Introduction

Heuristic methods make use of two search paradigms:

- construction rules (extends partial solutions)
- local search (modifies complete solutions)

These components are problem specific and implement informed search.
They can be improved by use of metaheuristics which are general-purpose guidance criteria for underlying problem specific components.

Final heuristic algorithms are often hybridization of several components.
Construction Heuristics

(aka Dispatching Rules, in scheduling)
Closely related to search tree techniques
Correspond to a single path from root to leaf

- search space = partial candidate solutions
- search step = extension with one or more solution components

Construction Heuristic (CH):
\[ s := \emptyset \]
While \( s \) is not a complete solution:
\[ \begin{align*}
&\text{choose a solution component } c \\
&\text{add the solution component to } s
\end{align*} \]

Greedy best-first search

An important class of Construction Heuristics are greedy algorithms
Always make the choice which is the best at the moment.
- Sometime it can be proved that they are optimal
  \[1 \| \sum w_j C_j, 1 \| L_{\text{max}}\]
- Other times it can be proved an approximation ratio

- Another class can be derived by the (variable, value) selection rules in CP and removing backtracking (ex, MRV, least-constraining-values).
Examples of Dispatching Rules in Scheduling

<table>
<thead>
<tr>
<th>Rule</th>
<th>Data</th>
<th>Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDD</td>
<td>(d_j)</td>
<td>Variance in Throughput Times</td>
</tr>
<tr>
<td>MS</td>
<td>(d_j)</td>
<td>Maximum Lateness</td>
</tr>
<tr>
<td>LPT</td>
<td>(p_j)</td>
<td>Load Balancing over Parallel Machines</td>
</tr>
<tr>
<td>SPT</td>
<td>(p_j)</td>
<td>Sum of Completion Times, WIP</td>
</tr>
<tr>
<td>WSPT</td>
<td>(p_j, w_j)</td>
<td>Weighted Sum of Completion Times, WIP</td>
</tr>
<tr>
<td>OP</td>
<td>(p_j, pre)</td>
<td>Makespan</td>
</tr>
<tr>
<td>LNS</td>
<td>(p_j, pre)</td>
<td>Mailbox</td>
</tr>
<tr>
<td>SIRO</td>
<td>(t_{ij})</td>
<td>Ease of Implementation</td>
</tr>
<tr>
<td>SST</td>
<td>(t_{ij})</td>
<td>Makespan and Throughput</td>
</tr>
<tr>
<td>LFJ</td>
<td>(t_{ij})</td>
<td>Makespan and Throughput</td>
</tr>
<tr>
<td>SQNO</td>
<td>-</td>
<td>Machine Idleness</td>
</tr>
</tbody>
</table>

Local Search

Example: Local Search for CSP

```plaintext
function MIN-CONFlicts(csp, max_steps) returns a solution or failure
inputs: csp, a constraint satisfaction problem
        max_steps, the number of steps allowed before giving up

        current ← an initial complete assignment for csp
        for \(i = 1\) to \(max_steps\) do
            if current is a solution for csp then return current
            var ← a randomly chosen, conflicted variable from VARIABLES[csp]
            value ← the value \(v\) for \(var\) that minimizes CONFLICTS(var, v, current, csp)
            set var ← value in current
        return failure
```

Solution Representation

The solution representation determines the search space \(S\)

- permutations
  - linear (scheduling)
  - circular (routing)
- assignment arrays (timetabling)
- sets or lists (timetabling)
Initial Solution

- Random
- Construction heuristic

Neighborhood Structure

- Neighborhood structure (relation): equivalent definitions:
  - $\mathcal{N} : S \times S \rightarrow \{T,F\}$
  - $\mathcal{N} \subseteq S \times S$
  - $\mathcal{N} : S \rightarrow 2^S$

- Neighborhood (set) of a candidate solution $s$: $\mathcal{N}(s) := \{s' \in S \mid \mathcal{N}(s,s')\}$

- A neighborhood structure is also defined by an operator. An operator $\Delta$ is a collection of operator functions $\delta : S \rightarrow S$ such that
  $$s' \in \mathcal{N}(s) \iff \exists \delta \in \Delta \mid \delta(s) = s'$$

Example

$k$-exchange neighborhood: candidate solutions $s,s'$ are neighbors iff $s$ differs from $s'$ in at most $k$ solution components

Acceptance Criterion

The acceptance criterion defines how the neighborhood is searched and which neighbor is selected.

Examples:
- uninformed random walk
- iterative improvement (hill climbing)
  - best improvement
  - first improvement

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).
Implementation Issues

At each iteration, the examination of the neighborhood must be fast!!

- 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.

- Special algorithms for solving efficiently the neighborhood search problem

Local Optima

Definition:

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

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

- Local maxima and strict local maxima: defined analogously.

Example: Iterative Improvement

First improvement for TSP

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

Permutations

$\Pi(n)$ indicates the set all permutations of the numbers $\{1, 2, \ldots, n\}$

$(1, 2, \ldots, n)$ is the identity permutation $\iota$.

If $\pi \in \Pi(n)$ and $1 \leq i \leq n$ then:

- $\pi_i$ is the element at position $i$
- $\pos_{\pi}(i)$ is the position of element $i$

Alternatively, a permutation is a bijective function $\pi(i) = \pi_i$

The permutation product $\pi \cdot \pi'$ is the composition $(\pi \cdot \pi')(i) = \pi'(\pi(i))$

For each $\pi$ there exists a permutation such that $\pi^{-1} \cdot \pi = \iota$
Neighborhood Operators for Linear Permutations

Swap operator
\[ \Delta_S = \{ \delta^i_S \mid 1 \leq i \leq n \} \]
\[ \delta^i_S(\pi_1 \ldots \pi_i \pi_{i+1} \ldots \pi_n) = (\pi_1 \ldots \pi_{i-1} \pi_j \pi_{j+1} \pi_{j+2} \pi_{i+1} \ldots \pi_n) \]

Interchange operator
\[ \Delta_X = \{ \delta^i_X \mid 1 \leq i < j \leq n \} \]
\[ \delta^i_X(\pi) = (\pi_1 \ldots \pi_{i-1} \pi_j \pi_i \pi_{j+1} \ldots \pi_n) \]

Insert operator
\[ \Delta_I = \{ \delta^i_I \mid 1 \leq i \leq n, 1 \leq l \leq n, j \neq i \} \]
\[ \delta^i_I(\pi) = \begin{cases} (\pi_1 \ldots \pi_{i-1} \pi_{i+1} \ldots \pi_n) & i < j \\ (\pi_1 \ldots \pi_{i-1} \pi_j \pi_{j+1} \pi_{i-1} \pi_i \pi_{i+1} \ldots \pi_n) & i > j \end{cases} \]

Neighborhood Operators for Assignments

An assignment can be represented as a mapping
\[ \sigma : \{X_1 \ldots X_n\} \to \{v : v \in \mathbb{D}, |\mathbb{D}| = k\} \]
\[ \sigma = \{X_i = v_i, X_j = v_j, \ldots\} \]

One exchange operator
\[ \Delta_{1E} = \{ \delta^i_{1E} \mid 1 \leq i \leq n, 1 \leq l \leq k \} \]
\[ \delta^i_{1E}(\sigma) = \{ \sigma' : \sigma'(X_i) = v_l \text{ and } \sigma'(X_j) = \sigma(X_j) \forall j \neq i \} \]

Two exchange operator
\[ \Delta_{2E} = \{ \delta^i_{2E} \mid 1 \leq i < j \leq n \} \]
\[ \delta^i_{2E}(\sigma) = \{ \sigma' : \sigma'(X_i) = \sigma(X_j), \sigma'(X_j) = \sigma(X_i) \text{ and } \sigma'(X_l) = \sigma(X_l) \forall l \neq i, j \} \]

Neighborhood Operators for Circular Permutations

Reversal (2-edge-exchange)
\[ \Delta_R = \{ \delta^i_R \mid 1 \leq i < j \leq n \} \]
\[ \delta^i_R(\pi) = (\pi_1 \ldots \pi_{i-1} \pi_j \pi_i \pi_{j+1} \ldots \pi_n) \]

Block moves (3-edge-exchange)
\[ \Delta_B = \{ \delta^i_B \mid 1 \leq i < j < k \leq n \} \]
\[ \delta^i_B(\pi) = (\pi_1 \ldots \pi_{i-1} \pi_j \pi_k \pi_i \pi_{j+1} \ldots \pi_{k-1} \pi_{k+1} \ldots \pi_n) \]

Short block move (Or-edge-exchange)
\[ \Delta_{SB} = \{ \delta^i_{SB} \mid 1 \leq i < j \leq n \} \]
\[ \delta^i_{SB}(\pi) = (\pi_1 \ldots \pi_{i-1} \pi_j \pi_{j+1} \pi_{j+2} \pi_i \ldots \pi_{i-1} \pi_{j+3} \ldots \pi_n) \]

Neighborhood Operators for Partitions or Sets

An assignment can be represented as a partition of objects selected and not selected
\[ s : \{X\} \to \{C, \overline{C}\} \]
(it can also be represented by a bit string)

One addition operator
\[ \Delta_{1E} = \{ \delta^i_{1E} \mid v \in \overline{C} \} \]
\[ \delta^i_{1E}(s) = \{ s' : C' = C \cup v \text{ and } \overline{C'} = \overline{C} \setminus v \} \]

One deletion operator
\[ \Delta_{1E} = \{ \delta^i_{1E} \mid v \in C \} \]
\[ \delta^i_{1E}(s) = \{ s' : C' = C \setminus v \text{ and } \overline{C'} = \overline{C} \cup v \} \]

Swap operator
\[ \Delta_{1E} = \{ \delta^i_{1E} \mid v \in C, u \in \overline{C} \} \]
\[ \delta^i_{1E}(s) = \{ s' : C' = C \cup u \setminus v \text{ and } \overline{C'} = \overline{C} \cup v \setminus u \} \]
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 add components sequentially to a partial solution $S_k = \{s_1, s_2, \ldots, s_k\}$.
- At the $k$-th iteration the master process evaluates seemingly feasible components to add based on a look-ahead strategy based on heuristic algorithms.
- 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
  - maximal value
Speed-ups:
- halt whenever cost of current partial solution exceeds current upper bound
- evaluate only a fraction of possible components

It is optimal if $H(S_k)$ is an
- admissible heuristic: never overestimates the cost to reach the goal
- consistent: $h(n) \leq c(n, a, n') + h(n')$; $c(n, a, n')$ cost to go from node $n$ to $n'$ with action $a$

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 heuristics 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:

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

Beam Search

Possible extension of tree based construction heuristics:
- maintains 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
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
\[ T := s \]
- greedily destruct part of \( s \)
- greedily reconstruct the missing part of \( s \) based on acceptance criterion,
  keep \( s \) or revert to \( s := T \)

Greedy Randomized Adaptive Search Procedure (GRASP)

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

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

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. (\( k \) 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 \( h_{\text{min}} \) is minimal value of \( h \) and \( h_{\text{max}} \) is maximal value of \( h \) for any \( l \). (\( \alpha \) is a parameter of the algorithm.)

Simulated Annealing

**Key idea:** Vary temperature parameter, i.e., probability of accepting worsening moves, in Probabilistic Iterative Improvement according to annealing schedule (aka cooling schedule).

Simulated Annealing (SA):
determine initial candidate solution \( s \)
set initial temperature \( T \) according to annealing schedule
While termination condition is not satisfied:
\[ \text{While maintain same temperature } T \text{ according to annealing schedule:} \]
- probabilistically choose a neighbor \( s' \) of \( s \) using proposal mechanism
  - If \( s' \) satisfies probabilistic acceptance criterion (depending on \( T \)):
    - \( s := s' \)
- update \( T \) according to annealing schedule
Tabu Search

Key idea: Use aspects of search history (memory) to escape from local minima.

- Associate tabu attributes with candidate solutions or solution components.
- Forbid steps to search positions recently visited by underlying iterative best improvement procedure based on tabu attributes.

Tabu Search (TS):

determine initial candidate solution \( s \)

While termination criterion is not satisfied:

1. determine set \( N' \) of non-tabu neighbors of \( s \)
2. choose a best improving candidate solution \( s' \) in \( N' \)
3. update tabu attributes based on \( s' \)

\( \text{s} := s' \)

Example: Simulated Annealing for the TSP

Extension of previous PII algorithm for the TSP, with

- proposal mechanism: uniform random choice from 2-exchange neighborhood;
- acceptance criterion: Metropolis condition (always accept improving steps, accept worsening steps with probability \( \exp [(f(s) - f(s'))/T] \));
- annealing schedule: geometric cooling \( T := 0.95 \cdot T \) with \( n \cdot (n-1) \) steps at each temperature (\( n \) = number of vertices in given graph), \( T_0 \) chosen such that 97% of proposed steps are accepted;
- termination: when for five successive temperature values no improvement in solution quality and acceptance ratio < 2%.

Improvements:

- neighborhood pruning (e.g., candidate lists for TSP)
- greedy initialization (e.g., by using NNH for the TSP)
- low temperature starts (to prevent good initial candidate solutions from being too easily destroyed by worsening steps)

Non-tabu search positions in \( N(s) \) are called admissible neighbors of \( s \).
Note: Performance of Tabu Search depends crucially on setting of tabu tenure \( tt \):
- \( tt \) too low \( \Rightarrow \) search stagnates due to inability to escape from local minima;
- \( tt \) too high \( \Rightarrow \) search becomes ineffective due to overly restricted search path (admissible neighborhoods too small).

Iterated Local Search

Key Idea: Use two types of LS steps:
- subsidiary local search steps for reaching local optima as efficiently as possible (intensification)
- perturbation steps for effectively escaping from local optima (diversification).

Also: Use acceptance criterion to control diversification vs intensification behavior.

Iterated Local Search (ILS):

determine initial candidate solution \( s \)
perform subsidiary local search on \( s \)
While termination criterion is not satisfied:
- \( \tau := s \)
- perform perturbation on \( s \)
- perform subsidiary local search on \( s \)
- based on acceptance criterion,
- keep \( s \) or revert to \( s := \tau \)

Memetic Algorithm

Population based method inspired by evolution

determine initial population \( sp \)
perform subsidiary local search on \( sp \)
While termination criterion is not satisfied:
- generate set \( spr \) of new candidate solutions by recombination
- perform subsidiary local search on \( spr \)
- generate set \( spm \) of new candidate solutions from \( spr \) and \( sp \) by mutation
- perform subsidiary local search on \( spm \)
- select new population \( sp \) from candidate solutions in \( sp, spr, \) and \( spm \)

Selection

Main idea: selection should be related to fitness
- Fitness proportionate selection (Roulette-wheel method)
  \[
  p_i = \frac{f_i}{\sum_j f_j}
  \]
- Tournament selection: a set of chromosomes is chosen and compared and the best chromosome chosen.
- Rank based and selection pressure
Recombination (Crossover)

- Binary or assignment representations
  - one-point, two-point, m-point (preference to positional bias w.r.t. distributional bias)
  - uniform crossover (through a mask controlled by a Bernoulli parameter $p$)
- Non-linear representations
  - (Permutations) Partially mapped crossover
  - (Permutations) mask based
- More commonly ad hoc crossovers are used as this appears to be a crucial feature of success
- Two off-springs are generally generated
- Crossover rate controls the application of the crossover. May be adaptive: high at the start and low when convergence

Mutation

- **Goal:** Introduce relatively small perturbations in candidate solutions in current population + offspring obtained from recombination.
- Typically, perturbations are applied stochastically and independently to each candidate solution; amount of perturbation is controlled by mutation rate.
- Mutation rate controls the application of bit-wise mutations. May be adaptive: low at the start and high when convergence.
- Possible implementation through Poisson variable which determines the $m$ genes which are likely to change allele.
- Can also use *subsidiary selection function* to determine subset of candidate solutions to which mutation is applied.
- The role of mutation (as compared to recombination) in high-performance evolutionary algorithms has been often underestimated.

New Population

- **Goal:** Obtain population of high-quality solutions while maintaining population diversity.
- Selection is based on evaluation function (fitness) of candidate solutions such that better candidate solutions have a higher chance of ‘surviving’ the selection process.
- It is often beneficial to use *elitist selection strategies*, which ensure that the best candidate solutions are always selected.
- Most commonly used: *steady state* in which only one new chromosome is generated at each iteration.
- Diversity is checked and duplicates avoided.

Example: crossovers for binary representations
The optimization problem is transformed into the problem of finding the best path on a weighted graph $G(V, E)$ called construction graph.

The artificial ants incrementally build solutions by moving on the graph.

The solution construction process is stochastic, biased by a pheromone model, that is, a set of parameters associated with graph components (either nodes or edges) whose values are modified at runtime by the ants.

All pheromone trails are initialized to the same value, $\tau_0$.

At each iteration, pheromone trails are updated by decreasing (evaporation) or increasing (reinforcement) some trail levels on the basis of the solutions produced by the ants.

Example: A simple ACO algorithm for the TSP

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

$$p_{ij} = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in \Lambda_i^k} [\tau_{il}]^\alpha \cdot [\eta_{il}]^\beta}$$

where $\alpha$ and $\beta$ are parameters.

Update pheromone trail levels according to

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij} + \rho \cdot \text{Reward}$$

where $\Delta_{ij}(s) := 1/C^s$ if edge $(i, j)$ is contained in the cycle represented by $s'$, and 0 otherwise.

Motivation: Edges belonging to highest-quality candidate solutions and/or that have been used by many ants should be preferably used in subsequent constructions.

Termination: After fixed number of cycles ($= \text{construction} + \text{local search phases}$).