DM87
SCHEDULING,
TIMETABLING AND ROUTING

Constraint Programming, Heuristic Methods

Marco Chiarandini

Outline

1. Heuristic Methods
   - Construction Heuristics and Local Search
   - Solution Representations and Neighborhood Structures in LS
   - Metaheuristics
     - Metaheuristics for Construction Heuristics
     - Metaheuristics for Local Search and Hybrids

Outline

1. Heuristic Methods
   - Construction Heuristics and Local Search
   - Solution Representations and Neighborhood Structures in LS
   - Metaheuristics
     - Metaheuristics for Construction Heuristics
     - Metaheuristics for Local Search and Hybrids

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 := ∅
While s is not a complete solution:
  choose a solution component c
  add the solution component to s

Greedy best-first search

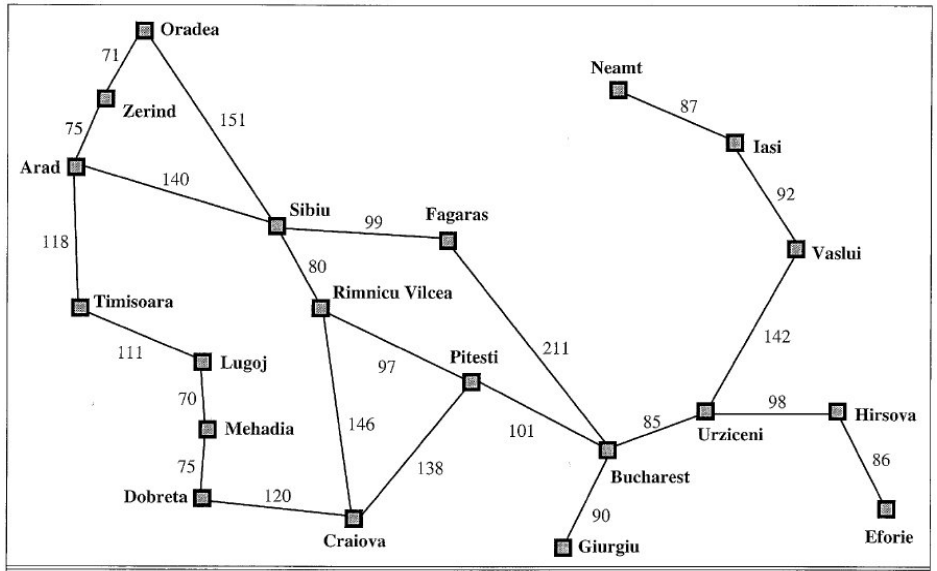


Figure 3.2 A simplified road map of part of Romania.

Greedy best-first search

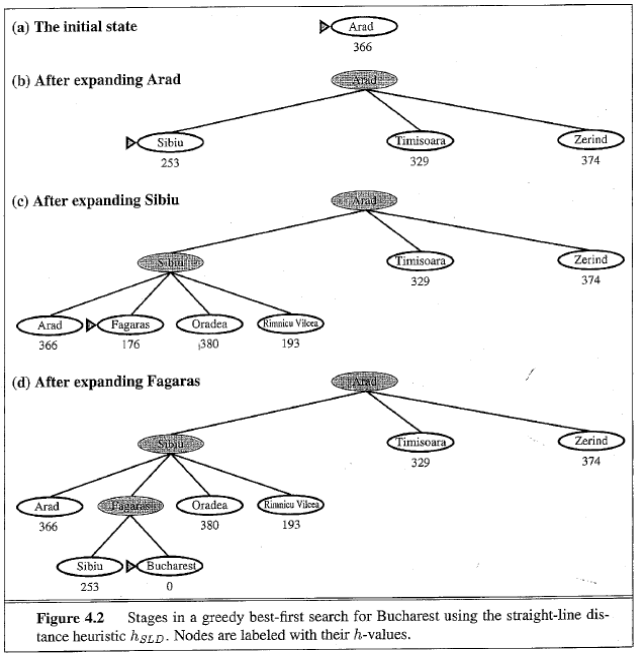


Figure 4.2 Stages in a greedy best-first search for Bucharest using the straight-line distance heuristic h_{SLD}. Nodes are labeled with their h-values.

- 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
    (Minimum Spanning Tree, Single Source Shortest Path,
    $1||\sum w_j C_j$, $1||L_{max}$)
  - 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 C.1. Summary of Dispatching Rules

	RULE	DATA	OBJECTIVES
Rules Dependent on Release Dates and Due Dates	*ERD*	r_j	Variance in Throughput Times
	EDD	d_j	Maximum Lateness
	MS	d_j	Maximum Lateness
Rules Dependent on Processing Times	*LPT*	p_j	Load Balancing over Parallel Machines
	SPT	p_j	Sum of Completion Times, WIP
	WSPT	p_j, w_j	Weighted Sum of Completion Times, WIP
	CP	p_j, *prec*	Makespan
	LNS	p_j, *prec*	Makespan
Miscellaneous	*SIRO*	–	Ease of Implementation
	SST	s_{jk}	Makespan and Throughput
	LFJ	M_j	Makespan and Throughput
	SQNO	–	Machine Idleness

Local Search

Example: Local Search for CSP

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

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

Local Search

Components

- solution representation
- initial solution
- neighborhood structure
- acceptance criterion

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:
  - $\mathcal{N} : S \times S \to \{T, F\}$
  - $\mathcal{N} \subseteq S \times S$
  - $\mathcal{N} : S \to 2^S$
- Neighborhood (set) of a candidate solution s: $N(s) := \{s' \in S \mid \mathcal{N}(s, s')\}$
- A neighborhood structure is also defined by an operator.
  An operator Δ is a collection of operator functions $\delta : S \to S$ such that

$$s' \in N(s) \iff \exists \delta \in \Delta \mid \delta(s) = s'$$

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 $f(\pi) : S(\pi) \mapsto \mathbb{R}$ that maps candidate solutions of a given problem instance π onto real numbers, such that global optima correspond to solutions of π;
- 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 $s' \in N(s)$.
- 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.

Example: Iterative Improvement

First improvement for TSP

procedure *TSP-2opt-first*(s)
  input: *an initial candidate tour* $s \in S(\in)$
  $\Delta = 0$;
  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**
        $\Delta_{ij} = d(c_i, c_j) + d(c_{i+1}, c_{j+1}) - d(c_i, c_{i+1}) - d(c_j, c_{j+1})$
        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 ι.

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

- π_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 π there exists a permutation such that $\pi^{-1} \cdot \pi = \iota$

$$\Delta_N \subset \Pi$$

Neighborhood Operators for Linear Permutations

Swap operator

$$\Delta_S = \{\delta_S^i | 1 \le i \le n\}$$

$$\delta_S^i(\pi_1 \dots \pi_i \pi_{i+1} \dots \pi_n) = (\pi_1 \dots \pi_{i+1} \pi_i \dots \pi_n)$$

Interchange operator

$$\Delta_X = \{\delta_X^{ij} | 1 \le i < j \le n\}$$

$$\delta_X^{ij}(\pi) = (\pi_1 \dots \pi_{i-1} \pi_j \pi_{i+1} \dots \pi_{j-1} \pi_i \pi_{j+1} \dots \pi_n)$$

Insert operator

$$\Delta_I = \{\delta_I^{ij} | 1 \le i \le n, 1 \le j \le n, j \ne i\}$$

$$\delta_I^{ij}(\pi) = \begin{cases} (\pi_1 \dots \pi_{i-1} \pi_{i+1} \dots \pi_j \pi_i \pi_{j+1} \dots \pi_n) & i < j \\ (\pi_1 \dots \pi_j \pi_i \pi_{j+1} \dots \pi_{i-1} \pi_{i+1} \dots \pi_n) & i > j \end{cases}$$

Neighborhood Operators for Circular Permutations

Reversal (2-edge-exchange)

$$\Delta_R = \{\delta_R^{ij} | 1 \le i < j \le n\}$$

$$\delta_R^{ij}(\pi) = (\pi_1 \dots \pi_{i-1} \pi_j \dots \pi_i \pi_{j+1} \dots \pi_n)$$

Block moves (3-edge-exchange)

$$\Delta_B = \{\delta_B^{ijk} | 1 \le i < j < k \le n\}$$

$$\delta_B^{ij}(\pi) = (\pi_1 \dots \pi_{i-1} \pi_j \dots \pi_k \pi_i \dots \pi_{j-1} \pi_{k+1} \dots \pi_n)$$

Short block move (Or-edge-exchange)

$$\Delta_{SB} = \{\delta_{SB}^{ij} | 1 \le i < j \le n\}$$

$$\delta_{SB}^{ij}(\pi) = (\pi_1 \dots \pi_{i-1} \pi_j \pi_{j+1} \pi_{j+2} \pi_i \dots \pi_{j-1} \pi_{j+3} \dots \pi_n)$$

Neighborhood Operators for Assignments

An assignment can be represented as a mapping
$\sigma : \{X_1 \dots X_n\} \to \{v : v \in D, |D| = k\}$:

$$\sigma = \{X_i = v_i, X_j = v_j, \dots\}$$

One exchange operator

$$\Delta_{1E} = \{\delta_{1E}^{il} | 1 \le i \le n, 1 \le l \le k\}$$

$$\delta_{1E}^{il}(\sigma) = \{\sigma : \sigma'(X_i) = v_l \text{ and } \sigma'(X_j) = \sigma(X_j) \ \forall j \ne i\}$$

Two exchange operator

$$\Delta_{2E} = \{\delta_{2E}^{ij} | 1 \le i < j \le n\}$$

$$\delta_{2E}^{ij}\{\sigma : \sigma'(X_i) = \sigma(X_j),\ \sigma'(X_j) = \sigma(X_i) \text{ and } \sigma'(X_l) = \sigma(X_l) \ \forall l \ne i, j\}$$

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

$$\Delta_{1E} = \{\delta_{1E}^{v} | v \in \overline{C}\}$$

$$\delta_{1E}^{v}(s) = \{s : C' = C \cup v \text{ and } \overline{C}' = \overline{C} \setminus v\}$$

One deletion operator

$$\Delta_{1E} = \{\delta_{1E}^{v} | v \in C\}$$

$$\delta_{1E}^{v}(s) = \{s : C' = C \setminus v \text{ and } \overline{C}' = \overline{C} \cup v\}$$

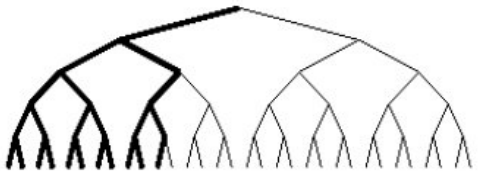
Swap operator

$$\Delta_{1E} = \{\delta_{1E}^{v} | v \in C, u \in \overline{C}\}$$

$$\delta_{1E}^{v}(s) = \{s : C' = C \cup u \setminus v \text{ and } \overline{C}' = \overline{C} \cup v \setminus u\}$$

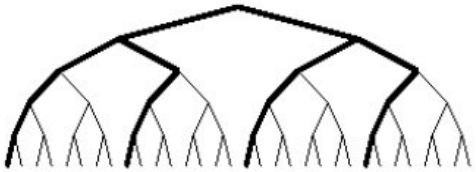
Construction Heuristics (Extensions)

Bounded-backtrack search:



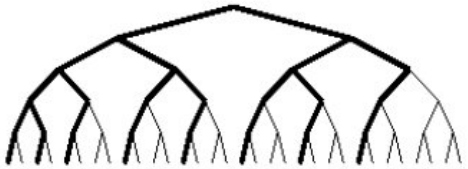
bbs(10)

Depth-bounded, then bounded-backtrack search:



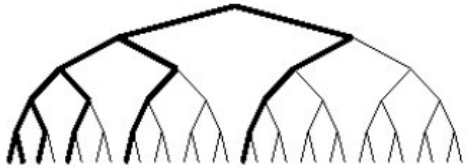
dbs(2, bbs(0))

Credit-based search:



credit(16)

Limited Discrepancy Search:



lds(1)

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 seemly 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

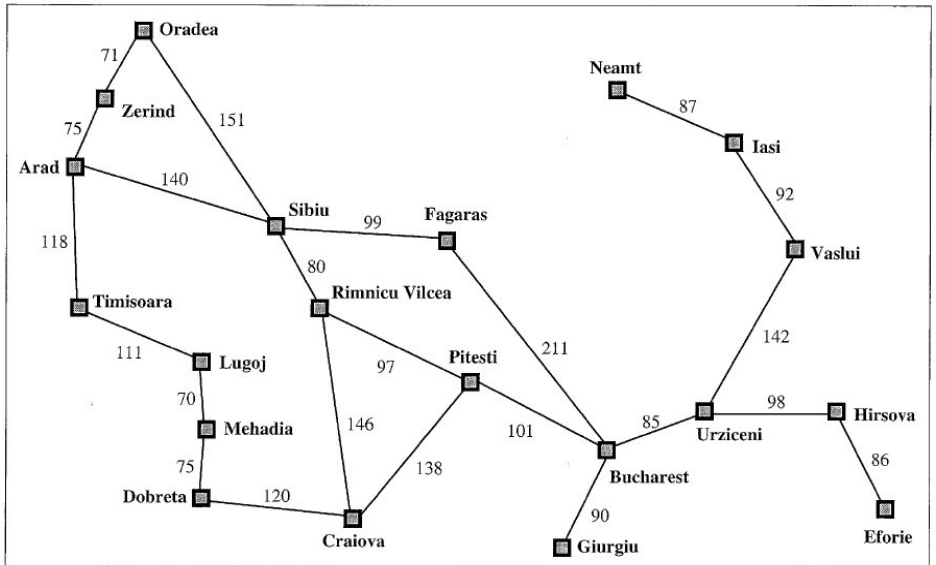


Figure 3.2 A simplified road map of part of Romania.

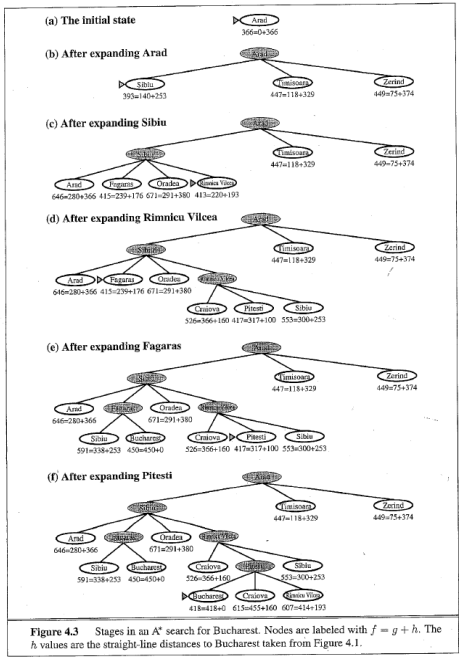


Figure 4.3 Stages in an A* search for Bucharest. Nodes are labeled with $f = g + h$. The h values are the straight-line distances to Bucharest taken from Figure 4.1.

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**
  $r := s$
  greedily destruct part of s
  greedily reconstruct the missing part of s
  based on acceptance criterion,
    keep s or revert to $s := r$

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 $h(l) \leq h_{min} + \alpha \cdot (h_{max} - h_{min})$, where h_{min} = minimal value of h and h_{max} = maximal value of h for any l. (α is a parameter of the algorithm.)

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

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:
  While maintain same temperature T 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*

Note:

- 2-stage neighbor selection procedure
  - proposal mechanism (often uniform random choice from $N(s)$)
  - acceptance criterion (often *Metropolis condition*)

$$p(T, s, s') := \begin{cases} 1 & \text{if } g(s') \leq f(s) \\ \exp \frac{f(s)-f(s')}{T} & \text{otherwise} \end{cases}$$

- Annealing schedule
  (function mapping run-time t onto temperature $T(t)$):
  - initial temperature T_0
    (may depend on properties of given problem instance)
  - temperature update scheme
    (*e.g.*, linear cooling: $T_{i+1} = T_0(1 - i/I_{max})$,
    geometric cooling: $T_{i+1} = \alpha \cdot T_i$)
  - number of search steps to be performed at each temperature
    (often multiple of neighborhood size)

- Termination predicate: often based on *acceptance ratio*,
  i.e., ratio of proposed *vs* accepted steps *or* number of idle iterations

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)

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:
  determine set N' of non-tabu neighbors of s
  choose a best improving candidate solution s' in N'
  update tabu attributes based on s'
  $s := s'$

Note:

- Non-tabu search positions in $N(s)$ are called *admissible neighbors of s*.
- After a search step, the current search position or the solution components just added/removed from it are declared *tabu* for a fixed number of subsequent search steps (*tabu tenure*).
- Often, an additional *aspiration criterion* is used: this specifies conditions under which tabu status may be overridden (*e.g.*, if considered step leads to improvement in incumbent solution).
- Crucial for efficient implementation:
  - keep time complexity of search steps minimal by using special data structures, incremental updating and caching mechanism for evaluation function values;
  - efficient determination of tabu status:
    store for each variable x the number of the search step when its value was last changed it_x; x is tabu if $it - it_x < tt$, where it = current search step number.

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:
  $r := s$
  perform *perturbation* on s
  perform *subsidiary local search* on s
  based on *acceptance criterion*,
    keep s or revert to $s := r$

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)

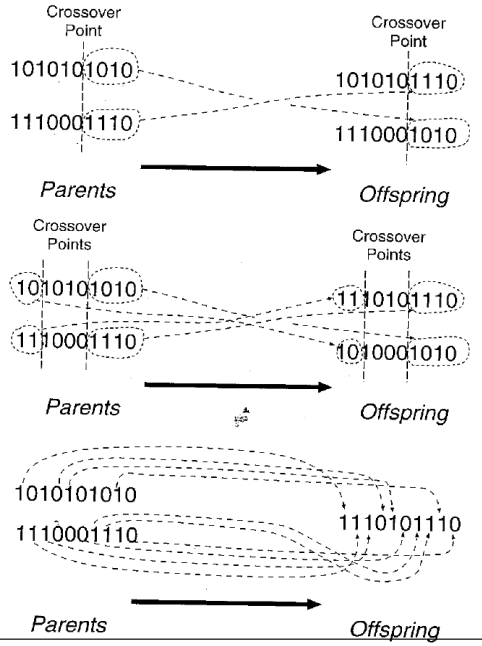
$$p_i = \frac{f_i}{\sum_j f_j}$$

- 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 cross over (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

Example: crossovers for binary representations



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

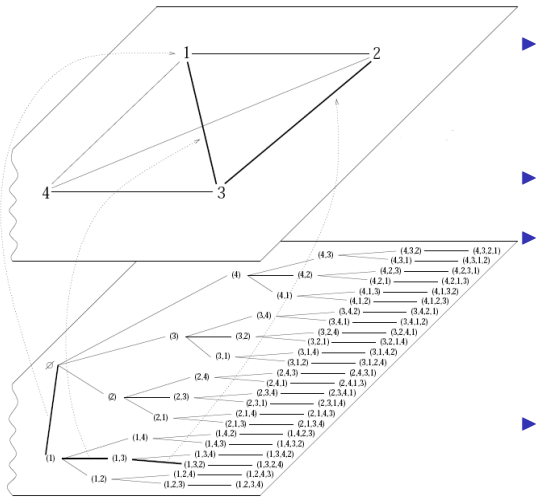
New Population

- Determines population for next cycle (*generation*) of the algorithm by selecting individual candidate solutions from current population + new candidate solutions obtained from *recombination*, *mutation* (+ *subsidiary local search*). (λ, μ) $(\lambda + \mu)$
- *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

The Metaheuristic

- 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, τ_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



- *Construction graph*
- To each edge ij in G associate
  - pheromone trails τ_{ij}
  - heuristic values $\eta_{ij} := \frac{1}{c_{ij}}$
- Initialize pheromones
- *Constructive search:*

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

- *Update pheromone trail levels*

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

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

- Search space and solution set as usual (all Hamiltonian cycles in given graph G).
- Associate pheromone trails τ_{ij} with each edge (i, j) in G.
- Use heuristic values $\eta_{ij} := \frac{1}{c_{ij}}$
- Initialize all weights to a small value τ_0 ($\tau_0 = 1$).
- *Constructive search:* Each ant starts with randomly chosen vertex and iteratively extends partial round trip π^k by selecting vertex not contained in π^k with probability

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

  α and β are parameters.

Example: A simple ACO algorithm for the TSP (2)

- *Subsidiary local search:* Perform iterative improvement based on standard 2-exchange neighborhood on each candidate solution in population (until local minimum is reached).
- *Update pheromone trail levels* according to

$$\tau_{ij} := (1-\rho) \cdot \tau_{ij} + \sum_{s \in sp'} \Delta_{ij}(s)$$

  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 (= construction + local search phases).