Metaheuristics and Local Search

Discrete optimization problems

- Variables $x_1, \dots, x_n$.
- Variable domains $D_1, \dots, D_n$, with $D_j \subseteq \mathbb{Z}$.
- Constraints $C_1, \dots, C_m$, with $C_i \subseteq D_1 \times \cdots \times D_n$.
- Objective function $f : D_1 \times \cdots \times D_n \to \mathbb{R}$, to be minimized.

Solution approaches

- Complete (exact) algorithms $\rightsquigarrow$ *systematic search*
  - Integer linear programming
  - Finite domain constraint programming
- Approximate algorithms
  - Heuristic approaches $\rightsquigarrow$ *heuristic search*
    - Constructive methods: construct solutions from partial solutions
    - **Local search:** improve solutions through neighborhood search
    - **Metaheuristics:** Combine basic heuristics in higher-level frameworks
  - Polynomial-time approximation algorithms for NP-hard problems

Metaheuristics

- Heuriskein ($\varepsilon\upsilon\rho\iota\sigma\kappa\varepsilon\iota\nu$): to find
- Meta: beyond, in an upper level
- *Survey paper:* C. Blum, A. Roli: Metaheuristics in Combinatorial Optimization, ACM Computing Surveys, Vol. 35, 2003.

Characteristics

- Metaheuristics are strategies that “guide” the search process.
- The goal is to efficiently explore the search space in order to find (near-) optimal solutions.
- Metaheuristic algorithms are approximate and usually non-deterministic.
- They may incorporate mechanisms to avoid getting trapped in confined areas of the search space.

Characteristics (2)

- The basic concepts of metaheuristics permit an abstract level description.
- Metaheuristics are not problem-specific.
- Metaheuristics may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy.

- Today more advanced metaheuristics use search experience (embodied in some form of memory) to guide the search.

Classification of metaheuristics

- Single point search (trajectory methods) vs. population-based search
- Nature-inspired vs. non-nature inspired
- Dynamic vs. static objective function
- One vs. various neighborhood structures
- Memory usage vs. memory-less methods

I. Trajectory methods

- Basic local search: iterative improvement
- Simulated annealing
- Tabu search
- Explorative search methods
  - Greedy Randomized Adaptive Search Procedure (GRASP)
  - Variable Neighborhood Search (VNS)
  - Guided Local Search (GLS)
  - Iterated Local Search (ILS)

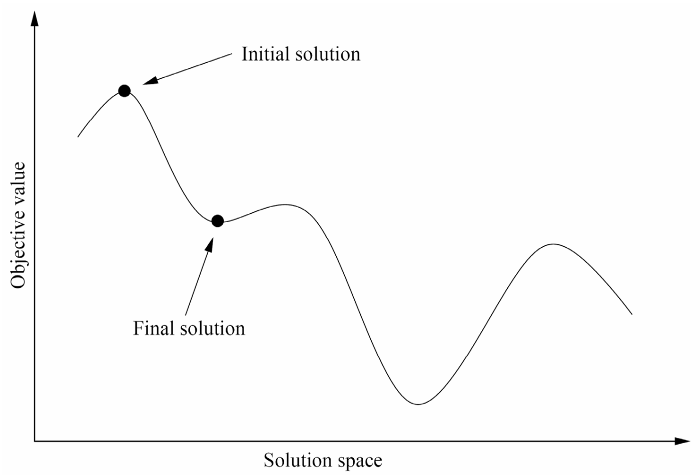
Local search

- Find an initial solution s
- Define a neighborhood $\mathcal{N}(s)$
- Explore the neighborhood
- Proceed with selected neighbor

Simple descent

```
procedure SimpleDescent(solution s)
   repeat
      choose s' ∈ N(s)
      if f(s') < f(s) then
         s ← s'
      end if
   until f(s') ≥ f(s), ∀s' ∈ N(s)
end
```

Local and global minima



Deepest descent

```
procedure DeepestDescent(solution s)
    repeat
        choose s' ∈ N(s) with f(s') ≤ f(s''), ∀s'' ∈ N(s)
        if f(s') < f(s) then
            s ← s'
        end if
    until f(s') ≥ f(s), ∀s' ∈ N(s)
end
```

Problem: Local minima

Multistart and deepest descent

```
procedure Multistart
    iter ← 1
    f(Best) ← ∞
    repeat
        choose a starting solution s_0 at random
        s ← DeepestDescent(s_0)
        if f(s) < f(Best) then
            Best ← s
        end if
        iter ← iter + 1
    until iter = IterMax
end
```

Simulated annealing

Kirkpatrick 83

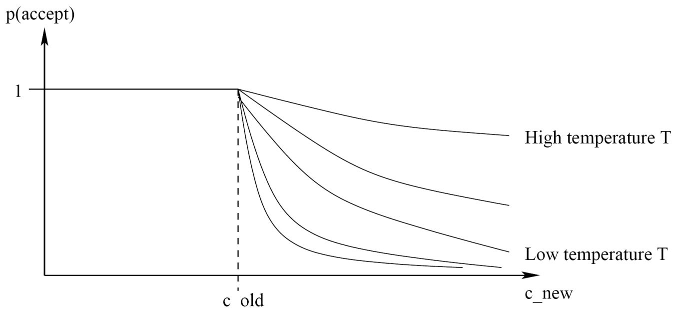
- *Anneal:* to heat and then slowly cool (esp. glass or metal) to reach minimal energy state
- Like standard local search, but sometimes accept worse solution.

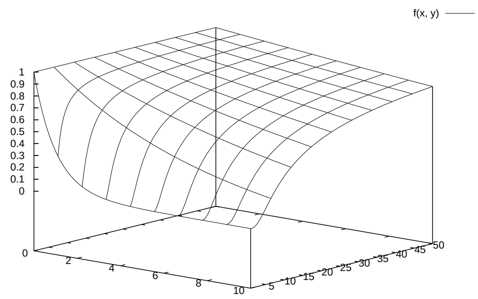
- Select random solution from the neighborhood and accept it with probability ⇝ *Boltzmann distribution*

$$p = \begin{cases} 1, & \text{if } f(new) < f(old), \\ exp(-(f(new) - f(old))/T), & \text{else.} \end{cases}$$

- Start with high *temperature* T, and gradually lower it ⇝ *cooling schedule*

Acceptance probability





Algorithm

$s \leftarrow$ GenerateInitialSolution()
$T \leftarrow T_0$
while termination conditions not met **do**
  $s' \leftarrow$ PickAtRandom($\mathcal{N}(s)$)
  if $(f(s') < f(s))$ **then**
    $s \leftarrow s'$
  else
    Accept s' as new solution with probability $p(T, s', s)$
  endif
  Update(T)
endwhile

Tabu search

Glover 86

- Local search with short term memory, to escape local minima and to avoid cycles.
- *Tabu list:* Keep track of the last r moves, and don't allow going back to these.
- *Allowed set:* Solutions that do not belong to the tabu list.
- Select solution from allowed set, add to tabu list, and update tabu list.

Basic algorithm

$s \leftarrow$ GenerateInitialSolution()
$\mathit{TabuList} \leftarrow \emptyset$
while termination conditions not met **do**
  $s \leftarrow$ ChooseBestOf($\mathcal{N}(s) \setminus \mathit{TabuList}$)
  Update($\mathit{TabuList}$)
endwhile

Choices in tabu search

- Neighborhood
- Size of tabu list $\rightsquigarrow$ *tabu tenure*
- Kind of tabu to use (complete solutions vs. attributes)
  $\rightsquigarrow$ *tabu conditions*
- *Aspiration criteria*
- Termination condition
- Long-term memory: recency, frequency, quality, influence

Refined algorithm

$s \leftarrow$ GenerateInitialSolution()
Initialize TabuLists ($TL_1, ..., TL_r$)
$k \leftarrow 0$
while termination conditions not met **do**
  $\mathit{AllowedSet}(s,k) \leftarrow \{s' \in \mathcal{N}(s) \mid$
    s does not violate a tabu condition
    or satisfies at least one aspiration condition }
  $s \leftarrow$ ChooseBestOf($\mathit{AllowedSet}(s,k)$)
  UpdateTabuListsAndAspirationConditions()
  $k \leftarrow k+1$
endwhile

II. Population-based search

- Evolutionary computation
- Ant colony optimization

Evolutionary computation

- Idea: Mimic evolution - obtain better solutions by combining current ones.

- Keep several current solutions, called *population* or *generation.*
- Create new generation:
  - select a pool of promising solutions, based on a *fitness function.*
  - create new solutions by combining solutions in the pool in various ways $\rightsquigarrow$ *recombination, crossover.*
  - add random *mutations.*
- *Variants:* Evolutionary programming, evolutionary strategies, genetic algorithms

Algorithm

$P \leftarrow$ GeneralInitialPopulation()
Evaluate(P)
while termination conditions not met **do**
  $P' \leftarrow$ Recombine(P)
  $P'' \leftarrow$ Mutate(P')
  Evaluate(P'')
  $P \leftarrow$ Select($P'' \cup P$)
endwhile

Crossover and mutations

- Individuals (solutions) often coded as bit vectors
- *Crossover* operations provide new individuals, e.g.

$$\begin{array}{l|l} 101101 & \mathit{0110} \\ 000110 & \mathbf{1011} \end{array} \rightsquigarrow \begin{array}{l|l} 101101 & \mathbf{1011} \\ 000110 & \mathit{0110} \end{array}$$

- *Mutations* often helpful, e.g., swap random bit.

Further issues

- Individuals vs. solutions
- Evolution process: generational replacement vs. steady state, fixed vs. variable population size
- Use of neighborhood structure to define recombination partners (structured vs. unstructured populations)
- Two-parent vs. multi-parent crossover
- Infeasible individuals: reject/penalize/repair
- Intensification by local search
- Diversification by mutations

Ant colony optimization

Dorigo 92

- Observation: Ants are able to find quickly the shortest path from their nest to a food source $\rightsquigarrow$ how ?
- Each ant leaves a *pheromone* trail.
- When presented with a path choice, they are more likely to choose the trail with higher pheromone concentration.

- The shortest path gets high concentrations because ants choosing it can return more often.

Ant colony optimization (2)

- Ants are simulated by individual (ant) agents $\rightsquigarrow$ *swarm intelligence*
- Each decision variable has an associated artificial *pheromone level*.
- By dispatching a number of ants, the pheromone levels are adjusted according to how useful they are.
- Pheromone levels may also *evaporate* to discourage suboptimal solutions.

Construction graph

- Complete graph $G = (C, L)$
  - C solution components
  - L connections
- Pheromone trail values τ_i, for $c_i \in C$.
- Heuristic values η_i
- Moves in the graph depend on transition probabilities

$$p(c_r \mid s_a[c_l]) = \begin{cases} \dfrac{[\eta_r]^\alpha[\tau_r]^\beta}{\sum\limits_{c_u \in J(s_a[c_l])} [\eta_u]^\alpha[\tau_u]^\beta} & \text{if } c_r \in J(s_a[c_l]) \\ 0 & \text{otherwise} \end{cases}$$

Algorithm (ACO)

```
InitializePheromoneValues
while termination conditions not met do
  ScheduleActivities
    AntBasedSolutionConstruction()
    PheromoneUpdate()
    DaemonActions()          % optional
  endScheduleActivities
endwhile
```

Pheromone Update

Set

$$\tau_j = (1 - \rho)\tau_j + \sum_{a \in A} \Delta\tau_j^{s_a} \ ,$$

where

$$\Delta\tau_j^{s_a} = \begin{cases} F(s_a) & \text{if } c_j \text{ is component of } s_a \\ 0 & \text{otherwise} \ . \end{cases}$$

Intensification and diversification

Glover and Laguna 1997

The main difference between intensification and diversification is that during an intensification stage the search focuses on examining neighbors of elite solutions. ... The diversification stage on the other hand encourages the search process to examine unvisited regions and to generate solutions that differ in various significant ways from those seen before.

Case study: Time tabling

Rossi-Doria et al. 2002 `http://iridia.ulb.ac.be/~meta/newsite/downloads/tt_comparison.pdf`

- Set of events E, set of rooms R, set of students S, set of features F
- Each student attends a number of events and each room has a size.
- Assign all events a timeslot and a room so that the following *hard constraints* are satisfied:
  - no student attends more than one event at the same time.
  - the room is big enough for all attending students and satisfies all features required by the event.
  - only one event is in each room at any timeslot.

Case study: Time tabling (2)

- Penalties for *soft constraint* violations
  - a student has a class in the last slot of a day.
  - a student has more than two classes in a row.
  - a student has a single class on a day.
- Objective: Minimize number of soft constraint violations in a feasible solution

Common neighborhood structure

- *Solution* $\rightsquigarrow$ ordered list of length $|E|$
  The i-th element indicates the timeslot to which event i is assigned.
- Room assignments generated by matching algorithm.
- *Neighborhood:* $N = N_1 \cup N_2$
  - N_1 moves a single event to a different timeslot
  - N_2 swaps the timeslots of two events.

Common local search procedure

Stochastic first improvement local search

- Go through the list of all the events in a random order.
- Try all the possible moves in the neighbourhood for every event involved in constraint violations, until improvement is found.
- Solve hard constraint violations first.
  If feasibility is reached, look at soft constraint violations as well.

Metaheuristics

1. Evolutionary algorithm
2. Ant colony optimization
3. Iterated local search
4. Simulated annealing
5. Tabu search

1. Evolutionary algorithm

- *Steady-state evolution process:* at each generation only one couple of parent individuals is selected for reproduction.
- *Tournament selection:* choose randomly a number of individuals from the current population and select the best ones in terms of fitness function as parents.
- *Fitness function:* Weighted sum of hard and soft constraint violations,

$$f(s) := \#hcv(s) \cdot C + \#scv(s)$$

1. Evolutionary algorithm (2)

- *Uniform crossover:* for each event a timeslot's assignment is inherited from the first or second parent with equal probability.
- *Mutation:* Random move in an extended neighbourhood (3-cycle permutation).
- *Search parameters:* Population size $n = 10$, tournament size = 5, crossover rate $\alpha = 0.8$, mutation rate $\beta = 0.5$
- Find a balance between the number of steps in local search and the number of generations.

2. Ant colony optimization

- At each iteration, *each of m ants constructs,* event by event, a *complete assignment* of the events to the timeslots.
- To make an assignment, an ant takes the next event from a pre-ordered list, and probabilistically chooses a timeslot, guided by two types of information:
  1. *heuristic information:* evaluation of the constraint violations caused by making the assignment, given the assignments already made,
  2. *pheromone information:* estimate of the utility of making the assignment, as judged by previous iterations of the algorithm.
- *Matrix* of pheromone values $\tau : E \times T \to \mathbb{R}_{\geq 0}$.
  Initialization to a parameter τ_0, update by local and global rules.

2. Ant colony optimization (2)

- An event-timeslot pair which has been part of good solutions will have a high pheromone value, and consequently have a higher chance of being chosen again.

- At the end of the iterative construction, an event-timeslot assignment is converted into a candidate solution (timetable) using the matching algorithm.
- This candidate solution is further improved by the local search routine.
- After all m ants have generated their candidate solution, a global update on the pheromone values is performed using the best solution found since the beginning.

3. Iterated local search

- Provide new starting solutions obtained from **perturbations** of a current solution
- Often leads to far better results than using random restart.
- Four subprocedures
  1. *GenerateInitialSolution:* generates an initial solution s_0
  2. *Perturbation:* modifies the current solution s leading to some intermediate solution s',
  3. *LocalSearch:* obtains an improved solution s'',
  4. *AcceptanceCriterion:* decides to which solution the next perturbation is applied.

Perturbation

- *Three types of moves*

  P1: choose a different timeslot for a randomly chosen event;

  P2: swap the timeslots of two randomly chosen events;

  P3: choose randomly between the two previous types of moves and a 3-exchange move of timeslots of three randomly chosen events.
- *Strategy*
  - Apply each of these different moves k times, where k is chosen of the set $\{1; 5; 10; 25; 50; 100\}$.
  - Take random choices according to a uniform distribution.

Acceptance criteria

- Random walk: Always accept solution returned by local search
- Accept if better
- Simulated annealing

  SA1: $P_1(s, s') = e^{-\frac{f(s)-f(s')}{T}}$

  SA2: $P_2(s, s') = e^{-\frac{f(s)-f(s')}{T \cdot f(s_{best})}}$

Best parameter setting (for medium instances):

P1, $k = 5$, **SA1** with $T = 0.1$

4. Simulated annealing

Two phases

1. Search for feasible solutions, i.e., satisfy all hard constraints.
2. Minimize soft constraint violations.

Strategies

- *Initial temperature:* Sample the neighbourhood of a randomly generated solution, compute average value of the variation in the evaluation function, and multiply this value by a given factor.
- *Cooling schedule*
  1. Geometric cooling: $T_{n+1} = \alpha \times T_n, \quad 0 < \alpha < 1$
  2. Temperature reheating: Increase temperature if *rejection ratio* (= number of moves rejected/number of moves tested) exceeds a given limit.
- *Temperature length:* Proportional to the size of the neighborhood

5. Tabu search

- Moves done by moving one event or by swapping two events.
- *Tabu list:* Forbid a move if at least one of the events involved has been moved less than l steps before.
- *Size of tabu list l:* number of events divided by a suitable constant k (here $k = 100$).
- *Variable neighbourhood set:* every move is a neighbour with probability 0.1 $\rightsquigarrow$ decrease probability of generating cycles and reduce the size of neighbourhood for faster exploration.
- *Aspiration criterion:* perform a tabu move if it improves the best known solution.

Evaluation

`http://iridia.ulb.ac.be/~msampels/ttmn.data/`

- 5 small, 5 medium, 2 large instances

Type	small	medium	large
$\|E\|$	100	400	400
$\|S\|$	80	200	400
$\|R\|$	5	10	10

- 500 resp. 50 resp. 20 independent trials per metaheuristic per instance.
- Diagrams show results of all trials on a single instance.
- Boxes show the range between 25% and 75% quantile.

Evaluation (2)

- *Small:* All algorithms reach feasibility in every run,
  ILS best, TS worst overall performance
- *Medium:* SA best, but does not achieve feasibility in some runs. ACO worst.
- *Large01:* Most metaheuristics do not even achieve feasibility. TS feasibility in about 8% of the trials.

- *Large02:* ILS best, feasibility in about 97% of the trials, against 10% for ACO and GA. SA never reaches feasibility. TS gives always feasible solutions, but with worse results than ILS and AC0 in terms of soft constraints.