Branch and Bound

Jesper Larsen & Jens Clausen

jla,jc@imm.dtu.dk

Informatics and Mathematical Modelling
Technical University of Denmark

DTU

Some definitions

- A branching is a division of the original feasible set S into subsets $S_1, \ldots, S_r$.
- A branching is **valid** if
  - $\cup_{i=1}^{r} S_i = S$
  - It is possible to determine and optimize over each set S_i
- A branching is called **partitive** if the sets S_i are disjoint.

Bounding

How do we get ourselves a bounding function? Relaxation.

- Leave out some constraints. Keep the same objective function f. So now we maximize f over a larger solution space P. If all original constraints are satisfied we have an **incumbent**.
- Change the objective function f to g. Here $g(x) \geq f(x)$ for all $x \in S$. Optimality of g does not automaticly guarantee optimality of f.
- Use the two above at the same time.

Strategy for selecting the next subproblem

- In choosing a search strategy we might consider two different goals:
  - Minimizing overall solution time.
  - Finding a good feasible solution to our original problem.

Mostly mentioned strategies

- **BeFS**: Best First Search strategy: select among the live subproblems one of those with the lowest bound. (sometime also called *global best node selection*).
- **BFS**: Breadth First Search strategy: select among the live subproblems one of those with the lowest level.
- **DFS**: Depth First Search strategy: select among the live subproblems one of those with the largest level.

The Best First Search strategy

- One way to minimize overall solution time is to try to minimize the size of the search tree.
- In theory we can achieve this by choosing the subproblem with the best bound.
- A subproblem (B&B node) is **critical** it its bound exceeds the value of the optimal solution.
- Best first is guaranteed to examine only critical nodes, thereby minimizing the search tree.

Lazy vs. Eager evaluation

- **Lazy** evaluation = only bound when it is really necessary.
- **Eager** evaluation = bound as soon as possible.

Branch and Bound for ATSP

- **Input:** A distance matrix D for a *digraf* $G = (V, E)$ with n vertices (so d_{ab} may be different from d_{ba}). If the edge (i, j) is in E then $d(i, j)$ equals the distance from i to j, otherwise d_{ij} equals ∞.
- **Output:** The length $d(C)$ of a shortest directed circuit C in G visiting each vertex exactly once, and an n-vector of edges $((v_1, v_2), \ldots, (v_{n-1}, v_n))$.
- **Method:** B&B: Bounding function, search and branching strategy is described below.

Mathematical formulation of the ATSP

$$\min \quad \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij}$$

$$\text{st.} \quad \sum_{j=1}^{n} x_{ij} = 1 \qquad i \in \{1,\ldots,n\}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j \in \{1,\ldots,n\}$$

$$\sum_{i,j\in S} x_{ij} < |S| \qquad \emptyset \subset S \subset V$$

$$x_{ij} \in \{0,1\} \qquad i,j \in \{1,\ldots,n\}$$

ATSP – Example

$$\begin{pmatrix} - & 3 & 93 & 13 & 33 & 9 \\ 4 & - & 77 & 42 & 21 & 16 \\ 45 & 17 & - & 36 & 16 & 28 \\ 39 & 90 & 80 & - & 56 & 7 \\ 28 & 46 & 88 & 33 & - & 25 \\ 3 & 88 & 18 & 46 & 92 & - \end{pmatrix}$$

ATSP – Lower bound calculation

$$\begin{pmatrix} - & 0 & 75 & 2 & 30 & 6 \\ 0 & - & 58 & 30 & 17 & 12 \\ 29 & 1 & - & 12 & 0 & 12 \\ 32 & 83 & 58 & - & 49 & 0 \\ 3 & 21 & 48 & 0 & - & 0 \\ 0 & 85 & 0 & 35 & 89 & - \end{pmatrix}$$

ATSP - B&B Components I

- **Bounding function:** Perform row and column reductions as for the assignment problem – subtract the smallest element in each row from all row elements and then the smallest element in each column of the resulting matrix from all column elements. The sum of the subtracted elements constitutes a lower bound for the length of the optimal ATSP tour.

ATSP - B&B Components II

- **Branching:** Find that 0-position (i, j) in the reduced matrix, for which the sum of the next-smallest elements in row i and column j is largest ("the most expensive edge to leave out in a new solution"). Construct two subproblems:
  - one in which (i, j) is forced into the solution (remove row i and column j from the matrix and set d_{ji} to ∞),
  - and one in which (i, j) is excluded from the solution (set d_{ij} equal to ∞).

ATSP - B&B Components III

- **Search strategy:** Depth first: Examine the subproblem with the *included* edge first. Because (i, j) is *most expensive* to leave out, there is a fair chance that the other subproblems is fathomed in a later iteration.

ATSP – Branching on $(6, 3)$

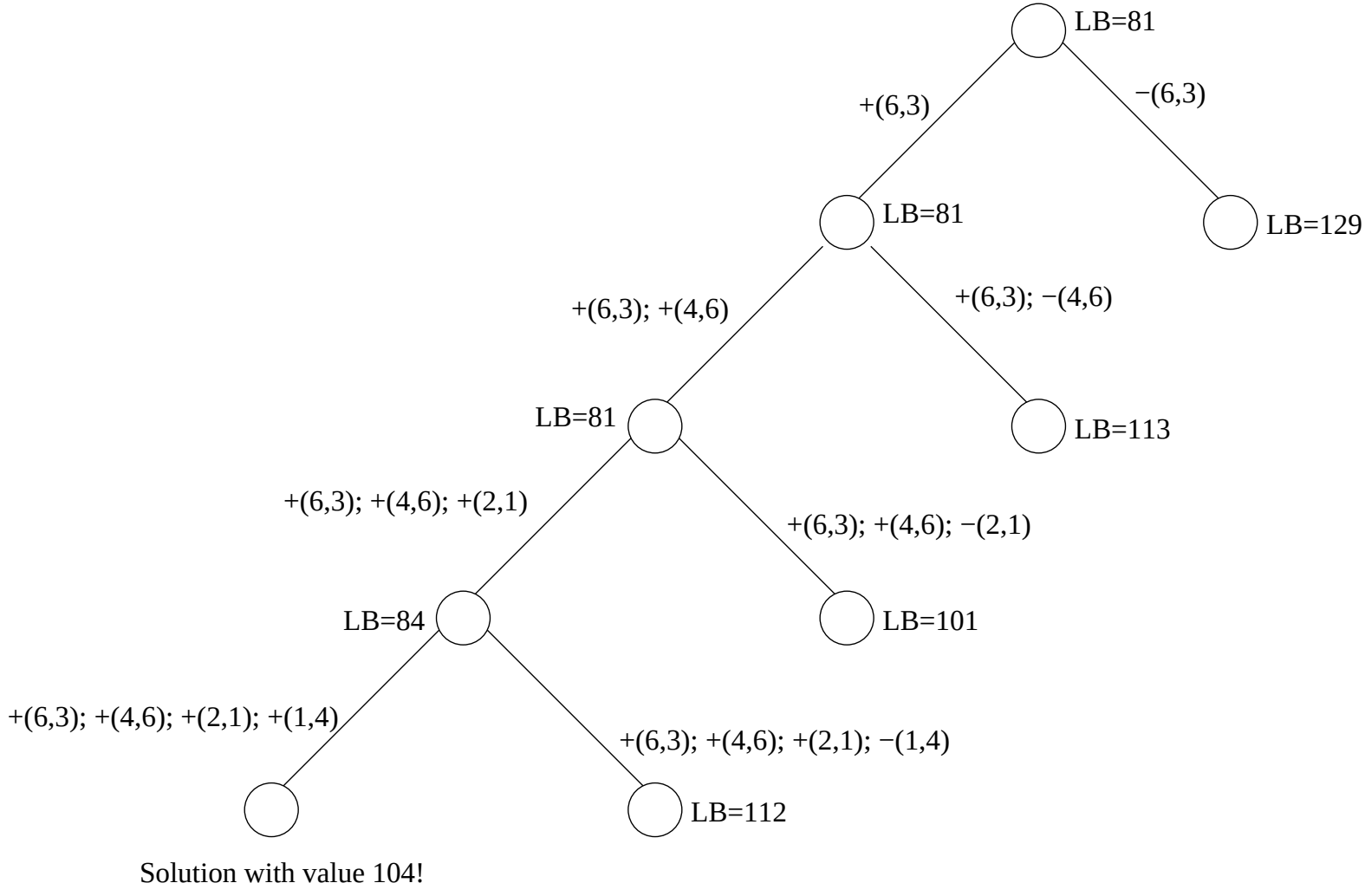
"1-branch"

$$\begin{pmatrix} - & 0 & 2 & 30 & 6 \\ 0 & - & 30 & 17 & 12 \\ 29 & 1 & 12 & 0 & - \\ 32 & 83 & - & 49 & 0 \\ 3 & 21 & 0 & - & 0 \end{pmatrix}$$

"0-branch"

$$\begin{pmatrix} - & 0 & 75 & 2 & 30 & 6 \\ 0 & - & 58 & 30 & 17 & 12 \\ 29 & 1 & - & 12 & 0 & 12 \\ 32 & 83 & 58 & - & 49 & 0 \\ 3 & 21 & 48 & 0 & - & 0 \\ 0 & 85 & - & 35 & 89 & - \end{pmatrix}$$

ATSP – Enumeration tree with solution



Using B&B as a heuristic

- From the root-node of the B&B-tree we have a (global) upper bound u.
- When we compute an initial solution at root node or bound by optimality we have a global lower bound l.
- This gives us the opportunity to estimate the quality of the solution. An estimate of the worst-case deviation of l from the optimum is:

$$100 \times \frac{u - l}{l}$$

Optimization-based heuristics

- Stop the B&B when the estimate on the deviation is below a certain threshold.
- Don't investigate all branches in the B&B-tree (especially applied within Binary integer programming)
- **Beam Search:** Usefull technique if memory is limited. Set an upper limit α on active nodes that are stored. If this limit is reached only keep the α best.

Keys to success

- Have a good bounding function.
- Get a good incumbent early (never underestimate the value of a good initial solution).
- Find ways to identify nodes that have no feasible decendents.
- Order constraints and variables so that the most restrictive are tested first.
- Experiment, experiment, experiment.