \rightarrow a \in G, (i, 1, A) \; is \; marked \; and \; W[A \rightarrow a] \leq u[i, 1, A]\};$
32:   return 1;

3 Decomposition of the Weighted CFG Constraint

As an alternative to this monolithic propagator, we propose a simple decomposition with which we can also enforce domain consistency. A decomposition has several advantages. For example, it is easy to add to any constraint solver. As a second example, decomposition gives an efficient incremental propagator, and 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. Given the table V obtained by Algorithm 1, we construct the corresponding AND/OR directed acyclic graph (DAG) as in [7]. We label an OR node by $n(i, j, A)$, and an AND node by $n(i, j, k, A \rightarrow BC)$. We denote the parents of a node nd as $PRT(nd)$ and the children as $CHD(nd)$. For each node two integer variables are introduced to compute l and u. For an OR-node nd, these are $l_O(nd)$ and $u_O(nd)$, whilst for an AND-node nd, these are $l_A(nd)$, $u_A(nd)$.

For each AND node $nd = n(i, j, k, A \rightarrow BC)$ we post a constraint to connect nd to its children $CHD(nd)$:

$$l_A(nd) = \sum_{n_c \in CHD(nd)} l_O(n_c) + W[A \rightarrow BC] \tag{1}$$

For each OR node $nd = n(i, j, A)$ we post constraints to connect nd to its children $CHD(nd)$:

$$l_O(nd) = \min_{n_c \in CHD(nd)} \{l_A(n_c)\} \tag{2}$$

$$u_O(nd) = u_A(n_c),\ n_c \in CHD(nd) \tag{3}$$

For each OR node $nd = n(i, j, A)$ we post a set of constraints to connect nd to its parents $PRT(nd)$ and siblings:

$$u_O(nd) = max_{n_p \in PRT(nd)}\{u_A(n_p) - l_O(n_{sb}) - W[P]\}, \tag{4}$$

where $P = B \rightarrow AC$ or $B \rightarrow CA$, $n_p = n(r, q, t, P)$ is the parent of $nd = n(i, j, A)$ and $n_{sb} = n(i_1, j_1, C)$.

Finally, we introduce constraints to prune X_i. For each leaf of the DAG that is an OR node $nd = n(i, 1, a)$, we introduce:

$$a \in D(X_i) \Rightarrow 0 \leq l_O(nd) \leq z \tag{5}$$

$$a \notin D(X_i) \Leftrightarrow l_O(nd) > z \tag{6}$$

$$l_O(nd) > u_O(nd) \Rightarrow a \notin D(X_i) \tag{7}$$

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

$$u_O(n(1, n, S)) \leq z \tag{8}$$

Bounds propagation will set the lower bound of $l_O(n(i, j, A))$ to the minimal weight of a derivation from A, and the upper bound on $u_O(n(i, j, A))$ to the maximum weight of a derivation from A. We forbid branching on variables $l_{A|O}$ and $u_{A|O}$ as branching on $l_{A|O}$ would change the weights matrix W and branching on $u_{A|O}$ would add additional restrictions to the weight of a derivation. Bounds propagation on this decomposition enforces domain consistency on the WCFG constraint. If we invoke 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.

We can speed up propagation by recognizing when constraints are entailed. If $l_O(nd) > u_O(nd)$ holds for an OR node nd then constraints (4) and (2) are entailed. If $l_A(nd) > u_A(nd)$ holds for an AND node nd then constraints (1) and (3) are entailed. To model entailment we augmented each of these constraints in such a way that if $l_O(nd) > u_O(nd)$ or $l_A(nd) > u_A(nd)$ hold then corresponding constraints are not invoked by the solver.

4 The Soft CFG Constraint

We can use the WCFG constraint to encode a soft version of CFG constraint which is useful for modelling over-constrained problems. The soft CFG$(G, z, [X_1, \dots, X_n])$ constraint holds iff the string $[X_1, \dots, X_n]$ is at most distance z from a string in G. We consider both Hamming and edit distances. We encode the soft CFG$(G, z, [X_1, \dots, X_n])$ constraint as a weighted CFG$(G', W, z, [X_1, \dots, X_n])$ constraint. For Hamming distance, for each production $A \rightarrow a \in G$, we introduce additional unit weight productions to simulate substitution:

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

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

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

To handle ε productions we modify Alg. 1 so loops in lines (13),(23) run from 0 to j.

5 Experimental Results

We evaluated these propagation methods on shift-scheduling benchmarks [2,1]. A personal schedule is subject to various regulation rules, e.g. a full-time employee has to have a one-hour lunch. This rules are encoded into a context-free grammar augmented with restrictions on productions [7,5]. A schedule for an employee has $n = 96$ 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:

$$\begin{array}{l} S \rightarrow RPR, f_P(i,j) \equiv 13 \leq j \leq 24,\; P \rightarrow WbW,\; L \rightarrow lL|l, f_L(i,j) \equiv j = 4 \\ S \rightarrow RFR, f_F(i,j) \equiv 30 \leq j \leq 38,\;\; R \rightarrow rR|r,\; W \rightarrow A_i, f_W(i,j) \equiv j \geq 4 \\ \;\; A_i \rightarrow a_i A_i | a_i, f_A(i,j) \equiv open(i), \quad F \rightarrow PLP \end{array}$$

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 $b(i,j,a_k)$, 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. The goal is to minimize the number of slots in which employees worked.

We used Gecode 2.0.1 for our experiments and ran them on an Intel Xeon 2.0Ghz with 4Gb of RAM[1]. In the first set of experiments, we used the weighted CFG(G, z_j, X), $j = 1, \dots, m$ with zero weights. Our monolithic propagator gave similar results to the unweighted CFG propagator from [7]. Decompositions were slower than decompositions of the unweighted CFG constraint as the former uses integers instead of Booleans.

[1] We would like to thank Claude-Guy Quimper for his help with the experiments.

Table 1. All benchmarks have 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.

			Monolithic				Decomposition				Decomption+entailment					
$\|A\|$	#	m	cost	time	bt	BT	cost	time	bt	BT	cost	time	bt	BT	Opt	Imp
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**		

In the second set of experiments, we assigned weight 1 to activity productions, like $A_i \rightarrow a_i$, and post an additional cost function $\sum_{j=1}^{m} z_j$ that is minimized. $\sum_{j=1}^{m} z_j$ is the number of slots in which employees worked. Results are presented in Table1. We improved on the best solution found in the first model in 4 benchmarks and proved optimality in one. The decomposition of the weighted CFG constraint was slightly slower than the monolithic propagator, while entailment improved performance in most cases.

References

1. Cote, M.-C., Bernard, G., Claude-Guy, Q., Louis-Martin, R.: Formal languages for integer programming modeling of shift scheduling problems. Technical Report, Center for Research on Transportation, Montreal (2007)
2. Demassey, S., Pesant, G., Rousseau, L.-M.: Constraint programming based column generation for employee timetabling. In: Barták, R., Milano, M. (eds.) CPAIOR 2005. LNCS, vol. 3524, Springer, Heidelberg (2005)
3. Ney, H.: Dynamic programming parsing for context-free grammars in continuous speech recognition. IEEE Trans. on Signal Processing 39(2), 336–340 (1991)
4. Pesant, G.: A regular language membership constraint for finite sequences of variables. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, Springer, Heidelberg (2004)
5. Quimper, C.-G., Louis-Martin, R.: A large neighbourhood search approach to the multi-activity shift scheduling problem. Technical Report, Center for Research on Transportation, Montreal (2007)
6. Quimper, C.-G., Walsh, T.: Global Grammar constraints. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, Springer, Heidelberg (2006)
7. Quimper, C.-G., Walsh, T.: Decomposing Global Grammar constraints. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, Springer, Heidelberg (2007)
8. Sellmann, M.: The theory of Grammar constraints. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, Springer, Heidelberg (2006)