Non-deterministic Search techniques

Emma Hart
Why do local search?

- Many real problems are too hard to solve with exact (deterministic) techniques.
- Modern, non-deterministic techniques offer ways of getting good solutions fast.
- They can be generalised across many problems.
- Evolutionary algorithms aren’t the only answer!
Local Search

- Often naïve, but very good way of obtaining near-optimal solutions
- Many variants:
  - Hill-climbers
  - Simulated annealing
  - Tabu search
- Often a good “first approach” or benchmark to compare your EAs to
Features of local search algorithms

- Starting from an initial solution or solutions, move through the search-space to a “better” position
- Can be hard because many search-spaces have:
  - Several optimal points
  - Many “local” optima
Local Search Illustrated

- Global optimum value
- Local optimum value

Local optimum value

Global optimum value

Local optimum value

Local optimum solution

Global optimum solution

Local optimum solution
Ingredients of all modern search algorithms

Some concepts are common to all search algorithms:

1. A solution representation
2. An objective and cost function (evaluation function)
3. A neighbourhood (or move operator)
Solution Representation

- A method of representing the problem
- For example
  - A binary string
  - A permutation
  - A vector of real numbers
- In general, choose the most natural way of representing the problem
Solution Representation

- The representation you choose defines the search space
  - And therefore the size of the problem
  - You need to choose the right search space for the problem
    - It can reduce the size of the search space
    - It can make the difference between solving the problem and not solving it!
Using the right search space

- How can you construct 4 equilateral triangles from 6 matches?
Using the right search space

- How can you construct 4 equilateral triangles from 6 matches?

Use a 3D search space!
Examples of Representing Solutions

Binary permutation vector:

```
1 0 1 0 1 1 0 1
```

Start:

```
A → D → E
B → C → F
```

Function:

\[ f(x) = x^3 + y^{12} - z^4 \]

Vector:

```
1.2    6.7    4.5
```
Objectives and Evaluation Functions

- The objective is what you are trying to achieve:
  - In the TSP, minimise the distance travelled
- The evaluation function allows you to
  - Determine the quality of a possible solution
  - Compare the quality of a number of solutions
  - It is helpful if the evaluation function tells you how much better a solution is than another however
Evaluation functions

- For TSP, the evaluation is straightforward:
  - Calculate the distance travelled in any solution
- For many problems, calculating an exact quality of a solution is computationally expensive
  - You might consider ways in which you can just rank solutions as better or worse than each other
What is a neighbourhood?

- The set of solutions that can be reached from a given state by applying some “move”
- Small neighbourhoods are cheap to explore but you might need to explore many of them
- Large neighbourhoods are hard to explore, but solution quality might be better
- Need to find a function that strikes the right balance between complexity of search and the quality of the solution
Effect of neighbourhood size

- Small neighbourhoods can be searched quickly but high risk of getting stuck in a local optimum
- Large neighbourhoods are expensive to search but less risk of getting stuck
Neighbourhood = “flip 1 bit”
Permutation Neighbourhoods

Swap 2 positions
**k-opt neighbourhoods**

The set of solutions that are reached by interchanging \( k \) non-adjacent edges

- Accept first improvement found.
- Size of neighbourhood grows exponentially with \( k \)
Hill-Climbing Algorithms

- So named because they move uphill through the solution space towards better solutions!
A Hillclimbing Algorithm

1. Pick a solution from the search space, $x$. Evaluate it.
2. Choose a solution $y$ in the neighbourhood of $x$. Evaluate it.
3. If $y$ is better than $x$, replace $x$ with $y$, else discard it.
4. Return to step 2 until some stopping criterion applies.
A Hillclimbing Algorithm - variants

1. Pick a solution from the search space, $x$

2. **Apply move operator to give a new potential solution** $y$

3. **If** $y > x$, replace $x$ with $y$, else discard

4. Return to step 2 until some stopping criterion applies

**First Ascent:**
Accept first move that gives a better solution
A Hillclimbing Algorithm - variants

1. Pick a solution from the search space, $x$
2. **Apply move operator to give a new potential solution $y$**
3. **If $y > x$, replace $x$ with $y$, else discard**
4. Return to step 2 until some stopping criterion applies

**Best Ascent:**
Check all possible moves and choose the one which gives the best improvement
An Example

Neighbourhood defined by

‘flip one bit’

<table>
<thead>
<tr>
<th>Initial solution</th>
<th>Neighbourhood states</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 1 fitness 9</td>
<td>0 0 1 fitness 1</td>
</tr>
<tr>
<td></td>
<td>1 1 1 fitness 12</td>
</tr>
<tr>
<td></td>
<td>1 0 0 fitness 20</td>
</tr>
</tbody>
</table>

First ascent

111 fitness 12

Best ascent

100 fitness 20

Reached by flipping 1 bit
But hillclimbers can get stuck...

Maximize \( f(x) = x^3 - 60x^2 + 900x + 100 \)

Move operator: flip 1 bit, use best-ascent hillclimber

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
<th>Iteration 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>10011 2188</td>
<td>1100 626</td>
<td>00000 1</td>
</tr>
<tr>
<td>11011 244</td>
<td>10101 1702</td>
<td>11000 865</td>
</tr>
<tr>
<td>10111 1128</td>
<td>10011 2300</td>
<td>10100 2001</td>
</tr>
<tr>
<td><strong>10001 2874</strong></td>
<td><strong>10000 3137</strong></td>
<td><strong>10010 2593</strong></td>
</tr>
<tr>
<td>10010 2593</td>
<td></td>
<td>10001 2874</td>
</tr>
</tbody>
</table>

(answer is 01010 = 4100)
What if it gets stuck?

- On functions with many peaks (multimodal functions), the algorithm is likely to stop on the first peak it finds.
- Easy solution is to repeat several hill climbs, each time starting from a different randomly chosen points.
- This method is sometimes known as iterated hill climbing.
Problems with Hill-Climbing Illustrated

http://www.ndsu.nodak.edu/instruct/juell/vp/cs724s00/hill_climbing/
Local Search Features:

- If the evaluation function has no local optima, then local search will find the solution.
- If the evaluation function has local optima, local search will find a locally optimum solution.
- (If we are lucky, this might be the global optimum.)
Is Hill-Climbing a Good Idea?

- It works well if there is not too many local optima in the search space.
- But if the fitness function is very 'noisy' with many small peaks, stochastic hill climbing is usually not a good method to use.
- However, it is very easy to implement and to give fairly good solutions very quickly.
Simulated Annealing (SA)

- SA is another local search algorithm which has proved very effective in practice.
- It works by searching neighbourhoods via a move operator.
- BUT - it reduces the chance of getting stuck in a local optimum by allowing moves to inferior solutions.
Simulated Annealing -
Background

- Idea from a physical process of heating and then slowly cooling a substance to obtain a strong crystalline structure - structure depends on the rate of cooling
  - Large crystals are grown from slow cooling, fast cooling results in imperfections
- SA algorithm simulates the energy changes in a system subject to a cooling process until it converges to a steady ‘frozen’ state.
A Generic SA Algorithm

1. Pick a solution and evaluate it. Call this the current solution
2. Apply a move operator to get a new solution and evaluate it.
3. If the new solution is better than the current solution keep it
4. If it is worse, accept it with some small probability that decreases over time
5. Decrease the probability
6. Return to step 2 until some stopping criterion applies
Deciding to accept solutions:

• The probability of accepting a worse move is determined by the system temperature.
• At high temperatures, there is a high probability of accepting inferior moves.
• At low temperatures, there is a low probability of accepting inferior moves.
• Accepting inferior moves allows the system to escape from local optima.
The temperature variable

- The temperature decreases as the algorithm runs:
  - At the beginning, high $T$ means lots of inferior moves get accepted
  - Near end, low $T$ means hardly any bad moves get accepted
- If the temperature decreases logarithmically, guarantee optimum
Minimising a Function

Hillclimber would get stuck here

Starting Point

SA algorithm allows uphill moves from here so algorithm might not get stuck

True Minima
How is this implemented?

- Assume we are trying to minimise a function:
- Moving from solution $x$ to solution $x'$ results in a change in fitness $\Delta c = c_{x'} - c_x$
- Accept move if

$$\exp(-\Delta c / T) > R$$

- $R$ is a uniform random number between 0 and 1
Look at that temp. function:
SA in practice-an example

- Maximise $f(x) = x^3 - 60x^2 + 900x + 100$
- Encode $x$ as 5 bit string,
- Neighbourhood - flip 1 bit at random
- Maximum is 01010 ($x=10, f(x)=4100$)
- Start from 10011 ($x=19, f(x)=2399$), with initial temperature=100
Attempt 1: $T=500$ (start 10011, $f=2399$)

<table>
<thead>
<tr>
<th>T</th>
<th>bit</th>
<th>string</th>
<th>F</th>
<th>$\Delta f$</th>
<th>move?</th>
<th>new string</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>1</td>
<td>00011</td>
<td>2287</td>
<td>112</td>
<td>Y (0.59)</td>
<td>00011</td>
</tr>
</tbody>
</table>
## Attempt 1: $T=500$ (start 10011)

<table>
<thead>
<tr>
<th>$T$</th>
<th>bit</th>
<th>string</th>
<th>$F$</th>
<th>$\Delta f$</th>
<th>move?</th>
<th>new string</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>1</td>
<td>00011</td>
<td>2287</td>
<td>112</td>
<td>Y (0.59)</td>
<td>00011</td>
</tr>
<tr>
<td>450</td>
<td>3</td>
<td>00111</td>
<td>3803</td>
<td>&lt;0</td>
<td>Y</td>
<td>00111</td>
</tr>
</tbody>
</table>
### Attempt 1: $T=500$ (start 10011)

<table>
<thead>
<tr>
<th>$T$</th>
<th>bit</th>
<th>string</th>
<th>$F$</th>
<th>$\Delta f$</th>
<th>move?</th>
<th>new string</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>1</td>
<td>00011</td>
<td>2287</td>
<td>112</td>
<td>Y (0.59)</td>
<td>00011</td>
</tr>
<tr>
<td>450</td>
<td>3</td>
<td>00111</td>
<td>3803</td>
<td>&lt;0</td>
<td>Y</td>
<td>00111</td>
</tr>
<tr>
<td>405</td>
<td>5</td>
<td>00110</td>
<td>3556</td>
<td>247</td>
<td>Y (0.54)</td>
<td>00110</td>
</tr>
</tbody>
</table>
**Attempt 1: T=500 (start 10011)**

<table>
<thead>
<tr>
<th>T</th>
<th>bit</th>
<th>string</th>
<th>F</th>
<th>Δf</th>
<th>move?</th>
<th>new string</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>1</td>
<td>00011</td>
<td>2287</td>
<td>112</td>
<td>Y(0.59)</td>
<td>00011</td>
</tr>
<tr>
<td>450</td>
<td>3</td>
<td>00111</td>
<td>3803</td>
<td>&lt;0</td>
<td>Y</td>
<td>00111</td>
</tr>
<tr>
<td>405</td>
<td>5</td>
<td>00110</td>
<td>3556</td>
<td>247</td>
<td>Y(0.54)</td>
<td>00110</td>
</tr>
<tr>
<td>364.5</td>
<td>2</td>
<td>01110</td>
<td>3684</td>
<td>&lt;0</td>
<td>Y</td>
<td>01110</td>
</tr>
</tbody>
</table>
### Attempt 1: $T=500$ (start $10011$)

<table>
<thead>
<tr>
<th>T</th>
<th>bit</th>
<th>string</th>
<th>F</th>
<th>$\Delta f$</th>
<th>move?</th>
<th>new string</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>1</td>
<td>00011</td>
<td>2287</td>
<td>112</td>
<td>Y (0.59)</td>
<td>00011</td>
</tr>
<tr>
<td>450</td>
<td>3</td>
<td>00111</td>
<td>3803</td>
<td>&lt;0</td>
<td>Y</td>
<td>00111</td>
</tr>
<tr>
<td>405</td>
<td>5</td>
<td>00110</td>
<td>3556</td>
<td>247</td>
<td>Y (0.54)</td>
<td>00110</td>
</tr>
<tr>
<td>364.5</td>
<td>2</td>
<td>01110</td>
<td>3684</td>
<td>&lt;0</td>
<td>Y</td>
<td>01110</td>
</tr>
<tr>
<td>328.0</td>
<td>4</td>
<td>01100</td>
<td>3998</td>
<td>&lt;0</td>
<td>Y</td>
<td>01100</td>
</tr>
</tbody>
</table>
### Attempt 1: T=500 (start 10011)

<table>
<thead>
<tr>
<th>T</th>
<th>bit</th>
<th>string</th>
<th>F</th>
<th>$\Delta f$</th>
<th>move?</th>
<th>new string</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>1</td>
<td>00011</td>
<td>2287</td>
<td>112</td>
<td>Y (0.59)</td>
<td>00011</td>
</tr>
<tr>
<td>450</td>
<td>3</td>
<td>00111</td>
<td>3803</td>
<td>&lt;0</td>
<td>Y</td>
<td>00111</td>
</tr>
<tr>
<td>405</td>
<td>5</td>
<td>00110</td>
<td>3556</td>
<td>247</td>
<td>Y (0.54)</td>
<td>00110</td>
</tr>
<tr>
<td>364.5</td>
<td>2</td>
<td>01110</td>
<td>3684</td>
<td>&lt;0</td>
<td>Y</td>
<td>01110</td>
</tr>
<tr>
<td>328.0</td>
<td>4</td>
<td>01100</td>
<td>3998</td>
<td>&lt;0</td>
<td>Y</td>
<td>01100</td>
</tr>
<tr>
<td>295.2</td>
<td>3</td>
<td>01000</td>
<td>3972</td>
<td>16</td>
<td>Y (0.94)</td>
<td>01000</td>
</tr>
</tbody>
</table>
## Attempt 1: T=500 (start 10011)

<table>
<thead>
<tr>
<th>T</th>
<th>bit</th>
<th>string</th>
<th>F</th>
<th>$\Delta f$</th>
<th>move?</th>
<th>new string</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>1</td>
<td>00011</td>
<td>2287</td>
<td>112</td>
<td>Y (0.59)</td>
<td>00011</td>
</tr>
<tr>
<td>450</td>
<td>3</td>
<td>00111</td>
<td>3803</td>
<td>&lt;0</td>
<td>Y</td>
<td>00111</td>
</tr>
<tr>
<td>405</td>
<td>5</td>
<td>00110</td>
<td>3556</td>
<td>247</td>
<td>Y (0.54)</td>
<td>00110</td>
</tr>
<tr>
<td>364.5</td>
<td>2</td>
<td>01110</td>
<td>3684</td>
<td>&lt;0</td>
<td>Y</td>
<td>01110</td>
</tr>
<tr>
<td>328.0</td>
<td>4</td>
<td>01100</td>
<td>3998</td>
<td>&lt;0</td>
<td>Y</td>
<td>01100</td>
</tr>
<tr>
<td>295.2</td>
<td>3</td>
<td>01000</td>
<td>3972</td>
<td>16</td>
<td>Y (0.94)</td>
<td>01000</td>
</tr>
<tr>
<td>265.7</td>
<td>4</td>
<td>01010</td>
<td>4100</td>
<td>&lt;0</td>
<td>Y</td>
<td>01010**</td>
</tr>
</tbody>
</table>
## Attempt 1: T=500 (start 10011)

<table>
<thead>
<tr>
<th>T</th>
<th>bit</th>
<th>string</th>
<th>F</th>
<th>$\Delta f$</th>
<th>move?</th>
<th>new string</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>1</td>
<td>00011</td>
<td>2287</td>
<td>112</td>
<td>Y (0.59)</td>
<td>00011</td>
</tr>
<tr>
<td>450</td>
<td>3</td>
<td>00111</td>
<td>3803</td>
<td>&lt;0</td>
<td>Y</td>
<td>00111</td>
</tr>
<tr>
<td>405</td>
<td>5</td>
<td>001100</td>
<td>3556</td>
<td>247</td>
<td>Y (0.54)</td>
<td>00110</td>
</tr>
<tr>
<td>364.5</td>
<td>2</td>
<td>01110</td>
<td>3684</td>
<td>&lt;0</td>
<td>Y</td>
<td>01110</td>
</tr>
<tr>
<td>328.0</td>
<td>4</td>
<td>01100</td>
<td>3998</td>
<td>&lt;0</td>
<td>Y</td>
<td>01100</td>
</tr>
<tr>
<td>295.2</td>
<td>3</td>
<td>01000</td>
<td>3972</td>
<td>16</td>
<td>Y (0.94)</td>
<td>01000</td>
</tr>
<tr>
<td><strong>265.7</strong></td>
<td>4</td>
<td><strong>01010</strong></td>
<td><strong>4100</strong></td>
<td>&lt;0</td>
<td>Y</td>
<td><strong>01010</strong>**</td>
</tr>
<tr>
<td>239.1</td>
<td>5</td>
<td>01011</td>
<td>4071</td>
<td>29</td>
<td>Y (0.89)</td>
<td>01011</td>
</tr>
</tbody>
</table>
**Attempt 2: T=100 (start 10011)**

<table>
<thead>
<tr>
<th>T</th>
<th>bit</th>
<th>string</th>
<th>F</th>
<th>$\Delta f$</th>
<th>move?</th>
<th>new string</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>00011</td>
<td>2287</td>
<td>112</td>
<td>N(0.32)</td>
<td>10011</td>
</tr>
<tr>
<td>90</td>
<td>3</td>
<td>10111</td>
<td>1227</td>
<td>1172</td>
<td>N($\sim10^{-6}$)</td>
<td>10011</td>
</tr>
<tr>
<td>81</td>
<td>5</td>
<td>10010</td>
<td>2692</td>
<td>$&lt;$0</td>
<td>Y</td>
<td>10010</td>
</tr>
<tr>
<td>72.9</td>
<td>2</td>
<td>11010</td>
<td>516</td>
<td>2176</td>
<td>N($\sim10^{-13}$)</td>
<td>10010</td>
</tr>
<tr>
<td>65.6</td>
<td>4</td>
<td>10000</td>
<td>3236</td>
<td>$&lt;$0</td>
<td>Y</td>
<td>10000</td>
</tr>
<tr>
<td>59.0</td>
<td>3</td>
<td>10100</td>
<td>2100</td>
<td>1136</td>
<td>Y</td>
<td>10000</td>
</tr>
</tbody>
</table>

*Temperature not high enough – algorithm gets stuck*
Factors to consider

- An initial temperature at which the system starts
- A method of decreasing the temperature
- A method of determining the probability of accepting a move
- A final system temperature
Factors to consider

- Choosing the initial temperature
  - must be high; acceptance rate of 40%-60% is good

- Choosing the final temperature
  - theory suggests 0, in practice when probability of accepting an uphill move is negligible

- Setting the cooling schedule
  - probably most important factor!
Setting the cooling schedule

- The temperature is normally reduced by a small amount after every move.
- Options are:
  - Geometrically: \( T \leftarrow \alpha T \)
  - (due to Lundy & Mees) \( T \leftarrow \frac{T}{1 + \beta T} \)
Comparison of SA with GA

- Simulated Annealing can be a serious competitor to Genetic Algorithms.
- GAs have some features which should make them more efficient.
  - they use a population based selection whereas SA only deals with one individual at each iteration.
  - they are expected to cover a much larger landscape of the search space at each iteration
  - a GA can be parallelised,
  - recombination operators are able to mix good characteristics from different solutions (Exploitation).
Advantages of SA/GA

- On the other hand
  - SA only deals with one individual at each iteration.
  - SA iterations are much more simple, and so, often much faster.
  - SA is still very simple to implement
  - Shown to give good results over a large spectrum of difficult problems, like the optimal layout of printed circuit board, or the famous travelling salesman problem.
Tabu Search

- Really an extension of local search
- It’s novel contribution is to add a **memory**
- The memory:
  - Maintains a list of recent decisions made by the algorithm
  - Is used to guide the search away from previously visited areas and so diversify the search and avoid cycling
Tabu Search

- The set of moves which are forbidden is called the ‘tabu list’
  - Comes from a Tongan word!
- The memory is of fixed size and continuously updated so moves become allowed later
- Inferior solutions can be selected if all the better ones are tabu
Tabu Search

- Specify a neighbourhood search algorithm by
  - an initial state
  - a search neighbourhood
  - a move evaluator
  - tabu criteria
  - aspiration criteria
  - one or more tabu lists
Tabu-search Meta-heuristic

- Construct a candidate list from the search neighbourhood
- Pick a move that doesn’t violate tabu criteria
  - but allow move if it meets aspiration criteria
- Update tabu-list if necessary
- Replace current solution with new solution if it is better
- Repeat...
Tabu Criteria
(Short Term Memory)

- The tabu criteria prevent certain moves:
- Based on Recency/Frequency:
  - don’t allow reversal of moves recently made
  - don’t allow move if solution quality is the same as on a previous iteration
  - same solution has been visited recently
  - Don’t allow moves that have been made often
Aspiration Criteria
(Long Term Memory)

- The aspiration criteria allow some moves that the tabu criteria prevent!
- Based on quality:
  - best in neighbourhood
  - similar to existing solution (intensification)
  - dissimilar to existing solution (diversification)
An Example - Finding a Subtree in a Weighted Graph

Total weight: $5 + 22 + 14 + 30 = 71$
An Example - Finding a Subtree in a Weighted Graph

Total weight: 5+22+14+30 = 71
Total weight: 26+12+2+17=57
An Example - Finding a Subtree in a Weighted Graph

Total weight: $5+22+14+30 = 71$
Total weight: $26+12+2+17=57$
Total weight: $2+21+31+7 = 61$
A Tabu search algorithm

- **initial state:**
  - random subtree of size 4

- **a search neighbourhood:**
  - subtrees generated by changing 1 edge

- **a move evaluator:**
  - total weight of edges
A Tabu search algorithm

- **tabu criteria:**
  - don’t allow move that has been made in the last 4 iterations

- **aspiration criteria:**
  - allow move if it improves the solution by at least 2

- **tabu list:**
  - list of 4 previous moves
Parameter Setting for Tabu Search Algorithms

- Lots of critical choices involved!
  - The tabu criteria
  - size of the tabu lists
  - the underlying search method
  - the aspiration criteria

Successful applications include network design and warehouse location problems
Many other modern heuristic search algorithms

- Ant Colony Optimisation algorithms
- Artificial Immune Systems
- GRASP (Greedy Randomised Adaptive Search Procedure)
- Memetic Algorithms
Summary

- Local search can be a cheap, easy way to perform search.
- For all these methods, we don’t need to know much about the problem we are trying to solve - we just need a ‘black box’ to evaluate the solution.
- But - in many cases, performance can be enhanced by building in problem-specific knowledge.
Hill-climbing is still a good first choice
(very simple to implement and run)
SA avoids problems with local optima
Tabu’s main advantage wrt SA is in the intelligent use of past history to influence future search
Remember, SA chooses points stochastically