

# Algorithms & Complexity

## Heuristics

Nicolas Stroppa - [nstroppa@computing.dcu.ie](mailto:nstroppa@computing.dcu.ie)

CA313@Dublin City University. 2006-2007.

November 21, 2006







# Some *NP*-complete problems

## Games

- ▶ Chess
- ▶ Sudoku
- ▶ Minesweeper

## Scheduling

- ▶ Find the best matching between rooms, teachers, and students.
- ▶ Find the best way to schedule processes in a computer.

## Positioning

- ▶ Positioning Antenna
- ▶ Designing the lines of a metro

# Heuristics

⇒ **Complex problems need approximate solutions**

## Definition (Heuristic)

- ▶ A *heuristic* is an approach for directing one's attention in learning, discovery, or problem-solving.
- ▶ A *heuristic* is usually based on a general principle that prefers some solutions to others.
- ▶ A *heuristic* usually leads to an approximate (but often good) solution, in tractable time.

## Note

Comes from the Greek "heurisko" (*I find*). ("eureka" ⇒ *I have found it!*)

## Heuristics - Driving in Paris

- ▶ You are visiting your friend in Paris for a week. You rent a car at the airport and you have to find the shortest path to go to his place.
- ▶ This is an optimization problem: among the set of all possible paths, you want to find the shortest one. Depending on the configuration of the streets, this problem can be more or less difficult.<sup>1</sup>
- ▶ In Paris, it is more complex. . .
- ▶ A heuristic in this case is to use the “main” avenues: this is not an optimal method, but it will always give you a decent solution.

---

<sup>1</sup>In Manhattan, this problem is very easy, all the paths have the same length.

# Heuristics - Traveling in Europe

- ▶ You want to visit the capital cities of Europe.
- ▶ You want to minimize the total cost of the trip.
- ▶  $\Rightarrow$  This problem is actually the same as the Traveling Salesman Problem! (distances are replaced with costs (flights,train))
- ▶ Heuristics: Go from one city  $A$  to the less expensive one from  $A$ .

# Heuristics - Chess

- ▶ In chess, you often swap pieces with the other player.
- ▶ A heuristic consists in assigning a number to each piece, and to try, during a swap, not to lose a piece with less value than the one the other player is losing.

# Some known Heuristics

- ▶ Genetic Algorithms
- ▶ Simulated Annealing
- ▶ Tabu Search

## Note

These methods are sometimes called *meta-heuristics* because they are very general, and (in principle) they can be applied to any problem.

# Genetic algorithms

- ▶ *Genetic algorithms* are *biologically inspired* algorithms, and are a particular class of *evolutionary algorithm*.
- ▶ They use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination).
- ▶ Intuitively, they rely on the principles of natural selection to solve complex optimisation problems.

## Genetic algorithms - Main ideas

- ▶ Each element of the search space is associated to an individual.  $\Rightarrow$  each individual represents a candidate for the solution of our problem. The *fitness* of an individual is the “quality” of the solution (for example, in the TSP, it would be the total cost of a path).
- ▶ The evolution starts from a population of randomly generated individuals and happens in generations.
- ▶ In each generation, the fitness of every individual in the population is evaluated, multiple individuals (usually the best ones) are selected from the current population, and modified (recombined and possibly mutated) to form a new population.
- ▶ The new population is then used in the next iteration of the algorithm.

# Genetic algorithms - Main ideas

## Selection

Given a population  $P = \{i_1, \dots, i_n\}$ , choose the  $k$  best individuals  $SELECT(P, k)$  of  $P$ , according to the fitness function.

## Mutation

Given an individual  $i$ , mutate  $i$  to get an individual  $m(i)$  that differs slightly from  $i$ .

## Cross-over/Recombination

Given two individuals  $i$  and  $j$ , combine them to get a new individual  $c(i, j)$ .

## Genetic algorithms - Formalization

Each element  $e$  of the search space  $S$  is associated to an individual, a genetic representation of  $e$ . The fitness function is noted  $f$ .

1. *Init.* Generate a random initial population  
 $P = \{i_1, \dots, i_n\} \subset V$  with  $n$  individuals.  $i = 0, P_i \leftarrow P$ .
2. *Selection.*  $P_{i+1} \leftarrow SELECT(P_i, k)$
3. If a stopping criterion is reached, go to (6); otherwise, go to step (4).
4. *Mutation and Cross-Over.* From  $P_{i+1}$ , create new individuals  $P_{new}$  through crossover and mutation.  $P_{i+1} \leftarrow P_{i+1} \cup P_{new}$
5.  $i \leftarrow i + 1$ . Go to step (2)
6. Propose the best individual in  $P_{i+1}$

## Genetic algorithms - Properties

- ▶ The goal of the mutation step is to be able to explore new areas and thus to avoid being stucked in local minima.
- ▶ The goal of the combination step is to combine the strength of two good individuals in order to create an individual (possibly) better than its parents.
- ▶ You can stop if you consider that your population is too homogeneous or if you have reached a given number of steps, or if you have reached a time limit. At each step, the best individual can be proposed, so it is an *anytime algorithm*.
- ▶ It is a stochastic algorithm (i.e. non deterministic).
- ▶ The main parameters of the algorithms are:  $n$  (number of individuals in a population),  $k$  (number of selected individuals),  $\mu$  (probability of mutation)

# Genetic algorithms and the TSP

## Representation

A path is a list of cities:  $\{a, b, d, f, e\}$

## Selection

The fitness function is the total cost of a path

## Mutation

Swap two cities in the path  $\{a, b, d, f, e, c\} \rightarrow \{a, d, b, f, e, c\}$

## Re-combination

Random Split Position.

$\{a, b, -, d, f, e, c\} + \{d, f, -, c, b, e, a\} \rightarrow \{a, b, -, c, e, d, f\}$

# Hill Climbing

*Hill-climbing* is an optimisation method in which you explore the search space by going from one solution to one of its neighbours. Intuitively, you try to climb the hill!

In the basic version of hill-climbing, you start from a random solution, you examine its neighbours and you chose the first one better than it according to the fitness function.

You stop when: (i) you have reached a local minimum (i.e. the current solution is better than all its neighbours), or (ii) you have reached a given number of steps.

# Hill Climbing

## Pseudo Code

```
solution := randomSolution()
MAX_FITNESS := FITNESS(solution)
found := True
WHILE found:
    found := False
    neighbours := Neighbours(solution)
    FOR s IN neighbours:
        IF (FITNESS(s) > MAX_FITNESS):
            MAX_FITNESS := FITNESS(s)
            solution := s
            found := True
            BREAK
RETURN point
```

# Hill Climbing

Hill climbing is a very simple optimization algorithm!

# Hill Climbing

Hill climbing is a very simple optimization algorithm!

## Complexity

$\Rightarrow O(t \times k)$ , where  $t$  is the total number of steps, and  $k$  the number of neighbours per path.

# Hill Climbing - Application to the TSP

## Representation

To paths are neighbours if you can turn one into the other one by swapping two cities.

## Complexity

$\Rightarrow O(t \times k)$ , where  $t$  is the total number of steps, and  $k$  the number of neighbours per path.

If  $n$  is the number of cities, what is  $k$ ?

# Hill Climbing - Application to the TSP

## Representation

Two paths are neighbours if you can turn one into the other one by swapping two cities.

## Complexity

$\Rightarrow O(t \times k)$ , where  $t$  is the total number of steps, and  $k$  the number of neighbours per path.

If  $n$  is the number of cities, what is  $k$ ?

$$k = n \Rightarrow \text{Complexity is } O(t \times n)$$

# Hill Climbing - Application to the TSP

## Advantages

- ▶ Very generic (can be applied to any optimisation problem)
- ▶ Very simple
- ▶ Is an anytime algorithm

# Hill Climbing - Application to the TSP

## Advantages

- ▶ Very generic (can be applied to any optimisation problem)
- ▶ Very simple
- ▶ Is an anytime algorithm

## Disadvantages

- ▶ Strongly dependent on the initial random point
- ▶ Gets stuck in local optima (greedy algorithm)

# Simulated annealing

## Avoiding local optima

*Simulated annealing (SA)* is a *generic* and *probabilistic meta-heuristic*, introduced in the 80s.

Genetic algorithms are *biologically inspired*, simulated annealing is “*metallurgy inspired*”.

Thermal annealing is a technique involved in metallurgy to reduce the defects of a material by heating and controlled cooling.

⇒ the heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one.

# Simulated annealing

By analogy with this physical process, each step of the SA algorithm replaces the current solution by a random “nearby” solution, chosen with a probability that depends on the difference between the corresponding function values and on a global parameter  $T$  (called the temperature), that is gradually decreased during the process. The dependency is such that the current solution changes almost randomly when  $T$  is large, but increasingly “downhill” as  $T$  goes to zero. The allowance for “uphill” moves saves the method from becoming stuck at local minima, which are the bane of greedier methods.

# Simulated annealing

The algorithm starts by generating an initial solution (usually a random solution) and by initializing the so-called temperature parameter  $T$ .

Then the following is repeated until the termination condition is satisfied: a solution  $s$  from the neighborhood  $Neighbours(solution)$  of the current solution  $solution$  is randomly sampled and it is accepted as new current solution if

$$FITNESS(solution) < FITNESS(s)$$

or, in case  $FITNESS(solution) \geq FITNESS(s)$ , with a probability which is a function of  $T$  and  $FITNESS(solution) - FITNESS(s)$ , usually  $\exp\left(-\frac{FITNESS(solution) - FITNESS(s)}{T}\right)$ .

# Simulated annealing

## Pseudo Code

```
solution := randomSolution()
T := T_0
WHILE termination conditions not met:
  s := chooseRandomlyFrom(Neighbours(solution))
  IF (FITNESS(solution) < FITNESS(s)):
    solution := s
  ELSE:
    DO:
      solution := s with probability
       $\exp(-(\text{FITNESS}(\text{solution}) - \text{FITNESS}(s))/T)$ .
  Update(T)
ENDWHILE
```