Introduction to Optimization
Branch and Bound

October 12, 2015

École Centrale Paris, Châtenay-Malabry, France

Dimo Brockhoff

INRIA Lille – Nord Europe

Course Overview

Date		Topic
Mon, 21.9.2015		Introduction
Mon, 28.9.2015	D	Basic Flavors of Complexity Theory
Mon, 5.10.2015	D	Greedy algorithms
Mon, 12.10.2015	**D**	**Branch and bound (switched w/ dynamic programming)**
Mon, 2.11.2015	D	Dynamic programming *[salle Proto]*
Fri, 6.11.2015	D	Approximation algorithms and heuristics *[S205/S207]*
Mon, 9.11.2015	C	Introduction to Continuous Optimization I *[S118]*
Fri, 13.11.2015	C	Introduction to Continuous Optimization II *[from here onwards always: S205/S207]*
Fri, 20.11.2015	C	Gradient-based Algorithms
Fri, 27.11.2015	C	End of Gradient-based Algorithms + Linear Programming
Fri, 4.12.2015	C	Stochastic Optimization and Derivative Free Optimization
Fri, 18.12.2015		Exam

all classes + exam last 3 hours (incl. a 15min break)

Branch and Bound: General Ideas

- Systematic enumeration of candidate solutions in terms of a rooted tree
- Each tree node corresponds to a set of solutions; the whole search space on the root
- At each tree node, the corresponding subset of the search space is split into (non-overlapping) sub-subsets:
  - the optimum of the larger problem must be contained in at least one of the subproblems
- If tree nodes correspond to small enough subproblems, they are solved exhaustively.
- The smart part of the algorithm is the estimation of upper and lower bounds on the optimal function value achieved by solutions in the tree nodes
  - the exploration of a tree node is stopped if a node's upper bound is already lower than the lower bound of an already explored node (assuming maximization)

Applying Branch and Bound

Needed for successful application of branch and bound:

- optimization problem
- finite set of solutions
- clear idea of how to split problem into smaller subproblems
- efficient calculation of the following modules:
  - upper bound calculation
  - lower bound calculation

Computing Bounds (Maximization Problems)

Assume w.l.o.g. maximization of f(x) here

Lower Bounds

- any actual feasible solution will give a lower bound (which will be exact if the solution is the optimal one for the subproblem)
- hence, sampling a (feasible) solution can be one strategy
- using a heuristic to solve the subproblem another one

Upper Bounds

- upper bounds can be achieved by solving a relaxed version of the problem formulations (i.e. by either loosening or removing constraints)

Note: the better/tighter the bounds, the quicker the branch and bound tree can be pruned

Properties of Branch and Bound Algorithms

- Exact, global solver
- Can be slow; only exponential worst-case runtime
  - due to the exhaustive search behavior if no pruning of the search tree is possible
- but might work well in some cases

Advantages:

- can be stopped if lower and upper bound are "close enough" in practice (not necessarily exact anymore then)
- can be combined with other techniques, e.g. "branch and cut" (not covered here)

Example Branching Decisions

0-1 problems:

- choose unfixed variable x_i
- one subproblem defined by setting x_i to 0
- one subproblem defined by setting x_i to 1

General integer problem:

- choose unfixed variable x_i
- choose a value c that x_i can take
- one subproblem defined by restricting $x_i \leq c$
- one subproblem defined by restricting $x_i > c$

Combinatorial Problems:

- branching on certain discrete choices, e.g. an edge/vertex is chosen or not chosen

The branching decisions are then induced as additional constraints when defining the subproblems.

Which Tree Node to Branch on?

Several strategies (again assuming maximization):

- choose the subproblem with highest upper bound
  - gain the most in reducing overall upper bound
  - if upper bound not the optimal value, this problem needs to be branched upon anyway sooner or later
- choose the subproblem with lowest lower bound
- simple DFS or BFS
- problem-specific approach most likely to be a good choice

4 Steps Towards a Branch and Bound Algorithm

Concrete steps when designing a branch and bound algorithm:

- How to split a problem into subproblems (“branching”)?
- How to compute upper bounds (assuming maximization)?
- Optional: how to compute lower bounds?
- How to decide which next tree node to split?

now: example of integer linear programming
example of knapsack problem

Application to ILPs

$$
\begin{aligned}
\text{maximize} \quad & c^T x \\
\text{subject to} \quad & Ax \le b \\
& x \ge 0 \\
\text{and} \quad & x \in \mathbb{Z}^n
\end{aligned}
$$

The ILP formalization covers many problems such as

- Traveling Salesperson Person (TSP)
- Vertex Cover and other covering problems
- Set packing and other packing problems
- Boolean satisfiability (SAT)

Ways of Solving an ILP

- Do not restrict the solutions to integers and round the solution found of the relaxed problem (=remove the integer constraints) by a continuous solver (i.e. solving the so-called *LP relaxation*)

  → no guarantee to be exact
- Exploiting the instance property of A being total unimodular:
  - feasible solutions are guaranteed to be integer in this case
  - algorithms for continuous relaxation can be used (e.g. the simplex algorithm)
- Using heuristic methods (typically without any quality guarantee)
  - we'll see these type of algorithms in next week's lecture
- Using exact algorithms such as branch and bound

Branch and Bound for ILPs

Here, we just give an idea instead of a concrete algorithm...

- How to split a problem into subproblems (“branching”)?
- How to compute upper bounds (assuming maximization)?
- Optional: how to compute lower bounds?
- How to decide which next tree node to split?

Branch and Bound for ILPs

Here, we just give an idea instead of a concrete algorithm...

- How to compute upper bounds (assuming maximization)?
- How to split a problem into subproblems (“branching”)?
- Optional: how to compute lower bounds?
- How to decide which next tree node to split?

Branch and Bound for ILPs

How to compute upper bounds (assuming maximization)?

- drop the integer constraints and solve the so-called LP-relaxation
- can be done by standard LP algorithms such as **`scipy.optimize.linprog`** or Matlab's **`linprog`**

What's then?

- The LP has no feasible solution. Fine. Prune.
- We found an integer solution. Fine as well. Might give us a new lower bound to the overall problem.
- The LP problem has an optimal solution which is worse than the highest lower bound over all already explored subproblems. Fine. Prune.
- Otherwise: Branch on this subproblem: e.g. if optimal solution has $x_i=2.7865$, use $x_i \leq 2$ and $x_i \geq 3$ as new constraints

Branch and Bound for ILPs

How to split a problem into subproblems (“branching”)?

- mainly needed if the solution of the LP-relaxation is not integer
- branch on a variable which is rational

Not discussed here in depth due to time:

- Optional: how to compute lower bounds?
- How to decide which next tree node to split?
  - seems to be good choice: subproblem with largest upper bound of LP-relaxation

Branch and Bound for the 0-1 Knapsack Problem

How would you implement a branch-and-bound algorithm for the 0-1 knapsack problem?

what are the subproblems?

how to compute upper bounds?

how to compute lower bounds?

Conclusions

I hope it became clear...

...what the basic algorithm design ideas of branch and bound are

...and for which problem types it is supposed to be suitable back to the exercise:

A Greedy Algorithm for the Knapsack Problem

http://researchers.lille.inria.fr/~brockhof/optimizationSaclay/