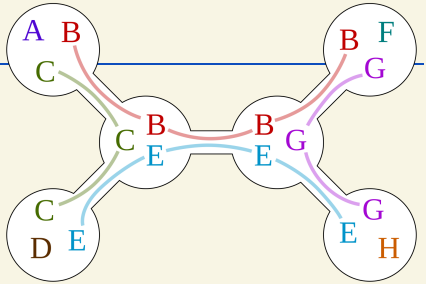
Branching II: Branch-and-Bound

DM898: Parameterized Algorithms
Lars Rohwedder



Today's lecture

Branching in practice:

- Combinatorial Branch-and-Bound
- LP-based Branch-and-Bound

Branch-and-Bound

In theory, we often reduce **optimization (and search) problems** to **decision problems**: given T output YES if there is a solution of value better than T or NO otherwise.

This is almost always equivalent . . . but the overhead of this reduction is not acceptable in practice.

Branch-and-Bound

In theory, we often reduce **optimization (and search) problems** to **decision problems**: given T output YES if there is a solution of value better than T or NO otherwise.

This is almost always equivalent ... but the overhead of this reduction is not acceptable in practice.

Branch-and-Bound is a refinement of branching for optimization problems

Branch-and-Bound and variations (Branch-and-Cut, Branch-and-Price, ...) are by far the most important branching algorithms in practice, specifically as the **main algorithm** in commercial solvers for **integer linear programs** (Gurobi, CPLEX, ...).

Branch-and-Bound

In theory, we often reduce **optimization (and search) problems** to **decision problems**: given T output YES if there is a solution of value better than T or NO otherwise.

This is almost always equivalent ... but the overhead of this reduction is not acceptable in practice.

Branch-and-Bound is a refinement of branching for optimization problems

Branch-and-Bound and variations (Branch-and-Cut, Branch-and-Price, ...) are by far the most important branching algorithms in practice, specifically as the **main algorithm** in commercial solvers for **integer linear programs** (Gurobi, CPLEX, ...).

Branching for optimization problems

Assume we have an instance I of a minimization problem (easy to adapt for maximization) and let $\mathrm{OPT}(I)$ denote the optimal value. Our goal is to generate several **easier** instances $I_1, \ldots, I_\ell$, $\ell \geq 2$ that we recurse on until we reach trivial instances. We require that $I_1, \ldots, I_\ell$ satisfy:

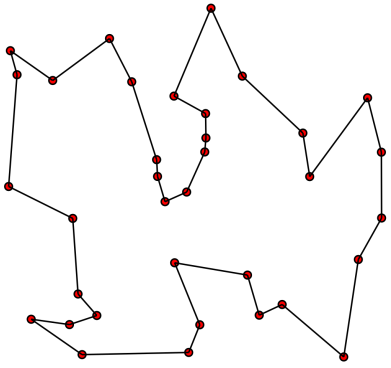
1. They can be generated efficiently
2. $\min\{\mathrm{OPT}(I_1), \ldots, \mathrm{OPT}(I_\ell)\} = \mathrm{OPT}(I)$
3. The complexities (or sizes) of $I_1, \ldots, I_\ell$ are smaller than of I and ℓ is bounded
4. Each solution of I_i, $i \leq \ell$, can be efficiently transformed into a solution of the same value for I

Problem with naive branching

Consider the travelling salesman problem (TSP) as an example: Given a map of n cities $C_1, \ldots, C_n$, find a tour of minimal length that travels to all cities and returns to the first.

A natural branching algorithm would branch on the first city to visit, then the next, etc.

Problem: no instance would ever be infeasible



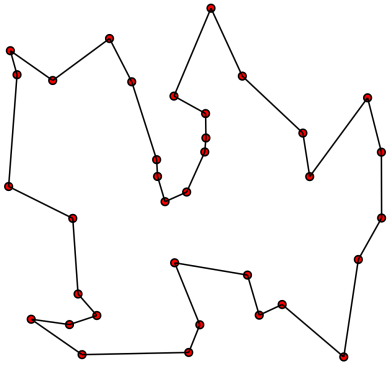
Problem with naive branching

Consider the travelling salesman problem (TSP) as an example: Given a map of n cities $C_1, \ldots, C_n$, find a tour of minimal length that travels to all cities and returns to the first.

A natural branching algorithm would branch on the first city to visit, then the next, etc.

Problem: no instance would ever be infeasible

$\rightsquigarrow$ if we prune (= stop explore) only infeasible nodes, **all** $n!$ **permutations** are always explored



Bounding

In each instance I (**minimization problem**) throughout the enumeration tree we calculate efficiently:

- an upper bound $\mathrm{UB}(I) \geq \mathrm{OPT}(I)$ from a solution of value $\mathrm{UB}(I)$ obtained by a heuristic
- a lower bound $\mathrm{LB}(I) \leq \mathrm{OPT}(I)$ from some structual insight of the problem.

Pruning

- By infeasibility: I has no solution
- By optimality: $\mathrm{LB}(I) = \mathrm{UB}(I)$
- By bounding: $\mathrm{LB}(I) \geq \mathrm{BEST}$, where $\mathrm{BEST} = \min_{I'} \mathrm{UB}(I')$ is the currently best solution among the entire enumeration tree

Elements of Branch-and-Bound

1. Branching
2. Bounding (lower, upper)
3. Instance reduction
4. Subproblem selection (where to continue exploring enumeration tree)
5. Pruning

Instance reduction $\approx$ preprocessing, but should be **very** fast, since it is applies many times in enumeration tree

Branch-and-Bound enumeration tree

The following is an example of how a Branch-and-Bound enumeration tree may evolve:

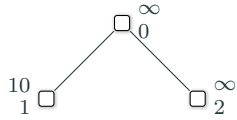
Numbers next to nodes indicate upper and lower bound.

Global best: ∞

Branch-and-Bound enumeration tree

The following is an example of how a Branch-and-Bound enumeration tree may evolve:

Numbers next to nodes indicate upper and lower bound.



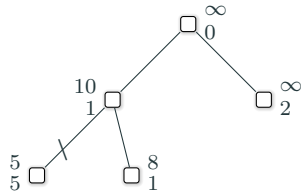
Global best: 10

Branch-and-Bound enumeration tree

The following is an example of how a Branch-and-Bound enumeration tree may evolve:

Numbers next to nodes indicate upper and lower bound.

Global best: 5

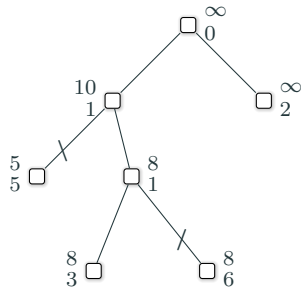


Branch-and-Bound enumeration tree

The following is an example of how a Branch-and-Bound enumeration tree may evolve:

Numbers next to nodes indicate upper and lower bound.

Global best: 5

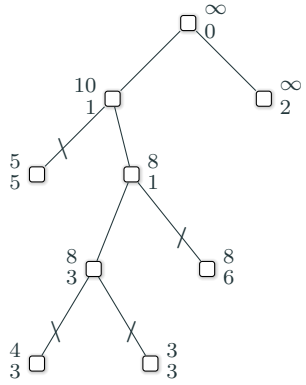


Branch-and-Bound enumeration tree

The following is an example of how a Branch-and-Bound enumeration tree may evolve:

Numbers next to nodes indicate upper and lower bound.

Global best: 3

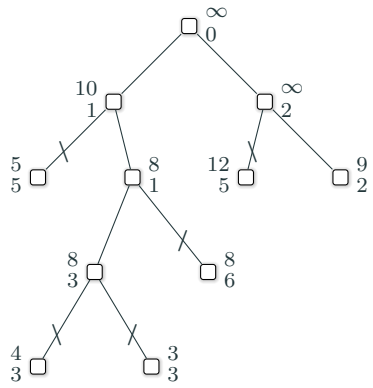


Branch-and-Bound enumeration tree

The following is an example of how a Branch-and-Bound enumeration tree may evolve:

Numbers next to nodes indicate upper and lower bound.

Global best: 3

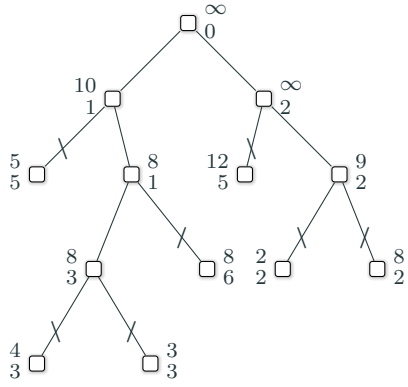


Branch-and-Bound enumeration tree

The following is an example of how a Branch-and-Bound enumeration tree may evolve:

Numbers next to nodes indicate upper and lower bound.

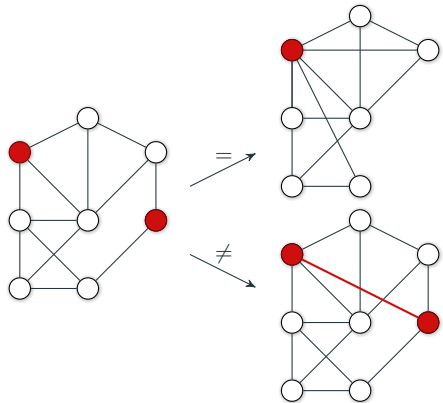
Global best: 2



Vertex Coloring

Branching over relation between vertices

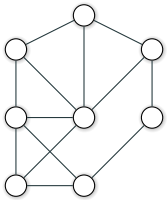
Select two vertices u, v are their colors equal or not?



equal: merge vertices; not equal: add edge between vertices

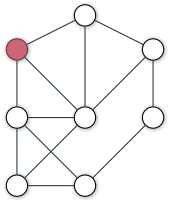
Bounding

Upper bound: Greedy coloring



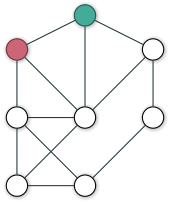
Bounding

Upper bound: Greedy coloring



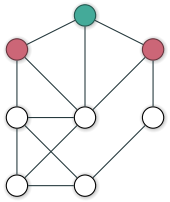
Bounding

Upper bound: Greedy coloring



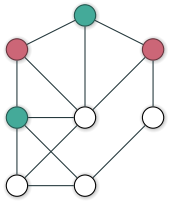
Bounding

Upper bound: Greedy coloring



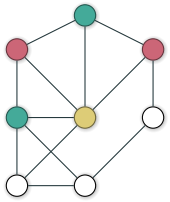
Bounding

Upper bound: Greedy coloring



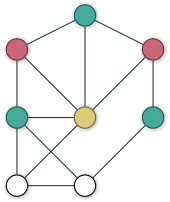
Bounding

Upper bound: Greedy coloring



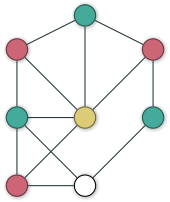
Bounding

Upper bound: Greedy coloring



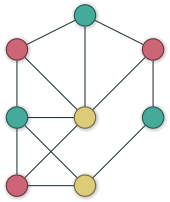
Bounding

Upper bound: Greedy coloring



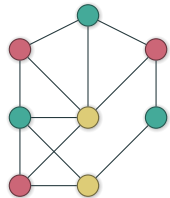
Bounding

Upper bound: Greedy coloring

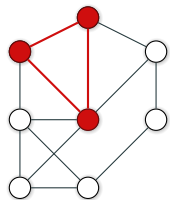


Bounding

Upper bound: Greedy coloring

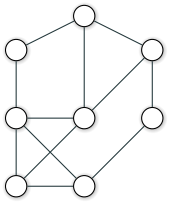


Lower bound: any clique (from e.g. a Greedy algorithm)



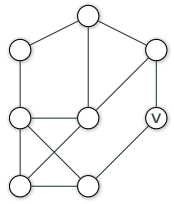
Instance reduction

Rule 1: if $\deg(v) <$ lower bound (clique), remove v



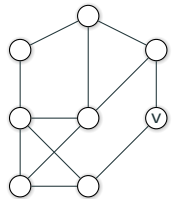
Instance reduction

Rule 1: if $\deg(v) <$ lower bound (clique), remove v

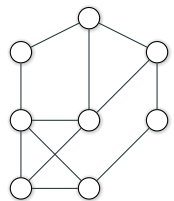


Instance reduction

Rule 1: if $\deg(v) <$ lower bound (clique), remove v

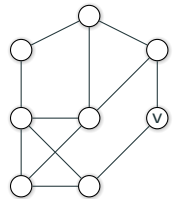


Rule 2: if $N(v) \subseteq N(u)$ (where $N(\cdot) =$ open neighborhood), remove v

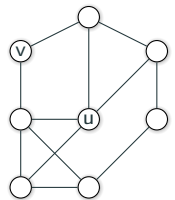


Instance reduction

Rule 1: if $\deg(v) <$ lower bound (clique), remove v



Rule 2: if $N(v) \subseteq N(u)$ (where $N(\cdot) =$ open neighborhood), remove v



Subproblem selection

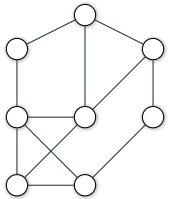
Which problem should we explore next? Not obvious. Examples:

1. fewest vertices (subproblem will be solved entirely soon),
2. smallest lower bound (likely to improve globally best solution)

Independent Set

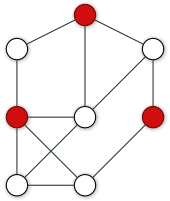
Independent Set

Given a graph $G = (V, E)$, find the largest set $U \subseteq V$ such that no pair of vertices $u, v \in U$ is connected by an edge.



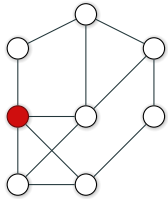
Independent Set

Given a graph $G = (V, E)$, find the largest set $U \subseteq V$ such that no pair of vertices $u, v \in U$ is connected by an edge.



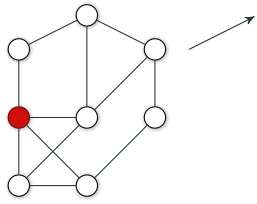
Branching

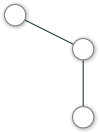
Choose a vertex $v \in V$ and branch on whether $v \in U$ or $v \notin U$:



Branching

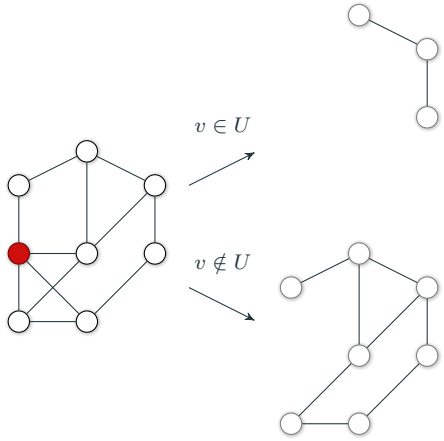
Choose a vertex $v \in V$ and branch on whether $v \in U$ or $v \notin U$:





Branching

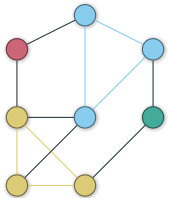
Choose a vertex $v \in V$ and branch on whether $v \in U$ or $v \notin U$:



Bounding

Lower bound: Greedy independent set

Upper bound: Greedy clique cover (cover of all vertices by cliques)



LP-based Branch-and-Bound

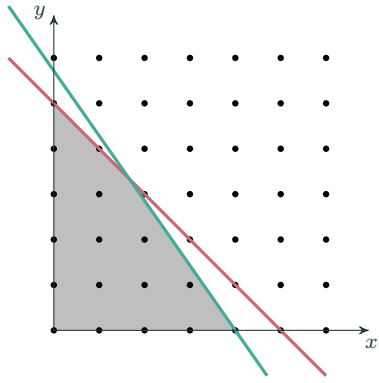
LP relaxations in Branch-and-Bound

Almost all commercial solvers for integer linear programming rely on Branch-and-Bound via the LP relaxation

LP relaxation

For a (mixed-)integer linear program, the LP relaxation is obtained by removing the integrality requirement of variables. This may allow infeasible (because fractional) solutions with better values, but can be solved efficiently.

$$\min -17x - 12y$$
$$10x + 7y \leq 40$$
$$x + y \leq 5$$
$$x, y \in \mathbb{Z}_{\geq 0}$$
[relaxation: $x, y \in \mathbb{R}_{\geq 0}$]



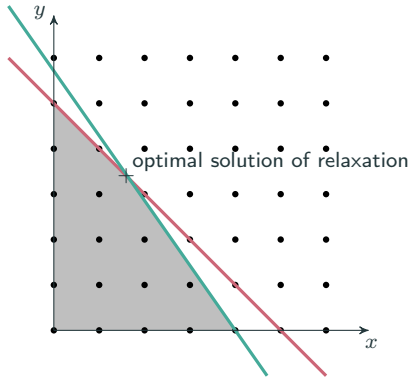
LP relaxations in Branch-and-Bound

Almost all commercial solvers for integer linear programming rely on Branch-and-Bound via the LP relaxation

LP relaxation

For a (mixed-)integer linear program, the LP relaxation is obtained by removing the integrality requirement of variables. This may allow infeasible (because fractional) solutions with better values, but can be solved efficiently.

$$\min -17x - 12y$$
$$10x + 7y \leq 40$$
$$x + y \leq 5$$
$$x, y \in \mathbb{Z}_{\geq 0}$$
[relaxation: $x, y \in \mathbb{R}_{\geq 0}$]



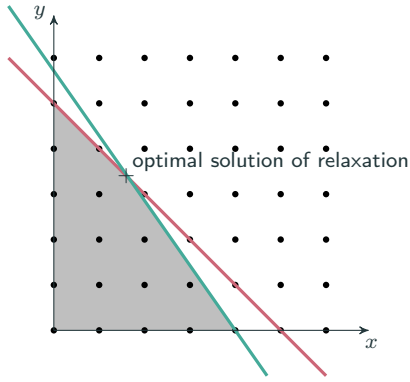
LP relaxations in Branch-and-Bound

Almost all commercial solvers for integer linear programming rely on Branch-and-Bound via the LP relaxation

LP relaxation

For a (mixed-)integer linear program, the LP relaxation is obtained by removing the integrality requirement of variables. This may allow infeasible (because fractional) solutions with better values, but can be solved efficiently.

$$\min -17x - 12y$$
$$10x + 7y \leq 40$$
$$x + y \leq 5$$
$$x, y \in \mathbb{Z}_{\geq 0}$$
$$[\text{relaxation: } x, y \in \mathbb{R}_{\geq 0}]$$



LP relaxations go together well with Branch-and-Bound because they provide:

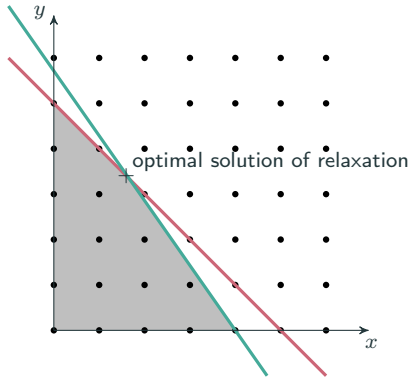
LP relaxations in Branch-and-Bound

Almost all commercial solvers for integer linear programming rely on Branch-and-Bound via the LP relaxation

LP relaxation

For a (mixed-)integer linear program, the LP relaxation is obtained by removing the integrality requirement of variables. This may allow infeasible (because fractional) solutions with better values, but can be solved efficiently.

$$\min -17x - 12y$$
$$10x + 7y \leq 40$$
$$x + y \leq 5$$
$$x, y \in \mathbb{Z}_{\geq 0}$$
$$[\text{relaxation: } x, y \in \mathbb{R}_{\geq 0}]$$



LP relaxations go together well with Branch-and-Bound because they provide:

- (often very strong) lower bounds

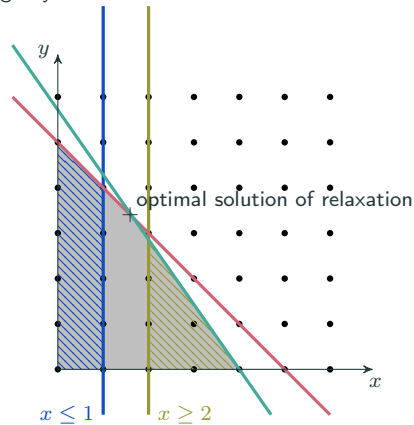
LP relaxations in Branch-and-Bound

Almost all commercial solvers for integer linear programming rely on Branch-and-Bound via the LP relaxation

LP relaxation

For a (mixed-)integer linear program, the LP relaxation is obtained by removing the integrality requirement of variables. This may allow infeasible (because fractional) solutions with better values, but can be solved efficiently.

$$\min -17x - 12y$$
$$10x + 7y \leq 40$$
$$x + y \leq 5$$
$$x, y \in \mathbb{Z}_{\geq 0}$$
[relaxation: $x, y \in \mathbb{R}_{\geq 0}$]



LP relaxations go together well with Branch-and-Bound because they provide:

- (often very strong) lower bounds
- a natural branching strategy: for fractional variable $i < x < i + 1$, $i \in \mathbb{Z}$, create subproblems with additional constraint $x \leq i$ or $x \geq i + 1$.

Details of LP-based Branch-and-Bound

Bounding

Lower bound: optimum of relaxation
Upper bound: **primal heuristics** attempt to find initial feasible solution, but for some ILPs this can be very difficult. Sometimes upper bound is only available after fully solving subproblem

Subproblem selection

Try to arrive quickly at an initial solution ($\approx$ depth-first)
Then most promising subproblem ($\approx$ breadth-first)

Where to branch

Option 1: "most fractional" variable (best: 0.5)
Option 2: greatest effect of branching (change of optimum)
...