Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

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 [LeBras *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 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 $\underline{S}$ and subsets of an upper bound $\bar{S}$. We say that a set s is a support iff $\underline{S} \subseteq s \subseteq \bar{S}$ and $G[s]$ is connected. We say that S is *domain consistent* (DC) iff $\underline{S}$ 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 $\underline{S}$ (Line 2). We compute a maximal connected component C of $G[\bar{S}]$ including any one vertex v in $\underline{S}$ and remove from $\bar{S}$ 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.

Algorithm 1: DC on CONNECTIVITY (G, S)

if $\exists v \in V$ *such that* $v \in \underline{S}$ **then**
1. $C \leftarrow \texttt{ConnectedComponent}(v, G[\bar{S}])$;
2. **foreach** $v \in V \setminus C$ **do** $\bar{S} \leftarrow \bar{S} \setminus \{v\}$;
3. $G \leftarrow G[\bar{S}]$;
4. $P_G, A_G, pred \leftarrow \texttt{Hopcroft\_Tarjan}(G)$;
   $E^T \leftarrow \emptyset$; $V^T \leftarrow P_G \cup \{\{a\} \mid a \in A_G\}$;
5. **foreach** $v \in \bar{S}$ **do**
   - $e \leftarrow \{v, pred[v]\}$;
   - **if** $e \cap A_G \neq \emptyset$ **then**
     - **if** $\exists p \in P_G$ *s.t.* $e \subseteq p$ **then**
       - **foreach** $a \in e \cap A_G$ **do** add $(\{a\}, p)$ to E^T;
     - **else if** $e \subseteq A_G$ **then** add $(\{v\}, \{pred[v]\})$ to E^T;
6. $V^T \leftarrow \texttt{prune\_tree}(V^T, E^T, G, \underline{S})$;
   foreach $v \in V^T$ **do**
7. - **if** $v \in A_G$ **then** $\underline{S} \leftarrow \underline{S} \cup \{v\}$;

Hopcroft and Tarjan's algorithm [Hopcroft and Tarjan, 1973] computes all articulation points in $O(|V| + |E|)$, that is $O(|E|)$ since $G[\bar{S}]$ 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 $lowpoint[u] \geq depth[v]$. 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 $lowpoint[v]$, 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 $pred$ corresponding to the depth-first-search tree. Next, in Loop 5, 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-

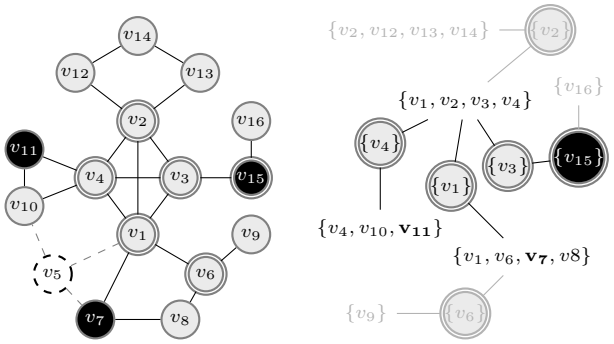


Figure 1: In all figures, black nodes are in $\underline{S}$, white nodes are not in $\bar{S}$ and gray nodes are undecided. A graph (left). Articulation points, delimiting biconnected components, are doubly circled. The corresponding tree $T(G)$ (right). Pruned branches are grayed, elements in $\underline{S}$ are in bold face. All remaining articulation points (v_1, v_3, v_4) must be in S.

tion point of G and it is indeed a tree, otherwise removing an articulation point in a cycle of $T(G)$ would not disconnect G.

Lemma 1 *Let x and y be two vertices of $T(G)$ such that $x \cap \underline{S} \neq \emptyset$ and $y \cap \underline{S} \neq \emptyset$, 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 $x \cap \underline{S} \neq \emptyset$, 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 $a \in \underline{S}$. 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 $\underline{S}$.

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 $G[\bar{S}]$ (Line 2), and proved by Lemma 1 for articulation points (Line 7). It remains to show completeness. We consider a vertex $v \in \bar{S} \setminus \underline{S}$ and show that there exists a solution where $v \in S$ and another where $v \notin S$. Since $\bar{S}$ 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 $G[\bar{S}]$). We can exclude v from S and by definition of a biconnected component, connect all remaining vertices. Therefore $\bar{S} \setminus \{v\}$ is a solution. Lastly, suppose that v is an articulation point. Since it has not been added to $\underline{S}$ 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 $\underline{S}$ that are connected in G. Therefore it is a solution where $v \notin S$. □

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 $F \cap \bar{S} = \emptyset$ 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 $V \setminus \bar{S}$. 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:

$$s \notin S \ \vee \ t \notin S \ \vee \ \bigvee_{u \in F} u \in S \tag{1}$$

Second, in Line 7 of Algorithm 1, articulation points are added to $\underline{S}$. The explanation is similar, except that we start from an articulation point v. We arbitrarily choose two nodes $s, t \in \underline{S}$ 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 :

$$s \notin S \ \vee \ t \notin S \ \vee \ v \in S \ \vee \ \bigvee_{u \in F} u \in S \tag{2}$$

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 (WEIGHTED-CONNECTIVITY**)** *Let G be a graph, S a set variable, W an integer variable and $\omega : V \mapsto \mathbb{N}$ a mapping from vertices to non-negative integers.*

$$\text{WEIGHTED-CONNECTIVITY}[G, \omega](S, W) \iff G[S] \text{ is connected } \wedge \sum_{v \in S} \omega(v) \leq W$$

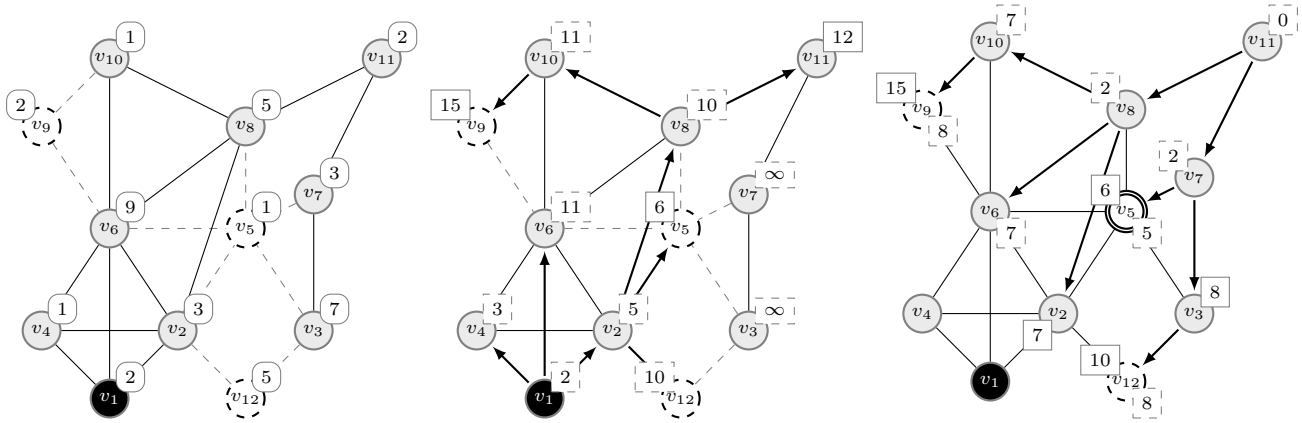
Deciding the existence of a support in the constraint WEIGHTED-CONNECTIVITY is NP-complete as it solves the STEINER TREE problem. Hence, enforcing DC is NP-hard, as NP-hardness of support implies NP-hardness of DC [Bessiere *et al.*, 2007]. Therefore, we do not try to achieve DC, but propose an incomplete 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 $\underline{S}$ that is short enough (see Fig. 2a and 2b).

Moreover, we bound the minimum value $\min(W)$ by $\sum_{v \in \underline{S}} \omega(v)$. From now on, we consider the graph $G[\bar{S}]$. 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 $\underline{S}$. 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 $\underline{S}$) plus the maximum over all minimum lengths of paths between v and any element of $\underline{S}$. We run Dijkstra's algorithm once for each vertex $u \in \underline{S}$ to compute the shortest path from u to all other vertices, which can be done in $O(|E| + |V| \log |V|)$.[1] Let $s(u \to v)$ be the length of the shortest path to vertex v. For every vertex v such that $s(u \to v) + \min W > \max(W)$, we remove v from $\bar{S}$. 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.



(a) $\{1\} \subseteq S \subseteq \{1, 2, 3, 4, 6, 7, 8, 10, 11\}$ (b) Dijkstra from v_1. (c) Explanation of $v_{11} \notin S$.

Figure 2: An example of propagation & explication. 2a: Data, labels are costs. 2b: Trace of Dijkstra's algorithm, labels are shortest paths from v_1 on $G[\bar{S}]$. Since the shortest paths from v_1 to v_3, v_7 and v_{11} are all longer than 11, these vertices will be pruned from $\bar{S}$. 2c: Explanation of the pruning $v_{11} \notin S$. Dijkstra's algorithm is run from v_{11} until going over the maximum length, or reaching a vertex in $\{v_5, v_9, v_{12}\}$. Only the path through v_5 is short enough ($s(v_1 \to v_5) = 6$, $s(v_{11} \to v_5) - \omega(v_5) = 5$), so the explanation clause will be $W > 11 \ \vee \ v_5 \in S \ \vee \ v_1 \notin S \ \vee \ v_{11} \notin S$.

4.2 Explanation

In order to explain this pruning, we use an algorithm similar as in the unweighted case, however again using Dijkstra's algorithm instead of a simple BFS. Let $u \in S, v \in \bar{S}$ be two vertices such that the length $s(u \to v)$ of the shortest path in G w.r.t. the weight function ω is greater than $\max(W)$.

We compute a cut F defined as follows: $F \cap \bar{S} = \emptyset$ and every path in G from u to v of length less than or equal to $\max(W) - \min(W)$ has a vertex in F. We first run Dijkstra's algorithm on $G[\bar{S} \cup B]$, where B is the set of vertices in $G \setminus G[\bar{S}]$ adjacent to vertices in $G[\bar{S}]$ to compute the shortest paths from u to every node in B. Next, we run Dijkstra's algorithm again, however from v and on the graph G. In this run, whenever we reach a node $w \in B$, we add it into the cut F if $s(u \to w) + s(v \to w) - \omega(w) + \min(W) \leq \max(W)$. 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 "$W > \max(W)$".

The final explanation is $W > \max(W) \ \vee \ u \notin S \ \vee \ v \notin S \ \vee \ \bigvee_{w \in F} w \in S$ (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 MULTICONNECTIVITY using CONNECTIVITY, but it hinders propagation since enforcing DC on 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 $G = \langle V, E \rangle$ and a labeling of the vertices $m : V \to \{0, 1, U\}$
PROBLEM: Does there exist a labeling $m' : V \to \{0, 1, U\}$ 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: 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 $\vee(v_1, v_2, v_3)$. 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 $\vee(v_1, v_2, v_3)$ 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 $\{v_1, v_2, v_3\}$

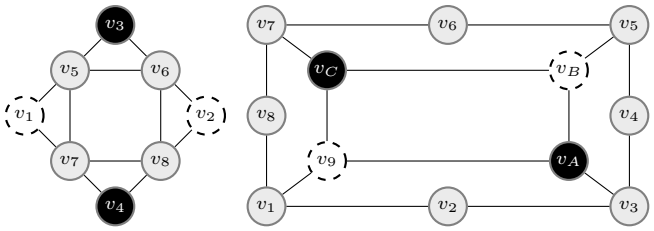


Figure 3: Examples where disentailment is not found by a decomposition into CONNECTIVITY. White vertices are labeled 0, black vertices are labeled 1 and gray vertices are unlabeled.

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 $\vee$ 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 $\vee(l_1, l_2, l_3)$, 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 $\vee$ 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 $\vee$ 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 1: Improvement w.r.t Gomes et al upper bound

size	5 terminals			7 terminals			10 terminals		
	imp	opt	time	imp	opt	time	imp	opt	time
8	7	100	0.10	10	100	0.20	16	100	0.49
9	7	100	0.44	9	100	1.00	16	100	4.32
10	8	100	3.63	7	100	9.46	16	100	39.17
11	8	95	4.56	14	90	33.29	17	75	75.33
12	5	95	7.64	14	85	36.70	8	30	53.88
13	4	75	19.42	6	50	66.53	6	25	138.66
14	9	75	57.71	4	25	57.50	2	10	139.73
15	5	65	26.50	1	20	20.99	2	0	-

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 ω 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

[2] http://www.inra.fr/mia/T/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: CONNECTIVITY$[G](T)$ and 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 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.[3]. 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 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:

[3]http://www.math.smith.edu/~jhenle/clueless

To ensure that the puzzle has a solution we use n^2 integer variables with domain $\{1, \ldots, n\}$, and $2n$ 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

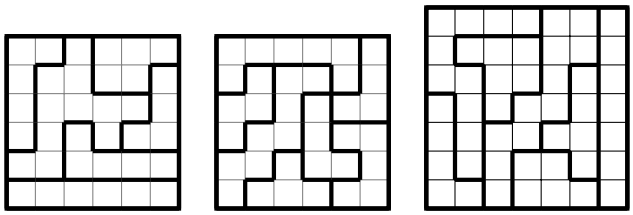


Figure 4: Examples of new puzzles found by the CP model.

able to generate several new instances, for classes ($\langle n, m \rangle$) 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).

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

[Bessiere *et al.*, 2007] Christian Bessiere, Emmanuel Hebrard, Brahim Hnich, Zeynep Kiziltan, and Toby Walsh. The Complexity of Reasoning with Global Constraints. *Constraints*, 12(2):239–259, 2007.

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

	n	connect			connect*			weighted			weighted*			connect+face			connect+face*			weighted+face			weighted+face*		
		opt	obj.	time	opt	obj.	time	opt	obj.	time	opt	obj.	time	opt	obj.	time	opt	obj.	time	opt	obj.	time	opt	obj.	time
10 res., 5 farms	8	**100**	**1104**	0.9	20	**1104**	131.1	**100**	**1104**	0.8	65	**1104**	89.5	**100**	**1104**	1.0	0	**1104**	-	**100**	**1104**	**0.7**	20	1112	10.5
	9	**100**	**1228**	54.6	0	1229	-	**100**	**1228**	15.8	20	1231	74.4	**100**	**1228**	50.6	0	1265	-	**100**	**1228**	**14.9**	5	1252	22.8
	10	30	1368	59.9	0	1390	-	70	1370	86.1	0	1386	-	30	1378	74.7	0	1411	-	**75**	**1362**	**96.8**	0	1413	-
	11	5	1489	291.2	0	1528	-	**35**	1506	153.2	0	3376	-	0	1475	-	0	1540	-	20	1562	214.9	0	**1470**	-
	12	5	1707	11.6	0	1668	-	**15**	1897	94.2	0	5037	-	5	**1639**	23.5	0	1655	-	10	1911	**18.7**	5	1686	94.1
	13	**0**	2289	-	**0**	3516	-	**0**	2815	-	**0**	7888	-	**0**	2255	-	**0**	**2200**	-	**0**	2878	-	**0**	2242	-
5 res., 10 farms	8	**100**	**901**	0.6	35	**901**	105.7	**100**	**901**	0.3	85	**901**	26.3	**100**	**901**	0.5	10	931	100.0	**100**	**901**	**0.3**	70	903	9.7
	9	**100**	**990**	6.5	0	1002	-	**100**	**990**	5.3	75	**990**	65.7	**100**	**990**	6.8	0	1014	-	**100**	**990**	**2.7**	45	1000	46.2
	10	70	1092	39.5	0	1091	-	**100**	**1037**	37.2	30	1077	104.6	80	1038	86.0	0	1126	-	**100**	**1037**	**31.5**	20	1055	68.2
	11	10	1222	60.3	0	1205	-	60	**1188**	**83.8**	20	1438	55.3	10	1242	109.0	0	1300	-	**65**	1199	112.5	5	1218	283.2
	12	10	1484	152.0	0	1382	-	20	1584	119.6	10	4052	15.5	0	1496	-	0	1439	-	**30**	1576	**63.0**	20	**1330**	170.4
	13	0	1713	-	0	3034	-	**5**	2128	135.8	0	7934	-	0	1687	-	0	**1595**	-	**5**	2082	288.3	0	2010	-

[Bessiere *et al.*, 2009] Christian Bessiere, George Katsirelos, Nina Narodytska, and Toby Walsh. Circuit Complexity and Decompositions of Global Constraints. In *IJCAI*, pages 412–418, 2009.

[Briers, 2002] Robert A. Briers. Incorporating connectivity into reserve selection procedures. *Biological Conservation*, 103(1):77–83, 2002.

[Conrad *et al.*, 2007] Jon Conrad, Carla P. Gomes, Willem Jan van Hoeve, Ashish Sabharwal, and Jordan Suter. Connections in Networks: Hardness of Feasibility Versus Optimality. In *CPAIOR*, pages 16–28, 2007.

[Conrad *et al.*, 2012] Jon M. Conrad, Carla P. Gomes, Willem-Jan van Hoeve, Ashish Sabharwal, and Jordan F. Suter. Wildlife corridors as a connected subgraph problem. *Journal of Environmental Economics and Management*, 63(1):1–18, 2012.

[Dilkina and Gomes, 2010] Bistra N. Dilkina and Carla P. Gomes. Solving Connected Subgraph Problems in Wildlife Conservation. In *CPAIOR*, pages 102–116, 2010.

[Fortune *et al.*, 1980] Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10:111–121, 1980.

[Hopcroft and Tarjan, 1973] John Hopcroft and Robert Tarjan. Efficient Algorithms for Graph Manipulation. *Commun. ACM*, 16(6):372–378, 1973.

[Karp, 1975] Richard M. Karp. On the computational complexity of combinatorial problems. *Networks*, 5:45–68, 1975.

[Katsirelos and Bacchus, 2005] George Katsirelos and Fahiem Bacchus. Generalized nogoods in CSPs. *Proceedings of the National Conference on Artificial Intelligence*, 20(1):390–396, 2005.

[LeBras *et al.*, 2013] Ronan LeBras, Bistra N. Dilkina, Yexiang Xue, Carla P. Gomes, Kevin S. McKelvey, Michael K. Schwartz, and Claire A. Montgomery. Robust Network Design For Multispecies Conservation. In *AAAI*, pages 1305–1312, 2013.

[Lynch, 1975] James F. Lynch. The Equivalence of Theorem Proving and the Interconnection Problem. *SIGDA Newsl.*, 5(3):31–36, 1975.

[Marques-Silva *et al.*, 2009] Joao Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. *SAT Handbook*, pages 131–154, 2009.

[Polasky *et al.*, 2005] Stephen Polasky, Erik Nelson, Eric Lonsdorf, Paul Fackler, and Anthony Starfield. Conserving species in a working landscape: land use with biological and economic objectives. *Ecological applications*, 15(4):1387–1401, 2005.

[Polasky *et al.*, 2014] Stephen Polasky, David J. Lewis, Andrew J. Plantinga, and Erik Nelson. Implementing the optimal provision of ecosystem services. *Proceedings of the National Academy of Sciences*, 111(17):6248–6253, 2014.

[Reed *et al.*, 1991] Bruce A. Reed, Neil Robertson, Alexander Schrijver, and Paul D. Seymour. Finding disjoint trees in planar graphs in linear time. In *Graph Structure Theory, Proceedings of a AMS-IMS-SIAM Joint Summer Research Conference on Graph Minors held June 22 to July 5, 1991, at the University of Washington, Seattle*, pages 295–301, 1991.