Single Solution-based Metaheuristics

E-G. Talbi
Single solution-based metaheuristics

- “Improvement” of a solution.
- Oriented exploitation:
  - Based on the descent.
  - Exploration of the neighborhood. (intensification)

![Diagram showing neighborhood and replacement of local optima]
Taxonomy (Single solution based metaheuristics)

- Metaheuristics
  - Solution
    - Hill climbing
    - Simulated annealing
    - Tabu search
    - Variable neighborhood search
Outline

- Hill Climbing.
- Simulated Annealing.
- Tabu Search.
- Variable Neighborhood search.
Hill Climbing
Outline

- Hill Climbing:
  - Main characteristics,
  - Algorithm,
  - Trajectory guidance,
  - Efficient evaluation/application of moves.
- Simulated Annealing.
- Tabu Search.
- Variable Neighborhood search.
Hill Climbing

- Easy to implement.
- It only leads to local optima.
- The found optima depends on the initial solution.
- No mean to estimate the relative error from the global optimum.
- No mean to have an upper bound of the computation time.
Strategies for the selection of a neighbour

- **Deterministic/full**: choosing the best neighbor (i.e. that improves the most the cost function).
- **Deterministic/partial**: choosing the first processed neighbour that is better than the current solution.
- **Stochastic/full**: processing the whole neighborhood and applying a random better one.
- …
Hill Climbing algorithm (1/2)

- **Function** HILL_CLIMBING (Problem) returns a solution state.
- **Inputs**: Problem, a problem.
- **Local Variables**:
  - Current, a state,
  - Next, a state,
  - Local, a boolean.
Hill Climbing algorithm (2/2)

- \textit{Current} = \textsc{make-node}(\text{initial state}[\textit{Problem}]);
- \textit{Local} = false;
- \textbf{Do}
  - \textit{Next} = a neighbour of \textit{Current}, selected according to a guidance strategy (det/stoch, full/partial, etc);
  - \textbf{If} (\textit{Next} is better than \textit{Current}) \textbf{then}
    - \textit{Current} = \textit{Next};
    - \textit{Local} = false;
  - \textbf{Else}
    - \textit{Local} = true;
- \textbf{Until not local}
- \textbf{Return} \textit{Current};
Efficiency issues

- The evaluation function is calculated for every candidate neighbour.
- Often the cost function is the most expensive part of the algorithm.
- Therefore,
  - We need to evaluate the cost function as efficiently as possible:
    - Use Incremental (Delta) Evaluation.
Application to TSP

- Example: “Two-opt”:
  - Two points within the permutation are selected and the segment between them is inverted.
  - This operator puts two new edges in the tour.

\[
\text{Delta} = -d(2,1) - d(5,3) + d(2,5) + d(1,3)
\]
Hill Climbing Pros & Cons

- Pros:
  - Easy to implement and fast at execution.

- Cons:
  - Get stuck at local optima,
  - Possible solutions:
    - Multi-start: try several runs, starting at different initial solutions,
    - Increasing the size of the neighborhood:
      - In TSP try 3-opt rather than 2-opt.
Simulated Annealing
Outline

- Hill Climbing.
- Simulated Annealing:
  - Main features,
  - Criterion of move acceptation,
  - Cooling schedule.
- Tabu Search.
- Variable Neighborhood search.
Simulated Annealing (1/2)

- Motivated by the physical annealing process.
- Material is heated and slowly cooled into a uniform structure.
- Simulated annealing mimics this process.
- The first SA algorithm was developed in 1953 (Metropolis).
- Kirkpatrick (1982) applied SA to optimization problems:
Simulated Annealing (2/2)

- Compared to hill climbing the main difference is that SA allows downwards steps.
- Simulated annealing also differs from hill climbing in that a move is selected at random and its acceptance is conditional:
  - Yet, moves that improve the cost function are always accepted.
The decision criterion in accepting a given move

- The law of thermodynamics states that at temperature, $t$, the probability of an increase in energy of magnitude, $\delta E$, is given by:

$$P(\delta E) = \exp\left(-\frac{\delta E}{kt}\right)$$

- Where $k$ is a constant known as Boltzmann’s constant.
Accepting a move or not (1/2)

\[ P = \exp\left(-\frac{c}{t}\right) > r \]

- Where:
  - \( c \) is change in the evaluation function,
  - \( t \) the current temperature,
  - \( r \) is a random number between 0 and 1.

- Example:

<table>
<thead>
<tr>
<th>Change</th>
<th>Temp</th>
<th>( \exp(-C/T) )</th>
<th>Change</th>
<th>Temp</th>
<th>( \exp(-C/T) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>0.95</td>
<td>0.810157735</td>
<td>0.2</td>
<td>0.1</td>
<td>0.135335283</td>
</tr>
<tr>
<td>0.4</td>
<td>0.95</td>
<td>0.656355555</td>
<td>0.4</td>
<td>0.1</td>
<td>0.018315639</td>
</tr>
<tr>
<td>0.6</td>
<td>0.95</td>
<td>0.53175153</td>
<td>0.6</td>
<td>0.1</td>
<td>0.002478752</td>
</tr>
<tr>
<td>0.8</td>
<td>0.95</td>
<td>0.430802615</td>
<td>0.8</td>
<td>0.1</td>
<td>0.000335463</td>
</tr>
</tbody>
</table>
Accepting a move or not (2/2)

- The probability of accepting a worse state is a function of both the temperature of the system and the change in the cost function.
- As the temperature decreases, the probability of accepting worse moves decreases.
- If \( t=0 \), no worse moves are accepted (i.e. hill climbing).
Simulated Annealing algorithm (1/2)

- **Function** SIMUL_ANNEALING(Problem) returns a solution state
- **Inputs:**
  - *Problem*, a problem,
  - *Schedule*, a mapping from time to temperature.
- **Local Variables:**
  - *Current*, a node,
  - *Next*, a node,
  - *T*, a “temperature” controlling the probability of downward steps,
  - *num_steps*, giving the number of moves performed for a given temperature value.
Simulated Annealing algorithm (2/2)

- \( T = T_{init}; \)
- \( Current = MAKE\_NODE(INITIAL\_STATE[Problem]); \)
- Do
  - For \( k \) from 1 to num\_steps do
    - \( Next = \) a randomly selected successor of \( Current; \)
    - \( \Delta E = VALUE[Next] - VALUE[Current]; \)
    - If \( \Delta E > 0 \) then \( Current = Next; \)
    - else \( Current = Next \) only with probability \( \exp(-\Delta E/T); \)
  - Done
- \( T = Schedule[T]; \)
- Until \( T < T_{min}; \)
- Return \( Current; \)
The cooling schedule

- The cooling schedule is *hidden* in this algorithm - but it is important (more later) -

- The algorithm assumes that annealing will continue until temperature is zero - this is not necessarily the case:
  - Starting temperature,
  - Final temperature,
  - Temperature update,
  - Iterations at each temperature.
Starting temperature

- *Not* be so hot that we conduct a **random search** for a period of time.

- *Hot* enough to allow moves to *almost* neighbourhood state (else hill climbing):
  - ➔ 60% of worse moves are accepted.

- If we know the **maximum change** in the cost function we can use this to estimate it.
Choosing the final temperature

- Compromise (Stopping criteria):
  - Low temperature,
  - When the system is “frozen” at the current temperature (i.e. no better or worse moves are being accepted).
Choosing the temperature decrement (1/2)

- Theory ⇔ Number of iterations at each temperature might be exponential to the problem size.

- Compromise:
  - A large number of iterations at a few temperatures,
  - A small number of iterations at many temperatures.
Choosing the temperature decrement (2/2)

- Decrement function in general:
  
  \[ temp = temp \times a \]

- Experience has shown that a should be between 0.8 and 0.99,

- Compromise: quality / search time.
Iterations at each temperature

- A constant number of iterations at each temperature (Depend on the Neighborhood size).

- Another method, first suggested by (Lundy, 1986) is to only do one iteration at each temperature, but to decrease the temperature very slowly:
  \[ t = \frac{t}{1 + \beta t} \]
  - The formula used is where \( \beta \) is a suitably small value.

- Dynamic:
  - Small number of iterations at high temperature,
  - Large number of iterations at low temperature.
References

Tabu Search
Outline

- Hill Climbing.
- Simulated Annealing.
- Tabu Search:
  - Main features,
  - Management of tabu moves/solutions,
  - Aspiration criterion,
  - Example.
- Variable Neighborhood search.
Main characteristics (1/2)

- Proposed independently by Glover (1986) and Hansen (1986).
- It behaves like Hill Climbing algorithm.
- But it accepts non-improving solutions in order to escape from local optima (where all the neighbouring solutions are non-improving).
- Deterministic algorithm.
Main characteristics (2/2)

- After exploring the neighbouring solutions, we accept the best one even if it decreases the cost function.
  → A worse move may be accepted.

- Three goals in the use of memory:
  - preventing the search from revisiting previously visited solutions (tabu list).
  - exploring the unvisited areas of the solution space (diversification).
  - Exploiting the elite (best found) solutions found (intensification).
Features of Tabu Search (1/3)

- Memory related - recency (How recent the solution has been reached):
  - Tabu List (short term memory): to record a limited number of attributes of solutions (moves, selections, assignments, etc) to be discouraged in order to prevent revisiting a visited solution,
  - Tabu tenure (length of tabu list): number of iterations a tabu move is considered to remain tabu.
Features of Tabu Search (2/3)

- Memory related – frequency:
  - Long term memory: to record attributes of elite solutions to be used in:
    - Diversification: Discouraging attributes of elite solutions in selection functions in order to diversify the search to other areas of solution space,
    - Intensification: giving priority to attributes of a set of elite solutions (usually in weighted probability manner).
If a move is good, but it is tabu, do we still reject it?

Aspiration criteria ⇔ accepting a solution even if generated by a tabu move:

- Best found solution.
Tabu Search algorithm (1/2)

- **Function** `TABU_SEARCH(Problem)` returns a solution state.
- **Inputs**: `Problem`, a problem.
- **Local Variables**:
  - `Current`, a state,
  - `Next`, a state,
  - `BestSolutionSeen`, a state,
  - `H`, a history of visited states.
Tabu Search algorithm (2/2)

- \( Current = MAKE-NODE(INITIAL-STATE[Problem]); \)
- **While not terminate**
  - \( Next = \) a highest-valued successor of \( Current; \)
  - **If(not Move_Tabu(H,Next) or Aspiration(Next)) then**
    - \( Current = Next; \)
    - Update \( BestSolutionSeen; \)
    - \( H = \) Recency\( (H + Current); \)
  - Endif
- **End-While
- Return \( BestSolutionSeen; \)
Example: TSP using Tabu Search (1/3)

- In our example of TSP:
  - Short term memory:
    - Maintain a list of $t$ edges and prevent them from being selected for consideration of moves for a number of iterations.
    - After a number of iterations, release those edges.
    - Make use of a squared table to decide if an edge is tabu or not.

<table>
<thead>
<tr>
<th></th>
<th>$V_1$</th>
<th>$V_2$</th>
<th>$V_3$</th>
<th>$V_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_1$</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$V_2$</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$V_3$</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>$V_4$</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Example: TSP using Tabu Search (2/3)

- In our example of TSP:
  - Long term memory:
    - Maintaining a list of \( t \) edges which have been considered in the last \( k \) best (worst) solutions
    - Encouraging (or discouraging) their selections in future solutions.
    - Using their frequency of appearance in the set of elite solutions and the quality of solutions which they have appeared in our selection function.
Example: TSP using Tabu Search (3/3)

- In our example of TSP:
  - Aspiration:
    - If the next moves consider those moves in the tabu list but generate better solution than the current one,
    - Accept that solution anyway,
    - Put it into tabu list.
Tabu Search Pros & Cons

- **Pros:**
  - Generate generally good solutions for optimisation problems compared with other solution based metaheuristics.

- **Cons:**
  - Tabu list construction is problem specific,
  - Long term memory is problem specific,
  - Different parameters (size tabu list, ...).
References

- R. Qu, “*Artificial Intelligence methods: Tabu search*”. 
Variable Neighborhood-search
Variable Neighborhood-search (1/3)

- Inspired from Hill Climbing.
- But manages several neighborhoods (hence different move concepts).
Variable Neighborhood-search (2/3)

- Definitions:
  - $N_k \rightarrow$ Neighborhood in $k$ distance:
    - Distance: how many locations we change.
  - $N_k(x) \rightarrow$ Set of solutions in the $k$th neighbourhood of $x$. 

![Diagram showing neighborhood search concept]

we change $k$ move
Variable Neighborhood-search (3/3)

- **Initialisation:**
  - Select the set of neighbourhood structures \( N_k \),
  - Find an initial solution \( x \).

- **Repeat** until stopping condition is met
  - Set \( K=1 \);
  - **Repeat** until \( k=k_{\text{max}} \)
    - **Shaking:** Generate a random point \( X' \) in \( N_k(x) \);
    - **Local Search:** \( x'' \) is the obtained optimum;
    - Move or not:
      - If \( x'' \) is better than \( x \) then \( x=x'' \) and \( k=1 \);
      - Otherwise \( k=k+1 \);