DIMACS Technical Report 95-14
June 1995

RESOLUTION SEARCH

by

Vašek Chvátal[1]
Dept. of Computer Science
Rutgers University
New Brumswick, New Jersey 08903

[1]Permanent Member. Supported by ONR grant N00014–92–J–1375

DIMACS is a cooperative project of Rutgers University, Princeton University, AT&T Bell Laboratories and Bellcore.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

Branch-and-bound is one of the most popular ways of solving difficult problems such as integer and mixed-integer linear programming problems; when the branching is done on zero-one valued variables, the resulting variant of branch-and bound is called implicit enumeration. Efficiency of branch-and-bound and implicit enumeration algorithms depends heavily on the branching strategy used to select the next variable to branch on and its value. We propose an alternative to implicit enumeration. Our algorithm, which we call resolution search, seems to suffer less from inappropriate branching strategies than implicit enumeration does.

A search problem

If $(u_1, u_2, \ldots, u_n)$ and $(v_1, v_2, \ldots, v_n)$ are elements of $\{0, 1, \star\}^n$ such that

$$v_j = u_j \text{ whenever } u_j \neq \star$$

then we shall call $(v_1, v_2, \ldots, v_n)$ an *extension* of $(u_1, u_2, \ldots, u_n)$ and we shall write

$$(u_1, u_2, \ldots, u_n) \sqsubseteq (v_1, v_2, \ldots, v_n).$$

Trivially, $\sqsubseteq$ is a partial order on $\{0, 1, \star\}^n$. A function f from $\{0, 1, \star\}^n$ into an ordered set will be called *nondecreasing* if, and only if, $f(u) \leq f(v)$ whenever $u \sqsubseteq v$. We shall be concerned with the following search problem:

$$\begin{array}{l} \text{Given a nondecreasing function } \texttt{oracle}, \\ \text{find a vector } v \text{ in } \{0,1\}^n \text{ that minimizes } \texttt{oracle}(v). \end{array} \tag{1}$$

Trivially, 2^n calls of `oracle` suffice to solve this search problem and 2^n calls of `oracle` are necessary in the worst case. However, one can often do better; that is the subject of the present paper.

A special case of (1) is the mixed zero-one linear programming problem

$$\begin{array}{lrcll} \text{minimize} & \sum_{j=1}^{N} c_j x_j & & & \\ \text{subject to} & \sum_{j=1}^{N} a_{ij} x_j & \geq & b_i & (i = 1, 2, \ldots, m), \\ & x_j & = & 0 \text{ or } 1 & (j = 1, 2, \ldots, n). \end{array}$$

Here, one appropriate `oracle` is defined by letting $\texttt{oracle}(u_1, u_2, \ldots, u_n)$ be the optimum value of the linear programming problem

$$\left.\begin{array}{lrcll} \text{minimize} & \sum_{j=1}^{N} c_j x_j & & & \\ \text{subject to} & \sum_{j=1}^{N} a_{ij} x_j & \geq & b_i & (i = 1, 2, \ldots, m), \\ & 0 \leq x_j & \leq & 1 & (j = 1, 2, \ldots, n), \\ & x_j & = & u_j & \text{whenever } u_j \neq \star \end{array}\right\} \tag{2}$$

if the optimum value exists and otherwise setting

$$\texttt{oracle}(u_1, u_2, \ldots, u_n) = \begin{cases} +\infty & \text{if (2) is infeasible,} \\ -\infty & \text{if (2) is unbounded.} \end{cases}$$

Implicit enumeration

A natural strategy for coping with the search problem (1) goes as follows. If we know a vector v in $\{0,1\}$ such that $\texttt{oracle}(v) = \texttt{oracle}(\star,\star,\ldots,\star)$ then we return v as a solution of (1); else we partition the search space, $\{0,1\}^n$, into two subsets, S_1 and S_2, and solve the two subproblems,

find a vector v in S_1 that minimizes $\texttt{oracle}(v)$,

find a vector v in S_2 that minimizes $\texttt{oracle}(v)$

separately. To be able to use the same strategy in coping with the two subproblems, we choose a subscript j, choose a value c in $\{0,1\}$, and set

$$\begin{aligned} S_1 &= \{(x_1,x_2,\ldots,x_n) : (x_1,x_2,\ldots,x_n) \in \{0,1\}^n, x_j = c\}, \\ S_2 &= \{(x_1,x_2,\ldots,x_n) : (x_1,x_2,\ldots,x_n) \in \{0,1\}^n, x_j = 1-c\}. \end{aligned}$$

A recursive implementation of this strategy is the function

implicit_enumeration(u, record, v)

defined in Table 1. This function, given a vector u in $\{0,1,\star\}^n$, a vector v in $\{0,1\}^n$, and the value record of oracle(v), either

replaces v by a vector x in $\{0,1\}^n$ that
minimizes oracle(x) subject to $u \sqsubseteq x$ and oracle(x)$<$ record,
in which case the function returns the value of oracle(x),

or

returns record to indicate that no such vector x exists;

search problem (1) can be solved by calling

implicit_enumeration(($\star,\star,\ldots,\star$), oracle(v), v)

with an arbitrary v in $\{0,1\}^n$.

```
implicit_enumeration(u, record, v)
{
   bound=oracle(u);

   if (record ≤ bound) return record;
   if (there is no subscript j with u_j = ⋆){
      v = u;
      return bound;
   }

   choose a subscript j with u_j = ⋆ and a value c in {0,1};
   u_j = c;
   try = implicit_enumeration(u, record, v);
   if (try = bound){
      u_j = ⋆;
      return bound;
   }
   if (try < record) record=try;

   u_j = 1 − c;
   try = implicit_enumeration(u, record, v);
   if (try < record) record=try;

   u_j = ⋆;
   return record;
}
```

Table 1: Definition of implicit enumeration

Each recursive call of `implicit_enumeration` may involve a choice of a subscript j with $u_j = \star$ and a choice of a value c in $\{0, 1\}$; algorithms for making these choices are called *branching strategies*; their outcomes may influence strongly the resulting total number of calls of `oracle`. For illustration, let us consider the problem

$$\begin{array}{lrcl} \text{minimize} & x_1 + x_2 + x_3 & & \\ \text{subject to} & x_1 + x_2 + x_3 + 2x_4 & \geq & 2, \\ & x_1 + x_2 + x_3 - 2x_4 & \geq & 0, \\ & x_1, x_2, x_3, x_4 & = & 0 \text{ or } 1 \end{array} \qquad (3)$$

with the `oracle` defined through (2); having initialized $v = (0, 0, 0, 0)$, we call

`implicit_enumeration((⋆,⋆,...,⋆), +∞, v).`

The primitive branching strategy "let j be the *smallest* subscript with $u_j = \star$ and set $c = 0$" leads to twenty-one calls of `oracle`:

```
v = (0,0,0,0);
calling implicit_enumeration((⋆,⋆,⋆,⋆),+∞,v)              oracle((⋆,⋆,⋆,⋆)) = 1
  calling implicit_enumeration((0,⋆,⋆,⋆),+∞,v)            oracle((0,⋆,⋆,⋆)) = 1
    calling implicit_enumeration((0,0,⋆,⋆),+∞,v)          oracle((0,0,⋆,⋆)) = 1
      calling implicit_enumeration((0,0,0,⋆),+∞,v)        oracle((0,0,0,⋆)) = +∞
        calling implicit_enumeration((0,0,0,0),+∞,v)      oracle((0,0,0,0)) = +∞
        implicit_enumeration((0,0,0,0),+∞,v) returns +∞
      implicit_enumeration((0,0,0,⋆),+∞,v) returns +∞
      calling implicit_enumeration((0,0,1,⋆),+∞,v)        oracle((0,0,1,⋆)) = 1
        calling implicit_enumeration((0,0,1,0),+∞,v)      oracle((0,0,1,0)) = +∞
        implicit_enumeration((0,0,1,0),+∞,v) returns +∞
        calling implicit_enumeration((0,0,1,1),+∞,v)      oracle((0,0,1,1)) = +∞
        implicit_enumeration((0,0,1,1),+∞,v) returns +∞
      implicit_enumeration((0,0,1,⋆),+∞,v) returns +∞
    implicit_enumeration((0,0,⋆,⋆),+∞,v) returns +∞
    calling implicit_enumeration((0,1,⋆,⋆),+∞,v)          oracle((0,1,⋆,⋆)) = 1
      calling implicit_enumeration((0,1,0,⋆),+∞,v)        oracle((0,1,0,⋆)) = 1
        calling implicit_enumeration((0,1,0,0),+∞,v)      oracle((0,1,0,0)) = +∞
        implicit_enumeration((0,1,0,0),+∞,v) returns +∞
        calling implicit_enumeration((0,1,0,1),+∞,v)      oracle((0,1,0,1)) = +∞
        implicit_enumeration((0,1,0,1),+∞,v) returns +∞
      implicit_enumeration((0,1,0,⋆),+∞,v) returns +∞
      calling implicit_enumeration((0,1,1,⋆),+∞,v)        oracle((0,1,1,⋆)) = 2
        calling implicit_enumeration((0,1,1,0),+∞,v)      oracle((0,1,1,0)) = 2
        v = (0,1,1,0);
        implicit_enumeration((0,1,1,0),+∞,v) returns 2
      implicit_enumeration((0,1,1,⋆),+∞,v) returns 2
    implicit_enumeration((0,1,⋆,⋆),+∞,v) returns 2
  implicit_enumeration((0,⋆,⋆,⋆),+∞,v) returns 2
  calling implicit_enumeration((1,⋆,⋆,⋆),2,v)             oracle((1,⋆,⋆,⋆)) = 1
    calling implicit_enumeration((1,0,⋆,⋆),2,v)           oracle((1,0,⋆,⋆)) = 1
      calling implicit_enumeration((1,0,0,⋆),2,v)         oracle((1,0,0,⋆)) = 1
        calling implicit_enumeration((1,0,0,0),2,v)       oracle((1,0,0,0)) = 2
        implicit_enumeration((1,0,0,0),2,v) returns 2
        calling implicit_enumeration((1,0,0,1),2,v)       oracle((1,0,0,1)) = 2
        implicit_enumeration((1,0,0,1),2,v) returns 2
      implicit_enumeration((1,0,0,⋆),2,v) returns 2
      calling implicit_enumeration((1,0,1,⋆),2,v)         oracle((1,0,1,⋆)) = 2
      implicit_enumeration((1,0,1,⋆),2,v) returns 2
    implicit_enumeration((1,0,⋆,⋆),2,v) returns 2
    calling implicit_enumeration((1,1,⋆,⋆),2,v)           oracle((1,1,⋆,⋆)) = 2
    implicit_enumeration((1,1,⋆,⋆),2,v) returns 2
  implicit_enumeration((1,⋆,⋆,⋆),2,v) returns 2
implicit_enumeration((⋆,⋆,⋆,⋆),+∞,v) returns 2
```

With another branching strategy, six calls of `oracle` suffice:

$v = (0, 0, 0, 0)$;
calling `implicit_enumeration`$((\star, \star, \star, \star), +\infty, v)$ `oracle`$((\star, \star, \star, \star)) = 1$
  calling `implicit_enumeration`$((\star, \star, \star, 0), +\infty, v)$ `oracle`$((\star, \star, \star, 0)) = 2$
    calling `implicit_enumeration`$((\star, \star, 0, 0), +\infty, v)$ `oracle`$((\star, \star, 0, 0)) = 2$
      calling `implicit_enumeration`$((\star, 1, 0, 0), +\infty, v)$ `oracle`$((\star, 1, 0, 0)) = 2$
        calling `implicit_enumeration`$((1, 1, 0, 0), +\infty, v)$ `oracle`$((1, 1, 0, 0)) = 2$
        $v = (1, 1, 0, 0)$;
        `implicit_enumeration`$((1, 1, 0, 0), +\infty, v)$ returns 2
      `implicit_enumeration`$((\star, 1, 0, 0), +\infty, v)$ returns 2
    `implicit_enumeration`$((\star, \star, 0, 0), +\infty, v)$ returns 2
  `implicit_enumeration`$((\star, \star, \star, 0), +\infty, v)$ returns 2
  calling `implicit_enumeration`$((\star, \star, \star, 1), 2, v)$ `oracle`$((\star, \star, \star, 1)) = 2$
  `implicit_enumeration`$((\star, \star, \star, 1), 2, v)$ returns 2
`implicit_enumeration`$((\star, \star, \star, \star), +\infty, v)$ returns 2

In `implicit_enumeration`, each choice of j and c means a commitment to solving a restricted search problem before moving on; it is the irrevocable nature of these commitments that makes branching strategies so important. We are going to propose an alternative to `implicit_enumeration`, where the commitments are tentative rather than binding, and so the penalties for using bad branching strategies are less severe.

Resolution search: definition

We shall refer to elements of $\{0,1,\star\}^n$ as *clauses*. Customarily, clauses are represented as subsets of a set of symbols $x_1, x_2, \ldots, x_n, \overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n$ that are called *literals*:

$(u_1, u_2, \ldots, u_n)$ is represented as $\{x_j : u_j = 0\} \cup \{\overline{x}_j : u_j = 1\}$.

We shall follow this custom and we shall follow the convention of recording each clause as a string of its elements. For instance, $(\star, 0, 1, \star, 0, \star, \star, \ldots, \star)$ will be represented as $\{x_2, \overline{x}_3, x_5\}$ and recorded as $x_2\overline{x}_3x_5$; in particular, $(\star, \star, \star, \ldots, \star)$ will be represented by the *empty clause* that is denoted $\emptyset$. The *complement*, $\overline{w}$, of a literal w is defined by

$$\overline{w} = \begin{cases} \overline{x}_j & \text{if } w = x_j, \\ x_j & \text{if } w = \overline{x}_j. \end{cases}$$

Clauses C_1, C_2 are said to *clash* if there is exactly one literal, w, such that $w \in C_1$, $\overline{w} \in C_2$; their *resolvent*, which we denote $C_1 \bigtriangledown C_2$, is the clause $(C_1 - \{w\}) \cup (C_2 - \{\overline{w}\})$.

When r is a value in the range of `oracle`, we shall say that a clause C is *r-forcing* to mean that

every x in $\{0,1\}^n$ with $C \sqsubseteq x$ satisfies `oracle`$(x) \geq r$.

Trivially, every C is r-forcing with $r =$`oracle`(C); note also that the resolvent of any two clashing r-forcing clauses is r-forcing.

One way of tackling the search problem is to maintain a vector v in $\{0,1\}^n$ and a family $\mathcal{F}$ of clauses subject to the following rules:

- if $x \in \{0,1\}^n$ and `oracle`(x) $<$ `oracle`(v) then v can be replaced by x;
- if S is a clause such that `oracle`$(S) \geq$ `oracle`(v) then S can be added to $\mathcal{F}$;
- if R, C are clashing clauses in $\mathcal{F}$ then $R \bigtriangledown C$ can be added to $\mathcal{F}$;
- if C is a clause in $\mathcal{F}$ then C can be deleted from $\mathcal{F}$.

It is easy to see that an initially empty $\mathcal{F}$ updated by these rules includes only r-forcing clauses with $r =$ `oracle`(v); in particular, as soon as the empty clause is added to $\mathcal{F}$, vector v can be returned as a solution of the search problem.

The algorithm we are about to present follows this general scheme. The family $\mathcal{F}$ that it maintains has a special structure: its members are enumerated as $C_1, C_2, \ldots, C_m$ and

> there are literals $w_1, w_2, \ldots, w_m$ such that
> - $w_i \in C_j$ if and only if $j = i$,
> - if $\overline{w}_i \in C_j$ then $j > i$,
> - if $w \in C_i$ and $\overline{w} \in C_j$, then $w = w_i$ or $w = w_j$.

We shall refer to such families as *path-like*.

The algorithm relies on a function `obstacle` that, given any u in $\{0,1,\star\}^n$ and any value `record`, replaces u with a u^* such that $u \sqsubseteq u^*$ and

$$\texttt{oracle}(u^*) \geq \texttt{record} \quad \text{or else} \quad u^* \in \{0,1\}^n,$$

constructs a clause S such that $S \sqsubseteq u^*$ and

$$\begin{array}{lcll} \texttt{oracle}(S) & \geq & \texttt{record} & \text{if } \texttt{oracle}(u^*) \geq \texttt{record}, \\ \texttt{oracle}(S) & = & \texttt{oracle}(u^*) & \text{if } \texttt{oracle}(u^*) < \texttt{record}, \end{array}$$

and returns the value of $\texttt{oracle}(u^*)$. Any such function $\texttt{obstacle}(u,\texttt{record},S)$ will do; for the moment, let us think of $\texttt{obstacle}(u,\texttt{record},S)$ as a black box.

Each iteration of our algorithm begins with a call of $\texttt{obstacle}(u,\texttt{record},S)$ such that u is the vector represented by

$$\left(\bigcup_{i=1}^{m}(C_i - \{w_i\})\right) \cup \{\overline{w}_1, \overline{w}_2, \ldots, \overline{w}_m\}.$$

If the value returned by `obstacle` is smaller than `record` then u has been replaced by its extension u^* in $\{0,1\}^n$ such that $\texttt{oracle}(u^*)\texttt{<record}$, in which case we set $v = u^*$ and $\texttt{record=oracle}(u^*)$; regardless of the value returned by `obstacle`, the clause S constructed by `obstacle` is used to update the family $C_1, C_2, \ldots, C_m$.

As for the update of $C_1, C_2, \ldots, C_m$, there are two cases to consider. If S includes a literal w such that $w \notin C_1 \cup C_2 \cup \ldots \cup C_m$ and $\overline{w} \notin C_1 \cup C_2 \cup \ldots \cup C_m$ then we simply set

$$m = m + 1,\ C_m = S,\ w_m = w.$$

In the other case, we have

$$S \subseteq \left(\bigcup_{i=1}^{m}(C_i - \{w_i\})\right) \cup \{\overline{w}_1, \overline{w}_2, \ldots, \overline{w}_m\}.$$

Here, we first produce a clause R by

$$\begin{array}{l} R = S; \\ \texttt{for}\ \ (i = m, m-1, \ldots, 3, 2, 1) \\ \quad\ \texttt{if}\ \ (\overline{w}_i \in R)\ \ R = R \bigtriangledown C_i\,; \end{array}$$

note that, after each execution of the `for` loop, we have

$$R \subseteq \left(\bigcup_{j=1}^{m}(C_j - \{w_j\})\right) \cup \{\overline{w}_1, \overline{w}_2, \ldots, \overline{w}_{i-1}\},$$

and so $R \subseteq \bigcup_{i=1}^{m}(C_i - \{w_i\})$ in the end. If $R = \emptyset$ then we return v as a solution of the search problem; else we

- find the smallest k such that $R \subseteq \bigcup_{i=1}^{k}(C_i - \{w_i\})$,
- replace C_k by R,
- choose a literal w in $R - (C_1 \cup C_2 \cup \ldots \cup C_{k-1})$,
- remove from $C_{k+1}, C_{k+2}, \ldots, C_m$ every clause that includes w.

Formally, this sequence of operations can be described as

$M = m$, $m =$ `smallest` i `such that` $R \subseteq C_1 \cup C_2 \cup \ldots \cup C_i$;
`choose a literal` w `in` $R - (C_1 \cup C_2 \cup \ldots \cup C_{k-1})$;
$C_m = R$, $w_m = w$;
`for` $(i = m+1, m+2, \ldots, M)$
    `if` $(w \notin C_i)$ $m = m+1$, $C_m = C_i$, $w_m = w_i$;

Example 1:

$C_1 = x_2x_3 \qquad (w_1 = x_2)$,
$C_2 = x_3x_6 \qquad (w_2 = x_6)$,
$C_3 = \overline{x}_1\overline{x}_5\overline{x}_9 \qquad (w_3 = \overline{x}_1)$,
$C_4 = \overline{x}_2\overline{x}_5\overline{x}_8 \qquad (w_4 = \overline{x}_8)$,
$C_5 = x_3\overline{x}_5x_7\overline{x}_9 \qquad (w_5 = x_7)$,
$C_6 = \overline{x}_2\overline{x}_4\overline{x}_6 \qquad (w_6 = \overline{x}_4)$,
$S = \overline{x}_5\overline{x}_6\overline{x}_7$.

We first construct
$R = S \bigtriangledown C_5 = x_3\overline{x}_5\overline{x}_6\overline{x}_9$,
$R = R \bigtriangledown C_2 = x_3\overline{x}_5\overline{x}_9$;
then we replace C_3 by R and,
having chosen $w = \overline{x}_5$, we remove C_4 and C_5 from $\mathcal{F}$.
The resulting updated family is
$C_1 = x_2x_3 \qquad (w_1 = x_2)$,
$C_2 = x_3x_6 \qquad (w_2 = x_6)$,
$C_3 = x_3\overline{x}_5\overline{x}_9 \qquad (w_3 = \overline{x}_5)$,
$C_4 = \overline{x}_2\overline{x}_4\overline{x}_6 \qquad (w_4 = \overline{x}_4)$.

If w happens to be not in S then the update can be iterated before further calls of `obstacle`. Let us illustrate this phenomenon on three additional examples.

EXAMPLE 2:

$$
\begin{aligned}
C_1 &= x_3 && (w_1 = x_3),\\
C_2 &= \overline{x}_2\overline{x}_6 && (w_2 = \overline{x}_2),\\
C_3 &= x_1x_5\overline{x}_6 && (w_3 = x_1),\\
C_4 &= \overline{x}_3\overline{x}_6\overline{x}_9 && (w_4 = \overline{x}_9),\\
C_5 &= x_5x_7x_9 && (w_5 = x_7),\\
C_6 &= x_5x_7x_9 && (w_6 = x_8),\\
S &= \overline{x}_6\overline{x}_7\overline{x}_8.
\end{aligned}
$$

We first construct

$$
\begin{aligned}
R &= S \bigtriangledown C_6 = x_2x_5\overline{x}_6\overline{x}_7,\\
R &= R \bigtriangledown C_5 = x_2x_5\overline{x}_6x_9,\\
R &= R \bigtriangledown C_4 = x_2\overline{x}_3x_5\overline{x}_6,\\
R &= R \bigtriangledown C_2 = \overline{x}_3x_5\overline{x}_6,\\
R &= R \bigtriangledown C_1 = x_5\overline{x}_6;
\end{aligned}
$$

then we replace C_3 by R and,
having set $w = x_5$, we remove C_5 and C_6 from $\mathcal{F}$.
The resulting updated family is

$$
\begin{aligned}
C_1 &= x_3 && (w_1 = x_3),\\
C_2 &= \overline{x}_2\overline{x}_6 && (w_2 = \overline{x}_2),\\
C_3 &= x_5\overline{x}_6 && (w_3 = x_5),\\
C_4 &= \overline{x}_3\overline{x}_6\overline{x}_9 && (w_4 = \overline{x}_9).
\end{aligned}
$$

Since $w \notin S$, a further update of $\mathcal{F}$ is possible:
Observing that $\overline{x}_7 \in S$, $\overline{x}_7 \notin C_1 \cup \ldots \cup C_4$, $x_7 \notin C_1 \cup \ldots \cup C_4$,
we simply add

$$C_5 = \overline{x}_6\overline{x}_7\overline{x}_8 \qquad (w_5 = \overline{x}_7).$$

EXAMPLE 3:

$$
\begin{array}{ll}
C_1 = x_3 & (w_1 = x_3), \\
C_2 = \overline{x}_2\overline{x}_6 & (w_2 = \overline{x}_2), \\
C_3 = x_5\overline{x}_6 & (w_3 = x_5), \\
C_4 = \overline{x}_3\overline{x}_6\overline{x}_9 & (w_4 = \overline{x}_9), \\
C_5 = \overline{x}_6\overline{x}_8 & (w_5 = \overline{x}_8), \\
C_6 = \overline{x}_1\overline{x}_3\overline{x}_5 & (w_6 = \overline{x}_1), \\
C_7 = \overline{x}_3\overline{x}_4\overline{x}_5 & (w_7 = \overline{x}_4), \\
C_8 = x_1x_4x_7 & (w_8 = x_7), \\
S = \overline{x}_3\overline{x}_5\overline{x}_7. &
\end{array}
$$

We first construct

$$
\begin{array}{l}
R = S \bigtriangledown C_8 = x_1\overline{x}_3x_4\overline{x}_5, \\
R = R \bigtriangledown C_7 = x_1\overline{x}_3\overline{x}_5, \\
R = R \bigtriangledown C_6 = \overline{x}_3\overline{x}_5, \\
R = R \bigtriangledown C_3 = \overline{x}_3\overline{x}_6, \\
R = R \bigtriangledown C_1 = \overline{x}_6;
\end{array}
$$

then we replace C_2 by R and,
having set $w = \overline{x}_6$, we remove C_3, C_4, C_5 from $\mathcal{F}$.
The resulting updated family is

$$
\begin{array}{ll}
C_1 = x_3 & (w_1 = x_3), \\
C_2 = \overline{x}_6 & (w_2 = \overline{x}_6), \\
C_3 = \overline{x}_1\overline{x}_3\overline{x}_5 & (w_3 = \overline{x}_1), \\
C_4 = \overline{x}_3\overline{x}_4\overline{x}_5 & (w_4 = \overline{x}_4), \\
C_5 = x_1x_4x_7 & (w_5 = x_7).
\end{array}
$$

Since $w \notin S$, a further update of $\mathcal{F}$ is possible:
Observing that $S \subseteq \left(\bigcup_{i=1}^{4}(C_i - \{w_i\})\right) \cup \{\overline{w}_1, \overline{w}_2, \overline{w}_3, \overline{w}_4\}$,
we first construct

$$
\begin{array}{l}
R = S \bigtriangledown C_5 = x_1\overline{x}_3x_4\overline{x}_5, \\
R = R \bigtriangledown C_4 = x_1\overline{x}_3\overline{x}_5, \\
R = R \bigtriangledown C_3 = \overline{x}_3\overline{x}_5, \\
R = R \bigtriangledown C_1 = \overline{x}_5;
\end{array}
$$

then we replace C_3 by R and,
having set $w = \overline{x}_5$, we remove C_4 from $\mathcal{F}$.
The resulting updated family is

$$
\begin{array}{ll}
C_1 = x_3 & (w_1 = x_3), \\
C_2 = \overline{x}_6 & (w_2 = \overline{x}_6), \\
C_3 = \overline{x}_5 & (w_3 = \overline{x}_5), \\
C_4 = x_1x_4x_7 & (w_4 = x_7).
\end{array}
$$

EXAMPLE 4:

$C_1 = x_3 \qquad (w_1 = x_3),$
$C_2 = \overline{x}_2\overline{x}_4\overline{x}_6 \qquad (w_2 = \overline{x}_4),$
$C_3 = \overline{x}_2\overline{x}_5\overline{x}_6 \qquad (w_3 = \overline{x}_5),$
$C_4 = \overline{x}_2\overline{x}_6 x_7 \qquad (w_4 = x_7),$
$C_5 = \overline{x}_1\overline{x}_2\overline{x}_3 \qquad (w_5 = \overline{x}_1),$
$C_6 = x_1 x_5 x_9 \qquad (w_6 = x_9),$
$S = \overline{x}_3\overline{x}_6\overline{x}_9.$

We first construct
$R = S \bigtriangledown C_6 = x_1\overline{x}_3 x_5\overline{x}_6,$
$R = R \bigtriangledown C_5 = \overline{x}_2\overline{x}_3 x_5\overline{x}_6,$
$R = R \bigtriangledown C_3 = \overline{x}_2\overline{x}_3\overline{x}_6,$
$R = R \bigtriangledown C_1 = \overline{x}_2\overline{x}_6;$
then we replace C_2 by R and,
having chosen $w = \overline{x}_2$, we remove C_3, C_4, C_5 from $\mathcal{F}$.
The resulting updated family is
$C_1 = x_3 \qquad (w_1 = x_3),$
$C_2 = \overline{x}_2\overline{x}_6 \qquad (w_2 = \overline{x}_2),$
$C_3 = x_1 x_5 x_9 \qquad (w_3 = x_9),$
Since $w \notin S$, a further update of $\mathcal{F}$ is possible:
Observing that $S \subseteq \left(\bigcup_{i=1}^{3}(C_i - \{w_i\})\right) \cup \{\overline{w}_1, \overline{w}_2, \overline{w}_3\}$,
we first construct
$R = S \bigtriangledown C_3 = x_1\overline{x}_3 x_5\overline{x}_6,$
$R = R \bigtriangledown C_1 = x_1 x_5\overline{x}_6;$
then we replace C_3 by R and
choose $w = x_1$.
The resulting updated family is
$C_1 = x_3 \qquad (w_1 = x_3),$
$C_2 = \overline{x}_2\overline{x}_6 \qquad (w_2 = \overline{x}_2),$
$C_3 = x_1 x_5\overline{x}_6 \qquad (w_3 = x_1).$
Since $w \notin S$, a further update of $\mathcal{F}$ is possible:
Observing that $\overline{x}_9 \in S$, $\overline{x}_9 \notin C_1 \cup C_2 \cup C_3$, $x_9 \notin C_1 \cup C_2 \cup C_3$,
we simply add
$C_4 = \overline{x}_3\overline{x}_6\overline{x}_9 \qquad (w_4 = \overline{x}_9).$

To summarize, `resolution_search` is defined in Table 2.

```
resolution_search(v)
{
   record = oracle(v);
   m = 0, u = (⋆,⋆,...,⋆);

   while(1){

      try = obstacle(u, record, S);
      if (try < record){
         v = u;
         record = try;
      }

      while(1){

         if(S includes a literal w such that
               w ∉ C_1 ∪ C_2 ∪ ... ∪ C_m and w̄ ∉ C_1 ∪ C_2 ∪ ... ∪ C_m){
            m = m + 1, C_m = S, w_m = w;
            break;
         }

         R = S;
         for (i = m, m − 1, ..., 3, 2, 1)
            if (w̄_i ∈ R)  R = R ▽ C_i;
         if (R = ∅) return record;

         M = m, m = smallest i such that R ⊆ C_1 ∪ C_2 ∪ ... ∪ C_i;
         choose a literal w in R − (C_1 ∪ C_2 ∪ ... ∪ C_{k−1});
         C_m = R, w_m = w;
         for (i = m + 1, m + 2, ..., M)
            if (w ∉ C_i)  m = m + 1, C_m = C_i, w_m = w_i;

         if (w ∈ S) break;
      }

      u = the vector represented by (∪_{i=1}^m (C_i − {w_i})) ∪ {w̄_1, w̄_2, ..., w̄_m}.
   }
}
```

Table 2: Definition of resolution search

Resolution search: an analysis

CLAIM. If a search problem over $\{0, 1, \star\}^n$ is solved by `resolution_search` then the inner `while` loop is executed at most 2^n times.

JUSTIFICATION. Let us define the *strength* of a path-like family $C_1, C_2, \ldots, C_m$ to be $\sum_{j=1}^{m} 2^{-n_j}$ with

$$n_j = \left| \left(\bigcup_{i=1}^{j} (C_i - \{w_i\}) \right) \cup \{\overline{w}_1, \overline{w}_2, \ldots, \overline{w}_j\} \right|$$

and with $w_1, w_2, \ldots, w_m$ as in the definition of a path-like family. Since every path-like family of clauses over n variables has

$$1 \leq n_1 < n_2 < \ldots < n_m \leq n, \tag{4}$$

its strength is a nonnegative multiple of 2^{-n} and less than 1; we are going to prove that strength of the family $\mathcal{F}$ maintained by `resolution_search` increases with each execution of the inner `while` loop except possibly the last, where an empty clause may be inserted into $\mathcal{F}$, in which case $\mathcal{F}$ ceases to be path-like. For this purpose, we shall distinguish between two cases.

If $S \not\subseteq (\bigcup_{i=1}^{m}(C_i - \{w_i\})) \cup \{\overline{w}_1, \overline{w}_2, \ldots, \overline{w}_m\}$ then $C_1, C_2, \ldots, C_m$ is simply extended by S, and so its strength increases trivially.

If $S \subseteq (\bigcup_{i=1}^{m}(C_i - \{w_i\})) \cup \{\overline{w}_1, \overline{w}_2, \ldots, \overline{w}_m\}$ then $C_1, C_2, \ldots, C_m$ is first replaced by $C_1, C_2, \ldots, C_{k-1}, R$ where $R \subseteq \bigcup_{i=1}^{k}(C_i - \{w_i\})$ and then possibly extended by additional terms from $C_{k+1}, C_{k+2}, \ldots, C_m$. By (4), strength of $C_1, C_2, \ldots, C_k$ is strictly less than

$$\sum_{j=1}^{k-1} 2^{-n_j} + 2 \cdot 2^{-n_k}; \tag{5}$$

since

$$\left(\bigcup_{i=1}^{k-1} (C_i - \{w_i\}) \right) \cup \{\overline{w}_1, \overline{w}_2, \ldots, \overline{w}_{k-1}\} \cup R \subseteq \left(\bigcup_{i=1}^{k} (C_i - \{w_i\}) \right) \cup \{\overline{w}_1, \overline{w}_2, \ldots, \overline{w}_{k-1}\},$$

we have

$$\left| \left(\bigcup_{i=1}^{k-1} (C_i - \{w_i\}) \right) \cup \{\overline{w}_1, \overline{w}_2, \ldots, \overline{w}_{k-1}\} \cup R \right| < n_k,$$

and so the strength of $C_1, C_2, \ldots, C_{k-1}, R$ alone is at least (5). □

The positive amount by which strength of the family $\mathcal{F}$ maintained by `resolution_search` increases with an execution of the inner `while` loop may depend on the choice of w in the line

```
choose a literal w in R − (C1 ∪ C2 ∪ ... ∪ Ck−1);
```

for illustration, let us return to EXAMPLE 1 from the previous section. Here, we start with

$$\begin{aligned}
C_1 &= x_2x_3 && (w_1 = x_2),\\
C_2 &= x_3x_6 && (w_2 = x_6),\\
C_3 &= \overline{x}_1\overline{x}_5\overline{x}_9 && (w_3 = \overline{x}_1),\\
C_4 &= \overline{x}_2\overline{x}_5\overline{x}_8 && (w_4 = \overline{x}_8),\\
C_5 &= x_3\overline{x}_5x_7\overline{x}_9 && (w_5 = x_7),\\
C_6 &= \overline{x}_2\overline{x}_4\overline{x}_6 && (w_6 = \overline{x}_4)
\end{aligned}$$

and with

$$S = \overline{x}_5\overline{x}_6\overline{x}_7.$$

Having constructed

$$\begin{aligned}
R &= S \bigtriangledown C_5 = x_3\overline{x}_5\overline{x}_6\overline{x}_9,\\
R &= R \bigtriangledown C_2 = x_3\overline{x}_5\overline{x}_9,
\end{aligned}$$

we replace C_3 by R and then come to the choice of w. If we choose $w = \overline{x}_5$, then we are led to remove C_4 and C_5 from $\mathcal{F}$; the resulting updated family is

$$\begin{aligned}
C_1 &= x_2x_3 && (w_1 = x_2),\\
C_2 &= x_3x_6 && (w_2 = x_6),\\
C_3 &= x_3\overline{x}_5\overline{x}_9 && (w_3 = \overline{x}_5),\\
C_4 &= \overline{x}_2\overline{x}_4\overline{x}_6 && (w_4 = \overline{x}_4).
\end{aligned}$$

If we choose $w = \overline{x}_9$ instead, then we are led to remove only C_5 from $\mathcal{F}$, after which we find that S can be added to $\mathcal{F}$; the resulting updated family is

$$\begin{aligned}
C_1 &= x_2x_3 && (w_1 = x_2),\\
C_2 &= x_3x_6 && (w_2 = x_6),\\
C_3 &= x_3\overline{x}_5\overline{x}_9 && (w_3 = \overline{x}_9),\\
C_4 &= \overline{x}_2\overline{x}_5\overline{x}_8 && (w_4 = \overline{x}_8),\\
C_5 &= \overline{x}_2\overline{x}_4\overline{x}_6 && (w_5 = \overline{x}_4),\\
C_6 &= \overline{x}_5\overline{x}_6\overline{x}_7 && (w_6 = \overline{x}_7).
\end{aligned}$$

Clearly, $w = \overline{x}_9$ is a better choice than $w = \overline{x}_5$.

Strength of $C_1, C_2, \ldots, C_m$ is a *lower bound* on the proportion of those vectors in $\{0,1\}^n$ that extend at least one C_i; being just a bound, it is just an approximation of the actual worth of $C_1, C_2, \ldots, C_m$. In particular, strength of a path-like family may sometimes be increased simply by permuting its members: for instance,

$$\begin{array}{ll} C_1 = x_1x_2x_5 & (w_1 = x_5), \\ C_2 = x_1x_3x_6 & (w_2 = x_6), \\ C_3 = x_1x_4x_7 & (w_3 = x_7), \\ C_4 = x_2x_3x_8 & (w_4 = x_8) \end{array}$$

and

$$\begin{array}{ll} C_1 = x_1x_2x_5 & (w_1 = x_5), \\ C_2 = x_1x_3x_6 & (w_2 = x_6), \\ C_3 = x_2x_3x_8 & (w_3 = x_8), \\ C_4 = x_1x_4x_7 & (w_4 = x_7), \end{array}$$

are identical as *unordered* families and yet the strength of the latter exceeds the strength of the former. In this context, it may be interesting to note that the following problem is $\mathcal{NP}$-complete:

INPUT :	A path-like family $\mathcal{F}$ and a number s.	
QUESTION :	Can members of $\mathcal{F}$ be ordered so that the resulting family remains path-like and has strength at least s?	(6)

To see that (6) is $\mathcal{NP}$-complete, we shall reduce the following $\mathcal{NP}$-complete problem to (6):

INPUT :	A graph G (without multiple edges) and an integer t.	
QUESTION :	Does G contain a clique with t vertices?	(7)

Given a graph G with vertices $v_1, v_2, \ldots, v_n$ and with m edges, let us construct a family $\mathcal{F}$ of m clauses, one clause for each edge of G: if the k-th edge of G has endpoints v_i, v_j then this edge gives rise to clause $x_ix_jx_{n+k}$. For each positive integer k, let us define

$$m_k = k + j \text{ such that } j \text{ is the smallest positive integer with } k \leq \frac{j(j-1)}{2}$$

(so that $m_1 = 3$, $m_2 = 5$, $m_3 = 6$, $m_4 = 8$, $m_5 = 9$, $m_6 = 10$, $m_7 = 12$, and so on); let us set

$$s = \sum_{k=1}^{t(t-1)/2} 2^{-m_k}.$$

No matter how members of $\mathcal{F}$ are ordered, the resulting ordered family is path-like and $n_k \geq m_k$ for all k; equalities $n_k = m_k$ hold for all $k = 1, 2, \ldots, t(t-1)/2$ if and only if, for each $j = 1, 2, \ldots, t$, the first $j(j-1)/2$ clauses correspond to edges of a clique with j vertices. Hence the answer to (7) is "yes" if and only if the answer to (6) is "yes".

Implementations of `obstacle`

Let us recall the specifications of `obstacle(`u`,record,`S`)`: Given any u in $\{0,1,\star\}^n$ and any value `record`, it replaces u with an u^* such that $u \sqsubseteq u^*$ and

$$\texttt{oracle}(u^*) \geq \texttt{record} \quad \text{or else} \quad u^* \in \{0,1\}^n, \tag{8}$$

it constructs a clause S such that $S \sqsubseteq u^*$ and

$$\begin{array}{lcll} \texttt{oracle}(S) & \geq & \texttt{record} & \text{if } \texttt{oracle}(u^*) \geq \texttt{record}, \\ \texttt{oracle}(S) & = & \texttt{oracle}(u^*) & \text{if } \texttt{oracle}(u^*) < \texttt{record}, \end{array} \tag{9}$$

and it returns the value of `oracle(`u^*`)`. To construct u^*, one may keep replacing each $\star$ in u by 0 or 1 until (8) is satisfied; to construct S, one may simply set $S = u^*$. This implementation of `obstacle`, defined formally in Table 3, reduces `resolution_search` to `implicit_enumeration` in disguise.

```
obstacle(u, record, S)
{
   bound = oracle(u);
   if (bound < record){
      while (u ∉ {0,1}^n){
         choose a subscript j with u_j = ⋆ and a value c in {0,1};
         u_j = c;
         bound = oracle(u);
         if (bound ≥ record) break;
      }
   }

   S = u;
   return bound;
}
```

Table 3: A quick `obstacle` that produces big S

Another option is to first replace u by its extension u^* as in Table 3 and then construct a *minimal* (with respect to $\sqsubseteq$) clause S that satisfies $S \sqsubseteq u^*$ and (9): one may begin with $S = u^*$ and then keep removing literals from S as long as (9) is satisfied. Here, the order in which the literals are considered for removal may be important: there may be many different minimal restrictions S of u^* that satisfy (9) and inappropriate choices of S may lead `resolution_search` to an unnecessarily large number of calls of `obstacle`. If we subscribe to the thesis that a tight overlap among $C_1, C_2, \ldots, C_m$ is desirable, then we are led to removing first those literals that have been added in the transition from the initial u to its extension u^*. One implementation of this policy is defined formally in Table 6.

```
obstacle(u, record, S)
{
   initialize an empty stack;
   for (j = 1, 2, ..., n)
      if (u_j ≠ ⋆) push j on the stack;

   bound = oracle(u);
   if (bound < record){
      while (u ∉ {0,1}^n){
         choose a subscript j with u_j = ⋆ and a value c in {0,1};
         u_j = c;
         bound = oracle(u);
         if (bound ≥ record) break;
         push j on the stack;
      }
      if (bound < record) record=bound;
   }

   S = u;
   while (the stack is nonempty){
      pop j off the stack;
      S_j = ⋆;
      if (oracle(S) < record) S_j = u_j;
   }

   return bound;
}
```

Table 4: A slow obstacle that produces small S

We have been, and will be, thinking of `oracle` as a black box. However, a particular `oracle` may sometimes be exploited to skip some of the tests "`if (oracle(S) < record)`" in Table 4. For instance, in the linear programming problem (2), optimal values of the dual variables may point out a *proper* subset J of $\{j : u_j \neq \star\}$ such that

$$\begin{array}{llll}
\text{minimize} & \sum_{j=1}^{N} c_j x_j & & \\
\text{subject to} & \sum_{j=1}^{N} a_{ij} x_j & \geq b_i & (i = 1, 2, \ldots, m), \\
& 0 \leq \; x_j & \leq 1 & (j = 1, 2, \ldots, n), \\
& x_j & = u_j & \text{whenever } j \notin J
\end{array}$$

and (2) have the same optimal value or are both infeasible.

Resolution search: examples

To illustrate `resolution_search`, we shall implement `obstacle` by Table 4 with the primitive branching strategy

"let j be the *smallest* subscript with $u_j = \star$ and set $c = 0$"

that we used earlier in solving (3) by `implicit_enumeration`; similarly, when it comes to a choice of w, we shall choose the candidate with the smallest subscript.

First, let us return to problem (3); again, we shall use the `oracle` defined through (2) and we shall initialize $v = (0, 0, 0, 0)$.

ITERATION 1:
calling `obstacle`$((\star, \star, \star, \star), +\infty, S)$
`oracle`$((\star, \star, \star, \star)) = 1$
`oracle`$((0, \star, \star, \star)) = 1$
`oracle`$((0, 0, \star, \star)) = 1$
`oracle`$((0, 0, 0, \star)) = +\infty$
$u^* = (0, 0, 0, \star)$;
`oracle`$((0, \star, 0, \star)) = 1$
`oracle`$((\star, 0, 0, \star)) = 1$
$S = x_1 x_2 x_3$;
`obstacle`$((\star, \star, \star, \star), +\infty, S)$ returns $+\infty$;

$C_1 = x_1 x_2 x_3 \qquad (w_1 = x_1)$

ITERATION 2:
calling `obstacle`$((1, 0, 0, \star), +\infty, S)$
`oracle`$((1, 0, 0, \star)) = 1$
`oracle`$((1, 0, 0, 0)) = +\infty$
$u^* = (1, 0, 0, 0)$;
`oracle`$((1, 0, \star, 0)) = 2$
`oracle`$((1, \star, 0, 0)) = 2$
`oracle`$((\star, 0, 0, 0)) = +\infty$
$S = x_2 x_3 x_4$;
`obstacle`$((1, 0, 0, \star), +\infty, S)$ returns $+\infty$;

$C_1 = x_1 x_2 x_3 \qquad (w_1 = x_1)$
$C_2 = x_2 x_3 x_4 \qquad (w_2 = x_4)$

Iteration 3:

calling `obstacle(`$(1,0,0,1)$`,`$+\infty, S$`)`

`oracle(`$(1,0,0,1) = +\infty$

$u^* = (1,0,0,1)$;

`oracle(`$(1,0,0,\star) = 1$

`oracle(`$(1,0,\star,1) = 2$

`oracle(`$(1,\star,0,1) = 2$

`oracle(`$(\star,0,0,1) = +\infty$

$S = x_2 x_3 \overline{x}_4$;

`obstacle(`$(1,0,0,1)$`,`$+\infty, S$`)` returns $+\infty$;

$R = S \bigtriangledown C_2 = x_2 x_3$;

$C_1 = x_2 x_3 \qquad (w_1 = x_2)$

Iteration 4:

calling `obstacle(`$(\star,1,0,\star)$`,`$+\infty, S$`)`

`oracle(`$(\star,1,0,\star) = 1$

`oracle(`$(0,1,0,\star) = 1$

`oracle(`$(0,1,0,0) = +\infty$

$u^* = (0,1,0,0)$;

`oracle(`$(0,1,\star,0) = 2$

`oracle(`$(0,\star,0,0) = +\infty$

`oracle(`$(\star,\star,0,0) = 2$

$S = x_1 x_3 x_4$;

`obstacle(`$(\star,1,0,\star)$`,`$+\infty, S$`)` returns $+\infty$;

$C_1 = x_2 x_3 \qquad (w_1 = x_2)$

$C_2 = x_1 x_3 x_4 \qquad (w_2 = x_1)$

ITERATION 5:

calling `obstacle(`$(1,1,0,0)$`,`$+\infty, S$`)`

`oracle(`$(1,1,0,0) = 2$

$u^* = (1,1,0,0)$;

`oracle(`$(1,1,0,\star) = 2$

`oracle(`$(1,1,\star,\star) = 2$

`oracle(`$(1,\star,\star,\star) = 1$

`oracle(`$(\star,1,\star,\star) = 1$

$S = \overline{x}_1\overline{x}_2$;

`obstacle(`$(1,1,0,0)$`,`$+\infty, S$`)` returns 2;

$v = (1,1,0,0)$, `record`$= 2$;

$R = (S \bigtriangledown C_2) \bigtriangledown C_1 = x_3x_4$;

$C_1 = x_2x_3$	$(w_1 = x_2)$
$C_2 = x_3x_4$	$(w_2 = x_4)$
$C_3 = \overline{x}_1\overline{x}_2$	$(w_3 = \overline{x}_1)$

ITERATION 6:

calling `obstacle(`$(0,1,0,1)$`,`$2, S$`)`

`oracle(`$(0,1,0,1) = +\infty$

$u^* = (0,1,0,1)$;

`oracle(`$(0,1,0,\star) = 1$

`oracle(`$(0,1,\star,1) = 2$

`oracle(`$(0,\star,\star,1) = 2$

`oracle(`$(\star,\star,\star,1) = 2$

$S = \overline{x}_4$;

`obstacle(`$(0,1,0,1)$`,`$2, S$`)` returns $+\infty$;

$R = S \bigtriangledown C_2 = x_3$;

$C_1 = x_3$	$(w_1 = x_3)$
$C_2 = \overline{x}_1\overline{x}_2$	$(w_2 = \overline{x}_1)$
$C_3 = \overline{x}_4$	$(w_3 = \overline{x}_4)$

ITERATION 7:

calling `obstacle(`$(0,1,1,0)$`,`$2,S$`)`

$\quad\quad$ `oracle(`$(0,1,1,0)=2$

$\quad u^* = (0,1,1,0)$;

$\quad\quad$ `oracle(`$(0,1,1,\star)=2$

$\quad\quad$ `oracle(`$(0,1,\star,\star)=1$

$\quad\quad$ `oracle(`$(0,\star,1,\star)=1$

$\quad\quad$ `oracle(`$(\star,1,1,\star)=2$

$\quad S = \overline{x}_2\overline{x}_3$;

`obstacle(`$(0,1,1,0)$`,`$2,S$`)` returns 2;

$R = S \bigtriangledown C_1 = \overline{x}_2$;

$C_1 = x_3$	$(w_1 = x_3)$
$C_2 = \overline{x}_2$	$(w_2 = \overline{x}_2)$
$C_3 = \overline{x}_4$	$(w_3 = \overline{x}_4)$

ITERATION 8:

calling `obstacle(`$(\star,0,1,0)$`,`$2,S$`)`

$\quad\quad$ `oracle(`$(\star,0,1,0)=2$

$\quad u^* = (\star,0,1,0)$;

$\quad\quad$ `oracle(`$(\star,0,1,\star)=1$

$\quad\quad$ `oracle(`$(\star,0,\star,0)=2$

$\quad\quad$ `oracle(`$(\star,\star,\star,0)=2$

$\quad S = x_4$;

`obstacle(`$(\star,0,1,0)$`,`$2,S$`)` returns 2;

$R = S \bigtriangledown C_3 = \emptyset$;

To take a larger and a less contrived example, let us use `resolution_search` to prove that

$$\begin{array}{l}\text{for every partition } \{1,2,3,4,5,6,7,8,9\} = A_1 \cup A_2,\\ \text{at least one } A_i \text{ contains an arithmetic progression with three terms.}\end{array} \tag{10}$$

With

$$\texttt{oracle}(u_1,u_2,\ldots,u_n) = \begin{cases} 1 & \text{if there are } a \text{ and } d \text{ such that } d \neq 0 \text{ and} \\ & u_a = u_{a+d} = u_{a+2d} = 0 \text{ or } u_a = u_{a+d} = u_{a+2d} = 1 \\ & \\ 0 & \text{otherwise,} \end{cases}$$

claim (10) amounts to saying that the minimum of `oracle` over $\{0,1\}^n$ equals 1. To solve the corresponding search problem, `resolution_search` goes through 43 iterations, in which it constructs the following 124 clauses.

$R_{1} = x_1 x_2 x_3$
$R_{2} = x_2 x_3 x_4$
$R_{3} = x_3 x_5 x_7$
$R_{4} = x_6 x_7 x_8$
$R_{5} = \overline{x}_4 \overline{x}_5 \overline{x}_6$
$R_{6} = \overline{x}_4 \overline{x}_5 x_7 x_8$
$R_{7} = x_3 \overline{x}_4 x_7 x_8$
$R_{8} = x_2 x_3 x_7 x_8$
$R_{9} = x_3 x_6 x_9$
$R_{10} = \overline{x}_1 x_5 \overline{x}_9$
$R_{11} = \overline{x}_1 x_3 \overline{x}_5 x_6$
$R_{12} = \overline{x}_1 x_3 \overline{x}_4 \overline{x}_5$
$R_{13} = \overline{x}_1 x_3 \overline{x}_4 x_7$
$R_{14} = \overline{x}_1 x_2 x_3 x_7$
$R_{15} = x_2 x_3 x_7$
$R_{16} = \overline{x}_1 x_2 x_3 \overline{x}_5$
$R_{17} = x_2 x_3 \overline{x}_5$
$R_{18} = \overline{x}_1 \overline{x}_4 \overline{x}_7$
$R_{19} = \overline{x}_1 x_2 x_3 \overline{x}_4$
$R_{20} = \overline{x}_1 x_2 x_3$
$R_{21} = x_2 x_3$
$R_{22} = x_1 x_3 x_5$
$R_{23} = \overline{x}_5 \overline{x}_7 \overline{x}_9$
$R_{24} = x_1 x_3 \overline{x}_7 \overline{x}_9$
$R_{25} = x_3 \overline{x}_4 \overline{x}_7 \overline{x}_9$
$R_{26} = x_3 \overline{x}_4 x_6 \overline{x}_7$
$R_{27} = x_1 x_3 x_6 \overline{x}_7$
$R_{28} = x_3 x_4 x_5$
$R_{29} = x_3 x_4 \overline{x}_7 \overline{x}_9$
$R_{30} = x_3 x_6 \overline{x}_7 \overline{x}_9$
$R_{31} = x_3 x_6 \overline{x}_7$
$R_{32} = x_3 x_5 x_6$
$R_{33} = x_1 x_4 x_7$
$R_{34} = x_4 \overline{x}_5 x_7 \overline{x}_9$
$R_{35} = x_3 x_4 x_6 x_7 \overline{x}_9$
$R_{36} = x_3 x_4 x_6 \overline{x}_9$
$R_{37} = x_3 x_4 x_6$
$R_{38} = \overline{x}_2 \overline{x}_5 \overline{x}_8$
$R_{39} = \overline{x}_2 \overline{x}_5 x_6 x_7$
$R_{40} = \overline{x}_2 x_3 x_6 x_7$
$R_{41} = \overline{x}_2 x_3 x_6$
$R_{42} = x_3 x_6$
$R_{43} = \overline{x}_2 \overline{x}_4 \overline{x}_6$
$R_{44} = x_1 \overline{x}_2 \overline{x}_6 x_7$
$R_{45} = \overline{x}_2 \overline{x}_5 \overline{x}_6 x_7 \overline{x}_9$
$R_{46} = \overline{x}_2 x_3 \overline{x}_5 x_7 \overline{x}_9$
$R_{47} = x_3 \overline{x}_5 x_7 \overline{x}_9$
$R_{48} = \overline{x}_5 \overline{x}_6 \overline{x}_7$
$R_{49} = x_3 \overline{x}_5 \overline{x}_6 \overline{x}_9$
$R_{50} = x_3 \overline{x}_5 \overline{x}_9$
$R_{51} = \overline{x}_2 x_3 x_5 \overline{x}_6$
$R_{52} = \overline{x}_2 x_3 \overline{x}_6 \overline{x}_9$
$R_{53} = \overline{x}_2 x_3 \overline{x}_9$
$R_{54} = x_3 \overline{x}_9$
$R_{55} = x_7 x_8 x_9$
$R_{56} = \overline{x}_2 \overline{x}_5 x_7 x_9$
$R_{57} = \overline{x}_2 x_3 x_4 x_7 x_9$
$R_{58} = \overline{x}_2 x_3 \overline{x}_6 x_7 x_9$
$R_{59} = \overline{x}_2 x_3 \overline{x}_6 x_7$
$R_{60} = \overline{x}_2 x_3 x_7$
$R_{61} = x_3 x_7$
$R_{62} = x_3 \overline{x}_5 \overline{x}_6$
$R_{63} = x_3 x_4 \overline{x}_6$
$R_{64} = \overline{x}_2 x_3 \overline{x}_6$
$R_{65} = \overline{x}_2 x_3$
$R_{66} = x_3$
$R_{67} = \overline{x}_1 \overline{x}_2 \overline{x}_3$
$R_{68} = \overline{x}_2 \overline{x}_3 x_4 x_7$
$R_{69} = \overline{x}_2 \overline{x}_3 x_4 \overline{x}_5 \overline{x}_6$
$R_{70} = \overline{x}_2 \overline{x}_3 \overline{x}_5 \overline{x}_6$
$R_{71} = \overline{x}_2 \overline{x}_5 \overline{x}_6$
$R_{72} = \overline{x}_2 \overline{x}_3 \overline{x}_6 x_7$
$R_{73} = \overline{x}_2 \overline{x}_6 x_7$
$R_{74} = x_1 x_5 x_9$
$R_{75} = \overline{x}_3 \overline{x}_6 \overline{x}_9$
$R_{76} = x_1 \overline{x}_3 x_5 \overline{x}_6$
$R_{77} = \overline{x}_2 \overline{x}_3 x_5 \overline{x}_6$
$R_{78} = \overline{x}_2 \overline{x}_3 \overline{x}_6$
$R_{79} = \overline{x}_2 \overline{x}_6$
$R_{80} = x_1 x_5 \overline{x}_6$
$R_{81} = x_5 x_7 x_9$
$R_{82} = x_2 x_5 x_8$
$R_{83} = \overline{x}_6 \overline{x}_7 \overline{x}_8$
$R_{84} = x_2 x_5 \overline{x}_6 \overline{x}_7$
$R_{85} = x_2 x_5 \overline{x}_6 x_9$
$R_{86} = x_2 \overline{x}_3 x_5 \overline{x}_6$
$R_{87} = \overline{x}_3 x_5 \overline{x}_6$
$R_{88} = x_5 \overline{x}_6$
$R_{89} = \overline{x}_1 \overline{x}_3 \overline{x}_5$
$R_{90} = \overline{x}_3 x_4 \overline{x}_5 x_7$
$R_{91} = \overline{x}_3 x_4 \overline{x}_5 \overline{x}_6 \overline{x}_8$
$R_{92} = \overline{x}_3 x_4 \overline{x}_6 \overline{x}_8$
$R_{93} = x_4 \overline{x}_6 \overline{x}_8$
$R_{94} = \overline{x}_3 \overline{x}_4 \overline{x}_5$
$R_{95} = \overline{x}_3 \overline{x}_5 \overline{x}_6 \overline{x}_8$
$R_{96} = \overline{x}_3 \overline{x}_6 \overline{x}_8$
$R_{97} = \overline{x}_6 \overline{x}_8$
$R_{98} = \overline{x}_3 \overline{x}_5 \overline{x}_7$
$R_{99} = x_1 \overline{x}_3 x_4 \overline{x}_5$
$R_{100} = x_1 \overline{x}_3 \overline{x}_5$
$R_{101} = \overline{x}_3 \overline{x}_5$
$R_{102} = \overline{x}_3 \overline{x}_6$
$R_{103} = \overline{x}_6$
$R_{104} = \overline{x}_5$
$R_{105} = x_4 x_5 x_6$
$R_{106} = x_4 x_6$
$R_{107} = x_4$
$R_{108} = x_5 x_6 x_7$
$R_{109} = \overline{x}_2 \overline{x}_3 \overline{x}_4$
$R_{110} = \overline{x}_3 \overline{x}_4 x_5 x_8$
$R_{111} = \overline{x}_3 x_5 x_8$
$R_{112} = \overline{x}_3 x_8$
$R_{113} = x_8$
$R_{114} = \overline{x}_4 x_5 \overline{x}_7 x_9$
$R_{115} = \overline{x}_4 x_5 x_6 x_9$
$R_{116} = x_5 x_6 x_9$
$R_{117} = x_6 x_9$
$R_{118} = x_9$
$R_{119} = \overline{x}_7 \overline{x}_8 \overline{x}_9$
$R_{120} = \overline{x}_7 \overline{x}_8$
$R_{121} = \overline{x}_7$
$R_{122} = x_5 x_6$
$R_{123} = x_6$
$R_{124} = \emptyset$

Here is a terse record of the 43 iterations.

ITERATION 1 : $S = R_1$, $\mathcal{F} = \{R_1\}$
ITERATION 2 : $S = R_2$, $\mathcal{F} = \{R_1, R_2\}$
ITERATION 3 : $S = R_3$, $\mathcal{F} = \{R_1, R_2, R_3\}$
ITERATION 4 : $S = R_4$, $\mathcal{F} = \{R_1, R_2, R_3, R_4\}$
ITERATION 5 : $S = R_5$, $\mathcal{F} = \{R_1, R_2, R_3, R_8, R_5\}$
ITERATION 6 : $S = R_9$, $\mathcal{F} = \{R_1, R_2, R_3, R_8, R_5, R_9\}$
ITERATION 7 : $S = R_{10}$, $\mathcal{F} = \{R_1, R_2, R_{15}, R_{17}, R_9\}$
ITERATION 8 : $S = R_{18}$, $\mathcal{F} = \{R_{21}, R_9, R_{18}\}$
ITERATION 9 : $S = R_{22}$, $\mathcal{F} = \{R_{21}, R_9, R_{18}, R_{22}\}$
ITERATION 10 : $S = R_{23}$, $\mathcal{F} = \{R_{21}, R_9, R_{26}, R_{27}, R_{23}\}$
ITERATION 11 : $S = R_{28}$, $\mathcal{F} = \{R_{21}, R_9, R_{31}, R_{28}\}$
ITERATION 12 : $S = R_3$, $\mathcal{F} = \{R_{21}, R_9, R_{31}, R_{32}\}$
ITERATION 13 : $S = R_{33}$, $\mathcal{F} = \{R_{21}, R_9, R_{31}, R_{32}, R_{33}\}$
ITERATION 14 : $S = R_{10}$, $\mathcal{F} = \{R_{21}, R_9, R_{31}, R_{32}, R_{37}, R_{10}\}$
ITERATION 15 : $S = R_4$, $\mathcal{F} = \{R_{21}, R_9, R_{31}, R_{32}, R_{37}, R_{10}, R_4\}$
ITERATION 16 : $S = R_{38}$, $\mathcal{F} = \{R_{21}, R_{42}, R_{10}, R_{38}\}$
ITERATION 17 : $S = R_{33}$, $\mathcal{F} = \{R_{21}, R_{42}, R_{10}, R_{38}, R_{33}\}$
ITERATION 18 : $S = R_{43}$, $\mathcal{F} = \{R_{21}, R_{42}, R_{10}, R_{38}, R_{47}, R_{43}\}$
ITERATION 19 : $S = R_{48}$, $\mathcal{F} = \{R_{21}, R_{42}, R_{50}, R_{43}\}$
ITERATION 20 : $S = R_{28}$, $\mathcal{F} = \{R_{21}, R_{42}, R_{54}, R_{43}, R_{28}\}$
ITERATION 21 : $S = R_{33}$, $\mathcal{F} = \{R_{21}, R_{42}, R_{54}, R_{43}, R_{28}, R_{33}\}$
ITERATION 22 : $S = R_{55}$, $\mathcal{F} = \{R_{21}, R_{42}, R_{54}, R_{43}, R_{28}, R_{33}, R_{55}\}$
ITERATION 23 : $S = R_{38}$, $\mathcal{F} = \{R_{21}, R_{42}, R_{54}, R_{43}, R_{28}, R_{61}, R_{38}\}$
ITERATION 24 : $S = R_{48}$, $\mathcal{F} = \{R_{66}, R_{43}, R_{38}, R_{48}\}$
ITERATION 25 : $S = R_{33}$, $\mathcal{F} = \{R_{66}, R_{43}, R_{38}, R_{48}, R_{33}\}$
ITERATION 26 : $S = R_{67}$, $\mathcal{F} = \{R_{66}, R_{43}, R_{71}, R_{73}, R_{67}\}$
ITERATION 27 : $S = R_{74}$, $\mathcal{F} = \{R_{66}, R_{43}, R_{71}, R_{73}, R_{67}, R_{74}\}$
ITERATION 28 : $S = R_{75}$, $\mathcal{F} = \{R_{66}, R_{79}, R_{80}, R_{75}\}$
ITERATION 29 : $S = R_{81}$, $\mathcal{F} = \{R_{66}, R_{79}, R_{80}, R_{75}, R_{81}\}$
ITERATION 30 : $S = R_{82}$, $\mathcal{F} = \{R_{66}, R_{79}, R_{80}, R_{75}, R_{81}, R_{82}\}$
ITERATION 31 : $S = R_{83}$, $\mathcal{F} = \{R_{66}, R_{79}, R_{88}, R_{75}, R_{83}\}$
ITERATION 32 : $S = R_{33}$, $\mathcal{F} = \{R_{66}, R_{79}, R_{88}, R_{75}, R_{83}, R_{33}\}$
ITERATION 33 : $S = R_{89}$, $\mathcal{F} = \{R_{66}, R_{79}, R_{88}, R_{75}, R_{83}, R_{93}, R_{89}\}$
ITERATION 34 : $S = R_{94}$, $\mathcal{F} = \{R_{66}, R_{79}, R_{88}, R_{75}, R_{97}, R_{89}, R_{94}\}$
ITERATION 35 : $S = R_{33}$, $\mathcal{F} = \{R_{66}, R_{79}, R_{88}, R_{75}, R_{97}, R_{89}, R_{94}, R_{33}\}$
ITERATION 36 : $S = R_{98}$, $\mathcal{F} = \{R_{66}, R_{103}, R_{104}, R_{33}\}$
ITERATION 37 : $S = R_{105}$, $\mathcal{F} = \{R_{66}, R_{103}, R_{104}, R_{107}\}$
ITERATION 38 : $S = R_{108}$, $\mathcal{F} = \{R_{66}, R_{103}, R_{104}, R_{107}, R_{108}\}$
ITERATION 39 : $S = R_{82}$, $\mathcal{F} = \{R_{66}, R_{103}, R_{104}, R_{107}, R_{108}, R_{82}\}$
ITERATION 40 : $S = R_{109}$, $\mathcal{F} = \{R_{66}, R_{103}, R_{104}, R_{107}, R_{108}, R_{113}, R_{109}\}$
ITERATION 41 : $S = R_{74}$, $\mathcal{F} = \{R_{66}, R_{103}, R_{104}, R_{107}, R_{108}, R_{113}, R_{109}, R_{74}\}$
ITERATION 42 : $S = R_{18}$, $\mathcal{F} = \{R_{66}, R_{103}, R_{104}, R_{107}, R_{108}, R_{113}, R_{109}, R_{118}, R_{18}\}$
ITERATION 43 : $S = R_{119}$, $R = \emptyset$

Resolution search and resolution refutations

A *resolution refutation* of the claim that

$$\text{there is a vector } u \text{ in } \{0,1\}^n \text{ with } \texttt{oracle}(u) < r \tag{11}$$

is a sequence $R_1, R_2, \ldots, R_M$ of clauses such that

(i) for each $k = 1, 2, \ldots, M$,
$\texttt{oracle}(R_k) \geq r$ or else $R_k = R_i \bigtriangledown R_j$ for some subscripts i, j smaller than k,

(ii) $R_M = \emptyset$.

Under assumption (i), induction on k shows that $\texttt{oracle}(u) \geq r$ whenever $u \in \{0,1\}^n$ and $R_k \sqsubseteq u$; hence (i) and (ii) together imply that claim (11) is false.

The chronologically ordered sequence of clauses constructed by `resolution_search` in the process of solving search problem (1) satisfies (i) with r equal to the optimal value of (1) and it satisfies (ii); hence it provides a certificate of optimality for the vector v constructed by `resolution_search`.

For example, in solving problem (3) in the last section. `resolution_search` constructed the sequence

$$\begin{array}{lll@{\qquad}lll}
R_1 = & x_1x_2x_3 & (\text{in Iteration } 1), & R_8 = & x_3x_4 & (\text{in Iteration } 5), \\
R_2 = & x_2x_3x_4 & (\text{in Iteration } 2), & R_9 = & \overline{x}_4 & (\text{in Iteration } 6), \\
R_3 = & x_2x_3\overline{x}_4 & (\text{in Iteration } 3), & R_{10} = & x_3 & (\text{in Iteration } 6), \\
R_4 = & x_2x_3 & (\text{in Iteration } 3), & R_{11} = & \overline{x}_2\overline{x}_3 & (\text{in Iteration } 7) \\
R_5 = & x_1x_3x_4 & (\text{in Iteration } 4), & R_{12} = & \overline{x}_2 & (\text{in Iteration } 7), \\
R_6 = & \overline{x}_1\overline{x}_2 & (\text{in Iteration } 5), & R_{13} = & x_4 & (\text{in Iteration } 8), \\
R_7 = & \overline{x}_2x_3x_4 & (\text{in Iteration } 5), & R_{14} = & \emptyset & (\text{in Iteration } 8)
\end{array}$$

which satisfies (i) with $r = 2$: we have

$$\begin{array}{ll}
\texttt{oracle}(R_1) = +\infty, & R_8 = R_7 \bigtriangledown R_4, \\
\texttt{oracle}(R_2) = +\infty, & \texttt{oracle}(R_9) = 2, \\
\texttt{oracle}(R_3) = +\infty, & R_{10} = R_9 \bigtriangledown R_8, \\
R_4 = R_3 \bigtriangledown R_2, & \texttt{oracle}(R_{11}) = 2, \\
\texttt{oracle}(R_5) = +\infty, & R_{12} = R_{11} \bigtriangledown R_{10}, \\
\texttt{oracle}(R_6) = 2, & \texttt{oracle}(R_{13}) = 2, \\
R_7 = R_6 \bigtriangledown R_5, & R_{14} = R_{13} \bigtriangledown R_9.
\end{array}$$

This sequence provides a certificate of optimality for the vector $(1, 1, 0, 0)$ constructed by `resolution_search`; so does its subsequence R_9, R_{13}, R_{14}.

Similarly, the sequence $R_1, R_2, \ldots, R_{124}$ constructed by `resolution_search` in the process of proving claim (10) satisfies (i) with $r = 1$: we have

$$
\begin{aligned}
R_6 &= R_5 \bigtriangledown R_4, \\
R_7 &= R_6 \bigtriangledown R_3, \\
R_8 &= R_7 \bigtriangledown R_2, \\
R_{11} &= R_{10} \bigtriangledown R_9, \\
R_{12} &= R_{11} \bigtriangledown R_5, \\
R_{13} &= R_{12} \bigtriangledown R_3, \\
R_{14} &= R_{13} \bigtriangledown R_2, \\
R_{15} &= R_{14} \bigtriangledown R_1, \\
R_{16} &= R_{12} \bigtriangledown R_2, \\
R_{17} &= R_{16} \bigtriangledown R_1, \\
R_{19} &= R_{18} \bigtriangledown R_{15}, \\
R_{20} &= R_{19} \bigtriangledown R_2, \\
R_{21} &= R_{20} \bigtriangledown R_1, \\
R_{24} &= R_{23} \bigtriangledown R_{22}, \\
R_{25} &= R_{24} \bigtriangledown R_{18}, \\
R_{26} &= R_{25} \bigtriangledown R_9, \\
R_{27} &= R_{24} \bigtriangledown R_9, \\
R_{29} &= R_{28} \bigtriangledown R_{23}, \\
R_{30} &= R_{29} \bigtriangledown R_{26}, \\
R_{31} &= R_{30} \bigtriangledown R_9, \\
R_{32} &= R_3 \bigtriangledown R_{31}, \\
R_{34} &= R_{10} \bigtriangledown R_{33}, \\
R_{35} &= R_{34} \bigtriangledown R_{32}, \\
R_{36} &= R_{35} \bigtriangledown R_{31}, \\
R_{37} &= R_{36} \bigtriangledown R_9, \\
R_{39} &= R_{38} \bigtriangledown R_4, \\
R_{40} &= R_{39} \bigtriangledown R_{32}, \\
R_{41} &= R_{40} \bigtriangledown R_{31}, \\
R_{42} &= R_{41} \bigtriangledown R_{21}, \\
R_{44} &= R_{43} \bigtriangledown R_{33}, \\
R_{45} &= R_{44} \bigtriangledown R_{10}, \\
R_{46} &= R_{45} \bigtriangledown R_{42}, \\
R_{47} &= R_{46} \bigtriangledown R_{21}, \\
R_{49} &= R_{48} \bigtriangledown R_{47}, \\
R_{50} &= R_{49} \bigtriangledown R_{42}, \\
R_{51} &= R_{28} \bigtriangledown R_{43}, \\
R_{52} &= R_{51} \bigtriangledown R_{50}, \\
R_{53} &= R_{52} \bigtriangledown R_{42}, \\
R_{54} &= R_{53} \bigtriangledown R_{21}, \\
R_{56} &= R_{38} \bigtriangledown R_{55}, \\
R_{57} &= R_{56} \bigtriangledown R_{28}, \\
R_{58} &= R_{57} \bigtriangledown R_{43}, \\
R_{59} &= R_{58} \bigtriangledown R_{54}, \\
R_{60} &= R_{59} \bigtriangledown R_{42}, \\
R_{61} &= R_{60} \bigtriangledown R_{21}, \\
R_{62} &= R_{48} \bigtriangledown R_{61}, \\
R_{63} &= R_{62} \bigtriangledown R_{28}, \\
R_{64} &= R_{63} \bigtriangledown R_{43}, \\
R_{65} &= R_{64} \bigtriangledown R_{42}, \\
R_{66} &= R_{65} \bigtriangledown R_{21}, \\
R_{68} &= R_{67} \bigtriangledown R_{33}, \\
R_{69} &= R_{68} \bigtriangledown R_{48}, \\
R_{70} &= R_{69} \bigtriangledown R_{43}, \\
R_{71} &= R_{70} \bigtriangledown R_{66}, \\
R_{72} &= R_{68} \bigtriangledown R_{43}, \\
R_{73} &= R_{72} \bigtriangledown R_{66}, \\
R_{76} &= R_{75} \bigtriangledown R_{74}, \\
R_{77} &= R_{76} \bigtriangledown R_{67}, \\
R_{78} &= R_{77} \bigtriangledown R_{71}, \\
R_{79} &= R_{78} \bigtriangledown R_{66}, \\
R_{80} &= R_{76} \bigtriangledown R_{66}, \\
R_{84} &= R_{83} \bigtriangledown R_{82}, \\
R_{85} &= R_{84} \bigtriangledown R_{81}, \\
R_{86} &= R_{85} \bigtriangledown R_{75}, \\
R_{87} &= R_{86} \bigtriangledown R_{79}, \\
R_{88} &= R_{87} \bigtriangledown R_{66}, \\
R_{90} &= R_{89} \bigtriangledown R_{33}, \\
R_{91} &= R_{90} \bigtriangledown R_{83}, \\
R_{92} &= R_{91} \bigtriangledown R_{88}, \\
R_{93} &= R_{92} \bigtriangledown R_{66}, \\
R_{95} &= R_{94} \bigtriangledown R_{93}, \\
R_{96} &= R_{95} \bigtriangledown R_{88}, \\
R_{97} &= R_{96} \bigtriangledown R_{66}, \\
R_{99} &= R_{98} \bigtriangledown R_{33}, \\
R_{100} &= R_{99} \bigtriangledown R_{94}, \\
R_{101} &= R_{100} \bigtriangledown R_{89}, \\
R_{102} &= R_{101} \bigtriangledown R_{88}, \\
R_{103} &= R_{102} \bigtriangledown R_{66}, \\
R_{104} &= R_{101} \bigtriangledown R_{66}, \\
R_{106} &= R_{105} \bigtriangledown R_{104}, \\
R_{107} &= R_{106} \bigtriangledown R_{103}, \\
R_{110} &= R_{109} \bigtriangledown R_{82}, \\
R_{111} &= R_{110} \bigtriangledown R_{107}, \\
R_{112} &= R_{111} \bigtriangledown R_{104}, \\
R_{113} &= R_{112} \bigtriangledown R_{66}, \\
R_{114} &= R_{18} \bigtriangledown R_{74}, \\
R_{115} &= R_{114} \bigtriangledown R_{108}, \\
R_{116} &= R_{115} \bigtriangledown R_{107}, \\
R_{117} &= R_{116} \bigtriangledown R_{104}, \\
R_{118} &= R_{117} \bigtriangledown R_{103}, \\
R_{120} &= R_{119} \bigtriangledown R_{118}, \\
R_{121} &= R_{120} \bigtriangledown R_{113}, \\
R_{122} &= R_{121} \bigtriangledown R_{108}, \\
R_{123} &= R_{122} \bigtriangledown R_{104}, \\
R_{124} &= R_{123} \bigtriangledown R_{103},
\end{aligned}
$$

and `oracle`$(R_k) = 1$ for each remaining R_k. This sequence provides a proof of (10); so does its subsequence obtained by removing R_6, R_7, R_8, R_{16}, R_{17}, R_{27}, R_{34}, R_{35}, R_{36}, R_{37}, R_{72}, R_{73}, R_{80}, R_{90}, R_{91}, R_{92}, R_{93}, R_{95}, R_{96}, R_{97}.

Resolution search versus implicit enumeration

Table 5 reports the results of a comparison of the two algorithms on ten problems with $n = 20$, ten problems with $n = 30$, and ten problems with $n = 40$. Both algorithms used the primitive branching strategy

"let j be the *smallest* subscript with $u_j = \star$ and set $c = 0$";

`oracle` was treated as a black box and `obstacle` was implemented as in Table 4. Over 90% of the total running time was spent in `oracle`; the number of calls of `oracle` is denoted I for `implicit enumeration` and R for `resolution_search`. A complete listing of the computer program is given in the Appendix.

$n = 20$			$n = 30$			$n = 40$		
I	R	I/R	I	R	I/R	I	R	I/R
2345	16728	0.14	6190	228221	0.03	70305722	5004465	14.05
15967	10201	1.57	841496	158603	5.31	1216079	763483	1.59
630	9815	0.06	117684	77576	1.52	15233194	33844363	0.45
8789	25453	0.35	632264	210149	3.01	1584197	5885578	0.27
18858	32706	0.58	453683	161672	2.81	6271851	511932	12.25
6102	27425	0.22	99099	1647189	0.06	5931796	1775280	3.34
6780	28124	0.24	877452	563456	1.56	25418525	5052669	5.03
13639	5786	2.36	1650591	603715	2.73	11420202	8216368	1.39
5867	5488	1.07	132808	247871	0.54	20996054	2424661	8.66
36652	14070	2.60	765818	123943	6.18	1386430	930165	1.49

Table 5: A bad branching strategy and a slow `obstacle`.

For a more meaningful comparison, one must consider the following two issues.

(i) BRANCHING STRATEGIES. Their effect on `implicit_enumeration` may be enormous; their effect on `resolution_search` remains to be seen.

(ii) IMPLEMENTATIONS OF OBSTACLE. Once u has been replaced by its extension u^*, Table 4 may require up to $n - 1$ additional calls of `oracle` to construct a minimal S such that $S \sqsubseteq u^*$ and

$$\begin{aligned} \texttt{oracle}(S) &\geq \texttt{record} && \text{if } \texttt{oracle}(u^*) \geq \texttt{record}, \\ \texttt{oracle}(S) &= \texttt{oracle}(u^*) && \text{if } \texttt{oracle}(u^*) < \texttt{record}, \end{aligned}$$

Sometimes a particular `oracle` may be exploited to skip at least some of these additional calls: we have already noted that this may be the case for the `oracle` defined through (2).

A call of `oracle(`u`)` may even permit, with only a slight amount of additional work, the construction of a minimal clause S such that $S \sqsubseteq u$ and `oracle(`S`)` $=$ `oracle(`u`)`. This is the case for the `oracle` used to prove claim (10) and this is the case for the `oracle` used to obtain Table 5. This last `oracle` is defined by a family $\mathcal{I}$ of clauses: the value of `oracle(`u`)` is the number of clauses C in $\mathcal{I}$ such that $C \sqsubseteq u$. (The $\mathcal{I}$ in each of our test problems consisted of $5n$ clauses with three literals per clause; these clauses were drawn independently at random for the uniform distribution.)

Proper consideration of these issues exceeds the scope of the present paper.

ACKNOWLEDGMENT

My interest in trying to design an alternative to implicit enumeration arose from enjoyable discussions with Ming Ouyang on the subject of branching strategies. I want to thank Ming for the pleasure and the stimulation that our conversations gave me.

Appendix: a computer program

```
#include <stdio.h>

#define MAXN 100
#define MAXNC 600

#define STAR -1
typedef int value;

int n, nc;
int inp[MAXNC][6];
int orcounter;
int obcounter;

value implicit_enumeration (char u[], value record, char best[]);
value resolution_search (char best[]);
value obstacle (char u[], value record, char s[]);

int depth (char what[], int d[]);
void update_depth (char c[][], int d[], int i);
void add_old_clause (int i, char c[][], int d[], int own[], int *pm);
void add_new_clause (char clause[], char c[][MAXN], int d[],
                     int own[], int *pm, char u[]);
int new_m (char r[], char c[][], int d[]);
void copy (char from[], char to[]);
void resolve (char what[], char with[]);

void report (char x[]);
void make_data (int example);
value oracle (char u[]);
```

```
void main (int argc, char *argv[])
{
    int i, example;
    value temp;
    char u[MAXN], best[MAXN];

    if (argc != 4) {
        printf ("SPECIFY THE NUMBER OF VARIABLES, THE NUMBER OF CLAUSES,
                AND THE EXAMPLE ! \n ");
                exit (1);
    }
    else {
        sscanf (argv[1], "%d", &n);
        sscanf (argv[2], "%d", &nc);
        sscanf (argv[3], "%d", &example);
    }
    printf ("%d VARIABLES AND %d CLAUSES; EXAMPLE %d\n\n", n, nc, example);
    make_data (example);

    printf ("IMPLICIT ENUMERATION:\n");
    orcounter = 0;
    for (i = 0; i < n; i++) u[i] = STAR, best[i]=0;
    temp = implicit_enumeration (u, oracle(best), best);
    printf ("optimal value = %d\n", temp);
    printf ("optimal solution\n");
    report (best);
    printf ("%d calls of oracle\n\n", orcounter);

    printf ("RESOLUTION SEARCH:\n");
    orcounter = 0;
    for (i = 0; i < n; i++) best[i]=0;
    temp = resolution_search (best);
    printf ("optimal value = %d\n", temp);
    printf ("optimal solution\n");
    report (best);
    printf ("%d calls of obstacle\n", obcounter);
    printf ("%d calls of oracle\n", orcounter);
}
```

```
value implicit_enumeration (char u[], value record, char best[])
{
    value bound, try;
    int j;

    bound = oracle (u);
    if (bound >= record) return record;

    for (j = 0; j < n; j++)
        if (u[j] == STAR) break;
    if (j == n) {
        copy (u, best);
        return bound;
    }

    u[j] = 0;
    try = implicit_enumeration (u, record, best);
    if (try == bound) {
        u[j] = STAR;
        return bound;
    }
    if (try < record) record = try;

    u[j] = 1;
    try = implicit_enumeration (u, record, best);
    if (try < record) record = try;

    u[j] = STAR;
    return record;
}
```

```
value resolution_search (char best[])
{
    value record, try;
    int i, j, m, oldm, d[MAXN], own[MAXN];
    char u[MAXN], s[MAXN], r[MAXN], c[MAXN][MAXN];

    record=oracle(best);
    m = 0;
    for (j = 0; j < n; j++) d[j] = n, u[j]=STAR;

    while (1) {
        try = obstacle (u, record, s);
        if (try < record) {
            copy (u, best);
            record = try;
        }

        while (1) {
            if (depth (s, d) >= m) {
                add_new_clause (s, c, d, own, &m, u);
                break;
            }

            copy (s, r);
            for (i = m - 1; i >= 0; i--)
                if (r[own[i]] != STAR) resolve (r, c[i]);
            for (j = 0; j < n; j++)
                if (r[j] != STAR) break;
            if (j == n) return record;

            oldm = m, m = new_m (r, c, d);
            add_new_clause (r, c, d, own, &m, u);
            j = own[m - 1];
            for (i = m; i < oldm; i++)
                if (c[i][j] == STAR) add_old_clause (i, c, d, own, &m);
            if (s[j] != STAR) break;
        }

        for (j = 0; j < n; j++)
            if (d[j] >= m) u[j] = STAR;
    }
}
```

```
value obstacle (char u[], value record, char s[])
{
    value bound;
    int j;
    int stack[MAXN];
    int stacktop;

    obcounter++;

    stacktop=0;
    for (j = 0; j < n; j++)
        if (u[j] != STAR) stack[stacktop++] = j;

    bound = oracle (u);
    if (bound < record) {
        for (j = 0; j < n; j++)
            if (u[j] == STAR) {
                u[j] = 0;
                bound = oracle (u);
                if (bound >= record) break;
                stack[stacktop++] = j;
            }
        if (bound < record) record = bound;
    }

    copy (u, s);
    while (stacktop) {
        j = stack[--stacktop];
        s[j] = STAR;
        if (oracle (s) < record) s[j] = u[j];
    }

    return bound;
}
```

```
int depth (char clause[], int d[])
{
    int i, j;

    i = 0;
    for (j = 0; j < n; j++)
        if (clause[j] != STAR && d[j] > i) i = d[j];
    return i;
}

void update_depth (char c[][MAXN], int d[], int i)
{
    int j;

    for (j = 0; j < n; j++)
        if (c[i][j] != STAR && d[j] > i) d[j] = i;
}

void add_old_clause (int i, char c[][MAXN], int d[], int own[], int *pm)
{
    copy (c[i], c[*pm]);
    update_depth (c, d, *pm);

    own[*pm] = own[i];
    (*pm)++;
}

void add_new_clause (char clause[], char c[][MAXN], int d[],
                     int own[], int *pm, char u[])
{
    int j;

    copy (clause, c[*pm]);
    update_depth (c, d, *pm);

    for (j = 0; j < n; j++)
        if (d[j] == *pm) break;
    own[*pm] = j, u[j] = 1 - u[j];
    (*pm)++;
}
```

```
int new_m (char r[], char c[][], int d[])
{
    int i, j, m;

    m = depth (r, d);
    for (j = 0; j < n; j++) d[j] = n;
    for (i = 0; i < m; i++) update_depth (c, d, i);
    return m;
}

void copy (char from[], char to[])
{
    int j;

    if (from != to)
        for (j = 0; j < n; j++) to[j] = from[j];
}

void resolve (char what[], char with[])
{
    int j;

    for (j = 0; j < n; j++) {
        if (what[j] == STAR)
            what[j] = with[j];
        else if (with[j] != STAR && with[j] != what[j])
            what[j] = STAR;
    }
}

void report (char x[])
{
    int j;

    for (j = 0; j < n; j++) printf ("%d", x[j]);
    printf ("\n\n");
}
```

```
void make_data (int example)
{
    int i;
    int a, b, c;

    srand48 (example);
    for (i = 0; i < nc; i++) {
        a = lrand48 () % n;
        do
            b = lrand48 () % n;
        while (b == a);
        do
            c = lrand48 () % n;
        while (c == a || c == b);
        inp[i][0] = a, inp[i][1] = b, inp[i][2] = c;
        inp[i][3] = lrand48 () % 2;
        inp[i][4] = lrand48 () % 2;
        inp[i][5] = lrand48 () % 2;
    }
}

value oracle (char u[])
{
    value bad;
    int i;

    orcounter++;

    bad = 0;
    for (i = 0; i < nc; i++) {
        if (u[inp[i][0]] != inp[i][3]) continue;
        if (u[inp[i][1]] != inp[i][4]) continue;
        if (u[inp[i][2]] != inp[i][5]) continue;
        bad++;
    }
    return bad;
}
```