Foot Note 621 by Alice Miller and Patrick Prosser
Limited Circulation[1] June 7, 2001

Maximal Independent Sets of a Hypergraph

1 Introduction

Given a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, an independent set V' is a subset of V such that for any $x, y \in V'$, $(x, y) \notin V'$. The set V' is a *maximal* independent set if there is no set V'' such that V'' is an independent set and $V' \subset V''$. We can extend the notion of an independent set to a hypergraph. If we allow G to contain hyperedges, i.e. edges that contain more than two vertices, V' is an independent set if there is no edge $e \in E$ such that $vert(e) \subseteq V'$, where $vert(e)$ is the set of vertices in an edge. The problem is then, given some integer $k < |V|$, is there a maximal independent set of size k?

This problem is NP-Complete. A variant of it, GT20 in Garey and Johnson, asks if there is an independent set of size k or larger. In this footnote we address the problem where we ask for a maximal independent set of size exactly k and we show how to encode the notion of maximality into a constraint program.

2 A Constraint Encoding of Maximality

We will use the hypergraph in Figure 1 as an example, using it to demonstrate our encoding. The hyper edges are shown as *blobs* containing integers, where the integers are the vertices[2]. We have 9 vertices, V_1 to V_9, and 4 edges $\{(V_1, V_2, V_3), (V_2, V_4, V_5), (V_4, V_6), (V_3, V_7, V_8, V_9)\}$. We represent a vertex with a 0/1 variable. Variable/Vertex $V_i = 0$ means that V_i is not in the independent set, and $V_i = 1$ means V_i is in the independent set.

For each edge we have a constraint that disallows all of its vertices to be selected. For example, the constraint for the edge (V_3, V_7, V_8, V_9) is $V_3 + V_7 + V_8 + V_9 < 4$, and the constraint for edge (V_4, V_6) is $V_4 + V_6 < 2$. More generaly for an edge e with arity r (i.e. $|vert(e)| = r$) we have a constraint that the sum of the vertices in the edge is less than its arity. We might call these the *edge constraints*.

We have a global constraints that states that the sum of the vertices is equal to k, i.e. the global constraint specifies the size of the independent set.

However, we have not yet captured the notion of maximality. We might do this by doctoring the search. That is, whenever the search process fails to extend a node at depth k we have proved that the node is maximal and of size k. But this does not exploit constraint propagation, and is a half way house between enumerate and test and constraint programming.

We need constraints to specify under what conditions a vertex is in the maximal independent set, and when it is not in the maximal independent set. Looking again at Figure 1 we can see that vertex V_2 is not in the maximal independent set V' if vertices V_4 and V_5 are in V' or V_1 and V_3 are in V'. We can express this as the following constraint:

$$(V_4 + V_5 = 2) \vee (V_1 + V_3 = 2) \leftrightarrow V_2 = 0$$

where $\leftrightarrow$ is the biconditional *if and only if*. Similarly we can express when V_2 is in V', i.e. when V_4 is not in V' or V_5 is not in V' and V_1 is not in V' or V_3 is not in V'. Therefore we have the constraint

$$(V_4 + V_5 < 2) \wedge (V_1 + V_3 < 2) \leftrightarrow V_2 = 1$$

[1] Should be distributed only to those in the APES group or to those people who should obviously be allowed to see it.
[2] ... and from now on we will drop the *hyper*.

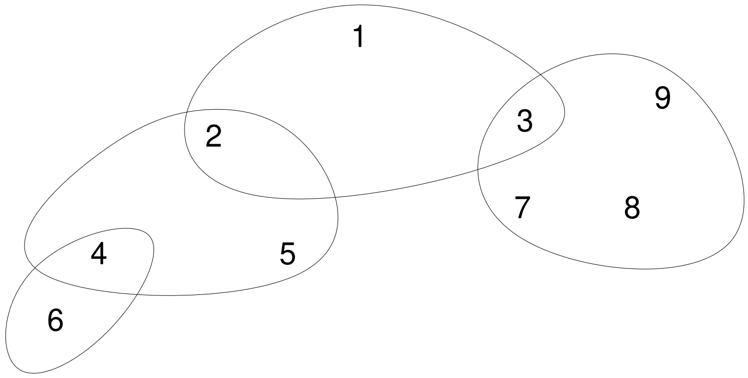

Figure 1: Our example hyper graph. Find a maximal independent set of size k. When $k = 5$ there are 3 solutions, the first Choco solution is $\{V_2, V_3, V_4, V_8, V_9\}$. When $k = 6$ there are 11 solutions, the first (Choco) being $\{V_2, V_3, V_5, V_6, V_8, V_9\}$, and when $k = 7$ there is a single solution $\{V_1, V_2, V_5, V_6, V_7, V_8, V_9\}$. There are no maximal independent sets for any other values of k.

We can simplify this. The biconditional $p \leftrightarrow q$ is logically equivalent to $(p \wedge q) \vee (\neg p \wedge \neg q)$. Consequently we can express our first constraint as follows:

$$(((V_4 + V_5 = 2) \vee (V_1 + V_3 = 2)) \wedge V_2 = 0) \vee ((V_4 + V_5 < 2) \wedge (V_1 + V_3 < 2) \wedge V_2 = 1)$$

The above constraint specifies when the vertex V_2 must not be in the maximal independent set and when it must be in the maximal independent set. Therefore we need only one such constraint for each variable. More generally, for a variable V_i we have the following constraints

$$\begin{aligned} V_i = 1 &\leftrightarrow \bigwedge_{e \in edges(V_i)} [arity(e) - 1 > \sum_{V_j \in vert(e) - V_i} V_j] \\ V_i = 0 &\leftrightarrow \bigvee_{e \in edges(V_i)} [arity(e) - 1 = \sum_{V_j \in vert(e) - V_i} V_j] \end{aligned}$$

where $edges(V_i)$ is the set of edges involving vertex V_i and $vert(e) - V_i$ is the set of vertices in the edge e excluding vertex V_i.

3 A Choco example

Below we have a Choco encoding of the problem in Figure 1.

```
[maximality(k:integer) : Problem
 -> let n  := 9,
        pb := makeProblem("maximality",n),
        v  := list{makeIntVar(pb,"V" /+ string!(i),{0,1}) | i in (1 .. n)}
    in (post(pb,v[1] + v[2] + v[3] < 3),
        post(pb,v[2] + v[4] + v[5] < 3),
        post(pb,v[4] + v[6] < 2),
        post(pb,v[3] + v[7] + v[8] + v[9] < 4),
        // v[1]
```

```
        post(pb,((v[1] == 0) and (v[2] + v[3] == 2)) or
                ((v[1] == 1) and (v[2] + v[3] < 2))),
        // v[2]
        post(pb,((v[2] == 0) and ((v[4] + v[5] == 2) or (v[1] + v[3] == 2)) or
                ((v[2] == 1) and ((v[4] + v[5] < 2) and (v[1] + v[3] < 2))))),
        // v[3]
        post(pb,((v[3] == 0) and ((v[1] + v[2] == 2) or (v[7] + v[8] + v[9] == 3))) or
                ((v[3] == 1) and ((v[1] + v[2] < 2) and (v[7] + v[8] + v[9] < 3)))),
        // v[4]
        post(pb,((v[4] == 0) and ((v[6] == 1) or (v[2] + v[5] == 2))) or
                ((v[4] == 1) and ((v[6] == 0) and (v[2] + v[5] < 2)))),
        // v[5]
        post(pb,((v[5] == 0) and (v[2] + v[4] == 2)) or
                ((v[5] == 1) and (v[2] + v[4] < 2))),
        // v[6]
        post(pb,((v[6] == 0) and (v[4] == 1)) or
                ((v[6] == 1) and (v[4] == 0))),
        // v[7]
        post(pb,((v[7] == 0) and (v[3] + v[8] + v[9] == 3)) or
                ((v[7] == 1) and (v[3] + v[8] + v[9] < 3))),
        // v[8]
        post(pb,((v[8] == 0) and (v[3] + v[7] + v[9] == 3)) or
                ((v[8] == 1) and (v[3] + v[7] + v[9] < 3))),
        // v[9]
        post(pb,((v[9] == 0) and (v[3] + v[7] + v[8] == 3)) or
                ((v[9] == 1) and (v[3] + v[7] + v[8] < 3))),
        // the size of the maximal independent set
        post(pb,occur(1,v) == k),
        pb)]
//
// Generate a representation, pb, of the Figure 1 problem
// The independent set is to be of size k
//
```

Below we have a trace of an execution of the above Choco code, to find all maximal independent sets of size 7 for the hypergraph in Figure 1.

```
Script started on Fri Jun  1 09:54:48 2001
>> claire -s 3 3 -m choco
increasing memory size by 2^3 and 2^3
-- CLAIRE run-time library v 2.5.44 [os: ntv, C++:gcc ] --
-- CLAIRE interpreter - Copyright (C) 1994-00 Y. Caseau (see about())
Choco version 0.38, Copyright (C) 1999-2000 F. Laburthe
Choco comes with ABSOLUTELY NO WARRANTY; for details read licence.txt
This is free software, and you are welcome to redistribute it
under certain conditions; read licence.txt for details.
choco> load("max")
eval[0]> ---- [load CLAIRE file: max]
true
choco> p:Problem := maximality(7)
eval[1]> p
```

```
choco> solve(p,true)
eval[2]>  !! branch 1: V1:[2]{0, 1} is 0
 !! branch 2: V4:[2]{0, 1} is 0
 !! branch 1: V4:[2]{0, 1} removeVal 0
 !! branch 0: V1:[2]{0, 1} removeVal 0
 !! branch 1: V1:1 is 1
 !! branch 2: V2:[2]{0, 1} is 0
 !! branch 3: V3:[2]{0, 1} is 0
 !! branch 2: V3:[2]{0, 1} removeVal 0
 !! branch 3: V3:1 is 1
 !! branch 4: V4:[2]{0, 1} is 0
 !! branch 3: V4:[2]{0, 1} removeVal 0
 !! branch 2: V3:1 removeVal 1
 !! branch 1: V2:[2]{0, 1} removeVal 0
 !! branch 2: V2:1 is 1
 !! branch 3: V4:[2]{0, 1} is 0
V1:1
V2:1
V3:0
V4:0
V5:1
V6:1
V7:1
V8:1
V9:1
... solution with 6 nodes, 6 backtracks
 !! branch 2: V4:[2]{0, 1} removeVal 0
 !! branch 1: V2:1 removeVal 1
 !! branch 0: V1:1 removeVal 1
solve => 1 solution in 9 bk. and 50 ms.
true
choco> q
eval[3]> Bye.
>> exit
exit

Script done on Fri Jun  1 09:55:43 2001
```