Complexity of the Cardpath Constraint
Christian Bessière

To cite this version:


HAL Id: lirmm-00106686
https://hal-lirmm.ccsd.cnrs.fr/lirmm-00106686
Submitted on 16 Oct 2006
Complexity of the cardpath constraint

Christian Bessiere

TR LIRMM 05036
(January 2005)

Abstract

The cardpath constraint enforces a constraint $C$ to be satisfied a given number of times over a sequence of variables. In this note, we give a polynomial propagation algorithm for a special case of this constraint, which in fact is the only case which was not known to be intractable.

The cardinality path constraint $[BC01]$, cardpath$(N, [X_1, \ldots, X_m], C)$, where $C$ is a constraint of arity $k < m$ and $N$ is an integer variable, ensures that $N = \sum_{i=1}^{m-k+1} C(X_{i+1}, \ldots, X_{i+k-1})$. ($C()$ returns 1 when satisfied, 0 otherwise.) In other words, we slide $C$ down the sequence $X_1, \ldots, X_m$ and ensure it holds $N$ times.

In $[BHHW04a]$ and $[BHHW04b]$, we proved that enforcing arc consistency on the constraint cardpath$(N, [X_1, \ldots, X_m], C)$ is NP-hard if $C$ has unbounded arity (even if arc consistency is polynomial on it) or if we allow repetition of variables in the sequence $X_1, \ldots, X_m$. The only remaining open question was therefore about the complexity of cardpath when $C$ has bounded arity and we don’t allow repetitions in $X_1, \ldots, X_m$. We show that it is in fact polynomial to achieve arc consistency on such a constraint. We present an algorithm for $C$ being binary, which is simpler to present. But the technique can be extended to any arity as long as it remains bounded.

The idea of the algorithm is the following. It first makes a double traversal of the sequence $[X_1, \ldots, X_m]$, one from $X_1$ to $X_m$ (lines 1 to 7) and the other from $X_m$ downto $X_1$ (lines 8 to 14). This step computes two sets of integers, $w(X_i, v)$ and $dw(X_i, v)$ for each value $(X_i, v)$. The set $w(X_i, v)$ contains all possible number of times $C$ is satisfied by a tuple on $[X_1, \ldots, X_i]$ belonging to $D(X_1) \times \cdots \times D(X_i)$, while $dw(X_i, v)$ contains all possible number of times $C$ is satisfied by a tuple on $[X_i, \ldots, X_m]$ in $D(X_i) \times \cdots \times D(X_m)$.

The second step (lines 15 to 18) makes the join of those two sets of integers, putting in the set $T(X_i, v)$ all possible number of times $C$ is satisfied by a complete tuple in $D(X_1) \times \cdots \times D(X_m)$. All values such that $T(X_i, v)$ does not intersect $D(N)$ are removed since there does not exist any tuple in $D(X_1) \times \cdots \times D(X_m)$, using $v$ for $X_i$ that satisfies $C$ a number of times allowed by $N$. 

1
Algorithm Propag-cardpath(N, [X1..Xm], C)
1. for each v in D(X1) do w(X1,v):={0}
2. for i:=2 to m do
3. for v in D(Xi) do w(Xi,v):={} 
4. for u in D(Xi-1) do
5. if C(u,v) then w(Xi,v):=w(Xi,v) U inc(w(Xi-1,u))
6. else w(Xi,v):=w(Xi,v) U w(Xi-1,u)
7. for each v in D(Xm) do dw(Xm,v):={0}
8. for i:=m-1 downto 1 do
9. for v in D(Xi) do dw(Xi,v):={} 
10. for u in D(Xi+1) do
11. if C(v,u) then dw(Xi,v):=dw(Xi,v) U inc(dw(Xi+1,u))
12. else dw(Xi,v):=dw(Xi,v) U dw(Xi+1,u)
13. for i:=1 to m do
14. for v in D(Xi) do
15. T(Xi,v):= {p+q | p in w(Xi,v), q in dw(Xi,v)}
16. if T(Xi,v) inter D(N) == {} then remove v from D(Xi)
17. T:= union(T(X1,v)), v in D(X1)
18. D(N):=D(N) inter T

Finally, D(N) is pruned from its impossible values by intersecting its domain with \( \bigcup_{v \in D(X_i)} T(X_i, v) \) for any \( X_i \) (since after line 18 we have \( \bigcup_{v \in D(X_i)} T(X_i, v) = \bigcup_{w \in D(X_j)} T(X_j, v), \forall i,j. \))

In the algorithm Propag-cardpath(N, [X1, ..., Xm], C), we use the function inc(S), which for any set \( S \) of integers returns the set \( \{p + 1 | p \in S\} \).

Theorem 1 The algorithm Propag-cardpath is a correct algorithm for achieving arc consistency on cardpath, and it runs in \( O(m^3d + m^2d^2) \) time.

Proof. (Very sketch.) If \( v \) is pruned from \( D(X_i) \), this is because \( T(X_i, v) \) doesn’t intersect \( D(N) \). \( T(X_i, v) \) contains the values that are the sum of a value from \( w(X_i, v) \) and one from \( dw(X_i, v) \), which themselves contain all possible number of times \( C \) is satisfied by a tuple from \( X_1 \) to \( X_i \) or from \( X_i \) to \( X_m \) using \( (X_i, v) \). This means that any tuple containing \( v \) for \( X_i \) cannot satisfy the constraint cardpath. Same reasoning for values pruned from \( D(N) \). Therefore, soundness.

Suppose now that \( (X_i, v) \) is not arc consistent. We cannot build a tuple containing it and satisfying cardpath. So, after the first step, \( w \) and \( dw \) don’t contain values \( p \) and \( q \) such that \( p + q \in D(N) \). So, \( v \) is pruned from \( D(X_i) \) in line 18. Same reasoning for \( N \). So, completeness.

Complexity. The algorithm Propag-cardpath is composed of a main step (lines 1–14) that traverses the variables and their values. For each pair (variable,value), the sub-loops in lines 5–7 and lines 12–14 are executed. Such sub-loops perform \( md \) operations since they traverse the domain of another variable (\( d \) possible values) and make the union of two sets of integers of size at most
$m$ (C cannot be satisfied more than $m$ times on a sequence of length $m$). So, the total complexity of lines 1–14 is $md \cdot md$. In lines 15–18, $md$ sets $T(X_i, v)$ are computed. A set $T(X_i, v)$ contains all possible sums of two elements from two sets of size $m$, so a cost in $O(m^2)$ for each $T(X_i, v)$. The cost of lines 15–18 is thus in $O(m^3d)$. Lines 19–20 are in $O(md)$. The total time complexity of Propag-cardpath is in $O(m^3d + m^2d^2)$.

Finally, let us point out that if $C$ is of bounded arity $k > 2$, we need to proceed the same as presented here, but we build the sets $w$ and $dw$ for all instantiations of size $k - 1$, thus introducing $md^k$ such sets, which is polynomial since $k$ is bounded.

References

