Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 20, no. 2, pp. 363–410 (2016)
DOI: 10.7155/jgaa.00397

Propagation Rules for Graph Partitioning Constraints

Radosław Cymer

Abstract

We review existing methods and present a generic propagation mechanism for graph partitioning constraints based on directed matchings. The task is also to give a set of several propagation rules according to specific partition properties. Every solution of the global constraint corresponds to a subgraph of the corresponding digraph associated with the constraint. The filtering identifies the arcs of the digraph that do not belong to a solution. We illustrate this principle on some common global constraints.

Submitted:	Reviewed:	Revised:	Accepted:	Final:
October 2015	April 2016	May 2016	May 2016	June 2016

Published:
June 2016

Article type:
Regular paper

Communicated by:
G. Liotta

E-mail address: r.cymer@web.de (Radosław Cymer)

1 Introduction

Graphs are often used to model different various computational problems. A number of combinatorial problems rely on the partitioning of a graph into a set of components. The usual graph partitioning problem comprises of dividing a graph into smaller subgraphs with specific properties, such that the subgraphs are isomorphic and each vertex of the graph belongs to exactly one of these components. Important applications of graph partitioning problems include many areas of mathematics and computer science.

Typically, graph partitioning problems fall under the category of intractable problems. Even for special graph classes it can be shown that no reasonable fully polynomial algorithms exist for these graphs. Solutions to these problems are generally derived using heuristics and approximation algorithms. One competitive approach to solving such problems is constraint programming (CP). The main motivation for this work is to expand the results of constraint programming to graph partitioning problems by proposing more effective filtering algorithm for the graph partitioning constraints.

Recently, the graph partitioning problem was gained importance due to its application in constraint programming. Some works about global constraints and constraint programming have appeared in the journals of operations research [31],[47],[61],[16]. In papers [15, 17] filtering algorithms were introduced that use the semantic of the constraint in terms of maximum matching. In this paper, we extend this concept to global constraints representable by directed graphs. We present a paradigm based on a directed matching. We illustrate our method with a complete study of specific global constraints.

Constraints that describe partitions of the vertices in a given initial graph have been considered from an early stage of constraint programming research. Some examples include the PROPER_FOREST [8], TOUR [17], CLIQUE [23],[51] (modeled using undirected graphs), CIRCUIT [38],[37], CYCLE [5], TREE [6],[22], PATH [9] (modeled as directed graphs), and COST_TOUR [16], SHORTER_PATH [55], WEIGHTED_SPANNING_TREE [52],[53] (modeled by weighted graphs) constraints. Most of the problems involving these constraints are intractable in general. This work goes one step further by introducing a set of specific propagation rules for global constraints on directed graphs.

The paper provides a new constraint programming technique to solve graph partitioning problems on digraphs. It is based on a combination of a new method to decompose a digraph into subgraphs by exploiting strongly connected components and dominators, and algorithms for solving the maximum matching problem. In principle this gives a somehow generic method to filter these global constraints.

This work describes a unified approach to propagating graph partitioning constraints such as CIRCUIT, CYCLE, DERANGEMENT (and its soft version), TREE, BINARY_TREE, PATH and MAP. Our goal is to show that a solution to one of them can be used to solve them all. The method is based on the reduction of some digraph matching problem (called directed matching) to the maximum matching problem on a bipartite graph. Decomposition theory for bipartite graphs is then used to come up with specific algorithms for filtering the arcs of the respective digraph representing the partition.

The approach consists of two steps. First, a solution representing a given partition in a digraph is constructed. This can be done by using a directed matching and any standard algorithm for finding a matching in a bipartite graph. Next, a set of allowed, forbidden and mandatory arcs is determined. The results provide a general algorithm for solving any graph partitioning problem. In all these situations the problem of interest can be formulated as the matching problem defined over the appropriate bipartite graph.

We want to point out that our goal is not to partition a given digraph D associated with a global constraint, but rather to find out whether it is possible to make and detect those arcs of D that do not belong to any partition corresponding to a specific pattern.

This paper is organized as follows. In Section 2 we first present the necessary formal background on constraint programming and graph theory. Section 3 deals with an algorithmic method based on the decomposition theory of directed graphs. We present an alternative way of computing strongly connected components in a directed graph, suited for our technique detailed later. Section 4 is the central part of the paper. A general tool is first presented (based on matching theory) which is then used to give new filtering algorithms for several graph partitioning constraints. The studied problems are formulated as a matching problem on a suitable bipartite graph associated with the input digraph corresponding to the constraint. In this section examples are given and filtering algorithms are developed. We demonstrate how decomposition theory applies to such problems. We also describe a well-known decomposition method which uses the strong components of D and present a more powerful decomposition method based on the dominators of D. Finally, in Section 5 we review related results, discuss future work and conclude.

The following tables summarize all the theoretical results for graph partitioning constraints we will deal with in this paper. The first table gives an overview of complexities. Here n denotes the number of vertices and m is the number of edges. The second table gives an exhaustive list of the basic properties for all graph partitioning constraints discussed in this paper.

global constraint	model	complexity	checking feasibility	hyper-arc consistency	Section
CIRCUIT	digraph/graph	NP-hard	–	–	4.1
CYCLE	digraph/bigraph	NP-hard	–	–	4.2
DERANGEMENT	digraph/bigraph	polynomial	$\mathcal{O}(\sqrt{n} \cdot m)$	$\mathcal{O}(m+n)$	4.3
SOFT_DERANGEMENT_VAR	digraph/bigraph	polynomial	$\mathcal{O}(\sqrt{n} \cdot m)$	$\mathcal{O}(m+n)$	4.4
TREE	digraph	linear	$\mathcal{O}(m+n)$	$\mathcal{O}(m+n)$	4.5
BINARY_TREE	digraph/bigraph	NP-hard	–	–	4.6
	DAG/bigraph	polynomial	$\mathcal{O}(\sqrt{n} \cdot m)$	$\mathcal{O}(m+n)$	
PATH	digraph/bigraph	NP-hard	–	–	4.7
	DAG/bigraph	polynomial	$\mathcal{O}(\sqrt{n} \cdot m)$	$\mathcal{O}(m+n)$	
MAP	digraph	NP-hard	–	–	4.8

Table 1: Summary of results for graph partitioning constraints (complexities)

global constraint	pattern	count variable	lower bound	upper bound	continuity property
CIRCUIT	cycle factor	−			
CYCLE	cycle factor	NCYCLE	non-sharp	non-sharp	-
DERANGEMENT	cycle factor	−			
SOFT_DERANGEMENT_VAR	cycle factor	−			
TREE	anti-arborescence	NTREE	sharp	sharp	+
BINARY_TREE	anti-arborescence	NTREE	non-sharp	sharp	+
PATH	path factor	NPATH	non-sharp	sharp	+
MAP	functional graph	NBCYCLE	sharp	non-sharp	+
		NBTREE	non-sharp	non-sharp	+

Table 2: Summary of results for graph partitioning constraints (properties)

2 Preliminaries

We start with formal definitions of the central concepts. We first recall some necessary terminology of the theory of digraphs, matching theory and constraint programming that we will use in the rest of the paper. We assume that the reader is familiar with the essentials of complexity theory (for a review, see [25], [39, Chapter 3] or [14, Chapter 34]).

2.1 Theory of Digraphs

We use standard terminology but for the sake of clarity we repeat the most important definitions and notations from [28]:

A *digraph* (directed graph) D is a pair (V, E), where V is a finite set of elements, called *vertices* (or *nodes*), and $E \subseteq V \times V$ is a set of ordered pairs (v_i, v_j) of vertices, called *arcs* (or *directed edges*). The number of vertices $n = |V|$ is the *order* of D. The number of arcs $m = |E|$ is the *size* of D.

A digraph H is a *subdigraph* of D if $V(H) \subseteq V(D)$ and $E(H) \subseteq E(D)$. If $V(H) = V(D)$, then H is called a *spanning subdigraph* (or a *factor*) of D.

If (v_i, v_j) is an arc of D, then v_i is called the *head* (or *initial endpoint*) and v_j is called the *tail* (or *terminal endpoint*). Graphically, the vertices can be represented by points, and (v_i, v_j) will be represented by an arrow connecting the points v_i and v_j, v_j being at the tip of the arrow.

An arc whose endpoints coincide is called a *loop*. Two arcs, or edges, are called *adjacent* if they have at least one endpoint in common. We consider graphs without multiple (parallel) edges but which may contain loops.

Vertex u is called a *successor* of vertex v if there is an arc with v as its initial endpoint and u as its terminal endpoint. The set of all successors of v is denoted by

$$\Gamma^+(v) = \{u \in V : (v, u) \in E\}.$$

Similarly, vertex u is called a *predecessor* of vertex v if there exists an arc of the form (u, v). The set of all predecessors of vertex v is denoted by

$$\Gamma^-(v) = \{u \in V : (u, v) \in E\}.$$

The set of all *neighbors* of v is denoted by

$$\Gamma(v) = \Gamma^+(v) \cup \Gamma^-(v).$$

For a set $X \subseteq V$, we let

$$\Gamma^+(X) = \bigcup_{x \in X} \Gamma^+(x) \text{ and } \Gamma^-(X) = \bigcup_{x \in X} \Gamma^-(x).$$

The *outward degree* (or *out-degree* for short) of a vertex v in D, denoted by $d^+(v)$, is the number of arcs starting at (leaving) v and the *inward degree* (or *in-degree* for short) of a vertex v, denoted by $d^-(v)$, is the number of arcs terminating at (entering) v. The *total degree* (or just *degree*) of a vertex v, denoted by $d(v)$, is defined by

$$d(v) = d^+(v) + d^-(v).$$

Clearly, we have:

$$d^+(v) = \left|\Gamma^+(v)\right| \text{ and } d^-(v) = \left|\Gamma^-(v)\right|.$$

It is easy to see that the sum of in-degrees of all vertices equals the sum of out-degrees of all vertices and both are equal to the number of arcs in the digraph:

$$\sum_{v \in V} d^+(v) = \sum_{v \in V} d^-(v) = m.$$

If $d^+(v) = d^-(v) = 0$, the vertex v is said to be an *isolated node*; if $d^+(v) \neq 0$ and $d^-(v) = 0$, the vertex v is called a *source*; if $d^+(v) = 0$ and $d^-(v) \neq 0$, the vertex v is called a *sink*. We use subscripts (e.g. $d^+_D(v)$) to specify the digraph D if the usage is not clear from the context.

A *directed path* P of length k is a sequence of $k+1$ distinct vertices $v_0, v_1, \ldots, v_k$ together with the k distinct arcs $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$. The vertex v_0 is the *initial endpoint* (or *source*) of the path P, the vertex v_k is the *terminal endpoint* (or *target*) of the path P, and the remaining vertices are the *internal nodes* of the path P. A *directed circuit* is a (directed) path that begins and ends at the same vertex.

A digraph is called *strongly connected* (or *strong*) if, for any two vertices v_i and v_j, there exists a path from v_i to v_j and a path from v_j to v_i. Finding the strong components in a directed graph has a variety of applications. Tarjan was the first to obtain an elegant linear time algorithm with complexity $\mathcal{O}(m+n)$ to compute the strong components of a digraph [58]. His algorithm utilizes *depth first search* in a clever way. Tarjan's algorithm, its complexity and correctness, is presented in the textbook [14].

A *strong component* of a digraph D is a maximal strongly connected subdigraph of D. We assume that the *trivial digraph*, consisting of exactly one vertex is vacuously strong since it does not contain two distinct vertices. Corresponding to any digraph D, there is a new digraph whose definition is based on the strong components of D. Let $S_1, S_2, \ldots, S_p$ be the strong components of a digraph D. The *strong component graph* (also called the *condensation*) of D, is the simple digraph $SC(D)$ with vertex set $V(SC(D)) = \{s_1, s_2, \ldots, s_p\}$, such that there is an arc in digraph $SC(D)$ from vertex s_i to vertex s_j if and only if there is an arc in digraph D from a vertex in strong component S_i to a vertex in strong component S_j. Multiple arcs from a given strongly connected component to another strongly connected component are merged.

Notice that the strong component graph of any digraph (some authors use the term *reduced digraph* to describe such digraphs) is a directed acyclic graph, that is, it contains no directed cycles. Note that at this point we allow acyclic digraphs to contain loops. An acyclic digraph is often referred to by its abbreviation, DAG. The term DAG is typically pronounced as a word, not spelled out as an acronym.

A *topological ordering* of a directed graph D is a total order $\prec$ on the nodes such that $u \prec v$ for every arc (u, v). Less formally, a topological ordering arranges the nodes horizontally or vertically so that all arcs point from one item to the next. A topological ordering is not possible if the digraph D has a cycle, since for two nodes v and w on the cycle, both $v \prec w$ and $w \prec v$. Furthermore, the topological ordering is not necessarily unique. Any DAG has at least one topological ordering, which can be found in linear time [14, Section 22.4].

Acyclic digraphs play a very important role in both theory and applications of digraphs and form a well-studied family of digraphs, in particular, due to the following important properties:

- In every directed acyclic graph there is at least one source (a vertex with no incoming edges: $d^-(v) = 0$) as well as at least one sink (a vertex of out-degree zero: $d^+(v) = 0$).
- Every acyclic digraph has a topological ordering of its vertices.
- For every vertex v there is a source s such that there is a path from s to v, and there is a sink t such that there is a path from v to t.

Clearly, in any digraph all the vertices on a cycle belong to the same strongly connected component. A strongly connected component will be called a *source component* if it corresponds to a source vertex in a strong component graph. Analogously, a strongly connected component of D that corresponds to a sink of $SC(D)$ is called a *sink component.* A strongly connected component is *trivial* if it consists of one vertex without a loop, and is *non-trivial* otherwise.

In order to obtain short proofs of various results on subdigraphs or efficiently solve many problems dealing with directed graphs the following transformation of a directed graph D to a bipartite graph is extremely useful [28, page 411].

Let D be a directed graph of order n and size m. Associated with D is a bipartite graph $BR(D)$ with two color classes $V_1 = \{v'_1, \ldots, v'_n\}$ and $V_2 = \{v''_1, \ldots, v''_n\}$. The elements of V_1 are called *outward*, and the elements of V_2 are called *inward*. A pair $\{v'_i, v''_j\}$ is an edge of $BR(D)$ if there exists an arc of the form (v_i, v_j) in D. We call $BR(D)$ the *bipartite representation* of D.

The *augmented bipartite graph* corresponding to the directed graph D is graph $BR^*(D)$ defined as the graph with the same vertex sets V_1 and V_2 as $BR(D)$. Every edge of $BR(D)$ is an edge of $BR^*(D)$ and in addition the n unordered pairs $\{v'_i, v''_i\}$, $i = 1, 2, \ldots, n$ are edges of $BR^*(D)$.

2.2 Matching Theory

Recall that a matching in an undirected graph $G = (V, E)$ is a set of edges from E, no two of which share a vertex, a maximum matching of G is a matching of maximal cardinality among all matchings of G and a perfect matching is a set of pairwise disjoint edges that cover all the vertices of G. For arbitrary graphs finding a maximum matching fast is quite complicated and it was a great breakthrough when Edmonds [20] found a polynomial algorithm. For both bipartite and general graphs the perfect matching problem is solvable in time $\mathcal{O}(\sqrt{n} \cdot m)$ [33],[46].

The notion of a matching is sometimes too restrictive and so we define a more general concept, a degree-matching. A degree-matching is a set of edges M where, for each vertex x of the graph, we restrict the number of edges of M incident with x to be within a given interval. More formally, let g and f be integer-valued functions, called *degree conditions*, such that $0 \leq g(x) \leq f(x) \leq d(x)$. A perfect f-matching is a degree-matching such that the number of edges belonging to M incident with vertex x equals $f(x)$. Analogously, a perfect (g, f)-matching is a degree-matching such that the number of matched edges incident with x has to be between $g(x)$ and $f(x)$. Naturally, if $g(x) = 0$ and $f(x) = 1$ for every vertex x then a degree-matching becomes a standard matching. Note that a perfect 2-matching is a collection of vertex-disjoint cycles, and an acyclic (1,2)-matching is a collection of vertex-disjoint paths.

An elementary graph is a graph such that the union of perfect matchings forms a connected spanning subgraph. An elementary bipartite graph is a graph in which every edge is contained in some perfect matching. The similar definitions can be given for graphs with degree-matchings.

Let $D = (V, E)$ be a digraph with vertex set $V(D)$ and arc set $E(D)$. Further, let $\vec{g} = (g^-, g^+)$ and $\vec{f} = (f^-, f^+)$ be pairs of non-negative integer-valued functions defined on $V(D)$ such that $0 \leq g^-(x) \leq f^-(x) \leq d^-_D(x)$ and $0 \leq g^+(x) \leq f^+(x) \leq d^+_D(x)$ for every $x \in V(D)$. We say that the digraph D has a directed perfect $\vec{f}$-matching if there exists a spanning subdigraph $F \subseteq D$ such that $d^-_F(x) = f^-(x)$ and $d^+_F(x) = f^+(x)$ for all $x \in V(D)$. Analogously, a directed perfect $(\vec{g}, \vec{f})$-matching of D is defined to be a spanning subdigraph $H \subseteq D$ such that $g^-(x) \leq d^-_H(x) \leq f^-(x)$ and $g^+(x) \leq d^+_H(x) \leq f^+(x)$ for all $x \in V(D)$.

The existence of a directed perfect matching in a digraph is equivalent to the existence of a perfect matching in its corresponding bipartite representation. For given pairs of functions $\vec{g} = (g^-, g^+)$ and $\vec{f} = (f^-, f^+)$ defined on $V(D)$, we define two functions $g, f : V(BR(D)) \to \mathbb{Z}^+$ by

$$g(x) = \begin{cases} g^+(x), & \text{if } x \in V_1 \\ g^-(x), & \text{if } x \in V_2 \end{cases}$$

and

$$f(x) = \begin{cases} f^+(x), & \text{if } x \in V_1 \\ f^-(x), & \text{if } x \in V_2. \end{cases}$$

Then it is easy to see that D has a directed perfect $\vec{f}$-matching if and only if $BR(D)$ has a perfect f-matching, and that D has a directed perfect $(\vec{g}, \vec{f})$-matching if and only if $BR(D)$ has a perfect (g, f)-matching.

The following necessary and sufficient conditions for the existence of a directed perfect $(\vec{g}, \vec{f})$-matching are easily verified using matching theory (for bipartite graphs).

Theorem 1 *Let $D = (V, E)$ be a digraph with non-negative integer-valued functions $\vec{g} = (g^-, g^+)$ and $\vec{f} = (f^-, f^+)$ defined on $V(D)$. If D has a directed perfect $(\vec{g}, \vec{f})$-matching then*

$$\sum_{x \in X} g^+(x) \leq \sum_{x \in \Gamma^+(X)} f^-(x)$$

and

$$\sum_{x \in X} g^-(x) \leq \sum_{x \in \Gamma^-(X)} f^+(x)$$

for all $X \subseteq V$.

Proof: The proof of this theorem is analogous as for Theorem 2.4.5 in [44]. □

A cycle cover of an undirected graph is a spanning subgraph that consists solely of single cycles in which every vertex is a part of exactly one cycle. Cycle covers are also known as 2-factors since every vertex has degree two in a cycle cover.

A *cycle factor* in a directed graph $D = (V, E)$ is a spanning subdigraph of D in which the inward and outward degree of every vertex v is equal to 1:

$$d^+(v) = d^-(v) = 1.$$

Observe that a strongly connected digraph needs not necessarily have a cycle factor. We can use matching theory to find a cycle factor in a given digraph or to prove that none exists. We now begin with the necessary and sufficient condition for the existence of a cycle factor in a digraph.

Theorem 2 *A directed graph $D = (V, E)$ has a cycle factor if and only if*

$$|X| \leq \left| \bigcup_{x \in X} \Gamma^+(x) \right|$$

and

$$|X| \leq \left| \bigcup_{x \in X} \Gamma^{-}(x) \right|$$

for each $X \subseteq V$.

This looks, in fact, very much like a translation of Hall's Theorem [30] into the language of directed graphs. Indeed, it is practically the same result. It can be proven by applying the Hall's Theorem to a bipartite representation $BR(D)$ constructed from digraph D. It is easy to see that D has a cycle factor if and only if the bipartite graph $BR(D)$ contains a perfect matching.

A *path factor* of a digraph D is a spanning subdigraph, each of whose components is a path. Note that a directed matching does not exclude cycles, whereas a path factor is acyclic. It is obvious that if a digraph has a cycle factor, then it also has a path factor. The converse is not true.

2.3 Constraint Programming

We will assume that the reader is familiar with the basic results of constraint programming, which we briefly review now and state some useful terminology to make the notation clear. For a thorough explanation of this area we refer the reader to the monographs [2], [18] and [62].

A *domain variable x* is a variable ranging over a finite set of integers denoted by D_x. The minimum and maximum values of D_x are denoted by $\min(D_x)$ and $\max(D_x)$, respectively.

A constraint network (CN) consists of a set $\{x_1, \ldots, x_n\}$ of variables, a set $\{D_{x_1}, \ldots, D_{x_n}\}$ of domains which represent the set of possible values that each variable can take, and a set of constraints $C \subseteq D_{x_1} \times \ldots \times D_{x_n}$ which link up the variables and define the set of combination of values that are allowed. The search for an instantiation of all variables that satisfy all the constraints is called a Constraint Satisfaction Problem (CSP), and such an instantiation is called a solution to a CSP.

A filtering algorithm for the constraint satisfaction problem removes values from domains that do not participate in a solution to it. A propagation algorithm helps to identify and detect values that cannot be taken by the variables of the constraints in any solution of the problem.

The pruning is a task which shrinks the domain of each variable without changing the set of solutions. We say that the pruning is incomplete if it removes some inconsistent values but not every inconsistent value. A pruning is complete if the removal of any additional value from any domain would change the set of solutions.

Many constraints that appear in modeling, and for which specialized domain reduction propagation algorithms have been developed, are called *global constraints*. Very often a global constraint can be modeled by a graph and a solution can be represented as some type of a degree-matching problem. Several examples are given of encodings of global constraints where values (or rather assignments) in the constraints correspond to edges (see, for example, [19], [15, 17]).

Moreover, it is shown that one can generalize the techniques used in standard matching to find out which edges belong to all degree-matchings, some degree-matchings or no degree-matching. With respect to a specified degree-matching such edges will be, respectively, called mandatory, allowed or forbidden.

Classification of the edges is performed by applying the so-called alternating depth-first search on the graph G associated with the global constraint. The fundamental tool used in this method is the Dulmage-Mendelsohn (DM) and Gallai-Edmonds (GE) Decomposition, which are canonical decompositions of bipartite and general graphs based on the notion of matching. We need to traverse the edges of the graph with respect to an initial matching. An alternating depth-first search simulates the traversing on a directed graph and constructs layers that alternately use matched and free edges. Clearly, an alternating depth-first search maintains alternating cycles (for more details, see [15]). The algorithm can also be used to maintain the strongly connected components.

Analogously to the classical depth-first search the alternating depth-first search induces two numberings of the vertices of the traversed graph G, one in the order in which the vertices are reached by a search and one in the order in which the vertex exploration is completed. The two numbers associated with the vertex are usually called its discovery and finishing time number. The core of a strongly connected component is the vertex with the smallest discovery number.

Throughout this paper we will use the convention that in figures the solid lines indicate the edges of the graph, the thick solid lines denote matched edges and the dashed lines shown the forbidden edges. The edges marked by crosses x are forbidden, as well. The degree conditions of every vertex are shown in the brackets near the vertex in the auxiliary bipartite graph.

3 Canonical Decomposition

In this section we consider the canonical decomposition of directed graphs with respect to strong connectivity. As a consequence, we give a new linear time algorithm for the detection of strongly connected components in a directed graph. We will see that the strong components of a directed graph become the elementary subgraphs of an associated bipartite graph. The results presented in this section are mainly based on the joint work of Johnson, Dulmage and Mendelsohn [35].

Pruning consists of classifying the arcs of the digraph into three mutually exclusive sets: forbidden arcs – those arcs which cannot be in any strongly connected component, mandatory arcs – those arcs which are required and must be in any strongly connected component, and the remaining, allowed arcs – those arcs which may be in some strongly connected component, but its removal has no effect on the component.

Theorem 3 *If $BR^*(D)$ is the augmented bipartite graph corresponding to the directed graph D, then D is strongly connected if and only if $BR^*(D)$ is elementary.*

Proof: See Theorem 4 in [35]. □

The last property gives us immediately a simple filtering algorithm with respect to the constraint STRONGLY_CONNECTED [4]. A brute force approach with complexity of $\mathcal{O}(m^2)$ has been described in [19]. A filtering algorithm according to a strong connectivity looks as follows. First, we create graph $BR^*(D)$. Next, we start with an initial empty matching and iterate over $i = 1, \ldots, n$. We add to the initial matching the edge $\{v_i', v_i''\}$. Then, the application of the alternating depth-first search to the initial matching M results in hyper-arc consistency with respect to the STRONGLY_CONNECTED constraint (Figure 1).

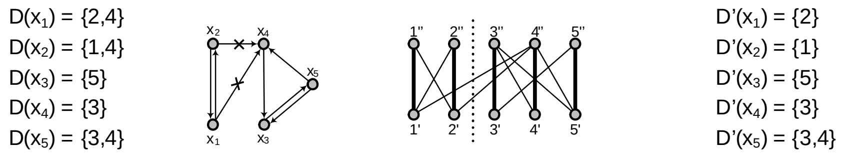

Figure 1: Pruning according to a strong connectivity

Let us explain our approach in some technical detail. A filtering algorithm for strong connectivity can be derived from properties of a perfect matching. It is based on the fact that an arc (i, j) belongs to a strongly connected component if the corresponding edge $\{i', j''\}$ is part of some perfect matching.

Consider the example in Figure 1. The problem is modeled as follows. On the left side of the figure a directed graph D in which we want to find strongly connected components is depicted. On the right side of the figure the augmented bipartite graph $BR^*(D)$ is presented. The bold edges in the graph denote an initial perfect matching.

The alternating depth-first search applied to this problem instance decomposes the graph into two elementary bipartite graphs. According to their properties the edge connecting two distinct elementary bipartite graphs are forbidden. Thus, since the edges $\{1', 4''\}$ and $\{2', 4''\}$ are forbidden in $BR^*(D)$, the corresponding arcs (x_1, x_4) and (x_2, x_4) don't belong to any strongly connected component of D.

Theorem 4 *For any directed graph D, let $G_1, \ldots, G_k$ be the canonical decomposition of $BR^*(D)$ into elementary subgraphs. If D_i, $i = 1, \ldots, k$, is the subdigraph of D such that $BR^*(D_i) = G_i$, then $D_1, \ldots, D_k$ are the strongly connected components of D.*

Proof: See Theorem 6 in [35]. □

Observe that our filtering routine which is used simultaneously with the method to find all strongly connected components does not detect mandatory arcs. Such arcs that are necessary for the strong connectivity of the digraph are called *strong bridges*. The strong bridges can be detected in linear time [34].

It is well-known (see, for example, Theorem 4 in [15]) that elementary subgraphs of a bipartite graph $G = (V_1 \cup V_2, E)$ with a perfect matching can be labeled in such a way that every edge in G from a subgraph G_i to a subgraph G_j with $i < j$ has one endpoint in V_2 and the other one in V_1. This property applies to the augmented bipartite graph $BR^*(D)$. In other words, the elementary subgraphs of $BR^*(D)$ encode the strongly connected components of the digraph D and moreover induce a topological ordering of the strong component graph $SC(D)$.

We now prove the theorem, which guarantees that the alternating depth-first search can be also used on $BR^*(D)$ in order to determine the topological ordering of $SC(D)$.

Theorem 5 *Let S_i and S_j be two strongly connected components of a directed graph D. If there is an arc (u, v) such that $u \in S_i$ and $v \in S_j$, then the core of S_i will have a higher finishing time than the core of S_j and be sorted first.*

Proof: There are two cases to consider. If alternating depth-first search visits any node in S_i before S_j, then clearly all of S_i and S_j will be explored before alternating depth-first search terminates. Therefore, the core of the strongly connected component S_i (the first node visited in S_i) will have a larger finishing time than any other node in S_j. On the other hand, if S_j is explored first, then alternating depth-first search will terminate after visiting all nodes of S_j but before visiting any node in S_i, in which case the property also holds. □

According to this theorem, a node with the largest finishing time must be in a source component. However, note that the theorem does not imply that a node with the smallest finishing time must be in a sink component. As a counterexample consider a digraph given by $V(D) = \{v_1, v_2, v_3\}$ with $E(D) = \{(v_1, v_2), (v_2, v_1), (v_1, v_3)\}$. Then $\{v_1, v_2\}$ is the source component, but v_2 will finish first.

The algorithm looks as follows. All steps can be performed in linear time.

Algorithm 1 Computing the strongly connected components

```
Require: Digraph $D$
Ensure: Strongly connected components of $D$, partition of arcs, and topological ordering of the nodes of $SC(D)$
  Construct the augmented bipartite graph $BR^*(D)$ associated with the digraph $D$
  Start with an initial perfect matching in $BR^*(D)$
  Perform an alternating depth-first search starting from some vertex of $V''$ (see Algorithm 2 in [15])
  Find the partition of mandatory (strong bridges), allowed and forbidden arcs
  Let the number of strongly connected components be the number of core vertices in the depth-first forest
  For every strongly connected component of $D$ determine its topological ordering number
```

Although decomposition using strong components is efficient and useful in practice, many graph partitioning problems have one or only a few strong components. Therefore, in the subsequent section of this paper we develop a more powerful decomposition technique based on dominators.

4 Graph Partitioning Constraints

In this section we present a graph-theoretic analysis of a directed matching, using matching theory. We show a direct reduction of the directed matching problem on a digraph to the maximum matching problem on a bigraph. This reduction yields an algorithm to determine the partition of edges in the directed matching making use of decomposition theory for bipartite graphs. In this section examples are given and filtering algorithms are developed.

In order to investigate global constraints we first introduce the digraph associated with any instance of these constraints[1]. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of n variables with respective finite domains $D_{x_i} \subseteq \{1, 2, \ldots, n\}$ for $i = 1, 2, \ldots, n$. To these variables we can associate the digraph $D = (V, E)$ with vertex set $V = \{v_i : 1 \leq i \leq n\}$ and arc set $E = \{(v_i, v_j) : j \in D_{x_i}, 1 \leq i, j \leq n\}$. Observe that the number of vertices is equal to the number of variables, and the number of arcs equals the sum of domain cardinalities. Thus, $n = |V|$ and $m = |E| = \sum |D_{x_i}|$ for all $x_i \in X$.

Further, we have

$$\begin{aligned} d^+(v_i) &= |D_{x_i}|, \\ d^-(v_i) &= |D_{x_i}^{-1}| = |\{j : i \in D_{x_j}\}|. \end{aligned}$$

Hence, a directed edge (v_i, v_j) exists if and only if j is in the domain of variable x_i. Moreover, elimination of an arc (v_i, v_j) from the associated digraph during the pruning means removing of the value j from the domain of x_i.

A digraph associated with a global constraint can be viewed as an undirected graph by forgetting the orientation of its arcs, removing loops and merging all multiple resulting edges. We call this graph the *underlying graph* associated with the global constraint. Clearly, elimination of an edge $\{v_i, v_j\}$ from the underlying graph during the pruning is equivalent both to the removing of the value j from the domain of variable x_i (if it exists) and the removing of the value i from the domain of variable x_j (if it exists).

Our algorithm to find a partition of edges is based on the following simple observation.

Theorem 6 *Let D be a digraph associated with the global constraint and assume that there exists a one-to-one correspondence between the solution of the constraint and the directed perfect $(\vec{g}, \vec{f})$-matching in D. Further, let $BR(D)$ denotes the bipartite representation of D. Then, the necessary condition to satisfiability of the constraint is that $BR(D)$ must have a perfect (g, f)-matching. Moreover, the mandatory (or forbidden) edges in a perfect (g, f)-matching of $BR(D)$ are mandatory (or forbidden) arcs in a directed perfect $(\vec{g}, \vec{f})$-matching of the digraph D.*

[1]We assume that the variables and their domain values represent the same set of elements.

Proof: Suppose that $BR(D)$ has a perfect (g, f)-matching M consisting of the edges $e_1, \ldots, e_{|M|}$. Then the arcs form a directed perfect $(\vec{g}, \vec{f})$-matching. Indeed, in the subdigraph D' induced by these arcs every vertex v_i has an out-degree and an in-degree equal to the number of matched edges incident to x_i on the outward side and to y_i on the inward side of $BR(D)$, respectively, and, according to the definition, such a subdigraph is precisely a directed perfect $(\vec{g}, \vec{f})$-matching in D. □

To illustrate this theorem, consider Figure 2.

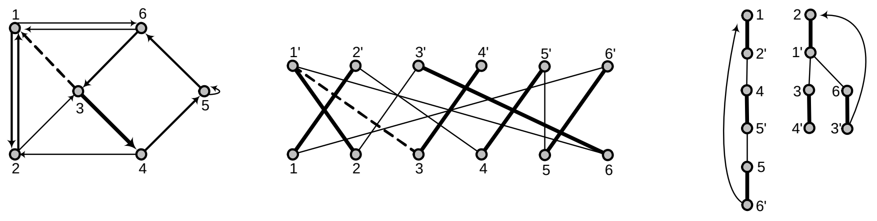

Figure 2: Pruning according to a directed matching

On the left side of the figure a directed graph D in which we wish to find a cycle factor is presented. In the middle of the figure the bipartite representation $BR(D)$ is depicted and a perfect matching M is shown. Observe that the perfect matching in $BR(D)$ corresponds to a cycle factor $1 \to 2 \to 1 \ \cup \ 3 \to 4 \to 5 \to 6 \to 3$ in D. In order to obtain the desired partition of edges we need only to apply an alternating depth-first search on $BR(D)$ with respect to M (for more details, see [15, Section 4]). The alternating depth-first forest (computed by the algorithm devised in [15]) is shown on the right side of the figure. Since edge $\{1', 3\}$ in $BR(D)$ is forbidden and edge $\{3, 4'\}$ is mandatory, arc $(3, 1)$ in D is forbidden (drawn as a dashed arrow), and arc $(3, 4)$ is mandatory (drawn as a bold arrow).

A graph partitioning constraint can be seen as a problem for finding a partial graph of a given digraph associated with the constraint. Graph partitioning constraints are the main subject of the thesis written by Lorca [42]. From an interpretation point of view the subdigraphs so obtained are called *functional graphs* and they have the characteristic property that the out-degree of each vertex is equal to 1 (i.e. every vertex of the partition has exactly one successor). Thus, we will always have $g^+(v) = f^+(v) = 1$ for all vertices of D which corresponds to $g(v) = f(v) = 1$ for all vertices on the outward side of the corresponding bipartite graph $BR(D)$.

The graph partitioning constraint can be formally defined as follows. The constraint can be written

$$\textsc{pattern}(\textsc{npattern},\ \textsc{nodes})$$

where NPATTERN is a count variable specifying the number of components in the graph partition, and NODES is a collection of n variables $\{x_1, \ldots, x_n\}$, whose domains consist of elements of $\{1, \ldots, n\}$, i.e. $D_{x_i} \subseteq \{1, \ldots, n\}$ for each i. An associated directed graph D contains an arc (x_i, x_j) if and only if j belongs to the domain of x_i. An arc (x_i, x_j) of D is selected when $x_i = j$ and the constraint enforces that the selected arcs form a satisfactory partition.

The constraint partitions a given associated digraph D described by the NODES collection into a set of vertex-disjoint components. The constraint requires that in the digraph D there are exactly NPATTERN components, such that every vertex of D belongs to exactly one component. The constraint holds if the associated digraph D is covered by a set NPATTERN functional graphs in such a way that each vertex of D belongs to one distinct component (to a single component).

By the *continuity property* for the count variable of a graph partitioning constraint we mean that if a digraph associated with the constraint can be decomposed into r and s connected components, where $r \leq s$, then it can be decomposed into k components for all k such that $r \leq k \leq s$.

In the following we will demonstrate the reduction technique on some graph partitioning constraints. The method is composed of two main phases. The first one focuses on checking if a constraint has a solution. The second phase makes it possible to find some arcs that do not participate in any solution. This phase can be split into three steps: bounds filtering of count variable(s), pruning forbidden arcs when the count variable is instantiated to one of its extrema, and structural filtering. The bounds filtering concentrates on the cardinality of the expected partition. It consists of removing the values of count variable that are out of range. The structural filtering detects the arcs that do not belong to the expected partition and reduces the domain variables of constraint. All steps are complementary and together form a (partial) filtering for a graph partitioning constraint.

It is now time to demonstrate our idea on concrete examples. For every global constraint we first give its formal definition and some applications, then we derive a transformation to the matching problem, and finally we discuss some related problems. All definitions are taken from the Global Constraint Catalog [4].

Some of the examples are illustrated by figures. On the left side of the figure the domains of the variables are given. In the middle of the figure, digraph associated with the global constraint and its corresponding bipartite representation are depicted and the partition of edges is highlighted. The thick edges indicate a matching, while the vertical dashed lines show forbidden edges. On the right side of the figure the reduced domains after pruning are presented.

4.1 The CIRCUIT constraint

The CIRCUIT constraint was first formulated by Laurière [38]. It can be viewed as describing a Hamiltonian circuit[2] on a directed graph D associated with the

[2]A digraph is *Hamiltonian* if it contains a directed circuit that visits each vertex once without touching any vertex more than once.

constraint. The constraint is defined as CIRCUIT(NODES), where NODES is a collection of variables $\{x_1, \ldots, x_n\}$ whose domains are subsets of $\{1, \ldots, n\}$. It requires that a tuple $(d_1, \ldots, d_n)$ be a cyclic permutation of $(1, \ldots, n)$, where each $d_{i+1} = x_{d_i}$ and $x_{d_n} = d_1$.

For the global constraint CIRCUIT there is a very immediate reduction from the Hamiltonian circuit, which demonstrates that reasoning with this constraint is generally intractable. For this reason, it is perhaps not surprising that, in the past, there has been little comment on it. Therefore, we now formally prove its computational intractability.

Theorem 7 *Deciding whether the* CIRCUIT *constraint has a solution is NP-complete.*

Proof: We use a transformation from DIRECTED HAMILTONIAN CIRCUIT [25, Problem GT38] into the CIRCUIT constraint. Given a digraph $D = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges. We construct a CIRCUIT constraint, CIRCUIT$(x_1, \ldots, x_n)$ in which $D_{x_i} = \{j : (v_i, v_j) \in E\}$. The constructed CIRCUIT constraint has a solution if and only if the original DIRECTED HAMILTONIAN CIRCUIT problem has a solution. □

A famous combinatorial problem that can be modeled with the CIRCUIT constraint is the *Traveling Salesperson Problem* (abbreviated as TSP) [39]. Many other problems can be expressed in terms of Hamiltonian cycles, such as the Euler Knight's Tour Problem on a chessboard, or the Chinese Postman Problem [44, Section 6.5].

Clearly, checking a CIRCUIT constraint for satisfiability is equivalent to checking if an associated digraph has a Hamiltonian circuit, which is an NP-complete problem [36]. There exist, however, several necessary conditions that can be verified in polynomial time.

An obvious necessary condition for a digraph to be Hamiltonian is that the graph must be strongly connected. However, this condition is not sufficient. Another obvious and quite powerful necessary condition for a digraph to be Hamiltonian is the existence of a cycle factor. Clearly, a Hamiltonian circuit is a cycle factor but the converse is not necessarily true because some cycle factors may consist of several disjoint circuits.

We know that achieving hyper-arc consistency for CIRCUIT is NP-hard. Therefore, we will now describe some useful incomplete filtering methods, that run in polynomial time, which are only partially related to those presented in [12],[37] and [57].

One of the elementary filtering methods for CIRCUIT is based on the ALLDIFFERENT constraint [49]. The ALLDIFFERENT filtering method can be applied because all the variables of the NODES collection have to take distinct values. A further necessary condition to satisfiability of this constraint is to have at most one single strongly connected component.

The filter removes inconsistent values by eliminating non-Hamiltonian arcs from the associated digraph, that is, arcs that belong to no Hamiltonian circuit. Filtering can also be based on sufficient conditions for non-Hamiltonicity of a digraph, some of which appear in [39, Chapter 11]. Most of the known sufficient conditions for a digraph D to be Hamiltonian assert that if the degrees of the vertices of D are sufficiently large [3],[45],[64], or D has enough arcs [41], then D is Hamiltonian. Unfortunately, the number of arcs must be nearly as large as the number of edges in a complete graph with n vertices.

We have taken care of many sufficient conditions for a digraph to have a Hamiltonian circuit. However, none of these are necessary conditions. For example, the oriented cycle C_n, the simplest Hamiltonian digraph of all, does not satisfy any of these conditions when n is large.

Kaya and Hooker presented in [37] a recursive algorithm that eliminates non-Hamiltonian arcs from the graph via vertex separators. Their filter is based on an idea put forward by Chvátal that every Hamiltonian graph is 1-tough[3] (for details, see [39, page 405]). Their algorithm identifies almost all unsatisfiable instances and eliminates about one-third of the inconsistent values from the variable domains. The authors give a filtering algorithm for the CIRCUIT constraint but its description in the book by Hooker [32] contains some flaws which can be confusing. For example, the filtering based on the ALLDIFFERENT constraint will remove the dashed arcs from the sample graphs depicted in Figure 3.15, contrary to the claim stated for two examples that the both situations are independent.

In fact, the alldiff filtering seems to be stronger than the vertex-degree filtering described in [32, Section 3.11.2]. Indeed, ALLDIFFERENT requires all variables to take distinct values. It corresponds to a cycle factor, since the variables and their domains represent the same set of elements. Therefore, to resolve the problem we state it in the terms of a directed matching.

Suppose that S separates D into p connected components denoted by $C = \{C_1, \dots, C_p\}$. The digraph D_S induced by the separator S is defined as follows. The node set of D_S is $V(D_S) = S \cup C$. The arc set $E(D_S)$ is obtained by the following rules: D_S contains an arc e if $e \in E(D)$, as well as an arc (v, w) whenever (v, c_i) and (c_j, w) are arcs of D for some pair of nodes c_i and c_j in $C_k \in C$ (possibly $c_i = c_j$, or c_i and c_j need not be joined by an arc).

We set then the degree conditions as follows: $f^-(x) = f^+(x) = 1$ for all $x \in S$, $f^-(C_k) = f^+(C_k) = \min\{d^-(C_k), d^+(C_k)\}$ for every $C_k \in C$. Obviously, we could set $g^-(x) = g^+(x) = 1$ and $f^-(x) = f^+(x) = \min\{d^-(x), d^+(x)\}$ with an additional condition that $d^-_M(x) = d^+_M(x)$ in a directed perfect (g, f)-matching, but we give a construction which guarantees that the incoming and outgoing degree of each node in the directed matching M is the same. To this end, we attach $f^+(x) - 1$ loops for every $x \in C$. This guarantees that the number of ingoing and outgoing arcs in every directed perfect matching will be identical, i.e. $d^-_F(x) = d^+_F(x)$ for every directed $\vec{f}$-factor F.

The following theorem classifies arcs of a digraph as non-Hamiltonian by looking for a certain kind of a directed degree-matching in a much smaller digraph.

[3]A graph G is called *t-tough* if the deletion of an arbitrary set S of vertices leaves the rest of the graph either connected or else broken into no more than $|S|/t$ connected components.

Theorem 8 *Suppose S is a vertex separator of a digraph D which separates D into p connected components. Let D_S be a digraph induced by S with degree conditions as described in the text. If D has a Hamiltonian circuit then D_S has a directed perfect $\vec{f}$-matching. Moreover, if an arc e connecting nodes of D_S is non-Hamiltonian then this arc is non-Hamiltonian in D. In summary, the following conditions hold:*

1. *If $|S| < p$ then D is non-Hamiltonian,*
2. *If $|S| = p$ then every arc connecting nodes of S is forbidden in D,*
3. *If $|S| > p$ then every arc forbidden in D_S is forbidden in D.*

Proof: The proof of this theorem is an easy consequence of Theorem 1, Corollary 1 and Corollary 2 of [37] adapted for directed graphs. □

According to this theorem a vertex separator induces a strongly connected digraph and a much smaller partial digraph is constructed from a vertex separator of the original graph. Then, by applying a filtering algorithm for the ALLDIFFERENT constraint, a cycle factor of a subdigraph is found, so that the forbidden arcs can be identified and eliminated from the digraph. The construction guarantees that every cycle factor of D_S will have at least one incoming and at least one outgoing arc joining S with each connected component C_k.

We now demonstrate a partial filtering method which explicitly uses the properties of the Gallai-Edmonds Canonical Decomposition and the results presented in [17, Theorem 9]. Our idea is based on identifying a perfect 2-matching in the underlying graph associated with the constraint. This follows from the fact that every Hamiltonian circuit is a 2-factor and each 2-factor can be considered as a generalization of a Hamiltonian circuit. Our routine looks for a perfect 2-matching in the underlying graph and the corresponding edge not belonging to it will be removed from the associated digraph (together with an opposite arc, if it exists). This will result in deleting some 2-cycles from the associated digraph, which cannot be a part of any Hamiltonian circuit. Note that the problem of removing all cycles of length two from the Hamiltonian digraph is NP-hard [25, Problem GT13]. However, our method coupled with the ALLDIFFERENT constraint gives a more effective pruning.

Observe that neither filtering method is redundant of the other, but both combined together improve the propagation behavior in some cases, as can be seen in the following example. Let CIRCUIT$(x_1, \ldots, x_8)$ constraint have the following domains $D_{x_1} = \{3, 4, 5\}$, $D_{x_2} = \{4, 7, 8\}$, $D_{x_3} = \{2, 7\}$, $D_{x_4} = \{1, 5\}$, $D_{x_5} = \{6\}$, $D_{x_6} = \{3, 5\}$, $D_{x_7} = \{2\}$ and $D_{x_8} = \{1, 4\}$. Then pruning based on ALLDIFFERENT removes values 4 and 7 from D_{x_2} and value 2 from the domain of variable x_3. But filtering based on our method removes first value 3 from D_{x_1} and value 2 from the domain of variable x_3 (see Figure 3). Then the next step with the ALLDIFFERENT constraint removes values 4 and 7 from D_{x_2} and value 5 from the domain of variable x_6. Hence, the filter has deleted 2-cycle $6 \to 5 \to 6$ from the associated digraph, as belonging to no Hamiltonian circuit.

The domains after two steps of the algorithm look as follows: $D_{x_1} = \{4,5\}$, $D_{x_2} = \{8\}$, $D_{x_3} = \{7\}$, $D_{x_4} = \{1,5\}$, $D_{x_5} = \{6\}$, $D_{x_6} = \{3\}$, $D_{x_7} = \{2\}$ and $D_{x_8} = \{1,4\}$.

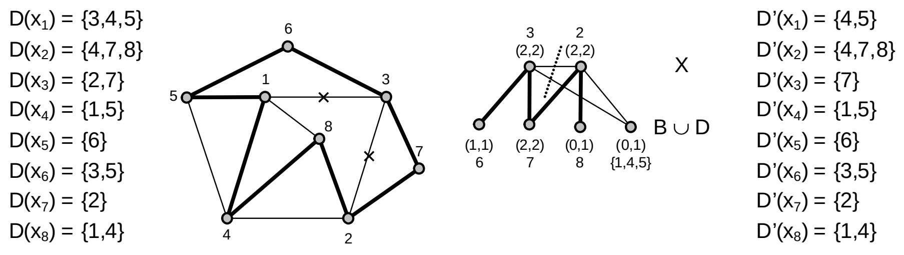

Figure 3: Pruning of the CIRCUIT constraint

A stronger propagation mechanism can also be implemented if we observe that CIRCUIT corresponds to a connection relationship. We can thus discard situations where it is impossible to build a strongly connected pattern. Pruning for imposing this condition can be done by choosing all strong bridges to belong to the solution, since otherwise the strongly connected component would be broken down. Computing the forbidden arcs can be then done in linear time, since the strong bridges can be computed in linear time [34].

We give a summary of our algorithm:

Algorithm 2 Partial filtering algorithm for the CIRCUIT constraint

```
Require: Digraph $D$ associated with CIRCUIT(NODES)
Ensure: Incomplete pruning
  Remove all loops from the associated digraph $D$
  Check the necessary condition whether $D$ is strongly connected
  If $D$ has more than one strongly connected component, then the constraint is inconsistent
  If any of the sufficient conditions holds, then the constraint is satisfiable
  Pruning associated with the ALLDIFFERENT(NODES) constraint (using perfect matching on $BR(D)$)
  If the underlying graph has no perfect 2-matching, then the constraint is unsatisfiable
  Pruning according to the Gallai-Edmonds Canonical Decomposition
  Pruning according to a vertex separator
```

All steps have a polynomial complexity. The removal of all loops takes $\mathcal{O}(n)$ time. Finding the strongly connected components of D requires $\mathcal{O}(m+n)$ time and space (see Section 3). The checking of any the sufficient conditions takes minimal $\mathcal{O}(1)$ [41] and maximal $\mathcal{O}(n^3)$ time [45]. A directed perfect matching M can be computed from scratch in time $\mathcal{O}(\sqrt{n}\cdot m)$ [33]. A perfect 2-matching of the underlying graph can then be found from M in time $\mathcal{O}(\sqrt{k}\cdot m)$, where

k is the number of exposed vertices in G. Pruning according to the Gallai-Edmonds Canonical Decomposition can be performed in $\mathcal{O}(p \cdot m)$ time [17], where p denotes the number of maximal extreme sets in the underlying graph. Pruning according to the Dulmage-Mendelsohn Canonical Decomposition can be realized in linear time [15]. The complexity of the procedure for each separator $|S|$ is approximately $\mathcal{O}(|S|^5)$ [37].

The last four steps of the algorithm must be repeated until no more arcs are pruned. Our experiments indicate that the directed matching connected with the method based on the Gallai-Edmonds decomposition of the underlying graph provides a better filtering. Additionally, approach $DM + GE$ requires a smaller number of iterations than $GE + DM$.

In our implementation, we have set the step of computing the strong components immediately after the pruning with respect to a directed matching (ALLDIFFERENT constraint), since strong connectivity in a given strongly connected digraph can be broken by removing the forbidden arcs.

4.2 The CYCLE constraint

The CYCLE is a useful constraint that was introduced in CHIP [5] in order to tackle hard combinatorial problems. The constraint specifies the number of cycles that must cover a directed graph. It can be used for modeling various problems such as the multiple traveling salesmen problem [39, pages 23-25 and 169-170], the vehicle routing problem [39, Chapter 12], [5], and the balanced Euler knight problem [11]. A thesis dealing with the CYCLE constraint is written by Bourreau [11].

The constraint has the form CYCLE(NCYCLE, NODES), where NCYCLE is a domain variable and NODES is a collection $\{x_1, \ldots, x_n\}$ of domain variables with $D_{x_i} \subseteq \{1, \ldots, n\}$ for each i.

The CYCLE constraint partitions a given associated digraph described by the NODES collection into a set of vertex-disjoint cycles. The constraint requires that in the digraph there are exactly NCYCLE directed circuits, such that every vertex belongs to exactly one cycle.

As the first interpretation, the CYCLE(NCYCLE, NODES) constraint can be seen as the problem of finding NCYCLE distinct cycles in a directed graph in such a way that each vertex is visited exactly once. In the second interpretation, this constraint can be considered as the number of NCYCLE cycles of a permutation $\langle x_1, \ldots, x_n \rangle$.

Both observations are equivalent to the formulation of the ALLDIFFERENT constraint. This can be seen from the fact that a cycle in a directed graph D is a spanning subdigraph of D in which the in-degree and out-degree of every vertex v is equal to 1:

$$d^+(v) = d^-(v) = 1.$$

Note that the global constraint CIRCUIT is a special case of the global constraint CYCLE in which the first parameter NCYCLE is fixed to 1. Thus, the elementary filtering methods for the CIRCUIT constraint presented in the last example can be simply adapted to the CYCLE constraint.

Beldiceanu, within his Global Constraint Catalog [4], proposes the following algorithm: Since all variables in NODES have to take distinct values one can reuse the algorithms associated with the ALLDIFFERENT constraint. A second necessary condition is to have no more than $\max(D_{\text{NCYCLE}})$ strongly connected components. Since all the vertices of a circuit belong to the same strongly connected component, an arc going from one strongly connected component to another strongly connected component has to be removed.

This method is redundant. We will prove that an arc going from one strongly connected component to another one will be detected during pruning according to a directed matching.

Theorem 9 *Let $D = (V, E)$ be a directed graph associated with the global constraint* CYCLE, *a bipartite graph $BR(D)$ be the bipartite representation of D, and let M be a perfect matching in $BR(D)$. Then an edge (v_i, v_j) belongs to some perfect matching in $BR(D)$ if and only if vertices v_i and v_j belong to the same strongly connected component of D.*

Proof: Since bipartite representation $BR(D)$ contains a perfect matching if and only if a digraph D has a cycle factor, thus when we take an arc e going from a strongly connected component S_i to another strongly connected component S_j, then we can never return to S_i in order to create a cycle involving the arc e. Hence, all such connecting arcs do not belong to any perfect matching. □

Moreover, a lower bound on the number of vertex-disjoint cycles in a digraph D is equal to the number of strongly connected components. Clearly, if the number of strongly connected components equals NCYCLE, then, for each connected component, we can enforce pruning according to the global constraint CIRCUIT, as discussed earlier.

However, the problem of finding an upper bound on the number of vertex-disjoint cycles in a given digraph D is NP-hard to compute. We will see that the maximum number of vertex-disjoint cycles in a digraph D is related to the minimum number of vertices in D needed to eliminate all cycles of D.

An upper bound on the number of the disjoint cycles can be obtained by solving the feedback vertex set problem. We will use some additional notation and terminology. Given a directed graph $D = (V, E)$, a *feedback vertex set* (abbreviated as FVS) is a set of vertices whose removal leaves an acyclic digraph. The problem is to find such a set with minimum cardinality. Obviously, forests and acyclic digraphs have a value of 0 since they have no cycles. In the literature, the term *cycle cutset* (or *cutset* in the short) has appeared as a synonym for feedback vertex set.

For a digraph D we denote by $\nu(D)$ the maximum number of vertex-disjoint cycles, and by $\tau(D)$ the minimum number of elements in a feedback vertex set of D. Clearly, we have $\nu(D) \leq \tau(D)$ and it is easy to construct an infinite family of digraphs such that only inequalities hold. Indeed, let $D = (V, E)$ be a digraph with vertex set $V(D) = \{x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4\}$ and arc set

$E(D) = \{(y_i, x_i) : 1 \leq i \leq 4\} \cup \{(x_i, y_j) : 1 \leq i \neq j \leq 4\}$. Then $\nu(D) = 2$, but $\tau(D) = 3$.

In summary, we have the following results:

Theorem 10 *A lower bound on the number of cycles in the digraph D associated with the global constraint* CYCLE *equals the number of strongly connected components in D.*

Theorem 11 *The problem of determining the minimum number of cycles in the digraph D associated with the global constraint* CYCLE *is NP-hard.*

Proof: The problem is clearly NP-hard as the answer is 1 if the digraph D has a Hamiltonian circuit, which is known to be an NP-complete task. □

Theorem 12 *The problem of determining the maximum number of cycles in the digraph D associated with the global constraint* CYCLE *is NP-hard.*

A natural greedy algorithm for finding the maximum number of vertex-disjoint cycles is to, repeatedly, find and remove the vertices belonging to the smallest cycle in the current digraph, until there are no more cycles left.

Karp [36] was the first to prove that the FVS problem is NP-complete (see also [25, Problem GT7]). It is thus not surprising that the above mentioned decision problem for the maximum number of vertex-disjoint cycles is also NP-complete.

Notice that the computing of the maximal number of vertices such that the corresponding induced subdigraph forms a directed acyclic graph is equivalent to enforcing satisfiability of the global constraint CUTSET [21], which holds if its corresponding digraph possesses no vertex-disjoint cycles. Because it is an NP-complete problem, recent research in this area has been concentrated on designing algorithms finding a minimum cutset for a restricted class of digraphs and a relatively small cutset for general digraphs. The algorithm described in [21] returns two vertex sets, S_1 and S_2, such that $S_1 \cap S_2 = \emptyset$, $S_1 \cup S_2$ is a cutset of a digraph D and such that $|S_1| \leq \tau(D) \leq |S_1 \cup S_2|$. Note that if $S_2 = \emptyset$ then S_1 is guaranteed to be a minimum cutset.

Theorem 13 *Enforcing hyper-arc consistency on the count variable* NCYCLE *of the global constraint* CYCLE *is NP-hard.*

Proof: It is easy to see that the problem is equivalent to the cycle cover problem. This employs a reduction from the PARTITION INTO HAMILTONIAN SUBGRAPHS problem [25, Problem GT13], originally shown NP-complete by Leslie G. Valiant [63]. □

Moreover, the count variable NCYCLE does not have the continuity property. As a counterexample, consider a directed cycle C_3 with a loop attached to every node. Then this digraph contains one or three cycles, but a cycle cover consisting of two cycles does not exist in it.

When we impose the additional condition that each cycle has only two vertices, then the CYCLE constraint can be expressed by means of the constraint SYMMETRIC_ALLDIFFERENT. This constraint can be then interpreted as covering the associated graph with disjoint circuits of length two. Clearly, there exists in this case no solution of the CYCLE constraint when the number of variables is odd or NCYCLE $\neq \{\frac{n}{2}\}$. A complete filtering algorithm achieving hyper-arc consistency for the SYMMETRIC_ALLDIFFERENT constraint was proposed by Régin in [50]. Its running time is $\mathcal{O}(n \cdot m)$. This complexity has been improved to $\mathcal{O}(p \cdot m)$ by making use of decomposition theory, where p denotes the number of maximal extreme sets in the underlying graph (for more details, see [17]).

We now give a summary of the algorithm:

Algorithm 3 Partial filtering algorithm for the CYCLE constraint

```
Require: Digraph $D$ associated with CYCLE(NCYCLE, NODES)
Ensure: Incomplete pruning
  If NCYCLE $= \{1\}$ then the constraint is equivalent to CIRCUIT(NODES)
  Pruning according to the perfect matching on $BR(D)$ (global constraint ALLDIFFERENT(NODES))
  Compute MINCYCLE as the number of strongly connected components of $D$
  Estimate MAXCYCLE (global constraint CUTSET(NCYCLE, NODES))
  Update variable NCYCLE according to MINCYCLE and MAXCYCLE values
  If there are no loops in $D$ and min(NCYCLE) $= \frac{n}{2}$ then the constraint is equivalent to SYMMETRIC_ALLDIFFERENT(NODES)
  If NCYCLE $= \{$MINCYCLE$\}$ then for each strong component $S_i$ pruning associated with the CIRCUIT($S_i$) constraint
```

All steps have a polynomial complexity. Finding the strongly connected components of D requires $\mathcal{O}(m + n)$ time and space (see Section 3). The existence of a cycle factor in a digraph can be checked and a cycle factor found, if it exists, in time $\mathcal{O}(\sqrt{n} \cdot m)$ [33]. The incomplete filtering algorithm for a CUTSET constraint has $\mathcal{O}(m + n \cdot \log n)$ time complexity [21]. Hyper-arc consistency for a SYMMETRIC_ALLDIFFERENT constraint can be achieved in polynomial time [17].

4.3 The DERANGEMENT constraint

The DERANGEMENT constraint is a special case of the CYCLE constraint. This constraint enforces the covering of an associated digraph by a set of vertex-disjoint proper cycles. In another interpretation it is required to have a permutation with no fixed points.

The pruning for achieving hyper-arc consistency is simple. It is based on the fact that x_i can take the value j if and only if the arc (x_i, j) belongs to a cycle factor. From a digraph D associated with the global constraint DERANGEMENT just remove all loops in an iterative way to obtain a reduced digraph D'. This step corresponds to the normalization of the domains of the variables. Then construct an auxiliary bipartite graph $BR(D')$ and compute a perfect matching in it. There is a one-to-one correspondence between the solution of the constraint and the existence of the perfect matching. Checking the feasibility can be realized in $\mathcal{O}(\sqrt{n}\cdot m)$ time, hyper-arc consistency can be established in $\mathcal{O}(m)$ time [15].

From an interpretation point of view this constraint is related to ALLDIFFERENT with unary constraints $x_i \neq i$ for all i, since the number of cycles is free, and the variables and their domains represent the same set of elements. Observe that the bipartite graph associated with the directed perfect matching is equivalent to the value graph associated with the ALLDIFFERENT constraint.

4.4 The SOFT_DERANGEMENT_VAR constraint

A problem is over-constrained when no assignment of values to variables is possible to satisfy the constraint. In this situation the goal is to find a compromise which allows some constraints to be violated and search for solutions that violate as few constraints as possible. The cost of the violation can be defined as the number of assigned values that should change in order to satisfy the constraint. This measure is represented by the cost variable z which is to be minimized.

In this section we apply our method to the soft version of the DERANGEMENT constraint by introducing the notion of deficiency to directed graphs.

Some preliminary terminology is needed. Recall that in the maximum matching of G the number of exposed vertices is called the deficiency of G and is denoted by $\delta(G)$. Let the deficiency of D be the number of exposed vertices on the outward side of its bipartite representation $BR(D)$. Clearly, the inward side has the same number of exposed vertices if a directed perfect matching does not exist in D.

We aim at computing a lower bound of z in order to check the consistency of the global constraint. The following result is a direct consequence of Theorem 16 from [15].

Theorem 14 *Assume that a global constraint can be represented by a directed graph D and there exists a one-to-one correspondence between the solution of the constraint and the directed perfect matching in D. Then a lower bound of the cost variable z equals the deficiency of D. Further, if $\delta(D) < \max(D_z)$ then all the values of domains of variables are consistent with the global constraint. If $\delta(D) = \max(D_z)$ then values of the domains which are represented by forbidden arcs can be removed. Otherwise, if $\delta(D) > \max(D_z)$ then the constraint is inconsistent.*

Proof: Recall that a directed perfect matching on a digraph D with n nodes and m arcs can be reduced to a perfect matching on a bipartite graph $BR(D)$ with $2n$ vertices and m edges. The existence of an instantiation of variables such that a global constraint can become satisfied by changing the value of k variables implies the existence of a maximum matching with $2k$ exposed vertices (k exposed vertices on the outward side and k exposed vertices on the inward side of the bipartite representation). By definition of the deficiency, $2k = \delta(BR(D))$, thus $k = \delta(D)$. $\square$

Using the above result, we can formulate the following filtering algorithm that enforces hyper-arc consistency on the SOFT_DERANGEMENT_VAR constraint (cf. Algorithm 5 in [15]):

Algorithm 4 Filtering algorithm for the SOFT_DERANGEMENT_VAR constraint

```
Require: Digraph $D$ associated with SOFT_DERANGEMENT_VAR($z$, NODES)
Ensure: Hyper-arc consistency or constraint not satisfied
  Remove all loops from the associated digraph $D$
  Compute a maximum matching in the bipartite graph $BR(D)$
  Compute the Dulmage-Mendelsohn Canonical Decomposition
  Determine subgraphs $G_O$, $G_U$ and $G_W$
  Determine the partition of edges
  Let $\delta$ denote the deficiency of the subgraph $G_O$
  If $\delta > \max(D_z)$ then the constraint is inconsistent
  If $\delta = \max(D_z)$ then all forbidden arcs must be removed from the digraph $D$
  If $\delta < \max(D_z)$ then all arcs in $D$ are allowed
  Update the domain of the cost variable $z$
```

The algorithm first removes iteratively all the loops from the digraph D associated with the global constraint. Next, it computes a maximum matching in the bipartite graph $BR(D)$. This takes $\mathcal{O}(\sqrt{n} \cdot m)$ time. The next two steps, computing the Dulmage-Mendelsohn decomposition and partition of edges, are of linear complexity. The remaining lines take constant time each.

The proof of Theorem 14 applies to any constraint whose graph representation resembles D and a solution corresponds to a directed perfect matching. For all such constraints that are consistent, hyper-arc consistency can be achieved in linear time, assuming that the maximum matching in $BR(D)$ is known. Note that this is equal to the complexity of achieving hyper-arc consistency on the hard version of these constraints.

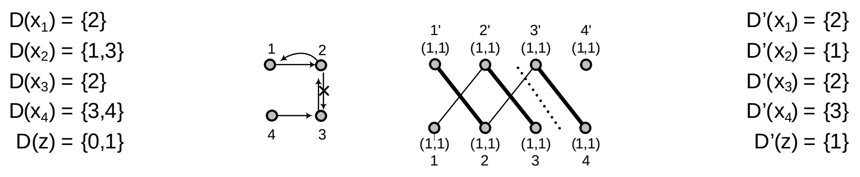

Figure 4: Pruning of the SOFT_DERANGEMENT_VAR constraint

Let us demonstrate our algorithm with an example illustrated in Figure 4. On the left side of the figure the directed graph D corresponding to the constraint is depicted; the loop (x_4, x_4) has been already removed from D in the normalization phase of the filtering algorithm.

The consistency of the constraint can be verified as follows. We first construct the bipartite graph $BR(D)$ associated with D, shown on the right side of the figure, and compute a maximum matching in $BR(D)$. The matching has deficiency 2, so $\delta(D) = 1$.

The constraint SOFT_DERANGEMENT_VAR can be made hyper-arc consistent by applying Algorithm 4. To identify the admissible arcs we use Theorem 6. Variable domains can be now filtered as follows. The cost variable z is equal to the number of variables that should change their values in order to make the constraint satisfied. If $\delta(D) > \max(D_z)$ we know that the constraint is inconsistent. We update the cost variable z to $min(D_z) = \delta(D)$ if $\min(D_z) < \delta(D)$. Since $\delta(D) = max(D_z)$ then all forbidden arcs must be removed from the digraph D. The edge $(2, 3')$ is forbidden, so the value 3 is deleted from the domain of x_2. Then our constraint is hyper-arc consistent.

The variables that need to be changed in order to obtain the solution satisfying the constraint are allowed vertices of $BR(D)$. These are variables x_1 and x_3. Since $\delta(D) = 1$, one variable needs to change its value in order to satisfy the constraint. The variable x_1 or x_3 can take the value 4. In the first case the digraph is covered with a cycle $1 \to 4 \to 3 \to 2 \to 1$, in the second case the cycle factor of D consists of two 2-cycles $1 \to 2 \to 1$ and $3 \to 4 \to 3$.

4.5 The TREE constraint

In this example we provide a quick description of the TREE constraint. The TREE constraint partitions a given directed graph into a forest of vertex-disjoint directed trees, where only certain vertices can be tree roots. More precisely, the digraph is partitioned into a set of vertex-disjoint anti-arborescences [4].

More formally, the global constraint TREE has the form TREE(NTREE, NODES), where NTREE is a domain variable specifying the number of trees in the tree partition, and NODES is a collection of n variables whose domains consist of elements of $\{1, \ldots, n\}$. The constraint holds if the associated digraph D is covered by a set of NTREE trees in such a way that each vertex of D belongs to one distinct tree. The arcs of the trees are directed from their leaves to their respective roots.

A hyper-arc consistency filtering algorithm for the global constraint TREE is described in [6]. This algorithm is based on the necessary and sufficient conditions that we now very briefly describe.

Before sketching a filtering algorithm for pruning the TREE constraint, we introduce some terminology regarding digraph $D = (V, E)$ and strong component graph $SC(D)$ associated with the TREE constraint. These definitions and notations are introduced in the original version of [6]:

- A vertex v such that $(v, v) \in E$ is called a *potential root.*
- A strongly connected component of D that contains at least one potential root is called a *rooted component.*
- A vertex v is a *door* of the strongly connected component if there exists an arc $(v, w) \in E$ such that v and w do not belong to the same strongly

[4]A digraph D is an anti-arborescence with anti-root r if and only if for each vertex v in D there is a path from v to r and the underlying undirected graph of D is a tree.

connected component of D.

- A vertex v is a *winner* if v is a door or a potential root.
- An arc $(v, w) \in E$ such that v and w do not belong to the same strongly connected component is called a *connecting arc*.
- Similarly, an arc $(v, w) \in E$ such that v and w belong to the same strongly connected component is called a *non-connecting arc*.
- *Enforcing an arc* (v, w) of D corresponds to removing from D all arcs (v, u) such that $u \neq w$.

Let D be a digraph associated with the TREE(NTREE, NODES) constraint and let $SC(D)$ be a strong component graph of D. Let MINTREE and MAXTREE respectively denote a lower and an upper bound on the number of trees for partitioning the digraph D into a set of vertex-disjoint anti-arborescences. MINTREE is equal to the number of sink components in $SC(D)$ (the number of strongly connected components in D with no outgoing arcs) and MAXTREE is equal to the number of potential roots in D. These bounds are sharp, this means that, for every MINTREE $\leq$ NTREE $\leq$ MAXTREE, we can construct the partition of edges into NTREE vertex-disjoint trees. This also shows that the variable NTREE has the continuity property. The constraint TREE has at least one solution if and only if all sink components of D contain at least one potential root and $D_{\text{NTREE}} \cap [\text{MINTREE}, \text{MAXTREE}] \neq \emptyset$ (see Proposition 3 in [6]).

In the original filtering algorithm proposed in [6] the constraint is propagated according to the *strong articulation points* of D. Recall that a strong articulation point of a strongly connected component S is such a vertex s that if we remove it then S will be broken into at least two strongly connected components. Equivalently, s is a strong articulation point of D if and only if $D - \{s\}$ has more strongly connected components than D. The strong articulation points can be found in linear time [34]. However, it was shown in [22] that the concept of strong articulation points is not practical and the authors propose a new formulation of pruning rules based on dominators.

Recall that in a flowgraph G a vertex v dominates another vertex w with respect to a designated start vertex s if every directed path in G from s to w contains v. From the above definition, it can be easily seen that every vertex dominates itself. Also, it can be seen that the initial vertex s dominates all the vertices in the digraph. Hence, if both u and v dominates w, one of u and v dominates the other.

Here are some properties of the dominance relation.

- For all x, x dominates itself (reflexivity).
- If x dominates y and y dominates x, then $x = y$ (antisymmetry).
- If x dominates y and y dominates z, then x dominates z (transitivity).
- If x dominates y, then y does not dominate x (asymmetry).

- If x and y both dominate z, then either x dominates y or conversely.
- There may exist vertices x and y such that neither x dominates y nor y dominates x.

Hence, dominance relation is a partial order (for short, a *poset*). Improving on a previous work by Tarjan [59], who discovered an $\mathcal{O}(m+n\cdot\log n)$-time algorithm for finding dominators in an arbitrary digraph, Lengauer and Tarjan [40] proposed an $\mathcal{O}(m\cdot\log n)$-time algorithm and a more complicated $\mathcal{O}(\alpha(m,n)\cdot m)$-time version, where $\alpha(m,n)$ is an extremely slow-growing functional inverse of the Ackermann function [60]. An implementation of this algorithm in linear time is presented in [1].

Theses dealing with dominators are [27] (see also [48]). An excellent general reference to filtering algorithms for tree partitioning constraints using a flow-based modeling is [43].

In terms of the graph partitioning constraints, dual concepts relating to dominators can be defined and utilized. A vertex d is a dominator of v with respect to a winner w if and only if there is no path from v to w in $D-\{d\}$. Other variants of the notion are defined analogously.

Clearly, every dominator is a strong articulation point, but not conversely (cf. Figure 6.8 and Figure 8.1 in [48]). Hence, using dominators instead of strong articulation points leads to the better filtering.

In general, if vertex u dominates vertex v then arc (v,u) does not belong to any solution. This follows from the fact that if every path from u to w requires v, then any path from v to u has to be forbidden.

Let us consider $S_i, 1 \leq i \leq p$, a strongly connected component of D, and let D_i be a set of dominators of S_i defined with respect to the winners in S_i. The removal of any dominator $d \in D_i$ creates two kinds of strongly connected components (for more details, see [9]):

- Δ_d is the (possibly empty) set of strong components from which no winner of S_i can be reached by a path that does not contain the dominator d,
- $\bar{\Delta}_d$ is the (possibly empty) set of strong components from which at least one winner of S_i can be reached by at least one path that does not contain the dominator d.

In addition, among the strongly connected components of Δ_d three types thereof, possibly empty, may be further distinguished:

- Δ_d^+ is the set of strong components corresponding to sources in $SC(\Delta_d)$,
- Δ_d^- is the set of strong components corresponding to sinks in $SC(\Delta_d)$,
- $\Delta_d^\mp$ is the set of the remaining strong components (neither sources nor sinks).

Let us consider some interesting properties of these strongly connected components.

- Let d be a dominator in strongly connected component S_i with respect to winners. Then d belongs to all paths from any vertex of Δ_d to any vertex of $\bar{\Delta}_d$.
- If there exists a path factor in D then there exists a Hamiltonian path in Δ_d leading from the source component Δ_d^+ to the sink component Δ_d^-, and finishing on the dominator d (see [9, Proposition 1]).
- Let $dom(v)$ be the set of vertices dominated by a vertex v and let d be a dominator in S_i with respect to winners. Then $d \in dom(v)$ for every $v \in \Delta_d$.

Pruning is then performed according to the following rule, which prevents the creation of (proper) cycles:

Theorem 15 *An arc (d, i) of a dominator d that reaches a vertex i of Δ_d is forbidden.*

Proof: The claim follows from the fact that enforcing such an arc would lead to some strong components with no winners, and hence creating a cycle, which is a contradiction. □

We now give a summary of the full algorithm:

Algorithm 5 Filtering algorithm for the TREE constraint

```
Require: Digraph $D$ associated with TREE(NTREE, NODES)
Ensure: Hyper-arc consistency or constraint not satisfied
  Compute MINTREE and MAXTREE
  Update variable NTREE according to MINTREE and MAXTREE values
  Check the conditions for satisfiability
  If NTREE = {MINTREE} then any potential root in a non-sink component is
  forbidden
  If NTREE = {MAXTREE} then every outgoing non-loop arc of each potential
  root is forbidden
  Pruning according to dominators of $D$ (see Theorem 15)
```

The presented filtering algorithm has a linear time complexity [22]. Computing the strongly connected components of D takes $\mathcal{O}(m + n)$ time (see Section 3). Checking that each sink component of D contains at least one potential root takes $\mathcal{O}(n)$ time. Testing whether MAXTREE $<$ min(NTREE) or max(NTREE) $<$ MINTREE takes $\mathcal{O}(1)$ time. Pruning according to dominators (Theorem 15) can be easily performed in linear time [1].

4.6 The BINARY_TREE constraint

The BINARY_TREE constraint is derived from the TREE constraint, which enforces the partitioning of an associated digraph into a set of vertex-disjoint binary trees.

The constraint has the form BINARY_TREE(NTREE, NODES), where NTREE is a domain variable specifying the number of binary trees in the tree partition, and NODES is a collection of n variables whose domains consist of elements of $\{1, \ldots, n\}$. The constraint holds if the associated digraph D is covered by a set of NTREE binary trees in such a way that each vertex of D belongs to one distinct tree.

The filtering algorithm for the BINARY_TREE constraint is not known. We show how to handle this constraint by means of the method described in this paper. Although the proposed algorithm does not achieve hyper-arc consistency (the problem to find a spanning tree in which no vertex has degree larger than some given integer is NP-complete [25, Problem ND1]), it is relatively simple to implement.

For any proper forest of binary trees the following holds:

$$\vec{g}(x) = \begin{cases} (0,1) \text{ for leaves} \\ (1,1) \text{ for internal nodes} \\ (1,0) \text{ for roots} \end{cases} \qquad \vec{f}(x) = \begin{cases} (0,1) \text{ for leaves} \\ (2,1) \text{ for internal nodes} \\ (2,0) \text{ for roots} \end{cases}$$

We model the TREE constraint by the associated digraph D in which the vertices represent the variables and the arcs represent the successor relation between them. Let R be the set of potential roots. In order to obtain the digraph D' associated with the BINARY_TREE constraint we add one dummy vertex v_0 to the input digraph D and declare that each of its predecessors is a potential root in R.

We construct a bipartite graph associated with the global constraint as described in the sequel with the following minor modifications. The vertices on both sides correspond to variables and there is an edge $\{v_i, v_j\}$ if and only if $j \in D_{x_i}$ and $i \neq j$. With every vertex we associate two functions g and f such that for each vertex v_i on the outward side we set $g(v_i) = f(v_i) = 1$ (since every vertex must have only one successor) and for the vertex v_j on the inward side we set $g(v_j) = 1$ and $f(v_j) = 2$ for roots and internal nodes (since every vertex can have at most two predecessors). Additionally, we connect all the vertices representing potential roots to a single vertex labeled *ntree* and set $g(\mathit{ntree}) = \min(D_{\text{NTREE}})$ and $f(\mathit{ntree}) = \max(D_{\text{NTREE}})$. Then, we use the filtering algorithm described in [15] to determine the partition of edges and the bounds of the NTREE variable.

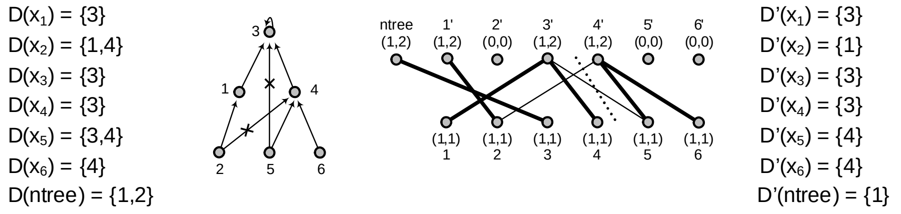

Figure 5: Pruning of the BINARY_TREE constraint

On acyclic digraphs, it is easy to see that the feasibility can be checked in polynomial time by computing the perfect (g, f)-matching in the bipartite graph associated with the constraint. Hyper-arc consistency can then be achieved in linear time by determining the partition of edges. Since there are no cycles in the digraph D, the matched edges in the bipartite graph $BR(D)$ will form the partition into the forest of vertex-disjoint binary trees (see Figure 5).

Our algorithm has the following form:

Algorithm 6 Partial filtering algorithm for the BINARY_TREE constraint

```
Require: Digraph D associated with BINARY_TREE(NTREE, NODES)
Ensure: Incomplete pruning
  Pruning associated with the TREE constraint
  Pruning according to a directed matching
```

4.7 The PATH constraint

From an interpretation point of view, the PATH constraint is the unary TREE constraint. This constraint requires the partitioning of a directed graph D into a set of vertex-disjoint (directed) paths.

The constraint has the form PATH(NPATH, NODES), where NPATH is a domain variable specifying the number of paths, and NODES is a collection of n variables. The constraint holds if the associated digraph D is covered by a set of NPATH paths in such a way that each vertex of D belongs to a single path. The arcs of the partition are directed from initial points to potential roots.

For this constraint there already exists a flow-based pruning algorithm to enforce partial filtering [9]. However, the bottleneck of the propagation algorithm relies on the detection of the arcs between two strongly connected components that do not belong to any feasible flow. This problem is polynomial, but the existence of an efficient algorithm, which is not based on a repetitive test of each arc, was stated as an open problem. In this example we will demonstrate that such approach is not well suited to practical implementations and propose a new formulation of the pruning conditions based on directed matching. The important point is that our technique does not need the concepts of maximum antichains and inter-scc arcs.

A directed Hamiltonian path problem is NP-complete [25, Problem GT39]. The problem can be solved in polynomial time for acyclic digraphs. For any proper path the following holds:

$$\vec{g}(x) = \begin{cases} (0,1) \text{ for sources} \\ (1,1) \text{ for internal nodes} \\ (1,0) \text{ for targets} \end{cases} \quad \vec{f}(x) = \begin{cases} (0,1) \text{ for sources} \\ (1,1) \text{ for internal nodes} \\ (1,0) \text{ for targets} \end{cases}$$

Let MINPATH and MAXPATH denote the minimum and the maximum number of paths in a path factor of D. Clearly, MAXPATH is the number of potential roots for paths (i.e. MAXPATH $= |\{i : i \in D_{x_i}\}|$). When the number of paths is not fixed (i.e. $|D_{\text{NPATH}}| > 1$), the key point of any approach solving the PATH constraint is the evaluation of the lower bound on the number of paths for partitioning the digraph D associated with the global constraint.

Theorem 16 *Let D be a digraph associated with the* PATH *constraint. Denote the number of source components in $SC(D)$ by s and the number of sink components by t. A lower bound on the number of vertex-disjoint paths for partitioning D is estimated by* $\max(s,t)$.

Proof: This follows from the fact that there is no path between two vertices that belong to two distinct source components of D. Similarly, there is no path between two vertices that belong to two distinct sink components of D. Thus, the rule requires that the number of source and sink components must be less than or equal to the minimum number of directed paths. □

However, the number of sink components in $SC(D)$ is not a sharp lower bound for the number of paths in D. In fact, finding a sharp lower bound makes the problem NP-complete, since we can easily reduce the Hamiltonian path problem to this problem. A sharper lower bound on the number of disjoint paths is introduced by the following result.

Theorem 17 *A lower bound on the number of vertex-disjoint paths in the digraph D associated with the* PATH *constraint is equal to* $\max\{1, \delta(D)\}$.

Proof: According to Theorem 6, we already know that the cycle factor problem in a digraph D is equivalent to the perfect matching problem in a bipartite graph $BR(D)$. We can deduce from this that if the bipartite graph contains a perfect matching then the cycle factor becomes the path factor, since all paths which form a solution are cycles. However, if in the bipartite graph corresponding to the PATH constraint the maximum matching is not perfect then certain vertices are exposed and the construction will then produce a partition of D in which some of the cycles are "broken" and become paths instead. We show that from any maximum matching M of $BR(D)$ we can build a partition of D consisting of at least $\delta(D)$ vertex-disjoint paths. For this purpose consider a digraph D and a maximum matching M in $BR(D)$. Let $k = \delta(D)$. The maximum matching M ensures that the partition of D consists of at least k connected components such that each one is a cycle or a path. This follows from the fact that for each arc $e_{ij} = (v_i, v_j)$ of D, either $e_{ij} \in M$ or $e_{ij} \notin M$ and for each vertex v_i of D there is at most one arc that belongs to a solution. Thus, the matching M in $BR(D)$ generates a subdigraph of D induced by $|M|$ arcs and composed of at least k vertex-disjoint paths. □

Theorem 18 *An upper bound on the maximum number of paths for partitioning the digraph D associated with a path constraint is the number of potential roots of D (i.e.,* MAXPATH*).*

Proof: The proof of this theorem follows in the same manner as the proof of Proposition 2 in [6]. Since each path has a distinct root, we cannot have more paths than the number of potential roots. □

Both the directed perfect matching in D and the smallest possible number MINPATH of paths can be found in $\mathcal{O}(\sqrt{n} \cdot m)$ time [33]. Observe that in the case of non-acyclic digraphs the non-sharp lower bound on the number of vertex-disjoint paths introduced by Theorem 17 can be generalized to the sharp lower bound. This follows from the fact that some matched edges in $BR(D)$ form a cycle in D, which reduces the minimum number of vertex-disjoint paths at the most by 1.

We create a directed graph D associated with the PATH constraint as follows. The vertices correspond to variables and there is an arc (v_i, v_j) if and only if $j \in D_{x_i}$ and $i \neq j$. We add a new dummy vertex v_0 representing count variable NPATH. There is an arc from v_i to v_0 if and only if $i \in D_{x_i}$. With every vertex we associate two pairs of functions $\vec{g} = (g^-, g^+)$ and $\vec{f} = (f^-, f^+)$ such that for each vertex v_i we set $g^-(v_i) = 0$, $f^-(v_i) = 1$ and $g^+(v_i) = f^+(v_i) = 1$. Additionally, we set $g^-(v_0) = \min(D_{\text{NPATH}})$, $f^-(v_0) = \max(D_{\text{NPATH}})$ and $g^+(v_0) = f^+(v_0) = 0$.

We construct a bipartite graph associated with the PATH constraint as follows. The vertices on both sides correspond to variables and there is an edge $\{v_i, v_j\}$ if and only if $j \in D_{x_i}$ and $i \neq j$. With every vertex we associate two functions g and f such that for each vertex v_i on the outward side we set $g(v_i) = f(v_i) = 1$ and for the vertex v_j on the inward side we set $g(v_j) = 0$ and $f(v_j) = 1$. Additionally, we connect all the vertices representing potential roots to a single vertex labeled *npath* and set $g(npath) = \min(D_{\text{NPATH}})$ and $f(npath) = \max(D_{\text{NPATH}})$. Then, we use the filtering algorithm described in [15] to determine the partition of edges and the bounds of the NPATH variable (see Figure 6).

Since a path factor in an acyclic digraph has no cycles, this implies that the path factor for acyclic digraphs is easy to find.

$D(x_1) = \{1,2\}$	$D'(x_1) = \{1\}$
$D(x_2) = \{1,2\}$	$D'(x_2) = \{2\}$
$D(x_3) = \{1\}$	$D'(x_3) = \{1\}$
$D(x_4) = \{1,2\}$	$D'(x_4) = \{2\}$
$D(x_5) = \{3,6\}$	$D'(x_5) = \{3,6\}$
$D(x_6) = \{3,5\}$	$D'(x_6) = \{3,5\}$
$D(npath) = \{2\}$	$D'(npath) = \{2\}$

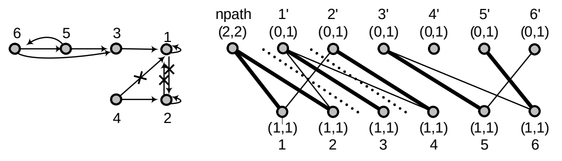

Figure 6: Pruning of the PATH constraint

Observe that the PATH constraint is very similar to the ALLDIFFERENT constraint, except that the potential roots have to be handled differently. In order to avoid cycles we have the additional restriction that each vertex on the path is not visited more than once, initial endpoints are excluded from the set, every element is distinct and must appear once yet the numbers representing potential roots appear exactly twice. On the bipartite representation $BR(D)$ of D

we use the filtering algorithm described in [15] to prune every arc of D that is incompatible with the PATH constraint.

Filtering with respect to the path factor can remove significantly more arcs than for the TREE partition. In this case there are some additional important propagation rules and according to dominators of D the pruning is performed in the following way (see [9]):

Theorem 19 *Let d be a dominator in D. Then the following arcs are forbidden in D:*

1. *An arc (d, i) going from the dominator d to Δ_d,*
2. *An arc (j, i) going from $\bar{\Delta}_d$ to Δ_d such that the strong component containing i is not a source,*
3. *An arc (i, d) going from Δ_d such that the strong component containing i is not a sink,*
4. *An arc (j, d) going from $\bar{\Delta}_d$ such that the strong component Δ_d is not empty.*

Proof: These rules have not been proven in [9]. Thus, we formally do it.

1. Proof analogous to that of Theorem 15.
2. The claim follows from the fact that there would be no way to visit some of the vertices of Δ_d if the strong component containing i were not to be a source.
3. The claim follows from the fact that there would be no way to visit some of the vertices of Δ_d if the strong component containing j were not to be a sink.
4. The claim follows from the fact that there would be no way to visit all the vertices of Δ_d if the arc (j, d) were to be enforced.

□

These propagation rules prevent the creation of (proper) cycles and enforce one single predecessor for each vertex of the strongly connected component. We demonstrate the theorem with Figure 7:

Theorem 20 ([9, Proposition 2]) *A lower bound on the number of vertex-disjoint paths partitioning the strongly connected component S_i with respect to a dominator d is provided by the minimum number of paths partitioning $\bar{\Delta}_d$ (the number of rooted components in $\bar{\Delta}_d$) minus 1 if there exists an arc $(u, v) \in S_i$ such that $u \in \bar{\Delta}_d$ and $v \in \Delta_d^+$.*

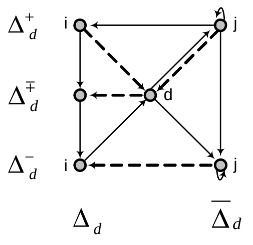

Figure 7: Pruning according to a dominator d

The authors of [9] do not include the filtering for arcs between two strongly connected components because they do not know how to do this efficiently by computing one single feasible flow. However, by means of our technique it is possible to make it during the pruning according to a directed matching. For example, our algorithm will detect the following mandatory arcs: $(0,1)$, $(1,2)$, $(3,3)$, $(7,7)$, $(8,8)$, $(9,9)$, $(10,10)$, $(11,13)$, $(13,12)$ and the following forbidden arcs: $(2,0)$, $(3,2)$, $(3,7)$, $(3,8)$, $(4,2)$, $(5,1)$, $(6,1)$, $(6,7)$, $(6,8)$, $(7,12)$, $(8,12)$, $(9,13)$, $(10,13)$, $(12,7)$, $(12,8)$, $(12,11)$ (see Example 5 and Figure 6 in [9]).

This will be realized in the following way. First, according to dominator 2 of C_0, the arc $(2,0)$, as leading to Δ_0, is detected by Case 1, the arcs $(5,1)$ and $(6,1)$ are detected by Case 2, the arcs $(3,2)$ and $(4,2)$ are detected by Case 4. Analogously, according to dominator 13 of C_1, the arc $(12,11)$, as leading to Δ_1, is detected by Case 1, the arcs $(9,13)$ and $(10,13)$ are detected by Case 2, the arcs $(7,12)$ and $(8,12)$ are detected by Case 4. Next, according to the directed matching, the arcs $(3,7)$, $(3,8)$, $(6,7)$, $(6,8)$, $(12,7)$, $(12,8)$ are detected. Additionally, the minimum/maximum number of vertex-disjoint paths is 5 and 7, respectively. Internal nodes are 1 and 13.

Our algorithm has the following form:

Algorithm 7 Partial filtering algorithm for the PATH constraint

```
Require: Digraph $D$ associated with PATH(NPATH, NODES)
Ensure: Incomplete pruning
  Compute MINPATH and MAXPATH
  Adjust variable NPATH according to MINPATH and MAXPATH values
  If NPATH = {MINPATH} then any potential root in a non-sink component is
  forbidden
  If NPATH = {MAXPATH} then all outgoing non-loop arcs for each potential
  root are forbidden
  If $D$ is not acyclic then pruning according to dominators of $D$ (see Theo-
  rem 19)
  Pruning according to a directed matching
```

We can improve our algorithm by the following result.

Theorem 21 *If* $\min(D_{\text{NPATH}}) = \text{MAXPATH}$ *then, for each potential root* r*, all* *the outgoing non-loop arcs* (r, v) *are forbidden and will be detected by pruning with respect to a directed matching.*

Proof: Observe that if NPATH is equal to MAXPATH then all the loops will be enforced and all the outgoing non-loop arcs will be removed during the pruning based on a directed matching. We present a very simple proof of this fact. First of all, if the condition holds then every edge adjacent with the vertex NPATH in $BR(D)$ is mandatory. Thus, every other edge leading from some neighbor of NPATH to an inward vertex, distinct from NPATH, is forbidden. This follows from the fact that $g(x) = f(x) = 1$ holds for all vertices on the outward side of the bipartite representation of D. □

The similar result holds for the BINARY_TREE constraint and the count variable NTREE.

Theorem 22 *The count variable* NPATH *has the continuity property.*

Proof: Consider a digraph D and a set of potential roots R. Assume that we have found a path factor of size $r < |R|$. We can build a path factor of size $r+1$ by decomposing one of its paths with respect to its potential roots. These steps may be continued until a path factor of size $|R|$ is achieved. □

4.8 The MAP constraint

In this example we provide a quick description of the MAP constraint. Next, we show how the propagation rules used for the constraints CIRCUIT, CYCLE and TREE can be implemented to generate a partial filtering algorithm.

The MAP constraint is a useful global constraint that can be used for covering a graph by a set of disjoint cycles and trees, and for modeling various problems such as random mappings [24], or graph related problems for the vertex-disjoint partitioning of graphs. However, before we define it more formally we need the description of the map. For a map, we take the definition from [54, page 459]:

> Every map decomposes into a set of connected components, also called *connected maps*. Each component consists of the set of all points that wind up on the same cycle, with each point on the cycle attached to a tree of all points that enter the cycle at that point.

The global constraint MAP has the form MAP(NBCYCLE, NBTREE, NODES), where NBCYCLE and NBTREE are domain variables, and NODES is a collection of vertices, which domain designates the successor vertex that will be used in the covering. The variables NBCYCLE and NBTREE are respectively equal to the number of cycles and the number of trees in the partition that can be interpreted as a map.

The MAP constraint was introduced within the Global Constraint Catalog [4] but no filtering algorithm is known. The purpose of this example is to present an incomplete filtering algorithm for the MAP constraint. The filter removes inconsistent values by eliminating arcs from the associated digraph, which do not belong to a map. We prove the necessary condition for an arc to be part of the MAP constraint, which provides the basis for eliminating arcs.

Before we more formally describe a filtering algorithm for the MAP constraint, we first need to introduce some terminology that will be used throughout this example. We will, as far as possible, use the notation introduced in [6], which we now extend:

- A strongly connected component that has a cycle factor is called a *cycle component*.
- A strongly connected component that has no cycle factor is called a *tree component*.

Observe that every tree component contains at least one tree. Thus, the number of trees in the MAP constraint is related to the number of sink components that have no cycle factor. Since a path is a degenerated tree the maximum number of paths in a MAP constraint is related to the number of trees.

We need to slightly modify the definition of the strong component graph associated with the MAP constraint. The strong component graph $SC(D)$ is derived from D with the following modification: to each strongly connected component of D that is a cycle component, we associate a vertex with a loop. The vertices without loops represent tree components.

It can be easily shown that if D contains a tree component then it must necessarily contain a tree. Thus, presence of trivial vertices in a digraph associated with the MAP constraint implies the presence of at least one tree.

We now introduce a theorem that will allow us to reduce the problem of finding the partition of a directed graph to the problem of estimating the bounds on the minimal and the maximal number of cycles and trees.

Theorem 23 *Let $D = (V, E)$ be an arbitrary finite (not necessarily connected) digraph such that every vertex has at least one successor. Then there exists a partition of D consisting of cycles, possibly loops, with trees having roots on their vertices.*

Proof: One can construct the partition of D from its arcs by first selecting an arbitrary arc among them and then successively adding a new arc in such a way that it has at least one endpoint in common with the arcs already selected. Since the domains are finite, each such sequence must eventually loop back on itself. □

Hence, according to this theorem, no pruning is required for the MAP constraint when there are no given bounds on the number of cycles and the number of trees (e.g. $D_{\text{NBCYCLE}} = D_{\text{NBTREE}} = \{0, \ldots, n\}$). Further, every map has at least one cycle (including loops). When the above operation is repeated, starting each time from an element not previously hit, the vertices group themselves into components. This leads to a valuable characterization of such a partition: a map is a set of connected components that are cycles of trees. Thus, every connected component is a collection of rooted trees arranged in a cycle.

Theorem 24 *The minimum number of cycles in the digraph D associated with the global constraint* MAP *equals the number of sink components in $SC(D)$.*

Proof: The claim follows from the fact that every sink component is strongly connected; this means it contains at least one cycle, and there is no path between two vertices that belong to two distinct sink components of D. □

Observe that the non-sharp lower bound on the number of cycles in the CYCLE constraint introduced by Theorem 10 is now generalized to the sharp lower bound on the number of cycles in the MAP constraint. But the results of Theorem 12 remain still valid.

Theorem 25 *The problem of determining the maximum number of cycles in the digraph D associated with the global constraint* MAP *is NP-hard.*

Proof: The problem of finding the maximum number of vertex-disjoint cycles is related to the problem of determining a feedback vertex set of minimum cardinality. The claim follows now from the fact that the minimum cutset problem is an NP-hard task. □

Theorem 26 *Given a digraph D associated with the* MAP *constraint, an upper bound on the number of cycles partitioning D is given by the minimum feedback vertex set of D.*

Proof: The claim follows from the fact that the size of the minimum feedback vertex set in a digraph D is no less than the maximum number of vertex-disjoint cycles in D. □

It is easy to see that a map where all the variables have distinct values leads to a set of cycles. Therefore, if digraph D associated with the MAP constraint has a cycle factor then we can immediately set MINTREE $= 0$. On the other hand, if digraph D has only cycles of length 1 (i.e. loops), we have a map that corresponds to a forest of trees and paths, and the algorithm for the TREE or PATH constraint can be used.

Theorem 27 *A lower bound on the number of trees in the digraph D associated with the global constraint* MAP *is equal to the minimal number of vertex-disjoint proper trees rooted on the sink components of $SC(D)$ and covering all tree components of $SC(D)$.*

Proof: According to the definition of the MAP constraint the number of trees equals the number of arcs that do not belong to any cycle yet their tails are located on a cycle. Therefore, the minimal number of trees is equal to the minimal number of trees in D rooted at the sink components of $SC(D)$. Thus, the claim is trivially derived from the definition of the TREE constraint. □

Recall that in a topological ordering of a given directed acyclic graph D, each vertex v is associated with a value $ord(v)$, such that for each arc (u, v) we have $ord(u) < ord(v)$ and for each arc (v, w) we have $ord(v) < ord(w)$. The topological ordering can be found in linear time (see Algorithm 1).

In order to compute a lower bound on the number of trees for the MAP constraint we will impose a constraint INCREASING_NVALUE(NVAL,VARIABLES) with the following variables and domains. Let VARIABLES be a set of tree components of $SC(D)$. For every tree component s_i we put into the domain D_{s_i} the topological number $ord(s_j)$ of the sink component s_j in $SC(D)$ if there is a directed path from s_i to the sink component s_j. For variable NVAL we set $D_{\text{NVAL}} = \{1, \ldots, |\text{VARIABLES}|\}$. Since enforcing hyper-arc consistency for the INCREASING_NVALUE constraint is $\mathcal{O}(m)$ [7] the computation of a lower bound on the number of trees is of linear complexity.

According to the property of DAGs we know that from every vertex v there exists a directed path in $SC(D)$ to at least one of its sinks. Thus, we make sure that we will explore a tree and all the trivial vertices of $SC(D)$ will be visited.

We know that for the MAP constraint the number of trees is the number of vertices directly connected to a cycle. According to this definition a trivial upper bound could be computed as follows: count the number of vertices for which at least one successor (that is not the vertex itself) is a part of a potential cycle. This can be made faster by using the following idea: every vertex that belongs to a strongly connected component containing more than one vertex or having a loop is a vertex that can be on a cycle. Therefore, we should find a vertex that has at least one successor different from those on the cycle.

It turns out that a sharper bound on the maximal number of trees can be obtained by solving the NVALUE constraint. More formally, an upper bound on the number of trees in the MAP(NBCYCLE, NBTREE, NODES) constraint equals n minus the minimum number of distinct values in the NVALUE(NVAL, NODES) constraint, where n is the number of variables and $D_{\text{NVAL}} = \{0, \ldots, n\}$. Since the computing of the minimum number of distinct values for the NVALUE constraint is an NP-hard task [10] then the computing of the maximum number of trees for the MAP constraint is NP-hard, as well.

Theorem 28 *An upper bound on the number of trees for the global constraint* MAP *equals n minus the minimum number of distinct values in the* NVALUE(NVAL, NODES) *constraint, where $D_{\text{NVAL}} = \{0, \ldots, n\}$. More formally,* MINTREE $= n - \min(D'_{\text{NVAL}})$.

Proof: Let D be a directed graph associated with the MAP constraint. We will use the induction on the number of trees in D. Suppose that we start from MINTREE $= 0$. In this case digraph D has, obviously, a cycle factor. This is the same as the NVALUE constraint with $D_{\text{NVAL}} = \{n\}$, since each value in NVAL must appear once, which leads to a set of cycles. Similarly, the case that MINTREE $= 1$ is the same as the NVALUE constraint with $D_{\text{NVAL}} = \{n-1\}$. This follows from the fact that when $n-1$ variables are distinct, two variables must take the same value. Since one value belongs to some cycle the second value is a terminal endpoint of a directed path leading to this cycle. Other cases can be handled similarly. Continuing this process, by the pigeonhole principle[5], we find that a map having exactly k distinct values has at most $n - k$ trees. A bound is non-sharp since a vertex must not necessarily belong to a cycle. The same argument holds for each of the $1, \ldots, n$ distinct values in the map, so the expected number of trees is obtained by the formula given in the theorem. Thus, we have proven the result. □

Theorem 29 *The problem of determining the maximum number of trees in the digraph D associated with the global constraint* MAP *is NP-hard.*

Proof: The claim follows from the fact that computing the lower bound on NVAL variable of the NVALUE constraint is NP-hard. Such a constraint is called ATMOST_NVALUE [10]. □

Since the MAP constraint is satisfiable when the values NBCYCLE and NBTREE are of allowed range the incomplete pruning algorithm consists of detection forbidden arcs when NBCYCLE and NBTREE are instantiated to one of their extrema.

We demonstrate our algorithm with the following example (the sample digraph is taken from [13, page 23]).

$D(x_1) = \{2,5,6\}$
$D(x_2) = \{1\}$
$D(x_3) = \{2,4,5\}$
$D(x_4) = \{9\}$
$D(x_5) = \{1,7\}$
$D(x_6) = \{5,8,10\}$
$D(x_7) = \{4\}$
$D(x_8) = \{7,10\}$
$D(x_9) = \{7\}$
$D(x_{10}) = \{8\}$
$D(x_{11}) = \{10,12\}$
$D(x_{12}) = \{8,11,13\}$
$D(x_{13}) = \{11\}$
$D(nbcycle) = \{1,2\}$
$D(nbtree) = \{1,2\}$

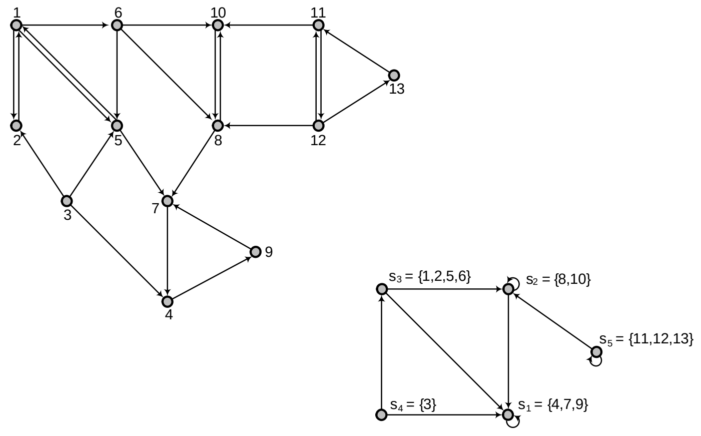

Figure 8: Checking feasibility of the MAP constraint

Figure 8 illustrates the different terms related to the MAP constraint. The MAP constraint is stated with the variable domains given on the left side of the figure. In the middle of the figure the digraph D associated with the MAP constraint is depicted. On the right side of the figure the strong component graph $SC(D)$ is shown. To each strongly connected component S_i of D corresponds

[5]The *pigeonhole principle*, called also *Dirichlet drawer principle*, states that if $n+1$ objects (pigeons) are placed into n boxes (pigeonholes), then some box contains more than one object.

a vertex s_i of $SC(D)$. Vertices s_1, s_2 and s_5 with loops represent cycle components, vertices s_3 and s_4 represent tree components. Further, vertices s_4 and s_5 represent source components and vertex s_1 represents a sink component.

The minimum number of cycles is 1 since there exists one sink component S_1. An upper bound on the number of cycles is 4 since $\tau(D) = 4$. A cutset is, for example, $\{1, 4, 10, 11\}$. Note that in our case $\nu(D) = \tau(D)$. The minimum number of trees is 1 since there exists one path leading to the sink component and covering all tree components of $SC(D)$. An upper bound on the number of trees is $5 (= 13 - 8)$, since $\min(D'_{\text{NVAL}}) = 8$ for the constraint ATMOST_NVALUE. The constraint MAP holds since the values of count variables NBCYCLE and NBTREE lie within the computed extrema. The reader is invited to check some possible solutions with an arbitrary number of cycles and trees from range $1 \ldots 2$.

We conclude this subsection with a summary of the incomplete filtering algorithm:

Algorithm 8 Partial filtering algorithm for the MAP constraint

```
Require: Digraph $D$ associated with MAP(NBCYCLE, NBTREE, NODES)
Ensure: Incomplete pruning
  If NBCYCLE $= \{1\}$ and NBTREE $= \{0\}$ then the constraint specializes into CIRCUIT(NODES)
  If $\max$(NBCYCLE) $> 1$ and NBTREE $= \{0\}$ then the constraint specializes into CYCLE(NBCYCLE, NODES)
  Compute MINCYCLE and MINTREE
  Estimate MAXCYCLE and MAXTREE
  If all sink components of $D$ are loops and their number is equal to NBCYCLE, and the minimum number of trees is equal to $\min$(NBTREE), and the number of vertices in $D$ which have a successor which is located on a sink component of $D$ equals $\max$(NBTREE), then the constraint is equivalent to TREE(NBCYCLE, NODES)
  If NBCYCLE $= \{$MINCYCLE$\}$ then the pruning is equivalent to the pruning for TREE(NBTREE, TC(D)), where TC(D) denotes a tree component of $D$, which is created in the following way: every strong component representing sink of $SC(D)$ is contracted to a single vertex with a loop
  If NBTREE $= \{$MINTREE$\}$ then the pruning is equivalent to the pruning for TREE(NBTREE, TT(D)), where TT(D) denotes a tree component of $D$, which is created in the following way: the tail of every arc going to a strong component representing sink of $SC(D)$ is replaced by a loop
```

Evaluating the complexity of the algorithm is done by analyzing the following steps. In order to compute the MINCYCLE value we need to find the strongly connected components of D. This takes $\mathcal{O}(m + n)$ time with the algorithm presented in Section 3. In order to estimate the MAXCYCLE value we use the algorithm for the CUTSET constraint that is of $\mathcal{O}(m + n \cdot \log n)$ time complexity [21]. The existence of a cycle factor can be found in time $\mathcal{O}(\sqrt{n} \cdot m)$ [33]. The construction of the strong component graph $SC(D)$ can be easily done in linear time by performing an alternating depth-first search on $BR^*(D)$. At the same time, the topological ordering of the vertices of $SC(D)$ may be obtained (see Algorithm 1). The minimum number of trees can be computed in linear time [7]. Estimating the maximum number of trees takes $\mathcal{O}(n^2)$ time when we use one of the approximation algorithms described in [10]. Adjusting the variables NBCYCLE and NBTREE can be carried out in constant time.

We analyzed the use of the global constraint MAP for modeling various problems and for partitioning graphs. We have shown that the constraint can be solved by using the other global constraints such as CUTSET, NVALUE, CIRCUIT, CYCLE, TREE or even SYMMETRIC_ALLDIFFERENT.

5 Conclusion

In this paper we have introduced a useful filtering technique based on a directed matching. A general method was described for solving partitioning problems on directed graphs. Such partitioning problems include finding cycles, paths, trees or various maps. In this paper we provided a unified setting for such problems.

Many important graph partitioning constraints are not tractable. For such problems there is in general no algorithm to compute a solutions and no filtering algorithm to remove all redundant values. In this case we must be satisfied with less pruning than the tractable case provides. We have presented efficient methods for incomplete pruning of the variable domains. Our result can be applied to other constraints with a similar graph representation and structure.

We have studied the application of matching theory to the constraints representable by bipartite [15], general [17], weighted [16], and directed graphs (this paper). We have investigated several graph partitioning constraints, some serious and others more entertaining, that can be modeled by directed graphs and filtered by means of structures such as directed matchings, dominators or strongly connected components.

Another natural extension of the matching problem arises when considering weighted directed matchings in digraphs. Our approach can be easily applied to the weighted case. The only difference is that the costs are assigned to the bipartite graph where a directed matching is constructed. However, an open problem would be to describe additional classes of problems that can be modeled by weighted digraphs.

In this paper we have given a partial filtering for the CIRCUIT constraint. Another interesting problem is to find a Hamiltonian cycle in a planar graph. In this case we can use for pruning a necessary condition discovered by the Latvian mathematician Emanuel Grinberg [29] in 1968 (see also [56], [65]). The Hamiltonian cycle problem is NP-complete even for planar graphs [26].

In many practical partitioning problems, we have to cover all the vertices of the associated digraph with connected components consisting of at least two vertices. We can extend all the mentioned constraints with the property of a proper partition.

One of the interesting questions for future work would be to apply our approach to balanced constraints such as BALANCE_CYCLE, BALANCE_PATH and BALANCE_TREE, introduced in [4]. These constraints are characterized by the additional parameter measuring the difference between the number of vertices in the smallest pattern and the number of vertices in the largest pattern.

We have proposed an algorithm for the detection of strongly connected components in a directed graph based on a bipartite matching. An open problem is whether a pruning according to dominators and strong bridges can be realized with the help of matching theory.

Finally, note that all algorithms we developed here are practicable and easy to implement. Thus, we expect this work to be relevant for many applications and practical approaches in the field of constraint programming. We consider an empirical evaluation of our algorithms to be an interesting question that deserves further study.

Acknowledgements

The author wish to thank anonymous referees for their valuable comments and suggestions that have considerably improved the presentation of this paper.

References

[1] S. Alstrup, D. Harel, P. W. Lauridsen, and M. Thorup. Dominators in linear time. *SIAM Journal on Computing*, 28(6):2117–2132, 1999. `doi:10.1137/S0097539797317263`.

[2] K. R. Apt. *Principles of Constraint Programming*. Cambridge University Press, New York, 2003.

[3] J. Bang-Jensen, G. Gutin, and H. Li. Sufficient conditions for a digraph to be Hamiltonian. *Journal of Graph Theory*, 22(2):181–187, 1996. `doi:10.1002/(SICI)1097-0118(199606)22:2<181::AID-JGT9>3.0.CO;2-J`.

[4] N. Beldiceanu, M. Carlsson, and J.-X. Rampon. Global Constraint Catalog. Technical Report T2012-03, Swedish Institute of Computer Science, 2012.

[5] N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Mathematical and Computer Modelling*, 20(12):97–123, 1994. `doi:10.1016/0895-7177(94)90127-9`.

[6] N. Beldiceanu, P. Flener, and X. Lorca. The tree constraint. In *CPAIOR 2005*, volume 3524 of *LNCS*, pages 64–78, 2005. `doi:10.1007/11493853_7`.

[7] N. Beldiceanu, F. Hermenier, X. Lorca, and T. Petit. The increasing nvalue constraint. In *CPAIOR 2010*, volume 6140 of *LNCS*, pages 25–39, 2010. `doi:10.1007/978-3-642-13520-0_5`.

[8] N. Beldiceanu, I. Katriel, and X. Lorca. Undirected forest constraints. In *CPAIOR 2006*, volume 3990 of *LNCS*, pages 29–43, 2006. `doi:10.1007/11757375_5`.

[9] N. Beldiceanu and X. Lorca. Necessary condition for path partitioning constraints. In *CPAIOR 2007*, volume 4510 of *LNCS*, pages 141–154, 2007. `doi:10.1007/978-3-540-72397-4_11`.

[10] C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. Filtering algorithms for the nvalue constraint. *Constraints*, 11(4):271–293, 2006. `doi:10.1007/s10601-006-9001-9`.

[11] E. Bourreau. *Traitement de contraintes sur les graphes en programmation par contraintes [Processing constraints on graphs in the framework of Constraint Programming]*. PhD thesis, University Paris, France, 1999. In French.

[12] Y. Caseau and F. Laburthe. Solving small TSPs with constraints. In *Proceedings of the 14th International Conference on Logic Programming (ICLP)*, pages 316–330, 1997.

[13] N. Christofides. *Graph Theory: An Algorithmic Approach*. Academic Press, London, 1975.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, USA, second edition, 2001. Previous edition 1990.

[15] R. Cymer. Dulmage-Mendelsohn canonical decomposition as a generic pruning technique. *Constraints*, 17(3):234–272, 2012. doi:10.1007/s10601-012-9120-4.

[16] R. Cymer. Weighted matching as a generic pruning technique applied to optimization constraints. *Annals of Operations Research*, 217(1):165–211, 2014. doi:10.1007/s10479-014-1582-x.

[17] R. Cymer. Gallai-Edmonds decomposition as a pruning technique. *Central European Journal of Operations Research*, 23(1):149–185, 2015. doi:10.1007/s10100-013-0309-4.

[18] R. Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, San Francisco, California, 2003.

[19] G. Dooms. *The CP(Graph) Computation Domain in Constraint Programming*. PhD thesis, Faculté des Sciences Appliquées, Université Catholique de Louvain, Louvain-La-Neuve, Belgium, 2006.

[20] J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.

[21] F. Fages and A. Lal. A constraint programming approach to cutset problems. *Computers & Operations Research*, 33(10):2852–2865, 2006. doi:10.1016/j.cor.2005.01.014.

[22] J.-G. Fages and X. Lorca. Revisiting the tree constraint. In *CP 2011*, volume 6876 of *LNCS*, pages 271–285, 2011. doi:10.1007/978-3-642-23786-7_22.

[23] T. Fahle. Cost based filtering vs. upper bounds for maximum clique. In *4th International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, pages 93–107, 2002.

[24] P. Flajolet and A. M. Odlyzko. Random mapping statistics. In *Advances in Cryptology - EUROCRYPT '89*, volume 434 of *LNCS*, pages 329–354, 1990. doi:10.1007/3-540-46885-4_34.

[25] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, California, 1979.

[26] M. R. Garey, D. S. Johnson, and R. E. Tarjan. The planar Hamiltonian circuit problem is NP-complete. *SIAM Journal on Computing*, 5(4):704–714, 1976. doi:10.1137/0205049.

[27] L. Georgiadis. *Linear-Time Algorithms for Dominators and Related Problems.* PhD thesis, Princeton University, Princeton, USA, 2005.

[28] M. Gondran and M. Minoux. *Graphs and Algorithms.* John Wiley & Sons, New York, 1984.

[29] E. J. Grinberg. Plane regular graphs of degree three without Hamiltonian circuits. *Latvian Mathematical Yearbook*, 4:51–58, 1968. In Russian. Translated by Dainis Zeps.

[30] P. Hall. On representatives of subsets. *The Journal of the London Mathematical Society*, 10(1):26–30, 1935. `doi:10.1112/jlms/s1-10.37.26`.

[31] M. Henz, T. Müller, and S. Thiel. Global constraints for round robin tournament scheduling. *European Journal of Operations Research*, 153(1):92–101, 2004. `doi:10.1016/S0377-2217(03)00101-2`.

[32] J. N. Hooker. *Integrated Methods for Optimization.* Springer, 2007.

[33] J. E. Hopcroft and R. M. Karp. An $\mathcal{O}(n^{5/2})$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. `doi:10.1137/0202019`.

[34] G. F. Italiano, L. Laura, and F. Santaroni. Finding strong bridges and strong articulation points in linear time. *Theoretical Computer Science*, 447:74–84, 2012. `doi:10.1016/j.tcs.2011.11.011`.

[35] D. M. Johnson, A. L. Dulmage, and N. S. Mendelsohn. Connectivity and reducibility of graphs. *Canadian Journal of Mathematics*, 14:529–539, 1962. `doi:10.4153/CJM-1962-044-0`.

[36] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. 1972. `doi:10.1007/978-1-4684-2001-2_9`.

[37] L. G. Kaya and J. N. Hooker. A filter for the circuit constraint. In *CP 2006*, volume 4204 of *LNCS*, pages 706–710, 2006. `doi:10.1007/11889205_55`.

[38] J.-L. Laurière. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence. An International Journal*, 10(1):29–127, 1978.

[39] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* John Wiley & Sons, New York, 1985.

[40] T. Lengauer and R. E. Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1(1):121–141, 1979. `doi:10.1145/357062.357071`.

[41] M. Lewin. On maximal circuits in directed graphs. *Journal of Combinatorial Theory, Series B*, 18(2):175–179, 1975. doi:10.1016/0095-8956(75)90045-3.

[42] X. Lorca. *Contraintes de Partitionnement de Graphe [Graph Partitioning Constraints]*. PhD thesis, Université de Nantes, Faculté des Sciences et des Techniques, Nantes, France, 2007. In French.

[43] X. Lorca. *Tree-based Graph Partitioning Constraint*. ISTE. John Wiley & Sons, 2013.

[44] L. Lovász and M. D. Plummer. *Matching Theory*. Annals of Discrete Mathematics (29). North-Holland, Amsterdam, 1986.

[45] Y. Manoussakis. Directed Hamiltonian graphs. *Journal of Graph Theory*, 16(1):51–59, 1992. doi:10.1002/jgt.3190160106.

[46] S. Micali and V. V. Vazirani. An $\mathcal{O}(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 17–27, Syracuse, New York, 1980.

[47] E. Poder, N. Beldiceanu, and E. Sanlaville. Computing a lower approximation of the compulsory part of a task with varying duration and varying resource consumption. *European Journal of Operational Research*, 153(1):239–254, 2004. doi:10.1016/S0377-2217(02)00756-7.

[48] L. Quesada. *Solving Constrained Graph Problems using Reachability Constraints based on Transitive Closure and Dominators*. PhD thesis, Université Catholique de Louvain, Belgium, 2006.

[49] J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI)*, pages 362–367, 1994.

[50] J.-C. Régin. The symmetric alldiff constraint. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 420–425, 1999.

[51] J.-C. Régin. Using constraint programming to solve the maximum clique problem. In *CP 2003*, volume 2833 of *LNCS*, pages 634–648, 2003. doi:10.1007/978-3-540-45193-8_43.

[52] J.-C. Régin. Simpler and incremental consistency checking and arc consistency filtering algorithms for the weighted spanning tree constraint. In *CPAIOR 2008*, volume 5015 of *LNCS*, pages 233–247, 2008. doi:10.1007/978-3-540-68155-7_19.

[53] J.-C. Régin, L.-M. Rousseau, M. Rueher, and W.-J. van Hoeve. The weighted spanning tree constraint revisited. In *CPAIOR 2010*, volume 6140 of *LNCS*, pages 287–291, 2010. doi:10.1007/978-3-642-13520-0_31.

[54] R. Sedgewick and P. Flajolet. *An Introduction to the Analysis of Algorithms.* Addison-Wesley, 1996.

[55] M. Sellmann, T. Gellermann, and R. Wright. Cost-based filtering for shorter path constraints. *Constraints*, 12(2):207–238, 2007. `doi:10.1007/s10601-006-9006-4`.

[56] Y. Shimamoto. On an extension of the Grinberg theorem. *Journal of Combinatorial Theory, Series B*, 24(2):169–180, 1978. `doi:10.1016/0095-8956(78)90018-7`.

[57] J. A. Shufelt and H. J. Berliner. Generating Hamiltonian circuits without backtracking from errors. *Theoretical Computer Science*, 132:347–375, 1994. `doi:10.1016/0304-3975(94)90239-9`.

[58] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. `doi:10.1137/0201010`.

[59] R. E. Tarjan. Finding dominators in directed graphs. *SIAM Journal on Computing*, 3(1):62–89, 1974. `doi:10.1137/0203006`.

[60] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the Association for Computing Machinery*, 22(2):212–225, 1975. `doi:10.1145/321879.321884`.

[61] M. A. Trick. A dynamic programming approach for consistency and propagation for knapsack constraints. *Annals of Operations Research*, 118(1):73–84, 2003. `doi:10.1023/A:1021801522545`.

[62] E. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.

[63] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979. `doi:10.1016/0304-3975(79)90044-6`.

[64] D. R. Woodall. Sufficient conditions for circuits in graphs. *Proceedings of the London Mathematical Society*, 24(4):739–755, 1972. `doi:10.1112/plms/s3-24.4.739`.

[65] J. Zaks. Extending an extension of Grinberg's theorem. *Journal of Combinatorial Theory, Series B*, 32(1):95–98, 1982. `doi:10.1016/0095-8956(82)90080-6`.