DM841

Discrete Optimization — Heuristics

Metaheuristics based on Construction Heuristics (I)

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Course Overview – Heuristics Part

✔ Local Search Algorithms: Components

✔ Local Search Based Metaheuristics

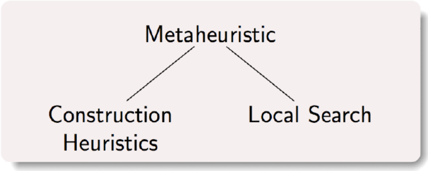
- Construction Heuristics Based Metaheuristics
- Working Environment and Solver Systems
- Population Based Metaheuristics
- Heuristics for the TSP
- Local Search: Neighborhoods and Search Landscape
- Efficient Local Search: Incremental Updates and Neighborhood Pruning (Focused LS)
- Very Large Scale Neighborhoods
- Methods for the Analysis of Experimental Results
- Methods for Algorithm Configuration and Tuning

Examples: GCP, CSP, TSP, SAT, MaxIndSet, SMTWP, Steiner Tree, p-median, set covering

Heuristics

Get inspired by approach to problem solving in human mind
[A. Newell and H.A. Simon. "Computer science as empirical inquiry: symbols and search." Communications of the ACM, ACM, 1976, 19(3)]

- effective rules without theoretical support
- trial and error



Applications:

- Optimization
- But also in Psychology, Economics, Management [Tversky, A.; Kahneman, D. (1974). "Judgment under uncertainty: Heuristics and biases". Science 185]

Basis on empirical evidence rather than mathematical logic. Getting things done in the given time.

Constructive search

What is a partial solution (as opposed to a complete solution)?

- Solutions as subsets of a larger ground set of solution components
- Partial solutions as a representation of all candidate solutions that contain them
- Not all subsets of components are valid/feasible partial solutions
- Construction rule
- Assessment of partial solutions inferred from the sets of solutions that they represent
- Lower bound (minimization) or upper bound (maximization)

Outline

1. Bounded backtrack
2. Limited Discrepancy Search
3. Random Restart
4. Rollout/Pilot Method
5. Beam Search
6. GRASP
7. Iterated Greedy

Complete Search Methods

Tree (or graph) search in

Uninformed settings (satisfaction probs)

- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search
- Bidirectional Search

Informed settings (optimization probs)

- best-first search, aka, greedy search
- A* search
- Iterative Deepening A*
- Memory bounded A*
- Recursive best first

In construction heuristics for this course, we can assume tree search of fixed known depth.

Best-first search

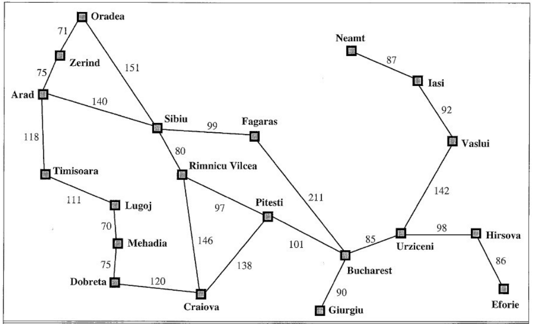


Figure 3.2 A simplified road map of part of Romania.

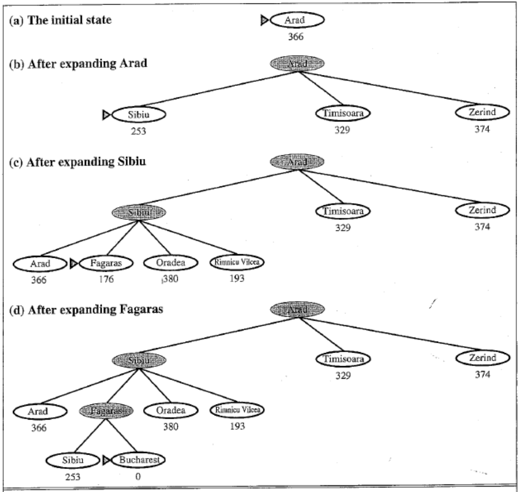


Figure 4.2 Stages in a greedy best-first search for Bucharest using the straight-line distance heuristic h_{SLD}. Nodes are labeled with their h-values.

A* search

A* search

- The priority assigned to a node x is determined by the function

$$f(x) = g(x) + h(x)$$

  $g(x)$: cost of the path so far
  $h(x)$: heuristic estimate of the minimal cost to reach the goal from x.
- It is optimal if $h(x)$ is an
  - admissible heuristic: *never overestimates* the cost to reach the goal
  - consistent: $h(n) \leq c(n, a, n') + h(n')$
    (consistent $\implies$ admissible, only necessary in graph search)

A* search

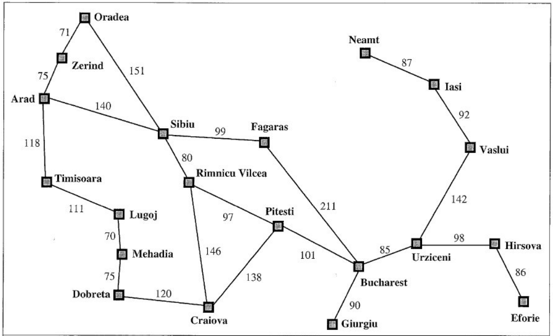


Figure 3.2 A simplified road map of part of Romania.

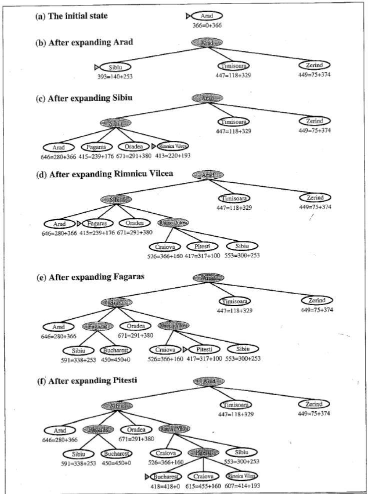


Figure 4.3 Stages in an A* search for Bucharest. Nodes are labeled with $f = g + h$. The h values are the straight-line distances to Bucharest taken from Figure 4.1.

A^* search

Possible choices for admissible heuristic functions

- optimal solution to an easily solvable **relaxed problem**
- optimal solution to an easily solvable **subproblem**
- learning from experience by gathering statistics on state features
- preferred: heuristic functions with higher values (provided they do not overestimate)
- if several heuristics available $h_1, h_2, \ldots, h_m$ and not clear which is the best then:

$$h(x) = \max\{h_1(x), \ldots, h_m(x)\}$$

A* search

Drawbacks

- Time complexity: In the worst case, the number of nodes expanded is exponential, (but it is polynomial when the heuristic function h meets the following condition:

$$|h(x) - h^*(x)| \leq O(\log h^*(x))$$

  h^* is the optimal heuristic, the exact cost of getting from x to the goal.)

- Memory usage: In the worst case, it must remember an exponential number of nodes. Several variants: including iterative deepening A* (IDA*), memory-bounded A* (MA*) and simplified memory bounded A* (SMA*) and recursive best-first search (RBFS)

Incomplete Search

On backtracking framework
(beyond depth-first search)

- Bounded backtrack
- Credit-based search
- Limited Discrepancy Search
- Barrier Search
- Randomization in Tree Search
- Random Restart

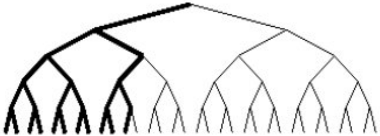
Outside the exact framework
(beyond greedy search)

- Random Restart
- Rollout/Pilot Method
- Beam Search
- Iterated Greedy
- GRASP
- (Adaptive Iterated Construction Search)
- (Multilevel Refinement)

Outline

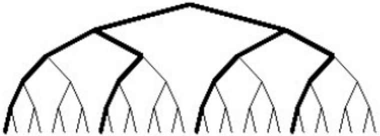
Bounded backtrack

Bounded-backtrack search:



bbs(10)

Depth-bounded, then bounded-backtrack search:

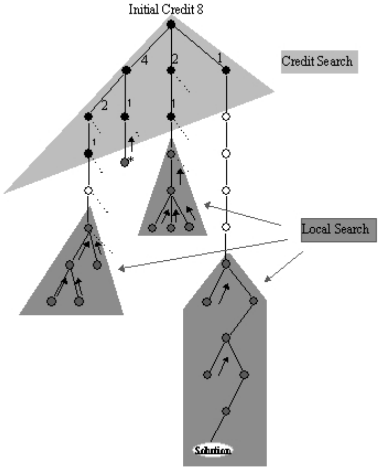


dbs(2, bbs(0))

http://4c.ucc.ie/~hsimonis/visualization/techniques/partial_search/main.htm

Credit-based search

- Key idea: important decisions are at the top of the tree
- Credit = backtracking steps
- Credit distribution: one half at the best child the other divided among the other children.
- When credits run out follow deterministic best-search
- In addition: allow limited backtracking steps (eg, 5) at the bottom
- Control parameters: initial credit, distribution of credit among the children, amount of local backtracking at bottom.

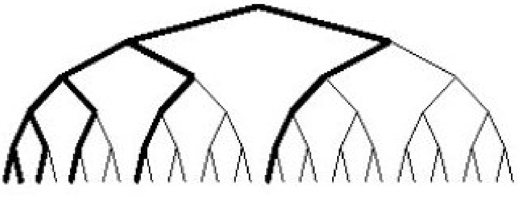


Outline

Limited Discrepancy Search

Limited Discrepancy Search (LDS)

- Key observation that often the heuristic used in the search is nearly always correct with just a few exceptions.
- Explore the tree in increasing number of discrepancies, modifications from the heuristic choice.
- Eg: count one discrepancy if second best is chosen
  count two discrepancies either if third best is chosen or twice the second best is chosen
- Control parameter: the number of discrepancies



Outline

1. Bounded backtrack
2. Limited Discrepancy Search
3. Random Restart
4. Rollout/Pilot Method
5. Beam Search
6. GRASP
7. Iterated Greedy

Randomization in Tree Search

The idea comes from complete search: the important decisions are made up in the search tree (backdoors - set of variables such that once they are instantiated the remaining problem simplifies to a tractable form)
$\leadsto$ random selections + restart strategy

Random selections

- randomization in variable ordering:
  - breaking ties at random
  - use heuristic to rank and randomly pick from small factor from the best
  - random pick among heuristics
  - random pick variable with probability depending on heuristic value
- randomization in value ordering:
  - just select random from the domain

Restart strategy in backtracking

- Example: $S_u = (1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 4, 8, 1, \ldots)$

Outline

1. Bounded backtrack
2. Limited Discrepancy Search
3. Random Restart
4. Rollout/Pilot Method
5. Beam Search
6. GRASP
7. Iterated Greedy

Rollout/Pilot Method

Derived from A*

- Each candidate solution is a collection of m components $S = (s_1, s_2, \ldots, s_m)$.
- Master process adds components sequentially to a partial solution $S_k = (s_1, s_2, \ldots s_k)$
- At the k-th iteration the master process evaluates feasible components to add based on an heuristic look-ahead strategy.
- The evaluation function $H(S_{k+1})$ is determined by sub-heuristics that complete the solution starting from S_k
- Sub-heuristics are combined in $H(S_{k+1})$ by
  - weighted sum
  - minimal value

Speed-ups:

- halt whenever cost of current partial solution exceeds current upper bound
- evaluate only a fraction of possible components

Outline

Beam Search

Based on the tree search framework:

- maintain a set B of bw (beam width) partial candidate solutions
- at each iteration extend each solution from B in fw (filter width) possible ways
- rank each $bw \times fw$ candidate solutions and take the best bw partial solutions
- complete candidate solutions obtained by B are maintained in B_f
- Stop when no partial solution in B is to be extended

Outline

GRASP

Greedy Randomized Adaptive Search Procedure

Key Idea: Combine randomized constructive search with subsequent local search.

Motivation:

- Candidate solutions obtained from construction heuristics can often be substantially improved by local search.
- Local search methods often require substantially fewer steps to reach high-quality solutions when initialized using greedy constructive search rather than random picking.
- By iterating cycles of constructive + local search, further performance improvements can be achieved.

Greedy Randomized "Adaptive" Search Procedure (GRASP):

while *termination criterion* is not satisfied **do**
  generate candidate solution s using
    subsidiary greedy randomized constructive search
  perform subsidiary local search on s

- Randomization in *constructive search* ensures that a large number of good starting points for *subsidiary local search* is obtained.
- Constructive search in GRASP is 'adaptive' (or dynamic): Heuristic value of solution component to be added to a given partial candidate solution may depend on solution components present in it.
- Variants of GRASP without local search phase (aka *semi-greedy heuristics*) typically do not reach the performance of GRASP with local search.

Restricted candidate lists (RCLs)

- Each step of *constructive search* adds a solution component selected uniformly at random from a restricted candidate list (RCL).

- RCLs are constructed in each step using a *heuristic function* h.

  - RCLs based on cardinality restriction comprises the k best-ranked solution components. (k is a parameter of the algorithm.)

  - RCLs based on value restriction comprise all solution components l for which
    $h(l) \leq h_{min} + \alpha \cdot (h_{max} - h_{min})$,
    where h_{min} = minimal value of h and h_{max} = maximal value of h for any l. (α is a parameter of the algorithm.)

  - Possible extension: reactive GRASP (*e.g.*, dynamic adaptation of α during search)

Example: Squeaky Wheel

Key idea: solutions can reveal problem structure which maybe worth to exploit.

Use a greedy heuristic repeatedly by prioritizing the elements that create troubles.

Squeaky Wheel

- Constructor: greedy algorithm on a sequence of problem elements.
- Analyzer: assign a penalty to problem elements that contribute to flaws in the current solution.
- Prioritizer: uses the penalties to modify the previous sequence of problem elements. Elements with high penalty are moved toward the front.

Possible to include a local search phase between one iteration and the other

Outline

Iterated Greedy

Key idea: use greedy construction

- alternation of construction and deconstruction phases
- an acceptance criterion decides whether the search continues from the new or from the old solution.

Iterated Greedy (IG):

determine initial candidate solution s
while termination criterion is not satisfied **do**
    $r := s$
    (randomly or heuristically) destruct part of s
    greedily reconstruct the missing part of s
    based on acceptance criterion,
        keep s or revert to $s := r$