Global Grammar Constraints

Claude-Guy Quimper[1] and Toby Walsh[2]

[1] School of Computer Science, University of Waterloo, Canada,
cquimper@math.uwaterloo.ca
[2] NICTA and UNSW, Sydney, Australia, tw@cse.unsw.edu.au

1 Introduction

Global constraints are an important tool in the constraint toolkit. Unfortunately, whilst it is usually easy to specify when a global constraint holds, it is often difficult to build a good propagator. One promising direction is to specify global constraints via grammars or automata. For example, the Regular constraint [1] permits us to specify a wide range of global constraints by means of a DFA accepting a regular language, and to propagate this constraint specification efficiently and effectively. More precisely, the Regular constraint ensures that the values taken by a sequence of variables form a string accepted by the DFA. In this paper, we consider how to propagate such grammar constraints and a number of extensions.

2 Regular constraint

Pesant has given a domain consistency algorithm for the Regular constraint based on dynamic programming that runs in $O(ndQ)$ time and space where d is the domain size, Q is the number of states in the DFA, and n is the number of variables. The Regular constraint can be encoded using a simple sequence of ternary constraints. Enforcing GAC on this decomposition achieves GAC on the original Regular constraint. and takes just $O(ndQ)$ time and space. We introduce a second sequence of variables, Q_0 to Q_n to represent the state of the automaton. We then post the sequence of transition constraints $C(X_{i+1}, Q_i, Q_{i+1})$ for $0 \leq i < n$ which hold iff $Q_{i+1} = T(X_{i+1}, Q_i)$ where T is the transition function of the DFA. In addition, we post the unary constraints $Q_0 = q_0$ and $Q_n \in F$. To enforce GAC on such ternary constraints, we can use the table constraint available in many solvers, or primitives like the implication.

One advantage of this encoding is that we have explicit access to the states of the automaton. These can be used, for example, to model the objective function. The states of the automaton also need not be finite integer domain variables but can, for example, be set variables. We can model the open stacks problem in this way. This encoding also works with a non-deterministic finite automaton. Whist NFA only recognize regular languages, they can do so with exponentially fewer states than the smallest DFA. Enforcing GAC on the encoding takes $O(nT)$ time, where T is the number of transitions in the automaton. This number can be exponentially smaller for a NFA compared to a DFA. We can also encode the soft form of the REGULAR constraint [3] in a similar way by introducing a variable whose value is the state of the automaton and the distance from it.

For repeating sequences, we introduce two cyclic forms of the REGULAR constraint. $\text{REGULAR}_+(\mathcal{A}, [X_1, \ldots, X_n])$ holds iff the string defined by $X_1 \ldots X_n X_1$ is part of the regular language recognized by $\mathcal{A}$. To enforce GAC on REGULAR_+, we can create a new automaton in which states also contain the first value used. We can therefore use this new automaton and a normal REGULAR constraint to enforce GAC on REGULAR_+ in $O(nd^2Q)$ time. The $\text{REGULAR}_\circ$ constraint ensures that any rotation of the sequence gives a string in the regular language. More precisely, $\text{REGULAR}_\circ(\mathcal{A}, [X_1, \ldots, X_n])$ holds iff the strings defined by $X_i \ldots X_{1+(i+n-1 \bmod n)}$ for $1 \leq i \leq n$ are part of the regular language recognized by $\mathcal{A}$. Unfortunately, enforcing GAC on a $\text{REGULAR}_\circ$ constraint is NP-hard (reduction from Hamiltonian cycle).

3 CFG constraint

Another generalization is to context-free languages. We introduce the global grammar constraint $\text{CFG}(\text{G}, [X_1, \ldots, X_n])$ which ensures that X_1 to X_n form a string accepted by the context-free grammar $\mathcal{G}$. Such a constraint might be useful in a number of applications like bioinformatics or natural language processing.

To achieve GAC on a CFG constraint, we give a propagator based on the CYK parser which requires the context-free grammar to be in Chomsky normal form. The propagator given in Algorithm 1 proceeds in two phases. In the first phase (lines 1 to 7), we use dynamic programming to construct a table $V[i, j]$ with the potential non-terminal symbols that can be parsed using values in the domains of X_i to X_{i+j-1}. $V[1, n]$ thus contains all the possible parsings of the sequence of n variables. In the second phase of the algorithm (lines 9 to 18), we backtrack in the table V and mark each triplet (i, j, A) such that there exists a valid sequence of size n in which A generates the substring of size j starting at i. When the triplet $(i, 1, A)$ is marked, we conclude there is a support for every value $a \in \text{dom}(X_i)$ such that $A \to a \in G$.

Theorem 1 `CYK-prop` *enforces GAC on* $\text{CFG}(\mathcal{G}, [X_1, \ldots, X_n])$ *in* $\Theta(|G|n^3)$ *time and* $\Theta(|G|n^2)$ *space.*

Our second propagator is based on the popular Earley chart parser which also uses dynamic programming to parse a context-free language. Whilst this propagator is more complex, it is not restricted to Chomsky normal form, and is often much more efficient than CYK as it parses strings top-down, particularly when the productions are left-recursive. The propagator again uses dynamic programming to build up possible support. Productions are annotated with a "dot" indicating position of the parser. WLOG, we assume a unique starting production $S \to U$. A successful parsing is thus $S \to U\bullet$.

Algorithm 2 is the Earley chart parser augmented with the sets S that keep track of the supports for each value in the domains. We use a special data structure to implement these sets S. We first build the basic sets $\{X_i = v\}$ for

Algorithm 1: CYK-prop(G, $[X_1, \dots, X_n]$)

```
for i = 1 to n do
    V[i, 1] ← {A | A → a ∈ G, a ∈ dom(X_i)}
for j = 2 to n do
    for i = 1 to n − j + 1 do
        V[i, j] ← ∅
        for k = 1 to j − 1 do
            V[i, j] ← V[i, j] ∪ {A | A → BC ∈ G, B ∈ V[i, k], C ∈ V[i + k, j − k]}
if S ∉ V[1, n] then return "Unsatisfiable"
mark (1, n, S)
for j = n downto 2 do
    for i = 1 to n − j + 1 do
        for A → BC ∈ G such that (i, j, A) is marked do
            for k = 1 to j − 1 do
                if B ∈ V[i, k], C ∈ V[k + k, j − k] then
                    mark (i, k, B)
                    mark (i + k, j − k, C)
for i = 1 to n do
    dom(X_i) ← {a ∈ dom(X_i) | A → a ∈ G, (i, 1, A) is marked}
return "Satisfiable"
```

every potential support $v \in \text{dom}(X_i)$. Once a set is computed, its value is never changed. To compute the union of two sets $A \cup B$, we create a set C with a pointer on A and a pointer on B. This allows to represent the union of two sets in constant time. The data structure forms a directed graph where the sets are the nodes and the pointers are the edges. To enumerate the content of a set S, one can perform a depth-first search. The basic sets $\{X_i = v\}$ that are visited in the search are the elements of S.

Theorem 2 `Earley-prop` *enforces GAC on* CFG$(\mathcal{G}, [X_1, \dots, X_n])$ *in* $O(|G|n^3)$ *time for an arbitrary context-free grammar, and in* $O(|G|n^3)$ *space.*

4 Related work

For the REGULAR constraint, a propagation algorithm based on dynamic programming that enforces GAC was given in [1]. Coincidently Beldiceanu, Carlsson and Petit proposed specifying global constraints by means of deterministic finite automaton augmented with counters [5]. Propagators for such automaton are constructed automatically by means of a conjunction of signature and transition constraints. The ternary encodings used here are similar to those proposed in [5]. However, there are a number of differences. One is that we permit non-deterministic transitions. As argued before, non-determinism can reduce the size

Algorithm 2: Earley-Prop(G, $[X_1, \ldots, X_n]$)

```
for i = 0 to n do C[i] ← ∅
queue ← {(s → •u, 0, ∅)}
for i = 0 to n + 1 do
    for state ∈ C[i] do push(state,queue)
    while queue is not empty do
        (r, j, S) ← pop(queue)
        add((r, j, S), C[i])
        if r = (u → v•) then
            foreach (w → ... •u ..., k, T) ∈ C[j] do
                add((w → ... u• ..., k, S ∪ T), queue)
        else if i ≤ n and r = (u → ... •v ...) and v ∈ dom(X_i) then
            add((u → ... v• ..., j, S ∪ {X_i = v}), C[i + 1])
        else if r = (u → ... •v ...) and non_terminal(v, G) then
            foreach v → w ∈ G such that (v → •w, i, ∅) ∉ C[i] ∪ queue do
                push((v → •w, i, ∅), queue)
    if C[i] = ∅ then
        return "Unsatisfiable"
if (s → u•, 0, S) ∈ C[n] then
    for i = 1 to n do
        dom(X_i) = {a | X_i = a ∈ S}
else
    return "Unsatisfiable"
```

of the automaton significantly. In addition, the counters used by Beldiceanu, Carlsson and Petit introduce complexity. For example, they need to achieve pairwise consistency to guarantee global consistency. Pesant encodes a cyclic STRETCH constraint into a REGULAR constraint in which the initial variables of the sequence are repeated at the end, and then dummy unconstrained variables are placed at the start and end [1]. Hellsten, Pesant and van Beek propose a domain consistency algorithm similar to that for the REGULAR constraint [6]. They also showed how to extend it to deal with cyclic STRETCH constraints.

Algorithm 3: add($(a, b, c), q$)

```
if ∃(a, b, d) ∈ q then
    q ← replace((a, b, d), (a, b, c ∪ d), q)
else
    push((a, b, c), q)
```

5 Conclusions

We have studied a range of grammar constraints. These are global constraints over a sequence of variables which restrict the values assigned to be a string within a given language. Such constraints are useful in a wide range of scheduling, rostering and sequencing problems. For regular languages, we gave a simple encoding into ternary constraints that can be used to enforce GAC in linear time. Experiments demonstrate that such encodings are efficient and effective in practice. This ternary encoding is therefore an easy means to incorporate this global constraint into constraint toolkits. We also considered a number of extensions including regular languages specified by non-deterministic finite automata, and soft and cyclic versions of the global constraint. For context-free languages, we gave two propagators which enforce GAC based on the CYK and Earley parsers. Experiments show that the propagator based on the CYK parser is faster at the top of the search tree while the propagator based on the Earley parser is faster at the end of the search. This shows some potential for a hybrid propagator. There are many directions for future work. One promising direction is to learn grammar constraints from examples. We can leverage on results and algorithms from grammar induction. For example, it is not possible to learn a REGULAR constraint from just positive examples.

References

1. Pesant, G.: A regular language membership constraint for finite sequences of variables. In Wallace, M., ed.: Proceedings of 10th International Conference on Principles and Practice of Constraint Programming (CP2004), Springer (2004) 482–295
2. Janssen, M., Hentenryck, P.V., Deville, Y.: Constraint satisfaction approach to parametric differential equations. In: Proceedings of the 17th International Conference on AI, International Joint Conference on Artificial Intelligence (2001)
3. van Hoeve, W.J., Pesant, G., Rousseau, L.M.: On global warming : Flow-based soft global constaints. Journal of Heuristics (2006) To appear.
4. Carlsson, M., Beldiceanu, N.: Arc-consistency for a chain of lexicographic ordering constraints. Technical report T2002-18, Swedish Institute of Computer Science (2002) ftp://ftp.sics.se/pub/SICS-reports/Reports/SICS-T–2002-18–SE.ps.Z.
5. Beldiceanu, N., Carlsson, M., Petit, T.: Deriving filtering algorithms from constraint checkers. In Wallace, M., ed.: Proceedings of 10th International Conference on Principles and Practice of Constraint Programming (CP2004), Springer (2004) 107–122
6. Hellsten, L., Pesant, G., van Beek, P.: A domain consistency algorithm for the stratch constraint. In Wallace, M., ed.: Proceedings of 10th International Conference on Principles and Practice of Constraint Programming (CP2004), Springer (2004) 290–304
7. Golden, K., Pang, W.: Constraint reasoning over strings. In Rossi, F., ed.: Proceedings of Ninth International Conference on Principles and Practice of Constraint Programming (CP2003), Springer (2003) 377–391