Open Constraints in a Closed World

Willem-Jan van Hoeve[1] and Jean-Charles Régin[2⋆]

[1] Department of Computer Science, Cornell University, Ithaca, NY 14853, USA
vanhoeve@cs.cornell.edu
[2] ILOG Sophia Antipolis, Les Taissounières HB2, 1681 route des Dolines,
06560 Valbonne, France
regin@ilog.fr

Abstract. We study domain filtering algorithms for *open constraints*, i.e., constraints that are not a priori defined on specific sets of variables. We present an efficient filtering algorithm, achieving set-domain consistency, for open global cardinality constraints. We extend this result to conjunctions of them, in case they are defined on disjoint sets of variables. We also analyze the case when the sets of variables may overlap. As establishing set-domain consistency is NP-complete in that case, we propose a weaker, though efficient, filtering algorithm instead. Finally, we extend our results to conjunctions of similar open constraints.

1 Introduction

Traditionally, constraint programming has focused on solving problems in closed-world scenarios: all variables and constraints are fixed from the beginning. In many real-life applications, however, the scope of a constraint may not be defined a priori. Instead, the variables on which the constraints are defined may only be revealed during the solution process. This happens very often in scheduling applications and other distributed settings.

For example, consider a set of activities and suppose that each activity can be processed either on the factory line 1 formed by the set of unary resources R_1, or on the factory line 2 formed by the set of unary resources R_2. Thus, at the beginning, the set of resources that will be used by an activity is not known. Also the set of activities that will be processed by a resource is not known. However, it is useful to express that the activities that will be processed on each line must be pairwise different. This can be done by defining two `alldifferent` constraints, involving the start variables of each activity, and by stating that a start variable will be involved in exactly one `alldifferent` constraint. Initially, each `alldifferent` constraint is defined on a set of variables formed by all start variables. Then this set will be modified when it can be proved that a variable cannot be a member of an `alldifferent` constraint (i.e., the corresponding activity cannot be processed on the corresponding factory line), or that a start variable (activity) will be processed on the specific factory line.

⋆ A large part of this work was carried out while the author was at Cornell University.

Constraints of this nature are called *dynamic constraints* [2] or *open constraints* [4, 5]. For instance, the above `alldifferent` constraints are examples of open constraints. In this case, they live in a *closed world*, because the set of possible variables is explicitly known. The extension of constraint programming with open constraints is called *open constraint programming* [4, 5].

The use of efficient domain filtering algorithms is a key element in solving problems with constraint programming. This is particularly true when the filtering is based on a *global constraint*, i.e., a constraint that encapsulates and exploits a substructure of the problem. Efficient filtering algorithms for *open global constraints* therefore have high potential to improve the solution process of open constraint programming, together with its rich application area. Nevertheless, such filtering algorithms have not yet been proposed, until now.

In this work we study the problem of filtering open global constraints in a closed world. We focus in particular on conjunctions of open *global cardinality constraints*, or `gcc`s, because of their practical applicability and generality. We present an efficient filtering algorithm, obtaining "set-domain" consistency, when the scopes of the `gcc`s are restricted to non-overlapping sets of variables. We also analyze the case when the scopes of the `gcc`s are allowed to share variables. In that case obtaining domain consistency is NP-complete. Hence we propose a weaker, though efficient, filtering algorithm.

Our filtering algorithms are based on techniques from flow theory. In fact, we are able to generalize the techniques used in the filtering algorithm for the original (closed) global cardinality constraint to conjunctions of open global cardinality constraints. It furthermore allows us to filter the domains of the set variables that underly the open global cardinality constraints. Finally, we extend our results to conjunctions of arbitrary flow-based open global constraints. Due to the application of efficient flow theoretic techniques, our algorithms are also efficient.

The outline of this paper is as follows. In Section 2, we present definitions and other preliminaries. In Section 3 we outline the general problem and give a motivating example. Section 4 describes our main result, the filtering algorithm for conjunctions of open global cardinality constraints on non-overlapping sets of variables. In Section 5 we consider the case where the sets of variables may overlap. In Section 6 we present a filtering algorithm for the set variables that underly the open constraints. In Section 7 we extend our results to conjunctions of similar open global constraints. Finally, we conclude in Section 8.

2 Background

2.1 Constraint Programming

Let x be a variable. The *domain* of x is a finite set of elements (also called domain values) that can be assigned to x. It is denoted by $D(x)$. For a set of variables X we define $D(X) = \cup_{x \in X} D(x)$.

A *set variable* is a variable whose domain values are sets. We often represent the domain of a set variable S by an "interval" $[L, U]$, where L and U are sets, such that $D(S) = \{s \mid L \subseteq s \subseteq U\}$. For example, let V be a set, and let S be a set variable with domain $D(S) = [\varnothing, V]$. Then $D(S)$ is the power set of V, i.e., it contains all possible subsets of V.

Let $X = \{x_1, x_2, \ldots, x_k\}$ be a set of variables. A *constraint* C on X is defined as a subset of the Cartesian product of the domains of the variables in X, i.e., $C \subseteq D(x_1) \times D(x_2) \times \cdots \times D(x_k)$. We say that X is the *scope* of C. A tuple $(d_1, \ldots, d_k) \in C$ is called a *solution* to C. We also say that the tuple *satisfies* C. C is *inconsistent* if it does not contain a solution. Otherwise, C is called *consistent*.

Sometimes a constraint C is defined on variables X together with a certain set of parameters p. In such cases, we denote the constraint as $C(X, p)$ for syntactical convenience, while admissible tuples are still of size $|X|$.

Next we introduce open constraints. For the purpose of this paper, we define a constraint to be *open* when its scope is the domain of a set variable whose domain values are sets of variables. The explicit representation of the domain of this set variable reflects that the constraint lives in a closed world. For example, let X be a set of variables and let S be a set variable with domain $D(S) = [\varnothing, X]$. The constraint $C(D(S))$ is an open constraint, whose scope depends on the actual instantiation of S. We also write $C(S)$ as a shorthand for $C(D(S))$, if there is no confusion.

A *constraint satisfaction problem*, or a *CSP*, is defined by a finite set of variables X, together with a finite set of constraints C, each on a subset of X. The goal is to find an assignment $x = d$ with $d \in D(x)$ for all $x \in X$, such that all constraints are satisfied simultaneously. This assignment is called a *solution to the CSP*. Note that by using set variables to define the scope of open constraints, we maintain this common definition of a CSP.

The solution process of constraint programming interleaves *constraint propagation*, and *search*. The search process essentially consists in enumerating all possible combinations of variable domain values, until we find a solution to the CSP or prove that none exists. We say that this process constructs a *search tree*. To reduce the exponential number of combinations, we *filter* the domains of the variables and *propagate* this information through all constraints:

> Given the current domains and a constraint C, remove domain values that do not belong to a solution to C. This is repeated for all constraints until no more domain values can be removed.

We typically apply constraint propagation at each node in the search tree. In order to be effective, filtering algorithms should be efficient, because they are applied many times during the solution process. Furthermore, they should remove as many domain values that are not part of a solution as possible. If a filtering algorithm for a constraint C removes *all* such values from the domains with respect to C, we say that it makes C *domain consistent*:

Definition 1 (domain consistency). *A constraint C on the variables $x_1, x_2, \ldots, x_k$ is called* domain consistent *if for each variable x_i and each domain value $d_i \in D(x_i)$ $(i = 1, \ldots, k)$, there exists a domain value $d_j \in D(x_j)$ for all $j \neq i$ such that $(d_1, \ldots, d_k) \in C$.*

In the literature, domain consistency is also referred to as *hyper-arc consistency* or *generalized-arc consistency*. Note that domain consistency does only guarantee that each individual constraint contains a solution; it does *not* guarantee that the CSP has a solution.

If we make an open constraint $C(S)$ domain consistent, we should remove from the domain of S all sets s of variables for which $C(s)$ has no solution. As set variables are only represented by an interval, we use *bounds consistency* for this purpose instead:

Definition 2 (bounds consistency). *An open constraint C on the set variables $S_1, S_2, \ldots, S_k$ is called* bounds consistent *if for each S_i and each $s_i \in \{\min S_i, \max S_i\}$ $(i = 1, \ldots, k)$, there exist sets $s_j \in [\min S_j, \max S_j]$ for all $j \neq i$ such that $C(s_1, \ldots, s_k)$ has a solution.*

Rather than filtering the domain of the set variable S, however, we would like to filter the domain of the actual variables that appear in any instantiation of S. Hence, we next introduce a slight variant of domain consistency for open constraints that captures exactly this:

Definition 3 (set-domain consistency). *An open constraint $C(S)$ is called* set-domain consistent *if for each variable $x \in \{y \mid y \in s, s \in D(S)\}$ and all domain values $d \in D(x)$ there exists a set $s' \in D(S)$ with $x \in s'$ such that $x = d$ belongs to a solution of $C(s')$.*

By introducing this notion of set-domain consistency, we separate the filtering of the set variable S and the variables that appear in its domain. An open constraint $C(S)$ can hence be made set-domain consistent, while $C(S)$ itself may not be bounds consistent.

2.2 Flow Theory

In this section we present some concepts of flow theory that are necessary to understand this paper. For more information on flow theory we refer to [1].

Let $G = (V, A)$ be a directed graph, or *digraph*, with vertex set V and arc set A. Let $s, t \in V$. A function $f : A \to \mathbb{R}$ is called a *flow from s to t*, or an *s-t flow*, if

$$\begin{array}{ll} (i) & f(a) \geq 0 \qquad\qquad\qquad\quad \text{for each } a \in A, \text{ and} \\ (ii) & f(\delta^{\text{out}}(v)) = f(\delta^{\text{in}}(v)) \text{ for each } v \in V \setminus \{s, t\}, \end{array}$$

where $\delta^{\text{in}}(v)$ and $\delta^{\text{out}}(v)$ denote the multiset of arcs entering and leaving v, respectively. Here $f(S) = \sum_{a \in S} f(a)$ for all $S \subseteq A$. Property (ii) ensures *flow conservation*, i.e., for a vertex $v \neq s, t$, the amount of flow entering v is equal to the amount of flow leaving v.

As we will always consider s-t flows in this paper, we will often speak of a *flow* instead of s-t flow, for convenience. Furthermore, we say that an arc a *belongs* to a flow if $f(a) > 0$.

Let $d : A \to \mathbb{R}_+$ and $c : A \to \mathbb{R}_+$ be a "demand" function and a "capacity" function, respectively[3]. We say that a flow f is *feasible* if $d(a) \leq f(a) \leq c(a)$ for each $a \in A$.

Let f be an s-t flow in G. The *residual graph* of G with respect to f, c and d is defined as $G_f = (V, A_f)$ where for each $(u, v) \in A$,

- if $f(u, v) < c(u, v)$ then $(u, v) \in A_f$ with residual demand $\max\{d(u, v) - f(u, v), 0\}$ and residual capacity $c(u, v) - f(u, v)$, and
- if $f(u, v) > d(u, v)$ then $(v, u) \in A_f$ with residual demand 0 and residual capacity $f(u, v) - d(u, v)$.

Finally, a digraph $G = (V, A)$ is *strongly connected* if for any two vertices $u, v \in V$ there is a directed path from u to v. A maximally strongly connected non-empty subgraph of a digraph G is called a *strongly connected component* of G.

3 Open Global Cardinality Constraints

A *global cardinality constraint* (`gcc`) on a set of variables specifies for each domain value in the union of their domains an upper and lower bound to the number of variables that are assigned to this value:

Definition 4 (global cardinality constraint). *Let $X = \{x_1, \ldots, x_n\}$ be a set of variables, and let $l_d, u_d \in \mathbb{N}$ for all $d \in D(X)$. Then*

$$\texttt{gcc}(X, l, u) = \{(d_1, \ldots, d_n) \mid \forall i \in \{1, \ldots, n\}\ d_i \in D(x_i), \\ \forall d \in D(X)\ l_d \leq |\{d_i \mid d_i = d\}| \leq u_d\}.$$

A special case of the `gcc` is the `alldifferent` constraint, which specifies that all variables should be pairwise different. If we set $l_d = 0$ and $u_d = 1$ for all $d \in D(X)$, the `gcc` is equal to the `alldifferent` constraint. A filtering algorithm for the `gcc`, establishing domain consistency, was developed in [7], making use of network flows.

3.1 A Single Open Global Cardinality Constraint

We first consider the case of a single open `gcc`. In order to filter this constraint, we compute a flow in a particular graph, similar to the filtering of the original (closed) `gcc` [7].

Let X be a set of variables, and let S be a set variable with domain $[L, U]$, such that $L \subseteq U \subseteq X$. Let $\texttt{gcc}(S, l, u)$ be the open `gcc` under consideration. We build the following graph. The vertex set of the graph is composed of U, $D(U)$, a source s, and a sink t. The arc set is composed of:

[3] Here $\mathbb{R}_+$ denotes $\{x \in \mathbb{R} \mid x \geq 0\}$.

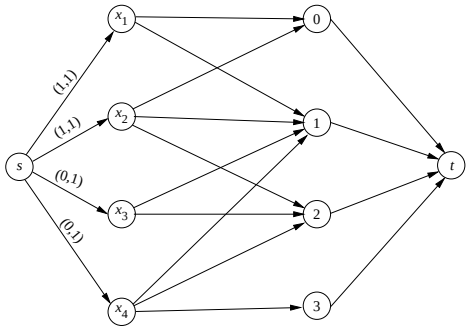

Fig. 1. Graph representation for the `gcc` of Example 1. For the (s, x_i) arcs, the demand d and capacity c is explicitly given as (d, c). All other arcs have demand 0 and capacity 1.

- Arcs (s, x) for all $x \in U$ with capacity 1. If $x \in L$, then this arc has demand 1, otherwise its demand is 0.
- Arcs (x, d) for all $x \in U$, $d \in D(x)$ with demand 0 and capacity 1,
- Arcs (d, t) for all $d \in D(U)$, with demand l_d and capacity u_d.

Call the resulting graph $\mathcal{G}_{\text{single}}$. We have the following result:

Theorem 1. *A feasible integer flow in* $\mathcal{G}_{\text{single}}$ *corresponds to a solution to* $\texttt{gcc}(S, l, u)$ *and vice versa.*

Proof. Let f be a feasible integer flow in $\mathcal{G}_{\text{single}}$. We construct a solution to $\texttt{gcc}(S, l, u)$ by defining $S = \{x \mid f(s, x) = 1\}$ and $x = d$ for all $x \in S, d \in D(x)$ with $f(x, d) = 1$.

Conversely, given a solution to $\texttt{gcc}(S, l, u)$ we construct a feasible integer flow f in $\mathcal{G}_{\text{single}}$ by defining

for all $x \in U$: $f(s, x) = 1$ if $x \in S$, and $f(s, x) = 0$ otherwise,
for all $x \in U$ and $d \in D(x)$: $f(x, d) = 1$ if $x = d$, and $f(x, d) = 0$ otherwise,
for all $d \in D(U)$: $f(d, t) = |\{x \mid x \in S, x = d\}|$.

□

Example 1. Let $X = \{x_1, x_2, x_3, x_4\}$ be a set of variables with integer domains: $x_1 \in \{0, 1\}$, $x_2 \in \{0, 1, 2\}$, $x_3 \in \{1, 2\}$, and $x_4 \in \{1, 2, 3\}$. Let $S \in [\{x_1, x_2\}, X]$ be a set variable. Furthermore, let $l_d = 0$ and $u_d = 1$ for all $d \in \{0, 1, 2, 3\}$.

The graph representation for the constraint $\texttt{gcc}(S, l, u)$ is presented in Figure 1. In fact, this `gcc` corresponds to an `alldifferent` constraint for this choice of l and d. Note that the demand of the arcs (s, x_1) and (s, x_2) is 1, because S must include $\{x_1, x_2\}$.

By applying Theorem 1, we get the following result:

Corollary 1. *The constraint* $\texttt{gcc}(S, l, u)$*, where* $S \in [L, U]$*, is set-domain consistent if and only if for all* $x \in U$ *and* $d \in D(x)$*, there exists an arc* (x, d) *with* $d \in D(x)$ *that belongs to a feasible integer flow in* $\mathcal{G}_{\text{single}}$*.*

The proof is immediate because there is a one to one correspondence between a solution of $\texttt{gcc}(S, l, u)$ and a feasible flow in $\mathcal{G}_{\text{single}}$.

Corollary 1 gives rise to a set-domain consistency algorithm, similar to the original (closed) `gcc`. First, we compute an initial feasible integer flow f in $\mathcal{G}_{\text{single}}$. This can be done in $O(nm)$ time, where $n = |U|$ and $m = \sum_{x \in U} D(x)$. If f does not exist, the constraint is not consistent. Otherwise, we identify and remove all arcs (x, d) that do not belong to any feasible integer flow. As indicated by [7], inconsistent arcs are exactly those that do not belong to a strongly connected component in the residual graph $(\mathcal{G}_{\text{single}})_f$. Computing the strongly connected components of $(\mathcal{G}_{\text{single}})_f$ can be done in $O(n + m)$ time [9], where n and m are defined as above. Moreover, our algorithm is incremental. When k variables have changed their value, we can recompute the flow in $O(km)$ time and re-establish domain consistency in $O(n + m)$ time.

3.2 The Conjunction of Open Global Cardinality Constraints

Next we consider a more general problem; the conjunction of several open `gcc`s. First, consider the following motivating example.

Example 2. Let $X = \{x_1, x_2, x_3, x_4, x_5\}$ be a set of variables with integer domains: $x_1 \in \{0, 1\}$, $x_2 \in \{0, 1\}$, $x_3 \in \{0, 1\}$, $x_4 \in \{0, 1\}$ and $x_5 \in \{0, 1, 2, 3, 4, 5\}$. Let $S_1 \in [\varnothing, X]$ and $S_2 \in [\varnothing, X]$ be set variables. We define the following conjunction of constraints:

$$\begin{array}{l} \texttt{alldifferent}(S_1) \wedge \\ \texttt{alldifferent}(S_2) \wedge \\ (S_1 \cup S_2) = X \wedge \\ (S_1 \cap S_2) = \varnothing. \end{array} \tag{1}$$

Here $\texttt{alldifferent}(S_1)$ and $\texttt{alldifferent}(S_2)$ are open constraints, as they are defined on the domain of set variables whose domain values are sets of variables.

From conjunction (1) we are able to deduce that

$$\begin{array}{l} 2 \leq |S_1| \leq 3, \\ 2 \leq |S_2| \leq 3, \\ x_5 \in \{2, 3, 4, 5\}. \end{array}$$

Namely, as x_1, x_2, x_3 and x_4 all have domain $\{0, 1\}$, no more than two of them can appear in one `alldifferent` constraint. Since we need to include all variables into both constraints, we have that $2 \leq |S_1| \leq 3$ and $2 \leq |S_2| \leq 3$. Moreover, each `alldifferent` constraint will involve exactly two variables from $\{x_1, x_2, x_3, x_4\}$, and the variables x_1, x_2, x_3 and x_4 will saturate the values 0 and 1 in both `alldifferent` constraints. Hence those values are removed from the domain of x_5.

Our general problem is the conjunction of k open gccs. Let X be a set of variables, and let $S_1, S_2, \ldots, S_k$ be set variables, with respective domains $[L_i, U_i]$, such that $L_i \subseteq U_i \subseteq X$ $(i = 1, \ldots, k)$. The conjunction of k open gccs is defined as:

$$\bigcap_{1 \leq i \leq k} \texttt{gcc}(S_i, l^i, u^i), \tag{2}$$

where $l^i_d, u^i_d \in \mathbb{N}$ for all $d \in D(X)$ and $i = 1, \ldots, k$.

If the set variables $S_1, S_2, \ldots, S_k$ are not constrained, there is not much that can be deduced. We know that, for $1 \leq i \leq k$,

$$\sum_{d \in D(X)} l^i_d \leq |S_i| \leq \sum_{d \in D(X)} u^i_d,$$

but in general this is not sufficient to make further deductions with respect to the domains of the variables. Hence, we impose additional constraints on the set variables. In particular we distinguish the following four cases and combinations thereof:

$$(\bigcup_{1 \leq i \leq k} S_i) = X, \tag{3}$$

$$(\bigcup_{1 \leq i \leq k} S_i) \subset X, \tag{4}$$

$$\forall_{1 \leq i < j \leq k}\ S_i \cap S_j = \varnothing, \tag{5}$$

$$\exists_{1 \leq i < j \leq k}\ S_i \cap S_j \neq \varnothing. \tag{6}$$

For example, the combination of (3) and (5) restricts the set variables to be a partition of X. In the remainder of this paper we will study filtering algorithms for the conjunction of k open gccs, in combination with one or more of the constraints (3) up to (6).

4 Disjoint Set Variables

In this section, we present an efficient set-domain consistency filtering algorithm for k open gccs together with restriction (5), i.e., all set variables should be pairwise disjoint. Our work is based on the domain consistency filtering algorithm for the single gcc as developed in [7], and an extension of the algorithm presented above for a single open gcc.

Again, we base our algorithm on finding a flow in a particular graph. The key observation is that for each open gcc, one duplicates the corresponding variables and domain values, and associates the corresponding lower and upper bounds to each domain value. This allows us to build a graph similar to the graph of a single gcc, and also to apply similar efficient flow algorithms to establish set-domain consistency.

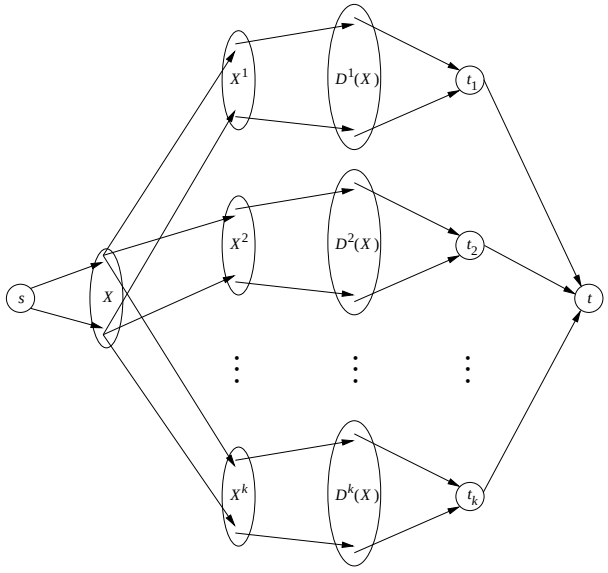

Fig. 2. Schematic graph representation for the conjunction of k open `gccs`.

We build our graph as follows; see Figure 2 for a schematic representation. In order to distinguish variables in different open `gccs`, we duplicate every variable $x \in U_i$ as x^i, for $i \in \{1, \ldots, k\}$, and denote the respective set of variables by X^i. We also duplicate the domain values $D(X)$ as $D^1(X), \ldots, D^k(X)$. Then the vertex set of the graph is composed of X, $X^1, \ldots, X^k$, $D^1(X), \ldots, D^k(X)$, a source s, "intermediate sinks" $t_1, \ldots t_k$ and a sink t. The arc set is composed of:

- arcs (s, x) for all $x \in X$, with demand 0 and capacity 1,
- arcs (x, x^i) for all $i \in \{1, \ldots, k\}$ and $x \in L_i$, with demand 1 and capacity 1,
- arcs (x, x^i) for all $i \in \{1, \ldots, k\}$ and $x \in U_i \setminus L_i$, with demand 0 and capacity 1,
- arcs (x^i, d) for all $i \in \{1, \ldots, k\}$, $x \in U_i$ and $d \in D^i(x)$, with demand 0 and capacity 1,
- arcs (d, t_i) for all $i \in \{1, \ldots, k\}$ and $d \in D^i(X)$, with demand l^i_d and capacity u^i_d,
- arcs (t_i, t) for all $i \in \{1, \ldots, k\}$, with demand $|L_i|$ and capacity $|U_i|$.

Call the resulting graph $\mathcal{G}$. Note that we may actually omit arcs (x, x^i) if $x \in L_j$ for some $j \neq i$. This follows from the disjointness of the set variables. We nevertheless prefer the above description, because it can be easily extended to the non-disjoint case, as we will see in Section 5.

We have the following result:

Theorem 2. *A feasible integer flow in $\mathcal{G}$ corresponds to a solution to the conjunction of (2) and (5) and vice versa.*

Proof. Let f be a feasible integer flow in $\mathcal{G}$. We construct a solution to the conjunction of (2) and (5) by defining $S_i = \{x \mid f(x, x^i) = 1\}$ $(1 \leq i \leq k)$ and $x = d$ for all $x \in X, d \in D(x)$ with $f(x^i, d) = 1$ for some $i \in \{1, \ldots, k\}$.

Conversely, given a solution to the conjunction of (2) and (5), we construct a feasible integer flow f in $\mathcal{G}$ by defining

for all $x \in X$: $f(s, x) = 1$ if $x \in S_i$ for some $i \in \{1, \ldots, k\}$, and $f(s, x) = 0$ otherwise,
for all $x \in X$ and $i \in \{1, \ldots, k\}$: $f(x, x^i) = 1$ if $x \in S_i$, and $f(x, x^i) = 0$ otherwise,
for all $x \in X$, $i \in \{1, \ldots, k\}$ and $d \in D^i(x)$: $f(x^i, d) = 1$ if $(x = d) \wedge (x \in S_i)$, and $f(x^i, d) = 0$ otherwise,
for all $i \in \{1, \ldots, k\}$ and $d \in D^i(X)$: $f(d, t_i) = |\{x \mid x \in S_i, x = d\}|$,
for all $i \in \{1, \ldots, k\}$: $f(t_i, t) = |\{x \mid x \in S_i\}|$.

□

Notice that if $L_i \cap L_j \neq \varnothing$ for some $i \neq j$, there is no feasible flow in $\mathcal{G}$, because the demand requirements on the arcs involving S_i and S_j cannot be fulfilled.

An illustration applied to Example 2 is given in Figure 3. In Figure 3.a we present the graph $\mathcal{G}$ corresponding to this example. In Figure 3.b we present a feasible flow in $\mathcal{G}$, corresponding to the solution $S_1 = \{x_1, x_2, x_5\}$, $S_2 = \{x_3, x_4\}$, $x_1 = 0$, $x_2 = 1$, $x_3 = 0$, $x_4 = 1$ and $x_5 = 5$.

By applying Theorem 2, we get the following result.

Corollary 2. *The conjunction of (2) and (5) is set-domain consistent if and only if for all $x \in X$ and $d \in D(x)$, there exists an arc (x^i, d) with $d \in D^i(x)$ for some $1 \leq i \leq k$, that belongs to a feasible integer flow in $\mathcal{G}$.*

The proof follows from the one to one correspondence between a solution of the conjunction of (2) and (5), and a feasible flow in $\mathcal{G}$. Note that Theorem 2 does not consider the set variables on which the `gcc`s are defined. We will deal with them in Section 6.

Similar to the above single open `gcc`, we apply Corollary 2 to design a set-domain consistency algorithm. First, we compute an initial feasible integer flow f in $\mathcal{G}$. This can be done in $O(nm)$ time, where $n = |X|$ and $m = k \cdot \sum_{x \in X} D(x)$. If f does not exist, the conjunction is not consistent. Otherwise, we identify and remove all arcs (x^i, d) that do not belong to any feasible integer flow. Note however, that one arc (x^i, d) with $d \in D^i(x)$ for some $1 \leq i \leq k$ is already sufficient to make $d \in D(X)$ consistent. Again, inconsistent arcs are exactly those that do not belong to a strongly connected component in the residual graph $\mathcal{G}_f$. Computing the strongly connected components of $\mathcal{G}_f$ can again be done in $O(n+m)$ time, where n and m are defined as above. Also this algorithm is incremental. When l variables have changed their value, we can recompute the flow in $O(lm)$ time and re-establish domain consistency in $O(n + m)$ time.

As an example, consider again Figure 3. In Figure 3.c we show the residual graph with respect to the flow given in Figure 3.b. Figure 3.d shows the graph after filtering, i.e., all inconsistent arcs are removed.

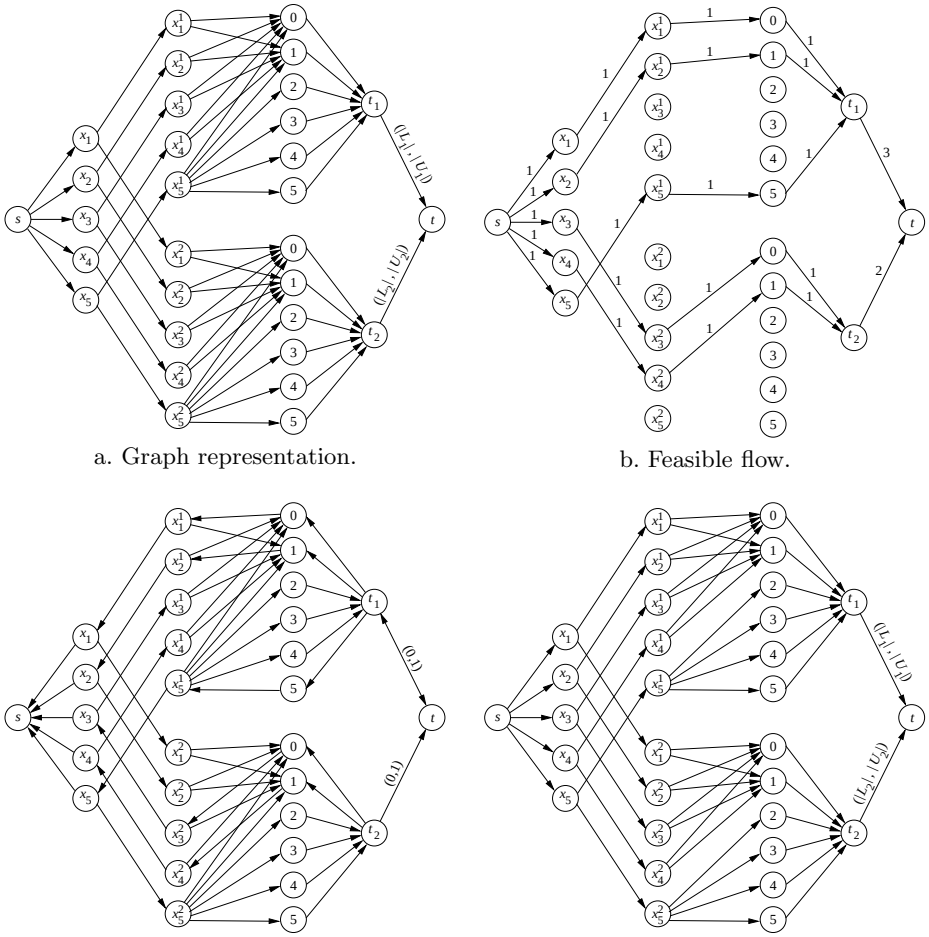

Fig. 3. Graph representation for the conjunction of `alldifferent`(S_1) and `alldifferent`(S_2) on disjoint set variables S_1 and S_2, where $S_1 \cup S_2 = \{x_1, x_2, x_3, x_4, x_5\}$ (following Example 2). Arcs (t_1, t) and (t_2, t) have demand and capacity determined by the cardinality of the lower bounds L_1, L_2 and upper bounds U_1, U_2 of the respective set variables. All other arcs have demand 0 and capacity 1.

5 Non-Disjoint Set Variables

We next study the conjunction of k open `gcc`s together with restriction (6), i.e., the set variables are allowed to be non-disjoint.

Unfortunately, establishing set-domain consistency in this case is an NP-complete problem. Namely, in [3], it is proved that establishing domain consistency on the conjunction of `alldifferent` constraints on overlapping sets of variables is NP-complete. As the `alldifferent` constraint is a special case of

the gcc, their result immediately implies that our task is NP-complete. To overcome the NP-completeness, we propose to relax the requirement of establishing set-domain consistency.

We use the same graph representation as in the previous section, with one modification. In the previous section, the graph representation does not allow a variable to appear in several gccs. Namely, because the capacity of the arcs (s, x) for $x \in X$ is 1, each variable can be assigned to at most one value in $D^1(X), \ldots, D^k(X)$. This means that each variable can only occur in a single open gcc. We relax this by defining the capacity of the arcs (s, x) to be k. On the positive side, this allows a variable to occur in several open gccs at the same time, which yields a filtering method for the non-disjoint case. On the negative side, a variable may take different values in different gccs, which is likely to weaken the filtering. With this modification, however, we can apply the same efficient algorithm as in the previous section.

6 Filtering the Set Variables

In this section we consider the filtering of the set variables on which the open constraints are defined. As stated earlier, we would like to establish bounds consistency with respect to these variables.

Consider again the conjunction of k open gccs (2) and restriction (5), i.e., the constraints are defined on disjoint sets of variables. We can use the graph $\mathcal{G}$ to establish bounds consistency on the set variables $S_1, \ldots, S_k$ as well. To this end, we apply the following three rules for all $1 \leq i \leq k$:

i) when there is no arc between x^i and $D^i(X)$, then x is removed from S_i, i.e., $U_i := U_i - x$,
ii) when there are only arcs between x^i and $D^i(X)$, and $f(x, x^i) = 1$ for all feasible flows f, then x is added to S_i, i.e., $L_i := L_i + x$,
iii) $|L_i| \geq \min\{f(t_i, t) \mid f \text{ feasible flow}\}$ and $|U_i| \leq \max\{f(t_i, t) \mid f \text{ feasible flow}\}$.

When we apply these rules, we know by Theorem 2 that we have established bounds consistency with respect to the set variables. The application of the above three rules can be done in $O(k^2nm)$ time, by subsequently computing minimum and maximum flows.

7 Extension

In the above, we have focused on conjunctions of gccs because of their generality and applicability to real-life problems with open constraints. The results can easily be extended to similar cases, however.

7.1 Optimization Constraints

A first extension is to apply our technique to optimization constraints. For example, consider a conjunction of open *weighted* global cardinality constraints [8]. In

that case, a weight is assigned to each pair (x, d), for all $x \in X$ and $d \in D(X)$. Then a solution to the problem induces a weight, defined by the sum of the weight of the pairs (x, d) if $x = d$ is in the solution. The aim is to find a solution with minimum total weight.

We can handle this case similar to the original filtering algorithm for weighted `gccs` [8]. With each arc (x^i, d), for all $x \in X^i$ $(i = 1, \ldots, k)$ and $d \in D(x)$ in $\mathcal{G}$, we associate a cost that is equal to the weight of this pair. Then a solution to the conjunction of open weighted `gccs` corresponds to a minimum-cost feasible flow in the graph. Hence, the cost-based version of our filtering algorithm is immediate.

7.2 Soft Constraints

Soft constraints can be viewed as special optimization constraints. A number of soft global constraints can be represented by a flow in a graph, similar to the `gcc`, see [6]. In this case, rather than associating a cost to an arc (x, d), for all $x \in X^i$ $(i = 1, \ldots, k)$ and $d \in D(x)$, costs may appear on "any" arc in the graph. We can again apply the same machinery as for the open weighted `gccs` to open soft global constraints.

7.3 Mixture

Finally, it is also possible to apply our results to a mixture of open constraints, provided that they are reasonably compatible. For example, we can group together open `alldifferent` constraints and open `gccs` in one conjunction. Another example is to join together open soft `gccs` and open weighted `alldifferent` constraints.

8 Conclusion

For the first time, we have proposed filtering algorithms for open global constraints. We have in particular studied open global cardinality constraints and conjunctions of them. We have proposed an efficient filtering algorithm, based on techniques from flow theory, establishing set-domain consistency, when the constraints are defined on disjoint sets of variables. In case the constraints are defined on non-disjoint sets of variables, this task becomes NP-complete. For that case we have proposed a weaker, but efficient, filtering algorithm. We have also presented a bounds consistency filtering algorithm for the set variables that underly these open constraints. Finally, we have shown how to extend our results to other conjunctions of open constraints, for example optimization constraints and soft global constraints.

References

1. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows.* Prentice Hall, 1993.

2. R. Barták. Dynamic Global Constraints in Backtracking Based Environments. *Annals of Operations Research*, 118(1–4):101–119, 2003.
3. K. Elbassioni, I. Katriel, M. Kutz, and M. Mahajan. Simultaneous matchings. In X. Deng and D. Du, editors, *Proceedings of the 16th Annual International Symposium on Algorithms and Computation (ISAAC 2005)*, volume 3827 of *LNCS*, pages 106–115. Springer, 2005.
4. B. Faltings and S. Macho-Gonzalez. Open Constraint Satisfaction. In P. Van Hentenryck, editor, *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP 2002)*, volume 2470 of *LNCS*, pages 356–370. Springer, 2002.
5. B. Faltings and S. Macho-Gonzalez. Open Constraint Programming. *Artificial Intelligence*, 161(1–2):181–208, 2005.
6. W.-J. van Hoeve, G. Pesant, and L.-M. Rousseau. On Global Warming: Flow-Based Soft Global Constraints. *Journal of Heuristics*, 2006. To appear.
7. J.-C. Régin. Generalized Arc Consistency for Global Cardinality Constraint. In *Proceedings of AAAI/IAAI*, volume 1, pages 209–215. AAAI Press/The MIT Press, 1996.
8. J.-C. Régin. Cost-Based Arc Consistency for Global Cardinality Constraints. *Constraints*, 7:387–405, 2002.
9. R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.