Reading for This Lecture

• Wolsey Section 9.6
• Nemhauser and Wolsey Section II.6
• “Constraint Integer Programming,” Achterberg.
• “Noncommercial Software for Mixed-Integer Linear Programming,” Linderoth and Ralphs.
• “Implementations of Cutting Plane separators for Mixed Integer Programs,” Walter.
• “Branch-and-Price: Column Generation for Huge Integer Programs,” Barnhart, Johnson, Nemhauser, Savelsbergh, and Vance.
Branch and Cut and Price

• **Branch and cut** is an LP-based branch-and-bound scheme in which the linear programming relaxations are augmented by valid inequalities.
  – The valid inequalities are generated dynamically using separation procedures.
  – We iteratively try to improve the current bound by adding valid inequalities.
  – In practice, branch and cut is the method typically used for solving difficult MILPs.

• **Branch and price** is an LP-based branch-and-bound scheme in which either
  – The original formulation has a large (exponential) number of columns, or
  – The problem has been reformulated using Dantzig-Wolfe decomposition.

• **Branch and cut and price** is an LP-based branch-and-bound scheme in which we have both a large number of column and cut generation.

• All of these methods are a very complex amalgamation of techniques whose application must be balanced very carefully.
Computational Components of Branch and Cut

• Modular algorithmic components
  – Initial preprocessing and root node processing
  – Bounding
  – Cut generation
  – Primal heuristics
  – Node pre/post-processing (bound improvement, conflict analysis)
  – Node pre-bounding

• Overall algorithmic strategy
  – Search strategy
  – Bounding strategy
    * What cuts to generate and when
    * What primal heuristics to run and when
    * Management of the LP relaxation
  – Branching strategy
    * When to branch
    * How to branch (which disjunctions)
    * Relative amount of effort spent on choosing branch
Tradeoffs

• Control of branch and cut is about tradeoffs.

• We are combining many techniques and must adjust levels of effort of each to accomplish an end goal.

• Algorithmic control is an optimization problem in itself!

• Many algorithmic choices can be formally cast as optimization problems.

• What is the objective?
  – Time to optimality
  – Time to first “good” solution
  – Balance of both?
Bounding

- For now, we focus on the use of cutting plane methods for bounding.
- We will discuss decomposition-based bounding in a later lecture.
- The bounding loop is essentially a cutting plane method for solving the subproblem, but with some kind of early termination criteria.
- After termination, branching is performed to continue the algorithm.
- The bounding loop consists of several steps applied iteratively (not necessarily in this order).
  - Solve the current LP relaxation.
  - Decide whether node can be fathomed (by infeasibility or bound).
  - Generate inequalities violated by the solution to the LP relaxation.
  - Perform primal heuristics.
  - Apply node pre/post-processing.
  - Manage/improve LP relaxation (add/remove cuts, change bounds)
  - Decide whether to branch
Solving the LP Relaxation

• The LP relaxation is typically solved using a simplex-based algorithm.
  – This yields the advantage of efficient warm-starting of the solution process.
  – Many standard cut generation techniques require a basic solution.

• Interior point methods may be useful in some cases where they are much more effective (set packing/partitioning is one case in which this is typical).

• It may also be fruitful in some cases to explore the use of alternatives, such as the Volume Algorithm.
Cut Generation

• Standard methods for generating cuts
  – Gomory, GMI, MIR, and other tableau-based disjunctive cuts.
  – Cuts from the node packing relaxation (clique, odd hole)
  – Knapsack cuts (cover cuts).
  – Single node flow cuts (flow cover).
  – Simple cuts from pre-processing (probing, etc).

• We must choose from among these various method which ones to apply in each node.

• We must in general decide on a general level of effort we want to put into cut generation.
Managing the LP Relaxations

- In practice, the number of inequalities generated can be **HUGE**.
- We must be careful to keep the size of the LP relaxations small or we will sacrifice efficiency.
- This is done in two ways:
  - Limiting the number of cuts that are added each iteration.
  - Systematically deleting cuts that have become *ineffective*.
- How do we decide which cuts to add?
- And what do we do with the rest?
- What is an ineffective cut?
  - One whose dual value is (near) zero.
  - One whose slack variable is basic.
  - One whose slack variable is positive.
Managing the LP Relaxations

- Below is a graphical representation of how the LP relaxation is managed in practice.
- Newly generated cuts enter a buffer (the *local cut pool*).
- Only a limited number of what are predicted to be the most effective cuts from the local are added in each iteration.
- Cuts that prove effective locally may eventually sent to a global pool for future use in processing other subproblems.
Cut Generation and Management

• A significant question in branch and cut is what classes of valid inequalities to generate and when?

• It is generally not a good idea to try all cut generation procedures on every fractional solution arising.

• For generic mixed-integer programs, cut generation is most important in the root node.

• Using cut generation only in the root node yields a procedure called cut and branch.

• Depending on the structure of the instance, different classes of valid inequalities may be effective.

• Sometimes, this can be predicted ahead of time (knapsack inequalities).

• In other cases, we have to use past history as a predictor of effectiveness.

• Generally, each procedure is only applied at a dynamically determined frequency.
Deciding Which Cuts to Add

• Predicting what cuts will be effective is difficult in general.

• Degree of violation is an easy-to-apply criteria, but may not be the most natural or intuitive measure.

• Other measures
  – Bound improvement (difficult to predict/calculate)
  – Euclidean distance from point to be cut off.

• It is possible to generate cuts using a different measure than that which is used to add them from the local pool.

• This might be done because generation by a criteria other than degree of violation is difficult.
Deciding When to Branch

• Because the cutting plane algorithm is a finite algorithm in itself (at least in the pure integer case), there is no strict requirement to branch.

• The decision to branch is thus a practical matter.

• Typically, branching is undertaken when “tailing off” occurs, i.e., bound improvement slows.

• Detecting when this happens is not straightforward and there are many ways of doing it.

• Ultimately, branching and cutting (using the same disjunction) have the same impact on the bound.

• Tailing off may simply be a result of numerical issues

• We will consider the numerics of the solution process in a later lecture.
Balancing the Effort of Branching and Bounding

• To a large extent, the more effort one puts into branching, the smaller the search tree will be.

• Branching effort can be tuned by adjusting
  – How many branching candidates to consider.
  – How much effort should be put into estimating impact (pseudo-cost estimates versus strong branching, etc.).
  – The same can be said about efforts to improve both primal and dual bounds.
    * For dual bounds, we need to determine how much effort to spend generating various classes of inequalities.
    * For primal bounds, we need to determine how much effort to put into primal heuristics.
  – One of the keys to making the overall algorithm work well is to tune the amount of effort allocated to each of these techniques.
  – This is a very difficult thing to do and the proper balance is different for different classes of problems.
Primal Bounding Strategy

- The strategy space for primal heuristics is similar to that for cuts.
- We have a collection of different heuristics that can be applied.
- We need to determine which heuristics to apply and how often.
- Generally speaking, we do this dynamically based on the historical effectiveness of each method.
Computational Aspects of Search Strategy

- The search must find the proper balance between several factors.
  - Primal bound improvement versus dual bound improvement.
  - The savings accrued by diving versus the effectiveness of best first.
- When we are confident that the primal bound is near optimal, such as when the gap is small, a diving strategy is more appropriate.
- We can also adjust our strategy based on what the user’s desire is.
Adjusting Strategy Based on User Desire

• In general, there is always a tradeoff between improvement of the dual and the primal bound.

• The user may have particular desires about which of these is more important.

• Some solvers change their strategy according to the emphasis preferred by the user.
  – Proving optimality
  – Finding good solution quickly
Branch and Price

• Branch and price is similar to branch and cut with the step of cut generation begin replaced by that of column generation.

• Similar techniques must be used in all aspects of managing the algorithm.

• Obviously, there are still a number of differences that must be accounted for, however.
  – Special methods must be used for branching.
  – “True” bounds must be obtained when, e.g., pruning by bound.
  – Other auxiliary methods such as primal heuristics and preprocessing must be done differently.
Branching with Dantzig-Wolfe Decomposition

• Unfortunately, branching on the variables of the reformulation doesn’t work well in many cases.

• It’s generally difficult to keep a variable from being generated again after it’s been fixed to zero.

• Branching must be done in a way that does not destroy the structure of the column generation subproblem.

• We can do this by branching on the original variables, i.e., before the reformulation.

• In a 0-1 problem, branching on the $j^{th}$ original variable is equivalent to fixing the value of some element of the columns to be generated.

• This can usually be incorporated into the column generation subproblem.

• By limiting column generation in this way, we can implement a much wider array of branching rules.

• It is also possible to branch by imposing constraints in the master.
Generic Dantzig-Wolfe

• Traditionally, Dantzig-Wolfe has been applied to problems with known structure.

• The idea was to exploit an efficient solution method for the subproblem, e.g., dynamic programming for solving the knapsack problem.

• Almost all of what we have talked about can be "blindly" applied to generic MILPs, however.

• We need to fill in a few pieces:
  – How to identify the decomposition.
  – How to solve the subproblem.

• The subproblem can be solved effectively using a generic solver.

• Identifying a good decomposition in an automatic way is more difficult and is the subject of ongoing research.

• One approach is to try to identify block structure in the constraint matrix using methods developed in the linear algebra community.
**Primal Heuristics in Branch and Price**

- Primal heuristics play a similar role here as in branch and cut.
- Solving the subproblem may produce more useful information than what we typically get by solving a relaxation in branch and cut.
- We may try to “repair” the solution produced by solving the subproblem.
- Another very effective option is to solve the Dantzig-Wolfe LP as an integer program.
Lagrangian Relaxation in Branch and Bound

- One can also embed Lagrangian relaxation within a branch and bound framework.
- The advantage is possibly faster updates of the dual solution, but the tradeoff is not clear.
- The challenge is that much of the primal information produced by solution of the Dantzig-Wolfe LP is lost.
- This makes generating cuts, branching, etc. more difficult.
- An compromise is to try to keep track of the solutions generated in each iteration and combine them to approximate the primal information that is lost.
- This is essentially the approach taken by the Volume Algorithm for approximate solution of linear programs.
- It is also similar to weighted Dantzig-Wolfe and may help improve convergence through stabilization.
Branch and Cut and Price

- It is also possible to combine cut and column generation.
- In some cases, cuts can be generated in the extended space of the lambda variables, but this typically destroys the structure of the subproblem.
- An alternative is to generate cuts in the original space and treat them as if they were part of the original formulation.
- This makes the cut generation transparent and requires no real modification to the overall framework.
- Unfortunately, we do not then have a basis in the original space from which to generate tableau-based cuts.
- It is possible to obtain such a basis using a crossover, but this has not been done in practice.
- Typically, we stick to generation of classes that do not require a basis.