Integer and Combinatorial Optimization: Options in Branch and Bound

John E. Mitchell

Department of Mathematical Sciences
RPI, Troy, NY 12180 USA

March 2019
Our problem

We consider the problem

\[
\min_{x \in \mathbb{R}^n} c^T x \\
\text{subject to } A x = b \\
x \geq 0 \\
x_j \text{ binary, } j = 1, \ldots, p
\]

where \( 1 \leq p \leq n \).

We have an incumbent feasible solution \( x^u \) providing an upper bound \( z^u \) on the optimal value.

(It’s possible, with an abuse of notation, that \( x^u = \emptyset \) and \( z^u = +\infty \).)
Which variable to branch on?

We have a solution $x^a$ to the LP relaxation of the current node $a$ of the branch and bound tree, with $x^a$ violating the integrality requirements and with $z^a := c^T x^a < z^u$.

We let $F_0^a \subseteq \{1, \ldots, p\}$ denote the variables fixed to 0 at node $a$ and $F_1^a \subseteq \{1, \ldots, p\}$ denote the variables fixed to 1 at node $a$.

We have decided to branch on this node.

Which variable do we branch on?
Looking just at $x^a$

The branch and bound algorithm terminates once the upper and lower bounds agree.

We can focus on trying to improve one of these bounds. In particular, if we branch on a variable with $x_j^a$ fractional but close to either 0 or 1, then we can perhaps construct a good feasible solution in just a few nodes.

Alternatively, if we branch on a variable with $x_j^a$ close to 0.5, then perhaps the lower bound $z^a$ on this node will increase most quickly.

The reasoning is that this forces the greatest change in the solution on both branches, so it forces the solution away from $x^a$.

In practice, it seems this rule doesn’t do much better than randomly picking a branching variable.
Strong branching

We have a candidate set of branching variables $J \subseteq \{1, \ldots, p\} \setminus (F_0^a \cup F_1^a)$. 

We can test out branching on each of these variables before making a decision.
2 LPs for each possible branching variables

\[ z_{j0}^a := \min_{x \in \mathbb{R}^n} c^T x \]
subject to \( Ax = b \)
\[ x \geq 0 \]
\[ x_i = 0 \quad i \in F_0^a \]
\[ x_i = 1 \quad i \in F_1^a \]
\[ x_j = 0 \]

\[ z_{j1}^a := \min_{x \in \mathbb{R}^n} c^T x \]
subject to \( Ax = b \)
\[ x \geq 0 \]
\[ x_i = 0 \quad i \in F_0^a \]
\[ x_i = 1 \quad i \in F_1^a \]
\[ x_j = 1 \]
Choosing the branching variable

We then calculate the product of the improvements:

$$\Delta_j^a := (z_{j0}^a - z^a)(z_{j1}^a - z^a).$$

We branch on the variable $j$ with the largest value of $\Delta_j^a$. 
Pseudocosts

Strong branching is expensive, requiring the solution of many additional LP subproblems.

An alternative is to try to estimate the values $z^a_j$ and $z^a_j$. Every time we branch, we get information.

Say we branched on variable $x_k$ at node $b$ with value $z^b$, and solved the subproblems, obtaining values $z^b_{k_0}$ and $z^b_{k_1}$.

We can create an estimate for the change in the objective function value per unit change in the variable:

$$Q^k_0 = \frac{z^b_{k_0} - z^b}{x^b_k} \quad \quad Q^k_1 = \frac{z^b_{k_1} - z^b}{1 - x^b_k}$$

The larger these pseudocost values, the more useful it is to branch on variable $x_k$. 
Averaging

We can get a more sophisticated estimate by averaging these values over all previous times we’ve branched on $x_k$.

We can also include in the average all the times we considered branching on $x_k$ when doing strong branching.

A typical way to implement the branching decision in modern commercial codes is to use strong branching in the first few levels of the tree and then switch to branching based on pseudocosts.

See for example Achterberg et al. [1].
Node selection

There are several alternative available for choosing the next node.

**Depth first search**
In depth first search, we immediately solve a child node after solving its parent.

One aim is to try to quickly find a good feasible solution. One benefit is that it should be quick to reoptimize the child node, using the solution to the parent as a warm start.

**Best bound**
We choose the active node \( a \) with the smallest value of \( z^a \). This node cannot be pruned by bounds, so it will have to be solved at some point.
Best estimate

The best estimate rule is based on the pseudocosts.

For each active node \( a \), we calculate an estimate of the optimal value of the best integer feasible solution for that node:

\[
E^a = z^a + \sum_{j=1}^{p} \min \left\{ Q_0^j x_j^a, Q_1^j (1 - x_j^a) \right\}
\]

This is the value \( z^a \) of the node plus the estimated change in value from forcing each of the binary variables to take an integer value.

The rule then selects the active node \( a \) with the smallest value of \( E^a \).
We don’t have to branch on variable disjunctions, \((x_j = 0) \lor (x_j = 1)\).

We can branch on any inequality \((\pi^T x \leq \pi_0) \lor (\pi^T x \geq \pi_0 + 1)\), where \(\pi\) and \(\pi_0\) are integral, and where the current solution \(x^a\) satisfies

\[\pi_0 < \pi^T x^a < \pi_0 + 1.\]
GUB branching

Unbalanced tree

For example, if we have a constraint

$$\sum_{k \in K} x_k \leq 1$$

for some subset $K$ of the binary variables, a variable disjunction results in an unbalanced tree:

*the branch with $x_j = 1$ forces all other $x_k = 0$,* whereas *the branch with $x_j = 0$ is far less restrictive.*
Try to get a more balanced tree

Thus, we look for a subset $K_1 \subseteq K$ with the cardinality of $K_1$ approximately half that of $K$, and also with

$$\sum_{k \in K_1} x_k \approx \sum_{k \in K \setminus K_1} x_k.$$ 

We then branch on the disjunction

$$\sum_{k \in K_1} x_k = 0 \lor \sum_{k \in K \setminus K_1} x_k = 0.$$ 

This leads to far more balanced trees and is known as GUB (Generalized Upper Bound) branching.

SOS (Special Ordered Sets) branching is similar.
Restarting the search

Restarting the search can be used to try to improve the early branching decisions, exploiting preliminary computations in the construction of branching rules.

Recent work on restarting a B&B search for an integer programming problem includes Kılınç-Karzan et al. [5] and Fischetti and Monaci [3, 4].

These references all run truncated B&B searches and then use the information gathered in these runs to set up branching rules to make a final complete run.
Running for a while and then restarting

Kılınç-Karzan et al. [5] makes one initial incomplete branch and bound run until 200 nodes are fathomed.

Sparse fathomed ancestors of these nodes are found by solving mixed integer programs.

Branch-and-bound is then restarted, with branching guided by the set of sparse ancestors: given a particular node in the tree, the process is more likely to branch on a variable that appears in many of the sparse ancestors, with the corresponding constraints violated by the solution to the LP relaxation.

The choice of branching node is determined by CPLEX in [5].
Try out a few preliminary runs

Fischetti and Monaci [4] make five runs of B&B, with each run exploring at most five nodes.
These runs all solve the initial LP relaxation, and then branch starting from different optimal solutions to the relaxation.
One of these preliminary runs is then run to completion.
A backdoor

Fischetti and Monaci [3] try to identify a small set of important branching variables (a “backdoor”), where knowing the values of these variables would force many other variables to take particular values.

The set of important variables is found by looking at good fractional solutions and ensuring that at least one of the fractional components appears in the backdoor.

The branch-and-bound search is then restarted with branching priority given to the backdoor.
Use machine learning

There has also been recent work on using machine learning techniques to determine branching rules, for example to derive a branching rule that can replicate strong branching at lower computational cost by Alvarez et al. [2].

Machine learning and related approaches to branching are surveyed by Lodi and Zarpellon [6].
References

T. Achterberg, T. Koch, and A. Martin.
Branching rules revisited.

A machine learning-based approximation of strong branching.

M. Fischetti and M. Monaci.
Backdoor branching.

M. Fischetti and M. Monaci.
Exploiting erraticism in search.

Information-based branching schemes for binary linear mixed integer problems.

A. Lodi and G. Zarpellon.
On learning and branching: a survey.
References

T. Achterberg, T. Koch, and A. Martin.
Branching rules revisited.

A machine learning-based approximation of strong branching.

M. Fischetti and M. Monaci.
Backdoor branching.

M. Fischetti and M. Monaci.
Exploiting erraticism in search.

Information-based branching schemes for binary linear mixed integer problems.

A. Lodi and G. Zarpellon.
On learning and branching: a survey.
References

T. Achterberg, T. Koch, and A. Martin.
Branching rules revisited.

A machine learning-based approximation of strong branching.

M. Fischetti and M. Monaci.
Backdoor branching.

M. Fischetti and M. Monaci.
Exploiting erraticism in search.

Information-based branching schemes for binary linear mixed integer problems.

A. Lodi and G. Zarpellon.
On learning and branching: a survey.
References

T. Achterberg, T. Koch, and A. Martin.
Branching rules revisited.

A machine learning-based approximation of strong branching.

M. Fischetti and M. Monaci.
Backdoor branching.

M. Fischetti and M. Monaci.
Exploiting erraticism in search.

Information-based branching schemes for binary linear mixed integer problems.

A. Lodi and G. Zarpellon.
On learning and branching: a survey.
References

T. Achterberg, T. Koch, and A. Martin.
Branching rules revisited.

A machine learning-based approximation of strong branching.

M. Fischetti and M. Monaci.
Backdoor branching.

M. Fischetti and M. Monaci.
Exploiting erraticism in search.

Information-based branching schemes for binary linear mixed integer problems.

A. Lodi and G. Zarpellon.
On learning and branching: a survey.
Learning branching rules

References

T. Achterberg, T. Koch, and A. Martin.
Branching rules revisited.

A machine learning-based approximation of strong branching.

M. Fischetti and M. Monaci.
Backdoor branching.

M. Fischetti and M. Monaci.
Exploiting erraticism in search.

Information-based branching schemes for binary linear mixed integer problems.

A. Lodi and G. Zarpellon.
On learning and branching: a survey.