# A Connectivity Constraint using Bridges

**Patrick Prosser**[1] and **Chris Unsworth**[2]

## 1 Introduction

We present a specialised constraint for enforcing graph connectivity. It is assumed that we have a square symmetrical array $A$ of 0/1 constrained integer variables representing potential undirected edges in a simple graph, such that variable $A[u, v]$ corresponds to the undirected edge $(u, v)$. A search process is then at liberty to select and reject edges. The connectivity constraint ensures that no *critical* edge may be deleted from $A$, i.e. an edge that if deleted disconnects the graph. This connectivity constraint might then be used when modelling network design problems, modelling molecular structures, problems in bioinformatics [4], or graph problems where connectivity is an essential property [3, 1].

The constraint's internals are founded on the depth first search (dfs) process. The dfs process generates two sets of edges: the set of tree edges $T$, and the set of back edges $B$. The connectivity constraint associates a counter $nc[u, v]$, initialised to zero, with each tree edge. On creation of a back edge $(t, w)$ we increment the counters on the tree edges spanned by that back edge, i.e. we increment the set of counters $\{nc[u, v] \mid (u, v) \in path(t, w)\}$ where $path(t, w)$ delivers the set of edges that make up the path from $t$ to $w$ in $T$. A counter is incremented to indicate that there is at least one more cycle in the graph involving edge $(u, v)$, i.e. $nc[u, v]$ is a count of the number of cycles passing through edge $(u, v)$ using edges in $T$ and $B$. If on termination of the dfs process any counter, say $nc[v, w]$, equals zero then that edge $(v, w)$ is a bridge (also known as an isthmus or cut-edge) and must be forced. Consequently the value 0 is removed from the domain of variable $A[u, v]$.

## 2 Depth First Search with Tree and Back Edges

We assume we have a graph $G$ with vertices $V$ and edges $E$. The dfs algorithm [2] will produce two sets of edges: $T$ the tree edges, and $B$ the back edges. A vertex may be given one of three colours: white, grey, or black. Initially all vertices are white, signifying that they have not been visited. A vertex $v$ is coloured grey when it has first been visited, i.e. a call to dfs($v$) is made. On completing the call to dfs($v$) the vertex is coloured black. Therefore a vertex starts white, may then turn grey, and eventually black. In the dfs algorithm, given below, two types of edges may be created.

(a) Edge $(v, w)$ is a tree edge when $v$ is adjacent to $w$ in $G$ and $v$ is grey and $w$ is white. In the algorithm the call treeEdge(v,w) adds the edge $(v, w)$ to $T$. A tree edge is created as dfs expands forwards to a new descendant.

(b) A back edge $(v, w)$ is created when $w$ is adjacent to $v$, $w$ has already been visited by dfs (i.e. $w$ is grey), the processing of $v$ is not yet complete (i.e. $v$ is grey), and $(w, v)$ is not already a tree edge. The call backEdge(v,w) does nothing if $(w, v) \in T$, otherwise the edge $(v, w)$ is added to $B$ and the set of counters $\{nc[x, y] \mid (x, y) \in path(v, w)\}$ are incremented, i.e. increment the cycle counters on the tree edges on the path from $v$ to $w$.

```
procedure dfs(v)
begin
 colour[v] := grey;
 for w in adjacent(v)
 do begin
     if colour[w] = white
     then begin
          treeEdge(v,w)
          dfs(w)
          end
     else if colour[w] = grey
          then backEdge(v,w)
    end
 colour[v] := black;
end
```

## 3 The Methods of the Constraint

We now present the methods that act upon the constraint. First we describe the actions that take place when the constraint is initially added to a model. Then we describe the methods that are performed when an edge is rejected by the search process. Note that no action need be taken when an edge is selected by the search process as this cannot result in the graph becoming disconnected.

### 3.1 On Awakening

When the constraint is initially awoken (i.e. added to the model) a dfs is performed from an arbitrary vertex. As noted above we associate cycle counters with each tree edge, i.e $nc[u, v]$ is a count of the number of cycles through the edge $(u, v)$ resulting from back edges and tree edges that span or include edge $(u, v)$. These counters are initially zero and are incremented when spanned by a back edge. On termination of dfs, any cycle counter $nc[u, v]$ equal to zero indicates that edge $(u, v)$ is a bridge in $G$ and that edge must be selected otherwise $G$ will be disconnected. In addition, if on completion of the call to dfs any vertex is white then that vertex is isolated and the graph cannot be connected; consequently we can raise an exception.

### 3.2 On Deletion of a Back Edge

If an edge $(u, v)$ is deleted from the graph and $(u, v)$ is a back edge then we decrement the set of counters $\{nc[x, y]|(x, y) \in path(u, v)\}$, i.e. decrement all cycle counters on the path from $u$ to $v$ in $T$. If any counter reaches zero then the corresponding edge is a bridge and must be forced.

[1] Department of Computing Science,University of Glasgow, Scotland, pat@dcs.gla.ac.uk

[2] Department of Computing Science,University of Glasgow, Scotland, chrisu@dcs.gla.ac.uk

### 3.3 On Deletion of a Tree Edge

If a tree edge $(u, v)$ is deleted a new subtree must be produced and one of the back edges spanning $(u, v)$ will become a tree edge. Such a candidate back edge must exist otherwise $(u, v)$ will have been a bridge and that bridge will have been detected and its selection already forced. The following actions (also shown pictorially in Figure 1) need to be performed when tree edge$(u, v)$ is deleted where we assume that $u$ is the parent of $v$:

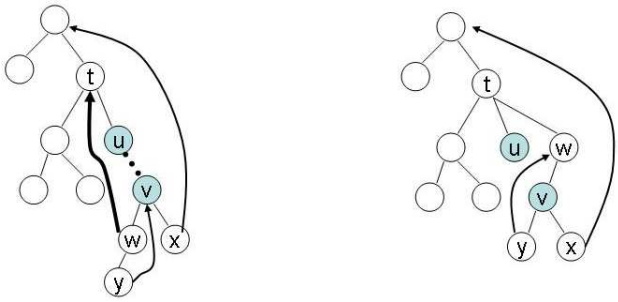


**Figure 1.** Tree edge (u,v) is deleted on the left and repaired on the right.

1. Let $V'$ be the set of vertices in the subtree rooted on $v$ (and includes $v$), $T'$ the set of tree edges in that subtree, and $B'$ the set of back edges $\{(t, w) \mid (t, w) \in B \wedge w \in V'\}$.
2. Decrement the *multi-set* of cycle counters $\{\{nc[x, y] \mid (x, y) \in path(t, w) \wedge (t, w) \in B'\}\}$, where $path(t, w)$ delivers the set of edges on the path from $t$ to $w$ in $T$.
3. Find the back edge $(t, w) \in B'$ where $depth(t) \leq depth(u) < depth(v) < depth(w)$ and $depth(t)$ is a maximum. (The back edge (t,w) is shown on the left of Figure 1).
4. Colour the vertices in the set $V'$ white (i.e. mark them as not visited by dfs).
5. Remove the tree edges $T'$ and back edges $B'$, i.e $T = T \setminus T'$ and $B = B \setminus B'$.
6. Colour grey the set of vertices on the path from $t$ to the root of T. Note, that if this was not done then no new back edges could be produced involving ancestors of $t$ in step 7 below.
7. Add new tree edge $(t, w)$ to $T$, i.e. what was back edge $(t, w)$ becomes a tree edge. Now make a call to dfs($w$). (The repaired subtree is shown on the right of Figure 1).
8. Colour black the set of vertices on the path from $t$ to the root of $T$. This needs to be done to prevent forward edges being produced by subsequent calls to dfs
9. If any cycle counter $nc[x, y]$, where $(x, y) \in T$, is zero then the edge is a bridge and must be selected.

Note that in step 3 we must select the deepest spanning back edge otherwise a cross edge may be produced in step 7 and the cycle counters would be corrupted, and that such a back edge must exist. On the termination of step 7 there will be no white vertices. This could only happen if edge $(u, v)$ was a bridge, and that would be a contradiction.

## 4 Complexity

The complexity of the algorithm is $O(n^3)$ for a graph with $n$ vertices. The principal activity of the algorithm is the number of times the cycle counters are incremented. The worst case graph would be the clique $K_n$. A clique would have a dfs tree with back edges as shown in Figure 2, using $K_6$ as an example. As can be seen, the vertices of the graph have been linearised, and the remaining edges in the clique

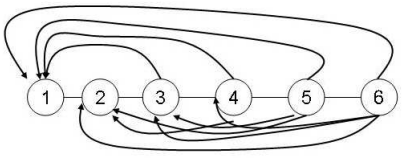


**Figure 2.** A dfs tree (straight lines) for $K_6$ with back edges (curved lines). The sum of the number of times edges participate in cycles is $\frac{n^3-7n}{6} + 1$

become back edges. The tree has $n - 1$ tree edges, and therefore $n-1$ cycle counters. A tree edge emanating from a vertex at position $i$ (where the first position is $i = 1$ and the last $i = n - 1$) will be involved in at most $i(n-i)-1$ cycles. That is, for a vertex at depth $i$ there will be $n - i$ vertices below it. Each one of these $n - i$ vertices can then have back edges to each of the first $i$ vertices. However, the vertex in position $i + 1$ cannot have a back edge to the vertex in position $i$, otherwise we have a cycle that involves only 2 vertices. Therefore we must remove 1 from our calculation. Consequently the sum of the cycle counters on the dfs tree $T$ for the clique $K_n$ will be as follows:

$$\sum_{i=1}^{n-1}[i(n-i)-1] = \frac{n^3-7n}{6} + 1$$

We should expect that the performance of this can be improved by taking a lazy approach, possibly based on [5].

## 5 Data Structures and Potential Enhancements

A number of reversible data structures are required to realise the above. Since we need to traverse a subtree in order to delete tree and back edges we associate with each vertex a boolean set of length $n$, representing the immediate children of that vertex. We also associate with a vertex a boolean set of length $n$ representing the back edges from a vertex. Each vertex also has a parent attribute so that we can traverse from a vertex upwards towards some other vertex downdating the cycle counters. We also associate a depth with a vertex so that we can compare back edges. All of the above additional space is of order $O(n^2)$ where $n$ is the number of vertices in $G$.

The constraint can be enhanced to deal with multi-edges. Assuming that an array variable $A[u, v]$ can have a value greater than 1, if the constraint forces the edge $(u, v)$ rather than setting $A[u, v]$ to 1 we remove the value 0. The efficiency of step 2 in 3.3 can be improved by setting counters in the tree edges rooted on $v$ to zero, and decrementing counters on tree edges above $u$.

## REFERENCES


[1] Ken Brown, Patrick Prosser, J. Christopher Beck, and Christine Wu, 'Exploring the use of constraint programming for enforcing connectivity during graph generation', in *The 5th workshop on modelling and solving problems with constraints (held at IJCAI05)*, (2005).

[2] T.H. Corman, C.E. Leirson, and R.L. Rivest, *Introduction to Algorithms*, Sept 2001.

[3] Gregorie Dooms, Yves Deville, and Pierre Dupont, 'CP(Graph): Introducing a Graph Computation Domain in Constraint Programming', in *CP2005 (LNCS 3709)*, (2005).

[4] Ian P. Gent, Patrick Prosser, Barbara M. Smith, and Christine Wu Wei, 'Supertree Construction with Constraint Programming', in *CP2003 (LNCS 2833)*, (2003).

[5] R. Endre Tarjan, 'A note on fi nding the bridges of a graph', *Information Processing Letters*, **2**, 160–161, (1974).



**Acknowledgements:** We would like to thank our five reviewers.