Metaheuristics and Local Search

Discrete optimization problems

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

Solution approaches

- Complete (exact) algorithms \( \Rightarrow \) systematic search
  - Integer linear programming
  - Finite domain constraint programming
- Approximate algorithms
  - Heuristic approaches \( \Rightarrow \) 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 (ευρισκειν): to find
- Meta: beyond, in an upper level

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.
- Metaheuristics range from simple local search to complex learning procedures.
- Metaheuristic algorithms are approximate and usually non-deterministic.
- They may incorporate mechanisms to avoid getting trapped in confined areas of the search space.
• Today more advanced metaheuristics use search experience (embodied in some form of memory) to guide
the search.

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.

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 \( s' \in \mathcal{N}(s) \)
  if \( f(s') < f(s) \) then
    \( s \leftarrow s' \)
  end if
until \( f(s') \geq f(s), \forall s' \in \mathcal{N}(s) \)
end
Deepest descent

procedure DeepestDescent(solution s)
    repeat
        choose $s' \in \mathcal{N}(s)$ with $f(s') \leq f(s'), \forall s'' \in \mathcal{N}(s)$
        if $f(s') < f(s)$ then
            $s \leftarrow s'$
        end if
    until $f(s') \geq f(s), \forall s' \in \mathcal{N}(s)$
end

Problem: Local minima

Local and global minima

Multistart and deepest descent

procedure Multistart
    iter $\leftarrow 1$
    $f(Best) \leftarrow \infty$
    repeat
        choose a starting solution $s_0$ at random
        $s \leftarrow$ DeepestDescent($s_0$)
        if $f(s) < f(Best)$ then
            Best $\leftarrow s$
        end if
        iter $\leftarrow$ 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 \( \rightarrow \) Boltzmann distribution

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

• Start with high temperature \( T \), and gradually lower it \( \rightarrow \) cooling schedule

**Acceptance probability**

![Graph showing the acceptance probability](image)

\( p(\text{accept}) \)

**Algorithm**

\[
s \leftarrow \text{GenerateInitialSolution}() \\
T \leftarrow T_0 \\
\textbf{while} \text{ termination conditions not met} \quad \textbf{do} \\
\quad s' \leftarrow \text{PickAtRandom}(\mathcal{X}(s)) \\
\quad \text{if } (f(s') < f(s)) \text{ then} \\
\quad \quad s \leftarrow s' \\
\quad \text{else} \\
\quad \quad \text{Accept } s' \text{ as new solution with probability } p(T, s', s) \\
\quad \textbf{endif} \\
\quad \text{Update}(T) \\
\textbf{endwhile}
\]

**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**
\begin{verbatim}
s ← GenerateInitialSolution()
TabuList ← 0
while termination conditions not met do
    s ← ChooseBestOf(\mathcal{N}(s) \setminus TabuList)
    Update(TabuList)
endwhile

Choices in tabu search

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

Refined algorithm

s ← GenerateInitialSolution()
Initialize TabuLists \((\text{TL}_1, ..., \text{TL}_r)\)
k ← 0
while termination conditions not met do
    AllowedSet(s, k) ← \{ s' ∈ \mathcal{N}(s) | s' does not violate a tabu condition or satisfies at least one aspiration condition \}
    s ← ChooseBestOf(AllowedSet(s, k))
    UpdateTabuListsAndAspirationConditions()
    k ← k + 1
endwhile

II. Population-based search

Use a set (i.e. a population) of solutions rather than a single solutions

- 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 \( \leftrightarrow \) recombination, crossover.
  - add random mutations.
\end{verbatim}
Variants: Evolutionary programming, evolutionary strategies, genetic algorithms

Algorithm

\[ P \leftarrow \text{GenerateInitialPopulation()}
\]
\[ \text{Evaluate}(P)\]
\[ \text{while} \text{ termination conditions not met do} \]
\[ P' \leftarrow \text{Recombine}(P) \]
\[ P'' \leftarrow \text{Mutate}(P') \]
\[ \text{Evaluate}(P'') \]
\[ P \leftarrow \text{Select}(P'' \cup P) \]
\[ \text{endwhile} \]

Crossover and mutations

- Individuals (solutions) often coded as bit or integer vectors
- Crossover operations provide new individuals, e.g.
  
  \[
  \begin{array}{c|c|c}
  101101 & 0110 & 101101 \\
  000110 & 1011 & 000110 \\
  \end{array}
  \]
- Mutations often helpful, e.g., swap random bit.

Further issues

- Individuals ("genotypes") vs. solutions ("phenotypes"): individuals are not necessarily solutions
- Evolution process: generational replacement vs. steady state fixed vs. variable population size
- Use of neighborhood structure to define recombination partners
- 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 \( \rightarrow \text{how?} \)
- 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 \( \text{(2)} \)

- Ants are simulated by individual (ant) agents \( \rightarrow \text{swarm intelligence} \)
• Artificial ants incrementally construct solutions by adding components to a partial solution.
• By dispatching a number of ants, the pheromone levels associated with the components 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 $\tau_i, \tau_{ij} \in \mathcal{T}$, for $c_i \in C, l_{ij} \in L$
- Heuristic values $\eta_i, \eta_{ij} \in \mathcal{H}$
- Moves in the graph depend on transition probabilities (use only $\tau_i, \eta_i$)

$$p(c_r \mid s_a[c_i]) = \begin{cases} \frac{[\eta_r]^{\alpha}[\tau_r]^{\beta}}{\sum_{u \in J(s_a[c_i])}[\eta_u]^{\alpha}[\tau_u]^{\beta}}, & \text{if } c_r \in J(s_a[c_i]), \\ 0, & \text{otherwise}. \end{cases}$$

$s_a$ denotes the solution constructed by ant $a$, $c_r$ its last component. $J(s_a[c_i])$ denotes the set of components allowed to be added.

**Algorithm: Ant System (AS)**

InitializePheromoneValues
while termination conditions not met do
  for all ants $a \in \mathcal{A}$ do
    $s_a \leftarrow \text{ConstructSolution}(\mathcal{T}, \mathcal{H})$
  endfor
  ApplyOnlineDelayedPheromoneUpdate($\mathcal{T}, \{s_a \mid a \in \mathcal{A}\}$)
endwhile

• ApplyOnlineDelayedPheromoneUpdate $\Rightarrow$ increase of pheromone on solutions components found in high-quality solutions
• The Ant System (AS) algorithm may be generalized to the Ant Colony Optimization (ACO) Metaheuristics.