

DM841  
Discrete Optimization

Lecture 8  
**Stochastic Local Search and Metaheuristics**  
**Simulated Annealing, Tabu Search, Variable**  
**Neighborhood Search**

Marco Chiarandini

Department of Mathematics & Computer Science  
University of Southern Denmark

1. Vertex Coloring
2. Stochastic Local Search
  - ▶ Randomized Iterative Improvement

1. Trajectory Based Metaheuristics
  - Stochastic Local Search
  - Simulated Annealing
  - Iterated Local Search
  - Tabu Search
  - Variable Neighborhood Search
  - Guided Local Search
  
2. Population Based Metaheuristics
  - Evolutionary Algorithms
  - Ant Colony Optimization

1. Trajectory Based Metaheuristics
  - Stochastic Local Search
    - Simulated Annealing
    - Iterated Local Search
    - Tabu Search
    - Variable Neighborhood Search
    - Guided Local Search
2. Population Based Metaheuristics
  - Evolutionary Algorithms
  - Ant Colony Optimization

For a general CSP:

**procedure** *MCH* (*P*, *maxSteps*)

**input:** *CSP instance P*, *positive integer maxSteps*

**output:** *solution of P or “no solution found”*

*a* := randomly chosen assignment of the variables in *P*;

**for** *step* := 1 **to** *maxSteps* **do**

**if** *a* satisfies all constraints of *P* **then return** *a* **end**

*x* := randomly selected variable from conflict set  $K(a)$ ;

*v* := randomly selected value from the domain of *x* such that

        setting *x* to *v* minimises the number of unsatisfied constraints;

*a* := *a* with *x* set to *v*;

**end**

**return** “no solution found”

**end** *MCH*

On the N-queens problem:

```
import cotls;
int n = 16;
range Size = 1..n;
UniformDistribution distr(Size);

Solver<LS> m();
var{int} queen[Size](m,Size) := distr.get();
ConstraintSystem<LS> S(m);

S.post(alldifferent(queen));
S.post(alldifferent(all(i in Size) queen[i] + i));
S.post(alldifferent(all(i in Size) queen[i] - i));
m.close();

int it = 0;
while (S.violations() > 0 && it < 50 * n) {
  select(q in Size : S.violations(queen[q])>0) {
    selectMin(v in Size)(S.getAssignDelta(queen[q],v)) {
      queen[q] := v;
      cout << "chng @ " << it << ": queen[" << q << "] := " << v << " viol: " << S.violations()
      << endl;
    }
  }
  it = it + 1;
}
cout << queen << endl;
```

# Min-Conflict + Random Walk for SAT

```
procedure WalkSAT ( $F, \text{maxTries}, \text{maxSteps}, \text{slc}$ )  
  input: CNF formula  $F$ , positive integers  $\text{maxTries}$  and  $\text{maxSteps}$ ,  
    heuristic function  $\text{slc}$   
  output: model of  $F$  or 'no solution found'  
  for  $\text{try} := 1$  to  $\text{maxTries}$  do  
     $a :=$  randomly chosen assignment of the variables in formula  $F$ ;  
    for  $\text{step} := 1$  to  $\text{maxSteps}$  do  
      if  $a$  satisfies  $F$  then return  $a$  end  
       $c :=$  randomly selected clause unsatisfied under  $a$ ;  
       $x :=$  variable selected from  $c$  according to heuristic function  $\text{slc}$ ;  
       $a := a$  with  $x$  flipped;  
    end  
  end  
  return 'no solution found'  
end WalkSAT
```

Example of  $\text{slc}$  heuristic: with prob.  $wp$  select a random move, with prob.  $1 - wp$  select the best

**Key idea:** Accept worsening steps with probability that depends on respective deterioration in evaluation function value:  
bigger deterioration  $\cong$  smaller probability

*Realization:*

- ▶ Function  $p(f, s)$ : determines probability distribution over neighbors of  $s$  based on their values under evaluation function  $f$ .
- ▶ Let  $\text{step}(s, s') := p(f, s, s')$ .

*Note:*

- ▶ Behavior of PII crucially depends on choice of  $p$ .
- ▶ II and RII are special cases of PII.



## Example: Metropolis PII for the TSP

- ▶ **Search space  $S$** : set of all Hamiltonian cycles in given graph  $G$ .
- ▶ **Solution set**: same as  $S$
- ▶ **Neighborhood relation  $\mathcal{N}(s)$** : 2-edge-exchange
- ▶ **Initialization**: an Hamiltonian cycle uniformly at random.
- ▶ **Step function**: implemented as 2-stage process:
  1. select neighbor  $s' \in \mathcal{N}(s)$  uniformly at random;
  2. accept as new search position with probability:

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

(**Metropolis condition**), where *temperature* parameter  $T$  controls likelihood of accepting worsening steps.

- ▶ **Termination**: upon exceeding given bound on run-time.

Another version of probabilistic iterative improvement:

**Key Idea:** start with a minimum water level,

- ▶ at each step a candidate move is generated at random, the move is accepted if its quality is greater than the water level.
- ▶ After the number of neighbors have been sampled at a certain water level, the water level is updated.
- ▶ The algorithm stops if we have reached the maximum water level or if we have done a certain number of non-improving solutions.

## In greatdeluge.hh

```
InitializeRun()
```

```
{  
    level = initial_level * this->current_state_cost;  
}
```

```
SelectMove()
```

```
{  
    this->SelectRandomMove();  
}
```

```
StopCriterion()
```

```
{  
    return level < min_level * this->best_state_cost;  
}
```

```
UpdateIterationCounter()
```

```
{  
    if (this->number_of_iterations % neighbors_sampled == 0)  
    {  
        level *= level_rate;  
    }  
}
```

```
AcceptableMove()
```

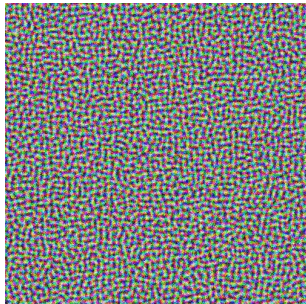
```
{  
    return LessOrEqualThan(this->current_move_cost,(CFtype)0) || LessOrEqualThan(((double)(  
        this->current_move_cost + this->current_state_cost),level);  
}
```

1. Trajectory Based Metaheuristics
  - Stochastic Local Search
  - Simulated Annealing**
  - Iterated Local Search
  - Tabu Search
  - Variable Neighborhood Search
  - Guided Local Search
  
2. Population Based Metaheuristics
  - Evolutionary Algorithms
  - Ant Colony Optimization

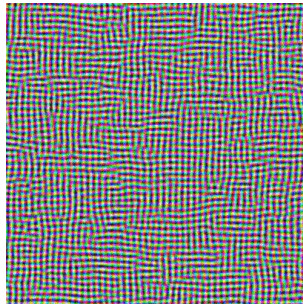
## Inspired by statistical mechanics in matter physics:

- ▶ candidate solutions  $\cong$  states of physical system
- ▶ evaluation function  $\cong$  thermodynamic energy
- ▶ globally optimal solutions  $\cong$  ground states
- ▶ parameter  $T \cong$  physical temperature

*Note:* In physical process (e.g., annealing of metals), perfect ground states are achieved by very slow lowering of temperature.



fast



slow

**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: **do**

**while** maintain same temperature  $T$  according to **annealing schedule** **do**

        probabilistically choose a neighbor  $s'$  of  $s$  using **proposal mechanism**

**if**  $s'$  satisfies probabilistic **acceptance criterion** (depending on  $T$ ) **then**

$s := s'$

    update  $T$  according to **annealing schedule**

- ▶ 2-stage step function based on
  - ▶ proposal mechanism (often uniform random choice from  $N(s)$ )
  - ▶ acceptance criterion (often *Metropolis condition*)
- ▶ 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)
  - ▶ may be *static* or *dynamic*
  - ▶ seek to balance moderate execution time with asymptotic behavior properties
- ▶ Termination predicate: often based on *acceptance ratio*,  
*i.e.*, ratio accepted / proposed steps or number of idle iterations

## Example: Simulated Annealing for TSP

Extension of previous PII algorithm for the TSP, with

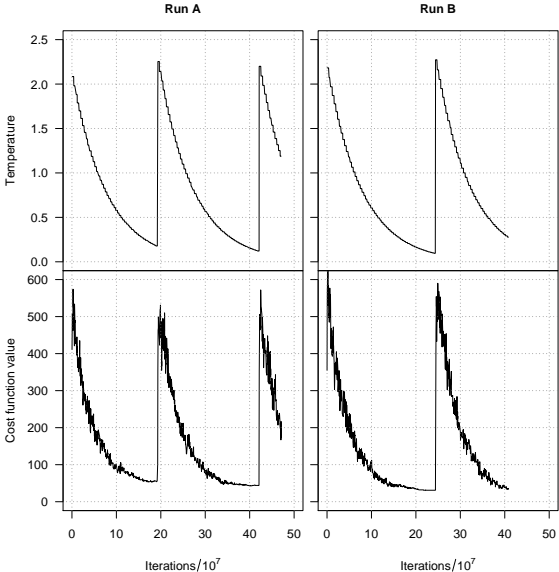
- ▶ **proposal mechanism:** uniform random choice from 2-exchange neighborhood;
- ▶ **acceptance criterion:** Metropolis condition (always accept improving steps, accept worsening steps with probability  $\exp[-(f(s') - f(s))/T]$ );
- ▶ **annealing schedule:** geometric cooling  $T := 0.95 \cdot T$  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)



# Profiling



1. Trajectory Based Metaheuristics
  - Stochastic Local Search
  - Simulated Annealing
  - Iterated Local Search**
  - Tabu Search
  - Variable Neighborhood Search
  - Guided Local Search
  
2. Population Based Metaheuristics
  - Evolutionary Algorithms
  - Ant Colony Optimization

# 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 **do**

$r := s$

    perform **perturbation** on  $s$

    perform **subsidiary local search** on  $s$

    based on **acceptance criterion**,

    keep  $s$  or revert to  $s := r$

## Note:

- ▶ *Subsidiary local search* results in a local minimum.
- ▶ ILS trajectories can be seen as walks in the space of local minima of the given evaluation function.
- ▶ *Perturbation phase* and *acceptance criterion* may use aspects of *search history* (i.e., limited memory).
- ▶ In a high-performance ILS algorithm, *subsidiary local search*, *perturbation mechanism* and *acceptance criterion* need to complement each other well.

## Subsidiary local search:

- ▶ More effective subsidiary local search procedures lead to better ILS performance.  
*Example: 2-opt vs 3-opt vs LK for TSP.*
- ▶ Often, subsidiary local search = iterative improvement, but more sophisticated LS methods can be used. (e.g., Tabu Search).

## Perturbation mechanism:

- ▶ Needs to be chosen such that its effect *cannot* be easily undone by subsequent local search phase.  
(Often achieved by search steps larger neighborhood.)  
*Example:* local search = 3-opt, perturbation = 4-exchange steps in ILS for TSP.
- ▶ A perturbation phase may consist of one or more perturbation steps.
- ▶ Weak perturbation  $\Rightarrow$  short subsequent local search phase;  
**but:** risk of revisiting current local minimum.
- ▶ Strong perturbation  $\Rightarrow$  more effective escape from local minima;  
**but:** may have similar drawbacks as random restart.
- ▶ Advanced ILS algorithms may change nature and/or strength of perturbation adaptively during search.

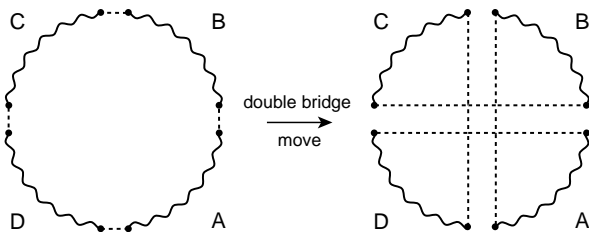
## Acceptance criteria:

- ▶ Always accept the **best** of the two candidate solutions  
⇒ ILS performs Iterative Improvement in the space of local optima reached by subsidiary local search.
- ▶ Always accept the **most recent** of the two candidate solutions  
⇒ ILS performs random walk in the space of local optima reached by subsidiary local search.
- ▶ Intermediate behavior: select between the two candidate solutions based on the *Metropolis criterion* (e.g., used in *Large Step Markov Chains* [Martin et al., 1991]).
- ▶ Advanced acceptance criteria take into account search history, e.g., by occasionally reverting to *incumbent solution*.

# Examples

Example: Iterated Local Search for the TSP (1)

- ▶ **Given:** TSP instance  $\pi$ .
- ▶ **Search space:** Hamiltonian cycles in  $\pi$ .
- ▶ **Subsidiary local search:** Lin-Kernighan variable depth search algorithm
- ▶ **Perturbation mechanism:**  
'double-bridge move' = particular 4-exchange step:



- ▶ **Acceptance criterion:** Always return the best of the two given candidate round trips.



1. Trajectory Based Metaheuristics
  - Stochastic Local Search
  - Simulated Annealing
  - Iterated Local Search
  - Tabu Search**
  - Variable Neighborhood Search
  - Guided Local Search
  
2. Population Based Metaheuristics
  - Evolutionary Algorithms
  - Ant Colony Optimization

**Key idea:** Avoid repeating history (memory)

How can we remember the history without

- ▶ memorizing full solutions (space)
- ▶ computing hash functions (time)

~> use attributes

**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 candidate solution  $s'$  in  $N'$

update **tabu attributes** based on  $s'$   
 $s := s'$

## Example: Tabu Search for SAT

- ▶ **Search space:** set of all complete assignments of  $X$ .
- ▶ **Solution set:** models of the formula.
- ▶ **Neighborhood relation:** 1-flip
- ▶ **Memory:** Associate tabu status (Boolean value) with each pair (literal,value)  $(x, val)$ .
- ▶ **Initialization:** a random assignment
- ▶ **Search steps:**
  - ▶ pairs  $(x, v)$  are tabu if they have been changed in the last  $tt$  steps;
  - ▶ neighboring assignments are admissible if they can be reached by changing a non-tabu pair or have fewer unsatisfied constraints than the best assignments seen so far (*aspiration criterion*);
  - ▶ choose uniformly at random admissible neighbors with minimal number of unsatisfied constraints.
- ▶ **Termination:** upon finding a feasible assignment *or* after given bound on number of search steps has been reached *or* after a number of idle iterations

## Note:

- ▶ **Admissible neighbors of  $s$** : Non-tabu search positions in  $N(s)$
- ▶ **Tabu tenure**: a fixed number of subsequent search steps for which the last search position or the solution components just added/removed from it are declared **tabu**
- ▶ **Aspiration criterion** (often used): specifies conditions under which tabu status may be overridden (e.g., if considered step leads to improvement in incumbent solution).
- ▶ Crucial for efficient implementation:
  - ▶ efficient **best improvement** local search  
↪ pruning, delta updates, (auxiliary) data structures
  - ▶ 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.

1. Trajectory Based Metaheuristics
  - Stochastic Local Search
  - Simulated Annealing
  - Iterated Local Search
  - Tabu Search
  - Variable Neighborhood Search**
  - Guided Local Search
  
2. Population Based Metaheuristics
  - Evolutionary Algorithms
  - Ant Colony Optimization

Variable Neighborhood Search is a method based on the systematic change of the neighborhood during the search.

## Central observations

- ▶ a local minimum w.r.t. one neighborhood function is not necessarily locally minimal w.r.t. another neighborhood function
- ▶ a global optimum is locally optimal w.r.t. **all** neighborhood functions

**Key principle:** change the neighborhood during the search

- ▶ Several adaptations of this central principle
  - ▶ (Basic) Variable Neighborhood Descent (VND)
  - ▶ Variable Neighborhood Search (VNS)
  - ▶ Reduced Variable Neighborhood Search (RVNS)
  - ▶ Variable Neighborhood Decomposition Search (VNDS)
  - ▶ Skewed Variable Neighborhood Search (SVNS)
  
- ▶ Notation
  - ▶  $\mathcal{N}_k$ ,  $k = 1, 2, \dots, k_m$  is a set of neighborhood functions
  - ▶  $N_k(s)$  is the set of solutions in the  $k$ -th neighborhood of  $s$



## How to generate the various neighborhood functions?

- ▶ for many problems different neighborhood functions (local searches) exist / are in use
- ▶ change parameters of existing local search algorithms
- ▶ use  $k$ -exchange neighborhoods; these can be naturally extended
- ▶ many neighborhood functions are associated with distance measures; in this case increase the distance

# Basic Variable Neighborhood Descent

## Procedure BVND

**input** :  $\mathcal{N}_k$ ,  $k = 1, 2, \dots, k_{max}$ , and an initial solution  $s$

**output**: a local optimum  $s$  for  $\mathcal{N}_k$ ,  $k = 1, 2, \dots, k_{max}$

$k \leftarrow 1$

**repeat**

$s' \leftarrow \text{FindBestNeighbor}(s, \mathcal{N}_k)$

**if**  $f(s') < f(s)$  **then**

$s \leftarrow s'$

$(k \leftarrow 1)$

**else**

$k \leftarrow k + 1$

**until**  $k = k_{max}$ ;

# Variable Neighborhood Descent

## Procedure VND

**input** :  $\mathcal{N}_k$ ,  $k = 1, 2, \dots, k_{max}$ , and an initial solution  $s$

**output**: a local optimum  $s$  for  $\mathcal{N}_k$ ,  $k = 1, 2, \dots, k_{max}$

$k \leftarrow 1$

**repeat**

$s' \leftarrow \text{IterativeImprovement}(s, \mathcal{N}_k)$

**if**  $f(s') < f(s)$  **then**

$s \leftarrow s'$

$k \leftarrow 1$

**else**

$k \leftarrow k + 1$

**until**  $k = k_{max}$ ;

- ▶ Final solution is locally optimal w.r.t. all neighborhoods
- ▶ First improvement may be applied instead of best improvement
- ▶ Typically, order neighborhoods from smallest to largest
- ▶ If iterative improvement algorithms  $I_k$ ,  $k = 1, \dots, k_{max}$  are available as black-box procedures:
  - ▶ order black-boxes
  - ▶ apply them in the given order
  - ▶ possibly iterate starting from the first one
  - ▶ order chosen by: *solution quality* and *speed*

# Basic Variable Neighborhood Search

## Procedure BVNS

**input** :  $\mathcal{N}_k$ ,  $k = 1, 2, \dots, k_{max}$ , and an initial solution  $s$

**output**: a local optimum  $s$  for  $\mathcal{N}_k$ ,  $k = 1, 2, \dots, k_{max}$

**repeat**

$k \leftarrow 1$

**repeat**

$s' \leftarrow \text{RandomPicking}(s, \mathcal{N}_k)$

$s'' \leftarrow \text{IterativeImprovement}(s', \mathcal{N}_k)$

**if**  $f(s'') < f(s)$  **then**

$s \leftarrow s''$

$k \leftarrow 1$

**else**

$k \leftarrow k + 1$

**until**  $k = k_{max}$ ;

**until** Termination Condition;

To decide:

- ▶ which neighborhoods
  - ▶ how many
  - ▶ which order
  - ▶ which change strategy
- 
- ▶ Extended version: parameters  $k_{min}$  and  $k_{step}$ ; set  $k \leftarrow k_{min}$  and increase by  $k_{step}$  if no better solution is found (achieves diversification)

1. Trajectory Based Metaheuristics
  - Stochastic Local Search
  - Simulated Annealing
  - Iterated Local Search
  - Tabu Search
  - Variable Neighborhood Search
  - Guided Local Search**
  
2. Population Based Metaheuristics
  - Evolutionary Algorithms
  - Ant Colony Optimization

# Guided Local Search

- ▶ **Key Idea:** Modify the evaluation function whenever a local optimum is encountered.
- ▶ Associate **weights** (**penalties**) with solution components; these determine impact of components on evaluation function value.
- ▶ Perform Iterative Improvement; when in local minimum, increase penalties of some solution components until improving steps become available.

## Guided Local Search (GLS):

determine *initial candidate solution*  $s$

initialize penalties

**while** *termination criterion* is not satisfied **do**

    compute **modified evaluation function**  $g'$  from  $g$   
        based on **penalties**

    perform **subsidiary local search** on  $s$   
        using **evaluation function**  $g'$

**update penalties** based on  $s$



## Guided Local Search (continued)

- ▶ **Modified evaluation function:**

$$g'(s) := f(s) + \sum_{i \in SC(s)} \text{penalty}(i),$$

where  $SC(s)$  is the set of solution components used in candidate solution  $s$ .

- ▶ **Penalty initialization:** For all  $i$ :  $\text{penalty}(i) := 0$ .
- ▶ **Penalty update** in local minimum  $s$ : Typically involves *penalty increase* of some or all solution components of  $s$ ; often also occasional *penalty decrease* or *penalty smoothing*.
- ▶ **Subsidiary local search:** Often *Iterative Improvement*.

## Potential problem:

Solution components required for (optimal) solution may also be present in many local minima.

## Possible solutions:

- A:** Occasional decreases/smoothing of penalties.
- B:** Only increase penalties of solution components that are least likely to occur in (optimal) solutions.

## Implementation of B:

Only increase penalties of solution components  $i$  with maximal utility [Voudouris and Tsang, 1995]:

$$\text{util}(s, i) := \frac{f_i(s)}{1 + \text{penalty}(i)}$$

where  $f_i(s)$  is the solution quality contribution of  $i$  in  $s$ .

## Example: Guided Local Search (GLS) for the TSP

[Voudouris and Tsang 1995; 1999]

- ▶ **Given:** TSP instance  $\pi$
- ▶ **Search space:** Hamiltonian cycles in  $\pi$  with  $n$  vertices;
- ▶ **Neighborhood:** 2-edge-exchange;
- ▶ **Solution components** edges of  $\pi$ ;  
 $f_e(G, p) := w(e)$ ;
- ▶ **Penalty initialization:** Set all edge penalties to zero.
- ▶ **Subsidiary local search:** Iterative First Improvement.
- ▶ **Penalty update:** Increment penalties of all edges with maximal utility by

$$\lambda := 0.3 \cdot \frac{w(s_{2-opt})}{n}$$

where  $s_{2-opt} = 2$ -optimal tour.

1. Trajectory Based Metaheuristics
  - Stochastic Local Search
  - Simulated Annealing
  - Iterated Local Search
  - Tabu Search
  - Variable Neighborhood Search
  - Guided Local Search
2. Population Based Metaheuristics
  - Evolutionary Algorithms
  - Ant Colony Optimization

1. Trajectory Based Metaheuristics
  - Stochastic Local Search
  - Simulated Annealing
  - Iterated Local Search
  - Tabu Search
  - Variable Neighborhood Search
  - Guided Local Search
  
2. Population Based Metaheuristics
  - Evolutionary Algorithms
  - Ant Colony Optimization

1. Trajectory Based Metaheuristics
  - Stochastic Local Search
  - Simulated Annealing
  - Iterated Local Search
  - Tabu Search
  - Variable Neighborhood Search
  - Guided Local Search
  
2. Population Based Metaheuristics
  - Evolutionary Algorithms**
  - Ant Colony Optimization

1. Trajectory Based Metaheuristics
  - Stochastic Local Search
  - Simulated Annealing
  - Iterated Local Search
  - Tabu Search
  - Variable Neighborhood Search
  - Guided Local Search
2. Population Based Metaheuristics
  - Evolutionary Algorithms
  - Ant Colony Optimization