Reasoning about Connectivity Constraints
Christian Bessiere, Emmanuel Hébrard, George Katsirelos, Toby Walsh

To cite this version:
Christian Bessiere, Emmanuel Hébrard, George Katsirelos, Toby Walsh. Reasoning about Connectivity Constraints. IJCAI: International Joint Conference on Artificial Intelligence, Jul 2015, Buenos Aires, Argentina. pp.2568-2574. hal-01606489

HAL Id: hal-01606489
https://hal.science/hal-01606489
Submitted on 21 Nov 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution| 4.0 International License
Reasoning about Connectivity Constraints

Christian Bessiere
CNRS, U. Montpellier
Montpellier, France
bessiere@lirmm.fr

Emmanuel Hebrard
CNRS, U. Toulouse
Toulouse, France
hebrard@laas.fr

George Katsirelos
INRA
Toulouse, France
gkatsi@gmail.com

Toby Walsh
NICTA & UNSW
Sydney, Australia
toby.walsh@nicta.com.au

Abstract

Many problems in computational sustainability involve constraints on connectivity. When designing a new wildlife corridor, we need it to be geographically connected. When planning the harvest of a forest, we need new areas to harvest to be connected to areas that have already been harvested so we can access them easily. And when town planning, we need to connect new homes to the existing utility infrastructure. To reason about connectivity, we propose a new family of global connectivity constraints. We identify when these constraints can be propagated tractably, and give some efficient, typically linear time propagators for when this is the case. We report results on several benchmark problems which demonstrate the efficiency of our propagation algorithms and the promise offered by reasoning globally about connectivity.

1 Introduction

Many problems involve seeking a subgraph of a larger graph and connectivity is one of the most natural property to ask for in such subgraphs. It is, in particular, an essential property in several problems related to ecosystem design. For instance, in the problem of selecting sites to be included in reserve networks, fragmented networks are undesirable [Briers, 2002], and many other problems, such as forest harvesting or town planning involve this criterion. The problem of designing connected wildlife corridors has received a lot of attention recently, with a multi-commodity flow encoding [Conrad et al., 2007; 2012], a subtour elimination scheme [Dilkina and Gomes, 2010], and a hybrid local search method [Le-Bras et al., 2013]. Whilst these methods are very efficient, a constraint programming approach has a lot to offer in addition. In particular, it can very easily be extended to incorporate additional constraints. For instance, establishing protected areas while taking into account socio-economic constraints [Polasky et al., 2005], designing incentives for private landowner to contribute to “ecosystem services” [Polasky et al., 2014] both require connectivity in conjunction with other constraints and criteria. However, standard constraint programming toolkits do not support connectivity constraints, and the alternative of using a decomposition is very unlikely to be efficient. The flow decomposition proposed in [Conrad et al., 2007] has a $\Omega(|E||V|)$ space complexity and hinders propagation. Moreover such decompositions are not handled efficiently by CP solvers [Bessiere et al., 2009]. A decomposition enforcing domain consistency is even larger ($\Omega(|V|^4)$) and hence impractical. We therefore propose a global constraint to ensure that a subset of vertices induces a connected subgraph in $O(|E|)$ time and space complexity. Furthermore, we also introduce efficient algorithms to compute minimal explanations for these global constraints, thus making it possible to use this constraint in clause learning solvers.

In Section 3, we consider the constraint simply ensuring that the subgraph is connected. We show that domain consistency can be enforced on this constraint with a worst case time complexity linear in the size of the graph and a minimal explanation can be computed in the same worst case time complexity. Next, in Section 4, we consider the case where a cost function is associated to vertices and there is an upper bound on the overall cost of the subgraph. This generalization is NP-hard. However, we propose incomplete filtering rules that are very efficient in practice. Then, in Section 5, we consider the case where the cost function is associated to vertices and there is an upper bound on the overall cost of the subgraph. This generalization is NP-hard. However, we propose incomplete filtering rules that are very efficient in practice. Then, in Section 5, we consider the case where we need to find several subgraphs. This case is also NP-hard in general, even for two shapes. However, we propose implied constraints for the case of two shapes in planar graphs. Finally, in Section 6, we test empirically the algorithms and implied constraints that we introduce and show that they are efficient in practice.

2 Background

We denote by $G = (V, E)$ a graph with vertices $V$ and edges $E$. $G[S]$ is the subgraph induced by the set $S \subseteq V$. A constraint satisfaction problem (CSP) involves finding an assignment of values to its variables such that all its constraints are satisfied. Variables take values in a finite domain of values.
In the family of constraints that we consider, the subset of vertices of a subgraph is represented as an array of Boolean variables. For each vertex \( v \in V \), we have a Boolean variable that takes the value 1 if \( v \) is in the subgraph and the value 0 if it is not in the subgraph. To simplify the notations, however, we view this array as a set variable \( S \), that is, a variable whose values are sets that must be supersets of a lower bound \( S \) and subsets of an upper bound \( \bar{S} \). We say that a set \( s \) is a support iff \( S \subseteq s \subseteq \bar{S} \) and \( G[s] \) is connected. We say that \( S \) is domain consistent (DC) iff \( S \) is the intersection of all supports and \( \bar{S} \) is the union of all supports \( s \).

Several modern CP solvers employ clause learning, a technique that has had great success in SAT solvers [Marques-Silva et al., 2009]. In order to be used with clause learning, a propagator needs to label each pruned value by a clausal explanation [Katsirelos and Bacchus, 2005]. Clauses can include literals corresponding to assigning a value, \( V = a \), or pruning a value \( V \neq a \equiv \neg(V = a) \) for finite domain variables and literals corresponding to including a value \( a \in S \), or excluding it \( a \notin S \equiv \neg(a \in S) \) for set variables. A clausal explanation is correct with respect to domain consistency on a constraint \( c \) if making all its literals false makes \( c \) domain inconsistent. It is minimal if no strict subset is correct.

3 Single Shape Connectivity

In this section, we consider the constraint that ensures that a set of vertices is connected in the graph given as parameter.

Definition 1 (Connectivity) Let \( G \) be a graph and \( S \) a set variable: \( \text{Connectivity}[G](S) \iff G[S] \) is connected

3.1 Propagation

Algorithm 1 enforces domain consistency on the constraint Connectivity. The algorithm works in two phases. In the first, we simply remove the vertices that are not reachable from any vertex in \( S \) (Line 2). We compute a maximal connected component \( C \) of \( G[S] \) including any vertex \( v \) in \( S \) and remove from \( S \) any vertex that does not belong to this connected component. After this phase, notice that \( \bar{S} \) is a support since \( G[\bar{S}] \) is connected. Therefore this is sufficient to detect disentailment. We can therefore set \( G \) to \( G[\bar{S}] \) (Line 3). In the second phase, we use the notions of biconnected component and articulation point. A biconnected component of \( G \) is a set of vertices \( p \) such that for any vertex \( v \in p \), \( G[p \setminus \{v\}] \) is connected. A vertex is an articulation point if and only if removing it increases the number of connected components. Let \( p_1 \) and \( p_2 \) be two distinct maximal biconnected components, it follows that \( |p_1 \cap p_2| \leq 1 \), otherwise \( p_1 \cup p_2 \) is biconnected and larger than \( p_1 \) and \( p_2 \).

Hopcroft and Tarjan’s algorithm [Hopcroft and Tarjan, 1973] computes all articulation points in \( O(|V| + |E|) \), that is \( O(|E|) \) since \( G[S] \) is connected. It explores the graph depth-first and maintains, for each vertex \( v \), the lowest depth of neighbors of all descendants of \( v \) in the depth-first-search tree, called the lowpoint. A vertex \( v \) is an articulation point if and only if it has a child \( u \) in the dfs such that \( \text{lowpoint}[u] \geq \text{depth}[v] \). Indeed, it means that the only path from \( u \) to vertices with lower depth in the dfs goes through \( v \) hence removing \( v \) would cut the graph. By very slightly modifying Hopcroft and Tarjan’s algorithm to record for each vertex \( v \) also the vertex whose depth is recorded as \( \text{lowpoint}[v] \), we get the collection \( P_G \) of \( G \)’s biconnected components.

A call to Hopcroft and Tarjan’s algorithm (Line 4) returns the collection of sets \( P_G \), the set of articulation points \( A_G \) of \( G \) and a table of predecessors \( \text{pred} \) corresponding to the depth-first-search tree. Next, in Loop 6, we build a tree \( T(G) = (V^T, E^T) \) whose vertices stand for articulation points and biconnected components of \( G \). (see Fig. 1):

- There is a vertex labeled \( p \) per biconnected component \( p \in P_G \) and one labeled \( \{a\} \) per articulation point \( a \) in \( G \), i.e., \( V^T = P_G \cup \{\{a\} \mid a \in A_G\} \);

- There is an edge \( \{a\}, p \) if the articulation point \( a \) is in the biconnected component \( p \), and an edge \( \{(a_1), (a_2)\} \) if the articulation points \( a_1 \) and \( a_2 \) share an edge but do not belong to the same biconnected component.

Observe that in \( T(G) \) every edge involves at least an articula-
tion point of $G$ and it is indeed a tree, otherwise removing an articulation point in a cycle of $T(G)$ would not disconnect it.

**Lemma 1** Let $x$ and $y$ be two vertices of $T(G)$ such that $x \cap S \neq \emptyset$ and $y \cap S \neq \emptyset$, then every articulation point $a$ of $G$ such that $\{a\}$ is on a path from $x$ to $y$ in $T(G)$ must be in $S$.

**Proof:** Suppose that there is an elementary path from $x$ to $y$ in $T(G)$ going through $\{a\}$ where $a$ is an articulation point of $G$. By construction of $T(G)$, there is no path in $G$ from any vertex in $x$ to any vertex in $y$ that does not go through $a$. Therefore, if vertices of both $x$ and $y$ are in the connected component, then $a$ must also be in the component. □

A corollary of Lemma 1 is that once we have pruned $T(G)$ until every leaf $x$ is such that $x \cap S \neq \emptyset$, we know that the same applies to every node of the tree that corresponds to an articulation point of $G$. It follows that for every articulation point $a$ such that $\{a\}$ remains in $T(G)$ after pruning, we can set $a \in S$. For reasons of space, we do not give the pseudo code for the procedure prune_tree. It iteratively removes nodes of degree 1 whose label contains no vertex in $S$.

**Theorem 1** Algorithm 1 achieves DC in $O(|E|)$ time

**Proof:** The correctness of the pruning is straightforward for vertices that are not connected to $G[S]$ (Line 2), and proved by Lemma 1 for articulation points (Line 7). It remains to show completeness. We consider a vertex $v \in S \setminus S$ and show that there exists a solution where $v \in S$ and another where $v \not\in S$. Since $S$ is a single connected component, it is a solution and it accounts for the case $v \in S$. Now consider first the case where $v$ is not an articulation point. Then $v$ belongs to a single biconnected component $p$ of $G[S]$. We can exclude $v$ from $S$ and by definition of a biconnected component, connect all remaining vertices. Therefore $S \setminus \{v\}$ is a solution. Lastly, suppose that $v$ is an articulation point. Since it has not been added to $S$ in Line 7 then it must have been removed from $T(G)$. Moreover, the union of the vertices of the residual tree is a superset of $S$ that are connected in $G$. Therefore it is a solution where $v \not\in S$. □

### 3.2 Explanation

We need to explain two types of pruning: First, in Line 2 of Algorithm 1, when the current maximal subgraph is not connected, we arbitrarily choose a node $s$ from one connected component and exclude the nodes of all other components from $S$. Given a node $t$ from another component, we compute a set of nodes $F$ such that $F \cap S = \emptyset$ and every path linking $s$ to $t$ has a node in $F$. In order to do so, we do a breadth first search from $s$ stopping at nodes in $V \setminus S$. Let $B$ denote the set of leaves of this BFS. We then perform another BFS from $t$ until we reach a node in $B$. The set of leaves of this second BFS is $F$. The explanation clause is then:

$$s \not\in S \lor t \not\in S \lor \bigvee_{u \in F} u \in S$$  \hspace{1cm} (1)

Second, in Line 7 of Algorithm 1, articulation points are added to $S$. The explanation is similar, except that we start from an articulation point $v$. We arbitrarily choose two nodes $s, t \in S$ by exploring the tree $T(G)$ breadth first from $\{v\}$. Then the first BFS in $G$ stops when exploring nodes in $(V \setminus \bar{S}) \cup \{v\}$ to return a set of leaves $B$. The second BFS is unchanged, and the explanation is:

$$s \not\in S \lor t \not\in S \lor v \in S \lor \bigvee_{u \in F} u \in S$$  \hspace{1cm} (2)

**Theorem 2** Explanations 1 and 2 are correct and minimal.

**Proof:** All paths from $s$ to $t$ go through a vertex in $F$, so if these vertices are removed from $G$, then $s$ and $t$ are disconnected, hence explanation 1 is correct. The argument for explanation 2 is nearly the same. By definition of articulation points, any path from $s$ to $t$ in $G$ goes through a node in $F$ or through $v$, hence removing $F \cup \{v\}$ disconnects $s$ and $t$ in $G$. Now consider any vertex $u \in F$, (resp. $u \in F \cup \{v\}$). If we remove from $G$ all vertices in $F$ (resp. $F \cup \{v\}$) except $u$, there exists a path from $s$ to $t$ in $G$. This path is a support for the constraint, hence explanations 1 and 2 are minimal. □

### 4 Weighted connectivity

We now consider the weighted version of the constraint:

**Definition 2** (W**EIGHTED-CO**NNECTIVITY) Let $G$ be a graph, $S$ a set variable, $W$ an integer variable and $\omega : V \rightarrow \mathbb{N}$ a mapping from vertices to non-negative integers.

**WEIGHTED-CO**NNECTIVITY$[G, \omega](S, W)$ $\iff$ $G[S]$ is connected $\land \sum_{v \in S} \omega(v) \leq W$

Deciding the existence of a support for the constraint WEIGHTED-CO**NNECTIVITY** is NP-complete as it solves the STEINER TREE problem. Hence, enforcing DC is NP-hard, as NP-hardness of support implies NP-hardness of DC [Bessiere et al., 2007]. Therefore, we do not try to achieve DC, but propose an incomplete algorithm taking into account the cost of the connected component.

#### 4.1 Propagation

The main difference with Algorithm 1 lies in the first step (Line 1). Instead of computing a connected component, we compute the subset of vertices for which there exists a path to every vertex in $S$ that is short enough (see Fig. 2a and 2b).

Moreover, we bound the minimum value $\min(W)$ by $\sum_{v \in S} \omega(v)$. From now on, we consider the graph $G[S]$. The length of a path in this graph is defined as the sum of the weights of the vertices in this path that are not in $S$. It follows that if a vertex $v$ is in the subgraph induced by $S$, then the total weight of this subgraph is at least $\min(W)$ (the weights of vertices in $S$) plus the maximum over all minimum lengths of paths between $v$ and any element of $S$. We run Dijkstra’s algorithm once for each vertex $u \in S$ to compute the shortest path from $u$ to all other vertices, which can be done in $O(|E| + |V| \log |V|)$.\(^1\) Let $s(u \rightarrow v)$ be the length of the shortest path to vertex $v$. For every vertex $v$ such that $s(u \rightarrow v) + \min(W) > \max(W)$, we remove $v$ from $S$. This step can thus be done in $O(|V||E| + |V|^2 \log |V|)$ time.

\(^1\)We used a binary heap hence a $O(|E| \log |V|)$ time complexity.
pruning is then done in the same way: by adding the articulation point to the set.

When \( w \) is added to the cut, we do not explore further through that vertex. Moreover, since the reachability depends on the maximum cost, we must add the literal \( \neg W > \max(W) \).

The final explanation is \( W > \max(W) \lor \forall u \not\in S \lor \forall v \not\in S \lor \bigvee_{w \in F} w \in S \) (see Fig. 2c). As in the unweighted case, explaining the second type of pruning is done in the same way: by adding the articulation point to the set \( B \).

The amortized worst case time complexity to explain this pruning is then \( O(|V||E| + |V|^2 \log |V|) \) down a branch.

### 5 Multiple Subgraphs

Finally we consider the case where we want to ensure that several disjoint subgraphs are connected.

**Definition 3 (MultiConnectivity)** Let \( G = (V, E) \) be a graph, \( \{X_i \mid i \in V\} \) a set of variables and \( R \) a set of integers.

\[
\text{MultiConnectivity}[G, R][X_1, \ldots, X_n] \iff \\
\forall j \in R, G[\{i \mid X_i = j\}] \text{ is connected}
\]

Clearly we can decompose \text{MultiConnectivity} using \text{Connectivity}, but it hinders propagation since enforcing \text{Dijkstra} on \text{MultiConnectivity} is NP-hard [Karp, 1975].

### 5.1 General graphs

We study the following problem, a special case of finding disjoint paths which is not covered by the results in [Karp, 1975] or [Fortune et al., 1980].

**Name:** Two-Shape-Connectivity

**Input:** A connected graph \( G = (V, E) \) and a labeling of the vertices \( m : V \rightarrow \{0, 1, U\} \)

**Problem:** Does there exist a labeling \( m' : V \rightarrow \{0, 1, U\} \) of the vertices so that \( m'(v) = m(v) \) if \( m(v) \in \{0, 1\} \) and the components \( C_0, C_1 \) with \( C_i = \{v \mid m'(v) = i\} \) are connected?

Let \( m[i] \) be the domain of a variable \( X_i \), it can be modeled as: \text{MultiConnectivity}\[G, \{0, 1\}\][\{X_i \mid i \in V\}]

**Theorem 3** Two-Shape-Connectivity is NP-hard.

**Proof:** We reduce from an instance of 3-SAT with \( n \) variables and \( m \) clauses. We first introduce two gadgets, \( \neq(v_1, v_2) \) and \( \lor(v_1, v_2, v_3) \). The \( \neq(v_1, v_2) \) gadget is a complete bipartite graph with partitions \( \{l_0, l_1, r_0, r_1\}, \{v_1, v_2\} \). We have \( m(l_0) = m(r_0) = 0, m(l_1) = m(r_1) = 1 \) and \( m(v_1) = m(v_2) = U \). This gadget ensures that \( m'(v_1) \neq m'(v_2) \).

Suppose wlog \( m'(v_1) = m'(v_2) = 1 \). Then \( l_0 \) is disconnected from \( r_0 \), which is not permitted. For the same reason, vertices \( v_1, v_2 \) cannot be both labeled 0. In order to combine many \( \neq \) gadgets in one graph, we need to make sure that the resulting \( C_0, C_1 \) partitions are connected. To do this, we introduce globally two distinguished vertices \( g_0, g_1 \) with \( m(g_0) = 0, m(g_1) = 1 \) and connect \( g_0 \) with the vertex \( l_0 \) and \( g_1 \) with \( l_1 \) in each \( \neq \) gadget. The \( \lor(v_1, v_2, v_3) \) gadget ensures that \( m(v_i) = 1 \) for at least one of \( v_1, v_2, v_3 \). It is a complete bipartite graph with partitions \( \{l, r\} \) and \( \{v_1, v_2, v_3\} \).
and \( m(l) = m(r) = 1 \). In order for \( l \) and \( r \) to be connected, at least one of \( v_1, v_2, v_3 \) must be in the component \( C_1 \).

From the 3-SAT formula, we construct a graph with \( n \) instances of the \( \neq \) gadget and \( m \) instances of the \( \lor \) gadget. In particular, for each variable \( x_i \), we have two vertices \( x_i \) and \( \overline{x}_i \) and the gadget \( \neq(x_i, \overline{x}_i) \). For each clause \((l_1, l_2, l_3)\), we have an instance of the gadget \( \lor(l_1, l_2, l_3) \), where \( l_i \) is a literal, i.e. either \( x_i \) or \( \overline{x}_i \). From a labeling \( m' \) we construct a solution of the 3-SAT formula with \( x_i = 1 \iff m'(x_i) = 1 \). Because of the \( \lor \) gadgets, at least one of the literals of each clause will be assigned 1, hence the formula will be satisfied. In the other direction, we use the same rule to construct a labeling from a solution of the 3-SAT formula. By construction, the vertices in the \( \neq \) gadgets will be connected. Since each clause is satisfied, one of the literal vertices of each \( \lor \) gadget will be labeled with 1. Hence, the vertices \( l \) and \( r \) of that gadget will be connected to each other through that literal vertex and, since that literal vertex appears in a \( \neq \) gadget, they are also connected to the rest of the \( C_1 \) partition.

5.2 Two Shapes in Planar Graphs

The conjunction of an arbitrary number of CONNECTIVITY constraints is known to be NP-hard even on planar graphs [Lynch, 1975], but fixed parameter tractable on the number of terminals [Reed et al., 1991]. It is not known whether this remains the case for two constraints on a planar graph. We show here that decomposition into two CONNECTIVITY constraints for the case of two shapes will hinder propagation. Consider the graphs in figure 3. In both examples the connectivity constraints are DC, but the conjunction is disentailed. In the example on the left, the border is assigned with pattern 1/0/1/0. This means that traversing the border we see the label 1, then 0, then 1 and then 0 again, potentially with repetitions or with unlabeled vertices in between. In the example on the right, the same pattern appears in the cycle \( v_9, v_A, v_B, v_C \).

In order to detect disentailment, one must forbid the pattern 1/0/1/0 in every face of a planar graph. This can be done, for example, with a REGULAR constraint forbidding two alternations between 1 and 0, which requires 8 states, hence can be enforced in linear time in the size of the face. Since an edge belongs to at most two faces, enforcing DC on all face constraints can be done in time \( O(|E|) \). However, having the face constraints is not sufficient for detecting disentailment. We do not show counterexamples for lack of space.

6 Experimental Results

In this section we report the results of a series of experiments on three benchmarks involving connectivity constraints.

<table>
<thead>
<tr>
<th>size</th>
<th>5 terminals</th>
<th>7 terminals</th>
<th>10 terminals</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>imp</td>
<td>opt</td>
<td>time</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>100</td>
<td>0.10</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>100</td>
<td>0.44</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>100</td>
<td>3.63</td>
</tr>
<tr>
<td>11</td>
<td>8</td>
<td>95</td>
<td>4.56</td>
</tr>
<tr>
<td>12</td>
<td>5</td>
<td>95</td>
<td>7.64</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>75</td>
<td>19.42</td>
</tr>
<tr>
<td>14</td>
<td>9</td>
<td>75</td>
<td>57.71</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>65</td>
<td>26.50</td>
</tr>
</tbody>
</table>

Wildlife corridor First, we test a hybrid CP/SAT approach using the constraint WEIGHTED-CONNECTIVITY for the wildlife corridor problem described in [Conrad et al., 2012]. We implemented all techniques with minicsp 2 and ran experiments on a cluster of 48-core Opteron 6176 nodes at 2.3 GHz with 378 GB RAM available. Given a graph representing a region, where vertices stand for parcels, a subset of parcels are wildlife reserves. The goal is to design a corridor so that all reserves are connected while minimizing a (positive) cost and/or maximizing a (positive) utility. In [Conrad et al., 2012], the problem is solved in two phases. First, a Steiner tree of minimal cost is obtained by computing all pairs shortest paths and selecting those connecting each reserves to the best “center point”. This method is very efficient, however it is optimal only for three reserves or less. Second, the utility maximization problem is solved with a MIP, using the Steiner tree above to either guide search, or to fix the selected parcels into the solution.

Let \( G = (V, E) \) be a graph describing the region with \( |V| = n \), \( \{r_1, \ldots, r_k\} \subseteq V \) a set of reserves and \( \omega \) a cost function for parcels. We have a single set variable \( S \) such that \( S = \{r_1, \ldots, r_k\} \) and \( \bar{S} = V \), an integer variable \( W \) with domain \([0, \ldots, \sum_{v \in V} \omega(v)]\). The first phase corresponds to minimizing \( W \) subject to a single constraint: WEIGHTED-CONNECTIVITY\((G, \omega, (S, W)) \). For the second phase, the same constraint can be used to ensure that the corridor is connected and that its cost is below a given budget (derived from the min cost solution by adding a certain slack). The utility can be maximized with a simple inequality constraint. However, this is not competitive with Gomes et al. when the slack grows. Indeed, for a large slack, the WEIGHTED-CONNECTIVITY is not so critical and the communication with the inequality constraint taking care of the utility is not as good as it is in MIP. However, our approach can be useful in the first phase when we have more than three reserves. We used the generator of Gomes et al. to create 20 random instances of 24 classes parameterized by the order of the grid graph (in \([8, 15] \)) and the number of reserves (in \([5, 7, 10] \)). We report in Table 1, for each class of

\[\text{http://www.inra.fr/fia/Tk/katsirelos/minicsp.html}\]
20 instances, the improvement of the cost with respect to the upper bound found by Gomes et al. (imp), the percentage of instances for which optimality was proven (opt), and the average cpu time for computing the proofs when it was possible within the cutoff of 300 seconds (time).

We can clearly see that the upper bound can be significantly improved. However, the computational time grows both with the number of terminals and with the size of the graph.

**Ecosystem design.** In order to compare the various algorithms that we have introduced, we use a problem similar to wildlife corridor design. However, besides reserves, we also have parcels that correspond to agricultural activities (farms). We want to compute a minimum cost connected wildlife corridor, but also to make sure that farms can be connected by roads. We therefore have two set variables $S$ and $T$ standing respectively for the set of parcels that are in the corridor, or witness of a connection between farms. We first ensure that a parcel can only account for one of these two cases, then we post the following constraints: \textsc{connectivity}$(G, T)$ and \textsc{weighted-connectivity}$(G, \omega)(S, W)$. The former ensures that the farms can be connected whilst the latter ensures that the corridor is connected and links its cost with $W$. The objective is to minimize the total cost of the corridor $W$.

We use 20 random instances of 12 classes, parameterized by number of reserves and farms (10/5 and 5/10), and by the order of the grid in $[8, 13]$. We compare four models. The connectivity and cost minimization of the corridor is either decomposed with a \textsc{connectivity} and a linear inequality (connect) or modeled directly with the weighted version (weighted). Then, in both cases, we try with or without face constraints (denoted “+face”). We also report results without clause learning (denoted by “*”). We omit comparison against a decompositon as it was several orders of magnitude slower even in very small instances. We report the same data in Table 2 as in Table 1, except for the objective value (obj.). The first observation is that learning has a large positive impact. In terms of number of instances solved to (proven) optimality, the strongest propagation method (weighted+face) is the best. However, when the size of the instances grows and finding the optimal solution within the time cutoff becomes difficult, the lower time complexity of the non-weighted variant can be more beneficial than the extra pruning of the weighted variant. The face constraints, however, are cheap and nearly always beneficial.

**Clueless Puzzles** Clueless Puzzles were proposed by Butters, Henle, Henle, & McGaughey as a mathematical challenge. The goal is to design Sudoku-like puzzles without numerical clues. The answer must be a latin square. However, there are no pre-filled cells and no extra \textsc{alldifferent} constraints. Instead, the grid is partitioned into regions of equal sum (see examples in Figure 4). Since the solution must form a latin square, we can deduce that, for a puzzle of order $n$ the sum in each of the $m$ regions equals $n^2(n+1)/(2m)$. We model the problem of generating puzzles as follows:

\[ \text{EIGHTED} \]

To ensure that the puzzle has a solution we use $n^2$ integer variables with domain $\{1, \ldots, n\}$, and 2$n$ \textsc{alldifferent} constraints to ensure we have a latin square. Next, to obtain a partition into regions, we use $m$ set variables whose elements are the cells of the grid. We post that the scalar product of the characteristic function of each set variable with the latin square is equal to $n^2(n+1)/(2m)$. Moreover, they must be disjoint, their union must cover the whole grid, and of course they must be connected. Notice that this model does not guarantee that the puzzle will have a single solution, which is a necessary feature. Indeed, this problem is not in NP, and thus is unlikely to be formulated as a simple CSP. We therefore check this property once a feasible puzzle is generated (by attempting to find a different solution), and continue searching when this does not hold. Using this simple program we were able to generate several new instances, for classes ($n, m$) of puzzles for which known solutions already exist (such as $\langle 4, 2 \rangle$, $\langle 5, 2 \rangle$, $\langle 5, 5 \rangle$, $\langle 6, 9 \rangle$ and $\langle 6, 14 \rangle$), but also for entirely new classes: $\langle 6, 6 \rangle$, $\langle 6, 7 \rangle$ and $\langle 7, 7 \rangle$ (see examples in Fig. 4).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{puzzles.png}
\caption{Examples of new puzzles found by the CP model.}
\end{figure}

7 Conclusions

We have studied a family of constraints to reason about connectivity that are useful in a range of problems in computational sustainability. First, we showed that one can enforce domain consistency and compute a minimal explanation for the pruning in $O(|E|)$ time for a basic connectivity constraint. Second, we proposed an incomplete algorithm to take weights into account since enforcing DC in this case is NP-hard. Third, we showed that partitioning into several connected subgraphs is intractable, but proposed efficient implied constraints for the case of two subgraphs of a planar graph. Finally, we empirically tested our algorithms on several benchmarks and showed that they can be practical and have promise for a wide range of problems.

Acknowledgements

We would like to thank Ashish Sabharwal for his help with the wildlife corridor datasets [Conrad et al., 2007].

References

Table 2: Algorithms comparison on Ecosystem design. Best results in bold face

<table>
<thead>
<tr>
<th></th>
<th>connect</th>
<th>connect*</th>
<th>weighted</th>
<th>weighted*</th>
<th>connect+face</th>
<th>connect+face*</th>
<th>weighted+face</th>
<th>weighted+face*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>opt. obj. time</td>
<td>opt. obj. time</td>
<td>opt. obj. time</td>
<td>opt. obj. time</td>
<td>opt. obj. time</td>
<td>opt. obj. time</td>
<td>opt. obj. time</td>
<td>opt. obj. time</td>
</tr>
<tr>
<td>8</td>
<td>100 1104 0.9</td>
<td>20 1104 131.1</td>
<td>100 1104 0.8</td>
<td>65 1104 89.5</td>
<td>100 1104 1.0</td>
<td>0 1104</td>
<td>-</td>
<td>100 1104 0.7</td>
</tr>
<tr>
<td>9</td>
<td>100 1228 54.6</td>
<td>0 1229 -</td>
<td>100 1228 15.8</td>
<td>20 1231 74.4</td>
<td>100 1228 50.6</td>
<td>0 1265</td>
<td>-</td>
<td>100 1228 14.9</td>
</tr>
<tr>
<td>11</td>
<td>30 1368 59.9</td>
<td>0 1369 -</td>
<td>70 1370 86.1</td>
<td>0 1386</td>
<td>30 1378 74.7</td>
<td>0 1411</td>
<td>75 1362 96.8</td>
<td>0 1413</td>
</tr>
<tr>
<td>12</td>
<td>5 1489 291.2</td>
<td>0 1428 -</td>
<td>35 1506 153.2</td>
<td>0 1475</td>
<td>0 1540 -</td>
<td>20 1562 214.9</td>
<td>0 1470</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>5 1707 11.6</td>
<td>0 1668</td>
<td>15 1897 94.2</td>
<td>0 5037</td>
<td>5 1639 23.5</td>
<td>0 1655</td>
<td>10 1911 18.7</td>
<td>5 1868 94.1</td>
</tr>
<tr>
<td>10 res., 5 farms</td>
<td>0 2289 -</td>
<td>0 3516 -</td>
<td>0 2815 -</td>
<td>0 7888</td>
<td>0 2255 -</td>
<td>0 2200</td>
<td>-</td>
<td>0 2878 -</td>
</tr>
<tr>
<td>8</td>
<td>100 901 0.6</td>
<td>35 901 103.7</td>
<td>100 901 0.3</td>
<td>85 901 26.3</td>
<td>100 901 0.3</td>
<td>10 931 100.0</td>
<td>100 901 0.3</td>
<td>70 903 9.7</td>
</tr>
<tr>
<td>9</td>
<td>100 990 6.5</td>
<td>0 1002 -</td>
<td>100 990 5.3</td>
<td>75 990 65.7</td>
<td>100 990 6.2</td>
<td>0 1014</td>
<td>-</td>
<td>100 990 2.7</td>
</tr>
<tr>
<td>10</td>
<td>70 1092 39.5</td>
<td>0 1091 -</td>
<td>100 1037 37.2</td>
<td>30 1077 104.6</td>
<td>80 1078 86.0</td>
<td>0 1126</td>
<td>-</td>
<td>100 1037 31.5</td>
</tr>
<tr>
<td>11</td>
<td>10 1222 60.3</td>
<td>0 1205 -</td>
<td>60 1188 83.8</td>
<td>20 1438 55.3</td>
<td>10 1242 109.0</td>
<td>0 1300</td>
<td>-</td>
<td>65 1199 112.5</td>
</tr>
<tr>
<td>12</td>
<td>10 1484 152.0</td>
<td>0 1382 -</td>
<td>20 1584 119.6</td>
<td>10 4052 15.5</td>
<td>0 1496</td>
<td>0 1439</td>
<td>-</td>
<td>30 1576 63.0</td>
</tr>
<tr>
<td>13</td>
<td>0 1713 -</td>
<td>0 3034 -</td>
<td>5 2128 135.8</td>
<td>0 7934</td>
<td>0 1687 -</td>
<td>0 1595</td>
<td>-</td>
<td>5 2082 288.3</td>
</tr>
</tbody>
</table>


