Artificial Intelligence Search Algorithms

Dr Rong Qu
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
rxq@cs.nott.ac.uk

Local Search Algorithms
Optimisation Problems

- For most of real world optimisation problems
  - An exact model cannot be built easily;
  - Number of feasible solutions grow exponentially with growth in the size of the problem.

- Optimisation algorithms
  - Mathematical programming
  - Tree search
  - Heuristic algorithms
Optimisation Problems

$Ras(x) = 20 + x_1^2 + x_2^2 - 10(\cos 2\pi x_1 + \cos 2\pi x_2)$. 

$x_i \in [-5.12, 5.12]$ 

$F(\bar{x}) = -20 \cdot \exp\left(-0.2 \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n} \sum_{i=1}^{n} \cos(2\pi x_i)\right) + 20 + e$

$x_i \in [-30, 30]$
Optimisation Problems: Methods

- Meta-heuristics
  - Guide an underlying heuristic/search to escape from being trapped in a local optima and to explore better areas of the solution space
  - Examples:
    - Single solution approaches: Simulated Annealing, Tabu Search, etc;
    - Population based approaches: Genetic algorithm, Memetic algorithm, Ant Algorithms, etc;
Optimisation Problems: Methods

- Meta-heuristics
  - +: Able to cope with inaccuracies of data and model, large sizes of the problem and real-time problem solving;
  - +: mechanisms to escape from local optima
  - +: ease of implementation;
  - +: no need for exact model of the problem;
  - -: usually no guarantee of optimality.
LOCAL SEARCH
Local Search: Method

- Starts from some initial solution, moves to a better neighbour solution until it arrives at a local optimum (does not have a better neighbour)
- Examples: k-opt algorithm for TSP, etc;
- +: ease of implementation;
- +: guarantee of local optimality usually in small computational time;
- +: no need for exact model of the problem;
- -: poor quality of solution due to getting stuck in poor local optima;
Local Search: terminology

- **Local maximum solution**
- **Global maximum value**
- **Neighbourhood of solution**

The diagram illustrates a function $f(X)$ with a local maximum at point $Y$, and a global maximum value. The neighbourhood of the solution is shown around $Y$.
Local Search: terminology

- **Neighbourhood function**: using the concept of a move, which changes one or more attributes of a given solution to generate another solution.

- **Local optimum**: a solution $x$ with respect to the neighbourhood function $N$, if $f(x) < f(y)$ for every $y$ in $N(x)$. 
Local Search: elements

- **Representation** of the solution
- **Evaluation function**
- **Neighbourhood function**
  - Solutions which are close to a given solution
  - Optimisation of real-valued functions, for a current solution $x^0$, neighbourhood is defined as an interval $(x^0 - r, x^0 + r)$
- **Acceptance criterion**
  - First improvement, best improvement, best of non-improving solutions, random criteria
Local Search: Hill Climbing

1. Pick a random point in the search space
2. Consider all the neighbours of the current state
3. Choose the neighbour with the best quality and move to that state
4. Repeat 2 thru 4 until all the neighbouring states are of lower quality
5. Return the current state as the solution state
Local Search: Hill Climbing

Possible solutions
- Try several runs, starting at different positions
- Increase the size of the neighborhood (e.g. 3-opt in TSP)
How can bad local optima be avoided?
SIMULATED ANNEALING

Motivated by the physical annealing process
Material is heated and slowly cooled into a uniform structure
The SA algorithm

- The first SA algorithm was developed in 1953 (Metropolis)
- Kirkpatrick (1982)* applied SA to optimisation problems

- Compared to hill climbing
  - SA allows downwards steps
  - A SA move is selected at random and then decides whether to accept it

- Better moves are always accepted
- Worse moves may be accepted, depends on a probability

To accept or not to accept?

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

$$P(\delta E) = \exp(-\delta E / kt)$$

- $k$ is a constant known as Boltzmann’s constant
To accept or not to accept?

\[ P = \exp(-c/t) > r \]

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

* Need to use a scientific calculator to calculate \( \exp() \)
To accept or not to accept?

- The probability of accepting a worse state is a function of
  - the temperature of the system
  - 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)

<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>
The SA algorithm

\[
\text{For } t = 1 \text{ to } \infty \text{ do }
\]
\[
T = Schedule[t]
\]
\[\text{If } T = 0 \text{ then return } \text{Current}\]
\[\text{Next} = \text{a randomly selected neighbour of } \text{Current}\]
\[\Delta E = VALUE[\text{Next}] - VALUE[\text{Current}]\]
\[\text{if } \Delta E > 0 \text{ then } \text{Current} = \text{Next}\]
\[\text{else } \text{Current} = \text{Next} \text{ with probability } \exp(-\Delta E/T)\]
The SA algorithm

- To implement an SA algorithm: implement hill climbing with an accept function and modify the acceptance criteria

- 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
SA - Cooling Schedule

- Starting Temperature
- Final Temperature
- Temperature Decrement
- Iterations at each temperature
SA - Cooling Schedule

- **Starting Temperature**
  - Must be *hot* enough to allow moves to *almost* neighbourhood state (else we are in danger of implementing hill climbing)
  - Must *not* be so hot that we conduct a random search for a period of time
  - Problem is finding a suitable starting temperature
SA - Cooling Schedule

- Starting Temperature
  - If we know the maximum change in the cost function we can use this to estimate
  - Start high, reduce quickly until about 60% of worse moves are accepted. Use this as the starting temperature
  - Heat rapidly until a certain percentage are accepted the start cooling
SA - Cooling Schedule

- Final Temperature
  - Usual decrease temperature to 0
  - However, the algorithm runs for a lot longer
  - In practise, it is not necessary to decrease the temperature to 0
  - Chances of accepting a worse move are almost the same as the temperature being equal to 0
  - Therefore, the stopping criteria can either be a suitably low temperature or when the system is “frozen” at the current temperature (i.e. no better or worse moves are being accepted)
SA - Cooling Schedule

- Temperature Decrement
  - Theory: allow enough iterations at each temperature so the system stabilises at that temperature
  - Unfortunately, theory: the number of iterations at each temperature to achieve this might be exponential to the problem size
- Compromise
  - Either do a large number of iterations at a few temperatures, a small number of iterations at many temperatures or a balance between the two
SA - Cooling Schedule

- Temperature Decrement
  - Linear
    - \( temp = temp - x \)
  - Geometric
    - \( temp = temp \times a \)
  - Experience has shown that \( a \) should be between 0.8 and 0.99. Of course, the higher the value of \( a \), the longer it will take
SA - Cooling Schedule

- Iterations at each temperature
  - A constant number of iterations at each temperature
  - Another method, first suggested by (Lundy, 1986) is to only do one iteration at each temperature, but to decrease the temperature very slowly
    - \( t = t/(1 + \beta t) \)
    - where \( \beta \) is a suitably small value

<table>
<thead>
<tr>
<th>Temperature of System</th>
<th>Change in Evaluation Function</th>
<th>Temperature of System</th>
<th>Change in Evaluation Function</th>
<th>( \exp(-C/T) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>100</td>
<td>0.367879441</td>
<td>100</td>
<td>0.367879441</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>0.818730753</td>
<td>20</td>
<td>0.135335283</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>0.740818221</td>
<td>30</td>
<td>0.049787068</td>
</tr>
<tr>
<td>40</td>
<td>100</td>
<td>0.670320046</td>
<td>40</td>
<td>0.018315639</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
<td>0.548811626</td>
<td>50</td>
<td>0.006737947</td>
</tr>
<tr>
<td>60</td>
<td>100</td>
<td>0.465833044</td>
<td>60</td>
<td>0.002147227</td>
</tr>
<tr>
<td>70</td>
<td>100</td>
<td>0.392999644</td>
<td>70</td>
<td>0.000691086</td>
</tr>
<tr>
<td>80</td>
<td>100</td>
<td>0.332871084</td>
<td>80</td>
<td>0.000205762</td>
</tr>
<tr>
<td>90</td>
<td>100</td>
<td>0.272531793</td>
<td>90</td>
<td>0.000073943</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>0.22313016</td>
<td>100</td>
<td>0.000023954</td>
</tr>
<tr>
<td>110</td>
<td>100</td>
<td>0.182683524</td>
<td>110</td>
<td>0.000008676</td>
</tr>
<tr>
<td>120</td>
<td>100</td>
<td>0.148856819</td>
<td>120</td>
<td>0.000003322</td>
</tr>
<tr>
<td>130</td>
<td>100</td>
<td>0.120523686</td>
<td>130</td>
<td>0.000001134</td>
</tr>
<tr>
<td>140</td>
<td>100</td>
<td>0.098063840</td>
<td>140</td>
<td>0.000000370</td>
</tr>
<tr>
<td>150</td>
<td>100</td>
<td>0.080068023</td>
<td>150</td>
<td>0.000000129</td>
</tr>
<tr>
<td>160</td>
<td>100</td>
<td>0.065291477</td>
<td>160</td>
<td>0.000000049</td>
</tr>
<tr>
<td>170</td>
<td>100</td>
<td>0.053693984</td>
<td>170</td>
<td>0.000000018</td>
</tr>
<tr>
<td>180</td>
<td>100</td>
<td>0.045142923</td>
<td>180</td>
<td>0.000000006</td>
</tr>
<tr>
<td>190</td>
<td>100</td>
<td>0.038451380</td>
<td>190</td>
<td>0.000000002</td>
</tr>
<tr>
<td>200</td>
<td>100</td>
<td>0.033378019</td>
<td>200</td>
<td>0.000000001</td>
</tr>
</tbody>
</table>
SA - Cooling Schedule

- Iterations at each temperature
  - An alternative: dynamically change the number of iterations as the algorithm progresses
    - At lower temperatures: a large number of iterations are done so that the local optimum can be fully explored
    - At higher temperatures: the number of iterations can be less
“a meta-heuristic superimposed on another heuristic. The overall approach is to avoid entrapment in cycles by forbidding or penalizing moves which take the solution, in the next iteration, to points in the solution space previously visited (hence tabu).”

Proposed independently by Glover (1986) and Hansen (1986)
The TS algorithm

- Accepts non-improving solutions deterministically in order to escape from local optima by guiding a steepest descent local search (or steepest ascent hill climbing) algorithm

- After evaluating a number of neighbourhoods, we accept the best one, even if it is low quality on cost function.
  - Accept worse move
The TS algorithm

- Uses of past experiences (memory) to improve current decision making in two ways
  - prevent the search from revisiting previously visited solutions
  - explore the unvisited areas of the solution space

- By using memory (a “tabu list”) to prohibit certain moves
  - makes tabu search a **global** optimizer rather than a local optimizer
Tabu Search vs. Simulated Annealing

- Accept worse move
- Selection of neighbourhoods

- Use of memory
  - Is memory useful during the search?

- Intelligence needs memory!
- Information on characteristics of good solutions (or bad solutions!)
Tabu Search - uses of memory

- Tabu move – what does it mean?
  - Not allowed to re-visit exactly the same state that we’ve been before
    - Discouraging some patterns in solution: e.g. in TSP problem, tabu a state that has the towns listed in the same order that we’ve seen before.
    - If the size of problem is large, lot of time just checking if we’ve been to certain state before.
Tabu Search - uses of memory

- Tabu move – what does it mean?
  - Not allowed to return to the state that the search has just come from.
    - just one solution remembered
    - smaller data structure in tabu list
  - Tabu a small part of the state
    - In TSP problem, tabu the two towns just been considered in the last move – search is forced to consider other towns.
Tabu Search algorithm

- **Current** = initial solution
- **While not terminate**
  - **Next** = a highest-valued neighbour 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**
Elements of Tabu Search

- 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;
Elements of Tabu Search

- Memory related - recency (How recent the solution has been reached)
  - Tabu tenure
    - List of moves does not grow forever – restrict the search too much
    - Restrict the size of list
    - FIFO
    - Other ways: dynamic
Elements of Tabu Search

- 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)
Elements of Tabu Search

- If a move is good, but it’s tabu-ed, do we still reject it?
- Aspiration criteria: accepting an improving solution even if generated by a tabu move
  - Similar to SA in always accepting improving solutions, but accepting non-improving ones when there is no improving solution in the neighbourhood;
Example: TSP using Tabu Search

Find the list of towns to be visited so that the travelling salesman will have the shortest route

- Short term memory:
  - Maintain a list of towns and prevent them from being selected for consideration of moves for a number of iterations;
  - After a number of iterations, release those towns by FIFO
Example: TSP using Tabu Search

- Long term memory:
  - Maintain a list of towns which have been considered in the last $k$ best (worst) solutions
  - Encourage (or discourage) 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

- **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
  - Generated generally good solutions for optimisation problems compared with other AI methods

- Cons
  - Tabu list construction is problem specific
  - No guarantee of global optimal solutions
Other Local Search

- Variable Neighbourhood Search
- Iterative Local Search
- Guided Local Search
- GRASP (Greedy Random Adaptive Search Procedure)
- ...

LOCAL SEARCH
Problem specific decisions
Cost Function

- The evaluation function is calculated at every iteration.

- 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 Delta Evaluation.
- Use Partial Evaluation.
Cost Function

- If possible, the cost function should also be designed so that it can lead the search
  - Avoid cost functions where many states return the same value
    This can be seen as a plateau in the search space, the search has no knowledge where it should proceed
  - Bin Packing
Cost Function

- Bin Packing
  - A number of items, a number of bins
  - Objective
    - As many items as possible
    - As less bins as possible
    - Other objectives depending on the problems

- Cost function?
  - a) number of bins
  - b) number of items
  - c) both a) and b)

- How about there are weights for the items?
Cost Function

- Graph Colouring
  - A undirected graph $G = (V, E)$, $V$: vertices; $E$: edges connecting vertices
  - Objective
    - colouring the graph with the minimal number of colours so that
    - no adjacent vertices are of the same colour

- Cost function?
  - a) number of colours
  - How about different colourings (during the search) of the same number of colours?
Cost Function

- Many cost functions cater for the fact that some solutions are illegal. This is typically achieved using constraints

  - **Hard Constraints**: these constraints cannot be violated in a feasible solution
  
  - **Soft Constraints**: these constraints should, ideally, not be violated but, if they are, the solution is still feasible

- Examples: bin packing, timetabling
Cost Function

- Weightings
  - Hard constraints: a large weighting. The solutions which violate those constraints have a high cost function
  - Soft constraints: weighted depending on their importance
  - Can be dynamically changed as the algorithm progresses. This allows hard constraints to be accepted at the start of the algorithm but rejected later
Neighbourhood

- How do you move from one state to another?
- When you are in a certain state, what other states are reachable?
  - Examples: bin packing, timetabling
- Some research
  - the neighbourhood structure should be symmetric. i.e. if move from state $i$ to state $j$, then possible to move from state $j$ to state $i$
- However, a weaker condition can hold in order to ensure convergence
- Every state must be reachable from every other.
  - Important: when thinking about your problem, ensure that this condition is met
Neighbourhood

- The smaller the search space, the easier the search will be.
- If we define cost function such that infeasible solutions are accepted, the search space will be increased.
- As well as keeping the search space small, also keep the neighbourhood small.
Performance

- What is performance?
  - Quality of the solution returned
  - Time taken by the algorithm

- We already have the problem of finding suitable SA parameters (cooling schedule)
Performance

- Improving Performance - Initialisation
  - Start with a random solution and let the annealing process improve on that.
  - Might be better to start with a solution that has been heuristically built (e.g. for the TSP problem, start with a greedy search)
Performance

- Improving Performance – Hybridisation
  - Combine two search algorithms
  - the primary search mechanism: a population based search strategy
  - a local search mechanism is applied to move each individual to a local optimum
APPENDIX
SA modifications in the literature
The probability of accepting a worse move in SA is normally based on the physical analogy (based on the Boltzmann distribution).

But is there any reason why a different function will not perform better for all, or at least certain, problems?
Acceptance Probability

Why should we use a different acceptance criteria?

- The one proposed does not work. Or we suspect we might be able to produce better solutions
- The exponential calculation is computationally expensive.
- Johnson (1991) found that the acceptance calculation took about one third of the computation time

Konstanz, May 2012
Acceptance Probability

- Johnson experimented with
  \[ P(\delta) = 1 - \frac{\delta}{t} \]

- This approximates the exponential

Please read in conjunction with the simulated annealing handout

<table>
<thead>
<tr>
<th>Set these parameters</th>
<th>Classic Acceptance Criteria</th>
<th>Approximate Acceptance Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change in Evaluation Function, (c)</td>
<td>(e^{(-c/t)})</td>
<td>(1 - \frac{c}{t})</td>
</tr>
<tr>
<td>Temperature, (t)</td>
<td>100</td>
<td>0.818730753</td>
</tr>
</tbody>
</table>

Konstanz, May 2012
Acceptance Probability

- A better approach was found by building a look-up table of a set of values over the range $\delta/t$
- During the course of the algorithm $\delta/t$ was rounded to the nearest integer and this value was used to access the look-up table
- This method was found to speed up the algorithm by about a third with no significant effect on solution quality
The Cooling Schedule

- If you plot a typical cooling schedule you are likely to find that at high temperatures many solutions are accepted.
- If you start at too high a temperature a random search is emulated and until the temperature cools sufficiently any solution can be reached and could have been used as a starting position.
The Cooling Schedule

- At lower temperatures, a plot of the cooling schedule, is likely to show that very few worse moves are accepted; almost making simulated annealing emulate hill climbing.

- Taking this one stage further, we can say that simulated annealing does most of its work during the middle stages of the cooling schedule.

- Connolly (1990) suggested annealing at a constant temperature.
The Cooling Schedule

- But what temperature?
- It must be high enough to allow movement but not so low that the system is frozen
- But, the optimum temperature will vary from one type of problem to another and also from one instance of a problem to another instance of the same problem
The Cooling Schedule

- One solution to this problem is to spend some time searching for the optimum temperature and then stay at that temperature for the remainder of the algorithm.
- The final temperature is chosen as the temperature that returns the best cost function during the search phase.
Neighbourhood

- The neighbourhood of any move is normally the same throughout the algorithm but...

- The neighbourhood could be changed as the algorithm progresses
  - For example, a different neighbourhood can be used to help jumping from local optimal
Cost Function

- The cost function is calculated at every iteration of the algorithm.
- Various researchers (e.g. Burke, 1999) have shown that the cost function can be responsible for a large proportion of the execution time of the algorithm.
- Some techniques have been suggested which aim to alleviate this problem.
Cost Function

- Rana (1996) - Coors Brewery
  - GA but could be applied to SA
  - The evaluation function is approximated (one tenth of a second)
  - Potentially good solution are fully evaluated (three minutes)
Cost Function

- Ross (1994) uses delta evaluation on the timetabling problem
- Instead of evaluating every timetable as only small changes are being made between one timetable and the next, it is possible to evaluate just the changes and update the previous cost function using the result of that calculation
Burke (1999) uses a cache

- The cache stores cost functions (partial and complete) that have already been evaluated
- They can be retrieved from the cache rather than having to go through the evaluation function again