RESOLUTION SEARCH AND DYNAMIC BRANCH-AND-BOUND

Saïd Hanafi

LAMIH - UMR CNRS n° 8530
Unité de Recherche Opérationnelle et d'Aide à la Décision
Université de Valenciennes et du Hainaut-Cambrésis
Le Mont Houy - B.P. 311 - 59304 Valenciennes Cedex – France

Said.Hanafi@univ-valenciennes.fr

Fred Glover

Hearin Center for Enterprise Science
School of Business Administration
University of Mississippi
University, MS 38677 USA

Fred.Glover@Colorado.EDU

January, 2001

Abstract

A novel approach to pure 0-1 integer programming problems called Resolution Search has been proposed by Chvatal (l997) as an alternative to implicit enumeration, with a demonstration that the method can yield more effective branching strategies. We show that an earlier method called Dynamic Branch-and-Bound (DB&B), due to Glover and Tangedahl (1976), yields the same branching strategies as Resolution Search, and other strategic alternatives in addition. Moreover, Dynamic B&B is not restricted to pure 0-1 problems, but applies to general mixed integer programs containing both general integer and continuous variables.

We provide examples comparing Resolution Search to enhanced variants. We also show the relation of these approaches to Dynamic B&B, suggesting the value of further study of this neglected approach.

Keywords: Branch-and-Bound; Dynamic Branch-and-Bound; Resolution Search; Mixed Integer Programming.

1. Introduction

A novel approach to pure 0-1 integer programming problems called Resolution Search (RS) has been proposed by Chvatal (l997) as an alternative to implicit enumeration, with a demonstration that the method can yield more effective branching strategies. We show that an earlier method called Dynamic Branch-and-Bound (DB&B), proposed by Glover and Tangedahl (1976), yields the same branching strategies as Resolution Search, and other strategic alternatives in addition. Moreover, DB&B is not restricted to pure 0-1 problems, but applies to general mixed integer programs containing both general integer and continuous variables. We provide examples comparing Resolution Search to enhanced variants. We also show the relation of these approaches to DB&B, suggesting the value of further study of this approach.

The RS and DB&B algorithms progressively restrict the set of feasible solutions that offer a possibility to improve on the best known solution. The methods can be viewed as generating an enumeration tree where the root corresponds to the original problem instance. The execution of the DB&B algorithm, in common with B&B methods generally, corresponds to a tree search starting from the root and exploring the descendant nodes until all terminal nodes are reached. Conversely, RS explores the B&B tree starting from terminal nodes until the root is reached. Nevertheless, we show how the methods can be reconciled within a common perspective.

2. LP-Based Branch and Bound

The *mixed integer programming* (MIP) problem consists of optimizing (Minimizing or Maximizing) a linear function subject to linear inequality and / or equality constraints, where some or all of the variables are required to be integral. The MIP problem can be expressed as follows

$$
\begin{array}{lllll}
 & \textit{Minimize} & z = cx & & \\
 & \text{Subject to} & A_i x \geq b_i & \text{for } i \in M = \{1, 2, \ldots, m\} & \\
\text{(MIP)} & & x_j \geq 0 & \text{for } j \in N = \{1, 2, \ldots, n\} & \\
 & & x_j \text{ integer} & \text{for } j \in P = \{1, 2, \ldots, p\} &
\end{array}
$$

where the input data are : the dimension m corresponding to the number of constraints, the number n corresponding to the number of decision variables x_j (with the first p integral and

the remainder continuous). The matrices $c(1 \times n)$, $A(m \times n)$, $b(m \times 1)$ are assumed without special structure. The general MIP problem is reduced to the *mixed binary integer program* (01-MIP) when all integer variables must equal 0 or 1, and becomes a *pure integer program* when p is equal to n. Numerous combinatorial optimization problems can be modeled as an MIP problem. Complexity results have not yet definitively identified the level of difficulty of these problems, but empirical findings suggest that the computational resources required to solve certain MIP problem instances can grow exponentially with the size of problem.

In order to establish terminology and conventions, we briefly sketch the well known branch-and-bound (B&B) algorithm in the form that is often applied for solving pure integer and mixed-integer linear programming problems. The branch-and-bound structure can be viewed as an enumeration tree where the root node corresponds to the original problem MIP. During the execution of a B&B algorithm, the tree grows by a branching process and shrinks by eliminating earlier branches that have been rendered *conditionally superfluous* by subsequent decisions.

The process of branching from a given node, denoted the parent node, creates two or more child nodes. Each of the problems at the child nodes is formed by adding constraints to the problem at the parent node, so that each feasible solution of the parent node problem is feasible for at least one of the child node problems. A *terminal* node of the B&B tree corresponds to a subproblem that can be solved directly, or that can be eliminated by *pruning* rules based on information about the likelihood that branching from this particular node will not lead to a feasible solution that improves the best known feasible solution. This information is generally deduced from a lower bound function on the nodes of B&B tree, structured so that the lower bound on a given node is no larger than the lower bound on its descendant. Clearly there is no need to create a subtree rooted at a terminal node. That is, in order to improve the best known feasible solution there is no reason to examine descendant nodes of a terminal node.

The goal of the B&B procedure is to find a terminal node of minimum cost. The process, starting with the root, successively expands some non-terminal node on the frontier of the tree until a terminal node is identified as an optimal solution. Expanding a node u means producing its children, thus identifying arcs in the tree from the node u to these children and generating the associated lower bounds. A node can be expanded only if it is the root of the tree or if it is a child of some node previously expanded. A frontier node is a node

that has been generated but not yet expanded. In other terms, the frontier of the current B&B tree is the set of nodes with no successors in this tree.

A simple way to make a branching move in B&B for solving an MIP problem is to partition the feasible region of node *u*, by creating two new nodes, associated with the two restrictions

$$x_j \le k \qquad \text{or} \qquad x_j \ge k + 1$$

where *k* is an integer value and *j* is a subscript of an integer variable ($1 \le j \le p$). In customary terminology, the tree is said to grow by creating two children at node *u*. One of these corresponds to the subproblem obtained by adding the constraint $x_j \le k$ to MIP(*u*) and the other by adding $x_j \ge k + 1$ (called *variable dichotomy* branching). The selection of the integer value *k* and the subscript *j* are called *branching strategies*. The efficiency of B&B algorithms depends heavily on the branching strategy used to select the next variable to branch on and its value.

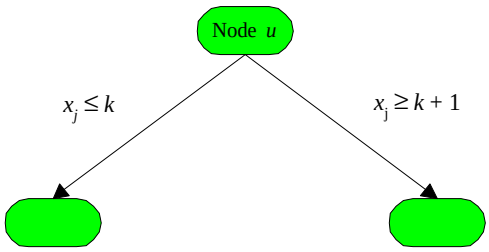

Figure 1 : Branching on variable dichotomy

A branch is defined by three parameters (*j*, *k*, *s*) where *j* is an index of the variable branched on; *k* is the "branching value" and *s* indicates the sense of inequality. First observe that the branching constraint $x_j \le k$ (or $x_j \ge k$), where $1 \le j \le p$ and *k* is a nonnegative integer value, can be expressed by

$$\text{sign}(h)x_j \le h$$

where *h* is an integer value and sign(*h*) = 1 if $h \ge 0$ and sign(*h*) = -1 if $h < 0$. Moreover, since all the branching values *k* are positive, we can drop the parameter *s* by extending the domain value to the negative space. More precisely, the branching constraint $x_j \le k$ will be parameterized by a positive integer value *h* ($h \in Z^+$) and the branching constraint $x_j \ge k$ will be parameterized by a negative integer *h* ($h \in Z^-$). Henceforth, a branch will be defined by a pair (*j*, *k*) where *j* is the branching variable and *k* is the branching value.

***Remark* 1 :** In the 0-1 case, one branch suffices to fix a variable, therefore a node can be represented by vector of dimension n. In the general integer case, it is necessary to have two branches on the same variable to fix it except where a branch forces a variable to an upper or lower bound. In this case, a node will be represented by a vector of dimension $2n$. Therefore, the space required to represent a node in discrete case is dn where d is the maximum domain size of any variable.

At any iteration during the B&B process a node of the tree is completely defined by the antecedent branches that lead to it and impose constraints on variables, understanding that the root node represents the original problem. Adapting notation originally introduced in Chvatal for solving the pure 0-1 integer program, we identify a node u as an element of $(Z \cup \{*\})^p$ where

$u_j \geq 0 \quad \Leftrightarrow$ Add the branch $x_j \leq u_j$

$u_j < 0 \quad \Leftrightarrow$ Add the branch $x_j \geq -u_j$

$u_j = * \quad \Leftrightarrow$ No branch on the variable x_j.

For instance, the node u = (*, 0, -1, *, 3, -9) corresponds to adding the set of constraints $\{x_1$ free, $x_2 \leq 0$, $x_3 \geq 1$, x_4 free, $x_5 \leq 3$, $x_6 \geq 9\}$.

Let $u = (u_1, u_2, \ldots, u_p)$ and $v = (v_1, v_2, \ldots, v_p)$ be two elements of $(Z \cup \{*\})^p$. We call v an *extension* of u, denoted by $u \, \pi \, v$, if :

$$v_j \leq u_j \text{ whenever } u_j \neq * \text{ for } j = 1, \ldots, p.$$

Trivially, π is a partial order $(Z \cup \{*\})^p$.

Another fundamental ingredient of B&B is a bounding procedure which computes a lower bound on the optimal value of the subproblem MIP(u) defined at node u. The bound function can be used to guide the order of generating the nodes of the B&B tree and / or to determine that certain nodes are terminal. Lower bounds are provided by relaxation or duality techniques. Given a lower bound function z, a node u is a terminal node if : 1) its lower bound $z(u)$ is greater than or equal to the value of a known feasible solution z^*; or 2) the subproblem MIP(u) has been solved optimally (including the infeasibility case where MIP(u) is demonstrated not to have a feasible solution). Most commercial B&B procedures use the LP-relaxation to compute the bound function.

Formally, let u and LP(u) denote the LP-relaxation of MIP(u), where all variables are allowed to be continuous, augmented by bounds on the integer variables and on the objective function value, which can be defined as follows :

(LP(u))

Minimize	$z = cx$	
Subject to	$A_i x \geq b_i$	for $i \in M$;
	$x_j \geq 0$	for $j \in N$;
	$cx \leq z^*$	
	$\text{sign}(u_j)x_j \leq u_j$	for $j \in P$ and $u_j \neq *$;

with

$z(u) = cx(u)$	if an optimal solution $x(u)$ of LP(u) exists;
$z(u) = +\infty$	if LP(u) is infeasible problem;
$z(u) = -\infty$	if LP(u) is unbounded problem.

where $z^* = cx^*$ is the value of the best known solution x^*, also called "the incumbent solution". Given a node u, in the evaluation step of B&B algorithm we solve the LP-relaxation LP(u) to determine an optimal solution $x(u)$ to LP(u) and its associated objective value $z(u)$. (By convention, $z(u) = \infty$ if LP(u) is infeasible. If the relaxation is solved by a dual algorithm, then the solution process can be terminated early if the dual value reaches or falls below z^-.)

A B&B method can be interpreted from a graph perspective. Given an MIP to solve, let G denote a digraph where the set of nodes, denoted by Γ, corresponds to the set of all possible nodes for an MIP, there is an arc between two nodes u and v if the node u can be reached by adding one branch to the node u. In a B&B procedure, a node u is partitioned into two distinct sets of variables u^- and u^+. The set u^- is contains the variables branched on from the root to this node, often called *past variables*. The set u^+ contains variables not branched on, called *future variables*. Let $\delta^{+1}(u)$ denotes the set of successors of the node u. Note that the set $\delta^{+1}(u)$ is empty if and only if u^+ is empty, and a branch move consists of transferring a variable from u^+ to u^-. (i.e. $\delta^{+1}(u) = \{ v : v^- = u^- + \beta; v^+ = u^+ - \beta \text{ with } \beta \in u^+\}$). Let $\delta^{+*}(u)$ be the set of descendants of u, which gives $\Gamma = \delta^*((*, *, \ldots, *))$. Similarly, a node v is a predecessor of a node u if the node u can be obtained from node v by adding one branch. Accompanying this, let $\delta^{-1}(u)$ denotes the set of predecessors of the node u and $\delta^{-*}(u)$ denotes the set of ascendants of u in graph G. Observe that a predecessor of each node u exists only if the set u^- is not empty. (i.e. $\delta^{-1}(u) = \{ v : v^- = u^- - \beta; v^+ = u^+ + \beta \text{ with } \beta \in u^-\}$).

The B&B approaches for solving an MIP consist of finding a tree with root (*, …, *) all of whose nodes on the frontier of this tree are terminal. The frontier of a tree π is the set $Fr(\pi) = \{ u \in \pi : \text{not } (\delta^{+1}(u) \subseteq \pi)\}$. A generic B&B algorithm for solving the *MIP* problem instance is given below.

Generic *Branch-and-Bound Algorithm*

Let $\pi = (*, \ldots, *)$;

While all nodes in $Fr(\pi)$ are non-terminal ***do***

- Select a non-terminal node u from $Fr(\pi)$.
- Add a new branch (u, v) to the tree π with $v \in \delta^{+1}(u)$ and $v \notin \pi$.
- Reduce the tree π by the rule : $\pi := \pi - \{v \in Fr(\pi) : v \text{ is a } terminal\ node\}$

Endwhile.

We may alternately formulate a generic B&B algorithm in the following way, where Γ^+ corresponds to the set of unvisited nodes, a subset of Γ.

Generic Branch-and-Bound Algorithm (Alternate Representation)

Let $\pi = \{MIP\}$; $\Gamma^+ = Fr(\pi)$;

While $(\Gamma^+ \neq \varnothing)$ ***do***

- Select a node u from $Fr(\pi) \cap \Gamma^+$.
- ***If*** the node u is terminal ***then***
  - $\Gamma^+ := \Gamma^+ - \delta^{+*}(u)$;
- ***Else***
  - Add a branch to the tree $\pi := \pi + (u, v)$; with v in $\delta^{+1}(u) \cap \Gamma^+$;
  - $\Gamma^+ := \Gamma^+ + \{v\}$;
- ***Endif*;**
- Reduce Γ^+ and/or π by the rules
  - $\Gamma^+ := \Gamma^+ - \{w : \delta^{+1}(w) \subseteq \Gamma^+\}$;
  - $\Gamma^+ := \Gamma^+ - \{v \text{ in } Fr(\pi) : v \text{ is a } terminal\ node\}$
  - $\pi := \pi - \{v \text{ in } Fr(\pi) : v \text{ is a } terminal\ node\}$

Endwhile.

***Remark* 2 :** The condition that all nodes in $Fr(\pi)$ are non-terminal is equivalent to the disjunction ($\Gamma^+ \neq \varnothing$) or ($\delta^{+*}(\pi) \supseteq X$).

In each iteration of a B&B algorithm, a visited node non-terminal *u*, situated at the frontier of the current B&B tree π, and an unvisited node *v* adjacent to *u*, are chosen to grow the current tree. Different rules can be used to decide the order in which nodes in the $Fr(\pi)$ are branched on. The *Depth-First* rule is used to branch on the most recently generated node. The use of a Depth-First strategy tends to minimize the number of nodes that are maintained at a given time, but may explore some nodes unnecessarily. The *best-first* rule is used to branch on a node having a smallest lower bound. The best-first rule tends to minimize the total number of nodes created by B&B procedure up to a given time, but also may need to maintain a large set of nodes that can challenge computer memory requirements.

***Remark* 3 :** A best known solution x^* is maintained and updated whenever a feasible terminal node *u* is reached during the search process, and the last x^* is optimal.

Example 1 : Consider the following example with three binary variables :

$$Min\ z = x_1 + 2x_2 + 3x_3 \quad \text{s.t.} \quad x_1 + 2x_2 + x_3 = 2 \text{ with } x_1, x_2, x_3 \in \{0, 1\}.$$

The graph *G* induced by this problem is shown in Figure 2. Arcs in blue constitute the tree explored by a classical B&B. Terminal nodes are depicted in red, feasible nodes are depicted in blue, and unvisited nodes are depicted in white.

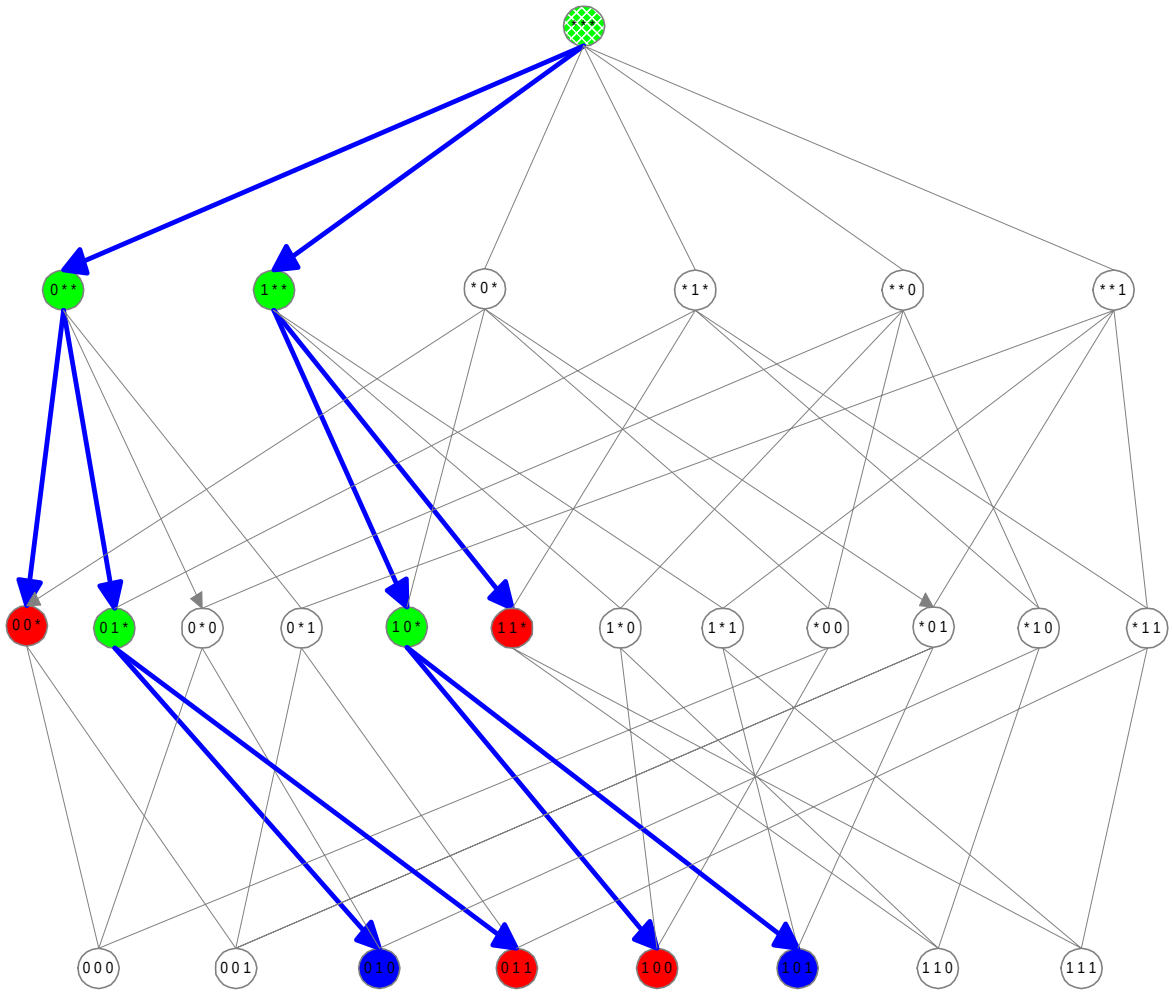

Figure 2 : B&B Tree

Example 2 : **Van der Waerden's Theorem**

This example is a particular case of the celebrated Van der Waerden's theorem first proven in 1927. Its topological proof can be found in Furstenberg and Weiss (1971) or Graham et al (1990). This theorem has a large number of applications in combinatorics and other fields. Van der Waerden's theorem states that for any partitioning of the set N into k sets, at least one of the subsets contains arbitrarily long arithmetic progressions. In our numerical experiments we consider the same problem as in Chvatal (1997) defined as follows.

For every partition of the set $N = \{1, 2, \ldots, 18\}$ into two sets A_0 and A_1 (i.e. $N = A_0 \cup A_1$ and $A_0 \cap A_1 = \varnothing$),

i) A_0 contains an arithmetic progression with three terms or else

ii) A_1 contains an arithmetic progression with four terms.

Observe that a partition A_i, for $i = 0,1$ of N can be defined by a given a vector x in $\{0, 1\}^{18}$ such that $x_j = i$ if and only if $j \in A_i$. Letting cx be an arbitrary objective function, a (0-1 IP) formulation of the Van der Waerden's theorem is as follows:

$$
\text{(0-1 IP)} \quad \left|
\begin{array}{ll}
\textit{Minimize } z = cx & \\
\text{Subject to} & \\
\quad x_a + x_{a+d} + x_{a+2d} \geq 1 & \forall\ a, d \text{ such that } a + 2d \leq 18, \\
\quad x_a + x_{a+d} + x_{a+2d} + x_{a+3d} \leq 3 & \forall\ a, d \text{ such that } a + 3d \leq 18, \\
\quad x \in \{0, 1\}^{18} &
\end{array}
\right.
$$

Van der Waerden's Theorem states that the optimum value of (0-1 IP) equals $+\infty$ (i.e. the problem (0-1 IP) has no zero-one solution).

B&B Execution with solver Cplex version 6.5.2

Node	Nodes Left	Objective	IInf	Best Integer	Cuts/ Best Node	ItCnt	Gap
0	0	128.2500	18		128.2500	20	
1	1	124.0000	15		126.7500	33	
2	2	120.7500	13		126.7500	41	
3	3	113.3636	14		126.7500	52	
4	4	104.0000	11		126.7500	64	
5	4	infeasible			126.7500	73	
6	3	121.2500	15		123.6667	93	
7	4	112.8000	11		123.6667	111	
8	5	infeasible			123.6667	123	
9	4	102.0000	8		123.6667	131	
10	5	infeasible			123.6667	137	
11	4	infeasible			123.6667	142	
12	3	115.0000	15		121.0000	156	
13	4	112.5000	13		121.0000	160	
14	5	106.8182	12		121.0000	171	
15	6	infeasible			121.0000	181	
16	5	infeasible			121.0000	187	
17	4	116.0000	9		119.5000	204	
18	5	115.3333	10		119.5000	211	
19	6	114.5000	7		119.5000	212	
20	7	infeasible			119.5000	221	
21	6	infeasible			119.5000	229	
22	5	119.5000	10		119.5000	232	
23	6	106.0000	9		116.0000	245	
24	6	infeasible			116.0000	256	
25	5	104.5000	12		115.3333	277	
26	6	infeasible			115.3333	285	
27	5	103.5000	12		115.3333	286	
28	6	infeasible			115.3333	289	
29	5	infeasible			115.3333	300	
30	4	103.3333	12		115.0000	311	
31	5	95.5000	9		115.0000	317	
32	6	infeasible			115.0000	319	
33	5	infeasible			115.0000	329	
34	4	108.5000	8		115.0000	344	
35	5	infeasible			115.0000	358	
36	4	infeasible			115.0000	372	
37	3	111.8000	13		113.2857	378	

```
     38     4    infeasible                          113.2857      388
     39     3    infeasible                          113.2857      404
     40     2    infeasible                          112.0000      423
     41     1      109.5000     9                    109.5000      431
     42     2    infeasible                          108.0000      441
     43     1       99.5000    11                    101.3333      452
     44     2    infeasible                          101.3333      457
     45     1    infeasible                          101.3333      466
     46     0    infeasible                                        480
No solution exists.
```

3. **Resolution Search**

In a customary B&B approach, when a terminal node is encountered, the backtracking is limited to the frontier of the current B&B tree.

Resolution search (RS) keeps a set of terminal nodes to record information about the portion of the search that has been eliminated and also to store the current tree being considered by the procedure. To avoid redundant search, RS retains a fixed order on branched variables appearing in terminal nodes stored. A terminal node is simply a node u such that $\delta^{+*}(u)$ has been eliminated from consideration. More precisely, a terminal node u rules out all solutions x such that :

$$x \notin X(u) = X(u_1) \cap \ldots \cap X(u_p) \qquad (1)$$

where $X(u_i) = \{x \in X : sign(u_j)\, x_j \leq u_j\}$ if $u_j \neq *$, otherwise $X(u_j) = X$, for $j = 1, \ldots, p$ where X is the set of feasible solutions of the MIP (i.e. $X = \{x : Ax \geq b, x \geq 0$ and x_j integer for $j \in P\}$). Therefore, $X(u)$ is the set of descendants of the node u which are in X (i.e. $X(u) = \delta^{*}(u) \cap X$).

The special terminal node is the root (*, …, *) of graph G. As soon as this root becomes a terminal node, the process can be stopped, it follows that no solution exists for improving the best known solution (if one exists).

The naïve approach that consists of accumulating all terminal nodes encountered during the process of search suffers from an exponential space complexity. To keep the space complexity polynomial, some terminal nodes need to be removed during the search. Nevertheless to assure the termination of the process, ordering conditions are imposed on the variables to avoid cycling.

The set of terminal nodes can be reduced by applying the following simple rules which extend Chvatal's rule to more general problems beyond the 0-1 IP case:

(a) if u and v are terminal nodes such that u is descendant of v then u is deleted;

(b) if u, v are terminal nodes such that u and v can be reached by a move that adds one branch move from a node w then the node w replaces the nodes u and v.

In the following, we give an extension of Chvatal's rule b) above.

3.1 Resolvent Operator

First observe that for any branching variable j associated with a given terminal node u (i.e. $u_j \neq *$), the expression (1) is logically equivalent to

$$x \in X(u - \{u_j\}) \Rightarrow x \in X(\bar{u}_j) \qquad (2)$$

where $\bar{u}_j$ is the complement of a u_j defined by $\bar{u}_j = - u_j - 1$ (by convention, we set $\bar{u}_j = u_j$ if $u_j = *$). Clearly There are many different ways of representing a given terminal node. This provides freedom in the choice of the variable appearing in the conclusion of the implication (2).

Different ways may be used to derive new terminal nodes. A simple approach consists of generating a new terminal node from the current set of terminal nodes already visited. In Chvatal, nodes u and v are called *clashing* if there is exactly one node w such that u and v are children of w. The node w is called the *resolvent* of the clashing nodes u and v. Formally, nodes u and v are clashing if there is exactly one subscript j such that

i) $u_{j'} = v_{j'}$ for all $j' \neq j$;

ii) $u_j \neq *$ and $v_j \neq *$;

iii) $\{x_j : \text{sign}(u_j)x_j \leq u_j\} \cup \{x_j : \text{sign}(v_j)x_j \leq v_j\} = Z$.

It is easy to see that the last condition iii) can be expressed as

iii') $u_j + v_j + 1 = 0$.

The resolvent of clashing clauses u and v, denoted by $u \nabla v = w$, is defined by $w_{j'} = u_{j'}$ for all $j' \neq j$; and $u_j = *$, where $u_j + v_j + 1 \geq 0$ and $u_j \neq * \neq v_j$.

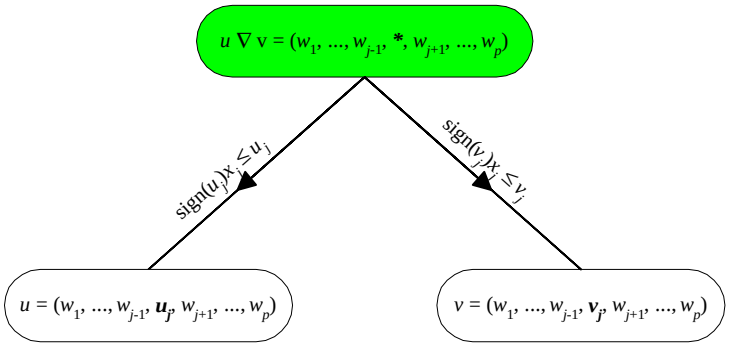

Figure 3 : The clause $u \nabla v$ is the resolvent of clashing clauses u and v

In other terms, if $u + \beta$ and $v + \overline{\beta}$ are terminal nodes then $u + v$ is a also valid terminal node. In fact, we have

$$x \notin \delta(u + \beta) \Leftrightarrow x \notin \delta(u) \cap \delta(\beta) \Leftrightarrow (x \notin \delta(u) \Rightarrow x \in \delta(\beta)) \qquad \text{(a)}$$

$$x \notin \delta(v + \overline{\beta}) \Leftrightarrow x \notin \delta(v) \cap \delta(\overline{\beta}) \Leftrightarrow (x \notin \delta(\overline{\beta}) \Rightarrow x \in \delta(v)) \qquad \text{(b)}$$

(a) and (b) together imply

$$(x \notin \delta(u) \Rightarrow x \in \delta(v)) \Leftrightarrow x \notin \delta(u) \cap \delta(v) \text{(c)}$$

The resolvent operator permits the generation of new terminal nodes by combining nodes already visited. Since in resolution search combined nodes are situated in the same current tree, the node resulting by the resolvent operator also belongs to this tree. On the other hand, in dynamic B&B combined nodes do not necessarily belong to the same tree, which makes possible to generate new terminal nodes not met in the past. For example if u = (0, *, 0) and v = (*, 0, 1) then we define the resolvent of nodes u and v, as $u \nabla v$ = (0, 0, *). This can not be generated by the resolvent used by Chvatal since these nodes belong to different trees.

More generally, if u and v are terminal nodes, then $u \nabla v = (u - \overline{v}) + (v - \overline{u})$ is also a terminal node, where, the complement of a node u denoted $\overline{u}$ is defined as follows :

$\overline{u}_j = -u_j - 1$ if $u_j \neq *$;

$\overline{u}_j = u_j$ otherwise.

In other terms, if $w = u \nabla v$ then we have,

$X(w_j) = X(u_j) \cap X(v_j)$ if $X(u_j) \cap X(v_j) \neq \varnothing$

$X(w_j) = X$ otherwise.

***Remark* 4 :** The resolvent operator is also used in classical B&B approaches by combining nodes situated in the same current tree. In fact, in these approaches the operator is particularly used to reduce the space needed to represent the current B&B tree.

3.2 The Obstacle Function and Its Extension

At each iteration, from the current node u, the Resolution Search algorithm generates two nodes u^+ and u^- by calling an *obstacle function* which is applied in two phases (first introduced in Chvatal): a *waxing* phase and a *waning* phase. In the waxing phase, the process moves from u to u^+ by adding branches to the current node u, until the node u^+ which is a descendant of u ($u \pi u^+$) becomes a terminal node. Recall that the node u is terminal if the feasible solution set of LP(u) is empty or MIP(u) is solved (i.e. the optimal solution of LP(u) is an integer solution or LP(u) is unbounded for optimality). In the case where LP(u^+) solves MIP(u^+) and the resulting solution improves the incumbent solution, then the value of z^* is updated. In the waning phase, the process moves from u^+ to u^- by dropping branches from the node u^+ to obtain u^- ($u^- \pi u^+$), as long as u^- is a terminal node. Similarly, in the case where LP(u^-) solves MIP(u^-) and the solution improves the incumbent solution, then the same type of update occurs.

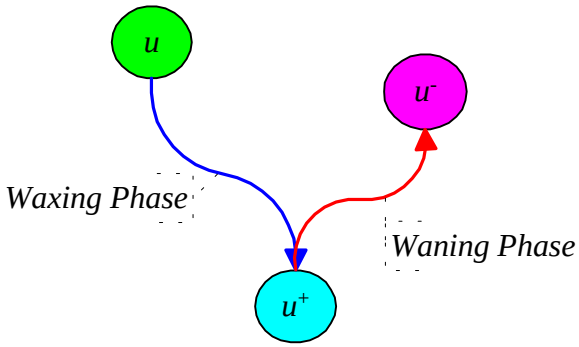

Figure 4 : An intuitive representation of obstacle(u, u^+, u^-) function

Given a node u the function *Obstacle*(u, u^+, u^-) constructs two terminal nodes u^- and u^+ such that the node u^+ is a descendant of node both nodes u and u^- ($u \pi u^+$ and $u^- \pi u^+$). An implementation of this function is described below.

Function *Obstacle*(u, u^+, u^-)

{

1. ***Waxing phase*** : Move from the node u to a descendant node u^+ of u by adding branches until u^+ becomes a terminal node.

   $u^+ = u$;

   do{

   Let $x(u^+)$ be an optimal solution *for LP*(u^+); and $z(u^+)$ its associated objective value.

   Choose a subscript j^* such that

   $$j^* \in Argmax\{min\{f_j, 1 - f_j\} : f_j = x_j(u^+) - \lfloor x_j(u^+) \rfloor \text{ and } f_j \neq 0 \text{ for } j \in P\}$$

   if (j^* exists) $u^+_{j^*} = -\lceil x_{j^*}(u^+) \rceil$; // or $\lfloor x_{j^*}(u^+) \rfloor$

   } while ($z(u^+) < z^*$) and (j^* exists);

   $z^* = min\{z(u^+), z^*\}$;

2. ***Waning phase*** : Move from the node u^+ to an ascendant node u^- by dropping branches from the node u^+ as long as the node u^- is terminal.

   $u^- = u^+$; $v = u^+$;

   do{

   Let $r(u^-)$ be a vector of reduced costs for each of the variables LP(u^-).

   Choose a subscript j^* such that

   $$j^* \in Argmin\{r_j(u^-) : r_j(u^-) \neq 0 \text{ and } v_j \neq * \text{ for } j \in P\}$$

   if (j^* exists)**{**

   $u^-_{j^*} = *$; $v_{j^*} = *$;

   if ($z(u^-) < z^*$) $u^-_{j^*} = u^+_{j^*}$;

   }

   } while (j^* exists);

}

***Remark* 5 :** In both phases above, the case where the LP-relaxation solves the integer problem at a given node has been taken in account.

- **Choice of the Branching Criterion in the obstacle function**

In the waxing phase several methods are available to choose the variable for branching. A simple choice is to select the most fractional variable x_g where :

$$g \in Argmax\{min\{f_j, 1 - f_j\} : \text{for } j \in P\}$$

and where $f_j = x_j - \lfloor x_j \rfloor$ is the associated fractional value. Other rules are based on the idea of estimating the cost of forcing the variable x_j to become integer.

In the waning phase branches are dropped (which frees some of the variables) according to rules based on identifying *influential choices* in retrospect, at the point where backtracking would occur (see Section 4).

***Remark* 6 :** The waning phase and the waxing phase can be accelerated by saving some calls of the Oracle function. This can be accomplished by adding (or dropping) more than one branch at each iteration in the waning phase (or waxing phase).

3.3 Family Updating to Assure Convergence

Trivially, since each terminal node eliminates a portion of the search space from consideration, the naive approach that maintains all terminal nodes visited during the process of search will terminate if the search space is bounded. However, the approach that removes terminal nodes during the search may not assure the convergence of the process. Therefore, to assure the termination when terminal nodes are removed, Chvatal's resolution search imposes a total order on the variables branched on. More precisely, for pure 0-1 IP problems with *n*-dimensional vectors, Chvatal maintains a set of terminal nodes of at most *n* nodes, called a *path-like* family. At each iteration, a new terminal node is generated so that the portion of the search space eliminated by this node includes the portions that were eliminated by the terminal nodes that are removed from the *path-like* family. Consequently the size of the eliminated search space increases monotonically which assures the convergence of resolution search.

The *path-like Family of* Chvatal (1997) can be easily extended to the MIP setting. The members of the path-like family F are enumerated as (u^i, β^i) for i=1, …,$|F|$, where u^i is a terminal node and β^i is a branch appearing in the node u^i such that

- the branch β^j appears in u^i if and only if $i = j$,
- if the branch $\overline{\beta}^j$ the complement of β^j appears in u^i then $i > j$,
- if a branch β appears in u^i and its complement $\overline{\beta}$ appears in u^j then $\beta = \beta^i$ or $\overline{\beta} = \beta^j$.

With each path-like family F we associate the node $u(F)$ defined as follows

$$u(F) = \cup_{i=1,|F|} (u^i - \beta^i + \overline{\beta}^i)$$

At each iteration during the execution of resolution search the node $u(F)$ corresponds to the current node u. The node $u(F)$ also corresponds to the current tree π defined as follows (see figure 5) :

$$\pi_0 = \varnothing$$

$$\pi_{i+1} = \begin{cases} \pi_i + \beta_i & \text{if } \beta_i \in F^- \\ \pi_i + \overline{\beta_i} & \text{if } \beta_i \in F^0 \end{cases} \qquad \text{for } i = 1,...,k-1$$

where $F^- = \cup_{i=1,|F|} (u^i - \beta^i)$, $F^0 = \cup_{i=1,|F|} \{\beta^i\}$ and $k = |u(F)|$.

Each terminal node of the current family F is attached to the current tree π by only one branch. More precisely, terminal nodes are enumerated such that for each node u^i in F we have $u^i = v^i + \beta^i$ with $v^i \subseteq \pi_i$.

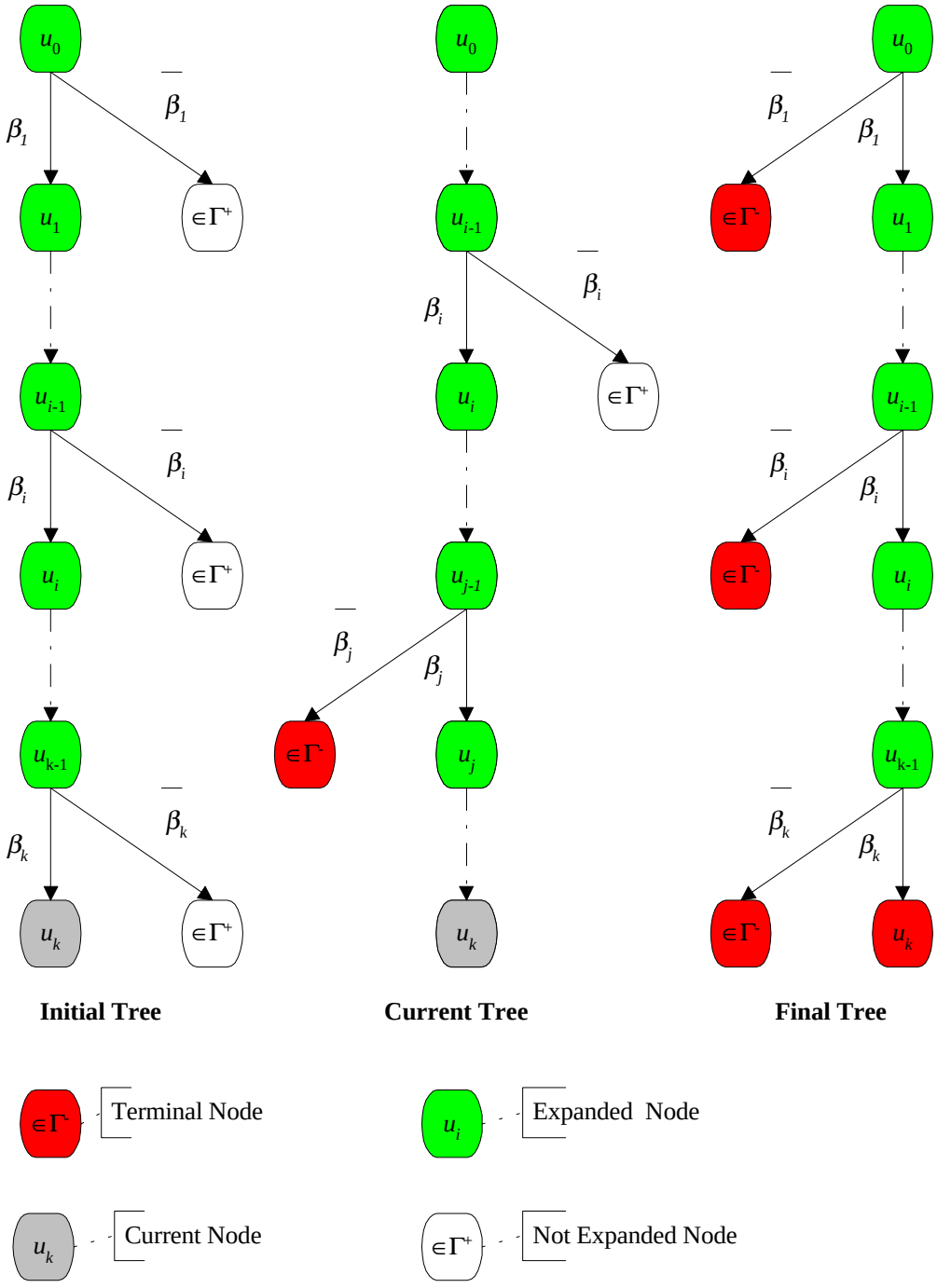

Figure 5 : States of Resolution Search

At each iteration of the Resolution Search algorithm, the node u^- constructed by *Obstacle*(u, u^+, u^-) is used to update the family F.

Update the *path-like Family* (F, u^-)

{

repeat

if ($u^- \not\subset u(F)$)**{**

choose a branch β in u^- such that $\beta \notin u(F)$ and $\overline{\beta} \notin u(F)$;

$F = F + \{u^-\}$, $u^{|F|} = u^-$ and $\beta^{|F|} = \beta$;

} else {

$w = u^-$;

for ($i = |F|, |F|\text{-}1, \ldots, 3, 2, 1$) **if** ($\overline{\beta}^i \in w$) $w = w \nabla u^i$;

if ($w \neq (*, *, \ldots, *)$){

find the smallest k such that w is a descendant of $u(F^k)$;

choose a branch β appearing in w but not in $u(F^{k-1})$;

replace u^k by w and set $\overline{\beta}^k = \beta$;

remove from $u^{k+1}, u^{k+2}, \ldots, u^{|F|}$ every node that includes β;

}

}

until ((*, *, …, *) in F) or (β in u^-);

}

Remark 7 : The branch β chosen not to lie in $u(F)$ ($\beta \notin u(F)$ and $\overline{\beta} \notin u(F)$) corresponds to the branch that is not covered by the family F (i.e. $\beta \notin F$ and $\overline{\beta} \notin F$). We say that a family of nodes *covers* those variables x_j if at least one of x_j and $\overline{x}_j$ belongs to at least one of the nodes in that family.

Remark 8 : The condition $u^- \subseteq u(F)$ means that the node $u(F)$ is a descendant of the node u^- by permuting the branches in $u(F)$. In executing this permutation some terminal nodes can be removed from F if they become "irrelevant" in the sense that their antecedents no longer match the node $u(F)$.

3.4 Resolution Search Algorithm

The resolution search method can be viewed as a crossing of the search tree with the root u = (*, *, …, *) which corresponds to the original MIP problem, as follows. The procedure maintains a family F of terminal nodes. For each node u in F, every feasible solution of the original MIP which is a descendant of u cannot strictly improve the current incumbent x^*. Initially, we start from a given node u in the B&B tree corresponding to the current node, with an empty F and an arbitrary incumbent solution x^*. If such a solution is not yet known, take any clause for x^*, for example $x^* = u$ is valid.

At each iteration, from the current node u, the algorithm generates two terminal nodes u^+ and u^- by calling the *obstacle function*. The terminal node u^+ becomes the new incumbent solution x^* if u^+ is an improving feasible solution and the terminal node u^- is added to F. Then, the algorithm generates a new node that is a descendant of no terminal node in F. This node becomes the new current node and the process is repeated until the whole initial tree has been fathomed (which corresponds to adding the node (*, *, …, *) to F), at which point the vector x^* can be returned as a solution of the search problem.

```
Resolution Search Method(x*, u in S)
{
    z* = cx*;
    F = ∅;
    While ((*, *, ...., *) ∉ F)){
        try = Obstacle(u, u+, u-);
        if (u+ ∈ X) and (try < z*) { x* = u+; z* = try;}
        Update the path-like Family (F, u-);
        u = u(F);
    }
}
```

***Remark* 9 :** The updating process of resolution search does not take account of the notion of influence, and also does not exploit information derived by finding a new best solution. Each iteration of resolution search, in the form proposed by Chvatal, tries to go deeply to fix as many variables as possible. If a new best solution is found, then for each node u in F, we can move from u to u^- by dropping branches from the node u as long as u^- is a terminal node and replace u by u^- in F. In other words, it is more interesting to obtain terminal nodes close to the root. Once a new best solution is found, it is possible to apply a waxing phase from each terminal node in F to obtain a new ascendant terminal node, which replaces it.

Example 2 : (continued)

To prove Van der Waerden's theorem for the 18-point example, the choices of branching proposed in Chvatal's resolution search need 29 calls of the obstacle function and 199 calls of the oracle function (see Table 1 below given by Chvatal and generated also by our implementation).

Iteration	u to be extended	u^-	Resulting family F
1	(*,*,*,*,*,*,*,*,*,…)	$x_1x_2x_3$	$\{x_1x_2x_3\}$
2	(0,0,1,*,*,*,*,*,*,…)	$x_1x_4x_5$	$\{x_1x_2x_3,\ x_1x_4x_5\}$
3	(0,0,1,0,1,*,*,*,*,…)	$x_1x_2x_4\,\bar{x}_5$	$\{x_1x_2x_3,\ x_1x_2x_4\}$
4	(0,0,1,1,*,*,*,*,*,…)	$x_1x_5x_6$	$\{x_1x_2x_3,\ x_1x_2x_4,\ x_1x_5x_6\}$
5	(0,0,1,1,0,1,*,*,*,…)	$x_1x_2\,\bar{x}_4\ \bar{x}_6$	$\{x_1x_2x_3,\ x_1x_2x_4,\ x_1x_2x_5,\ x_1x_2\,\bar{x}_4\ \bar{x}_6\ \}$
6	(0,0,1,1,1,0,*,*,*,…)	$x_1x_2\,\bar{x}_5\,x_7$	$\{x_1x_2x_3,\ x_1x_2x_4,\ x_1x_2x_5,\ x_1x_2\,\bar{x}_4\ \bar{x}_6\ ,\ x_1x_2\,\bar{x}_5\,x_7\}$
7	(0,0,1,1,1,0,1,*,*,…)	$x_1\,\bar{x}_4\ \bar{x}_5\ \bar{x}_7$	$\{x_1x_2,\ x_1\,\bar{x}_4\ \bar{x}_5\ \bar{x}_7\ \}$
8	(0,1,*,1,1,*,0,*,*,…)	$x_1\,\bar{x}_2\ \bar{x}_5\,x_7$	$\{x_1x_2,\ x_1\,\bar{x}_4\ \bar{x}_5\ \}$
9	(0,1,*,1,0,*,*,*,*,…)	$x_1x_3x_5$	$\{x_1x_2,\ x_1\,\bar{x}_4\ \bar{x}_5\ ,\ x_1x_3x_5\}$
10	(0,1,1,1,0,*,*,*,*,…)	$\bar{x}_2\ \bar{x}_3\ \bar{x}_4\,x_6$	$\{x_1x_2,\ x_1\,\bar{x}_4\ \bar{x}_5\ ,\ x_1x_3x_5,\ \bar{x}_2\ \bar{x}_3\ \bar{x}_4\,x_6\}$
11	(0,1,1,1,0,1,*,*,*,…)	$\bar{x}_2\ \bar{x}_3\ \bar{x}_4\ \bar{x}_6$	$\{x_1x_2,\ x_1\,\bar{x}_4\ ,\ x_1x_3x_5\}$
12	(0,1,1,0,0,*,*,*,*,…)	$x_1x_4x_5$	$\{x_1x_2,\ x_1\,\bar{x}_4\ ,\ x_1x_5\}$
13	(0,1,*,0,1,*,*,*,*,…)	$x_1x_4x_6$	$\{x_1x_2,\ x_1\,\bar{x}_4\ ,\ x_1x_5,\ x_1x_4x_6\}$
14	(0,1,*,0,1,1,*,*,*,…)	$x_1x_4\,\bar{x}_5\ \bar{x}_6$	$\{x_1\}$
15	(1,*,*,*,*,*,*,*,*,…)	$x_2x_3x_4$	$\{x_1,\ x_2x_3x_4\}$
16	(1,0,0,1,*,*,*,*,*,…)	$x_2x_3x_5$	$\{x_1,\ x_2x_3x_4,\ x_2x_3x_5\}$
17	(1,0,0,1,1,*,*,*,*,…)	$\bar{x}_1x_2x_6$	$\{x_1,\ x_2x_3x_4,\ x_2x_3x_5,\ \bar{x}_1x_2x_6\}$
18	(1,0,0,1,1,1,*,*,*,…)	$x_2x_3\,\bar{x}_5\ \bar{x}_6$	$\{x_1,\ x_2x_3,\ \bar{x}_1x_2x_6\}$
19	(1,0,1,*,*,*,*,*,*,…)	$x_2\,\bar{x}_3\ \bar{x}_6\,x_7$	$\{x_1,\ x_2x_3,\ \bar{x}_1x_2x_6,\ x_2\,\bar{x}_3\ \bar{x}_6\,x_7\}$
20	(1,0,1,*,*,*,*,*,*,…)	$\bar{x}_1x_2\,\bar{x}_3\ \bar{x}_6\,x_7$	$\{x_1,\ x_2\}$
21	(1,1,*,*,*,*,*,*,*,…)	$\bar{x}_2x_3x_5$	$\{x_1,\ x_2,\ \bar{x}_2x_3x_5\}$
22	(1,1,0,*,1,*,*,*,*,…)	$\bar{x}_1\ \bar{x}_2x_3\,\bar{x}_5\,x_6$	$\{x_1,\ x_2,\ \bar{x}_2x_3x_5,\ \bar{x}_1\ \bar{x}_2x_3\,\bar{x}_5\,x_6\}$
23	(1,1,0,*,1,1,*,*,*,…)	$\bar{x}_2x_3x_7$	$\{x_1,\ x_2,\ \bar{x}_2x_3x_5,\ \bar{x}_1\ \bar{x}_2x_3\,\bar{x}_5\,x_6,\ \bar{x}_2x_3x_7\}$
24	(1,1,0,*,1,1,1,*,*,…)	$\bar{x}_1\ \bar{x}_5\ \bar{x}_6\ \bar{x}_7$	$\{x_1,\ x_2,\ x_3,\ \bar{x}_1\ \bar{x}_5\ \bar{x}_6\ \bar{x}_7\ \}$
25	(1,1,1,*,1,1,0,*,*,…)	$\bar{x}_1\ \bar{x}_2\ \bar{x}_3\ \bar{x}_5\ \bar{x}_6$	$\{x_1,\ x_2,\ x_3,\ \bar{x}_5\ \bar{x}_6\ \}$
26	(1,1,1,*,1,0,*,*,*,…)	$\bar{x}_1\ \bar{x}_2\ \bar{x}_3\,x_6$	$\{x_1,\ x_2,\ x_3,\ \bar{x}_5\ ,\ \bar{x}_1\ \bar{x}_2\ \bar{x}_3\,x_6\}$
27	(1,1,1,*,0,1,*,*,*,…)	$\bar{x}_1x_5x_7$	$\{x_1,\ x_2,\ x_3,\ \bar{x}_5\ ,\ \bar{x}_1\ \bar{x}_2\ \bar{x}_3\,x_6,\ \bar{x}_1x_5x_7\}$
28	(1,1,1,*,0,1,1,*,*,…)	$\bar{x}_1\ \bar{x}_2\ \bar{x}_3\,x_8$	$\{x_1,\ x_2,\ x_3,\ \bar{x}_5\ ,\ \bar{x}_1\ \bar{x}_2\ \bar{x}_3\,x_6,\ \bar{x}_1x_5x_7,\ \bar{x}_1\ \bar{x}_2\ \bar{x}_3\,x_8\}$
29	(1,1,1,*,0,1,1,1,*,…)	$\bar{x}_1\ \bar{x}_2\ \bar{x}_3\ \bar{x}_7\ \bar{x}_8$	$\{\varnothing\}$

Table 1 : Twenty-nine calls of obstacle function in Chvatal's resolution search

4. **Dynamic Branch-and-Bound**

In the customary B&B approach branching decisions are generally based on very limited information about the likelihood that a particular branch will lead to an optimal solution. This information is especially pronounced at early stages of the branch and bound tree. Moreover, even 'reasonable' choices can be extremely poor if they are not sufficiently influential to reduce the alternatives for other variables. Unless a current branch has the power to inhibit the range of remaining alternatives, the branch and bound process can degenerate into the disastrous approximation to total enumeration sometimes observed. So, it seems worthwhile to consider a branching technique that has the ability to rid itself of certain types of uninfluential branches on the basis of more reliable information available at later stages. A chief component of this technique is to shrink the branch and bound tree by eliminating earlier branches that have been rendered *conditionally superfluous* by subsequent decisions.

The theme of Dynamic B&B (Glover and Tangedahl, 1976) introduces a strategy that modifies the sequence of decisions in B&B by:

(1) Discarding certain earlier branches as the process continues, thereby shrinking the B&B tree.

(2) Allowing for branches to be reversed based on the solution state created by the other currently imposed branches.

(3) Resequencing certain branches, according to rules based on identifying "influential choices" in retrospect, at the point where backtracking would occur (since the influence of branches can be identified more clearly when they are accompanied by other branch choices - as generally occurs at backtracking points - than when they may first be selected as branches). The goal is to maintain the most influential branches earlier in the tree by moving less influential branches to the end.

Component (3) is the one that includes the basic idea of Chvatal's Resolution Search, although introduced in a proposal that appears somewhat earlier. The notion of an influential branch, identified in retrospect, is critical. It is a conditional concept, that depends on other branches currently imposed, and embodies the idea of limiting the possibilities that exist for remaining choices. (It is related to the idea of a "strong cut", for example.)

***Remark* 10 :** Solving an MIP by this design consists of finding a tree with root (*, …, *) in its associated graph *G* such that all the nodes in the frontier of this tree are terminal nodes. Dynamic Branch-and-Bound and resolution search take advantage of the search space being a graph rather than a tree, in contrast to the approach of classical B&B.

Generic *Dynamic Branch-and-Bound Algorithm*

Let the current tree π = (*, …, *);

While all node in $Fr(\pi)$ are not terminal ***do***

- Select an unvisited terminal node u from Γ.
- **If** the node u is a descendant of π, add branches to π such that u will be in $Fr(\pi)$.
- **Else** change the tree π into a new tree one covering the node u.

Endwhile.

We may express the algorithm in greater detail as follows, where Γ^- corresponds to the set of visited nodes, a subset of Γ.

Generic *Dynamic Branch-and-Bound Algorithm* (Detailed Form)

- π = Any tree, Γ^- = *Interior*(π); $\Gamma^+ = \Gamma - \Gamma^-$;
- let $\Gamma^- = \varnothing$, $\Gamma^+ = \Gamma$;
- ***while*** ($\Gamma^- \neq \Gamma$) ***and*** ($\Gamma^+ \neq \varnothing$) ***and*** ($x \notin X^*$) ***do***
  - Select a node u from $Fr(\pi) \cap \Gamma^+$.
  - ***If*** u is not terminal ***then***
    - Add a branch to the tree $\pi := \pi + (u, v)$; with v in $\delta^{+1}(u) \cap \Gamma^+$;
    - $\Gamma^+ := \Gamma^+ + \{v\}$;
  - Reduce Γ^+ and/or π by the rules
    - $\Gamma^+ := \Gamma^+ - \{w\}$ if $\delta^{+1}(w) \subseteq \Gamma^+$;
    - $\Gamma^+ := \Gamma^+ - \{v \in Fr(\pi) : v$ is a *terminal node*$\}$
    - $\pi := \pi - \{v \in Fr(\pi) : v$ is a *terminal node*$\}$
  - ***Else***
    - $\Gamma^- := \Gamma^- + \delta^{+*}(u)$;
  - Update Γ^- and/or *Tree* by the rules
    - $\Gamma^- := \Gamma^- - \delta^{+1}(w) + \{w\}$ if $\delta^{+1}(w) \subseteq \Gamma^-$;

- Change a new tree from the current state such that Γ^- increases or Γ^+ decreases.

- Reduce Γ^+ and/or π by the rules
  - Drop a branch of the tree $\pi := \pi - (u, v)$; with v in $\delta^{-1}(u) \cap \Gamma^+$;
  - $\pi := \pi - \{v \in Fr(\pi) : v \text{ is a } terminal\ node\}$

Endwhile.

Influence can be measured also in terms of the effect on the objective function, as by the values of updated objective function coefficients for slack variables associated with branches.

i) *currently uninfluential* : The branching constraint $x_j \leq v_j$ (or $x_j \geq v_j$), where $j \in P$ and v_j is an integer value, will be called *currently uninfluential* if the constraint does not affect the optimality of the current LP solution. For example, if x_j receives the value v_j in the current LP solution, then clearly the branching constraint does not affect LP optimality.

ii) *highly influential* : A branch qualifies as be *highly influential* when it creates an immediate infeasibility or a bound violation when it is reversed. (By keeping a constraint that compels the objective function to improve at each step, the infeasibility criterion includes the bound violation criterion.)

iii) *more (or less) influential* : The branch $x_j \geq v_j$ is *more* (or *less*) *influential* than the branch $x_k \geq v_k$ if upon introducing their branching slacks $s_j = x_j - v_j \geq 0$ and $s_k = x_k - v_k \geq 0$ (using substitution to replace x_j by $s_j + v_j$ and to replace x_k by $s_k + v_k$), and optimizing the LP problem, the objective function coefficient of s_j in the optimal LP tableau is larger (smaller) than the one of s_k.

iv) A branch whose alternative has been eliminated by fathoming or by examination (i.e. by a tree search that exhausts all relevant solution possibilities on the branch) will be called a *compulsory* branch. (Note : a highly influential branch is a special case of a compulsory branch.)

Remark 11 : Emulating strategies applied in tabu search (Glover and Laguna, 1997), it can be judicious to keep historical records of influence and evaluations of branching alternatives to supplement the decision process.

When the concept of influence is applied in retrospect as a basis for resequencing the choices, as in (3), with the goal of maintaining the most influential branches earlier in the tree,

then the most extreme case is where one of the current branches will violate feasibility if it is moved to the end of the sequence and reversed. That means that the collection of preceding branches is so influential that the branch moved to the end cannot be changed - hence, by the rules of B&B, such a branch at the end of the tree can be dropped. Also, more generally, this retrospective analysis gives strategies for moving branches to the end even when the extreme case just described does not occur. That is, even if infeasibility will not occur by a reversal, it is still possible to identify which branches will come closest to violating feasibility when moved to the end, and therefore one of these will indeed be the one chosen for this relocation.

The shrinking operation in a sense is a special case of the resequencing - if the operation is postponed until a backtracking step occurs - but it can be done more efficiently because it isn't necessary to reverse the branch to discover that it can be dropped by an additional backtrack step.

Example 3 : These observations are clarified by the following example (Glover and Tangedahl, 1976). Shrinking occurs in this example when branches become "redundant" as a result of later branches which are independent of them in the tree.

$$\text{Minimize } z = 8x_1 + 8x_2 - 4x_3$$

$$3x_1 \quad - 2x_3 \geq 2$$

$$2x_1 - 2x_2 \quad \geq 1$$

$$-8x_1 + 20x_2 \geq -1$$

$$x_1, x_2, x_3 \geq 0 \text{ and integer.}$$

The illustrated sequence of steps is:

Step 1: Initial LP solution: $x = (0.75, 0.25, 0.13)$

Branch: $x_1 \geq 1$.

Step 2: LP reoptimization: $x = (1, 0.35, 0.50)$

Branch: $x_3 \geq 1$.

Step 3: LP reoptimization: $x = (1.33, 0.48, 1)$

Shrink: Drop $x_1 \geq 1$ Branch: $x_1 \geq 2$.

Step 4: LP reoptimization: $x = (2, 0.75, 2)$

Shrink: Drop $x_3 \geq 1$ Branch $x_2 \geq 1$.

Step 5: LP reoptimization: $x = (2, 1, 2)$

Feasible integer solution: $z = 16$

Resequence: current branch sequence $x_1 \geq 2$, $x_2 \geq 1$

new branch sequence $x_2 \geq 1, x_1 \geq 2$

Backtrack: impose $x_1 \leq 1$.

Step 6: LP reoptimization: no feasible solution

Backtrack: impose $x_2 \leq 0$.

Step 7: LP reoptimization: no feasible solution

Problem solution complete (no branches left).

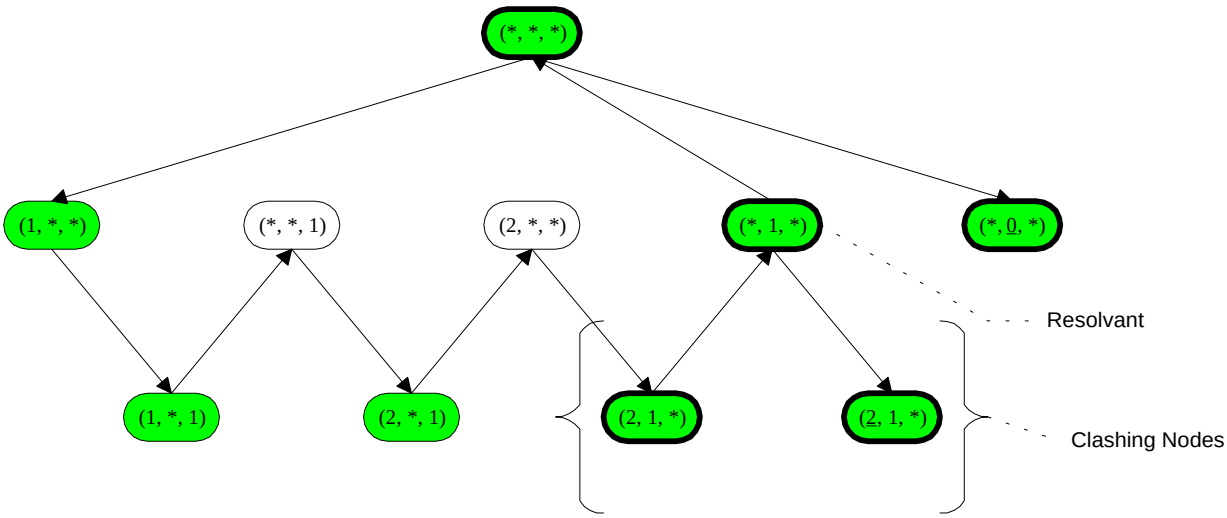

Figure 6 : An example of Dynamic B&B

In the preceding steps, the opportunity to shrink the tree occurred twice, the opportunity to resequence at backtracking occurred once, and these operations were implemented in each case. Using the same branching rules, the solution of the foregoing problem requires eleven steps (LP reoptimizations) if either the option to shrink or the option to resequence is bypassed, and requires fifteen steps if both options are bypassed. The resequencing step of this example uses exactly the special (extreme) case criterion that is the basis of Chvatal's approach. That is, by moving $x_1 \geq 2$ to the end of the sequence, the opposite branch $x_1 \leq 1$ becomes infeasible (violating the bound). Thus the backtrack continues farther. The result of continuing farther frees the variable branched on. However, in the example there is now no remaining choice to move something to the end, so the method simply reverses the single remaining branch and the method terminates.

Resolution search observes that the last branch that caused a bound to be violated may possibly be more important than other decisions, and therefore this last branch is never given

a lower priority than other branches. That is, the approach incorporates a partial approximation of the idea of influence (without using that term).

In the numerical example the backtracking occurs not because an objective function bound was violated by the last branch, but rather because the last branch led to a new best solution (with $z = 16$). When the objective function constraint associated with this solution is imposed ($z \leq 15$), the current LP becomes infeasible, and hence backtracking results, but there is no reason to assume the last branch is the one that is most important. For example, in the illustration given, the choice sequence that led to this solution could easily have been $x_2 \geq 1$ followed by $x_1 \geq 2$. Then, if the last branch was maintained as "primary", Chvatal's rule would free the x_2 branch and then impose $x_1 \leq 1$. But this choice would be inferior to the one identified in the example. That is, the branch $x_2 \geq 1$ is decidedly more influential than the branch $x_1 \geq 2$, keeping in mind that we will be dropping one of the branches and reversing the other. (The example does not identify the source of information that discloses which branch is more influential, but this knowledge, however obtained, is the reason for the resequencing step that moves the less influential branch $x_1 \geq 2$ to the end of the sequence. In other words, according to Chvatal's terminology, an "Oracle" is used to identify when the property is present.)

The preceding illustration shows that the concept of influence can be more effective for resequencing branches than the approach used in Resolution Search, for two reasons:

(1) Resolution Search gives no rule for the case when backtracking may occur as a result of finding a new best solution (where potential impact on the decision process can be significant).

(2) Resolution Search can also miss opportunities to make advantageous decisions in other cases, as demonstrated by the example just given where the bound $z \leq 15$ might have been pre-established. Then the sequence $x_2 \geq$ l, $x_1 \geq 2$ could have been the one that was discovered to violate this bound.

In short, the resequencing mechanism of Dynamic B&B gives the options proposed in Resolution Search, and also gives additional strategic possibilities. These observations motivate a more thorough examination of the practical potential of the Dynamic B&B approach, which has remained largely unexplored to date.

***Remark* 12** : To prove Van der Waerden's theorem, the e*x*tension of resolution search that results by using influential information, as in Dynamic B&B, needs only 2 calls of the obstacle function and 11 calls of the oracle function.

Acknowledgements:

We would like to thank the anonymous referees for their detailed comments and suggestions which improved this paper.

References:

[1] V. Chvatal. "Resolution Search", *Discrete Applied Mathematics,* 73, 81-99, 1997.

[2] F. Glover and L. Tangedahl. "Dynamic Strategies for Branch and Bound", *Omega,* 4, 571-576, 1976.

[3] F. Glover and M.Laguna, "Tabu Search*"*, *Kluwer Academic Publishers*, 1997.

[4] R. L. Graham, B. L. Rothschild and J. H. Spencer, "Ramsey Theory", *2nd ed., Wiley*, 1990.

[5] B.L. van der Waerden, "How the proof of Baudet's conjecture was found", *In: Studies in Pure Mathematics* (Presented to Richard Rado), pp. 251-260. *Academic Press*, *London*, 1971.