Artificial Intelligence:

Search Methods for Problem Solving

Stochastic Local Search:

Randomized Algorithms

A First Course in Artificial Intelligence: Chapter 4

Deepak Khemani

Department of Computer Science & Engineering

IIT Madras

Exploding Search Spaces

N	Candidates in SAT: 2^N	Candidates in TSP: N!	Ratio
1	2	1	0.5
2	4	2	0.5
3	8	6	0.75
4	16	24	1.5
5	32	120	3.75
6	64	720	11.25
7	128	5040	39.38
8	256	40,320	157.5
50	1,125,899,906,842,624	$3.041409320 \times 10^{64}$	3×10^{49}
100	$1.267650600 \times 10^{30}$	$9.332621544 \times 10^{157}$	10^{127}

Worst case time to solve SAT with 50 variables

N	SAT: 2^N	TSP (brute force): N!	Ratio
50	1,125,899,906,842,624	$3.041409320 \times 10^{64}$	3×10^{49}

Inspecting a million nodes a second: 1,125,899,906 seconds

Assuming a 100 seconds in a minute: 11258999 minutes

Assuming a 100 minutes in an hour: 112589 hours

Assuming a 100 hours in a day: 1125 days

Assuming a 1000 days in a year: 1.1 year

Worst case time to solve SAT with 100 variables

N	SAT: 2^N	TSP (brute force): N!	Ratio
100	$1.267650600 \times 10^{30}$	$9.332621544 \times 10^{157}$	10^{127}

Inspecting a million nodes a second: 1.27×10^{24} seconds

Assuming a 100 seconds in a minute: 1.27×10^{22} minutes

Assuming a 100 minutes in an hour: 1.27×10^{20} hours

Assuming a 100 hours in a day: 1.27×10^{18} day

Assuming a 1000 days in a year: 1.27×10^{15} years

Given a 100 years in a century: 1.27×10^{13} centuries

A thousand trillion centuries!

Escaping local optima

Given that

it is difficult to define heuristic functions
that are monotonic and well behaved

the alternative is

to look for algorithms that can do better than Hill Climbing.

We look at another deterministic method next,
and will look at randomized methods later

First we look at searching in
the solution space or plan space

Local Optima

Hill Climbing – a local search algorithm

Move to the best neighbour if it is better, else terminate

```
Hill Climbing
    N ← Start
    newNode ← best(moveGen (N))
    While newNode is better than N Do
            N ← newNode
            newNode ← best(moveGen (N))
    endWhile
    return N
End
```

In practice sorting is not needed, only the best node

Algorithm Hill Climbing

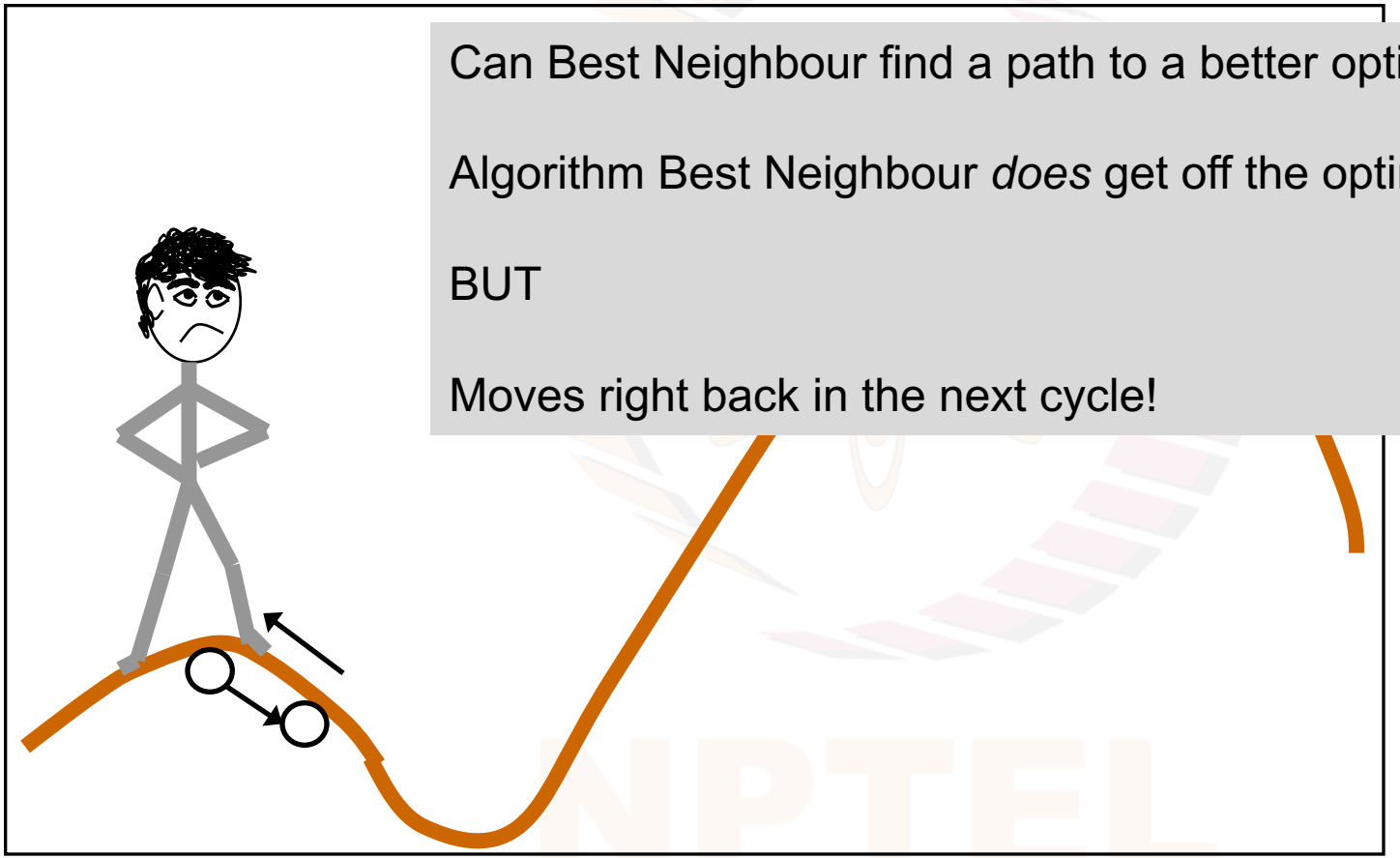
Best Neighbour– a variation of Hill Climbing

Move to the best neighbour anyway

```
Best Neighbour
    N ← Start
    bestSeen ← N
    Until some termination criterion
            N ← best(moveGen (N))
            IF N better than bestSeen
                    bestSeen ← N
    return bestSeen
End
```

Algorithm needs external criterion to terminate

Getting off a Local Optima



Can Best Neighbour find a path to a better optimum?

Algorithm Best Neighbour *does* get off the optimum

BUT

Moves right back in the next cycle!

Tabu Search – not allowed back immediately

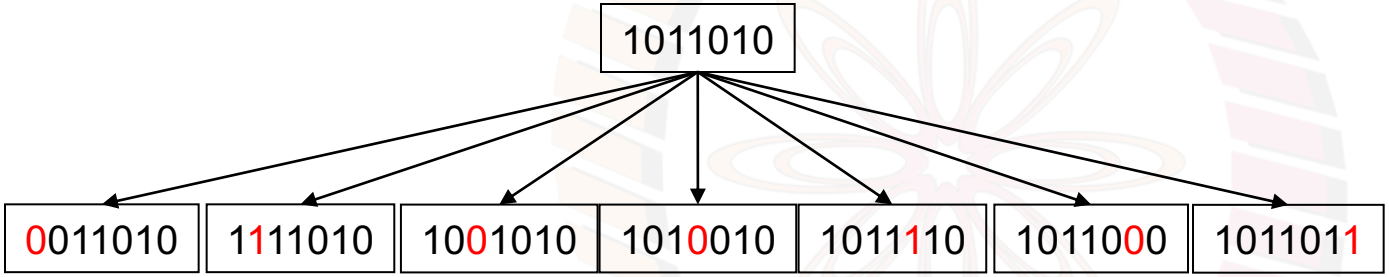
```
Best Neighbour
    N ← Start
    bestSeen ← N
    Until some termination criterion
            N ← best(allowed(moveGen (N))
            IF N better than bestSeen
                    bestSeen ← N
    return bestSeen
End
```

Move to the best neighbour anyway,
but only if it is not tabu

The word *tabu* comes from the Tongan word to indicate things that cannot be touched because they are sacred. - *Wikipedia*

Tabu Search for SAT

When N= 7, the *neigbourhood function* N_1 changes one bit.

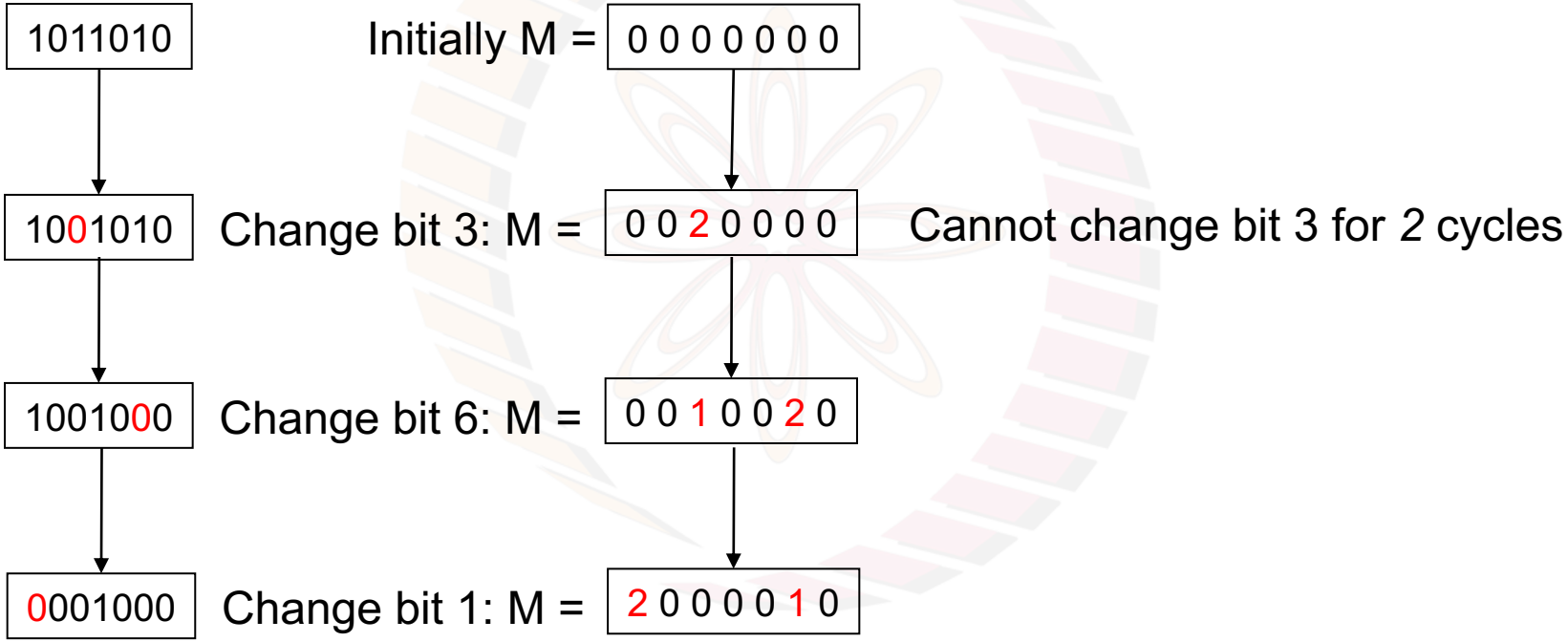


Maintain a memory array M that keeps track of how many moves ago a bit was changed

Initially M = 0 0 0 0 0 0 0

A *tabu tenure tt* defines the window within which any bit cannot be changed

Tabu Search: an illustration (*let tt=2*)



A *tabu tenure tt* defines the window within which any bit cannot be changed

Tabu Search – ASPIRATION criterion

```
Best Neighbour
    N ← Start
    bestSeen ← N
    Until some termination criterion
            N ← best(allowed(moveGen (N))
            IF N better than bestSeen
                    bestSeen ← N
    return bestSeen
End
```

Move to the best neighbour anyway,
but only if it is not tabu

If a *tabu* move results in a node that is better than bestSeen then make an exception for it (that is, add it to allowed set).

Tabu Search: driving search into newer areas

Maintain a frequency array F that keeps track of how many times a bit was changed

Initially F = 0 0 0 0 0 0 0

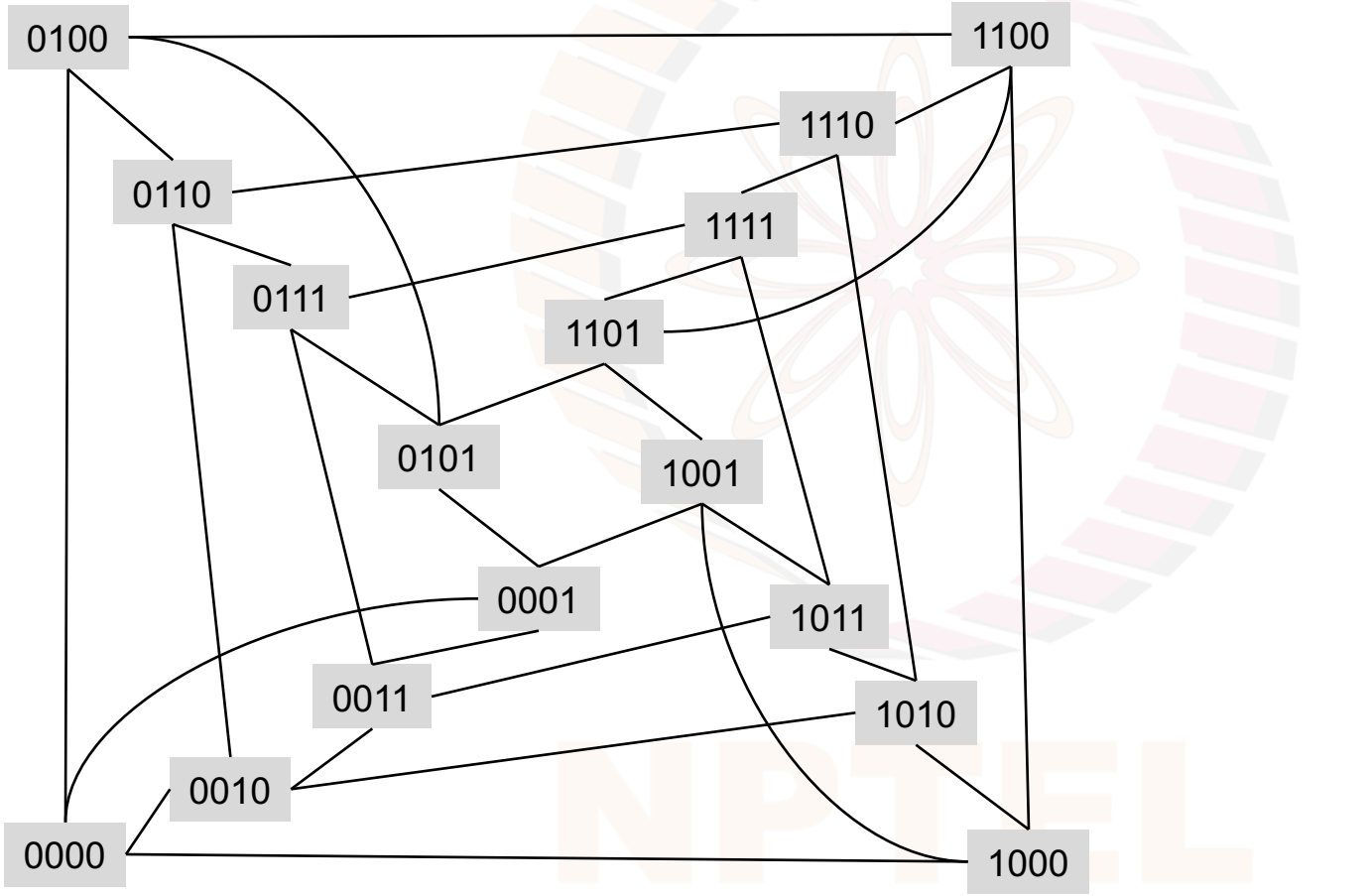
Let F = $F_1\ F_2\ F_3\ F_4\ F_5\ F_6\ F_7$

Penalize each move by a factor proportional to frequency. Let $node_i$ be generated by changing the i^{th} bit

$eval(node_i) \leftarrow eval(node_i) - F_i$

\- for a maximization problem

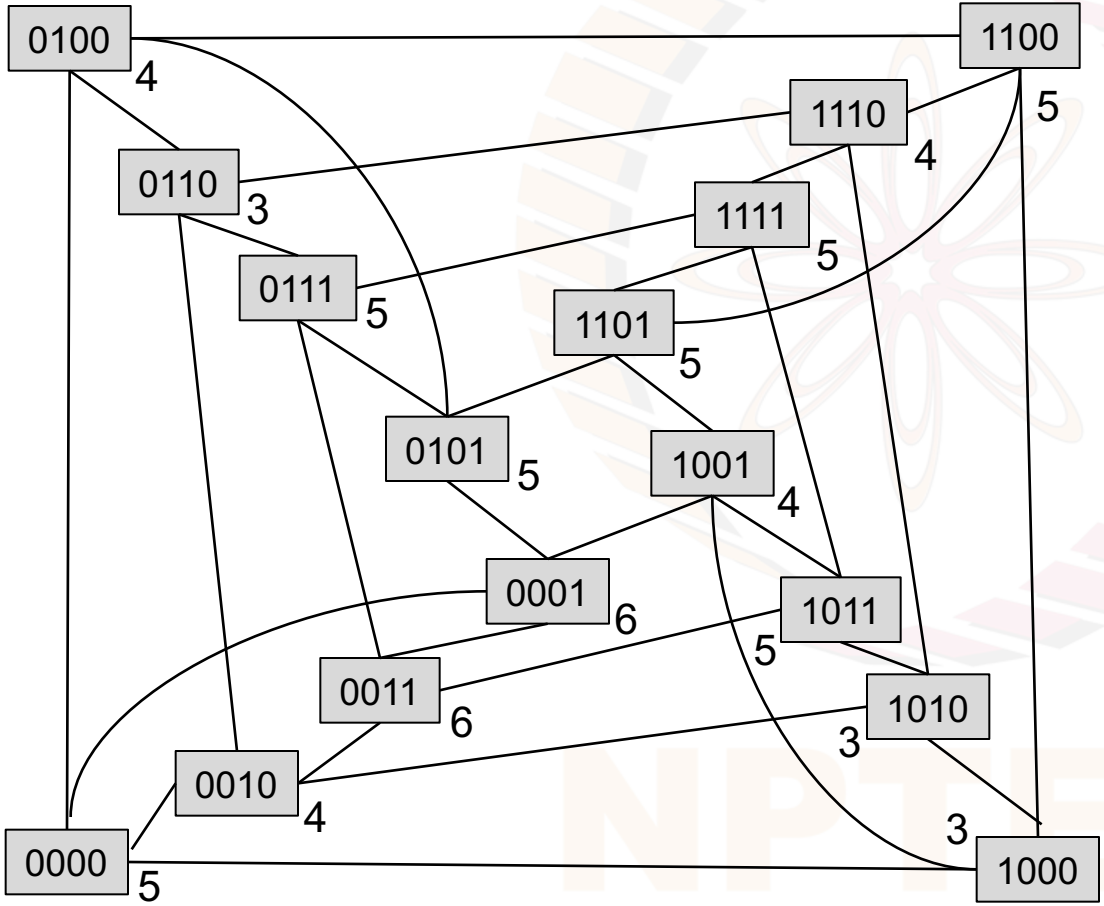
The solution space for a 4 variable SAT problem



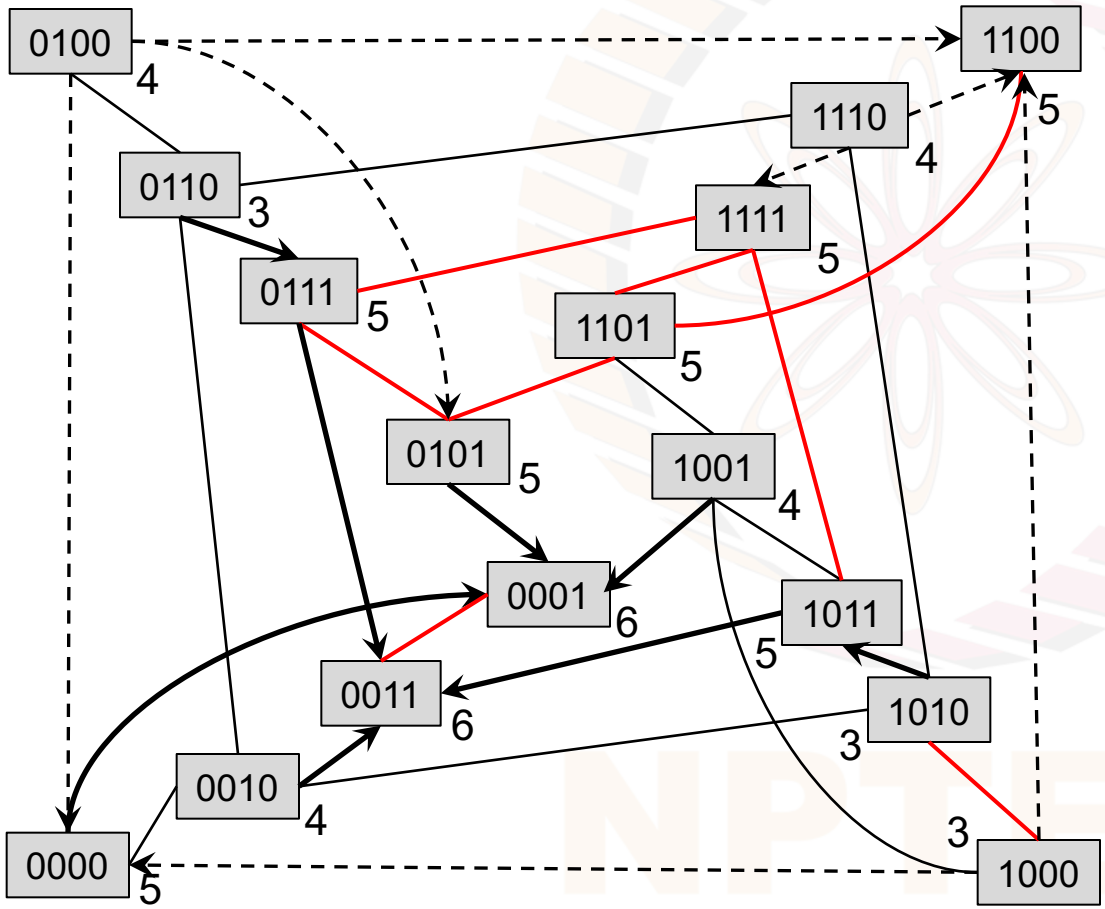
A formula with 6 clauses

$F = (a \vee \neg b) \wedge (\neg a \vee b \vee c) \wedge (\neg c \vee d) \wedge (d) \wedge (\neg a \vee b \vee d) \wedge (\neg a \vee \neg d)$

h = number of satisfied clauses



Gradients and ridges



F = (a∨¬b)∧(¬a∨b∨c)∧(¬c∨d)
∧(d)∧(¬a∨b∨d)∧(¬a∨¬d)
h = number of satisfied clauses

a ridge

one of multiple best neighbours

unique best neighbour

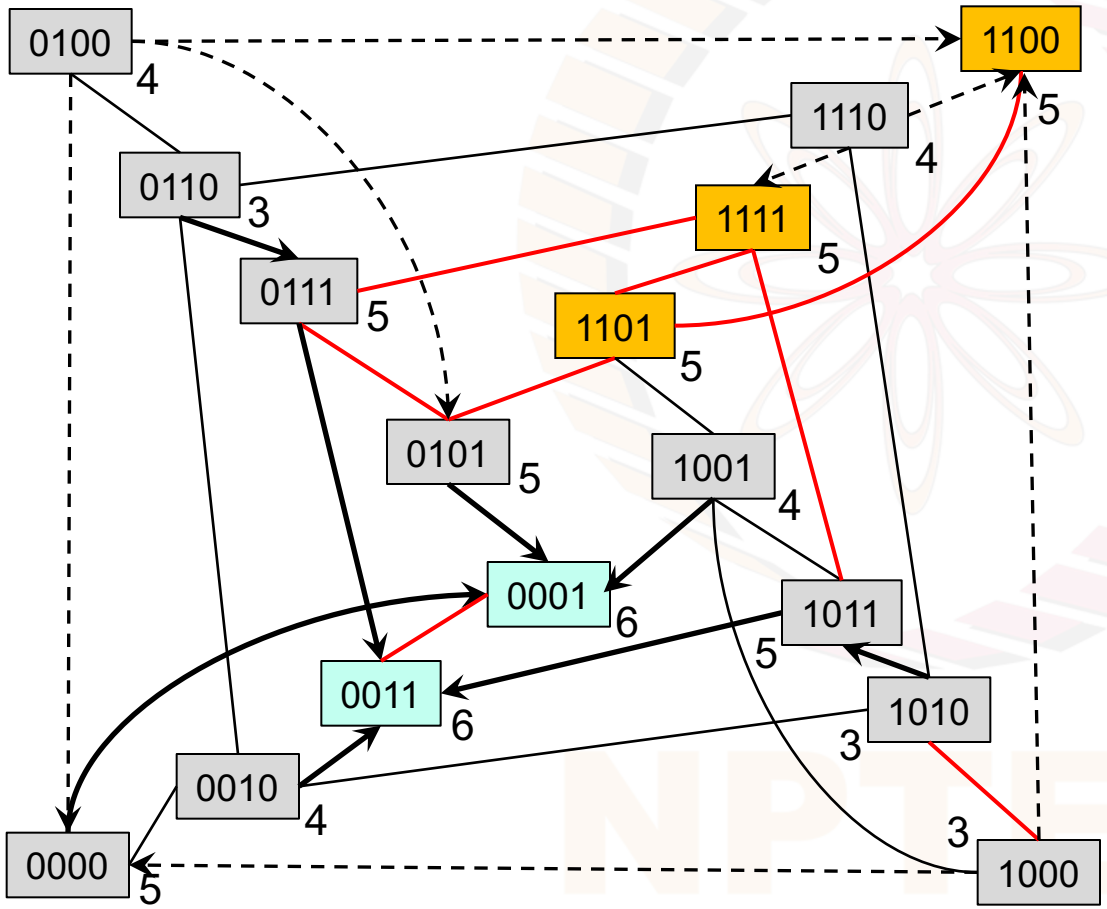
Maxima: global and local

$F = (a \vee \neg b) \wedge (\neg a \vee b \vee c) \wedge (\neg c \vee d) \wedge (d) \wedge (\neg a \vee b \vee d) \wedge (\neg a \vee \neg d)$

h = number of satisfied clauses

local maximum

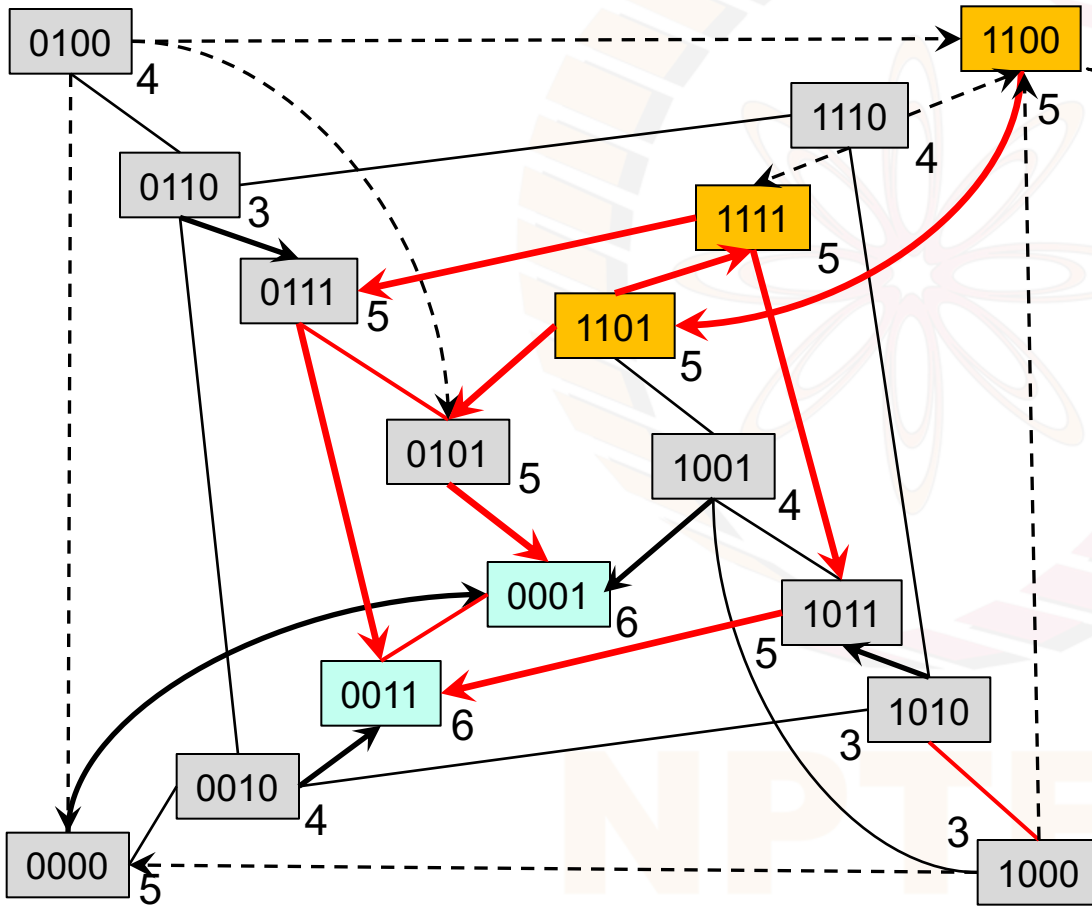
global maximum



Tabu Search

$F = (a \vee \neg b) \wedge (\neg a \vee b \vee c) \wedge (\neg c \vee d) \wedge (d) \wedge (\neg a \vee b \vee d) \wedge (\neg a \vee \neg d)$

h = number of satisfied clauses



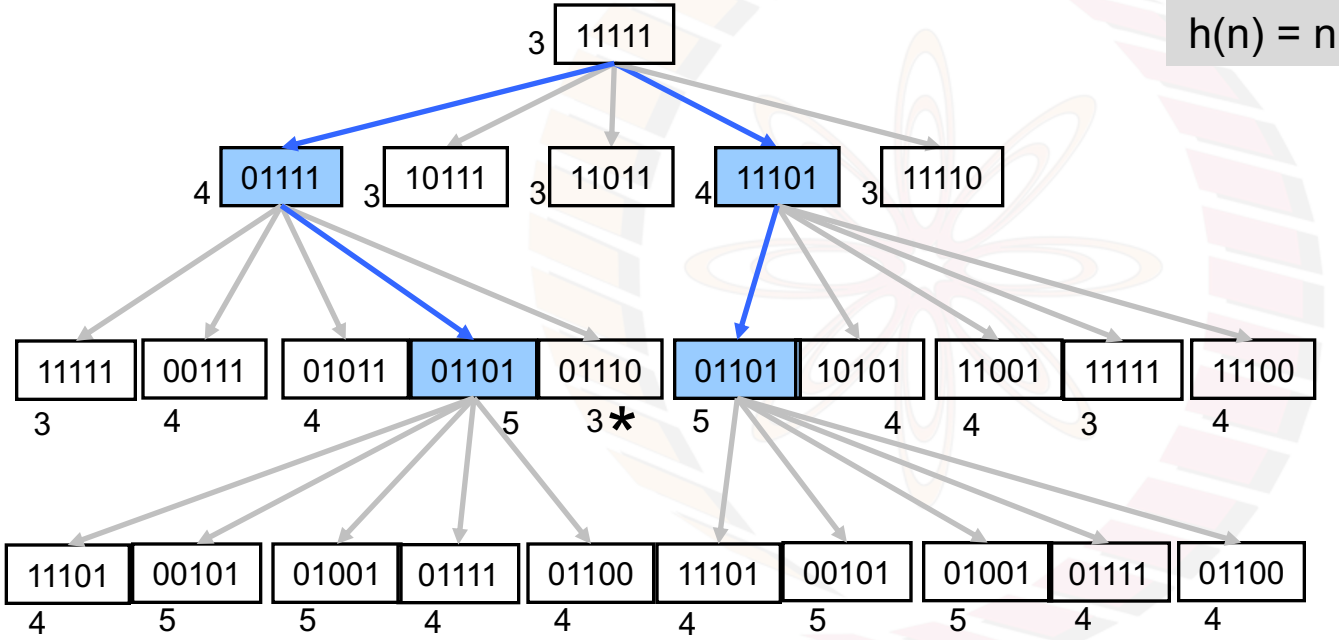
Starting from here Tabu Search traverses one of the three possible routes along the ridges to reach the goal

Stochastic Local Search Methods

Beam Search for SAT

$(b \vee \neg c) \wedge (c \vee \neg d) \wedge (\neg b) \wedge (\neg a \vee \neg e) \wedge (e \vee \neg c) \wedge (\neg c \vee \neg d)$

h(n) = number of clauses satisfied



In solution space search why should we have a specific start node.

Iterated Hill Climbing tries many different start nodes.

Iterated Hill Climbing

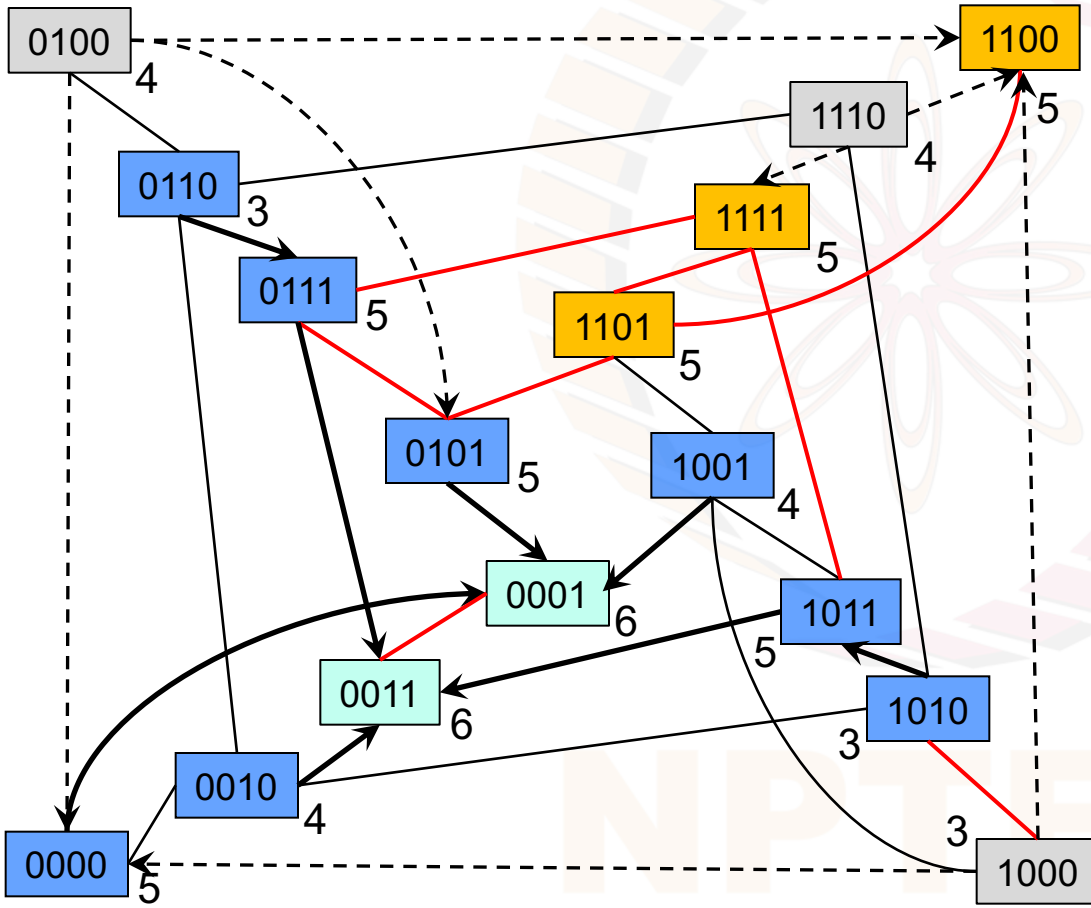
ITERATED-HILL-CLIMBING(N)

```
1  bestNode ← random candidate solution
2  repeat N times
3        currentBest ← HILL-CLIMBING(new random candidate solution)
4        if h(currentBest) is better than h(bestNode)
5              bestNode ← currentBest
6  return bestNode
```

Iterated Hill Climbing

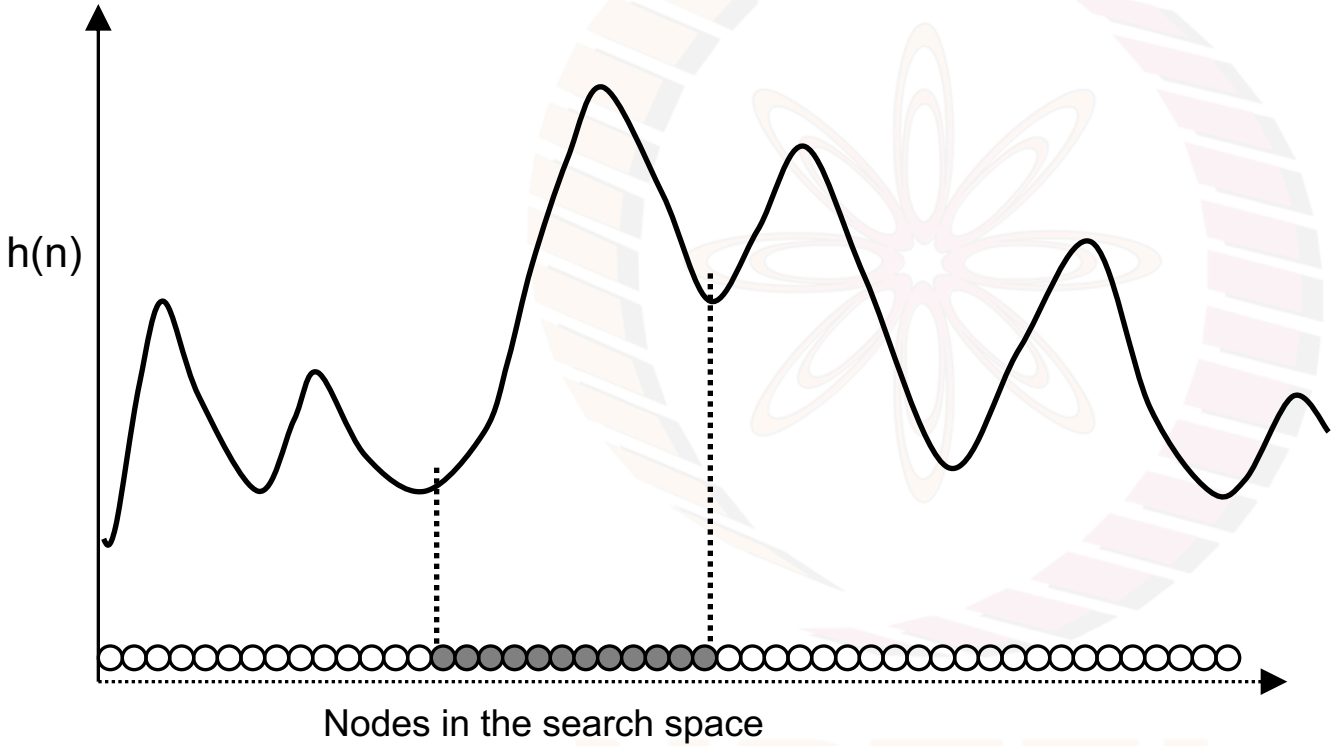
$F = (a \vee \neg b) \wedge (\neg a \vee b \vee c) \wedge (\neg c \vee d) \wedge (d) \wedge (\neg a \vee b \vee d) \wedge (\neg a \vee \neg d)$

h = number of satisfied clauses



Iterated Hill Climbing reaches solution from any of these nodes as the starting point

Iterated Hill Climbing



A random starting point from any of the shaded nodes would lead to the global maximum.

Random Walk – Pure Exploration

```
RandomWalk()
    1   node ← random candidate solution or start
    2   bestNode ← node
    3   for i ← 1 to n
    4       do node ← RandomChoose(MoveGen(node))
    5           if node is better than bestNode
    6               then bestNode ← node
    7   return bestNode
```

RandomWalk explores the search space in a random fashion.

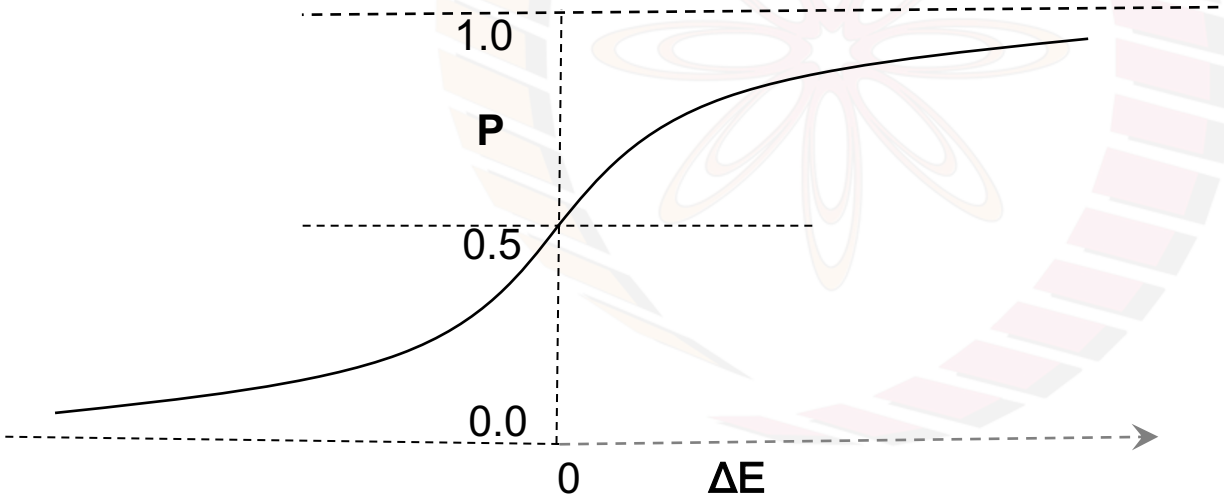
Hill Climbing does pure exploitation (of the gradient).

Stochastic Hill Climbing

Accept a good random move with high probability

Accept a bad random move too but with low probability

- combines exploration with exploitation



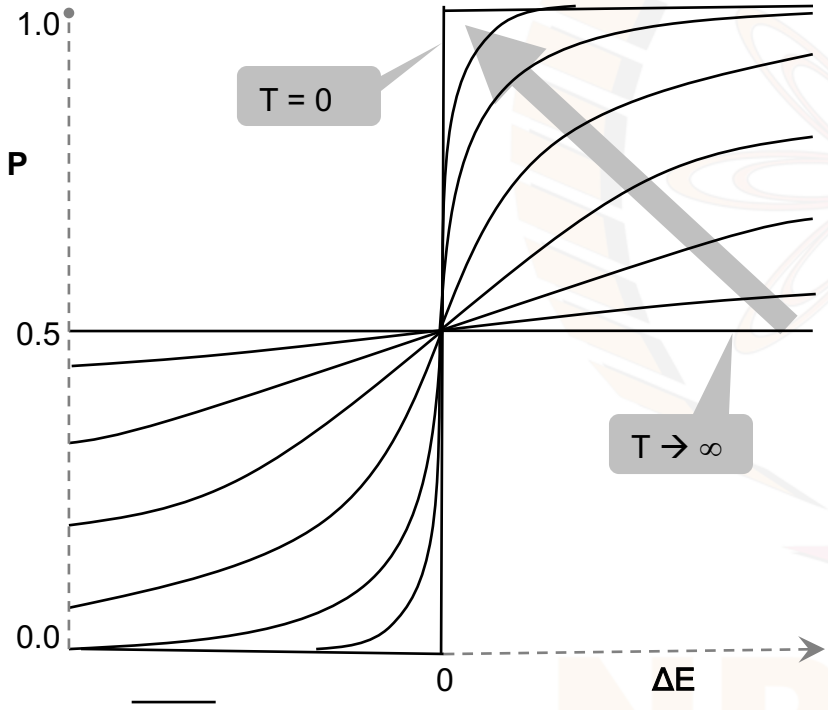
Accept a move from v_c to v_n with probability $P = 1/(1 + e^{-\Delta E/T})$ given by the Sigmoid function, where $\Delta E = (\text{eval}(v_n) - \text{eval}(v_c))$ – for a maximization problem, and T is a parameter

Begin with Exploration → End with Exploitation

```
SimulatedAnnealing()
    1   node ← random candidate solution or start
    2   bestNode ← node
    3   T ← some large value
    4   for time ← 1 to numberOfEpochs
    5     do while some termination criteria     /* M cycles in a simple case */
    6       do
    7         neighbour ← RandomNeighbour(node)
    8         ΔE ← Eval(neighbour) − Eval(node)
    9         if Random(0, 1) < 1 / (1+e^(−ΔE/T))
   10           then node ← neighbour
   11             if Eval(node) > Eval(bestNode)
   12               then bestNode ← node
   13     T ← CoolingFunction(T, time)
   14 return bestNode
```

We use the function *Eval* instead of *h* in the style used in optimization. Function *CoolingFunction* lowers the temperature after each. Function *Random*(0,1) generates a random number in the range 0 to 1 with uniform probability.

Effect of T on the probability



The probability curves (Sigmoid function) for different values of T. The probability of making a move increases as ΔE increases. For very large T the algorithm behaves like Random Walk. As T tends to zero the behaviour approaches the Hill Climbing algorithm.

Effect of temperature

$-\Delta E = eval(v_c) - eval(v_n) = -13$

T	$e^{-13/T}$	P
1	0.000002	1.0
5	0.0743	0.93
10	0.2725	0.78
20	0.52	0.66
50	0.77	0.56
10^{10}	0.99999…	0.5

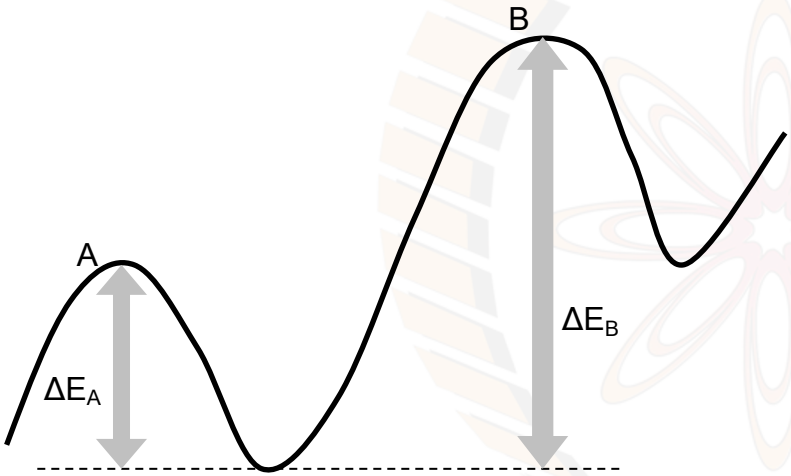
Effect of ΔE

$eval(v_c) = 107, T = 10$

$eval(v_n)$	$eval(v_c) - eval(v_n)$	$e^{eval(vc) - eval(vn)/10}$	P
80	27	14.88	0.06
100	7	2.01	0.33
107	0	1.00	0.50
120	-13	0.27	0.78
150	-43	0.01	0.99

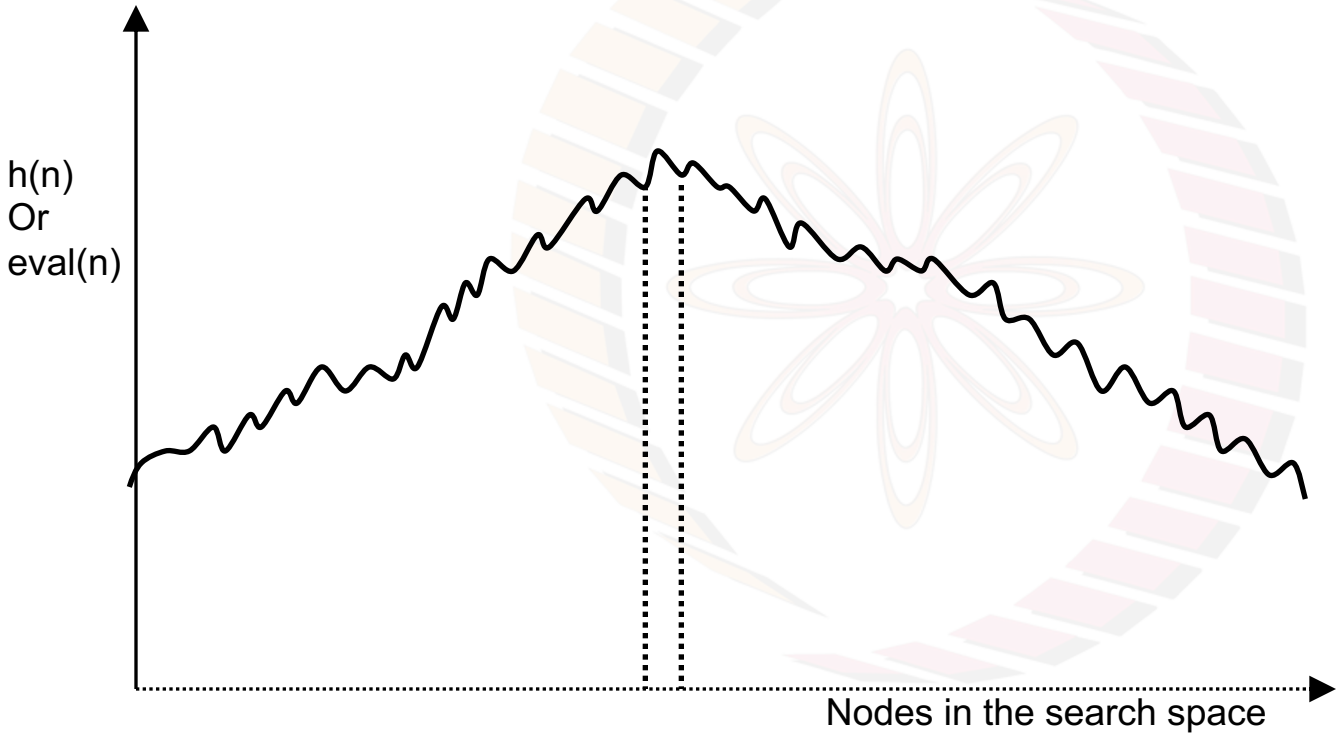
Source: Michalewicz and Fogel. How to Solve It: Modern Heuristics, page 119.

Intuition – why Simulated Annealing works



To move from A to B the algorithm has to incur a smaller loss than to move from B to A. Simulated Annealing is more likely to move from A to B than vice versa.

Simulated Annealing works well when…



A jagged surface with many local optima but a broader trend towards the optimum is well suited for Simulated Annealing. Observe that the footprint of Hill Climbing is very small.

Population Based Methods

Evolution in Nature

Some fundamental questions…

What is life?

Why do we have different life forms?

Is evolution a design process?

Is Nature looking for a perfect design?

If yes, to what purpose?

What is Life?

In his book on Artificial Life and Creatures

"Creation: Life and how to Make it"

author Steve Grand profoundly observes that he has "*uncovered the most important law of nature*". And it is this:

Things that persist, persist.

Things that don't, don't.

Things that persist, persist.

Grand goes on to clarify that much of what we see persisting,

like clouds on a hilltop, ripples in a pond,

or even human bodies,

are in some way a persistent impression

on a constantly changing landscape.

Our own bodies are made up of atoms that are entirely different from when we were children.

Life

Inanimate matter has a natural persistence
for example, rocks, water and so on

Life forms have a different form of persistence
which includes growth of bodies
and procreation of offspring

Plants and animals grow by consuming food
both for adding matter
and energy for activity

Life forms

Procreation makes similar copies

passing on the design of bodies

defining species

Individuals in a species consume resources

food for matter and energy

Resources are limited

creating competition

Competition

Species compete for the same resources

for example, lions and hyenas for prey

While there is competition between the species

there is competition within too

The fastest, or the wiliest, foxes get the rabbits,

the fastest or the cleverest rabbits escape

Survival of the fittest!

Evolution

Species evolve continually.

A species is like the design of a life form.

Nature has evolved a process of creative adaptation

Sexual reproductions allows
inheritance of genes
from two parents

Paul Valery the French poet

Mixing up genes via sexual reproduction

"It takes two to invent anything.
The one makes up combinations,
the other chooses…"

Survival of the fittest via natural selection

An instance of Generate & Test

Genotypes and Phenotypes

Mixing up genes via sexual reproduction

- An organism's genotype is the set of genes in its DNA responsible for a particular trait.

Survival of the fittest via natural selection

- An organism's phenotype is the physical expression of those genes.

It is the physical entity that competes in the world.

Selection by attractiveness

In Nature the mixing up of genes *does not* happen at random.

Individuals *actively* choose mates.

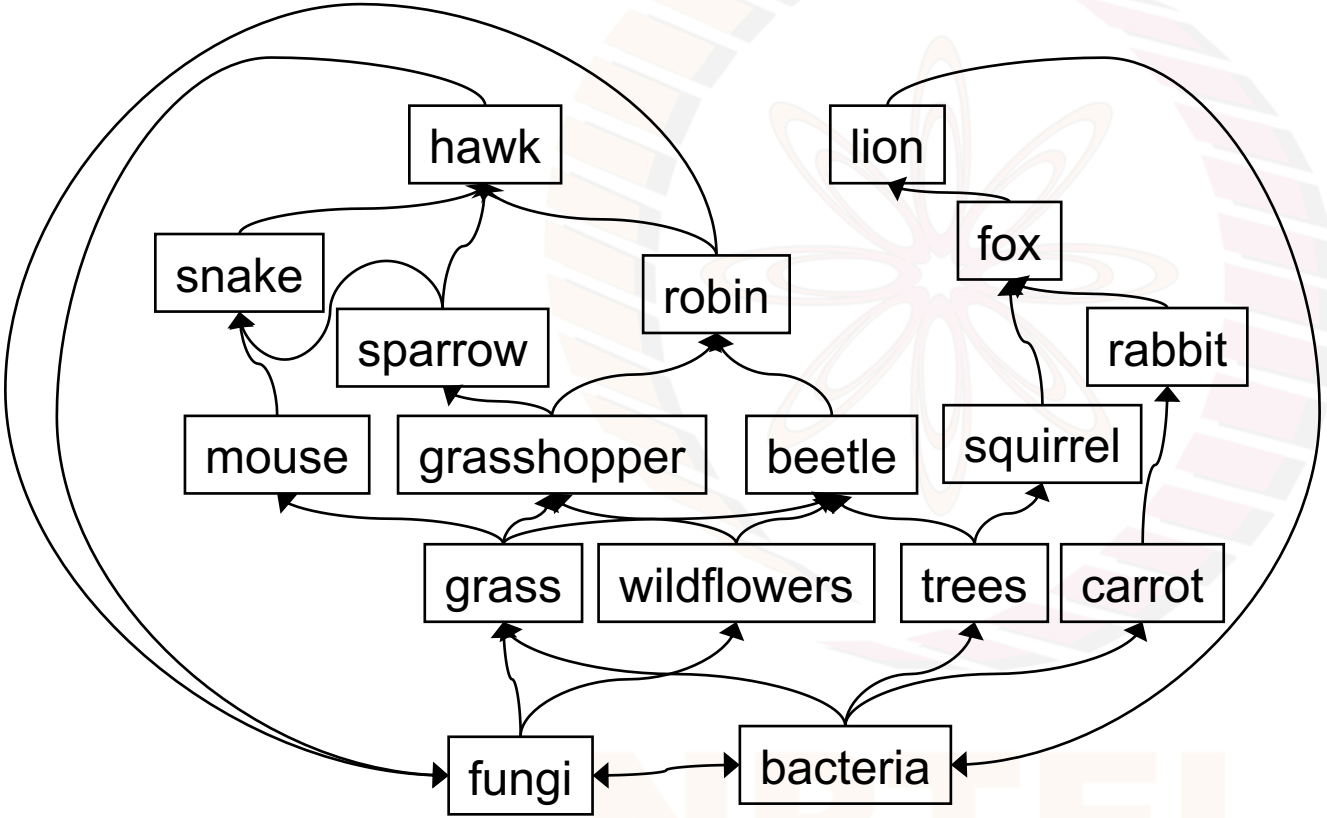
All life forms have evolved ways to attract the opposite sex.

- Most male birds put up a song and dance show.
- Humans are more complex.

Health, wealth, cleverness

are all part of the mating game.

An Ecosystem of Predators and Prey



A small fragment of the vast ecosystem. The natural world contains millions of species interacting with each other. Arrows depict a positive influence of the population of one species on another.

Evolution: Survival of the Fittest

French poet Paul Valéry: "*It takes two to invent anything. The one makes up combinations: the other chooses ... what is important to him in the mass of the things which the former has imparted to him*."

Genetic Algorithms

• The process of sexual reproduction experiments with the genotype by making up different combinations of genes inherited by the two parents.

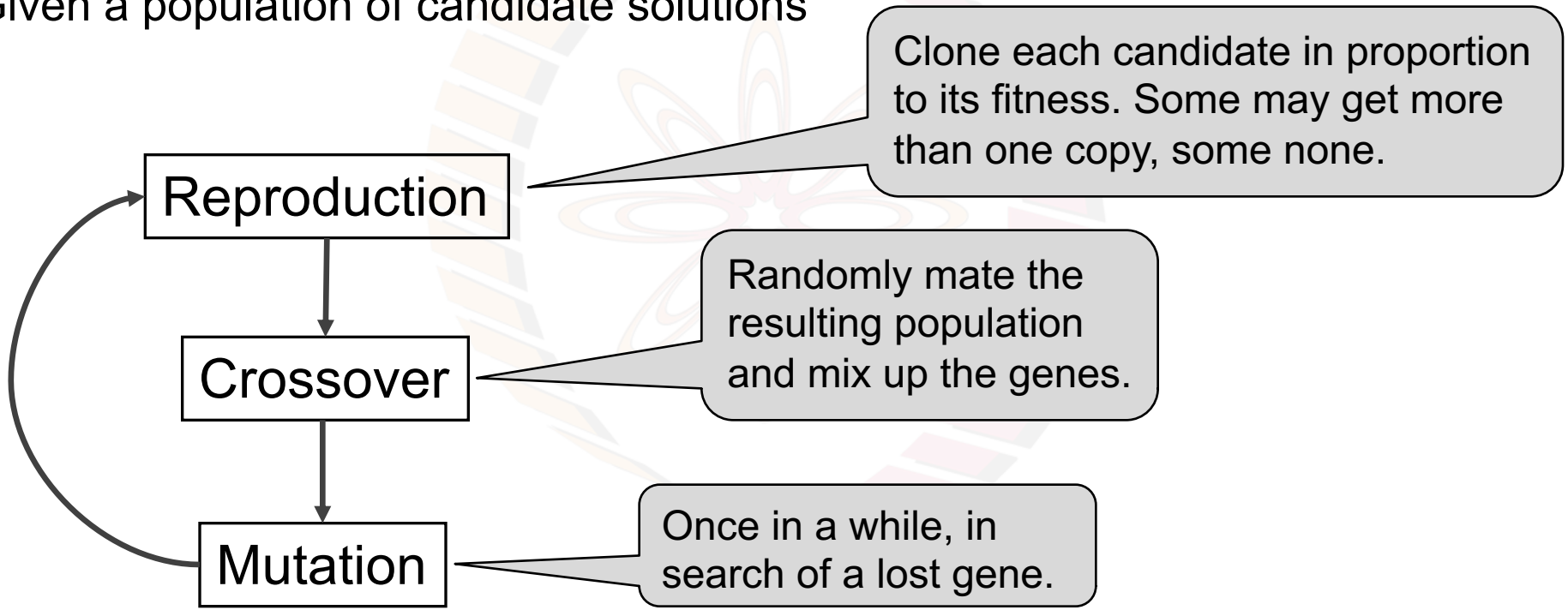
• The process of competition for survival of the phenotype (the physical entity) in the real world selects the best candidates and their genes propagate.

Genetic Algorithms (GAs)

- Devised by John Holland (1975), popularized by his student David Goldberg via his book "*Genetic Algorithms*".
- A class of methods for optimization problems, more generally known as Evolutionary Algorithms.
- Heuristic stochastic adaptive search algorithms
- Implemented on a population of candidates in the solution space
- Inspired by the process of natural selection
- A fitness function evaluates each candidate
- The fittest candidates get to mate and reproduce

Genetic Algorithms : Artificial Selection

Given a population of candidate solutions



Crossover operators

A single point crossover simply cuts the two parents at a randomly chosen point and recombines them to form two new solution strings.

$P_1 = X_1\ X_2\ X_3\ X_4 \mid X_5\ X_6\ X_7\ X_8$
$P_2 = Y_1\ Y_2\ Y_3\ Y_4 \mid Y_5\ Y_6\ Y_7\ Y_8$

$C_1 = X_1\ X_2\ X_3\ X_4\ Y_5\ Y_6\ Y_7\ Y_8$
$C_2 = Y_1\ Y_2\ Y_3\ Y_4\ X_5\ X_6\ X_7\ X_8$

For example, SAT

Any operator that mixes up the genes will do.
One can have multi-pont crossovers, for example,

$P_1 = X_1\ X_2\ X_3\ X_4\ X_5\ X_6\ X_7\ X_8$
$P_2 = Y_1\ Y_2\ Y_3\ Y_4\ Y_5\ Y_6\ Y_7\ Y_8$

$C_3 = X_1\ Y_2\ X_3\ Y_4\ X_5\ Y_6\ X_7\ Y_8$
$C_4 = Y_1\ X_2\ Y_3\ X_4\ Y_5\ X_6\ Y_7\ X_8$

Genetic Algorithms

```
GeneticAlgorithm()
    Initialize an initial population of candidate solutions p[1..n]
    repeat
        Calculate the fitness value of each member in p[1..n]
        selected[1..n] ← the new population obtained by picking n members
                         from p[1..n] with probability proportional to fitness
        Partition selected[1..n] into two halves, and randomly mate and
                                 crossover members to generate offspring[1..n]
        With a low probability mutate some members of offspring[1..n]
      Replace k weakest members of p[1..n] with the k strongest
                                 members of offspring[1..n]
    until some termination criteria
    return the best member of p[1..n]
```

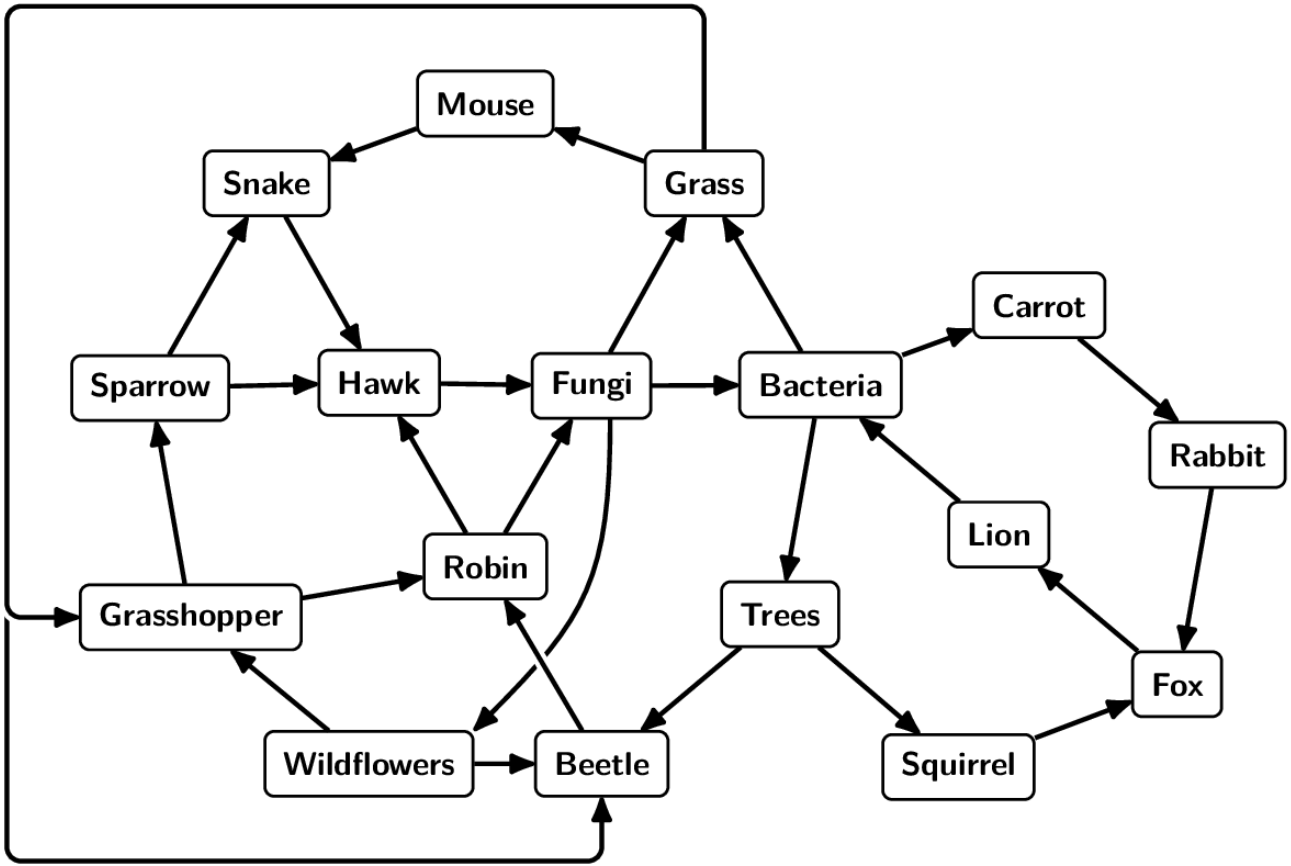
Genetic Algorithms

Survival of the Fittest

Competition

Evolution

An Ecosystem of Predators and Prey



A small fragment of the vast ecosystem. The natural world contains millions of species interacting with each other. Arrows depict a positive influence of the population of one species on another.

Evolution

Species evolve continually.

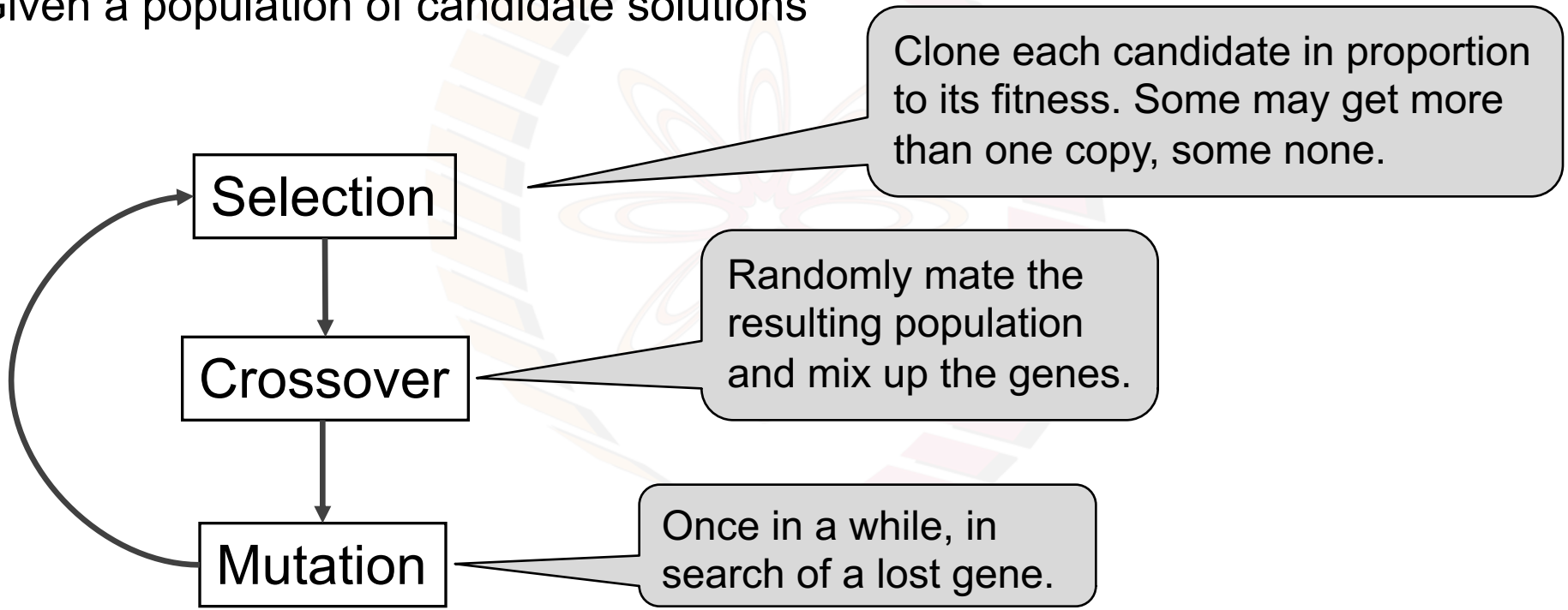
A species is like the design of a life form.

Nature has evolved a process of creative adaptation

In his famous book "*The Blind Watchmaker*" the British evolutionary biologist Richard Dawkins has emphatically argued that the "*ratchet mechanism*" in nature (remember – *Things that persist, persist…*) has been responsible for the evolution of complex life forms, rather than a supernatural creator….

Genetic Algorithms : Artificial Selection

Given a population of candidate solutions



The Algorithm

GENETIC-ALGORITHM()

```
1  P ← create N candidate solutions        ▷ initial population
2  repeat
3      compute fitness value for each member of P
4      S ← with probability proportional to fitness value,
           randomly select N members from P
5      offspring ← partition S into two halves, and randomly mate
                   and crossover members to generate N offsprings
6      with a low probability mutate some offsprings
7      replace k weakest members of P with k strongest offsprings
8  until some termination criteria
9  return the best member of P
```

Complexity of SAT

$F = (a \vee \neg b) \wedge (\neg a \vee c) \wedge (\neg c \vee f) \wedge (\neg e \vee f) \wedge (\neg a \vee b) \wedge (a \vee \neg d)$

2SAT - problems where each clause has at most two literals
- known to be in P

$F = (a \vee \neg b \vee e) \wedge (\neg a \vee b \vee c) \wedge (\neg c \vee d \vee f) \wedge (d \vee \neg e \vee f) \wedge (\neg a \vee b \vee d) \wedge (a \vee \neg d \vee f)$

3SAT - problems where each clause has at most three literals
- known to be NP-complete

kSAT - $k > 3$
- could be much harder

A SAT problem with 6 variables

$F = (a \vee \neg b \vee e) \wedge (\neg a \vee b \vee c) \wedge (\neg c \vee d \vee f) \wedge (d \vee \neg e \vee f) \wedge (\neg a \vee b \vee d) \wedge (a \vee \neg d \vee f)$

h = number of satisfied clauses

Single Point Crossover

P_1 010110 $0 + 1 + 1 + 1 + 1 + 0 = 4$

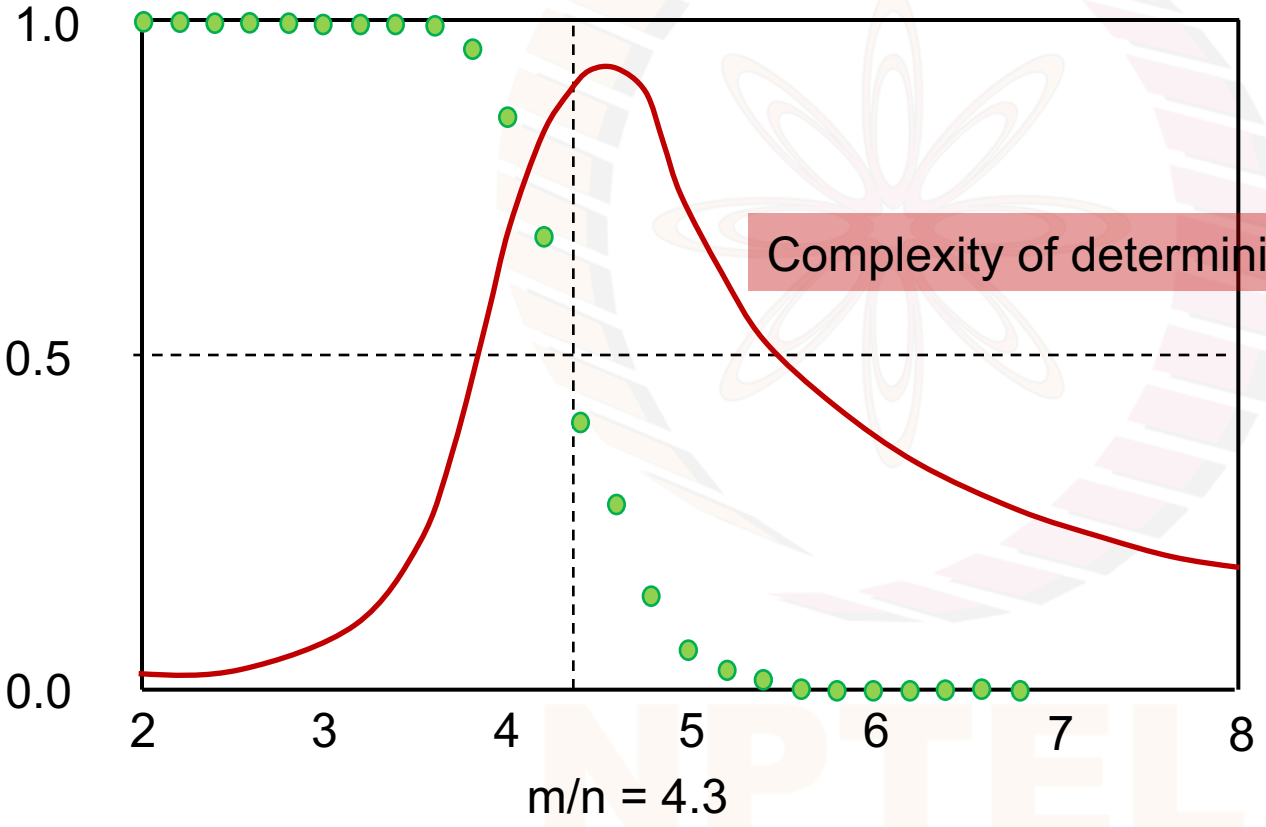
P_2 111010 $1 + 1 + 0 + 0 + 1 + 1 = 4$

C_1 111110 $1 + 1 + 1 + 1 + 1 + 1 = 6$

C_2 010010 $0 + 1 + 1 + 0 + 1 + 1 = 4$

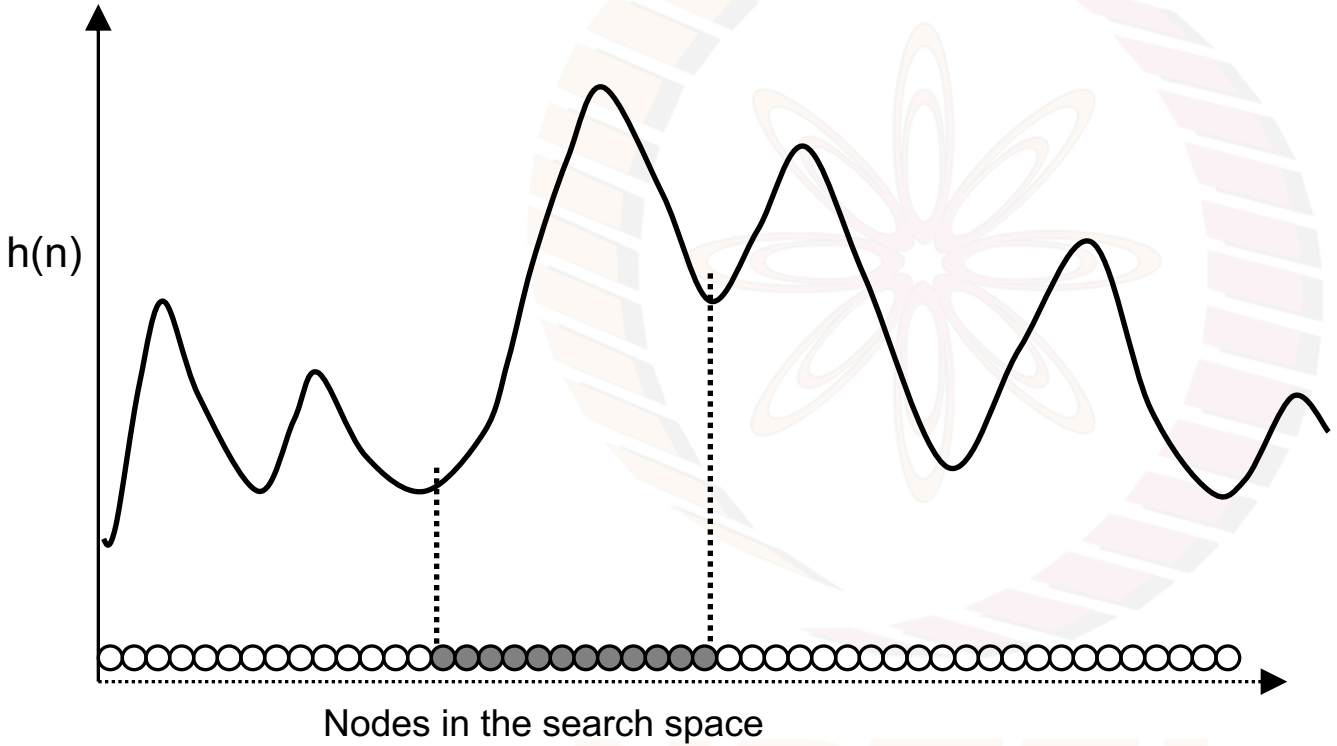
Probability of 3SAT being satisfiable

m clauses
n variables



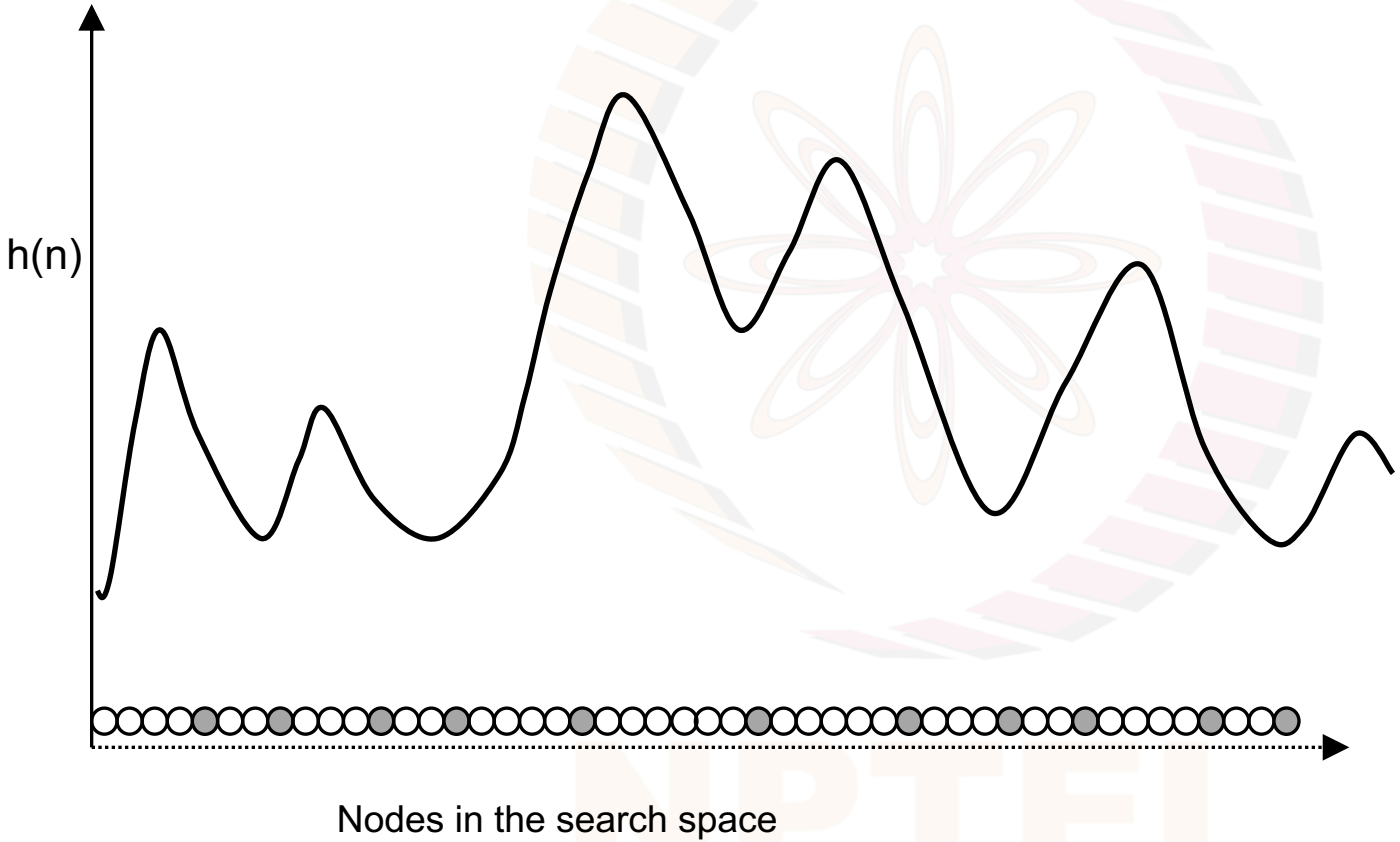
Iterated Hill Climbing

Recap



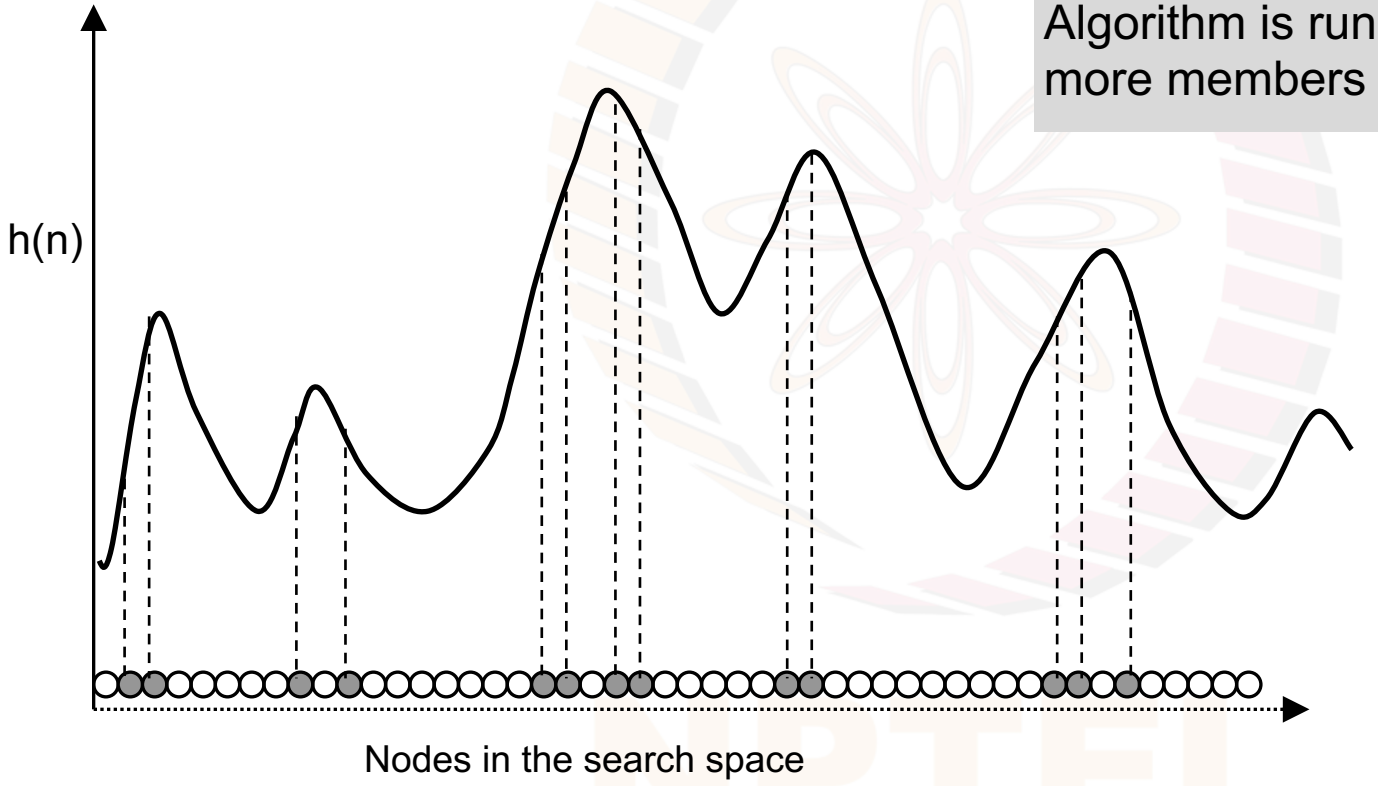
A random starting point from any of the shaded nodes would lead to the global maximum.

Genetic Algorithms: An initial population



GAs: The evolved population

The Initial population may be randomly distributed, but as Genetic Algorithm is run the population has more members around the peaks.



A Tiny Example

From: David E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning , Addison-Wesley, 1989

Let the chromosome for a tiny problem be a 5-bit string

Let the fitness function *f(x)* be the *square* of the 5-bit binary number

Let the population contain 4 candidates

We illustrate the algorithm with a single point crossover

A Tiny Example: Selection

Initial	Binary	f(x)	Prob.	Expected	Actual	Selected
01101	13	169	0.14	0.58	1	01101
11000	24	576	0.49	1.97	2	11000
01000	8	64	0.06	0.22	0	11000
10011	19	361	0.31	1.23	1	10011

Total 1170
Avg. 293

A Tiny Example: Single Point Crossover

Selected	Crossed-over	Binary	f(x)
01101	01100	12	144
11000	11001	25	625
crossover →			
11000	11011	27	729
10011	10000	16	256

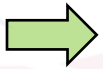
Fitter population

Total 1754
Avg. 493

A Tiny Example: Cycle 2

Crossed-over	Binary	f(x)	Prob.	Expected	Actual	Selected
01100	12	144	0.08	0.33	0	11001
11001	25	625	0.36	1.42	2	11001
11011	27	729	0.42	1.66	2	11011
10000	16	256	0.16	0.58	0	11011

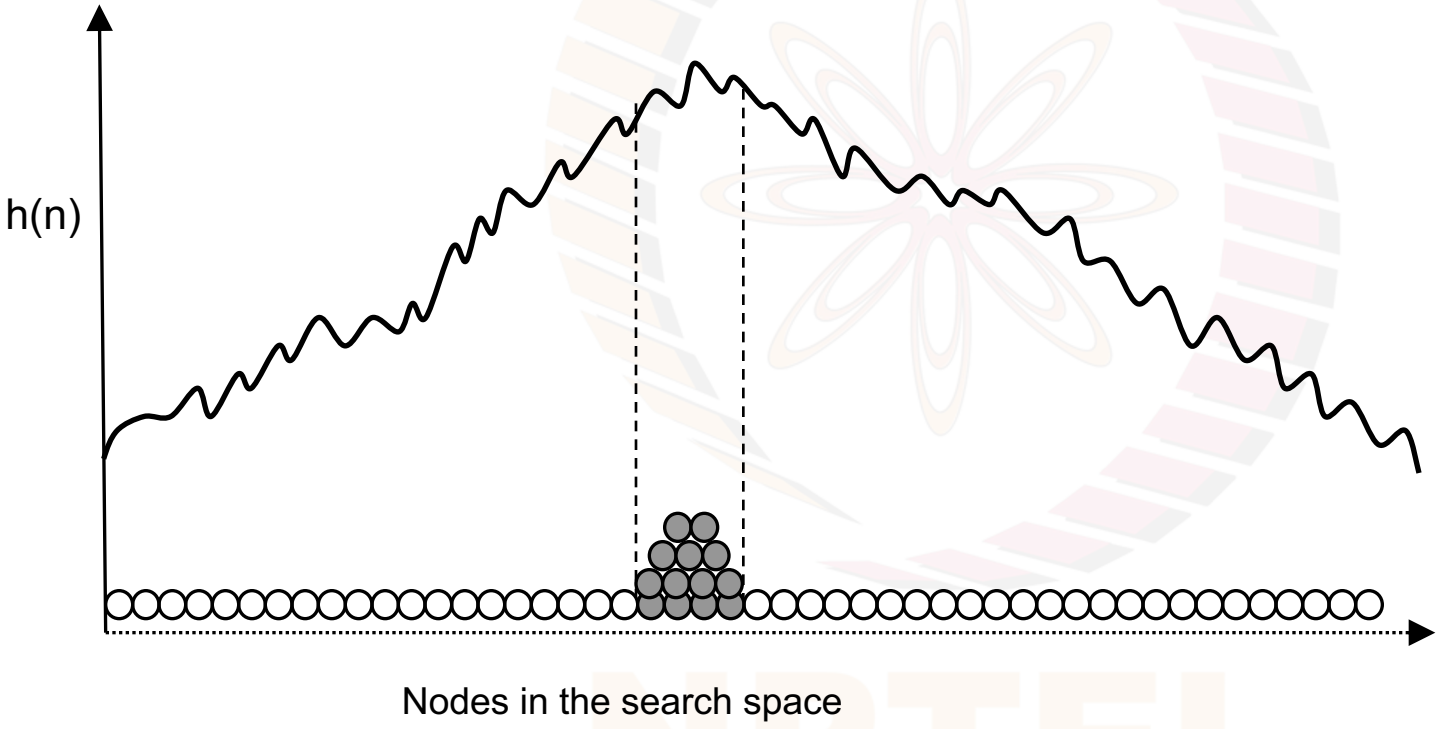
Total 1754
Avg. 493



Total 2708
Avg. 677

A fitter population BUT less diverse

The population may become less diversified



GAs: practical matters

Candidates are encoded as chromosomes

Genes are like components in the candidates

GAs work by inheriting components from two parents

A careful encoding is needed
- we look at Gas for TSPs next

A large diverse population
is critical to for performance

Solving TSP using Genetic Algorithms

GAs for TSP

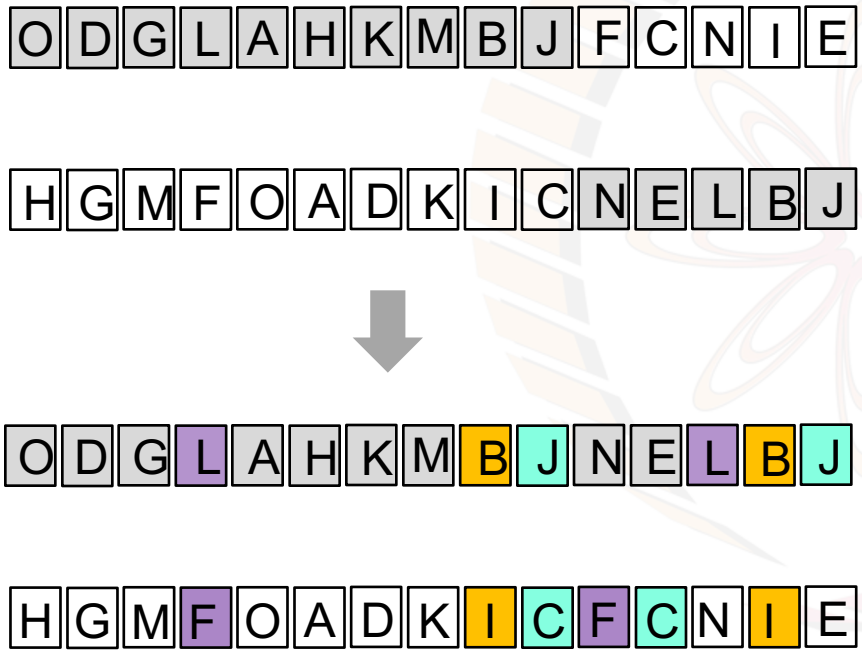
Genetic Algorithms can be used for the TSP problem as follows –

Create a population of candidate TSP solutions. Let the fitness function be the cost of the tour. It is a *minimization problem*.

In the *Path Representation* the tour is represented by a permutation of the cities, with the assumption that one returns from the last city in the permutation to the first.

Selection:	Clone each tour in proportion to fitness. The cheapest tours are the fittest.
Crossover:	Randomly pair the resulting population and perform crossover.
Mutation:	Randomly permute a tour once in a while

TSP: Single point crossover does not work

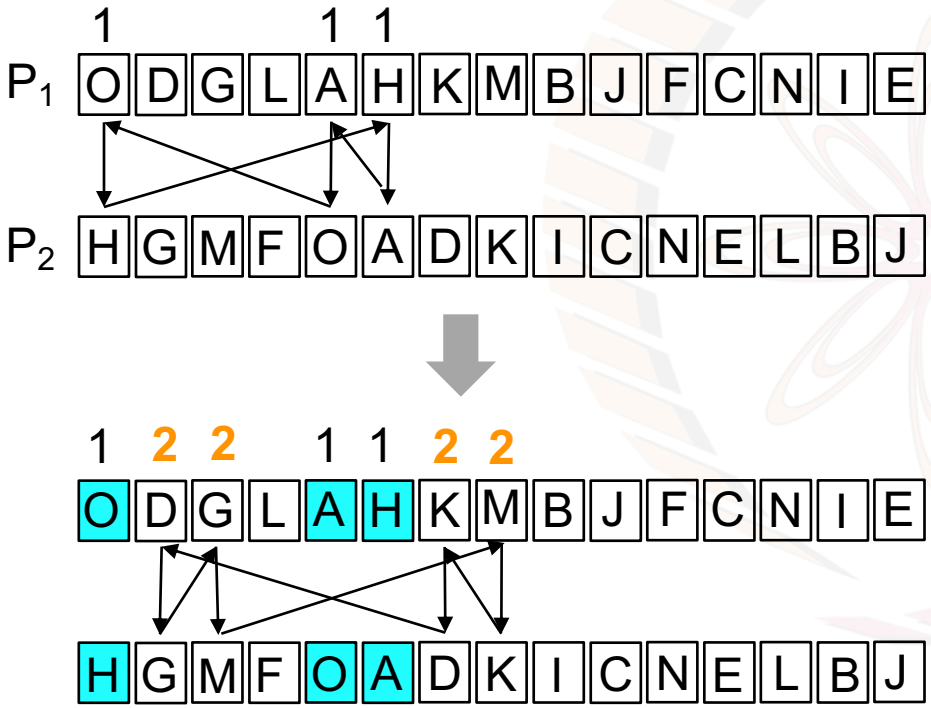


Both offspring are *not* valid tours

Crossover operators for TSP

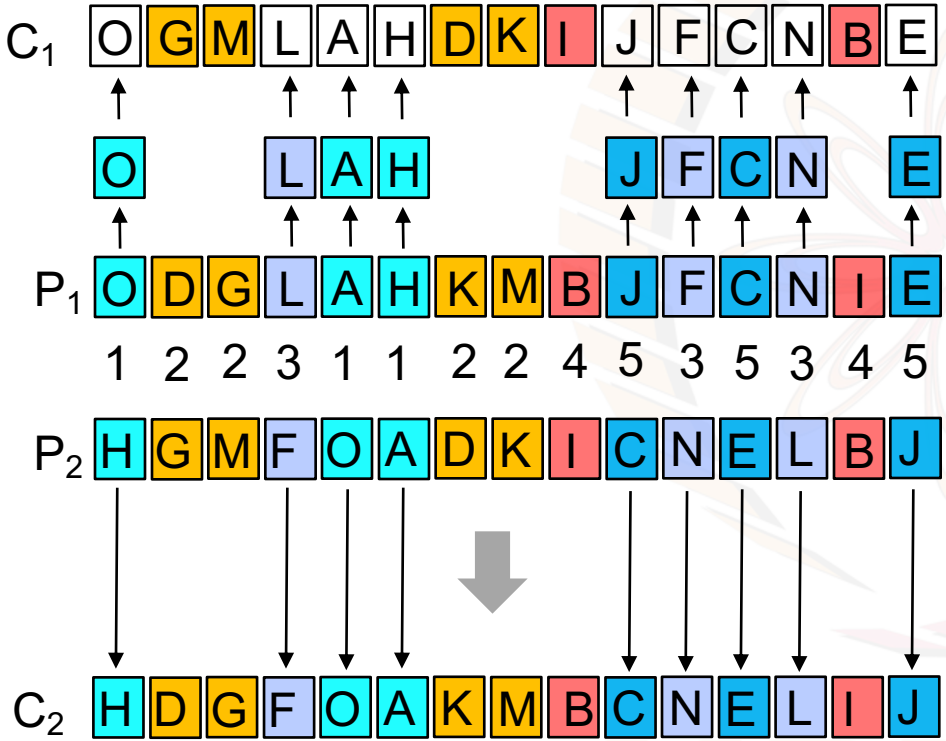
We look at some crossover operators used for the TSP

TSP: Cycle Crossover



Step 1: Identify cycles

TSP: Cycle Crossover



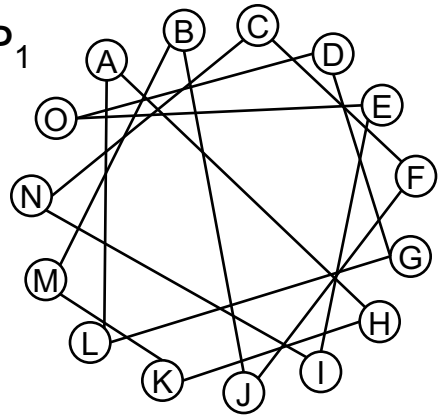
C_1 gets even numbered cycles from P_2

C_1 gets odd numbered cycles from P_1

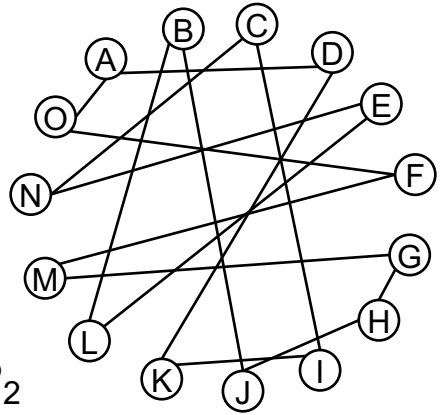
C_2 gets odd numbered cycles from P_2 and even from P_1

TSP: Cycle Crossover

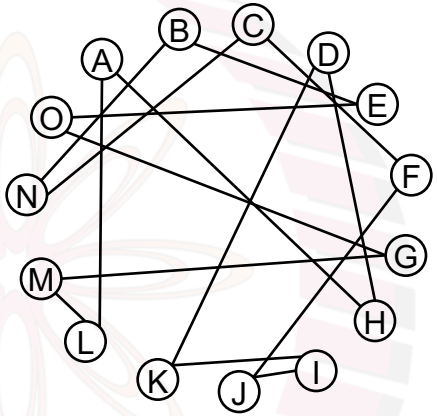
P_1



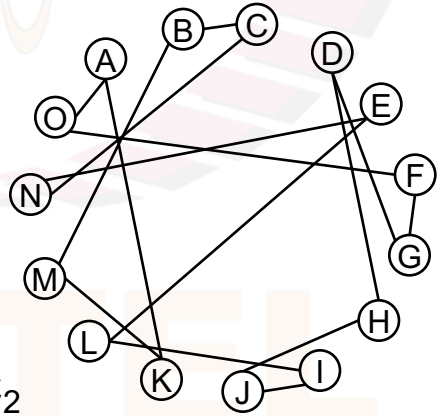
P_2



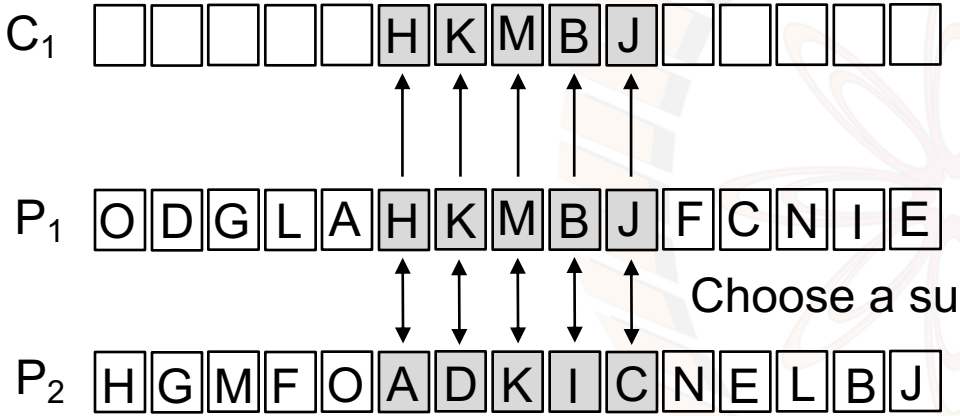
C_1



C_2



TSP: Partially Mapped Crossover (PMX)

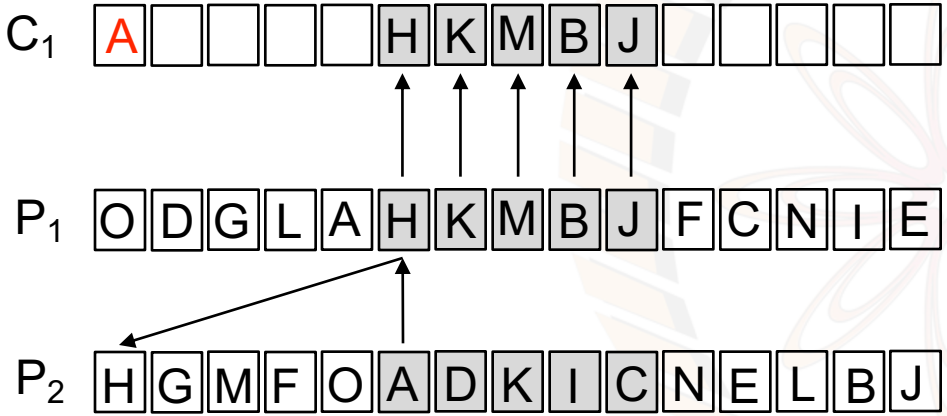


Copy subtour from P_1 into C_1

Choose a subtour and establish a mapping

Would like to copy remaining cities from P_2
but
the locations for cities A, D, I, C are occupied
by cities H, K, B,J respectively

TSP: Partially Mapped Crossover (PMX)

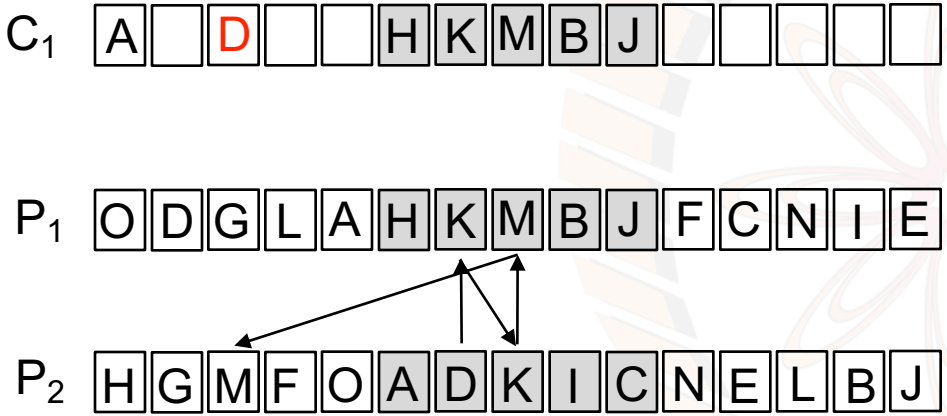


Copy subtour from P_1 into C_1

Where should city A be in C_1?

Follow the partial map…

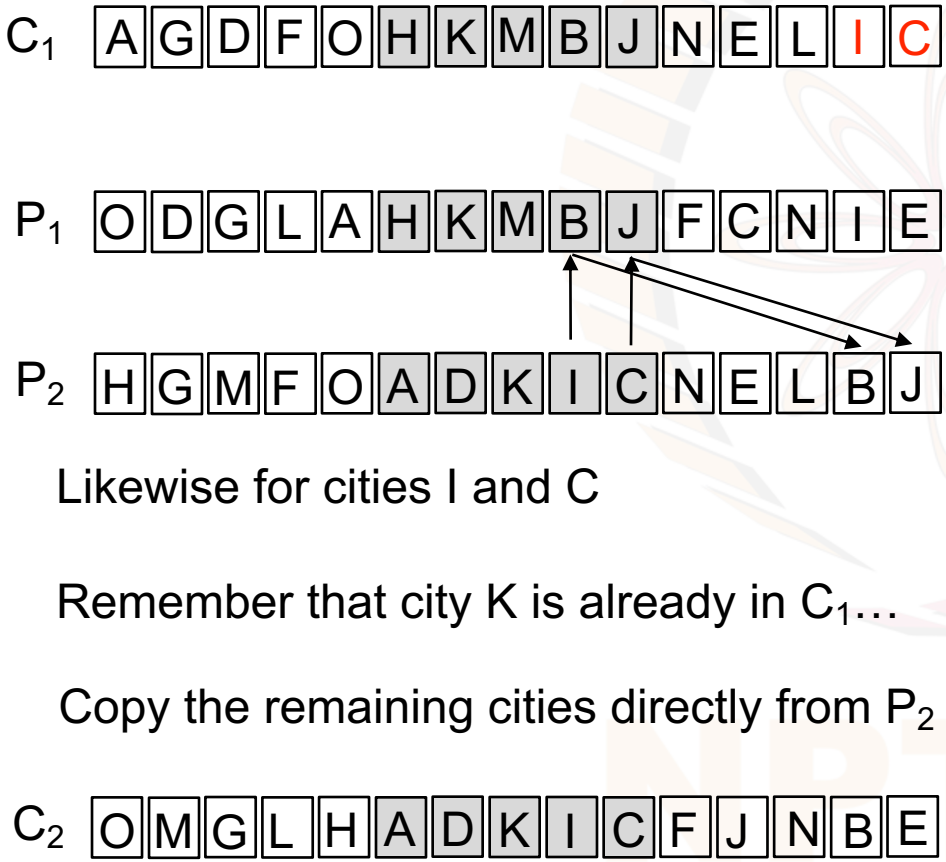
TSP: Partially Mapped Crossover (PMX)



Where should city D be in C_1?

Follow the partial map…

TSP: Partially Mapped Crossover (PMX)



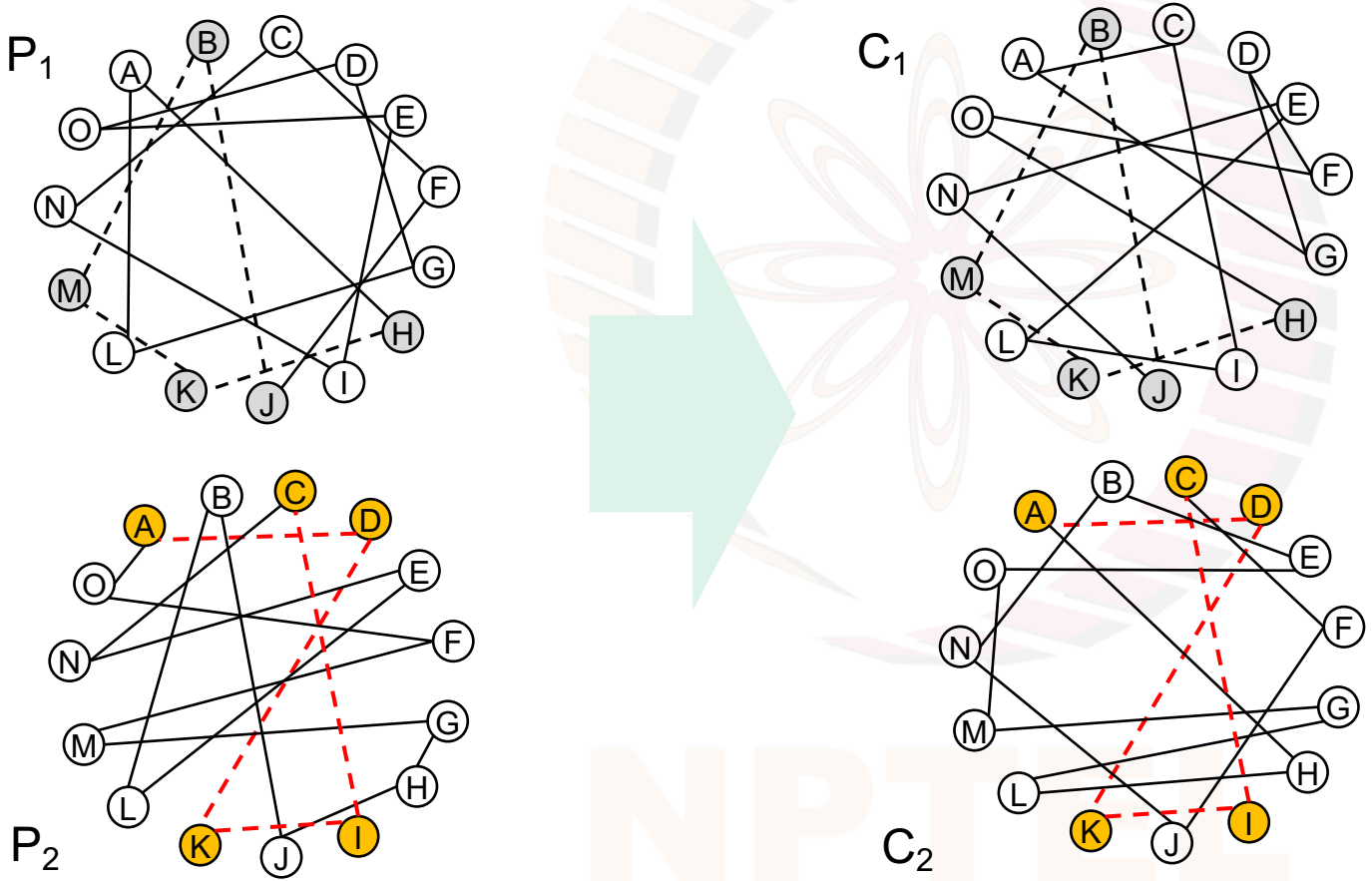
The second child C_2 is constructed in a similar manner, first copying the subtour from P_2

Likewise for cities I and C

Remember that city K is already in C_1…

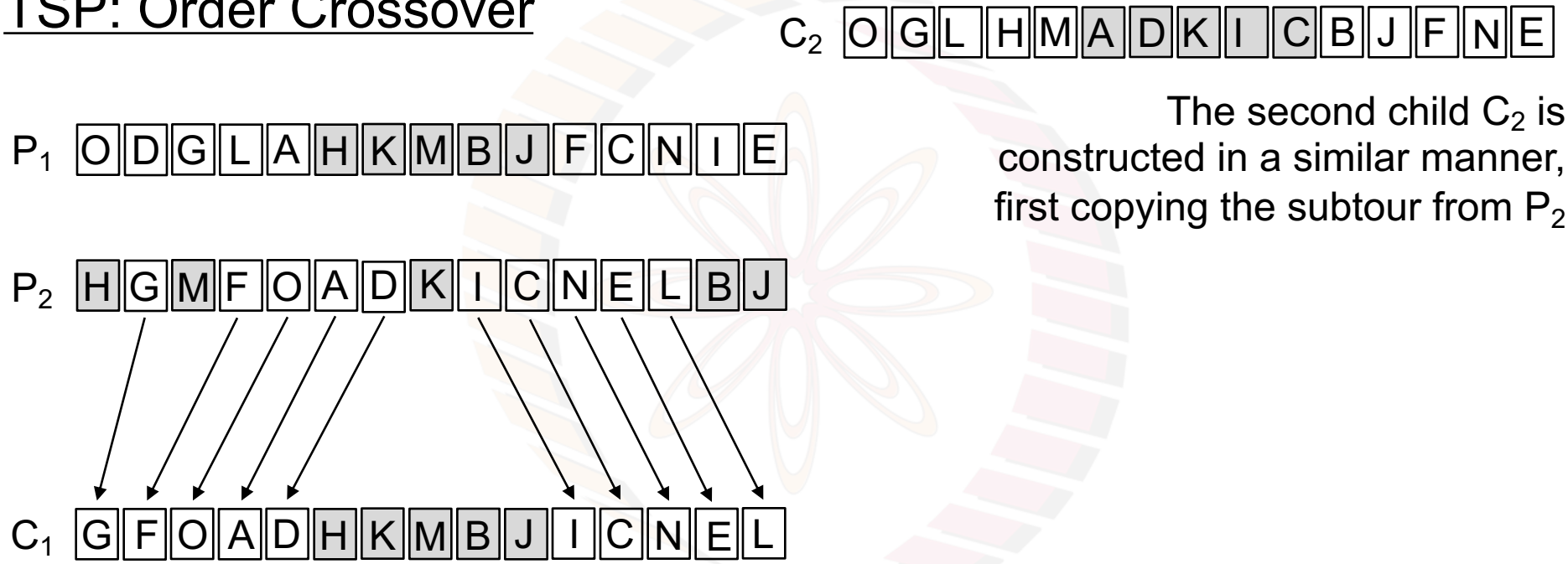
Copy the remaining cities directly from P_2

TSP: PMX



Dashed edges show copied subtour

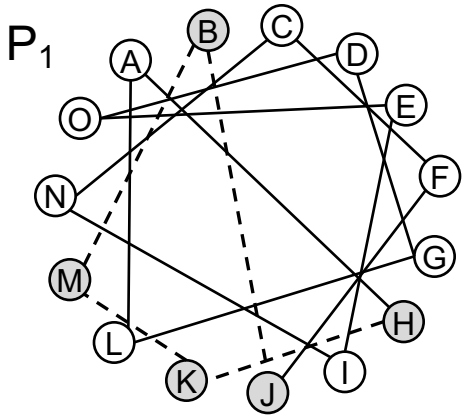
TSP: Order Crossover

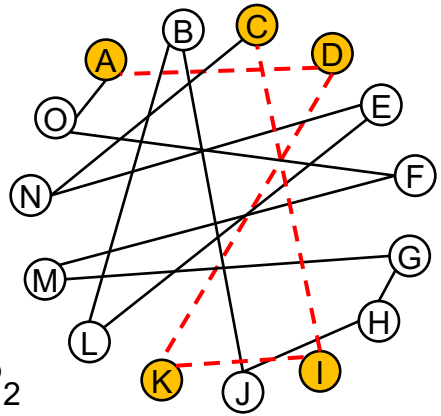


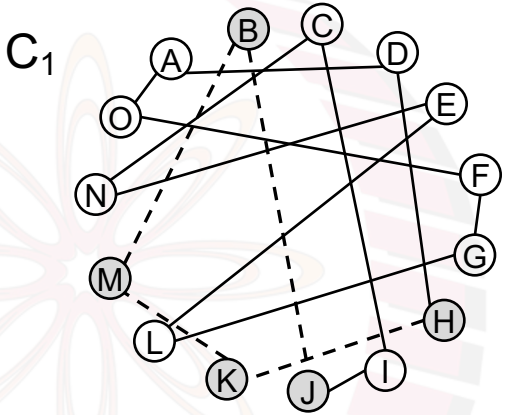
The second child C_2 is constructed in a similar manner, first copying the subtour from P_2

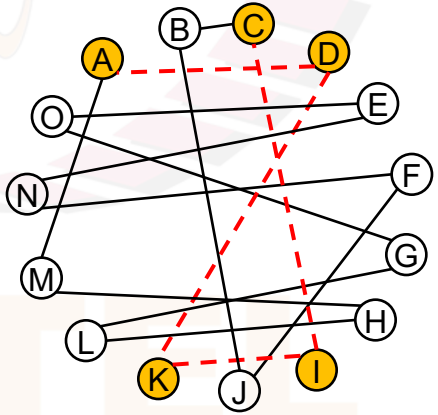
Copy a subtour from P_1 into C_1 and the remaining from P_2 in the order they occur in P_2.

TSP: Order Crossover



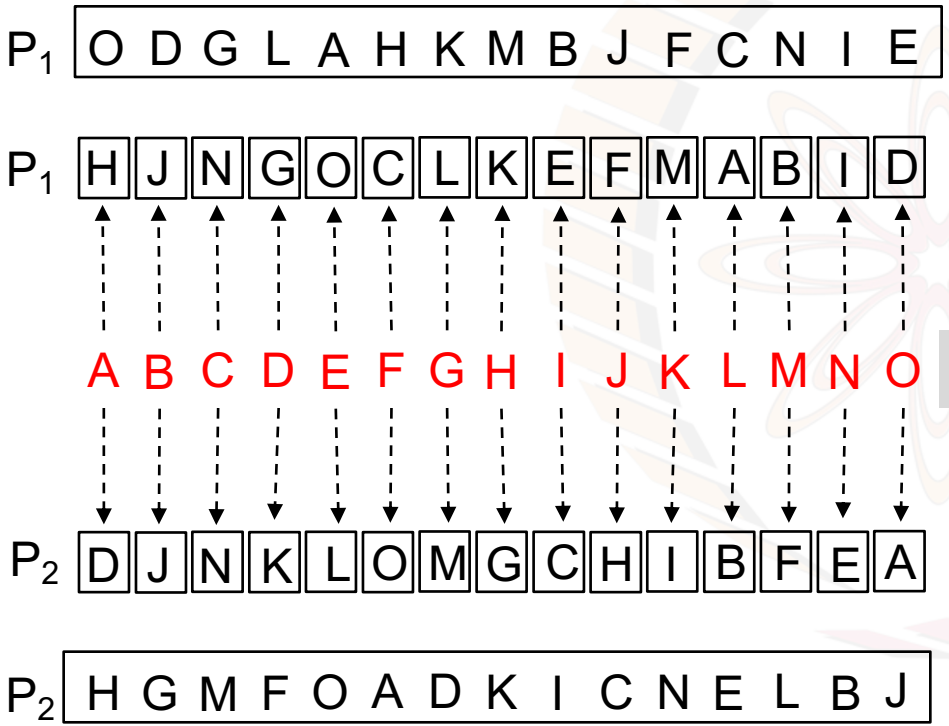






Dashed edges show copied subtour

TSP: Adjaceny Representation

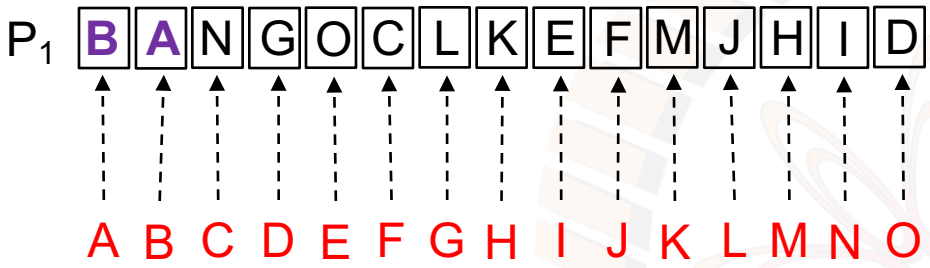


The Adjacency Representation from P_2

In *Adjacency Representation* the cities are arranged based on *where they come from* w.r.t. the INDEX.

For example A→H, B→ J, …

TSP: Adjaceny Representation



This permutation cannot be a tour because it says
- go from A to B
- go from B to A
…….which is a cycle

Does this mean the Adjacency Representation cannot represent the entire candidate space?

No!

In Path Representation every rotation of a permutation represents the same tour

Q: How many representations does a tour have in Adjacency representation?

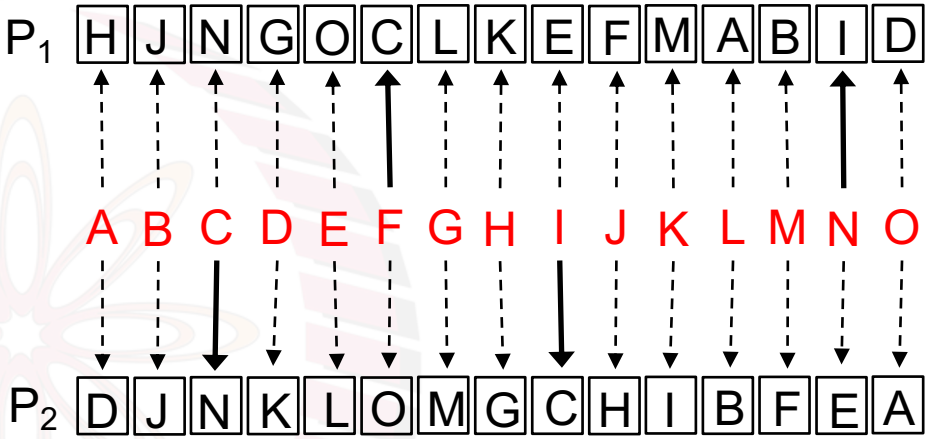
Adjacency Representation: Crossover Operators

- Alternating Edges Crossover
  - Construct a child as follows
  - From a given city A choose the next city B from P_1
  - From the city B choose the next city from P_2
  - … and so on

- Heuristic Crossover
  - For each city choose the from that parent (P_1 or P_2) which is closer

Adjacency representation facilitates these choices

TSP: Alternating Edges - careful



C_1 : Let's say we start with city F

from P_1: C

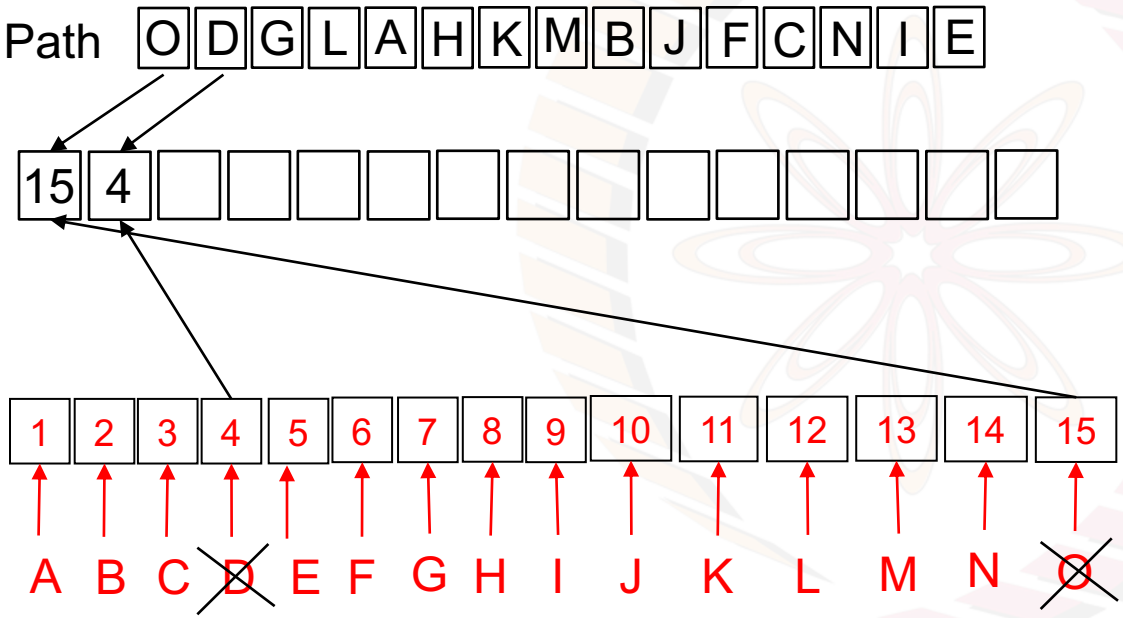
from P_2: N

from P_1: I

from P_2: **C**…. Oops!

Choose random next city…

TSP: Ordinal Representation



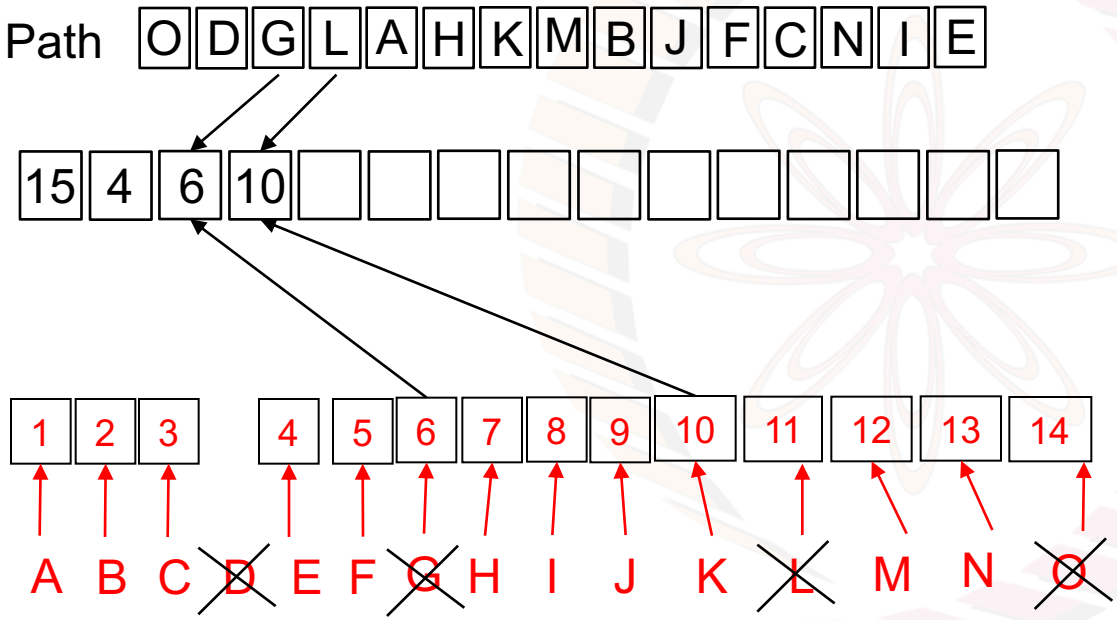
Current numeric index of remaining cities

Replace the name of the city
in *Path Representation*
one by one
by *its current numeric index*

First replace city O and remove it from the current index

Next replace D and remove it from current index

TSP: Ordinal Representation



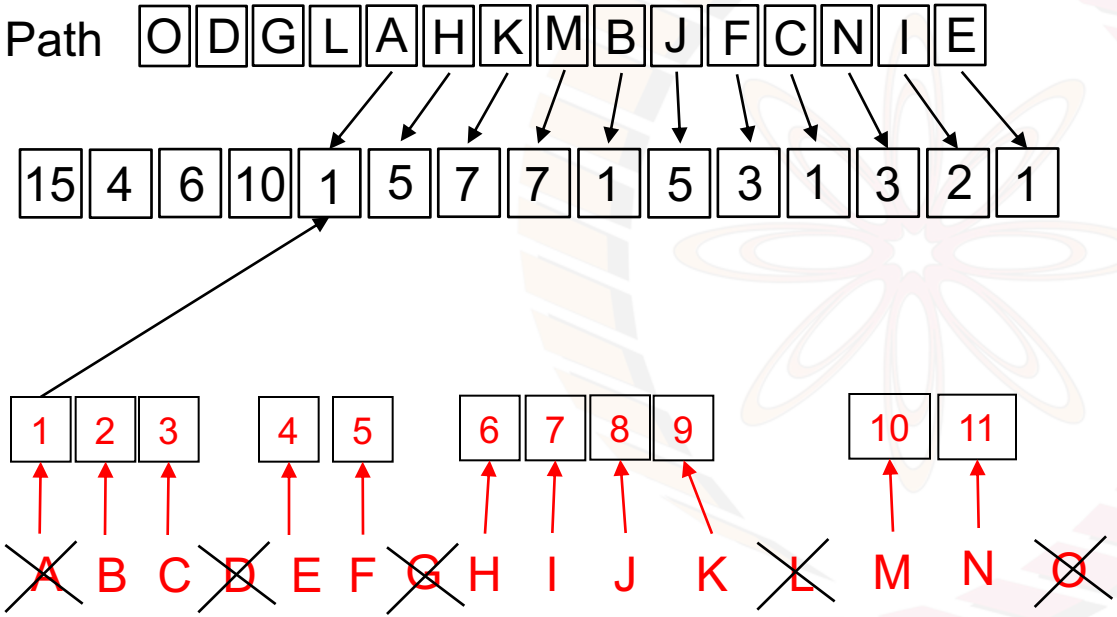
Current numeric index of remaining cities

Replace the name of the city
in *Path Representation*
one by one
by *its current numeric index*

Observe that for the next city G the current numeric index is 6

And after G is removed L will become 10

TSP: Ordinal Representation



Current numeric index of remaining cities

Replace the name of the city
in *Path Representation*
one by one
by *its current numeric index*

Next is A with the index 1, and indices of all will decrement

And so on….(not showing the contracting index after city A)

TSP: Ordinal Representation

The advantage of using the Ordinal Representation is that the Single Point Crossover produces valid offspring.

Try it out!

Next

Another nature inspired optimization method

Emergent Systems

Complex Behaviour from Simple Elements

Complex Dynamical Systems

Emergent Systems

Collections of simple entities *organize* themselves and *a larger more sophisticated entity emerges*. The behaviour of this complex system is a property that emerges from interactions amongst its components.

- An ant colony is like a living organism that can find food very quickly
- The brain is made up of billions of comparatively simple neurons
- A flock of birds moves in synchrony
- Elaborate termite mounds
- Stock market behaviour
- Formation of sand dunes
- Hurricanes and cyclones

Conway's Game of Life

John Conway, 1970

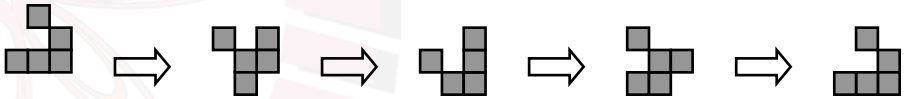
The Game of Life is a Cellular Automaton in which cells are *alive* or *dead*. Each cell obeys the following rules to decide its fate in the next time step.

Cell state	Number of *alive* neighbours	New cell state	Explanation
alive	< 2	dead	lonely
alive	2 or 3	alive	stable
alive	> 3	dead	overcrowded
dead	3	alive	resurrection

From these simple rules
many stable and persistent patterns emerge!

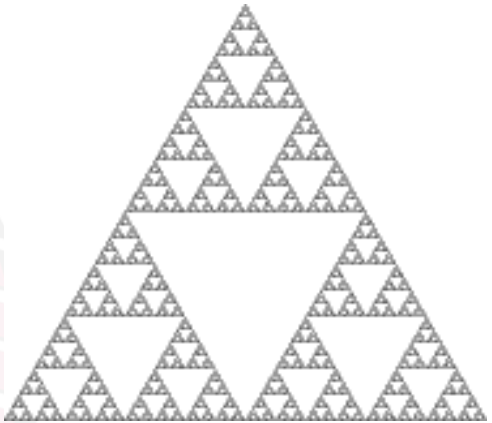
Illusion of Movement

Cell state	Number of *alive* neighbours	New cell state	Explanation
alive	< 2	dead	lonely
alive	2 or 3	alive	stable
alive	> 3	dead	overcrowded
dead	3	alive	resurrection



Gosper's Glider Gun – illusion of movement of an emergent entity.
Source: http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

Chaos and Fractals



A fractal is a never-ending pattern. Fractals are infinitely complex patterns that are *self-similar across different scales*.

They are created by repeating a simple process over and over in an ongoing feedback loop. Driven by recursion, fractals are images of dynamic systems – the pictures of Chaos.

Fractal patterns are extremely familiar, since nature is full of fractals. For instance: trees, rivers, coastlines, mountains, clouds, seashells, hurricanes, etc.

- Fractal Foundation
https://fractalfoundation.org/resources/what-are-fractals/

Sierpinski Triangle

The *Sierpinski triangle* is a fractal described in 1915 by Waclaw Sierpinski. It is a self similar structure that occurs at different levels of iterations, or magnifications.

“The most complex object in the known universe”

- Christof Koch, Allen Institute for Brain Science

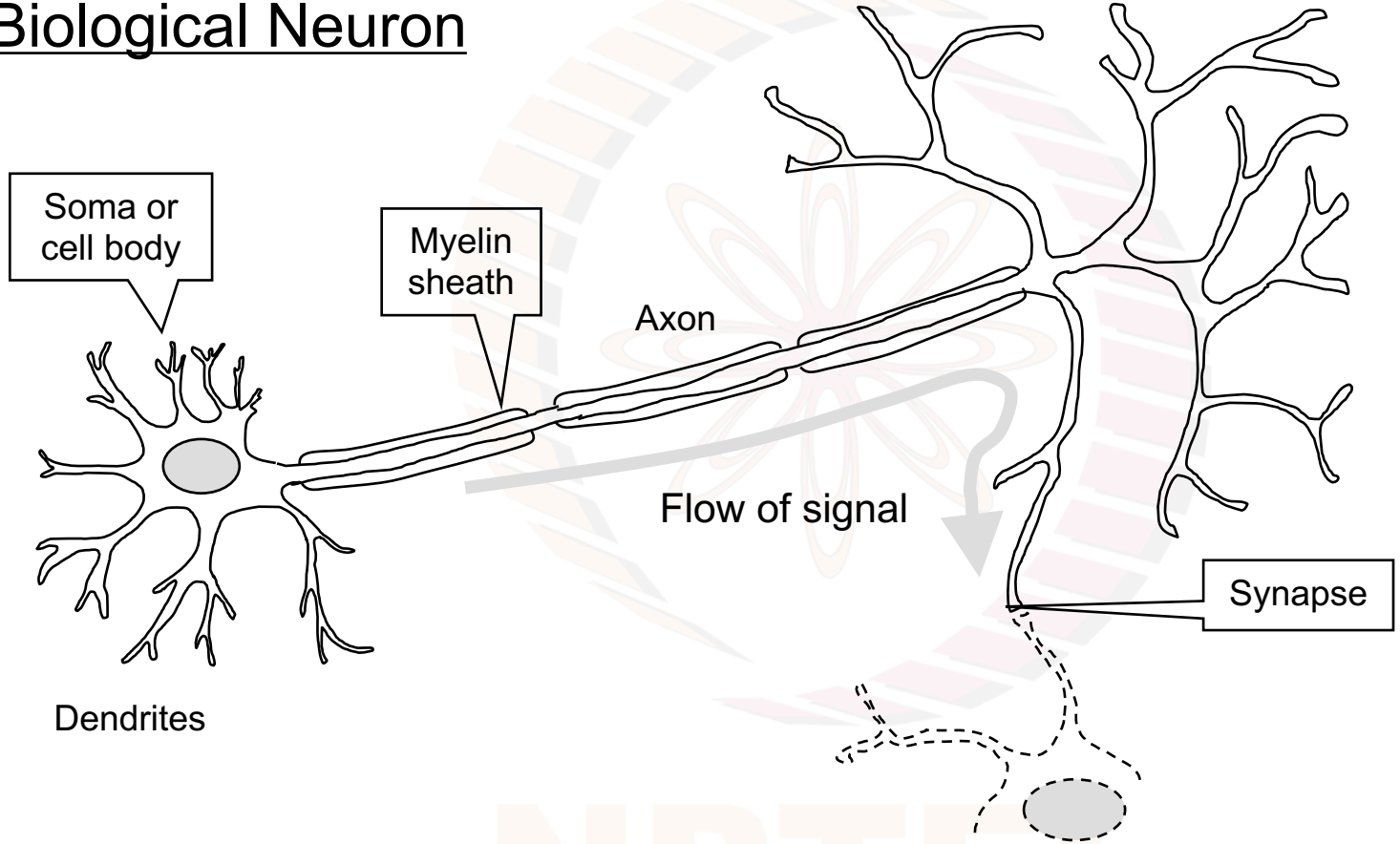
The brain is the most complex organ in the human body. It produces our every thought, action, memory, feeling and experience of the world. This jelly-like mass of tissue, weighing in at around 1.4 kilograms, contains a staggering one hundred billion nerve cells, or neurons.

The complexity of the connectivity between these cells is mind-boggling. Each neuron can make contact with thousands or even tens of thousands of others.

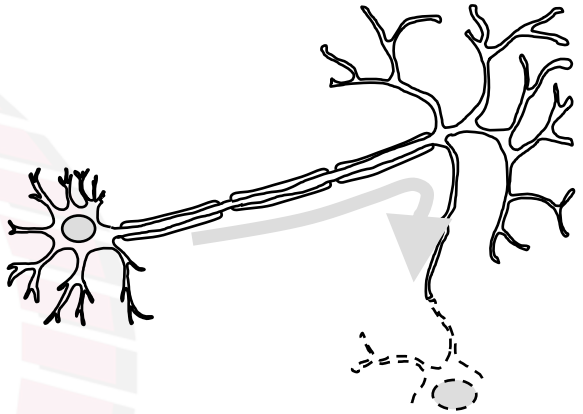
Our brains form a million new connections for every second of our lives. The pattern and strength of the connections is constantly changing and no two brains are alike.

- The New Scientist, 4 Sept 2006

A Biological Neuron



The Biological Neuron



A biological neuron from the brain receives several inputs via its dendrites and sends a signal down its axon.

The axon branches out as well and connects to dendrites of other neurons via synapses, which transmit the signal chemically to the other neurons.

The shaded portion of the soma is the nucleus of the cell.

The Human Brain: Massive Connectivity

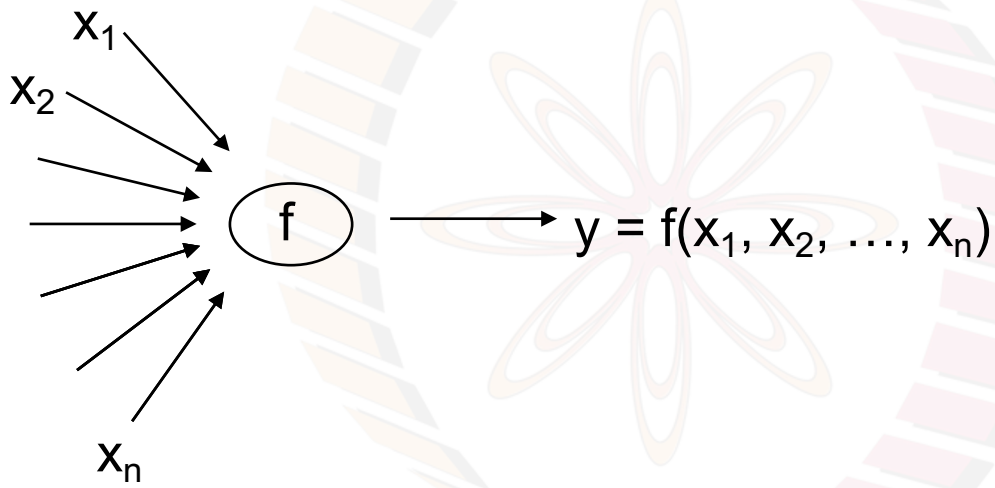
We each have something approaching 100 billion nerve cells – neurons – in the human brain (more than the number of stars in the Milky Way).

Each of them can be connected directly with maybe 10,000 others, totalling some 100 trillion nerve connections.

If each neuron of a single human brain were laid end to end they could be wrapped around the Earth twice over.

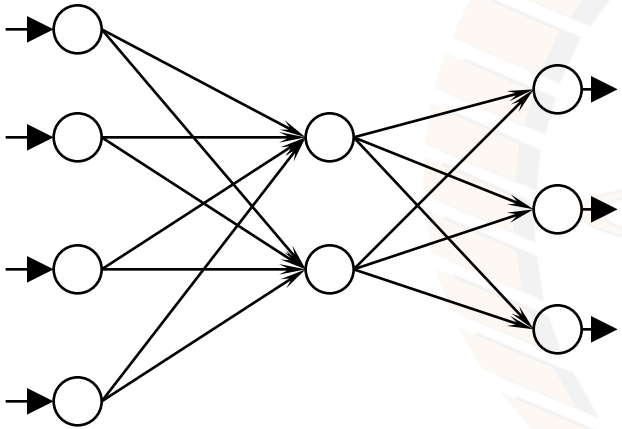
The Independent – Editorial, Wednesday 2 April 2014

A computational neuron



The neuron is a simple device that computes some function of all the inputs it receives from the other neurons.

Artificial Neural Networks (ANNs)



Input layer | Hidden layer | Output layer

The ANN learns the function through a process of *training*,

in which a large number of *labeled patterns* are shown as input,

the ANN *generates* an output,

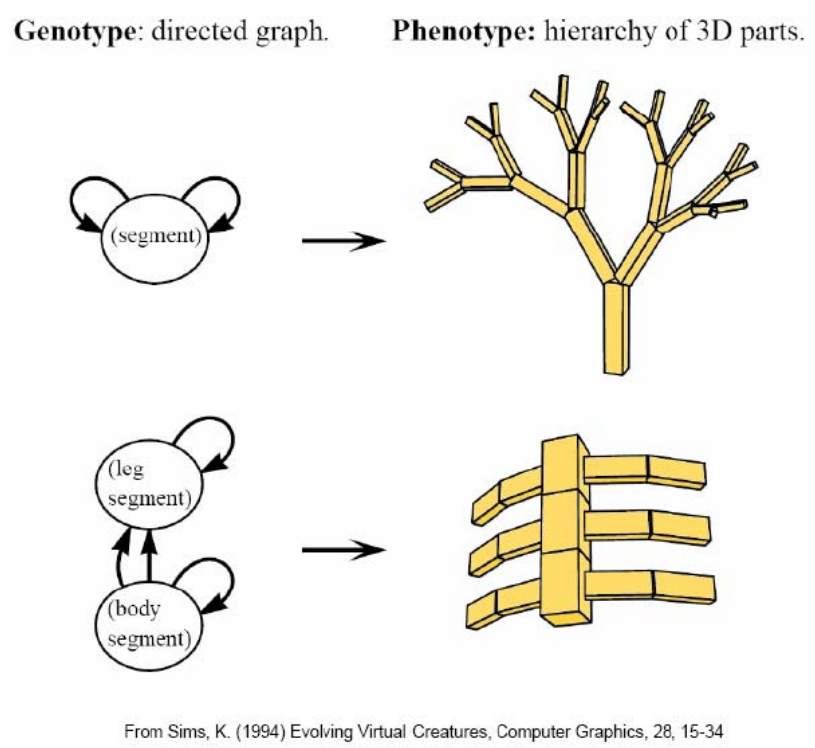
and the *BackProp* algorithm *propagates* the error back to *tune* the weights of the connections.

A Feedforward Artificial Neural Network

encapsulates a function

where the output is a function of the input

Karl Sims: Evolved Creatures



Two simple recursive genotypes and their corresponding phenotypes.

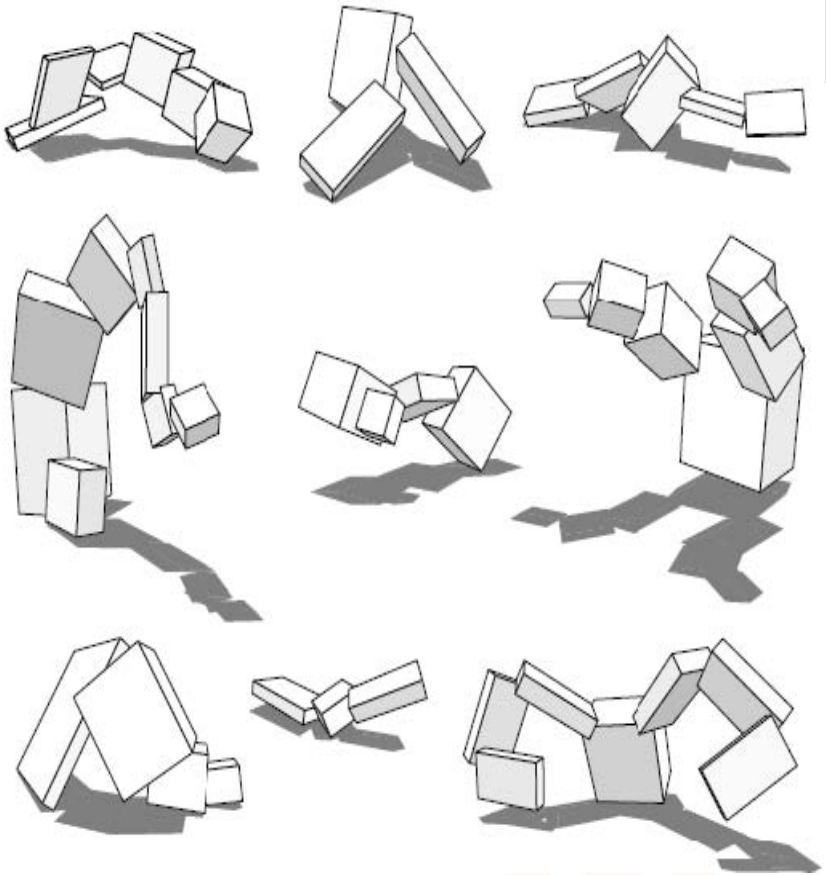
The genotype is a directed graph, and the phenotype a hierarchy of 3-D parts.

Figure from [Sims 1994]

Population size: Hundreds

Task: Locomotion

Virtual Evolved Creatures



See also: http://www.karlsims.com/evolved-virtual-creatures.html

http://www.youtube.com/watch?v=JBgG_VSP7f8

http://www.youtube.com/watch?v=GS18h-_h6IM

Different designs evolved for walking creatures. - Figure from [Sims 1994]

See also the documentary
- The Secret Life of Chaos

Next

Inspiration from Ants

Cooperation

Biosemiotics

Road signs for example

Semiotics

1.the study of signs and symbols and their use or interpretation.

Origin late 19th century: from Greek *sēmeiotikos* ‘of signs’, from *sēmeioun* ‘interpret as a sign’.

Biosemiotics (from the Greek βίος *bios*, "life" and σημειωτικός *sēmeiōtikos*, "observant of signs") is a field of semiotics and biology that studies the prelinguistic meaning-making, or production and interpretation of signs and codes in the biological realm.

Animals frequently mark their territory for example

Ant Colonies

Ants live in complex social colonies, with the queen being the leader and the workers foraging for food and protecting their home.

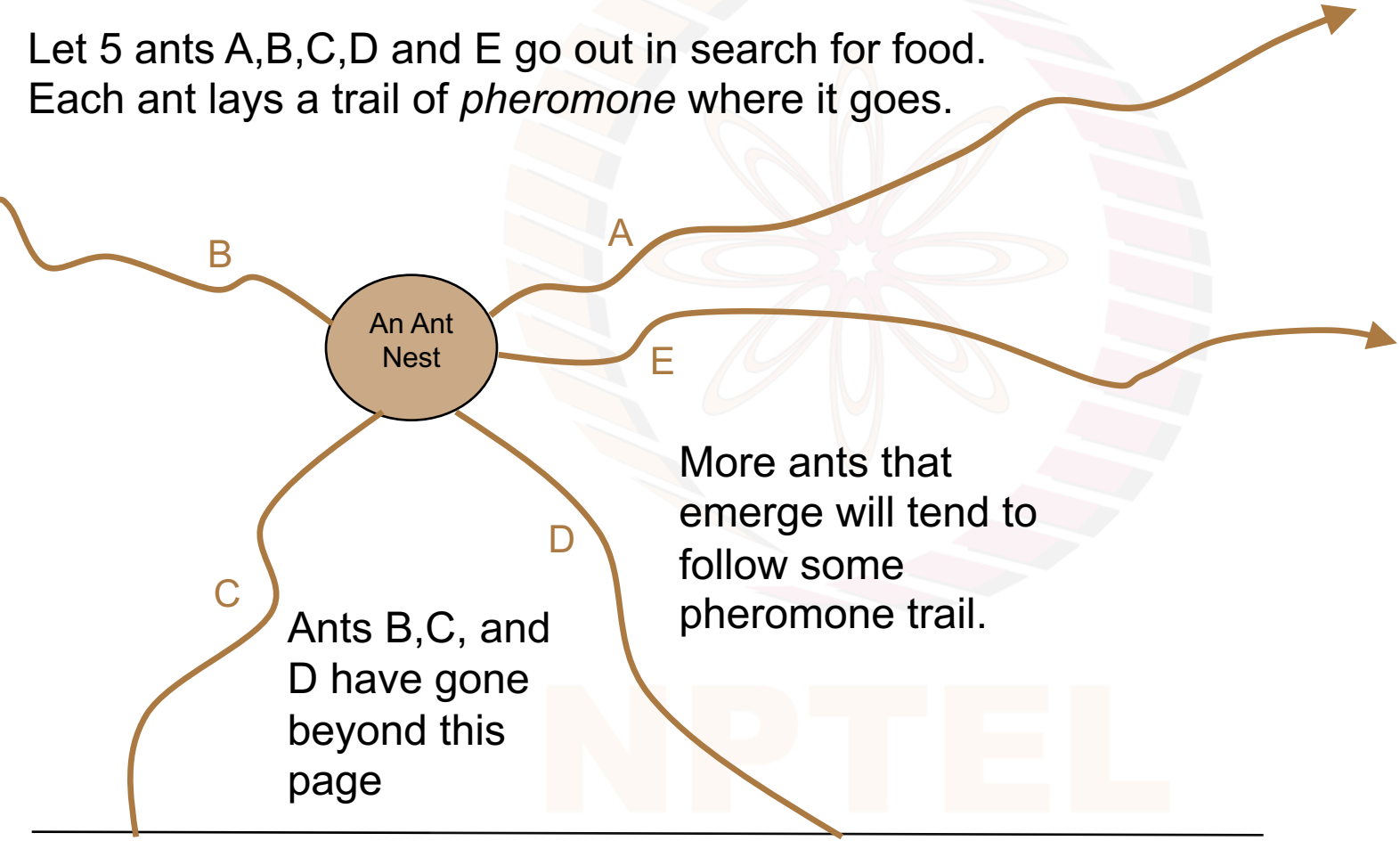
The term "ant colony" describes not only the physical structure in which ants live, but also the social rules by which ants organize themselves and the work they do.

Some workers become foragers when they mature, leaving the nest to search for food.

This cooperation and division of labour, combined with their well-developed communication systems, has allowed ants to utilise their environment in ways approached by few other animals.

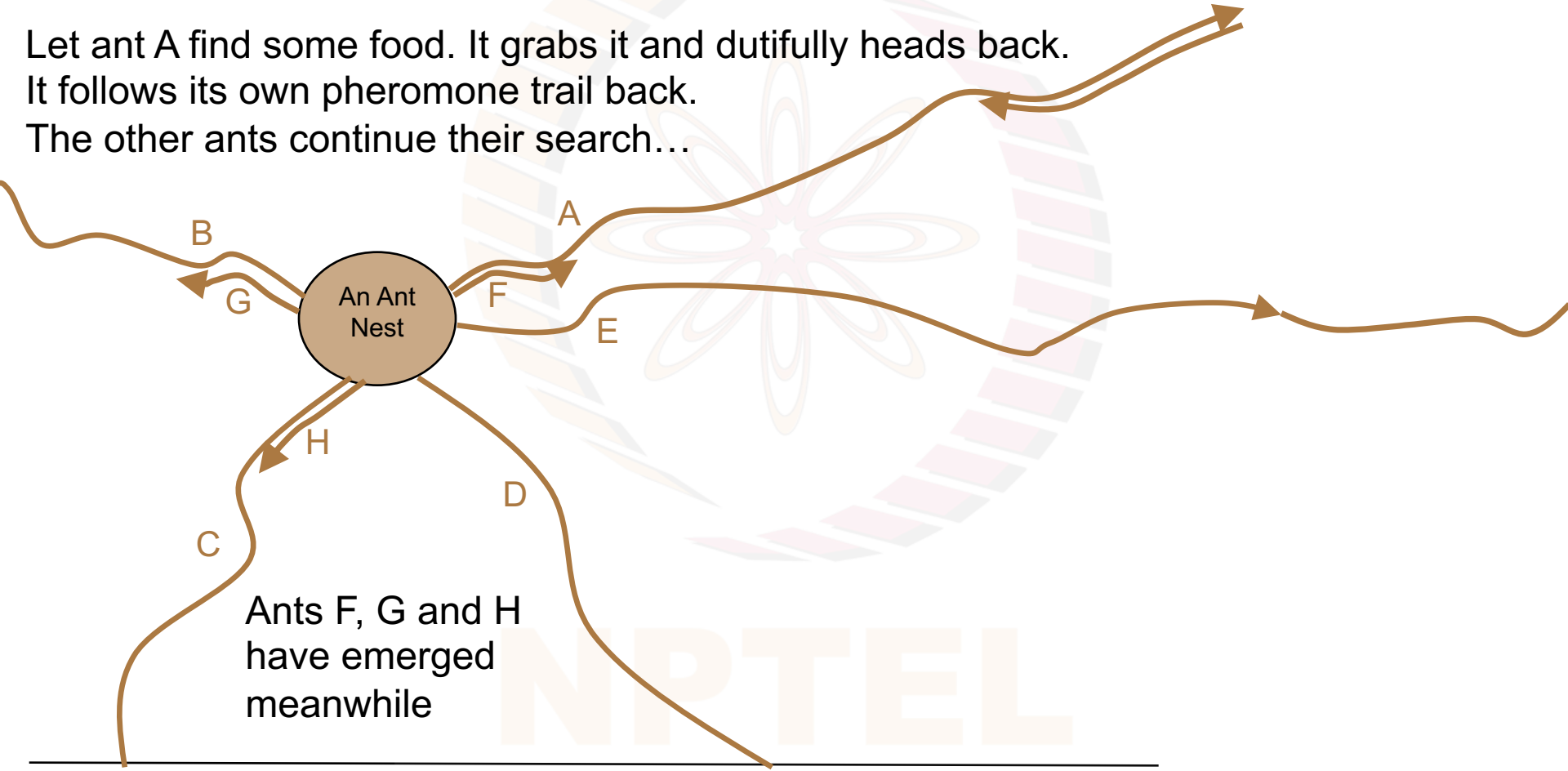
Ants foraging for food

Let 5 ants A,B,C,D and E go out in search for food.
Each ant lays a trail of *pheromone* where it goes.



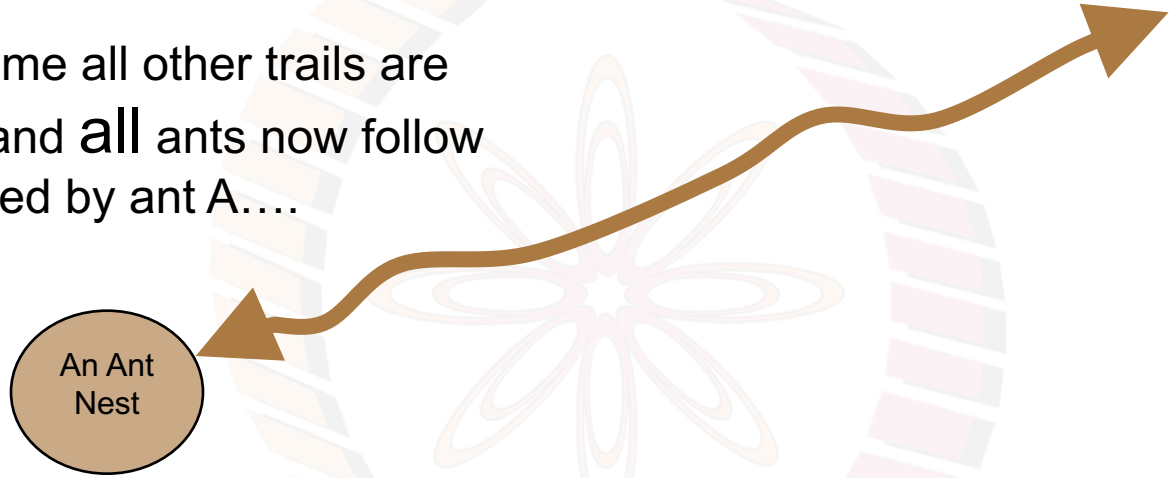
Ants A finds some food

Let ant A find some food. It grabs it and dutifully heads back.
It follows its own pheromone trail back.
The other ants continue their search…



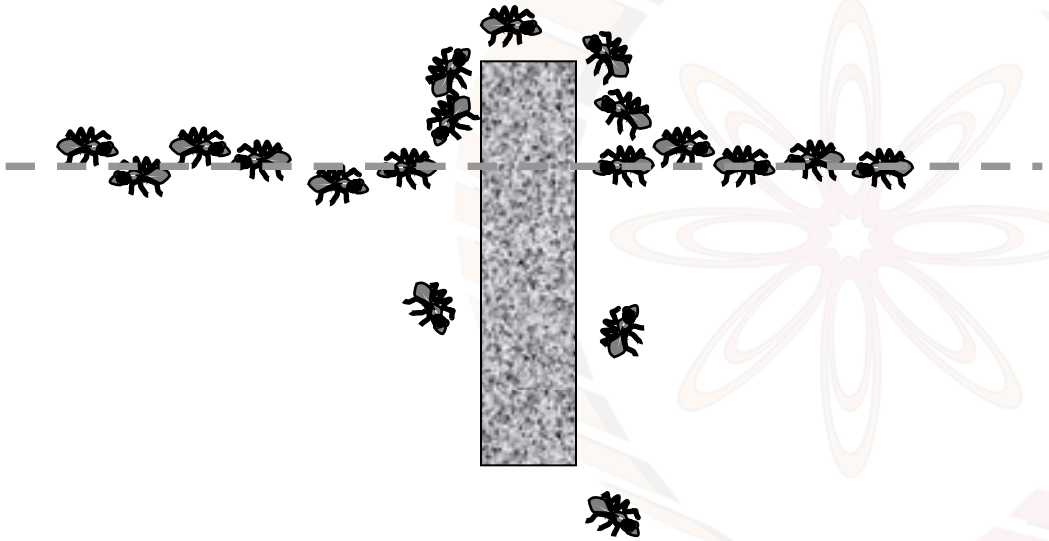
Ants foraging for food

After some time all other trails are abandoned and all ants now follow the trail formed by ant A….



As more ants travel on the trail they deposit more pheromone and the trail gets stronger and stronger, eventually becoming the caravan you might have see raiding your food.

Ants follow pheromone trails left by ants



When an obstacle is placed

on the pheromone trail of an ant colony,

they quickly find the shortest diversion around it.

Ant Colony Optimization (ACO)

Ant Colony Optimization is an optimization method that takes inspiration by the bio-semiotic communication between ants

Each ant constructs a solution using a stochastic greedy method using a combination of a heuristic function and pheromone trail following

Related to the class of algorithms known as *Swarm Optimization*

Applicable to problems that can be reduced to graph search

Presented by Marco Dorigo in 1992

We illustrate the algorithm with the TSP

The ACO Algorithm

TSP-ACO()

1 bestTour ← **nil**
2 **repeat**
3 **randomly place** M **ants on** N **cities**
4 **each ant constructs a tour in a greedy stochastic manner**
5 **update** bestTour
6 **each ant deposits pheromone only on its tour edges**
7 **until some termination criteria**
8 **return** bestTour

The ACO Algorithm (more detail)

TSP-ACO()

```
1  bestTour ← nil
2  repeat
3      randomly place M ants on N cities
4      for each ant a          ▷ construct tour
5          for n ← 1 to N
6              ant a selects an edge from the distribution P_n^a
7      update bestTour
8      for each ant a          ▷ update pheromone
9          for each edge (u, v) in the ant's tour
10             deposit pheromone ∝ 1/tour-length on edge (u, v)
11 until some termination criteria
12 return bestTour
```

Ant Colony Optimization for TSP

On a city *i* the k^{th} ant moves to city *j* with a probability given by,

$$P^k_{ij}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha} * [\eta_{ij}]^{\beta}}{\sum_{h \in allowed_k(t)} ([\tau_{ih}(t)]^{\alpha} * [\eta_{ih}]^{\beta})} & \text{if } j \in allowed_k(t) \text{ the cities ant}_k \text{ is allowed to move to} \\ 0 & \text{otherwise} \end{cases}$$

Where, $\tau_{ij}(t)$ is pheromone on $edge_{ij}$ and η_{ij} is called visibility which is inversely proportional to the distance between cities *i* and *j*.

ACO: Updating Pheromone

After constructing a tour in *n* time steps

each ant *k* deposits an amount of pheromone Q/L_k on the edges it traversed

which is inversely proportional to the cost of the tour L_k it found. Q is an appropriate constant.

Total pheromone deposited on $edge_{ij}$ is $\Delta\tau_{ij}(t,t+n)$

The total pheromone on $edge_{ij}$ is updated as

$$\tau_{ij}(t+n) = (1-\rho)*\tau_{ij}(t) + \Delta\tau_{ij}(t, t+n)$$

where ρ is the rate of evaporation of pheromone

End
Randomized Algorithms