Metaheuristics for Construction Heuristics. Local Search.

Marco Chiarandini

Outline

1. Metaheuristics for Construction Heuristics
   - Iterated Greedy
   - GRASP
   - Adaptive Iterated Construction Search
   - Multilevel Refinement

2. Local Search
   - Components
   - Iterative Improvement

Past Lecture

- Example Problems
  - The Constraint Satisfaction Problem
  - Problem Solving
  - The Single Machine Total Tardiness Problem
- Heuristic Methods
- Metaheuristics for Construction Heuristics
  - Complete Tree Search
  - Rollout/Pilot Method
  - Limited Discrepancy Search
  - Beam Search

Bin Packing

One dimensional

**Given:** A set \( L = (a_1, a_2, \ldots, a_n) \) of items, each with a size \( s(a_i) \in (0, 1] \) and an unlimited number of unit-capacity bins \( B_1, B_2, \ldots, B_m \).

**Task:** Pack all the items into a minimum number of unit-capacity bins \( B_1, B_2, \ldots, B_m \).

Related: stock cutting
Analytical Analysis

Definition: Approximation Algorithms
An algorithm $A$ is said to be a $\delta$-approximation algorithm if it runs in polynomial time and for every problem instance $\pi$ with optimal solution value $OPT(\pi)$

\[
\begin{align*}
\text{minimization:} & \quad \frac{A(\pi)}{OPT(\pi)} \leq \delta \quad \delta \geq 1 \\
\text{maximization:} & \quad \frac{A(\pi)}{OPT(\pi)} \geq \delta \quad \delta \leq 1
\end{align*}
\]

($\delta$ is called worst case bound, worst case performance, approximation factor, approximation ratio, performance bound, performance ratio, error ratio)

Definition: Polynomial approximation scheme
A family of approximation algorithms for a problem $\Pi$, $\{A_\epsilon\}_\epsilon$, is called a polynomial approximation scheme (PAS), if algorithm $A_\epsilon$ is a $(1 + \epsilon)$-approximation algorithm and its running time is polynomial in the size of the input for a fixed $\epsilon$.

Definition: Fully polynomial approximation scheme
A family of approximation algorithms for a problem $\Pi$, $\{A_\epsilon\}_\epsilon$, is called a fully polynomial approximation scheme (FPAS), if algorithm $A_\epsilon$ is a $(1 + \epsilon)$-approximation algorithm and its running time is polynomial in the size of the input and $1/\epsilon$

Outline

1. Metaheuristics for Construction Heuristics
   - Iterated Greedy
   - GRASP
   - Adaptive Iterated Construction Search
   - Multilevel Refinement

2. Local Search
   - Components
   - Iterative Improvement

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
\[
r := s \\
\text{greedily destruct part of } s \\
\text{greedily reconstruct the missing part of } s \\
\text{based on acceptance criterion,} \\
\text{keep } s \text{ or revert to } s := r
\]
Extension: 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.

Hybridize with subsidiary perturbative search
Example: on the SMTWTP

Greedy Randomized Adaptive Search Procedure (GRASP)

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

Motivation:
- Candidate solutions obtained from construction heuristics can often be substantially improved by perturbative search.
- Perturbative search methods typically 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 + perturbative search, further performance improvements can be achieved.

Greedy Randomized "Adaptive" Search Procedure (GRASP):
While termination criterion is not satisfied:
| generate candidate solution \( s \) using subsidiary greedy randomized constructive search
| perform subsidiary perturbative search on \( s \)

Note:
- Randomization in constructive search ensures that a large number of good starting points for subsidiary perturbative search is obtained.
- Constructive search in GRASP is 'adaptive' (or dynamic): Heuristic value of solution component to be added to given partial candidate solution \( r \) may depend on solution components present in \( r \).
- Variants of GRASP without perturbative search phase (aka semi-greedy heuristics) typically do not reach the performance of GRASP with perturbative 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 comprise the \( k \) best-ranked solution components. \((k \text{ is a parameter of the algorithm.})\)
- RCLs based on value restriction comprise all solution components \( l \) for which \( h(l) \leq h_{\text{min}} + \alpha \cdot (h_{\text{max}} - h_{\text{min}}) \), where \( h_{\text{min}} \) = minimal value of \( h \) and \( h_{\text{max}} \) = maximal value of \( h \) for any \( l \). \((\alpha \text{ is a parameter of the algorithm.})\)
Example: GRASP for SAT [Resende and Feo, 1996]

- Given: CNF formula $F$ over variables $x_1, \ldots, x_n$
- Subsidiary constructive search:
  - start from empty variable assignment
  - in each step, add one atomic assignment (i.e., assignment of a truth value to a currently unassigned variable)
  - heuristic function $h(i, v) :=$ number of clauses that become satisfied as a consequence of assigning $x_i := v$
  - RCLs based on cardinality restriction (contain fixed number $k$ of atomic assignments with largest heuristic values)
- Subsidiary perturbative search:
  - iterative best improvement using 1-flip neighborhood
  - terminates when model has been found or given number of steps has been exceeded

GRASP has been applied to many combinatorial problems, including:
- SAT, MAX-SAT
- various scheduling problems

Extensions and improvements of GRASP:
- reactive GRASP (e.g., dynamic adaptation of $\alpha$ during search)

Adaptive Iterated Construction Search

Key Idea: Alternate construction and perturbative local search phases as in GRASP, exploiting experience gained during the search process.

Realisation:
- Associate weights with possible decisions made during constructive search.
- Initialise all weights to some small value $\tau_0$ at beginning of search process.
- After every cycle (= constructive + perturbative local search phase), update weights based on solution quality and solution components of current candidate solution.

Adaptive Iterated Construction Search (AICS):

initialise weights

While termination criterion is not satisfied:
  - generate candidate solution $s$ using subsidiary randomized constructive search
  - perform subsidiary local search on $s$
  - adapt weights based on $s$
Subsidiary constructive search:

- The solution component to be added in each step of constructive search is based on weights and heuristic function $h$.
- $h$ can be standard heuristic function as, e.g., used by greedy construction heuristics, GRASP or tree search.
- It is often useful to design solution component selection in constructive search such that any solution component may be chosen (at least with some small probability) irrespective of its weight and heuristic value.

Subsidiary perturbative local search:

- As in GRASP, perturbative local search phase is typically important for achieving good performance.
- Can be based on Iterative Improvement or more advanced LS method (the latter often results in better performance).
- Tradeoff between computation time used in construction phase vs local search phase (typically optimized empirically, depends on problem domain).

Weight updating mechanism:

- Typical mechanism: increase weights of all solution components contained in candidate solution obtained from local search.
- Can also use aspects of search history; e.g., current incumbent candidate solution can be used as basis for weight update for additional intensification.

Example: A simple AICS algorithm for the TSP (1)

(Based on Ant System for the TSP [Dorigo et al., 1991].)

- Search space and solution set as usual (all Hamiltonian cycles in given graph $G$).
- Associate weight $\tau_{ij}$ with each edge $(i, j)$ in $G$.
- Use heuristic values $\eta_{ij} := 1/w((i, j))$.
- Initialize all weights to a small value $\tau_0$ (parameter).
- Constructive search starts with randomly chosen vertex and iteratively extends partial round trip $\phi$ by selecting vertex not contained in $\phi$ with probability

$$\frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in N(i)} [\tau_{il}]^\alpha \cdot [\eta_{ij}]^\beta}$$
Example: A simple AICS algorithm for the TSP (2)

- **Subsidiary local search**: iterative improvement based on standard 2-exchange neighbourhood (until local minimum is reached).

- **Weight update** according to

  \[ \tau_{ij} := (1 - \rho) \cdot \tau_{ij} + \Delta(i, j, s') \]

  where \( \Delta(i, j, s') := 1 / f(s') \), if edge \((i, j)\) is contained in the cycle represented by \(s'\), and 0 otherwise.

- Criterion for weight increase is based on intuition that edges contained in short round trips should be preferably used in subsequent constructions.

```
Example: Multilevel Refinement for TSP

Coarsen: fix some edges and contract vertices
Solve: matching
(always match vertices with the nearest unmatched neighbors)
Extend: uncontract vertices
Refine: LK heuristic
```

Multilevel Refinement

**Key idea**: make the problem recursively less refined creating a hierarchy of approximations of the original problem.

- an initial solution is found on the original problem or at a refined level
- solutions are iteratively refined at each level
- use of projection operators to transfer the solution from one level to another

```
while Termination criterion is not satisfied do
    coarse the problem \(\pi_0\) into \(\pi_i\), \(i = 0, \ldots, k\) successive non degenerate problems
    \(i = k\)
    determine an initial candidate solution for \(\pi_k\)
    repeat
        \(i = i - 1\)
        extend the solution found in \(\pi_i+1\) to \(\pi_i\)
        use subsidiary perturbative search to refine the solution on \(\pi_i\)
    until \(i \geq 0\)
```

**Note**

- crucial point: the solution to each refined problem must contain a solution of the original problem (even if it is a poor solution)

**Applications to**

- Graph Partitioning
- Traveling Salesman
- Graph Coloring
1. Metaheuristics for Construction Heuristics
   - Iterated Greedy
   - GRASP
   - Adaptive Iterated Construction Search
   - Multilevel Refinement

2. Local Search
   - Components
     - Iterative Improvement

Definition: Local Search Algorithm (1)

For given problem instance $\pi$:

- **search space** $S(\pi)$
  (e.g., for SAT: set of all complete truth assignments to propositional variables)

- **solution set** $S'(\pi) \subseteq S(\pi)$
  (e.g., for SAT: models of given formula)

- **neighborhood relation** $N(\pi) \subseteq S(\pi) \times S(\pi)$
  (e.g., for SAT: neighbouring variable assignments differ in the truth value of exactly one variable)

Definition: Local Search Algorithm (2)

- **set of memory states** $M(\pi)$
  (may consist of a single state, for LS algorithms that do not use memory)

- **initialization function** $\text{init} : \emptyset \mapsto \mathcal{P}(S(\pi) \times M(\pi))$
  (specifies probability distribution over initial search positions and memory states)

- **step function** $\text{step} : S(\pi) \times M(\pi) \mapsto \mathcal{P}(S(\pi) \times M(\pi))$
  (maps each search position and memory state onto probability distribution over subsequent, neighboring search positions and memory states)

- **termination predicate** $\text{terminate} : S(\pi) \times M(\pi) \mapsto \mathcal{P}\{\top, \bot\}$
  (determines the termination probability for each search position and memory state)
**Definition: Local Search Algorithm**

For given problem instance $\pi$:

- **search space** $S(\pi)$
- **solution set** $S'(\pi) \subseteq S(\pi)$
- **neighbourhood relation** $N(\pi) \subseteq S(\pi) \times S(\pi)$
- **set of memory states** $M(\pi)$
- **initialization function** $init : \emptyset \mapsto \mathcal{P}(S(\pi) \times M(\pi))$
- **step function** $step : S(\pi) \times M(\pi) \mapsto \mathcal{P}(S(\pi) \times M(\pi))$
- **termination predicate** $terminate : S(\pi) \times M(\pi) \mapsto \mathcal{P}\{\top, \bot\}$

**Example: Uninformed random walk for SAT**

- **search space** $S$: set of all truth assignments to variables in given formula $F$
- **solution set** $S'$: set of all models of $F$
- **neighbourhood relation** $N$: 1-flip neighbourhood, i.e., assignments are neighbours under $N$ if they differ in the truth value of exactly one variable
- **memory**: not used, i.e., $M := \{0\}$
Example: Uninformed random walk for SAT (continued)

- **initialisation**: uniform random choice from $S$, i.e.,
  \[ \text{init}(\{a', m\}) := 1/\#S \] for all assignments $a'$ and memory states $m$.

- **step function**: uniform random choice from current neighbourhood, i.e.,
  \[ \text{step}(\{a, m\}, \{a', m\}) := 1/|N(a)| \] for all assignments $a$ and memory states $m$,
  where $N(a) := \{a' \in S \mid N(a, a')\}$ is the set of all neighbours of $a$.

- **termination**: when model is found, i.e.,
  \[ \text{terminate}(\{a, m\}, \{\top\}) := 1 \] if $a$ is a model of $F$, and 0 otherwise.

**Definition: LS Algorithm Components (continued)**

**Note:**
- Procedural versions of *init*, *step* and *terminate* implement sampling from respective probability distributions.
- Memory state $m$ can consist of multiple independent attributes, i.e.,
  \[ M(\pi) := M_1 \times M_2 \times \ldots \times M_l(\pi). \]
- LS algorithms realize Markov processes: behavior in any search state $\{s, m\}$ depends only on current position $s$ and (limited) memory $m$.

**The Max Independent Set Problem**

Also called “stable set problem” or “vertex packing problem”.

**Given:** an undirected graph $G(V, E)$ and a non-negative weight function $\omega$ on $V$ ($\omega : V \rightarrow \mathbb{R}$).

**Task:** A largest weight independent set of vertices, i.e., a subset $V' \subseteq V$ such that no two vertices in $V'$ are joined by an edge in $E$.

Related Problems:

- **Vertex Cover**
  **Given:** an undirected graph $G(V, E)$ and a non-negative weight function $\omega$ on $V$ ($\omega : V \rightarrow \mathbb{R}$).
  **Task:** A smallest weight vertex cover, i.e., a subset $V' \subseteq V$ such that each edge of $G$ has at least one endpoint in $V'$.

- **Maximum Clique**
  **Given:** an undirected graph $G(V, E)$
  **Task:** A maximum cardinality clique, i.e., a subset $V' \subseteq V$ such that every two vertices in $V'$ are joined by an edge in $E$.

**Search Space**

Defined by the solution representation:

- permutations
  - linear
  - circular
- assignment arrays
- sets or lists
Definition: LS Algorithm Components (continued)

Neighborhood relation (structure): \( \mathcal{N} : S \times S \rightarrow \{T, F\} \) or \( \mathcal{N} \subseteq S \times S \)

- neighborhood (set) of candidate solution \( s \): \( N(s) := \{ s' \in S \mid \mathcal{N}(s, s') \} \)
- neighborhood graph of problem instance \( \pi \): \( G_\mathcal{N}(\pi) := (S(\pi), \mathcal{N}(\pi)) \)

Note: Diameter of \( G_\mathcal{N} \) is worst-case lower bound for number of search steps required for reaching (optimal) solutions. The maximal shortest path between any two vertices in the neighborhood graph.

Example:

SAT instance with \( n \) variables, 1-flip neighbourhood: \( G_\mathcal{N} = n \)-dimensional hypercube; diameter of \( G_\mathcal{N} = n \).

Definition: LS Algorithm Components (continued)

Step function

- Search step (or move): pair of search positions \( s, s' \) for which \( s' \) can be reached from \( s \) in one step, i.e., \( \mathcal{N}(s, s') \) and \( \text{step}(\{s, m\}, \{s', m'\}) > 0 \) for some memory states \( m, m' \in M \).

- Search trajectory: finite sequence of search positions \( (s_0, s_1, \ldots, s_k) \) such that \( (s_{i-1}, s_i) \) is a search step for any \( i \in \{1, \ldots, k\} \) and the probability of initializing the search at \( s_0 \) is greater zero, i.e., \( \text{init}(\{s_0, m\}) > 0 \) for some memory state \( m \in M \).

- Search strategy: specified by \( \text{init} \) and \( \text{step} \) function; to some extent independent of problem instance and other components of LS algorithm.
  - Random
  - Evaluation function
  - ...
Evaluation function:
- function \( f(\pi) : S(\pi) \rightarrow \mathbb{R} \) that maps candidate solutions of a given problem instance \( \pi \) onto real numbers, such that global optima correspond to solutions of \( \pi \);
- used for ranking or assessing neighbors of current search position to provide guidance to search process.

Evaluation vs objective functions:
- Evaluation function: part of LS algorithm.
- Objective function: integral part of optimization problem.
- Some LS methods use evaluation functions different from given objective function (e.g., dynamic local search).

Iterative Improvement
- does not use memory
- init: uniform random choice from \( S \)
- step: uniform random choice from improving neighbors, i.e., \( \text{step}(s, \{s'\}) := 1/|\mathcal{I}(s)| \) if \( s' \in \mathcal{I}(s) \), and 0 otherwise, where \( \mathcal{I}(s) := \{s' \in S \mid \mathcal{N}(s, s') \land f(s') < f(s)\} \)
- terminates when no improving neighbor available (to be revisited later)
- different variants through modifications of step function (to be revisited later)

Note: II is also known as iterative descent or hill-climbing.

Example: Iterative Improvement for SAT
- search space \( S \): set of all truth assignments to variables in given formula \( F \)
- solution set \( S' \): set of all models of \( F \)
- neighbourhood relation \( \mathcal{N} \): 1-flip neighbourhood (as in Uniform Random Walk for SAT)
- memory: not used, i.e., \( M := \{0\} \)
- initialisation: uniform random choice from \( S \), i.e., \( \text{init}(\emptyset, \{a'\}) := 1/|S| \) for all assignments \( a' \)

Example: Iterative Improvement for SAT (continued)
- evaluation function: \( f(a) := \) number of clauses in \( F \) that are unsatisfied under assignment \( a \)
  (Note: \( f(a) = 0 \) iff \( a \) is a model of \( F \).)
- step function: uniform random choice from improving neighbours, i.e., \( \text{step}(a, a') := 1/|\mathcal{I}(a)| \) if \( s' \in I(a) \), and 0 otherwise, where \( I(a) := \{a' \mid \mathcal{N}(a, a') \land f(a') < f(a)\} \)
- termination: when no improving neighbour is available i.e., \( \text{terminate}(a, \top) := 1 \) if \( I(a) = \emptyset \), and 0 otherwise.
Incremental updates (aka delta evaluations)

- **Key idea:** calculate effects of differences between current search position \(s\) and neighbors \(s'\) on evaluation function value.

- Evaluation function values often consist of independent contributions of solution components; hence, \(f(s)\) can be efficiently calculated from \(f(s')\) by differences between \(s\) and \(s'\) in terms of solution components.

- Typically crucial for the efficient implementation of II algorithms (and other LS techniques).

**Example: Incremental updates for TSP**

- Solution components = edges of given graph \(G\)
- Standard 2-exchange neighborhood, i.e., neighboring round trips \(p, p'\) differ in two edges

\[
\Delta_{ij} = d(e_i, e_j) + d(e_{i+1}, e_{j+1}) - d(e_i, e_{i+1}) - d(e_j, e_{j+1})
\]

- \(w(p') := w(p) - \text{edges in } p \text{ but not in } p' + \text{edges in } p' \text{ but not in } p\)

**Note:** Constant time (4 arithmetic operations), compared to linear time (\(n\) arithmetic operations for graph with \(n\) vertices) for computing \(w(p')\) from scratch.

**Definition:**

- **Local minimum:** search position without improving neighbors w.r.t. given evaluation function \(f\) and neighborhood \(N\), i.e., position \(s \in S\) such that \(f(s) \leq f(s')\) for all \(s' \in N(s)\).

- **Strict local minimum:** search position \(s \in S\) such that \(f(s) < f(s')\) for all \(s' \in N(s)\).

- **Local maxima** and **strict local maxima:** defined analogously.

**Iterative Improvement (2 OPT)**

```plaintext
procedure TSP-2opt-first(s)
    input: an initial candidate tour \(s \in S(\in)\)
    output: a local optimum \(s \in S(\pi)\)
    \(\Delta = 0;\)
    Improvement=FALSE;
    do
        for \(i = 1\) to \(n - 2\) do
            if \(i = 1\) then \(n' = n - 1\) else \(n' = n\)
            for \(j = i + 2\) to \(n'\) do
                \(\Delta_{ij} = d(e_i, e_j) + d(e_{i+1}, e_{j+1}) - d(e_i, e_{i+1}) - d(e_j, e_{j+1})\)
                if \(\Delta_{ij} < 0\) then
                    UpdateTour(s,i,j);
                    Improvement=TRUE;
                end
            end
        end
    until Improvement==TRUE;
end TSP-2opt-first
```