

Lesson 25: Settle For Good Solutions



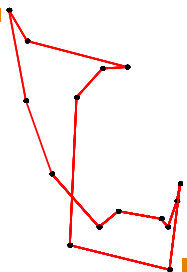
neighbourhood search, heuristics, simulated annealing, GA

Heuristic Algorithms

- Given that we know of no efficient algorithms for finding the optimal solution to NP-hard problems we must content ourselves with either
 - ★ Spending a very long time (e.g. using branch and bound)
 - ★ Accepting good solutions which aren't necessarily optimal
- Algorithms for finding good solutions are often called **approximation algorithms** or **heuristic algorithms**

Constructive Algorithms

- Constructive algorithms build-up a solution
- They usually rely on a greedy heuristic
- They are very fast
- Once you have got a solution that's it
- They can give reasonable solutions quickly, but they are not usually very good

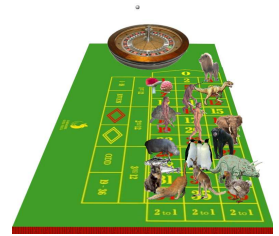


Iterative Improvement at its Best

- There are times when a neighbourhood algorithm will find the optimal solution
- The classic example of this is in **linear programming** where the simplex method leads to the optimal solution
- Other examples include
 - ★ Maximum Flow
 - ★ Maximum Matching in Bipartite Graphs
- Unfortunately, this doesn't always work

Outline

1. **Heuristic Search**
 - Constructive algorithms
 - Neighbourhood search
2. Simulated Annealing
3. Evolutionary Algorithms



Heuristics

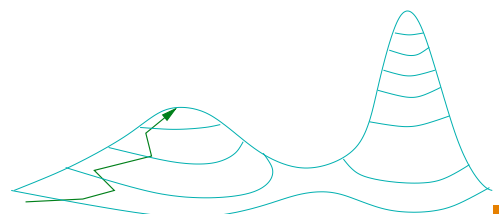
- The idea behind heuristic algorithms is to use a rough guide or **heuristic** pointing you in reasonable direction■
- If this heuristic is good you should find good solutions much faster than exhaustive search■
- Two commonly used heuristics are
 - ★ A greedy heuristic (take the best move)■
 - ★ Believe that good solutions are 'close' to each other■

Neighbourhood search

- An alternative to constructive algorithms are search algorithms relying on good solutions being close to each other
- In neighbourhood search we
 1. Start from some solution
 2. Examine the neighbouring solutions
 3. Move to a neighbour if it is better or, at least, not worse
 4. Repeat 2 until some stopping criteria
- If we are maximising this is often called a **hill-climber**
- If we are minimising it is often called **descent**

Local Optima

- Neighbourhood search is usually much slower than a constructive algorithm but tends to find better quality solutions
- However, it will often get stuck



- One simple fix is to restart from many different starting positions
- Or perturb the current solution and restart
- These give improvements over doing nothing, but aren't necessarily great strategies
- You can also increase the size of the neighbour to decrease the chance of getting stuck (e.g. in TSP swap more cities)

Simulated Annealing

- Simulated annealing is an example of a stochastic hill-climber
- Sometimes you go in the wrong direction (down-hill)
- Historically it is an idea from physics—where you tend to minimise energy
- Idea is to obtain a (low energy) crystalline material you very slowly let the material cool from a liquid state (opposite of quenching)

Simulated Annealing

- Algorithm to minimise energy $E(\mathbf{X})$ where $\mathbf{X} = (X_1, X_2, \dots, X_N)$
- Starting from a random configuration \mathbf{X}
- Choose a neighbour \mathbf{X}'
- If the neighbour is better (lower energy) move to it
- Otherwise move to the neighbour with some probability
- The parameter β controls the probability of moving to a neighbour
- We increase β to reduce the probability of going uphill over time

Convergence Theorem

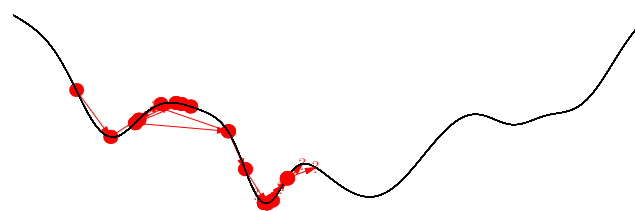
- There is a theorem that says if you choose a slow enough cooling schedule you will end up in the global minimum eventually
- Unfortunately eventually is a very long time
- It is quicker to search all possible states
- Still people get very excited about convergence proofs

1. Heuristic Search
 - Constructive algorithms
 - Neighbourhood search
2. **Simulated Annealing**
3. Evolutionary Algorithms



Stochastic Descent

- It is easier to fall down hill than to go back up



Cooling Schedule

- The parameter β is known as the inverse temperature because of an analogy with physics
- Over time we have to increase β (decrease the temperature) so that the system will remain in the low energy state
- The way you reduce the temperature (increase β) is known as the cooling schedule
- Choosing a good cooling schedule can be critical
- Choosing a good cooling schedule is something of a black art

Outline

1. Heuristic Search
 - Constructive algorithms
 - Neighbourhood search
2. Simulated Annealing
3. **Evolutionary Algorithms**



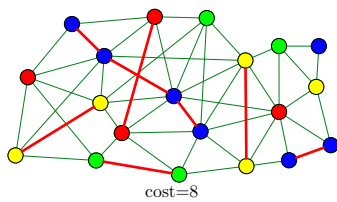
- Genetic algorithms are methods to evolve a population of potential candidates to find a good solution to an optimisation problem
- There are a whole set of related methods that go by the name of evolutionary algorithms, GAs are a subspecies of EAs
- They can be viewed as an engineering approach to solving hard problems
- I'm going to present my, highly prejudiced view of what's important in making a GA work

AICE1005

Algorithms and Analysis

17

E.g. Graph Colouring



- Given a graph $G = (\mathcal{V}, \mathcal{E})$
- Assign colours, $c(v)$, to the vertices of the graph $v \in \mathcal{V}$
- Minimise the number of edges $e = (v, v') \in \mathcal{E}$ with the same coloured vertices $c(v) = c(v')$ (colour conflicts)

AICE1005

Algorithms and Analysis

19

Selection

- Select a new population of P members preferentially choosing the fitter members
- Let w_α be a measure of the fitness of member α
- E.g. choose members α with a probability

$$p_\alpha = \frac{w_\alpha}{\sum_{\alpha'=1}^P w_{\alpha'}}$$

- Many different ways of doing this (some better than others)

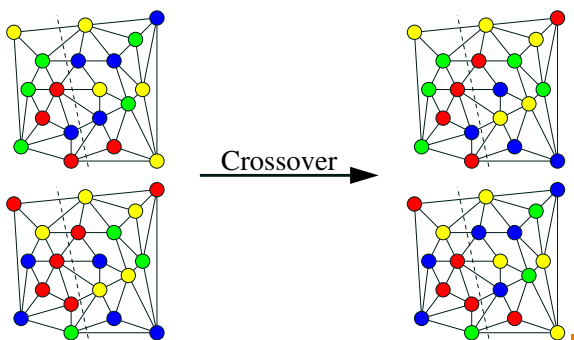
AICE1005

Algorithms and Analysis

21

Crossover

Take two solutions and combine them to form a new solution E.g.



AICE1005

Algorithms and Analysis

23

1. Initialise population

2. for $t = 1$ to T

- Evaluate fitness
- Select a new population based on fitness
- Mutate members of the population
- Crossover members of the population

3. Return best member of the population

AICE1005

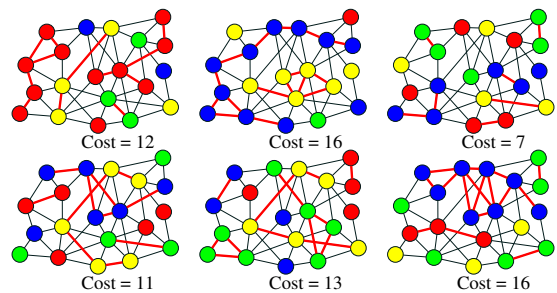
Algorithms and Analysis

18

Initialise Population

Generate random colourings E.g.

Generation 0: evaluate fitness



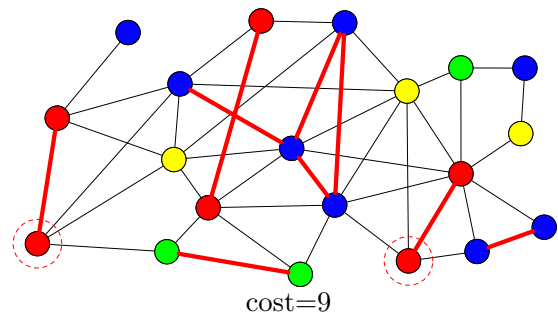
AICE1005

Algorithms and Analysis

20

Mutation

Change the colour of one or more of the vertices E.g.



AICE1005

Algorithms and Analysis

22

Crossover Operators

- Single-point crossover

★ Take two strings and cut them at some random site

$$\begin{pmatrix} GRBGBR & | & BGGBGBG \\ RRBRGB & | & RGRBBGB \end{pmatrix} \rightarrow \begin{pmatrix} GRBGBR & | & RGRBBGB \\ RRBRGB & | & BGGBGBG \end{pmatrix}$$

- Multi-point crossover

★ Take two strings and cut them at several sites

$$\begin{pmatrix} GRBGBR & | & BGGB & | & GBG \\ RRBRGB & | & RGRB & | & BGB \end{pmatrix} \rightarrow \begin{pmatrix} GRBGBR & | & RGRB & | & GBG \\ RRBRGB & | & BGGB & | & BGB \end{pmatrix}$$

- Uniform Crossover

★ Take two strings and create children by a random shuffle

$$\begin{pmatrix} GRBGBR & | & BGGBGBG \\ RRBRGB & | & RGRBBGB \end{pmatrix} \rightarrow \begin{pmatrix} GRBGBR & | & RGRBBGB \\ RRBRGB & | & BGGBGBG \end{pmatrix}$$

- Any of these crossover can be biased towards one parent

AICE1005

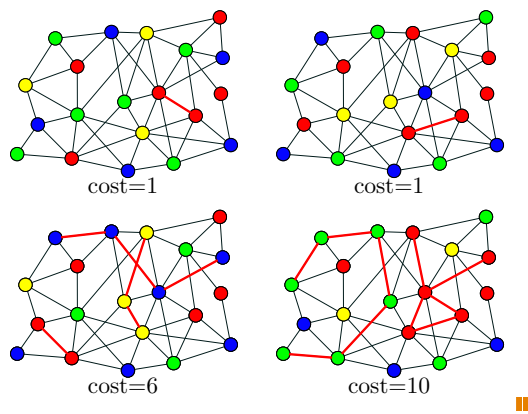
Algorithms and Analysis

24

Algorithms and Analysis

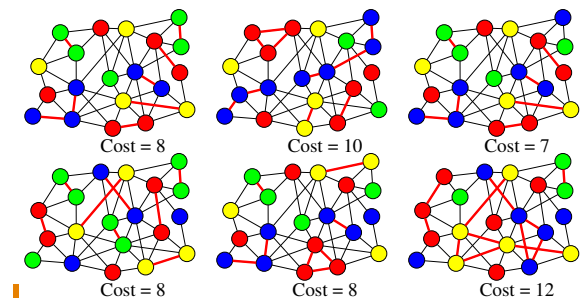
24

Cost of Crossover



GA

Final Population



Bit Simulated Crossover

- Choose each variable independently with the probability proportional to the frequency of the allele in the population

$$\begin{pmatrix} \dots & B & \dots \\ \dots & R & \dots \\ \dots & B & \dots \\ \dots & B & \dots \end{pmatrix} \quad \begin{pmatrix} \dots & R & \dots \\ \dots & B & \dots \\ \dots & G & \dots \\ \dots & B & \dots \end{pmatrix} \quad \begin{pmatrix} \dots & G & \dots \\ \dots & G & \dots \\ \dots & B & \dots \\ \dots & B & \dots \end{pmatrix}$$

$$p_i(B) = 0.5, \quad p_i(G) = 0.3, \quad p_i(R) = 0.2$$

- New algorithms built on this idea, "Estimation of Distribution Algorithms" EDAs

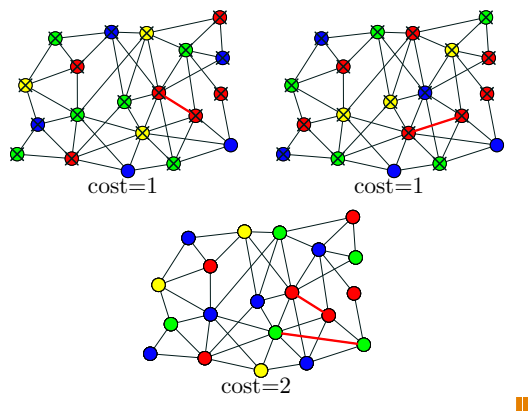
Galinier and Hao's Crossover Operator

- Choose two parents
- Sort nodes into colour-groups

	Parent 1	Parent 2
B	{1,3,4,8, ... }	{3,5,7,10, ... }
G	{2,6,7,10, ... }	{1,11,12,13, ... }
R	{5,9,11,12, ... }	{2,3,6,8, ... }
:	:	:

- Choose largest colour-group in parent 1
- Eliminate all nodes from that colour-group in parent 2
- Choose largest colour-group in parent 2
- etc.

Cost of Crossover



Other Heuristics

- There are many extensions to neighbourhood search, simulated annealing and genetic algorithms
- There are other techniques such as Tabu search
 - Construct a list of place you cannot go to (usually the last few configurations)
 - Make the best move you are allowed to make
 - Rather a large number of *ad hoc* rules to make it work
 - Often very fast but runs out of steam
- Many other EAs including particle swarm optimisations (PSO), ant colony optimisation (ACO), evolutionary strategies, ...

Which Heuristic is Best?

- The best heuristic depends on the application
- Descent is very fast, but only finds local optima—good starting place
- Tabu search is often very fast, but sometimes fails to find really good solutions
- Simulated annealing and Genetic Algorithms are slow, but often find good solutions
- The best algorithms tend to be special purpose algorithms designed for the problem

Lessons

- For many problems the best strategy is to find a good solution, but not the best
- Iterative search usually give good quality solutions
- There are many variants of heuristic search
- Heuristic search algorithms aren't fast (don't use these techniques in an interactive program if you want to keep customers)
- For large combinatorial optimisation problems this is often the only choice