Local search

Artificial intelligence (CK0031)

Francesco Corona

Beyond search

So far a single category of problems: observable, deterministic, known environments where the solution is a sequence of actions

We look at what happens when these assumptions are relaxed

- Algorithms that perform pure **local search** in a state space
- Evaluation and modification of one or more current states
- Alternative to systematic path exploration from a start state

These algorithms are suitable for problems in which all that matters is the solution state, not the path cost to reach it

- Algorithms for continuous state and action spaces
- **Local search in continuous space**, or simply **numerical optimisation**

Beyond search (cont.)

We relax the assumptions of determinism and observability

- The idea is that if an agent cannot predict exactly what percept it will receive, then it will need to consider what to do under each **contingency** that its percepts may reveal
- With partial observability, the agent will also need to keep track of the states it might be in

Beryond search (cont.)

Finally, we investigate **online search**, in which the agent is faced with a state space that is initially unknown and must be explored

Local search and optimisation

Local search and optimisation

Search algos so far are designed to explore spaces systematically

- Systematicity : Keep paths in memory and record which alternatives have been explored at each point along path

A path to that goal also constitutes a solution to the problem

- In many problems, however, path to goal is irrelevant

Local search and optimisation

Example

In the 8-queens problem what matters is the final configuration

- The order in which the queens are added is irrilevant

The same general property holds for many important applications

- Integrated-circuit design, factory-floor layout, job-shop scheduling, automatic programming, telecom network optimisation, vehicle routing, and portfolio management

Local search and optimisation (cont.)

If the path to goal does not matter, we consider a different class of algorithms, ones that do not worry about paths at all

Definition

- **Local search** algorithms operate using a single current node (rather than multiple paths) and generally move only to neighbours of that node

Typically, the paths followed by the search are not retained

Local search and optimisation (cont.)

Local search algos are not systematic, with two key advantages:

- They use very little memory, usually a constant amount;
- They can find reasonable solutions in infinite (continuous) state spaces for which systematic algorithms are unsuitable

Local search is valid for solving pure **optimisation problems**

- Find the best state according to an **objective function**

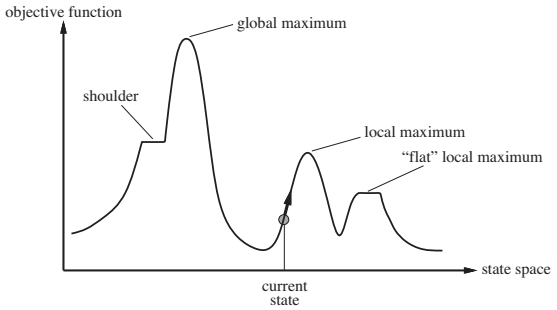
Local search

UFC/DC
AI (CK0031)
2016.2

Local search and optimisation
Hill-climbing search
Simulated annealing
Local beam search
Genetic algorithms
Local search in continuous spaces

Local search and optimisation (cont.)

To understand local search, consider the state-space landscape



A landscape has both a 'location' (the state) and an 'elevation' (the value of the heuristic cost function or objective function)

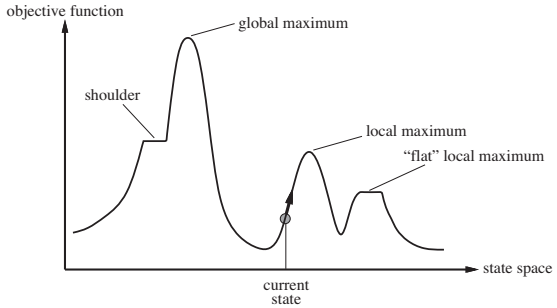
- Local search algorithms explore this landscape
- Current state is modified to improve the objective

Local search

UFC/DC
AI (CK0031)
2016.2

Local search and optimisation
Hill-climbing search
Simulated annealing
Local beam search
Genetic algorithms
Local search in continuous spaces

Local search and optimisation (cont.)



Definition

If elevation corresponds to an objective function, the highest peak

- A **global maximum**

If elevation corresponds to cost, the aim is to find the lowest valley

- A **global minimum**

Conversion from one to the other just by inserting a minus sign

Local search

UFC/DC
AI (CK0031)
2016.2

Local search and optimisation
Hill-climbing search
Simulated annealing
Local beam search
Genetic algorithms
Local search in continuous spaces

Local search and optimisation (cont.)

Definition

A **complete** local search algorithm always finds a goal if one exists

An **optimal** algorithm always finds a global minimum/maximum

Hill-climbing search

Local search and optimisation

Hill-climbing search

The **hill-climbing search** algorithm (**steepest-ascent version**) is a loop that continually moves in the direction of increasing value

- That is, uphill

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

current ← MAKE-NODE(*problem*.INITIAL-STATE)
loop do
  neighbor ← a highest-valued successor of *current*
  if neighbor.VALUE ≤ current.VALUE **then return** *current*.STATE
  current ← *neighbor*

At each step, current node is replaced by the best of its neighbours

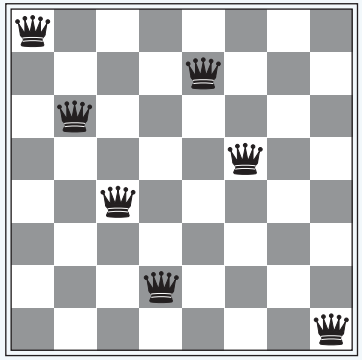
- Terminates at a 'peak' where no neighbour has a higher value

The algorithm does not keep a search tree, so the data structure for current node need only record state and value of the objective

No ahead looking beyond the immediate neighbours of a state

Hill-climbing search (cont.)

Example



Local search algorithms often use a **complete-state formulation**

- Each state has 8 queens on the board, one per column

Hill-climbing search (cont.)

The successors of a state are all possible states generated by moving a single queen to another square in the same column

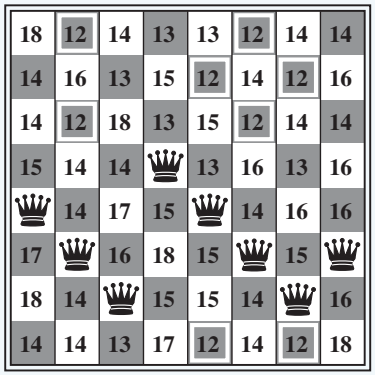
- So, each state has $8 \times 7 = 56$ successors

The heuristic cost function h is the number of pairs of queens that are attacking each other, either directly or indirectly

- The global minimum of this function is zero
- It occurs only at perfect solutions

Hill-climbing search (cont.)

Example



A state with $h = 17$ and the values of all its successors

- The best successors have $h = 12$

If more than one best successor, choose randomly to break the tie

Local search

UFC/DC
AI (CK0031)
2016.2

Local search and optimisation
Hill-climbing search
Simulated annealing
Local beam search
Genetic algorithms
Local search in continuous spaces

Hill-climbing search (cont.)

Hill climbing is also called **greedy local search** because it grabs a good neighbour state without thinking ahead where to go next

- Hill climbing often makes rapid progress toward a solution because it is usually quite easy to improve a bad state

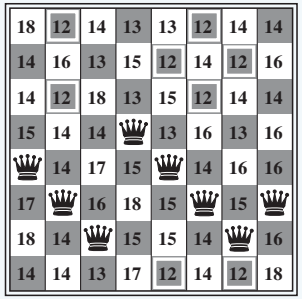
Local search

UFC/DC
AI (CK0031)
2016.2

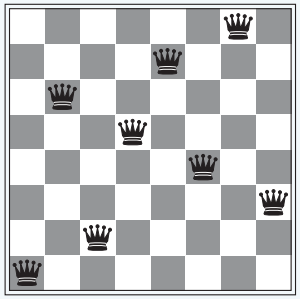
Local search and optimisation
Hill-climbing search
Simulated annealing
Local beam search
Genetic algorithms
Local search in continuous spaces

Hill-climbing search (cont.)

Example



(a)



(b)

From the state (a) it takes just five steps to reach the state in (b)

- State (b) has $h = 1$ and it is very nearly a solution
- Every successor of (b) has a higher cost (dead-end)

Local search

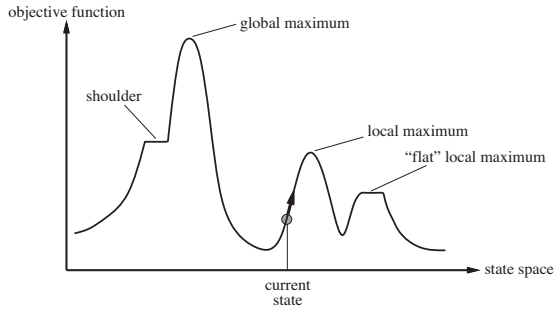
UFC/DC
AI (CK0031)
2016.2

Local search and optimisation
Hill-climbing search
Simulated annealing
Local beam search
Genetic algorithms
Local search in continuous spaces

Hill-climbing search (cont.)

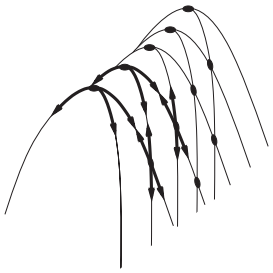
Hill climbing often gets stuck for the following reasons:

- **Local maxima**: It is a peak that is higher than each of its neighbouring states but lower than global maximum
- **Plateaux**: It is a flat area of the state-space landscape, like a flat local maximum or a **shoulder**



Hill-climbing search (cont.)

- **Ridges**: A sequence of local maxima that are not directly connected to each other



The grid of states (dark circles) is superimposed on a ridge

- From each local maximum, all the available actions point downhill

In each case, the algorithm reaches a no-more-progress point

Local search

UFC/DC
AI (CK0031)
2016.2

Local search and optimisation
Hill-climbing search
Simulated annealing
Local beam search
Genetic algorithms
Local search in continuous spaces

Hill-climbing search (cont.)

Example

Steepest-ascent hill climbing gets stuck 86% of the time, solving 14% of problem instances, starting from a random 8-queens state

It works quickly, taking 4 steps on average when it succeeds and 3 when it gets stuck, not bad for a state space with $8^8 \approx 17$M states

Local search

UFC/DC
AI (CK0031)
2016.2

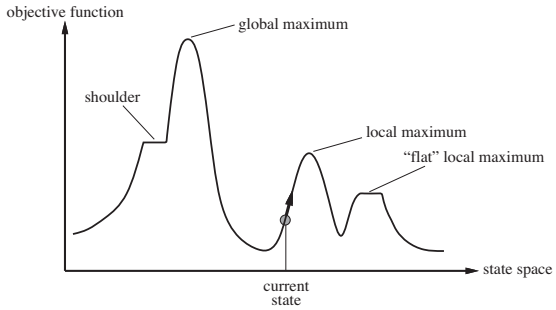
Local search and optimisation
Hill-climbing search
Simulated annealing
Local beam search
Genetic algorithms
Local search in continuous spaces

Hill-climbing search (cont.)

The algorithm halts if it reaches a plateau where best successors have the same value as the current state

It may be good idea to keep going, to allow a **sideways move**

- In the hope that the plateau is really a shoulder



Local search

UFC/DC
AI (CK0031)
2016.2

Local search and optimisation
Hill-climbing search
Simulated annealing
Local beam search
Genetic algorithms
Local search in continuous spaces

Hill-climbing search (cont.)

If we always allow sideways moves when there is no uphill

- An infinite loop will occur whenever the algorithm reaches a flat local maximum that is not a shoulder

Solution: Limit the number of consecutive sideways moves allowed

Hill-climbing search (cont.)

Example

A limit of 100 consecutive sideways moves in the 8-queens problem

- The percentage of problem instances solved by hill climbing raises from 14% to 94%
- The algorithm averages 21 steps for each successful instance and 64 for each failure

Local search

UFC/DC
AI (CK0031)
2016.2

Local search and optimisation
Hill-climbing search
Simulated annealing
Local beam search
Genetic algorithms
Local search in continuous spaces

Hill-climbing search (cont.)

Many variants of hill climbing have been invented

Stochastic hill climbing chooses at random from among the uphill moves, with a probability of selection that can vary with the steepness of the uphill move

- This usually converges more slowly than steepest ascent, but in some state landscapes, it finds better solutions

First-choice hill climbing implements stochastic hill climbing by generating successors randomly until one is generated that is better than the current state

- Good strategy, when a state has thousands of successors

Local search

UFC/DC
AI (CK0031)
2016.2

Local search and optimisation
Hill-climbing search
Simulated annealing
Local beam search
Genetic algorithms
Local search in continuous spaces

Hill-climbing search (cont.)

The hill-climbing algorithms described so far are incomplete

- Even when a goal exist, they can get stuck on local maxima

Random-restart hill climbing conducts a series of hill-climbings from randomly generated start states, until a goal is found

- It is trivially complete with probability approaching 1, because it will eventually generate a goal state as the initial state
- If each hill-climbing search has a probability p of success, then the expected number of restarts required is $1/p$

Local search

UFC/DC
AI (CK0031)
2016.2

Local search and optimisation
Hill-climbing search
Simulated annealing
Local beam search
Genetic algorithms
Local search in continuous spaces

Hill-climbing search (cont.)

Example

For 8-queens instances with no sideways moves allowed, $p \approx 0.14$

- Roughly 7 iterations to find a goal (6 failures and 1 success)

The expected number of steps is the cost of one good iteration plus $(1 - p)/p$ times the cost of failure (roughly 22 steps in all)

When we allow sideways moves, $1/0.94 \approx 1.06$ iterations are needed on average and $(1 \times 21) + (0.06/0.94) \times 64 \approx 25$ steps

For 8-queens, random-restart hill climbing is thus very effective

For 3M queens, the approach can find solutions in under a minute

Hill-climbing search (cont.)

Remark

Success of hill climbing depends on the shape of the state-space

- If there are few local maxima and plateaux, random-restart hill climbing will find a good solution very quickly

Many real problems have a rather complex state space landscape

- NP-hard problems typically have an exponential number of local maxima

Despite this, a reasonably good local maximum can often be found

Simulated annealing

Local search and optimisation

Simulated annealing

A hill-climbing algorith that never makes 'downhill' moves toward states with lower value is guaranteed to be incomplete

- Because it can get stuck on a local maximum

In contrast, a random walk, that is moving to a successor chosen uniformly at random from the set of successors, is complete

- But, it is also extremely inefficient

It seems reasonable to try to combine hill climbing with a random walk in some way that yields both efficiency and completeness

Simulated annealing is such an algorithm

Simulated annealing (cont.)

function SIMULATED-ANNEALING($problem$, $schedule$) **returns** a solution state
  inputs: $problem$, a problem
    $schedule$, a mapping from time to "temperature"

  $current \leftarrow$ MAKE-NODE($problem$.INITIAL-STATE)
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** $current$
    $next \leftarrow$ a randomly selected successor of $current$
    $\Delta E \leftarrow next$.VALUE $-$ $current$.VALUE
    if $\Delta E > 0$ **then** $current \leftarrow next$
    else $current \leftarrow next$ only with probability $e^{\Delta E/T}$

The innermost loop of simulated annealing is hill climbing alike

- Instead of picking the best move, it picks a random move
- If the move improves the situation, it is always accepted
- If not, the algorithm accepts the move, with some probability

The probability decreases exponentially with the 'badness' of the move, the amount ΔE by which the evaluation is worsened

Simulated annealing (cont.)

The probability also decreases as T goes down:

- 'Bad' moves are more likely to be allowed at the start when T is high, and they become more unlikely as T decreases

If the schedule lowers T slowly enough, the algorithm will find a global optimum with probability approaching 1

Local search

UFC/DC
AI (CK0031)
2016.2

Local search and optimisation
Hill-climbing search
Simulated annealing
Local beam search
Genetic algorithms
Local search in continuous spaces

Simulated annealing (cont.)

Compare its performance to that of random-restart hill climbing

Exercise

Generate a large number of 8-puzzle and 8-queens instances and solve them (where possible) by hill climbing (steepest-ascent and first-choice variants), hill climbing with random restart, and simulated annealing

Measure the search cost and percentage of solved problems, and graph these against the optimal solution cost

Comment on your results

Local search

UFC/DC
AI (CK0031)
2016.2

Local search and optimisation
Hill-climbing search
Simulated annealing
Local beam search
Genetic algorithms
Local search in continuous spaces

Local beam search
Local search and optimisation

Local search

UFC/DC
AI (CK0031)
2016.2

Local search and optimisation
Hill-climbing search
Simulated annealing
Local beam search
Genetic algorithms
Local search in continuous spaces

Local beam search

Keeping just one node in memory might seem to be extreme

- The **local beam search** algorithm keeps track of k states rather than just one

It begins with k randomly generated states and at each step all the successors of all k states are generated

- If any one is a goal, the algorithm halts
- Otherwise, the algorithm selects the k best successors from the complete list and repeats

Local beam search (cont.)

Remark

A local beam search with k states might seem to be nothing more than running k random restarts in parallel instead of in sequence

- In fact, the two algorithms are quite different

- In a random-restart search, each search runs independently
- In a local beam search, information is passed among threads

Remark

The algorithm quickly abandons unfruitful searches and it moves to where the most progress is being made

Local search

UFC/DC
AI (CK0031)
2016.2

Local search and optimisation
Hill-climbing search
Simulated annealing
Local beam search
Genetic algorithms
Local search in continuous spaces

Local beam search (cont.)

In its simplest form, the algo can suffer from a lack of diversity

- The states k can quickly become concentrated in a small region of the state space (expensive hill climbing)

A variant called **stochastic beam search** helps with this issue

- Instead of choosing the best k from the pool of candidates, stochastic beam search chooses k successors at random
- The probability of choosing a given successor being an increasing function of its value

Local search

UFC/DC
AI (CK0031)
2016.2

Local search and optimisation
Hill-climbing search
Simulated annealing
Local beam search
Genetic algorithms
Local search in continuous spaces

Local beam search (cont.)

Stochastic beam search bears and the process of natural selection

- 'Successors' (offspring) of a 'state' (organism) populate the next generation according to its 'value' (fitness)

Local search

UFC/DC
AI (CK0031)
2016.2

Local search and optimisation
Hill-climbing search
Simulated annealing
Local beam search
Genetic algorithms
Local search in continuous spaces

Genetic algorithms

Local search and optimisation

Genetic algorithms

A **genetic algorithm** or **GA** is a variant of stochastic beam search

- Successor states are generated by combining two parent states rather than by modifying a single state

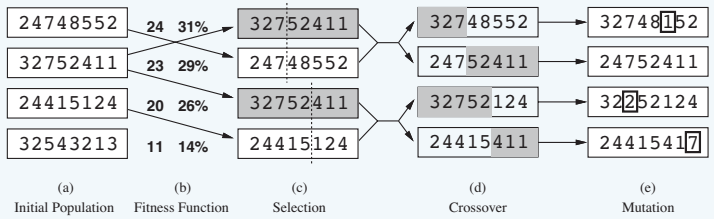
Like beam searches, GAs begin with a set of k randomly generated states (**population**) and each state (**individual**) is represented as a string over a finite alphabet (commonly, a string of 0s and 1s)

Genetic algorithms (cont.)

Example

An 8-queens state must specify the positions of 8 queens, each in a column of 8 squares, and so requires $8 \times \log_2(8) = 24$ [bits]

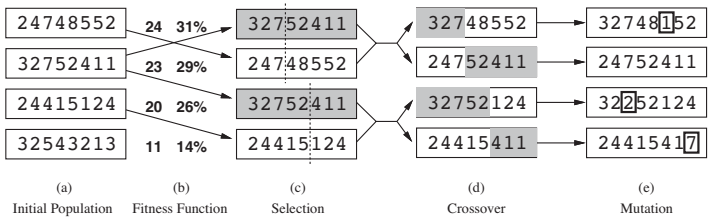
Alternatively, the state could be represented as 8 digits, each in the range from 1 to 8



A population of four 8-digit strings representing 8-queens states

Genetic algorithms (cont.)

The next step is the production of a new generation of states



In (b), each state is rated by the **objective** (**fitness**) **function**

- A fitness function should take higher values for better state

The number of non-attacking pairs of queens (28 for a solution)

- The values fo the four states are 24, 23, 20, and 11

In this variant of the genetic algorithm, the probability of being chosen for reproducing is directly proportional to the fitness score

Genetic algorithms (cont.)

In (c), two state pairs are selected at random for reproduction

- In accordance with the probabilities in (b)

Notice that one individual is selected twice and one not at all

For each pair to be mated, a **crossover point** is chosen randomly

- Crossover points here are after the third digit in the first pair and after the fifth digit in the second pair

Genetic algorithms (cont.)

Offspring are created by crossing over parents at crossover point

- The first child of the first pair gets the first three digits from first parent and the remaining digits from second parent
- The second child gets the first three digits from the second parent and the rest from the first parent

Local search

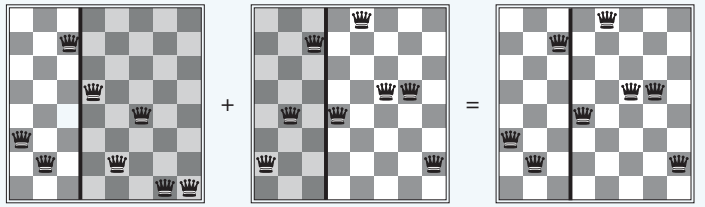
UFC/DC
AI (CK0031)
2016.2

Local search and optimisation
Hill-climbing search
Simulated annealing
Local beam search
Genetic algorithms
Local search in continuous spaces

Genetic algorithms (cont.)

Example

The 8-queens states involved in this reproduction step



When two parent states are quite different, the crossover operation can produce a state that is a long way from either parent state

Common that the population is diverse early on in the process

- Crossover (like simulated annealing) frequently takes large steps in the state space early in the search process
- Steps get smaller later on when individuals get similar

Local search

UFC/DC
AI (CK0031)
2016.2

Local search and optimisation
Hill-climbing search
Simulated annealing
Local beam search
Genetic algorithms
Local search in continuous spaces

Genetic algorithms (cont.)

function GENETIC-ALGORITHM(*population*, FITNESS-FN) **returns** an individual
  inputs: *population*, a set of individuals
    FITNESS-FN, a function that measures the fitness of an individual

  repeat
    new_population ← empty set
    for $i = 1$ **to** SIZE(*population*) **do**
      x ← RANDOM-SELECTION(*population*, FITNESS-FN)
      y ← RANDOM-SELECTION(*population*, FITNESS-FN)
      child ← REPRODUCE(x, y)
      if (small random probability) **then** *child* ← MUTATE(*child*)
      add *child* to *new_population*
    population ← *new_population*
  until some individual is fit enough, or enough time has elapsed
  return the best individual in *population*, according to FITNESS-FN

---

function REPRODUCE(x, y) **returns** an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING($x, 1, c$), SUBSTRING($y, c+1, n$))

Local search

UFC/DC
AI (CK0031)
2016.2

Local search and optimisation
Hill-climbing search
Simulated annealing
Local beam search
Genetic algorithms
Local search in continuous spaces

Local search in continuous spaces

Local search in continuous spaces

The discrete/continuous characterisation applies to the state of the environment, to the way time is handled, and to the percepts and actions of the agent

- Most real-world environments are continuous

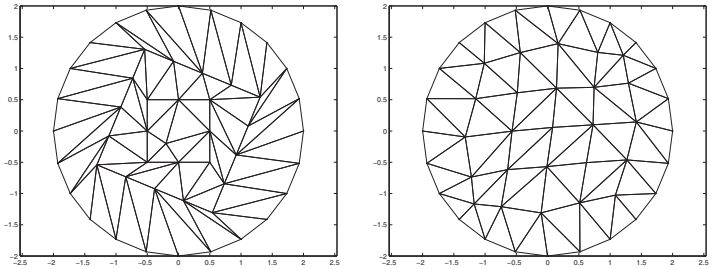
Local search in continuous spaces (cont.)

Example

Consider a given triangulation of a domain $\Omega \subset \mathbb{R}^2$

We want to modify the location of the vertices of the triangles inside Ω in order to optimize some measure of triangles' quality

- We want to minimise distortion wrt an equilateral triangle

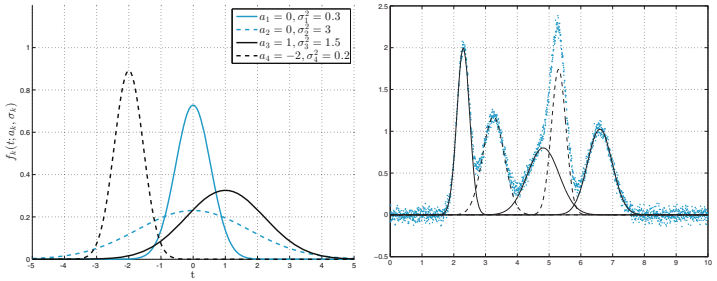


Local search in continuous spaces (cont.)

Example

Applications of voice identification, like those on phones, compress the acoustic signal into a set of parameters that fully characterise it

The signal intensity is modelled as a sum of m Gaussian functions (peaks), each parameterised by two coefficients (centre + spread)



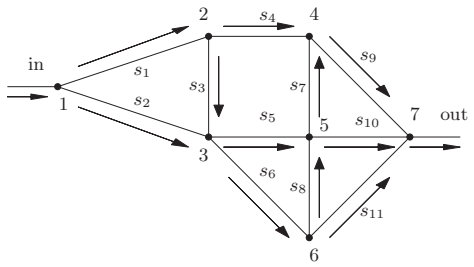
Find the set of $(2 \times m)$ parameters that best (in the least-squares sense) represent the intensity of the recorded audio signal

Local search in continuous spaces (cont.)

Example

Consider a network of n roads and p cross roads

- Every minute M vehicles travel through the network
- On the j-th road the maximum speed limit is $v_{j,m}$ [km min^{-1}]
- Max $\rho_{j,m}$ vehicles per km can transit on the j-th road s_j



Find the density ρ_j [vehicles km^{-1}] on road s_j s.t. $\rho_j \in [0, \rho_{j,m}]$ that minimises the average travel time from entrance to exit

Local search in continuous spaces (cont.)

None of the algorithms we have described can handle continuous state and action spaces, at least not in their basic formulation

- This is because they have infinite branching factors

Local search for finding optimal solutions in continuous spaces

- **Numerical optimisation**