Filtering Atmost1 on Pairs of Set Variables

Willem-Jan van Hoeve[1] and Ashish Sabharwal[2⋆]

[1] Tepper School of Business, Carnegie Mellon University
[2] Department of Computer Science, Cornell University

1 Introduction

Many combinatorial problems, such as bin packing, set covering, and combinatorial design, can be conveniently expressed using set variables and constraints over these variables [3]. In constraint programming such problems can be modeled directly in their natural form by means of *set variables*. This offers a great potential in exploiting the structure captured by set variables during the solution process, for example to break problem symmetry or to improve domain filtering.

We present an efficient filtering algorithm, establishing bounds consistency, for the `atmost1` constraint on pairs of set variables with fixed cardinality. Computational results on social golfer benchmark problems demonstrate that with this additional filtering, these problems can be solved up to 50 times faster.

2 Domain Filtering for Set Constraints

A *set variable* is a variable whose domain values are sets. As the number of possible values of a set variable can be enormous (the size of a power set, in the worst case), one usually represents the domain of a set variable S by an interval $[L(S), U(S)]$, where $L(S)$ and $U(S)$ are a 'lower' and 'upper' bound on the values that S can take. In addition, a lower bound $l(S)$ and upper bound $u(S)$ on the *cardinality* of S are maintained. A natural (and widely adopted) representation for the domain of set variables is based on the *subset ordering* of the domain. That is, the lower bound $L(S)$ represents all *mandatory* elements, while the upper bound $U(S)$ represents all *possible* elements, i.e., $D(S) = \{s \mid L(S) \subseteq s \subseteq U(S), l(S) \leq |S| \leq u(S)\}$. We refer to this representation as the subset+cardinality representation. It is applied in CP solvers such as ILOG Solver, Eclipse, and Gecode.

For constraints involving set variables, the filtering task is to increase the lower bounds and decrease the upper bounds of the domains such that we achieve *bounds consistency*, which should formally be called *subset+cardinality-bounds consistency* in our case:

Definition 1. *Let $S_1, \ldots, S_n$ be set variables. A constraint $C(S_1, \ldots, S_n)$ is called* subset+cardinality-bounds consistent *if for all $i = 1, \ldots, n$, $L(S_i)$ and*

⋆ This research was partly supported by the Intelligent Information Systems Institute, Cornell University under AFOSR Grant FA-9550-04-1-0151.

$U(S_i)$ are the intersection and the union, respectively, of all values in $D(S_i)$ that can be assigned to S_i in a solution to C, while in addition $l(S_i)$ and $u(S_i)$ are equal to the minimum and maximum cardinality over these values, respectively.

When a filtering algorithm for set constraints does not necessarily establish bounds consistency, we call it a *partial* filtering algorithm.

The Atmost1 Constraint on Pairs of Set Variables

The `atmost1` constraint was introduced by Sadler and Gervet [5] and specifies, for a collection of n set variables with given cardinalities, that each pair of variables overlaps in at most one element. Filtering the `atmost1` constraint to bounds consistency is NP-hard [1]. Therefore, Sadler and Gervet [5] give in on bounds consistency and present a partial filtering algorithm. In this work, we given in on the number of variables instead, and consider the `atmost1` constraint involving two set variables only, which we will refer to as the `pair-atmost1` constraint. Formally, `pair-atmost1`$(S_1, S_2, c_1, c_2) = \{(s_1, s_2) \mid s_1 \in D(S_1), s_2 \in D(S_2), |s_1| = c_1, |s_2| = c_2, |s_1 \cap s_2| \leq 1\}$, where S_1 and S_2 are set variables and $c_1, c_2 \geq 1$ are integers representing the cardinalities of S_1 and S_2, respectively.

A natural way of implementing the `pair-atmost1` constraint is to use the following decomposition of `pair-atmost1`(S_1, S_2, c_1, c_2) into three constraints: $|S_1| = c_1, \quad |S_2| = c_2, \quad |S_1 \cap S_2| \leq 1$. We will refer to this as the *decomposition* for `pair-atmost1`. Unfortunately, filtering these constraints separately does not establish bounds consistency on the `pair-atmost1` constraint, as illustrated by the following example:

Example 1. Let $D(S_1) = [\{1, 2\}, \{1, 2, 3, 5, 6\}]$, $D(S_2) = [\{3\}, \{1, 2, 3, 4\}]$, and $c_1 = c_2 = 3$. Establishing bounds consistency on `pair-atmost1`(S_1, S_2, c_1, c_2) leads to $D(S_1) = [\{1, 2\}, \{1, 2, 5, 6\}]$, $D(S_2) = [\{3, 4\}, \{1, 2, 3, 4\}]$. This will not be achieved by the decomposition.

3 The Bounds Consistency Filtering Algorithm

We next present the filtering algorithm that establishes bounds consistency on the `pair-atmost1` constraint, which we call BC-FilterPairAtmost1 (shown as Algorithm 1).

First, we partition each of $D(S_1)$ and $D(S_2)$ into six disjoint sets. For this purpose we define L1= $L(S_1)$ and P1= $U(S_1) \setminus L(S_1)$, i.e., L1 represents the lower bound, and P1 the possible values, for S_1. We define L2 and P2 similarly for $D(S_2)$. Using these shorthands, we define the partition of $D(S_1)$ into L1only = $L1 \setminus U(S_2)$, L1L2 = $L1 \cap L2$, L1P2 = $L1 \cap P2$, P1L2 = $P1 \cap L2$, P1P2 = $P1 \cap P2$, and P1only = $P1 \setminus U(S_2)$. $D(S_2)$ is similarly partitioned into L2only, L2L1, L2P1, P2L1, P2P1, and P2only. Note that L1L2 = L2L1, P1L2 = L2P1, and P2L1 = L1P2. For these three pairs, we explicitly maintain only one set per pair, namely, L1L2, P1L2, and P2L1, respectively. (While P1P2 = P2P1 as well, we still need to maintain both of these sets.)

```
BC-FilterPairAtmost1(S1, S2, c1, c2)
begin
    Scan L(S1), U(S1), L(S2), and U(S2) to compute the cardinality of each of the 9 sets:
        L1only, L2only, L1L2, P1only, P2only, P1L2, P2L1, P1P2, P2P1
    Initialize the 'can-have' and 'not-necessary' flags of each of the 9 sets to FALSE
    if |L1L2| > 1 then Fail
    if |L1L2| = 1 then
        Perform BC-CASE0(c1 − 1, c2 − 1, nil)
        Perform BC-UPDATEDOMAINS
        Return
    // |L1L2| = 0
    Perform BC-CASE0(c1, c2, nil)    // no shared element
    for each s ∈ { P1L2, P2L1, P1P2, P2P1 } do
        // possible solution has a shared element from s
        if  BC-CASE0(c1 − 1, c2 − 1, s) then s.can-have ← TRUE
    Perform BC-UPDATEDOMAINS
end
sub BC-CASE0(c1, c2, s)
begin
    k1 ← c1 − (|L1only| + |L1L2| + |P2L1|); if s = P2L1 then k1++
    k2 ← c2 − (|L2only| + |L1L2| + |P1L2|); if s = P1L2 then k2++
    slack1 ← (|P1only| + |P1P2|) − k1
    slack2 ← (|P2only| + |P2P1|) − k2
    slack3 ← (|P1only| + |P2only| + |P1P2|) − (k1 + k2)
    if (slack1 ≥ 0) and (slack2 ≥ 0) and (slack3 ≥ 0) then
        // solution exists
        P1only.can-have ← TRUE;  P2only.can-have ← TRUE
        P1L2.not-necessary ← TRUE;  P2L1.not-necessary ← TRUE
        if slack1 > 0 then
            P2P1.can-have ← TRUE;  P1P2.not-necessary ← TRUE
            if slack3 > 0 then  P1only.not-necessary ← TRUE
        if slack2 > 0 then
            P1P2.can-have ← TRUE;  P2P1.not-necessary ← TRUE
            if slack3 > 0 then  P2only.not-necessary ← TRUE
        return TRUE;
    else
        return FALSE;
end
sub BC-UPDATEDOMAINS
begin
    for each s ∈ { P1L2, P2L1, P1P2, P2P1 } do
        if s.can-have = FALSE or s.not-necessary = FALSE then
            for all y ∈ s computed by re-scanning L(S1), U(S1), L(S2), U(S2) do
                if s.can-have = FALSE then  Remove y from U(Si) for corresponding i
                if s.not-necessary = FALSE then  Add y to L(Si) for corresponding i
end
```

Algorithm 1: Bounds consistency domain filtering for `pair-atmost1`.

Example 2. For the scenario of Example 1, we have L1 $= \{1, 2\}$, P1 $= \{3, 5, 6\}$, L2 $= \{3\}$, and P2 $= \{1, 2, 4\}$. The 9 sets in this case are: L1only $= \varnothing$, L2only $= \varnothing$, L1L2 $= \varnothing$, P1only $= \{5, 6\}$, P2only $= \{4\}$, P1L2 $= \{3\}$, P2L1 $= \{1, 2\}$, P1P2 $= \varnothing$, and P2P1 $= \varnothing$.

For each of the 9 sets, we maintain two Boolean flags: The "can-have" flag and the "not-necessary" flag, that are all initialized to FALSE. Some of them will be set to TRUE during the course of the algorithm when we find a solution. If at the end, for a set s, s.can-have is still FALSE, we remove s from the upper bound of the corresponding domain. If s.not-necessary is still FALSE, we add s to the lower bound.

We find a solution by comparing the cardinalities of the 9 sets. In our *base case* (BC-CASE0), we assume that the variables already have one element in common. For S_1 we need $k_1 = c_1 - |L(S_1)| - 1$ additional values (or one more, if the common element was in $L(S_1)$). Similarly, we need k_2 more values for S_2. If we can meet the demand (verified by nonnegativity of slack1, slack2, and slack3 Algorithm 1), there exists a solution, and we update the flags for our 9 sets.

When we are not in the base case, i.e., L1L2 = 0, there are two possibilities. First, there could be a solution in which there is no common element. For this we run the base case, as is. Second, there will be a shared element, originating from P1L2, P2L1, P1P2, or P2P1. For each of these possibilities, we 'remove' the shared element from S_1 and S_2, which brings us in the base case again.

Theorem 1. *Algorithm 1 establishes bounds consistency on the* `pair-atmost1` *constraint.*

Theorem 1 can be proved by a careful case analysis. The time complexity of BC-FILTERPAIRATMOST1 is dominated entirely by the creation of the 9 sets during search, which takes $O(n)$ time where n is the integer domain size. The rest of the algorithm has only a constant number of calls to BC-CASE0 and one call to BC-UPDATEDOMAINS. BC-UPDATEDOMAINS takes time $O(n + k \log n)$, where k is the number of elements removed from an upper bound or added to a lower bound, assuming standard set operations used for maintaining these upper and lower bounds take time $O(\log n)$. We can tighten this analysis by amortizing over an entire path in the search tree from the root to any leaf, such that the total filtering complexity is $O(n \log n)$, while updating the flags takes total time $O(n)$, for the path.

4 Experimental Results

We evaluated the performance of the `pair-atmost1` constraint on the well-known social golfer problem (problem prob010 in CSPLib). The problem golf-g-s-w asks for a partition of n golfers into g groups, each of size s, for w weeks, such that no two golfers are in the same group more than once throughout the whole schedule. We apply the following standard model, using set variables S_{ij} to represent the set of golfers of week i and group j:

$$
\begin{array}{ll}
\texttt{partition}(S_{i1}, \ldots, S_{ig}, \{1, \ldots, n\}), & 1 \le i \le w \\
\texttt{pair-atmost1}(S_{ij}, S_{kl}, s, s), & 1 \le i < k \le w, 1 \le j \le g, 1 \le l \le g \\
|S_{ij}| = s, & 1 \le i \le w, 1 \le j \le g \\
S_{ij} \in [\varnothing, \{1, \ldots, n\}], & 1 \le i \le w, 1 \le j \le g.
\end{array}
$$

To speed up the computation, we also applied a redundant global cardinality constraint [4] on integer variables x_{ij} representing the group in which golfer j plays in week i. Our search strategy is a smallest-domain-first on these variables.

Problem	**Decomposition** (partial filtering)		**BC-FilterPairAtmost1** (bounds consistency)	
	time (s)	**backtracks**	**time (s)**	**backtracks**
`golf-6-5-5`	2106.7	10,986,224	75.5	239,966
`golf-6-5-4`	1517.7	10,930,370	39.7	197,837
`golf-6-5-3`	1060.5	10,930,016	29.6	197,607
`golf-6-5-2`	635.5	10,879,368	17.2	171,664
`golf-8-4-4`	226.7	1,555,561	157.7	738,393
`golf-10-3-10`	128.1	150,911	67.2	78,976
`golf-10-3-9`	86.0	150,452	52.4	78,613
`golf-10-3-6`	21.3	110,429	17.3	57,364
`golf-10-4-5`	51.3	310,110	4.5	22,044
`golf-10-4-4`	42.5	310,109	4.0	22,043
`golf-7-4-4`	22.5	184,641	4.4	27,877

Table 1. *Computational results on a number of social golfer instances.*

Finally, to account for some symmetry-breaking, we partly instantiate some of the set variables before starting the search, following Fahle et al. [2]. We note that our filtering algorithm can be applied to any model, including those with more advanced symmetry-breaking techniques.

We implemented our model in ILOG Solver 6.3, and all experiments run on a 3.8 GHz Intel Xeon machine with 2 GB memory running Linux 2.6.9-22.ELsmp. We evaluated the performance of the decomposition implementation of `pair-atmost1` (achieving partial filtering) with our filtering algorithm BC-FilterPairAtmost1 (achieving bounds consistency) on a number of instances, as reported in Table 1. The results demonstrate that using the bounds consistency algorithm, one can solve these instances up to 50 times faster, with a similar reduction in the number of search tree backtracks.

References

[1] C. Bessiere, E. Hebrard, B. Hnich, and T. Walsh. Disjoint, partition and intersection constraints for set and multiset variables. In *CP'04*, volume 3258 of *LNCS*, pages 138–152, 2004.

[2] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In *CP'01*, volume 2239 of *LNCS*, pages 93–107, 2001.

[3] C. Gervet. Constraints over structured domains. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*. Elsevier, 2006.

[4] J.-C. Régin. Generalized arc consistency for global cardinality constraint. In *AAAI'96*, volume 1, pages 209–215, 1996.

[5] A. Sadler and C. Gervet. Global reasoning on sets. In *Proc. of Workshop on Modelling and Problem Formulation (FORMUL'01)*, 2001.