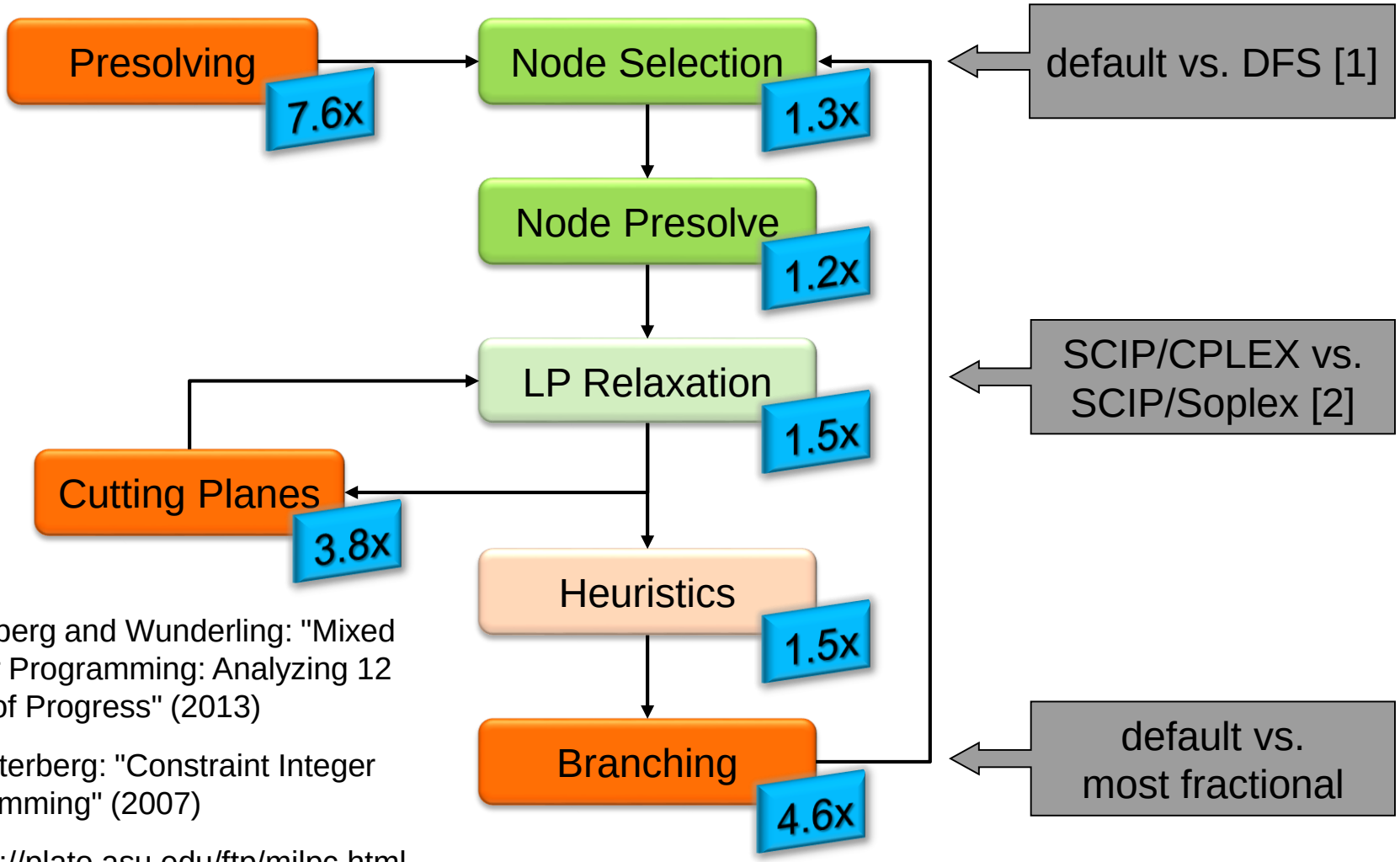
Solving MIPs via a branch-and-cut approach

The branch-and-cut approach is a combination of branch-and-bound and the cutting plane approach. This is what all commercial MIP solvers use.

1. **Initialize.**
   - $\mathcal{L} := \{N_0\}$;
   - $\underline{z} := -\infty$;
   - $(x^*, y^*) := \emptyset$;
2. **Terminate?**
   - if $\mathcal{L} = \emptyset$, the solution (x^*, y^*) is optimal.
3. **Select node.**
   - choose a node N_i in $\mathcal{L}$ and delete it from $\mathcal{L}$;
4. **Bound.**
   - solve LP_i;
   - if LP_i is infeasible, go to step 2;
   - if LP_i is feasible, let (x^i, y^i) be an optimal solution of LP_i, and z_i its objective value;
5. **Prune.**
   - if $z_i \leq \underline{z}$ go to step 2;
   - if (x^i, y^i) is feasible to the MIP, update:
     - $\underline{z} := z_i$;
     - $(x^*, y^*) := (x^i, y^i)$;
     - go to step 2;
6. **Add Cuts?**
   - Decide whether to strengthen formulation LP_i.
   - If so, add the desired cutting planes to LP_i and go to step 4.
7. **Branch.**
   - from LP_i construct $k \geq 2$ linear programs $\text{LP}_{i_1}, \ldots, \text{LP}_{i_k}$ with smaller feasible regions whose union does not contain (x^i, y^i), but contains the solutions of LP_i with $x \in \mathbb{Z}^n$.
   - add the new nodes $N_{i_1}, \ldots, N_{i_k}$ to $\mathcal{L}$ and go to step 2;

What might you consider when choosing to add cuts in step 6?

Performance Impact of MIP Solver Components (CPLEX 12.5 or SCIP)



Achterberg and Wunderling: "Mixed Integer Programming: Analyzing 12 Years of Progress" (2013)

[1] Achterberg: "Constraint Integer Programming" (2007)

[2] http://plato.asu.edu/ftp/milpc.html

Bixby, R.E., Fenelon, M., Gu, Z., Rothberg, E., and Wunderling, R. Mixed integer programming: progress report. In M. Grotschel, editor, *The Sharpest Cut*, chapter 18, pages 309—325. SIAM Philadelphia, PA, USA, 2004.

18.5.3 Factors contributing to speedups

We have listed some of the new features in recent versions of CPLEX. A natural question is, which of these features is most important to the improved performance? To measure the effects of the different features, at least in terms of proving optimality, we performed the following test. From the 978 models in the original testset, there were 106 models that were solvable by CPLEX 8.0 in less than 1000 seconds and were not solvable by CPLEX 5.0 in 100,000 seconds or less.[5] For each of several features of the code, we turned off that feature, ran CPLEX 8.0 on the 106 models, and compared the running times with the default CPLEX 8.0 runs. The results are summarized in Table 18.5.

Table 18.5. *CPLEX 8.0–effects of individual features.*

Feature	Degradation
No cuts	53.7
No presolve	10.8
CPLEX 5.0 presolve	3.1
CPLEX 5.0 variable selection	2.9
No heuristics	1.4
No node presolve	1.3
No probing on dives	1.1

The "No cuts" entry in this table means that CPLEX 8.0 defaults were compared with CPLEX 8.0 run with all cuts disabled. The other entries headed by the word "No" have a similar meaning. The "CPLEX 5.0 presolve" entry means that CPLEX 8.0 defaults were compared with CPLEX 8.0 with presolve disabled but applied to the model produced by CPLEX 5.0 presolve. Finally, the "CPLEX 5.0 variable selection" entry refers to comparing CPLEX 8.0 defaults with running CPLEX 8.0 using the default variable selection rule from CPLEX 5.0.

[5]For this test we changed the CPLEX 5.0 default settings slightly. CPLEX 5.0 included methods for generating cliques and knapsack covers. To more accurately measure the effect of adding cutting planes, we disabled these cuts in the CPLEX 5.0 runs.

The clear winner in these tests was cutting planes. Disabling this feature resulted in a deterioration of a *factor* of almost 54 in overall performance, a remarkable difference. At the other extreme, we see that heuristics, node presolve, and probing on dives had a much smaller overall effect. However, it should be noted that heuristics and probing on dives are primarily aimed at more quickly and effectively generating good feasible solutions. Proving optimality often focuses on the effectiveness with which the linear programming bound can be moved.

18.5.4 A cut comparison

Our final test used the 106 models isolated for testing in the previous section to compare the 9 different cutting planes that are implemented in CPLEX. The test was performed as follows. For each of the 8 kinds of default cutting planes, we disabled that one kind of cut and compared the result with the default running time. For disjunctive cuts, off by default, these cuts were enabled and compared to defaults. The results are given in Table 18.6.

Table 18.6. *CPLEX 8.0–effects of individual cuts.*

Cut type	Factor
Gomory mixed-integer	2.52
MIR	1.83
Knapsack cover	1.40
Flow cover	1.22
Implied bound	1.19
Path	1.04
Clique	1.02
GUB cover	1.02
Disjunctive	0.53

Gomory cuts are the clear winner by this measure. The degradation in performance caused by disabling these cuts was a factor of roughly 2.5. At the other extreme, enabling disjunctive cuts resulted in a degradation in performance of a factor of almost two. Note that this latter result is not to be interpreted as meaning that disjunctive cuts are ineffective, but rather as a measure of their relative contribution and the fact that they are by far the most expensive to compute of the various cutting planes implemented in CPLEX. When applied in the absence of other cutting planes, disjunctive cuts can be demonstrated to be quite effective.