Artificial Intelligence:
Search Methods for Problem Solving

Solution Space Search:
Escaping Local Optima

A First Course in Artificial Intelligence: Chapter 3

Deepak Khemani

Department of Computer Science & Engineering

IIT Madras

Hill Climbing – a local search algorithm

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

```
Hill Climbing
    node ← Start
    newNode ← head(sort_h moveGen (node)))
    While h(newNode) < h(node) Do
            node ← newNode
            newNode ← head(sort_h (moveGen (node))))
    endWhile
    return newNode
End

Algorithm Hill Climbing
```

In practice sorting is not needed, only the best node

Change of termination criterion

Local search – Hill Climbing has burnt its bridges by not storing OPEN

Hill Climbing – a constant space algorithm

HC only looks at local neighbours of a node. It's space requirement is thus *constant*!

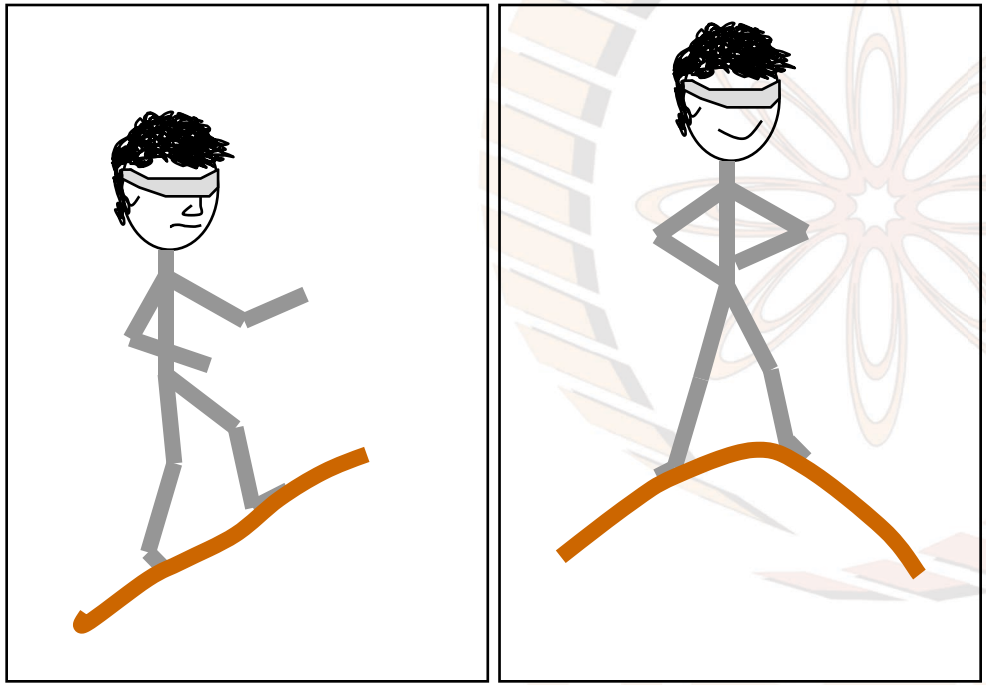
A vast improvement on the exponential space for BFS and DFS

HC only moves to a better node. It terminates if cannot. Consequently the *time complexity is linear*.

It's *termination criterion is different*. It stops when no better neighbour is available. It treats the problem as an *optimization problem*.

However, it is not complete, and may not find the global optimum which corresponds to the solution!

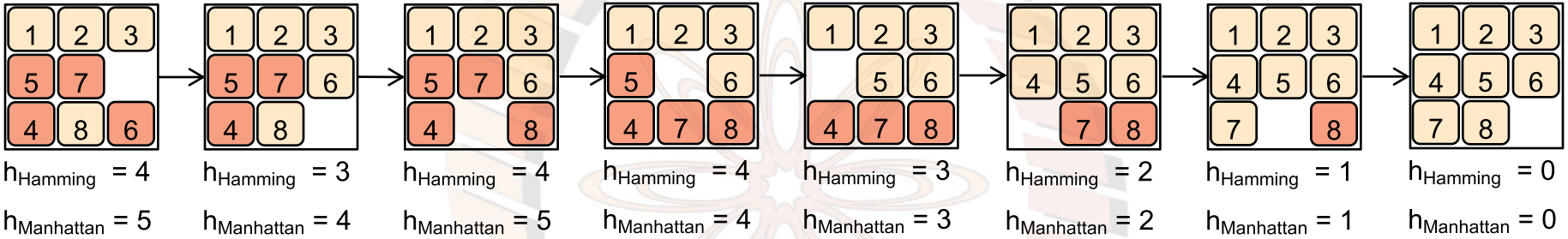
Steepest gradient ascent

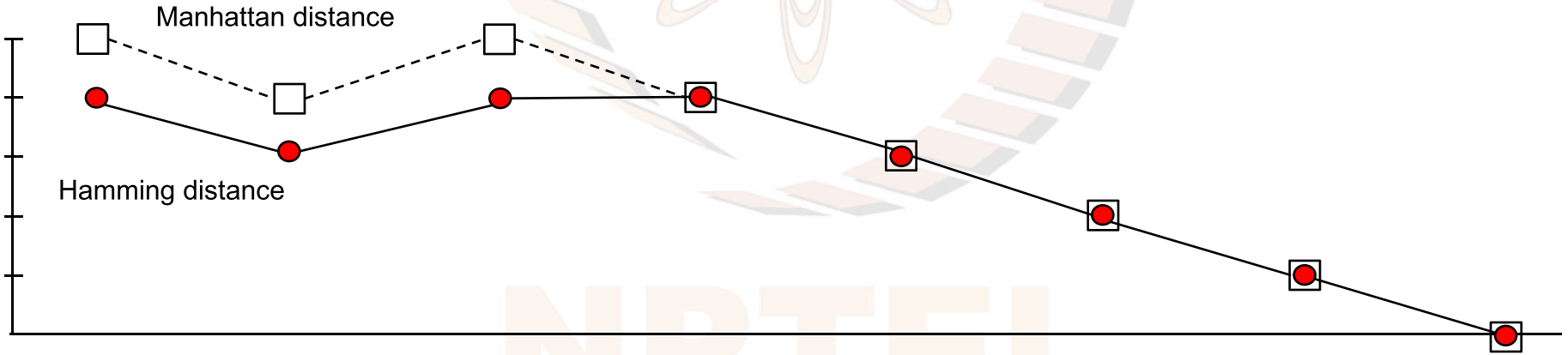


Hill Climbing (for a maximization problem)

May end on a local maximum

8-puzzle: A Solution





A plot of heuristic value along the solution path

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 three deterministic methods next,

and will look at randomized methods later

First we look at searching in

the solution space or plan space

The SAT problem

Given a Boolean formula made up of a set of propositional variables V= {a, b, c, d, e, … } each of which can be *true* or *false*, or *1* or *0*, to find an assignment for the variables such that the given formula evaluates to *true* or *1*.

For example, F = ((aV~e) $\wedge$ (eV~c)) $\supset$ (~cV~d) can be made *true* by the assignment {a=*true*, c=*true*, d=*false*, e=*false*} amongst others.

Very often *SAT* problems are studied in the *Conjunctive Normal Form (CNF).* For example, the following formula has five variables (a,b,c,d,e) and six clauses.

$$(bV\sim c) \wedge (cV\sim d) \wedge (\sim b) \wedge (\sim aV\sim e) \wedge (eV\sim c) \wedge (\sim cV\sim d)$$

Solution Space Search and Perturbative methods

The Solution Space is the space of candidate solutions.

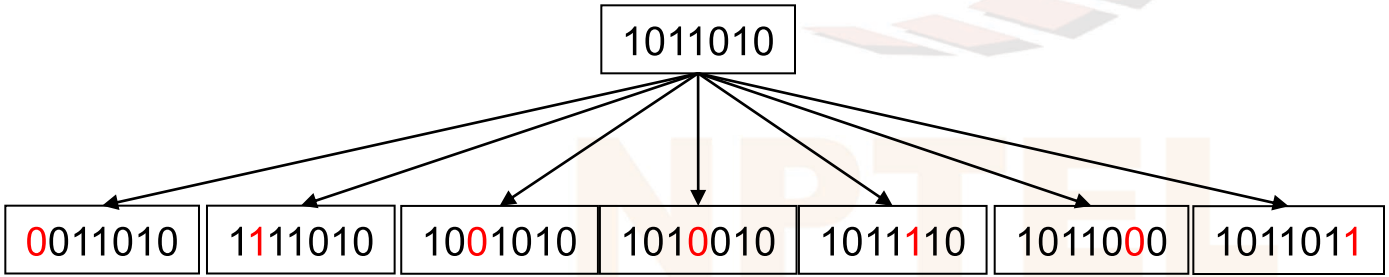
A local search method generates the neighbours of a candidate by applying some perturbation to the given candidate

MoveGen function = neighbourhood function

A SAT problem with N variables has 2^N candidates

- where each candidate is a N bit string

When N= 7, a *neigbourhood function* may change one bit.

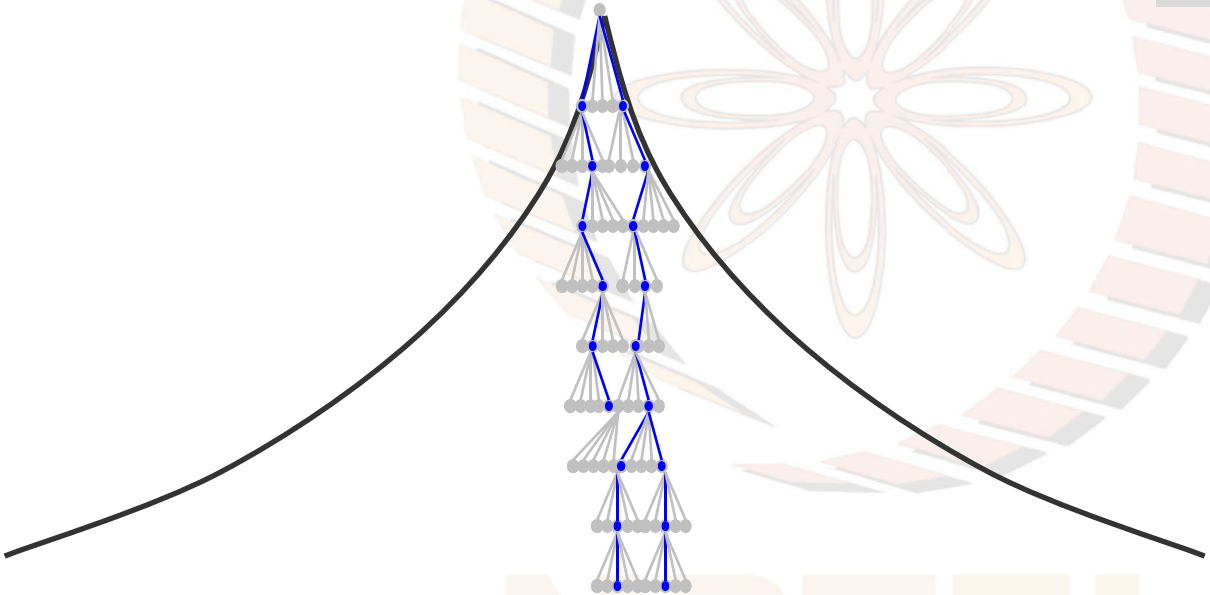


Beam Search

Look at more than one option at each level.

For a beam width *b*, look at best *b* options.

After all computer memory is becoming cheaper

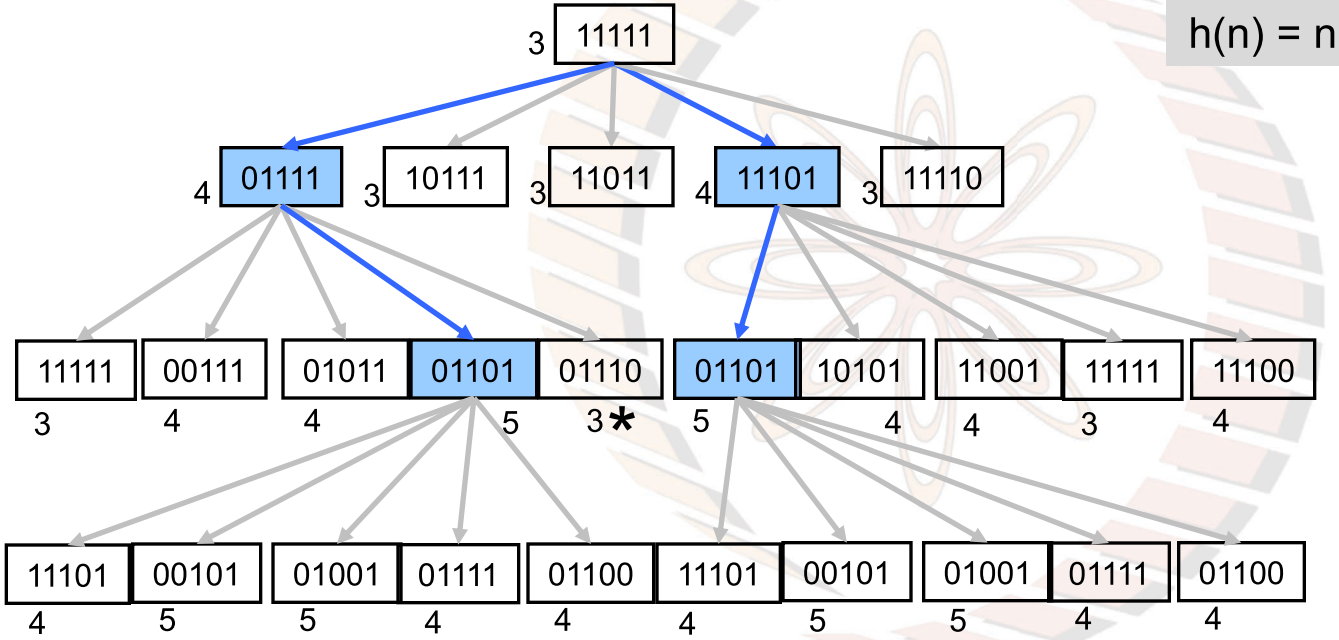


Beam Search with beam width = 2

Beam Search for SAT

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

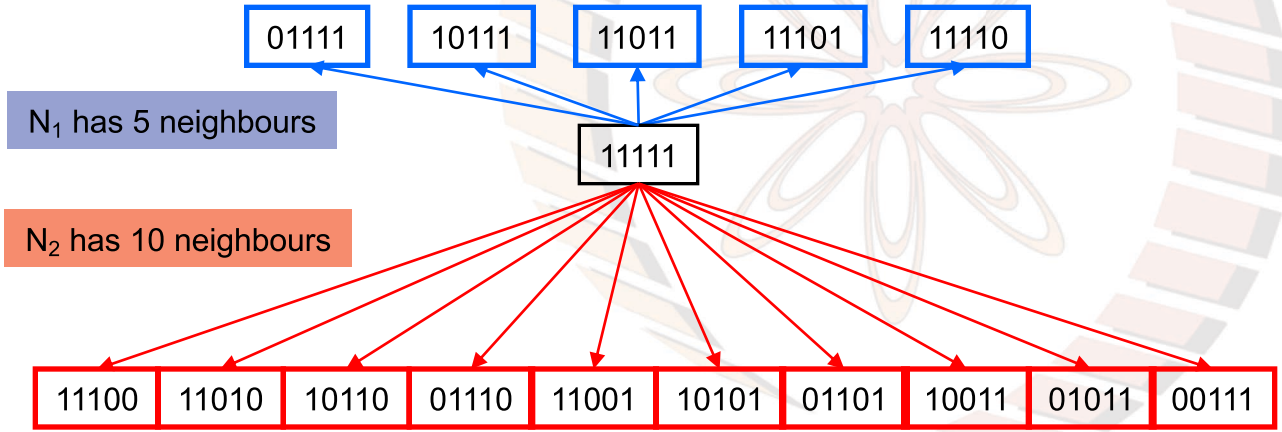
h(n) = number of clauses satisfied



Beam Search with width = 2 fails to solve the *SAT* problem starting with 11111. Starting with a value 3 the solution should be reached in 3 steps, because it is Hill Climbing. In fact the node marked * leads to a solution in three steps.

Different neighbourhood functions

Consider a SAT problem with 5 variables. We can try different neighbourhood functions. Let N_i stand for a neighbourhood *function* that flips *i* bits. Consider N_1 and N_2



Let N_{12} stand for changing 1 or 2 bits. This will have 15 neighbours

We can devise more and more dense neighbourhood functions with the most dense being $N_{1\text{-}n}$ where *n* is the number of variables, and $N_{1\text{-}n}$ stands for changing any of 1 to *n* bits.

Effect of density on performance of Hill Climbing

Hill Climbing

Inspect all neighbours

Move to the best neighbour if it is better than current node

Hill climbing gets stuck on a local optimum when none of the neighbours is better than the current node

The denser the neighbourhood the less likely this is, but
the denser the neighbourhood is the more the number of neighbours that has to be inspected at each step

Observe that $N_{1\text{-}n}$ has 2^n neighbours !

Inspecting them amounts to brute force search

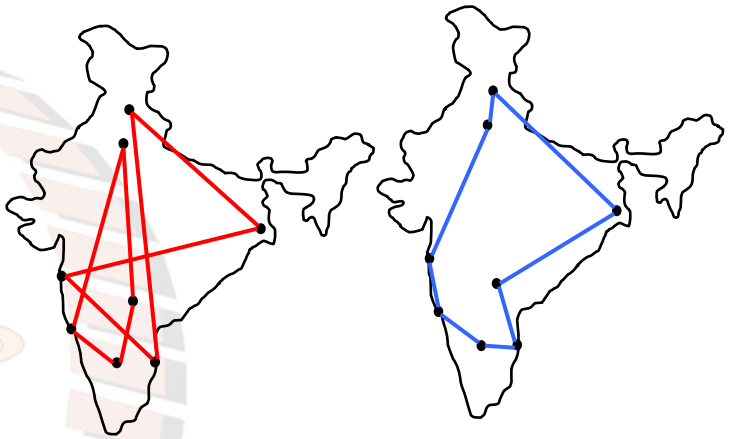
Variable Neighbourhood Descent

```
VariableNeighbourhoodDescent()
1       node ← start
2       for i ← 1 to n
3           do moveGen ← MoveGen(i)
4              node ← HillClimbing(node, moveGen)
5       return node
```

The algorithm assumes that the function *moveGen* can be passed as a parameter. It assumes that there are *N moveGen* functions sorted according to the density of the neighbourhoods produced.

The hope is that most of the “climbing” will be done with sparser neighbourhood functions.

The Traveling Salesman Problem (TSP)



Given a set of cities and given a distance measure between every pair of cities, the task is to find a Hamiltonian cycle having least cost.

When the underlying graph is not completely connected we can add the missing edges with very high cost.

The TSP is NP-hard.

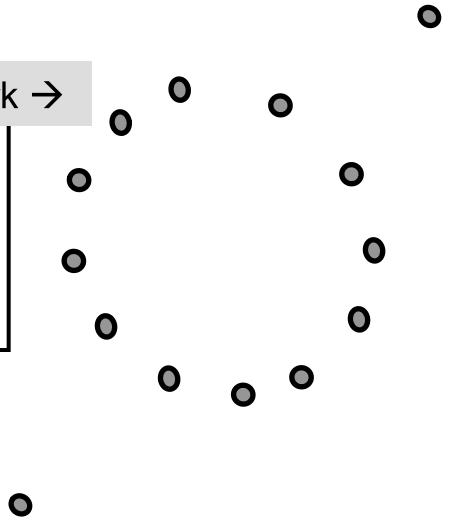
A collection of problems from various sources and problems with known optimal solutions ia available at TSPLIB

http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

TSP : Greedy Constructive Methods

Nearest Neighbour Heuristic
- Start at some city.
- Move to nearest neighbour
  - as long as it does not close the loop prematurely

An example where it does not work →



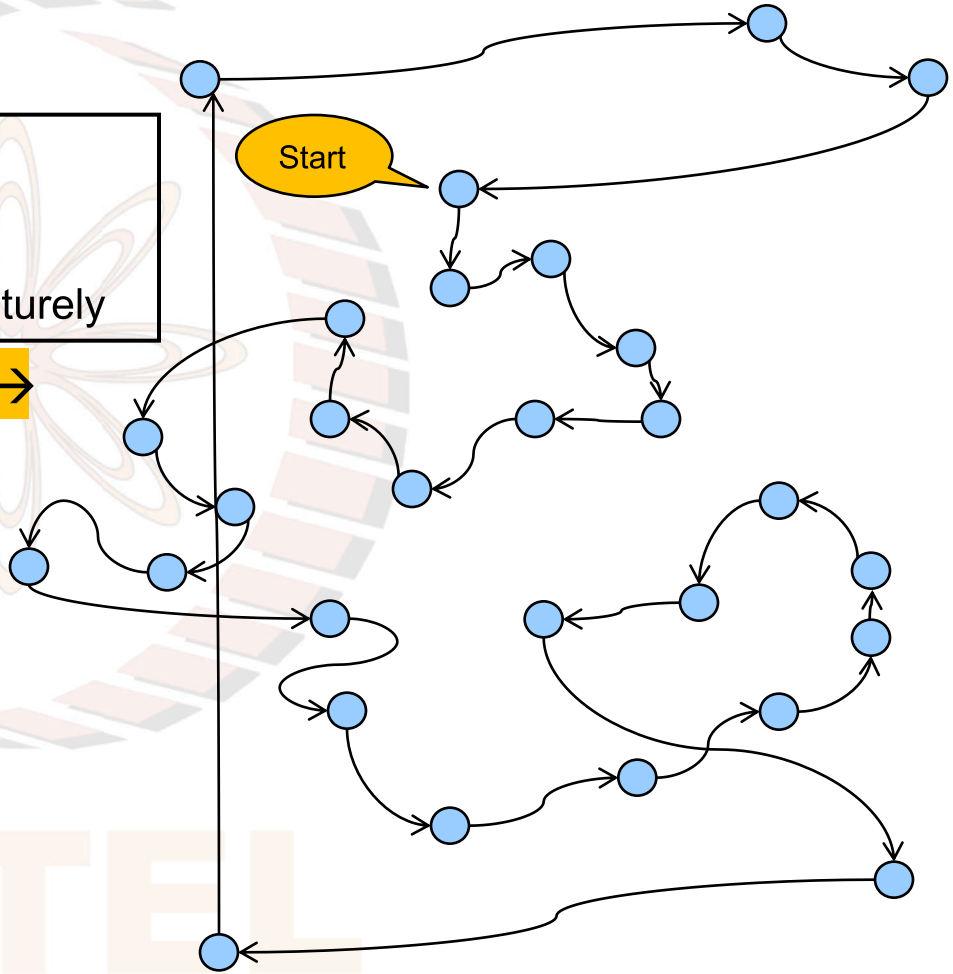
Variations:
- Extend the partial tour at either end.

Greedy Heuristic
- Sort the edges
- Add shortest available edge to the tour
  - as long as it does not close the loop prematurely

TSP : Greedy Constructive Methods

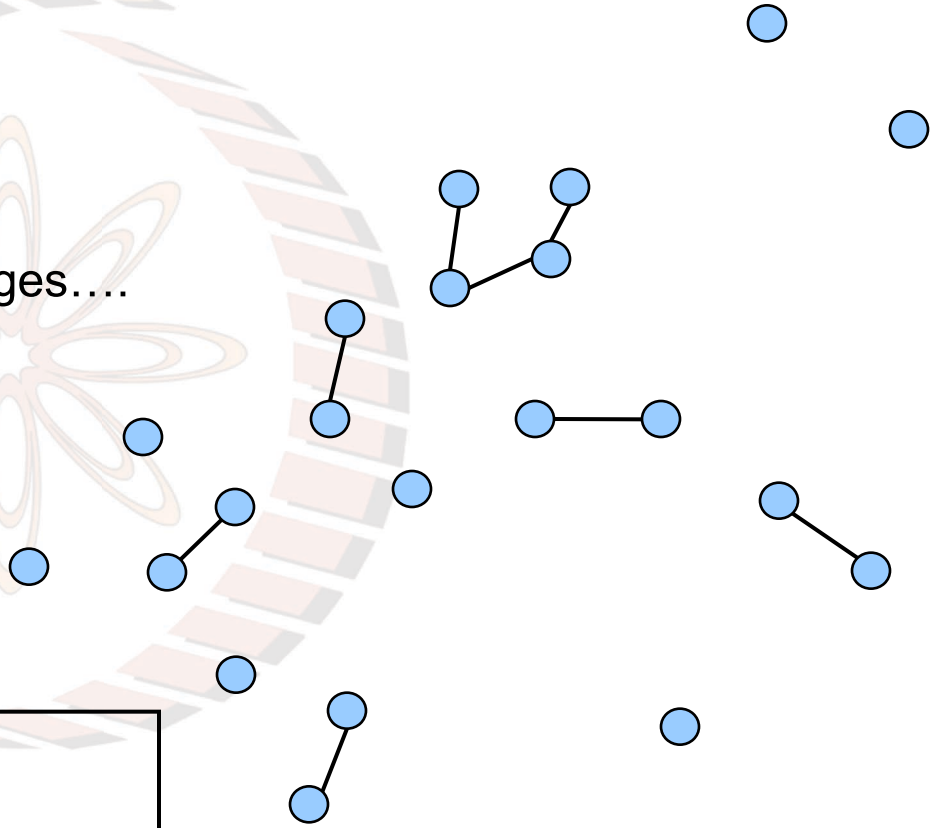
Nearest Neighbour Heuristic
- Start at some city.
- Move to nearest neighbour
  - as long as it does not close the loop prematurely

An example run →



TSP : Greedy Constructive Methods

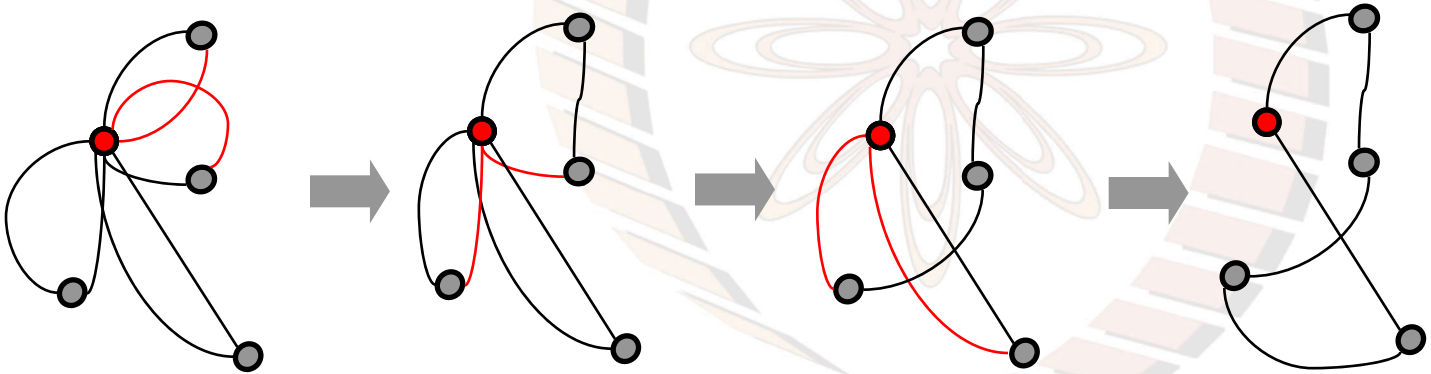
Some early edges….



Greedy Heuristic
- Sort the edges
- Add shortest available edge to the tour
  - as long as it does not close the loop prematurely

Savings Heuristic

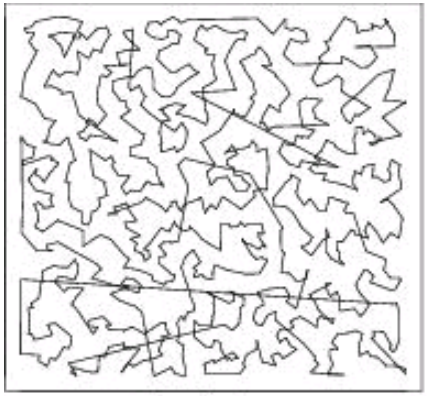
The Savings Heuristic starts with
(n-1) tours of length 2 anchored on a base vertex
and performs (n-2) merge operations to construct the tour.



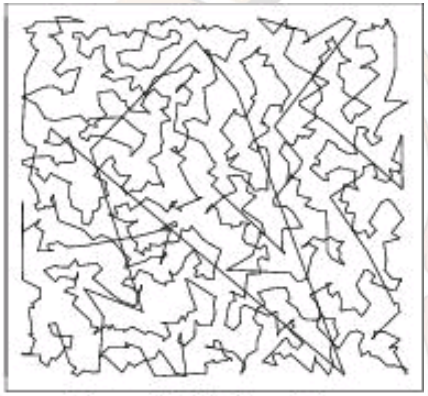
Remove two edges connected the base vertex from two loops and add an edge to connect the two hanging vertices.

The two edges are chosen such that there is maximal savings of (total) cost in the resulting graph.

The constructive algorithms in action

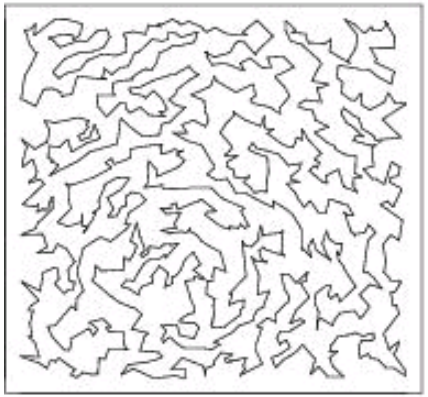


Greedy Tour

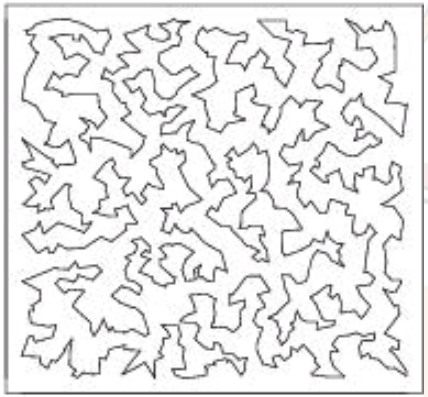


Nearest Neighbour Tour

Tours found by some heuristic constructive algorithms. Figure taken from http://www.research.att.com/~dsj/chtsp/

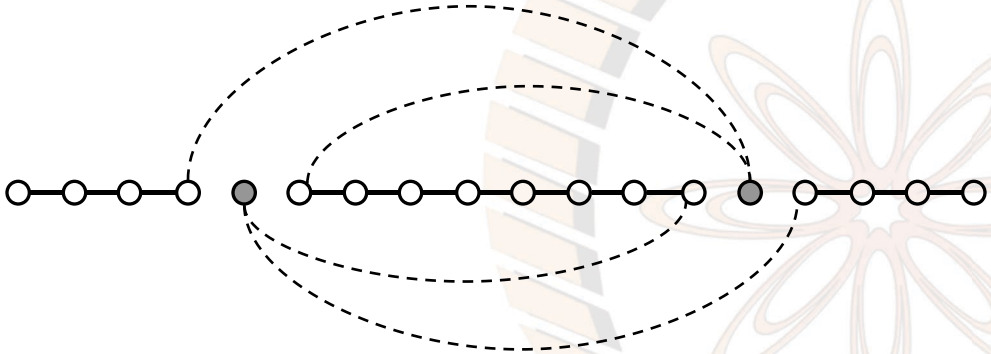


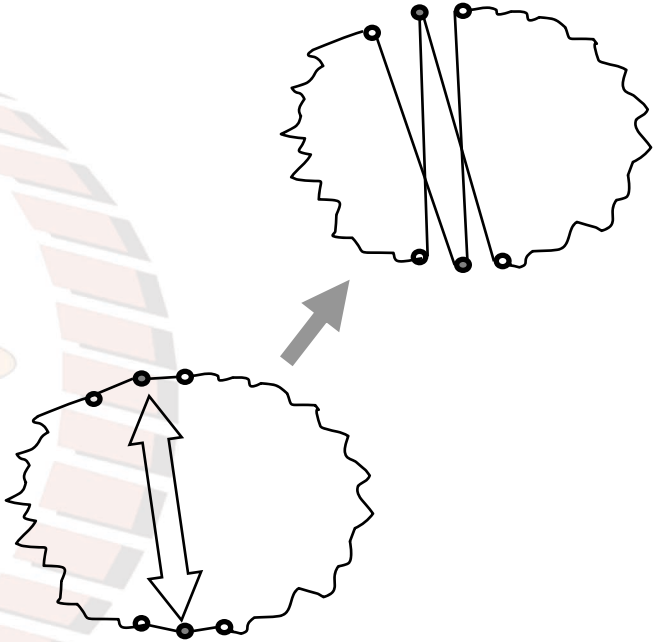
Savings Tour



Optimal Tour

Solution space : Perturbation operators

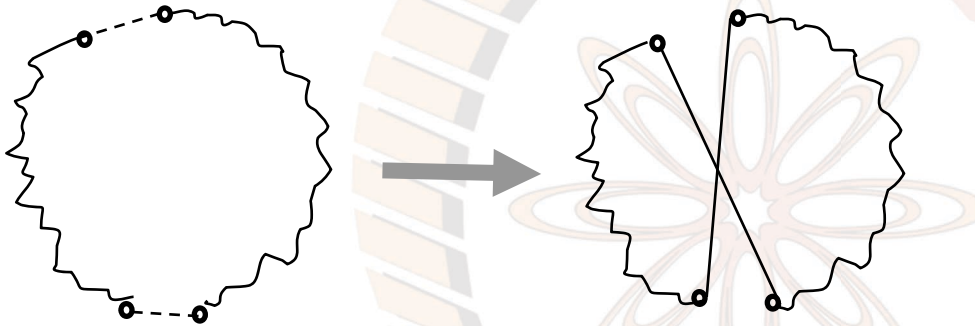




In the 2-city-exchange the two shaded cities exchange their position in the path/order representation. The new tour has the dashed edges instead of the four broken ones in the linear order.

The two cities can be selected in ${}^{n}C_{2}$ ways

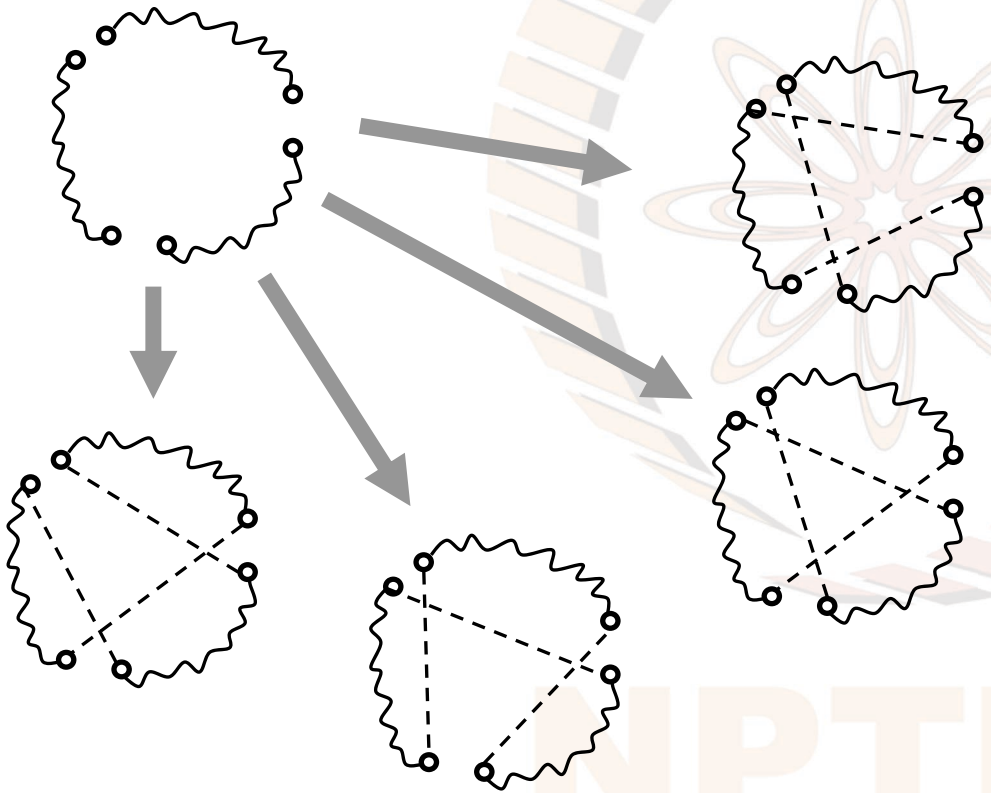
Two Edge Exchange



In the 2-edge-exchange the tour is altered as shown.

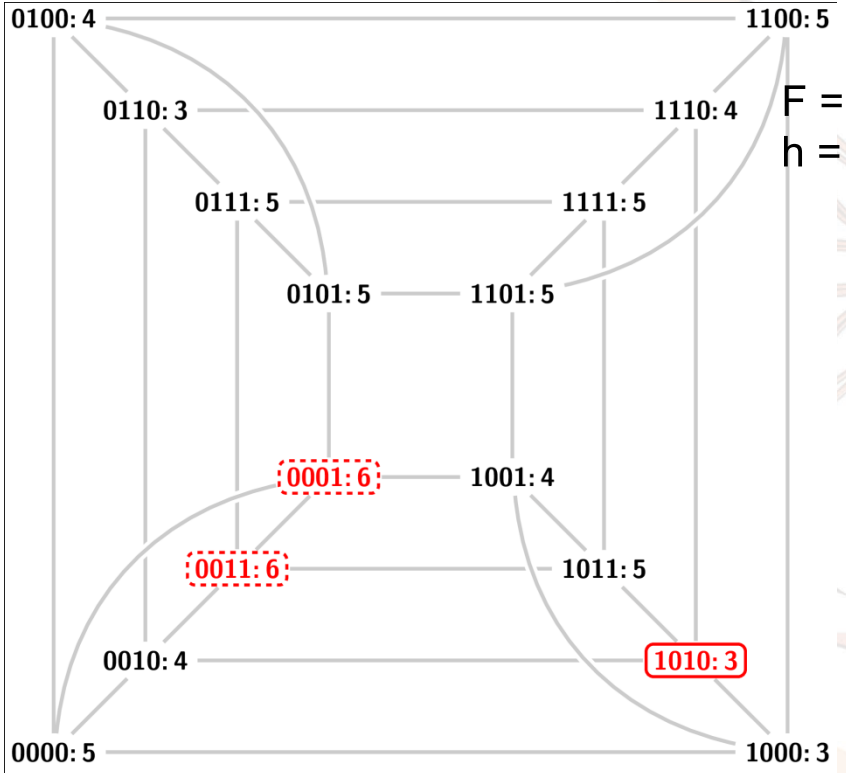
The two edges can be selected in ${}^{n}C_{2}$ ways

Three-edge Exchange



In 3-edge-exchange three edges are removed and replaced by other edges. This can be done in four ways. The three edges can be selected in ${}^{n}C_3$ ways.

The 2-city exchange is one case of the 4-edge exchange.

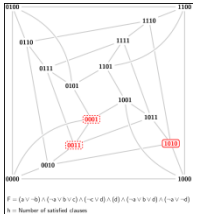


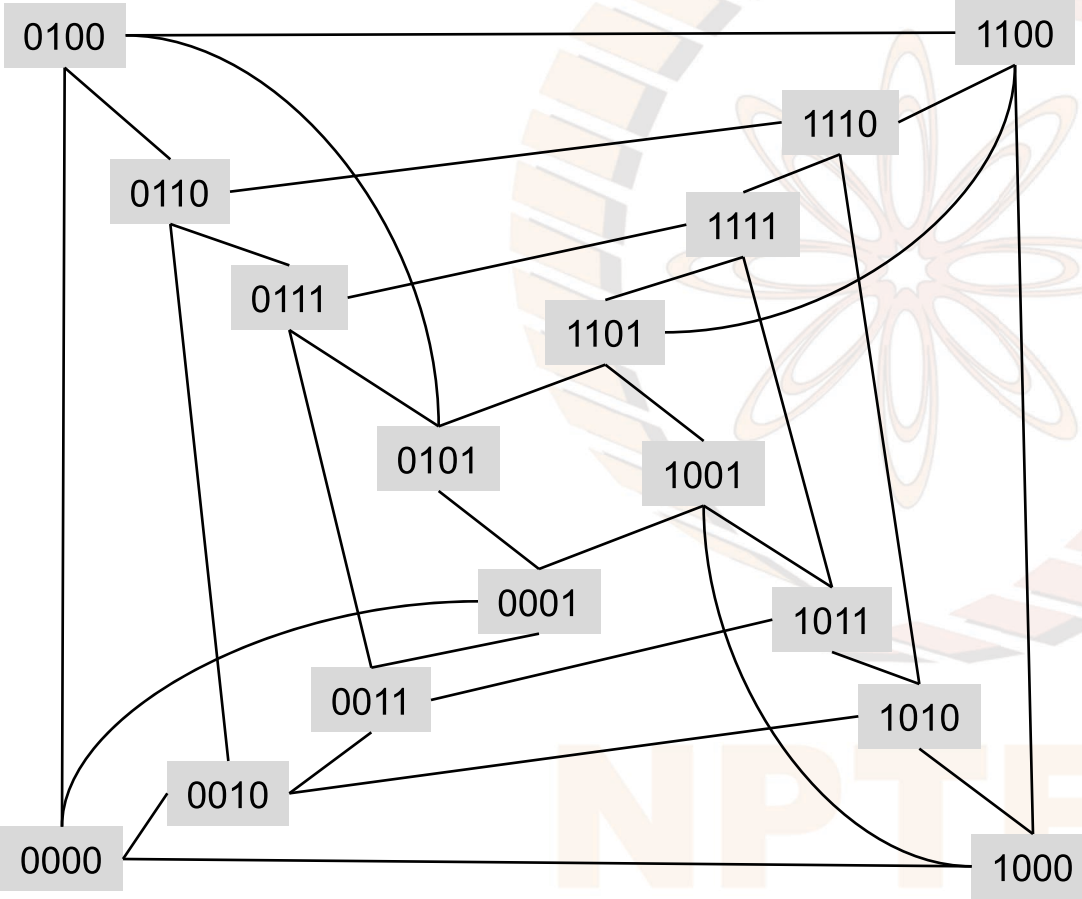
$F = (a \vee \neg b) \wedge (\neg a \vee b \vee c) \wedge (\neg c \vee d) \wedge (d) \wedge (\neg a \vee b \vee d) \wedge (\neg a \vee \neg d)$

h = Number of satisfied clauses

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

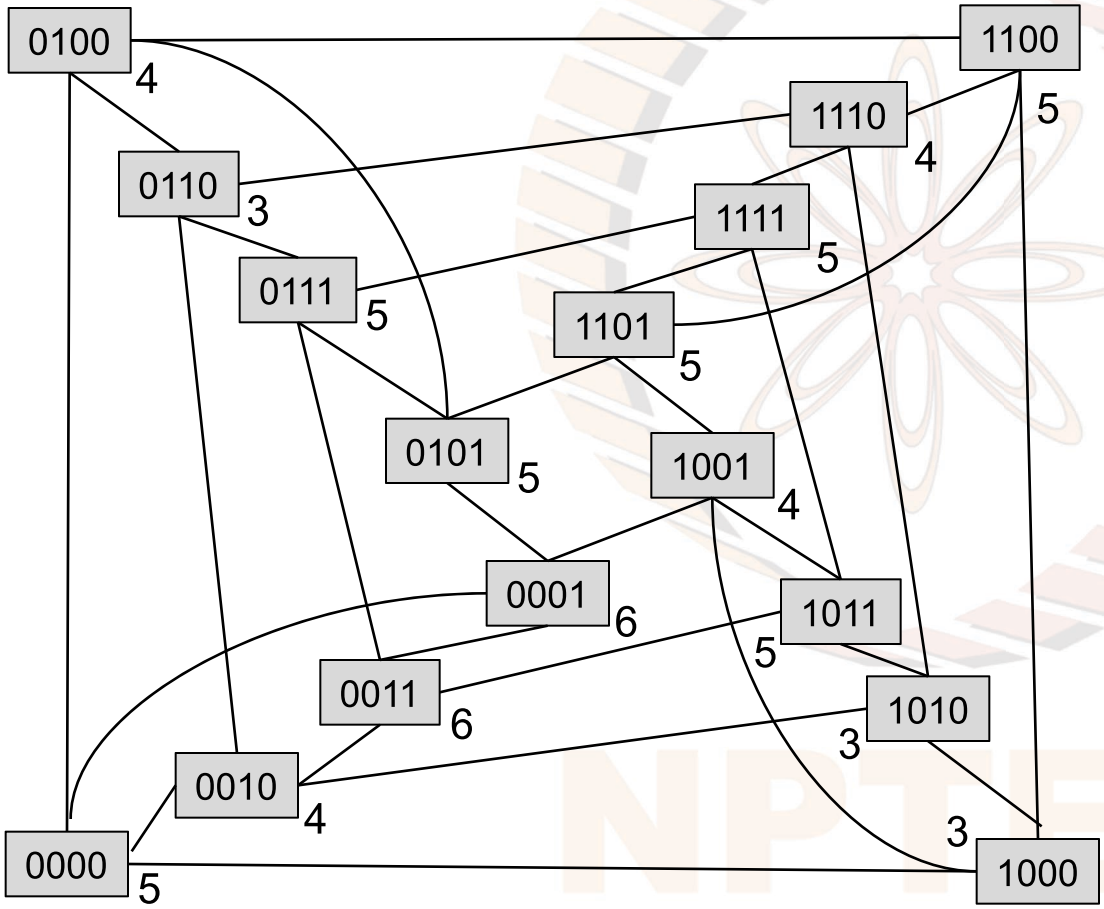
h = number of satisfied clauses

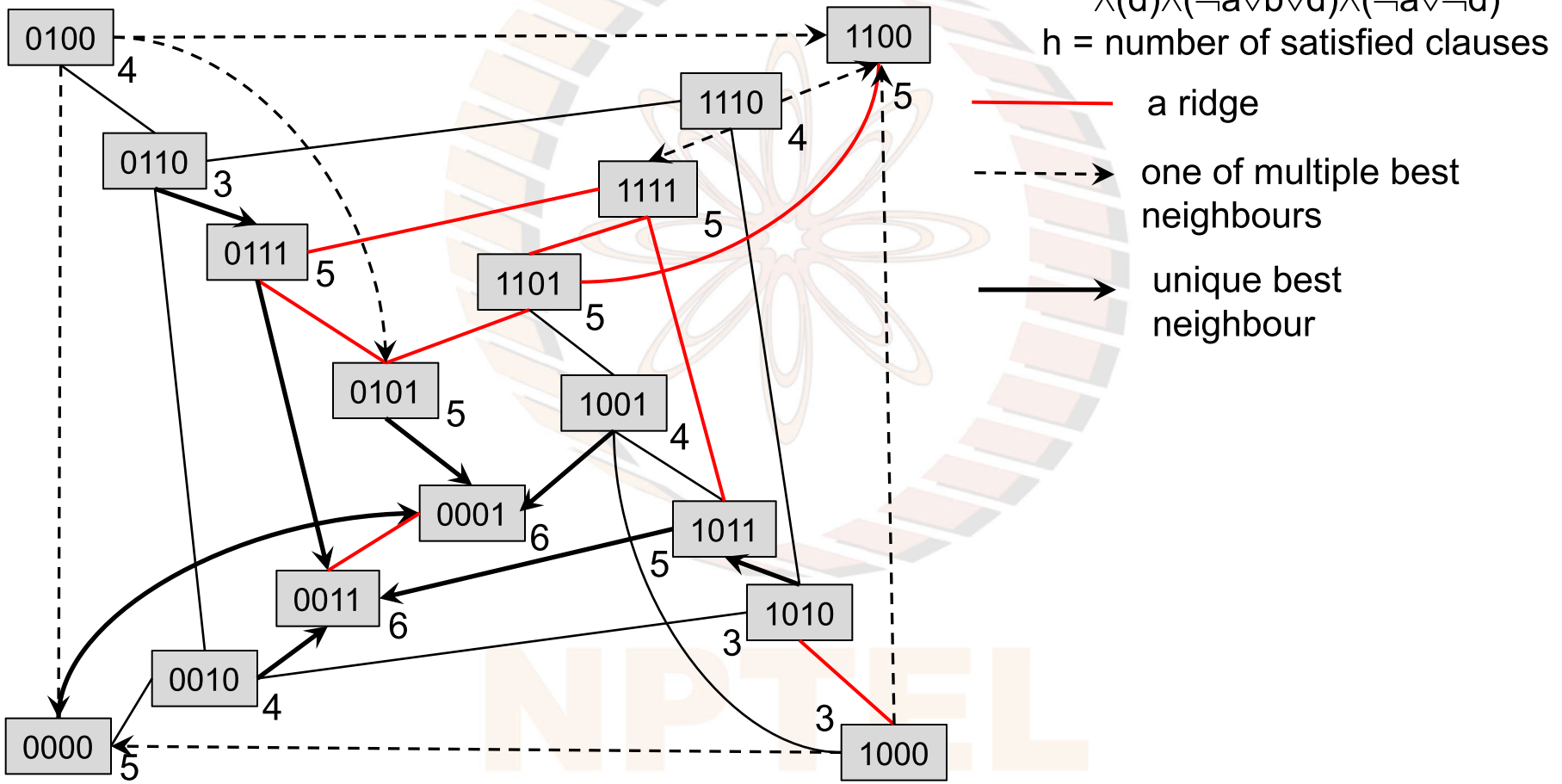

0100
1100
1110
0110
1111
0111
1101
0101
1001
0001
1011
0011
1010
0010
0000
1000

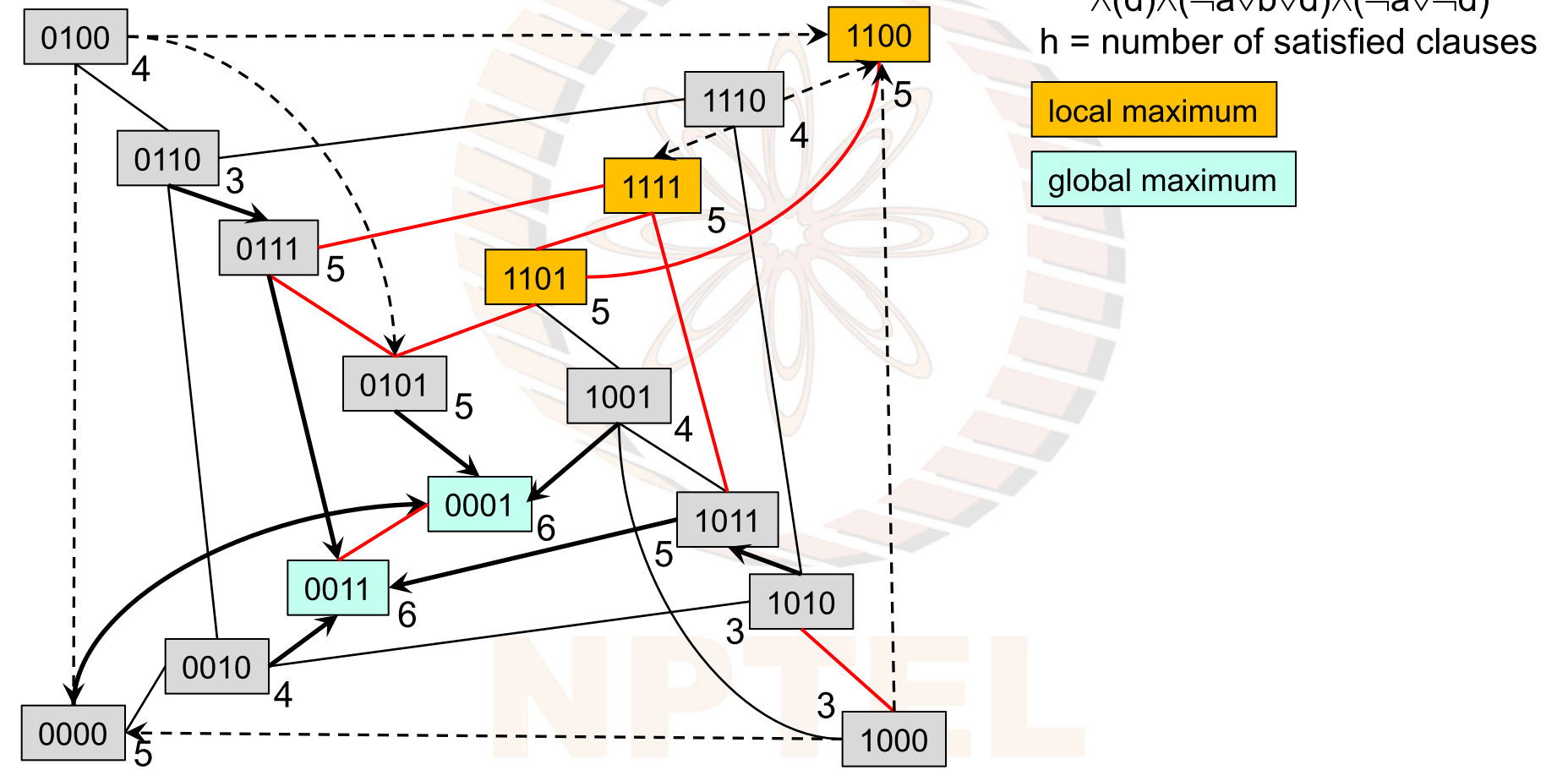

0100
1100
1110
0110
1111
0111
1101
0101
1001
0001
1011
0011
1010
0010
0000
1000

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

h = number of satisfied clauses



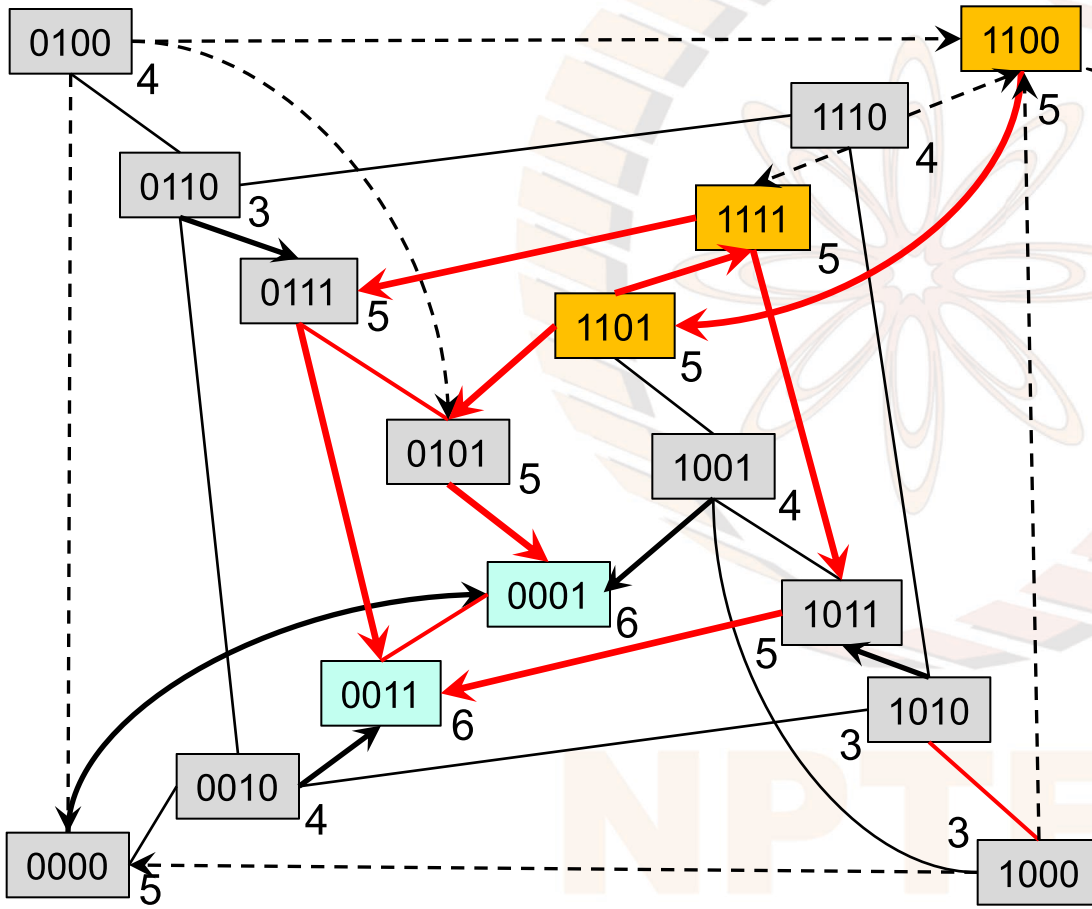

F = (a∨¬b)∧(¬a∨b∨c)∧(¬c∨d)
∧(d)∧(¬a∨b∨d)∧(¬a∨¬d)
h = number of satisfied clauses
a ridge
one of multiple best neighbours
unique best neighbour
0100
1100
1110
0110
1111
0111
1101
0101
1001
0001
1011
0011
1010
0010
0000
1000
4
3
5
6


F = (a∨¬b)∧(¬a∨b∨c)∧(¬c∨d)
∧(d)∧(¬a∨b∨d)∧(¬a∨¬d)
h = number of satisfied clauses
local maximum
global maximum
0100
1100
1110
0110
1111
0111
1101
0101
1001
0001
1011
0011
1010
0010
0000
1000
4
4
3
5
5
5
5
4
6
5
6
3
4
3
5

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



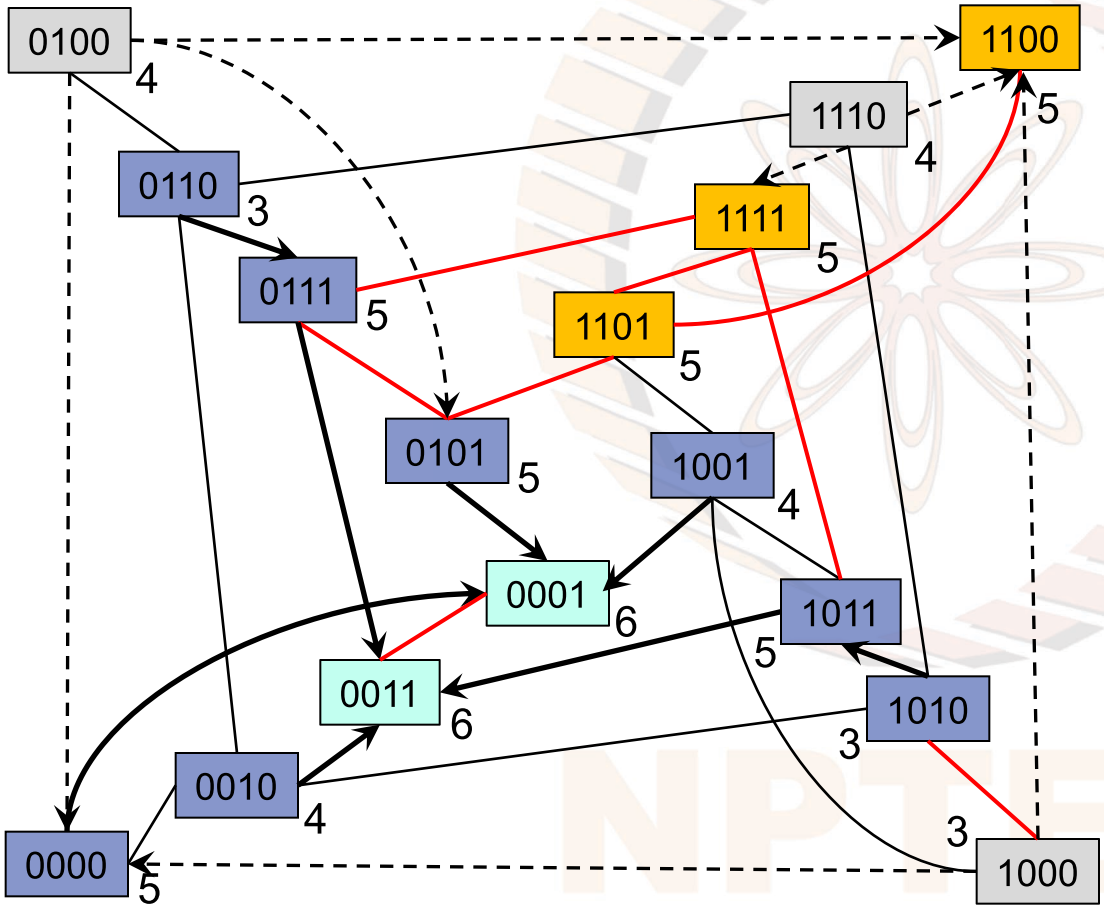
Starting from here Tabu Search traverses one of the three possible routes along the ridges to reach the goal

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

Hill Climbing reaches solution from here



End
Deterministic Heuristic Search