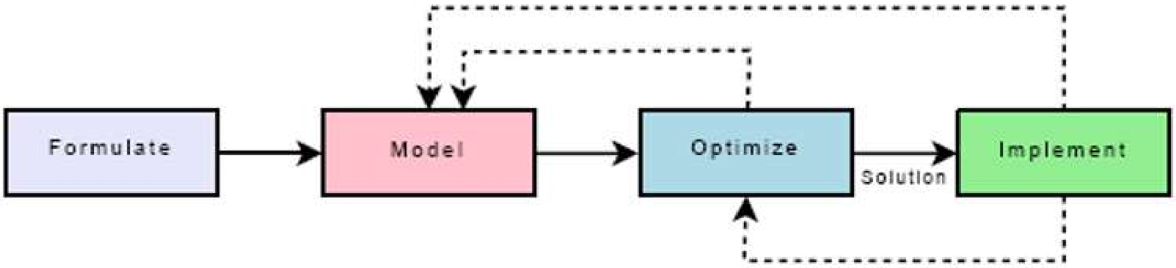
Lec1: Introduction to Advanced Intelligent Search

– From modeling to decision making (Optimization Problems)



In practice, we find solutions for models representing problems. Where usually models are simplifications of the reality. The classical process in decision making: formulate, model, solve, and implement. In practice, this process may be iterated to improve the optimization model or algorithm until an acceptable solution is found.

– Optimization problem

Definition (minimization problem): couple (S,f)

Given a search space S which represents feasible solutions

Given an objective function f: S →R

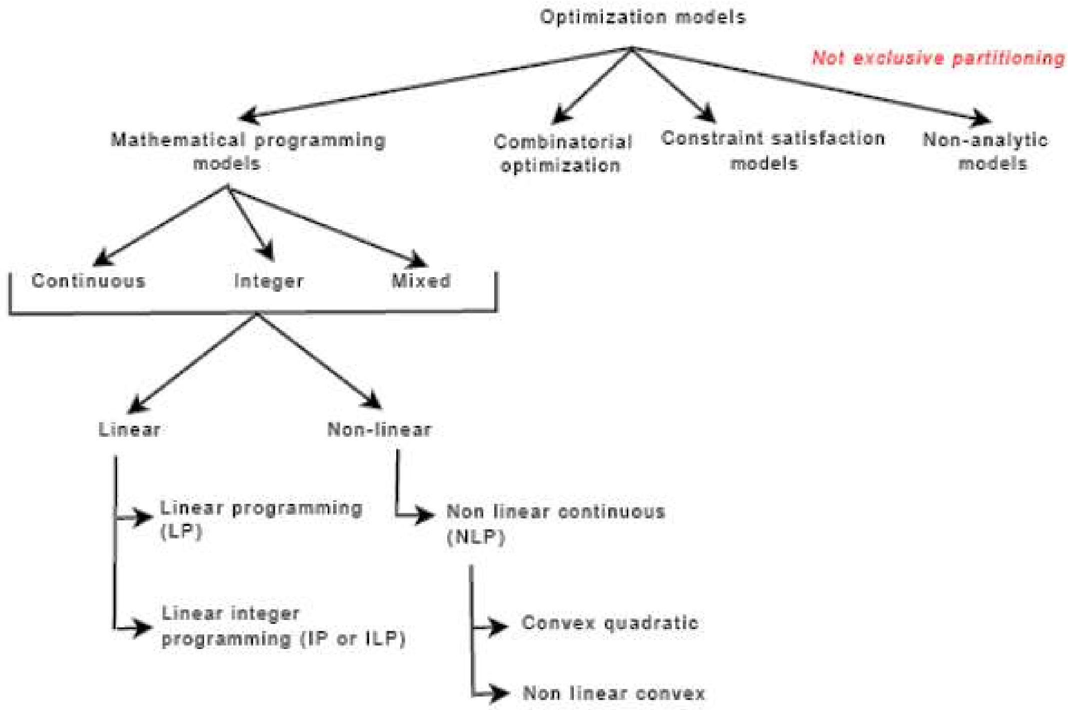
Find s*∈S such as:

$$f(s^*) \leq f(s) \qquad \forall s \in S$$

s*: global optimum

- Large scale and complex optimization problems in many areas of science and industry (Telecommunications, Biology, Transportation-Logistics, Environment, Finance, Design, ...).
- Problem size more and more important (combinatorial explosion) and/or Delays more and more reduced.

- Optimization models


Optimization models
Not exclusive partitioning
Mathematical programming models
Combinatorial optimization
Constraint satisfaction models
Non-analytic models
Continuous
Integer
Mixed
Linear
Non-linear
Linear programming (LP)
Linear integer programming (IP or ILP)
Non linear continuous (NLP)
Convex quadratic
Non linear convex

- Combinatorial problems: When the decision variables are discrete
- Continuous problems: When the decision variables are continuous
- Mixed problems

Combinatorial Problems

- Examples of real-world combinatorial optimization problems include:
  - Assembly-line balancing problems
  - Vehicle routing and scheduling problems
  - Facility location problems
  - Facility location problems
  - Facility layout
  - design problems
  - Job sequencing and machine scheduling problems
  - Manpower
  - planning problems and Production planning and distribution

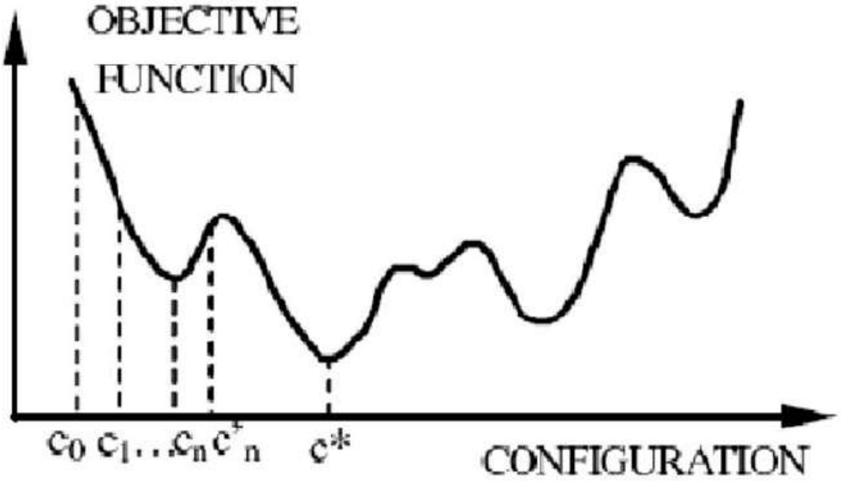
Combinatorial Optimization

- Combinatorial optimization problems are often easy to state but very Difficult to solve.
- Many of the problems arising in applications NP-hard , that is, it is strongly believed that are cannot they be solved to optimality within polynomially bounded computation time.
- Two classes of algorithms are available for the solution of combinatorial Optimization problems:
- Exact algorithms
- Approximate algorithms
- Exact algorithms are guaranteed to find the optimal solution and to prove Its optimality for every finite size instance of a combinatorial instance- optimization problem within an dependent run time .
- In the case of NP-hard problems, in the worst exponential time case, to Find the optimum.
- For most NP-hard problems the performance Of exact algorithms is not satisfactory.
- For optimal solutions cannot be efficiently obtained in practice, the only Possibility is to trade optimality for efficiency.
- Approximate algorithms, often also called heuristic methods or simply , seek to obtain good, that is, near-optimal solutions at relatively low computational cost without being able to guarantee the optimality of solutions
- A disadvantage of heuristic methods is that they:
- either generate only a very limited number of different solutions, or
- they stop at poor quality local optima, which is the case for iterative improvement methods.
- Metaheuristics have been proposed which try to bypass these problems.

- Metaheuristics apply to solve the problems difficult known as of optimization
- Available from the 1980s

Definition

- The word heuristic has its origin in the old Greek word heuriskein: art of discovering new strategies (rules) to solve problems Generate « high quality » solutions in a reasonable time for practical use. No guarantee to find the global optimal solution. The suffix *meta* also a Greek word: upper level methodology. Introduced by F. Glover in 1986. Metaheuristic: Upper level general methodology (templates) that can be used as guiding strategies in designing underlying heuristics to solve specific optimization problems.
- A metaheuristic is a set of algorithmic concepts that can be used to define heuristic methods applicable to a wide set of different problems.
- A Metaheuristic can be seen as a general purpose heuristic method toward promising regions of the search space containing high-quality solutions.
- A metaheuristic is a general algorithmic framework which can be applied to different optimization problems with relatively few modifications to make them adapted to a specific problem
- Metaheuristics have capability to be extracted from a local minimum



- The metaheuristics are from now on regularly employed in all the of engineering sectors,

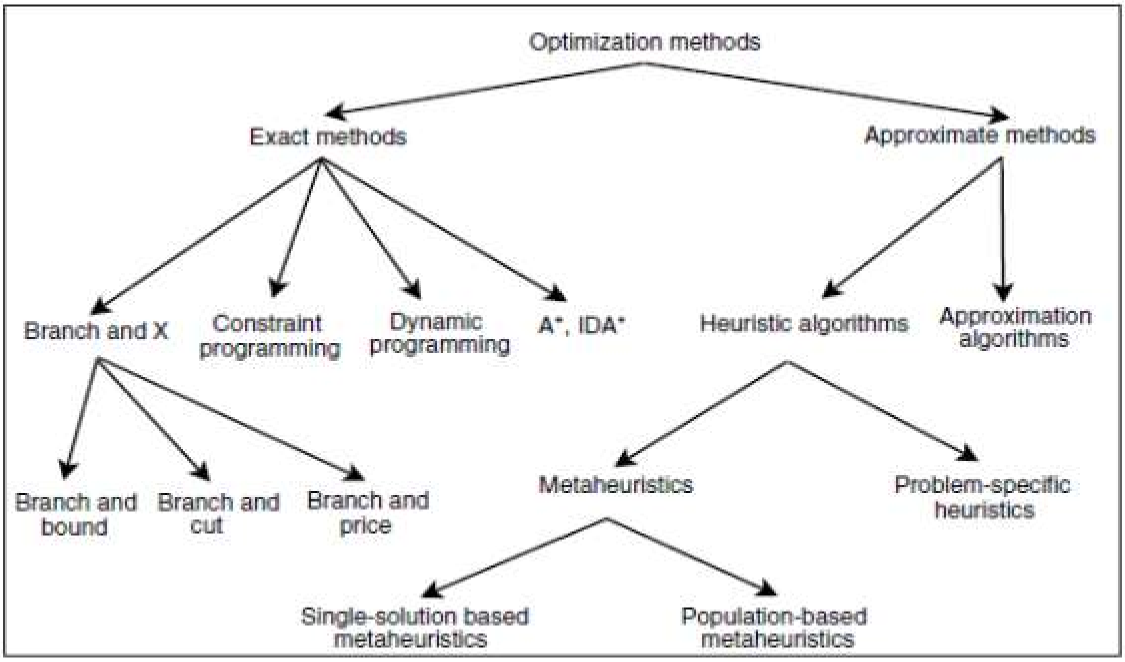
– Examples of metaheuristics algorithms:

– The evolutionary algorithms

–The tabu search method

– The ant colony optimization

– The simulated annealing method



Some Metaheuristic Classifications

1. Nature inspired versus non-nature inspired: many metaheuristics are inspired by natural processes: evolutionary algorithms and artificial immune systems from biology; ants, bees colonies, and particle swarm optimization from swarm intelligence into different species (social sciences); and simulated annealing from physics.

2. Memory usage versus memoryless methods: Some metaheuristic algorithms are memoryless; that is, no information extracted dynamically is used during the search. Some representatives of this class are local search, GRASP, and simulated annealing. While other metaheuristics use a memory that contains some information extracted online during the search. For instance, short-term and long-term memories in tabu search.

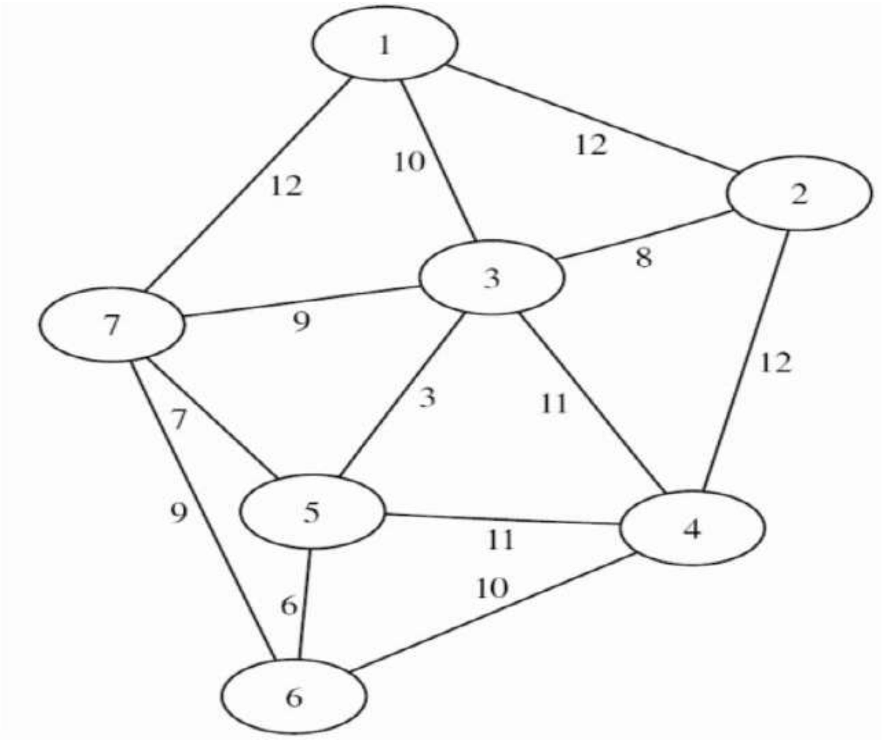
3. Deterministic versus stochastic: A deterministic metaheuristic solves an optimization problem by making deterministic decisions (e.g., local search, tabu search). In stochastic metaheuristics, some random rules are applied during the search (e.g., simulated annealing, evolutionary algorithms).

4. Population-based search versus single-solution based search: Single-solution based algorithms (e.g., local search, simulated annealing) manipulate and transform a single solution during the search while in population-based algorithms (e.g., particle swarm, evolutionary algorithms) a whole population of solutions is evolved. These two families have complementary characteristics:

single-solution based metaheuristics are exploitation oriented; they have the power to intensify the search in local regions. Population based metaheuristics are exploration oriented; they allow a better diversification in the whole search space.

5. Iterative versus greedy: In iterative algorithms, starting with a complete solution (or population of solutions) and transform it at each iteration using some search operators. Greedy algorithms start from an empty solution, and at each step a decision variable of the problem is assigned until a complete solution is obtained. Most of the metaheuristics are iterative algorithms.

Travelling salesman Problem (TSP)



- We will use the travelling salesmen problem (TSP) on the graph as an example problem for the metaheurisics discussed. Travelling Salesman Problem(TSP):
- A salesman spends his time visiting n cities (or nodes) cyclically. Starting from the home city, the salesman wishes to determine which route to follow to visit each city exactly once before returning to the home city so as to minimize the total distance of the tour.
- The difficulty of the travelling salesman problem increases rapidly as the number of cities increases. For a problem with n cities and a link between every pair of cities, the number of feasible routes to be considered is (n-1)!/2. Due to enormous difficulty in solving the TSP, heuristic methods guided by the metaheuristics, address such problems.
- Heuristic methods involve sequence of feasible trial solutions, where each solution is obtained by making certain adjustment in current trial solution.
- Sub tour Reversal: Adjusting a sequence of cities visited in the current solution by selecting a subsequence of the cities and simply reversing the order.

Example

- Initial trial solution is the following sequence of cities

visited: 1-2-3-4-5-6-7-1

- with total distance = 69.
- While reversing the sequence 3-4 , we obtain new trial
- solution: 1-2-4-3-5-6-7-1
- with total distance = 65.
- Neighbors: We say 2 tours/solutions/cycles are neighbors if we can transform one to the other by a subtour reversal.
- Degree of Neighbor: The degree of a neighbor A to B equals the minimum number of sub tour reversals required to get from A to B.

Informed Search

–Use evaluation function f(n)

–More efficient

–informed Heuristic Search Examples: A*, D*, etc.

informed Heuristic Search: A*

Notation

•n →node/state

•c(n1,n2)→the length of an edge connecting between n1and n2

•b(n1) = n2→backpointerof a node n1to a node n2.

•Evaluation function, f(n) = g(n) + h(n)

•Operating cost function, g(n)

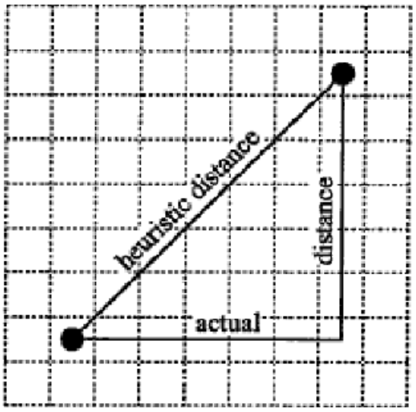
–the length of the shortest path found so far from the starting position to n

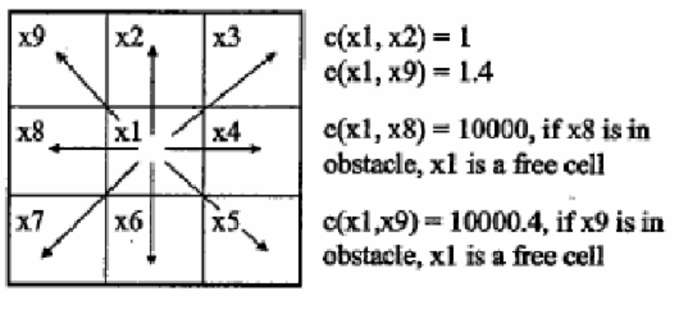
•Heuristic function, h(n)

–heuristic evaluation of the length of the shortest path from *n* to the goal

–Admissible → If *h* (*n*) is *admissible* (optimistic – always smaller or equal to the length of the shortest path from *n* to the goal), A* finds the shortest path.)

- The agent’s environment is divided into squares, some of them impassable.
- The agent can move up, down, left and right.
- The distance between adjacent squares is 1.
- *h* (*n*) is the Manhattan distance from *n* to the goal.



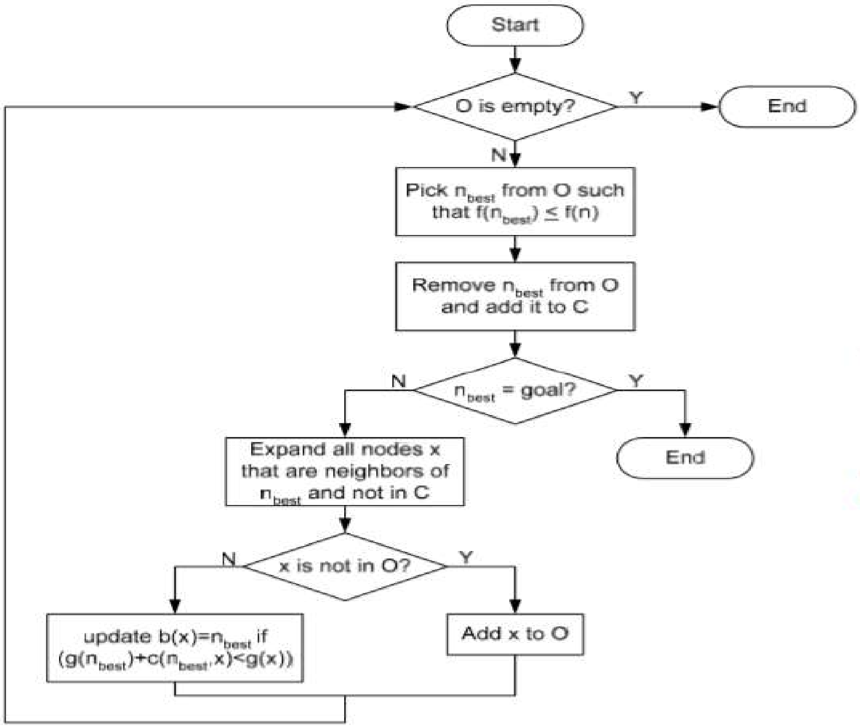

x9
x2
x3
x8
x1
x4
x7
x6
x5
c(x1, x2) = 1
c(x1, x9) = 1.4
c(x1, x8) = 10000, if x8 is in obstacle, x1 is a free cell
c(x1,x9) = 10000.4, if x9 is in obstacle, x1 is a free cell

Cost on a grid

A*: Algorithm

The search requires 2 lists to store information about nodes

1) Open list (O)stores nodes for expansions
2) Closed list (C)stores nodes which we have explored


Start
O is empty?
Y
End
N
Pick n_{best} from O such that $f(n_{best}) \leq f(n)$
Remove n_{best} from O and add it to C
n_{best} = goal?
N
Y
End
Expand all nodes x that are neighbors of n_{best} and not in C
x is not in O?
N
Y
update $b(x)=n_{best}$ if $(g(n_{best})+c(n_{best},x)<g(x))$
Add x to O

A* Algorithm Properties

1) Admissibility

- Admissibility means that h(n) is less than or equal to the cost of the minimal path from n to the goal.
- the admissibility should satisfy these two conditions: 1- $h(n) \leq h^*(n)$ 2- $g(n) \geq g^*(n)$

2) Monotonicity (consistency)

A heuristic function h is monotone if

a. For all state ni and nj , where nj is a descendant of ni h(ni)-h(nj) ≤ cost(ni,nj).

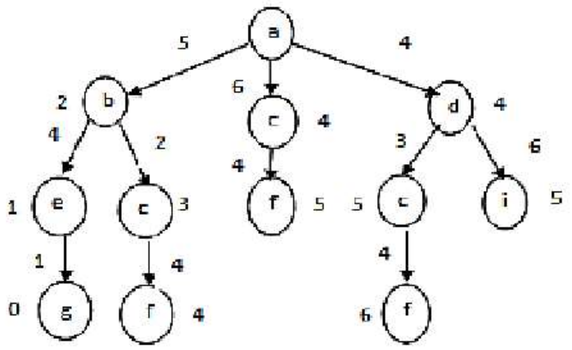
Where cost (ni,nj) is the actual cost of going from state ni to nj.

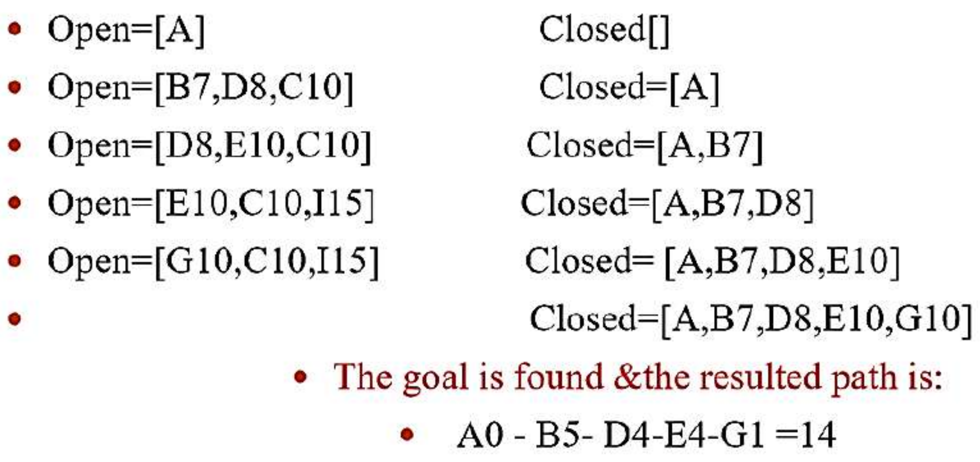
b. The heuristic evaluation of the goal state is zero , or h(goal)=0.

3) Informedness

For two A* heuristics h1 and h2 , if h1(n) ≤ h2(n), for all states n in the search space , heuristics h2 is said to be more informed than h1.

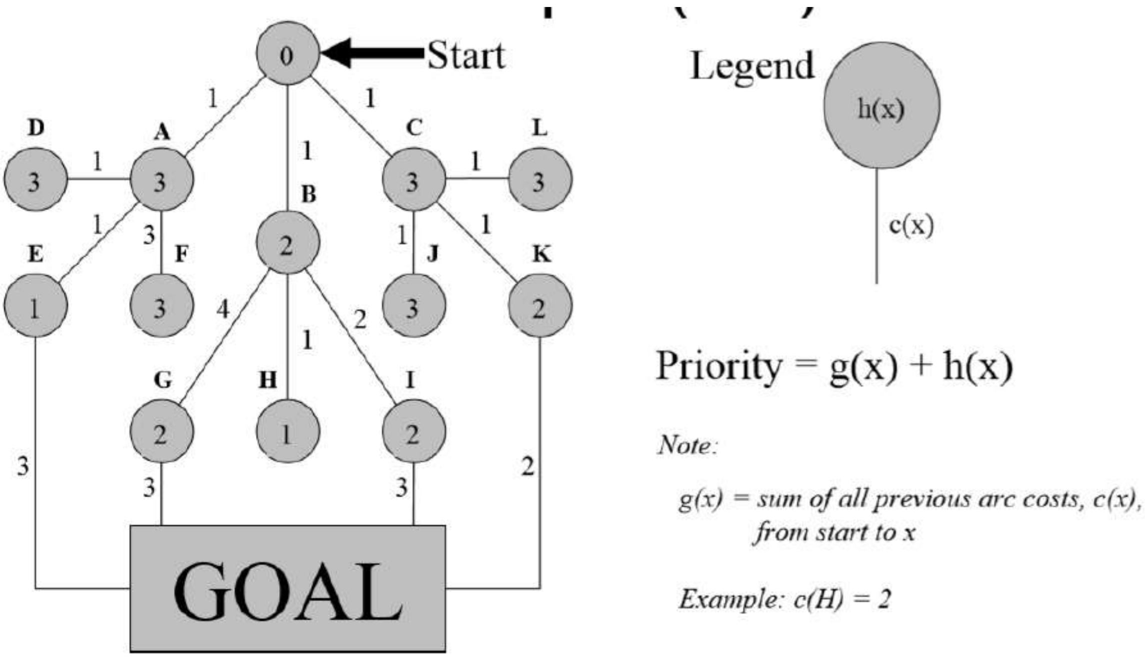
A* Example





- Open=[A]   Closed[]
- Open=[B7,D8,C10]   Closed=[A]
- Open=[D8,E10,C10]   Closed=[A,B7]
- Open=[E10,C10,I15]   Closed=[A,B7,D8]
- Open=[G10,C10,I15]   Closed= [A,B7,D8,E10]
- Closed=[A,B7,D8,E10,G10]
  - The goal is found &the resulted path is:
    - A0 - B5- D4-E4-G1 =14

A* Example

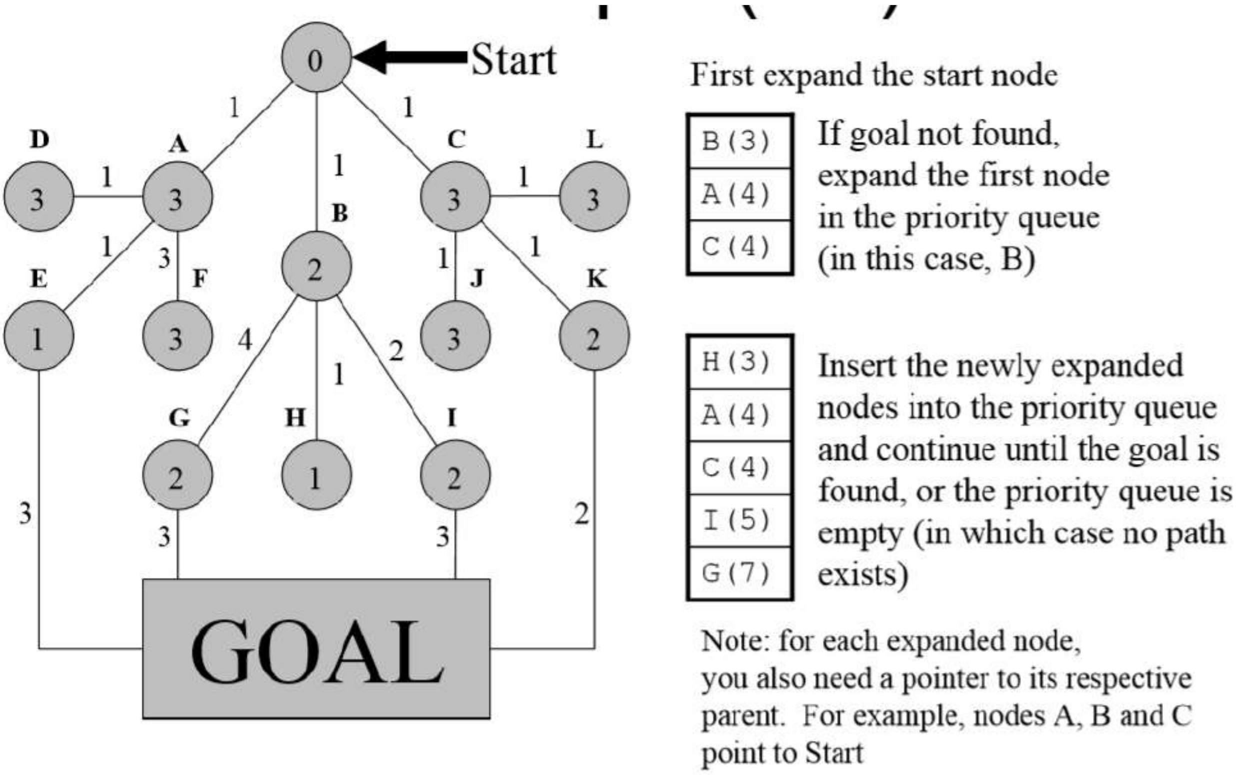


Priority = g(x) + h(x)

Note:

g(x) = sum of all previous arc costs, c(x), from start to x

Example: c(H) = 2



First expand the start node

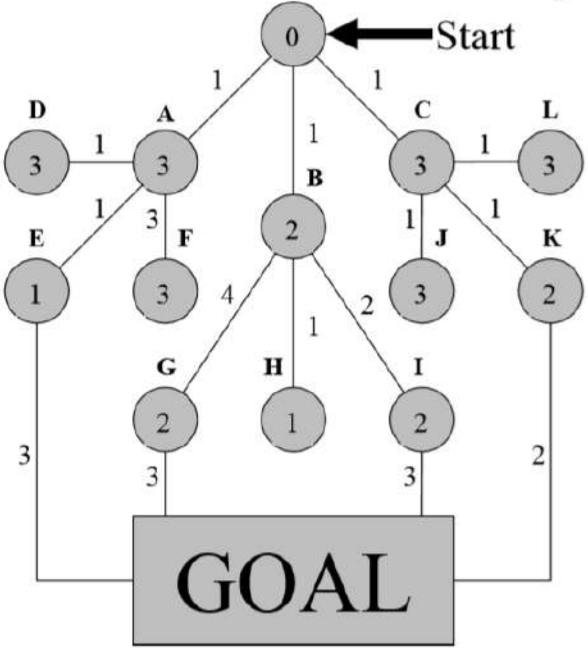
B(3)
A(4)
C(4)

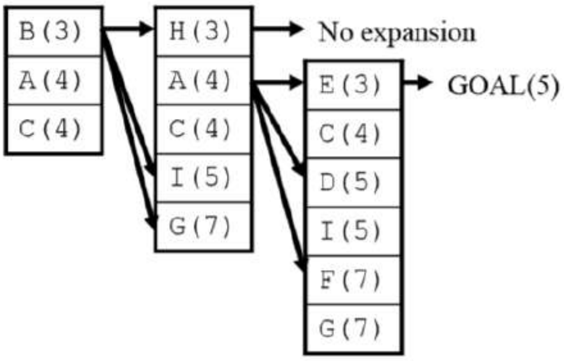
If goal not found, expand the first node in the priority queue (in this case, B)

H(3)
A(4)
C(4)
I(5)
G(7)

Insert the newly expanded nodes into the priority queue and continue until the goal is found, or the priority queue is empty (in which case no path exists)

Note: for each expanded node, you also need a pointer to its respective parent. For example, nodes A, B and C point to Start




B(3)
A(4)
C(4)

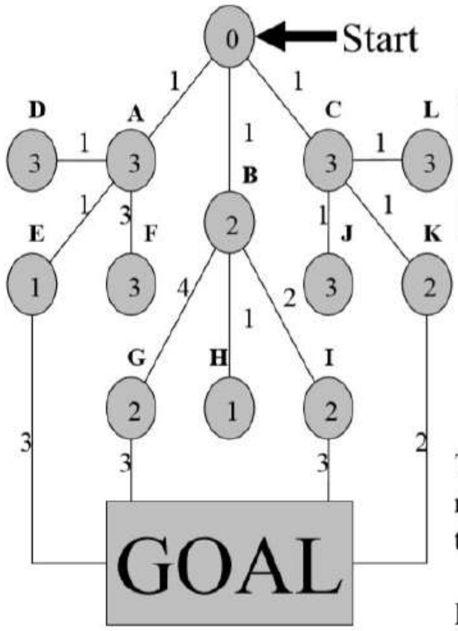
H(3) → No expansion
A(4)
C(4)
I(5)
G(7)

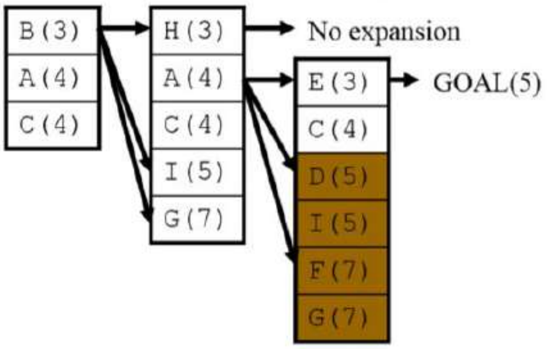
E(3) → GOAL(5)
C(4)
D(5)
I(5)
F(7)
G(7)

We've found a path to the goal:
Start => A => E => Goal
(from the pointers)

Are we done?

Example (4/5)



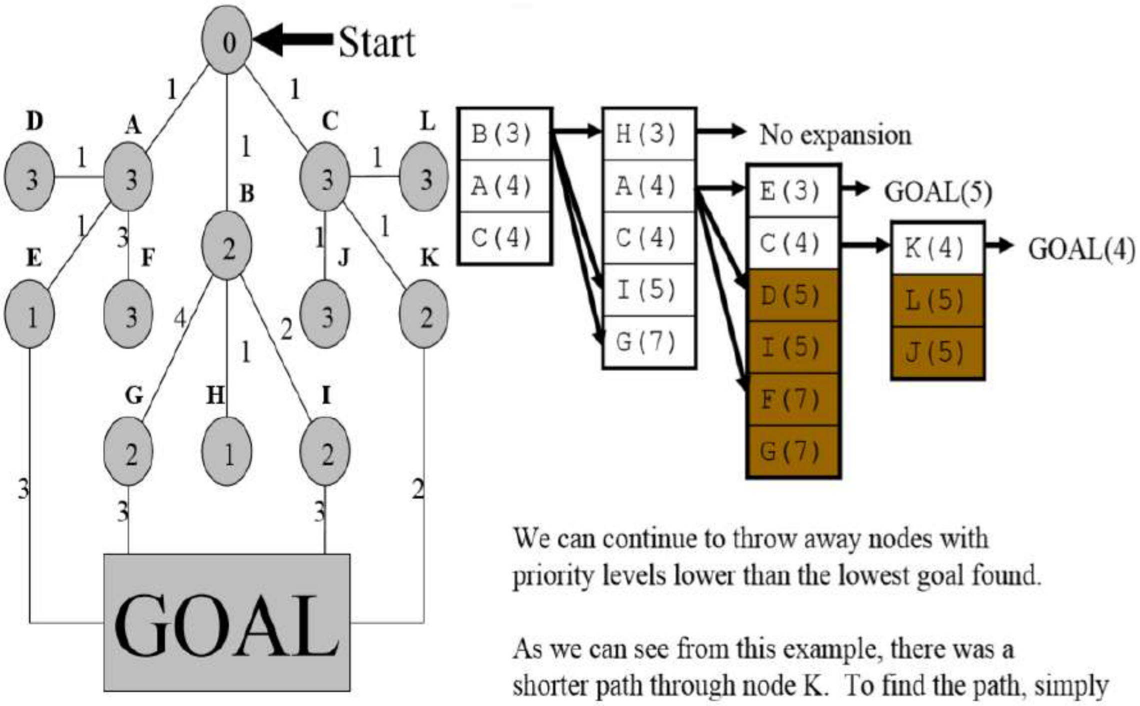

B(3)
A(4)
C(4)

H(3) → No expansion
A(4)
C(4)
I(5)
G(7)

E(3) → GOAL(5)
C(4)
D(5)
I(5)
F(7)
G(7)

There might be a shorter path, but assuming non-negative arc costs, nodes with a lower priority than the goal cannot yield a better path.

In this example, nodes with a priority greater than or equal to 5 can be pruned.



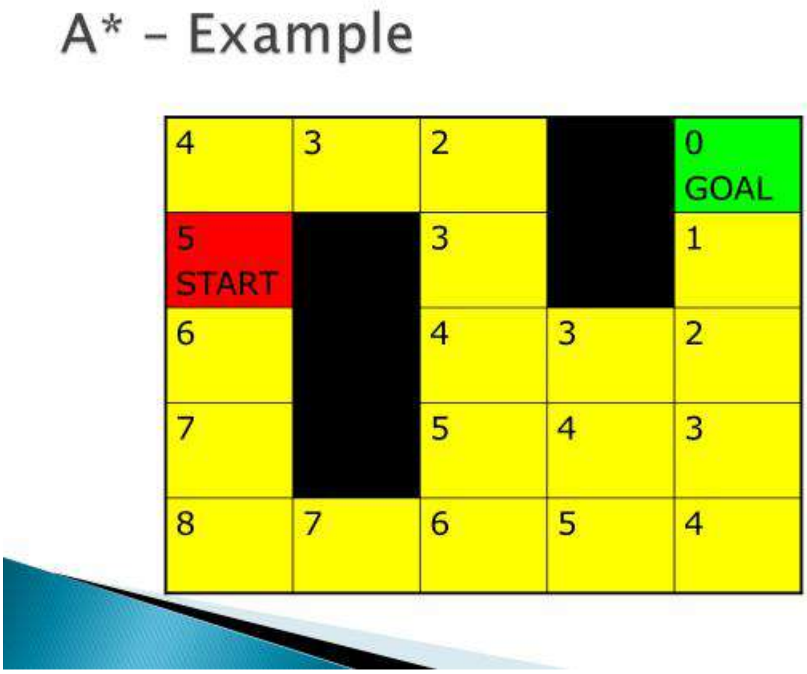
If the priority queue still wasn't empty, we would continue expanding while throwing away nodes with priority lower than 4.
(remember, lower numbers = higher priority)

We can continue to throw away nodes with priority levels lower than the lowest goal found.

As we can see from this example, there was a shorter path through node K. To find the path, simply follow the back pointers.

Therefore the path would be:
Start => C => K => Goal

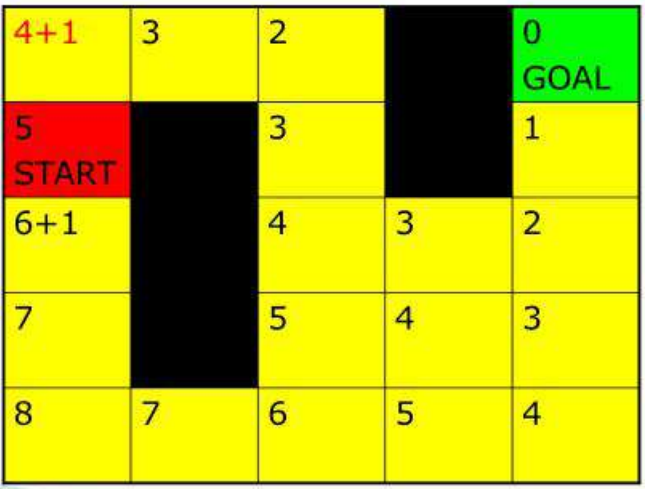
Example path planning using A*



A* – Example

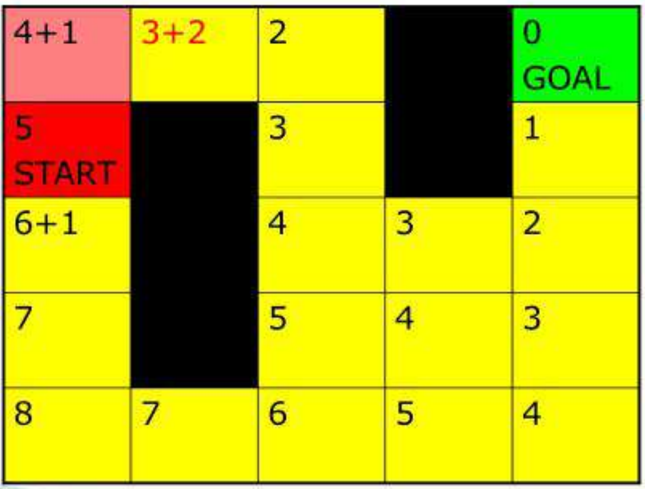
4	3	2	■	0 GOAL
5 START	■	3	■	1
6	■	4	3	2
7	■	5	4	3
8	7	6	5	4

A* – Example



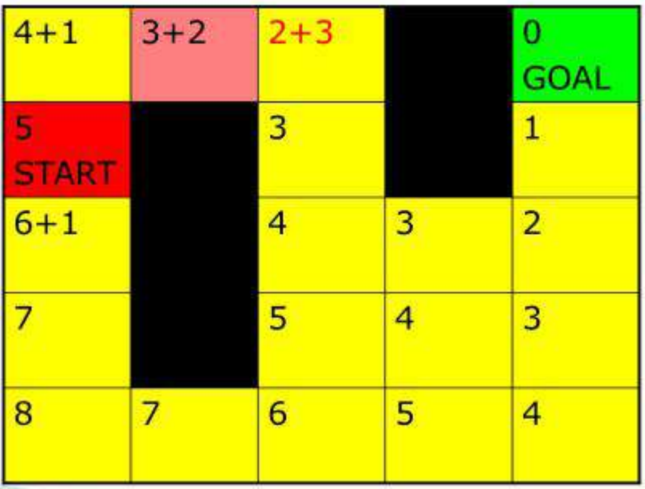
4+1	3	2	■	0 GOAL
5 START	■	3	■	1
6+1	■	4	3	2
7	■	5	4	3
8	7	6	5	4

A* – Example



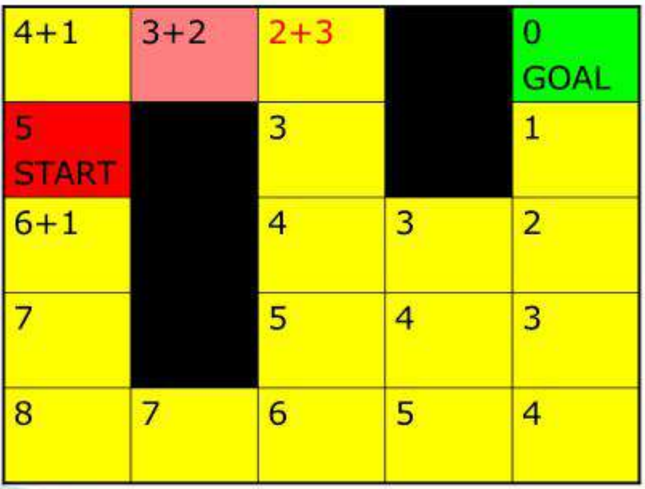
4+1	3+2	2	■	0 GOAL
5 START	■	3	■	1
6+1	■	4	3	2
7	■	5	4	3
8	7	6	5	4

A* – Example



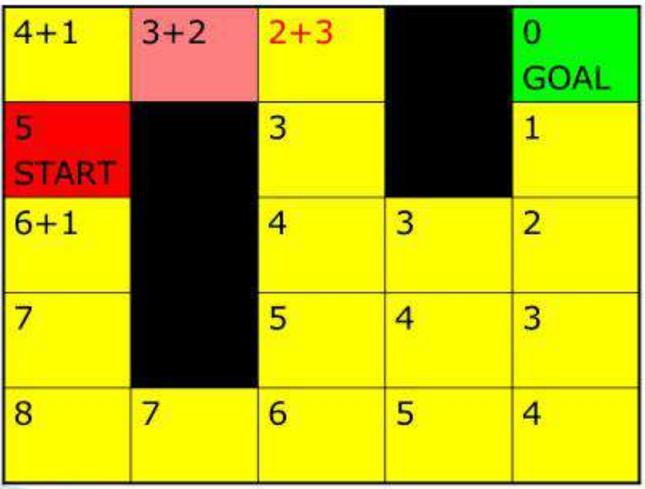
4+1	3+2	2+3		0 GOAL
5 START		3		1
6+1		4	3	2
7		5	4	3
8	7	6	5	4

A* – Example



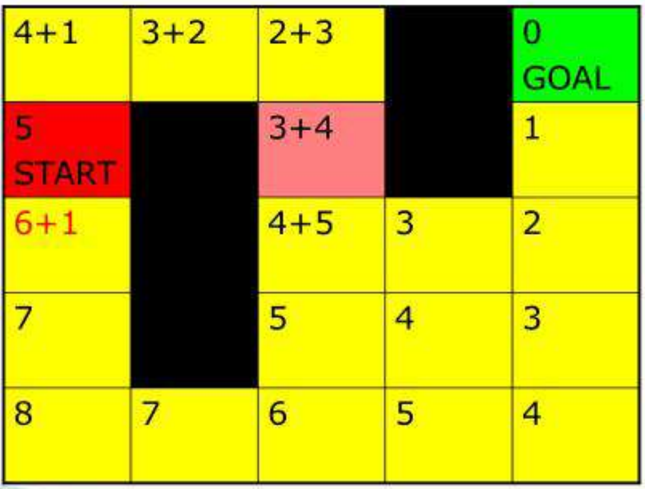
4+1	3+2	2+3		0 GOAL
5 START		3		1
6+1		4	3	2
7		5	4	3
8	7	6	5	4

A* – Example



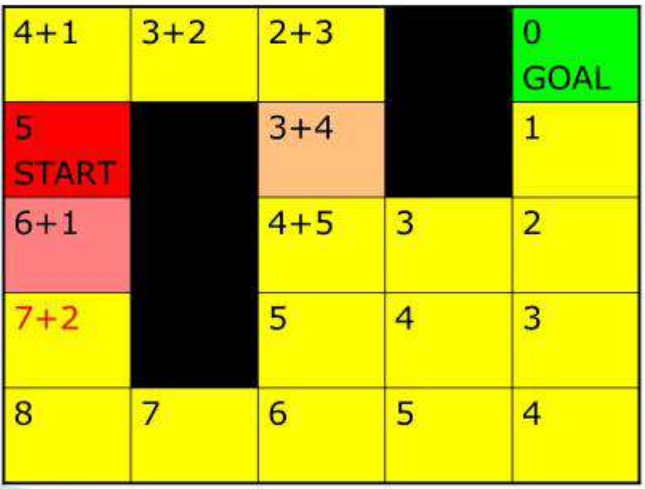
4+1	3+2	2+3	■	0 GOAL
5 START	■	3	■	1
6+1	■	4	3	2
7	■	5	4	3
8	7	6	5	4

A* – Example



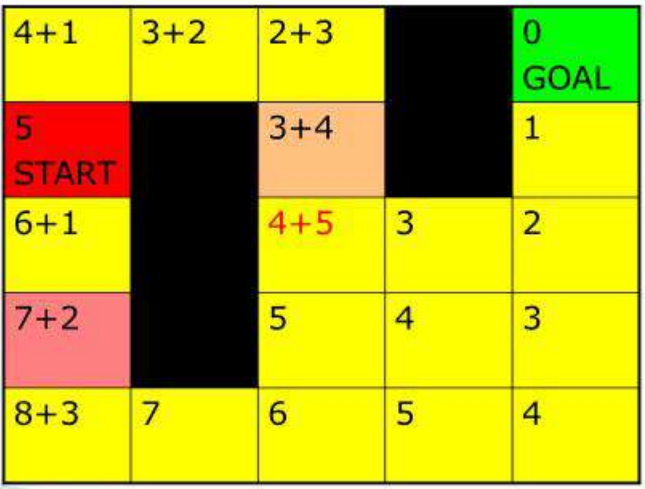
4+1	3+2	2+3	■	0 GOAL
5 START	■	3+4	■	1
6+1	■	4+5	3	2
7	■	5	4	3
8	7	6	5	4

A* – Example



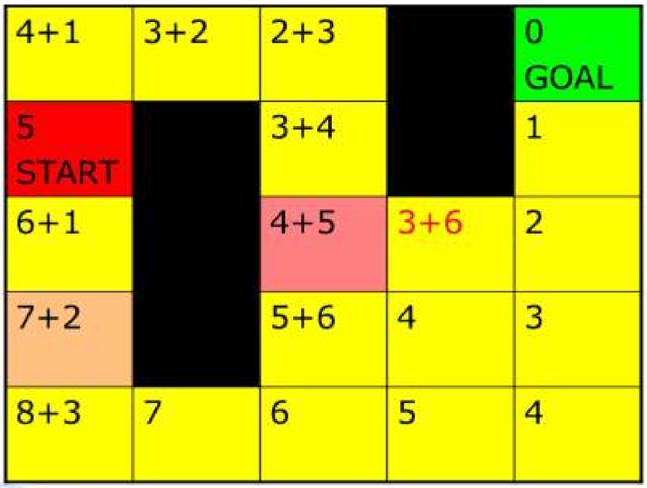
4+1	3+2	2+3	■	0 GOAL
5 START	■	3+4	■	1
6+1	■	4+5	3	2
7+2	■	5	4	3
8	7	6	5	4

A* – Example



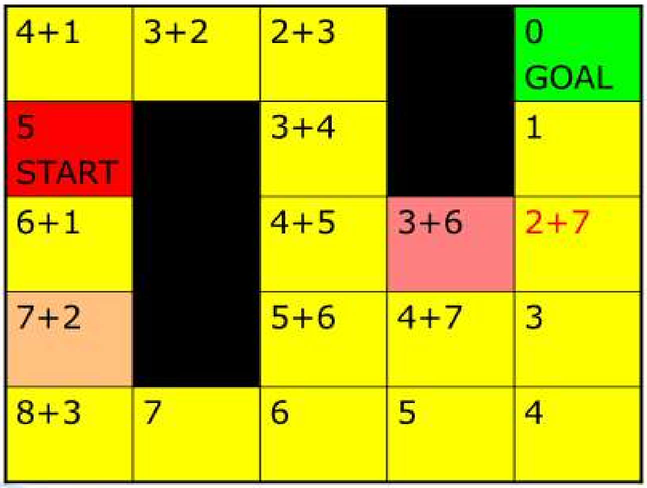
4+1	3+2	2+3	■	0 GOAL
5 START	■	3+4	■	1
6+1	■	4+5	3	2
7+2	■	5	4	3
8+3	7	6	5	4

A* – Example



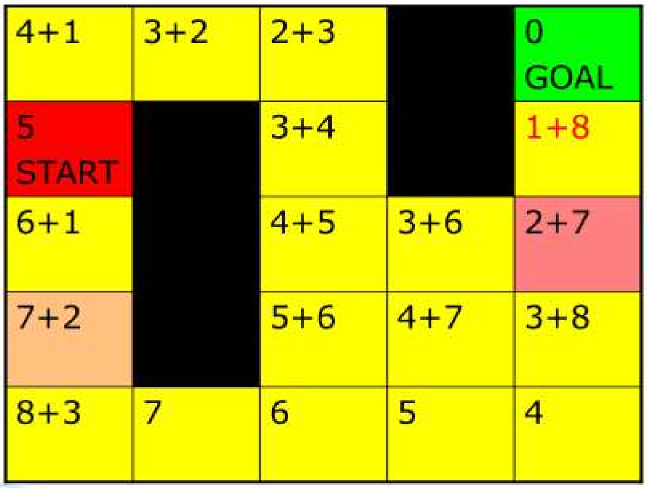
4+1	3+2	2+3	■	0 GOAL
5 START	■	3+4	■	1
6+1	■	4+5	3+6	2
7+2	■	5+6	4	3
8+3	7	6	5	4

A* – Example



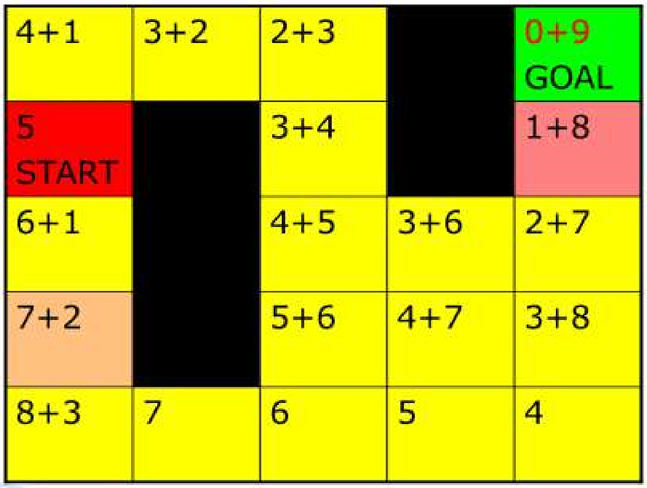
4+1	3+2	2+3	■	0 GOAL
5 START	■	3+4	■	1
6+1	■	4+5	3+6	2+7
7+2	■	5+6	4+7	3
8+3	7	6	5	4

A* – Example



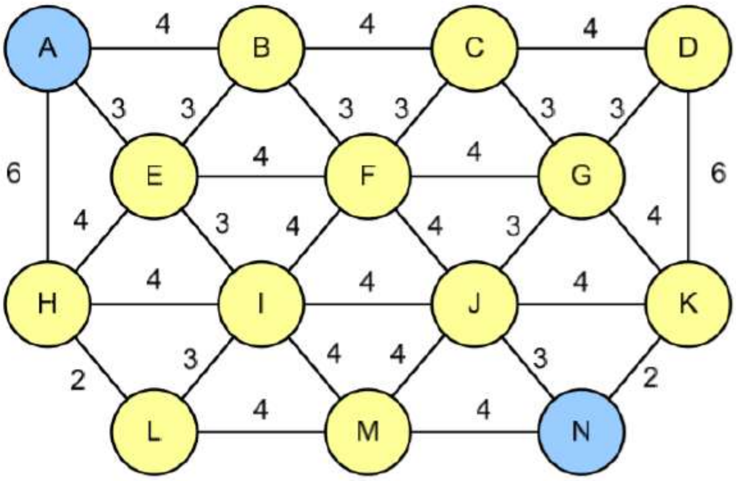
4+1	3+2	2+3	■	0 GOAL
5 START	■	3+4	■	1+8
6+1	■	4+5	3+6	2+7
7+2	■	5+6	4+7	3+8
8+3	7	6	5	4

A* – Example



4+1	3+2	2+3	■	0+9 GOAL
5 START	■	3+4	■	1+8
6+1	■	4+5	3+6	2+7
7+2	■	5+6	4+7	3+8
8+3	7	6	5	4

H.W.



Heuristics

A = 14	H = 8
B = 10	I = 5
C = 8	J = 2
D = 6	K = 2
E = 8	L = 6
F = 7	M = 2
G = 6	N = 0

A*: Performance Analysis

- Complete provided the finite boundary condition and that every path cost is greater than some positive constant δ
- Optimal in terms of the path cost
- Memory inefficient →IDA*
- Exponential growth of search space with respect to the length of solution

D* Algorithm

In an unknown area (or known only partially) the task is to reach a target position of given coordinates. Costs of transitions and obstacles are learned on-the-go during the path is traversed (examples: real mobile robots, artificial players in computer games).

Stentz's algorithm (1994) — properties

- Also known as D∗, with an intention the name is understood as dynamic A∗.
- Despite its name, carried out in a manner closer to Dijkstra's algorithm.
- Algorithm performs optimal behavior taking into account information learned so far.
- Works iteratively — the planned path is derived multiple times.
- The first run of the algorithm is a backward versions of Dijkstra's algorithm — we built a queue of states by going from the goal towards the start point.
- During the actual traversal of the path, when a discrepancy between the knowledge so-far occurs, we update the path.
- States in the queue are allowed to change their costs multiple times and to enter the queue multiple times.
- The algorithm is more effective then multiple executions of Dijkstra's algorithm from scratch.

Sets of actions and states

- Let A denote set of possible actions. In particular, for regular grid of squares: $A = \{\uparrow, \rightarrow, \downarrow, \leftarrow\}$.
- Let X denote set of states. For regular grid of squares:

$$X = \{xij\},$$

where i is row index, j is column index.

- By A(x) set of actions possible for state x shall be denoted.

Executing actions

Let t(x,a) (transition) denote a function transiting given state x via action a into a new state x0, i.e.

$t(x,a) = x0$.

E.g. for grid of squares $t(x25, \rightarrow) = x26$.

Formally, t is a function $t: X \times A \rightarrow X$.

Costs of transitions

Let c(x,a) (costs of transition) denote a function of taken cost, which has to be taken when executing in x an action a.

Formally, $c: X \times A \rightarrow R+ \cup \{\infty\}$.

If in x execution of a is impossible (obstacle or map border), then $c(x,a) = \infty$.

Such form of c allows for general map representation, where transitions to certain state, but from different directions, can have different costs.

If, for grid of squares, we want to identify all transition costs to the same state xij, then:

$c(xi-1,j, \downarrow) = c(xi+1,j, \uparrow) = c(xi,j-1, \rightarrow) = c(xi,j+1, \leftarrow) = map(i,j).$

for all i, j.

Initial assumptions

In the most pessimistic case we assume complete lack of map knowledge, except for start and goal coordinates.

If we assume that transitions untroubled by anything cost 1 unit, then in the case of complete unawareness, one shall impose:

$\forall x,a\ c(x,a) = 1.$

In particular, we assume that positions of map borders are not known as well.

Functions/quantities used in the algorithm

Let **gcurrent(x)** denote currently known cost of transition from x to goal G. Let **gvia(x,x0)** denote currently known cost of transition from x to goal

G, when traveling via x0. Function gvia shall in fact be considered only for such x, x0 which are direct neighbors. Let a be an action, s.t. t(x,a) = x0. Then:

gvia(x,x0) = c(x,a) + gcurrent(x0). (1)

Let Q denote the queue of states kept in algorithm (analogically to Dijkstra’s or A∗ algorithms).

Let V denote the map of visited states.

Let gbest(x) denote the best (the lowest) known cost value for x during its lifetime in Q. It is known that:

gbest(x) 6 gcurrent(x). (2)

Q is ordered according to gbest.

Temporary plan

Let p(x) denote a planned action currently assigned to be executed in state x. The algorithm calculates (multiple times) a temporary plan, i.e. a certain sequence of actions

p1,p2, . . . ,pk,pk+1, . . . ,

which allow to travel from current start state to goal according to current knowledge of transition costs c. Therefore, a sequence of states is in the same time derived:

x1,x2, . . . ,xk,xk+1, . . . ,

such that

xk+1 = t(xk,p(xk)).

A person (agent, robot) shall travel on in accordance with the plan, until he (it) experiences a discrepancy between known (assumed) transition cost and a true one.

Core issue of Stentz's algorithm

For certain x assume that there exists a which transits x into its neighbor x0. If we have that: gvia(x,x0) < gcurrent(x), then there is a chance that cost gcurrent(x) can be reduced.

Additionally, if:

gcurrent(x0) 6 gbest(x), then the cost gcurrent(x0) is for certain optimal in the light of information at disposal. When both conditions are met then gcurrent(x) is updated to gvia(x,x0) and p(x) is updated to a.

For certain x assume that there exists a which transits x into its neighbor x0. If we have that: gvia(x,x0) < gcurrent(x), then there is a chance that cost gcurrent(x) can be reduced.

Additionally, if:

gcurrent(x0) 6 gbest(x), then the cost gcurrent(x0) is for certain optimal in the light of information at disposal. When both conditions are met then gcurrent(x) is updated to gvia(x,x0) and p(x) is updated to a.

„Outer" algorithm

1 Initialize all gbest,gcurrent,gvia with zeros, and all p with voids. 2 Insert goal state G into queue Q. 3 In a loop, perform iteratively Stentz's algorithm, until as its result the start state S is returned. (at that moment Stentz's algorithm works equivalently to a backward Dijkstra's algorithm)

4 Main loop: 1 Carry out current plan p1,p2, . . . visiting sequence of states xk+1 = t(xk,pk), where x1 denotes current S.

1 If for certain xk it can be observed that executing pk would result in a cost greater than known c(xk,pk), then update c(xk,pk) to the true one, and assign: gvia(xk,xk+1) := c(xk,pk) + gcurrent(xk+1), gcurrent(xk) := gvia(xk,xk+1). Abort further execution of plan.

2 If xk = G then stop the algorithm. (stop condition) 3 Insert xk into Q. Memorize glast := gcurrent(xk). Set S := xk. 4 As long as Q is non-empty or until the condition gbest(x) > glast is not met for all x in Q:

1 Perform Stentz's algorithm.

1 Poll from Q the state x with the lowest gbest. 2 If gbest(x) < gcurrent(x) (it means x has increased its cost while being in Q, and if this cost could be reduced by traveling via some neighbor for which an optimal cost is known, then one should do so) 1 For all a ∈ A(x), s.t. c(x,a) < ∞, check for x0 = t(x,a) if: gvia(x,x0) < gcurrent(x) and gcurrent(x0) 6 gbest(x)? If so, then: 1 gcurrent(x) := gvia(x,x0). 2 p(x) := a.

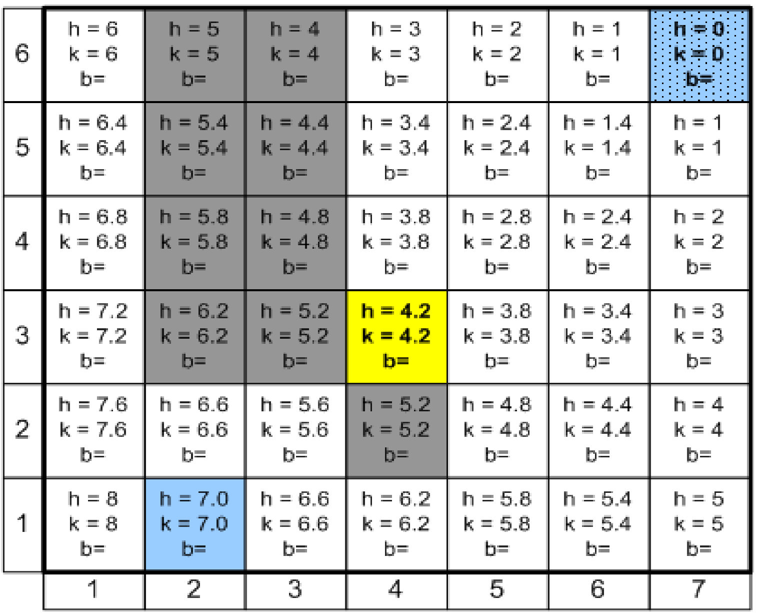
3 For all x0, s.t. there exists a0 ∈ A(x0) causing t(x0,a0) = x and c(x0,a0) < ∞: 1 gvia(x0,x) := c(x0,a0) + gcurrent(x). 2 If x0 is not in V then: 1 gcurrent(x0) := gvia(x0,x), gbest(x0) := gvia(x0,x). 2 p(x0) := a0. 3 Insert x0 into Q. 3 If cost for x0 seems to be incorrect because p(x0) = a0, but gvia(x0,x) , gcurrent(x0), then: 1 gcurrent(x0) := gvia(x0,x). 2 Insert x0 into Q.

. . .

. . .

3 (continuation of loop’s 3 body) 4 If p(x0) , a0 and gvia(x0,x) < gcurrent(x0) then: (it means that it is better to go from x0 via x than to use action p(x0)) 1 If gcurrent(x) = gbest(x) then: p(x0) := a0 and insert x0 into Q, because optimal cost for x is known. 2 Otherwise: gbest(x) := gcurrent(x) (if x < Q), and insert x into Q. 5 (avoiding cycles in p) If x0 ∈ V and x0 < Q, and p(x0) , a0 and gvia(x,x0) < gcurrent(x) and gcurrent(x) > gbest(x), then insert x0 into Q again. 4 Put x into V.

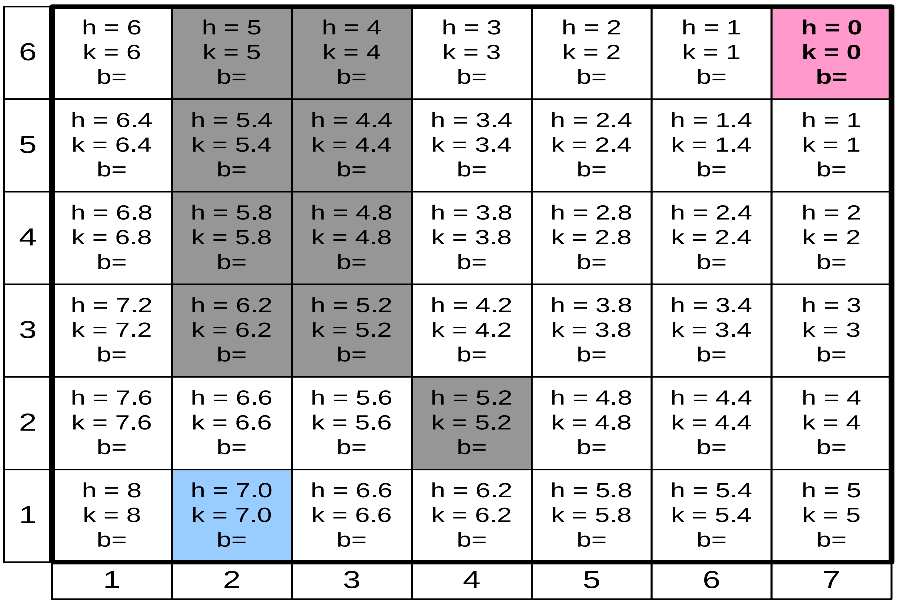
D*: Example (2/16)



6	h = 6 k = 6 b=	h = 5 k = 5 b=	h = 4 k = 4 b=	h = 3 k = 3 b=	h = 2 k = 2 b=	h = 1 k = 1 b=	**h = 0 k = 0 b=**
5	h = 6.4 k = 6.4 b=	h = 5.4 k = 5.4 b=	h = 4.4 k = 4.4 b=	h = 3.4 k = 3.4 b=	h = 2.4 k = 2.4 b=	h = 1.4 k = 1.4 b=	h = 1 k = 1 b=
4	h = 6.8 k = 6.8 b=	h = 5.8 k = 5.8 b=	h = 4.8 k = 4.8 b=	h = 3.8 k = 3.8 b=	h = 2.8 k = 2.8 b=	h = 2.4 k = 2.4 b=	h = 2 k = 2 b=
3	h = 7.2 k = 7.2 b=	h = 6.2 k = 6.2 b=	h = 5.2 k = 5.2 b=	**h = 4.2 k = 4.2 b=**	h = 3.8 k = 3.8 b=	h = 3.4 k = 3.4 b=	h = 3 k = 3 b=
2	h = 7.6 k = 7.6 b=	h = 6.6 k = 6.6 b=	h = 5.6 k = 5.6 b=	h = 5.2 k = 5.2 b=	h = 4.8 k = 4.8 b=	h = 4.4 k = 4.4 b=	h = 4 k = 4 b=
1	h = 8 k = 8 b=	h = 7.0 k = 7.0 b=	h = 6.6 k = 6.6 b=	h = 6.2 k = 6.2 b=	h = 5.8 k = 5.8 b=	h = 5.4 k = 5.4 b=	h = 5 k = 5 b=
	1	2	3	4	5	6	7

- □ Clear
- ▨ Obstacle
- ▦ Goal
- 🟦 Start
- 🟨 Gate

D*: Example (3/16)



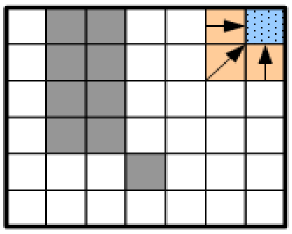
6	h = 6 k = 6 b=	h = 5 k = 5 b=	h = 4 k = 4 b=	h = 3 k = 3 b=	h = 2 k = 2 b=	h = 1 k = 1 b=	**h = 0 k = 0 b=**
5	h = 6.4 k = 6.4 b=	h = 5.4 k = 5.4 b=	h = 4.4 k = 4.4 b=	h = 3.4 k = 3.4 b=	h = 2.4 k = 2.4 b=	h = 1.4 k = 1.4 b=	h = 1 k = 1 b=
4	h = 6.8 k = 6.8 b=	h = 5.8 k = 5.8 b=	h = 4.8 k = 4.8 b=	h = 3.8 k = 3.8 b=	h = 2.8 k = 2.8 b=	h = 2.4 k = 2.4 b=	h = 2 k = 2 b=
3	h = 7.2 k = 7.2 b=	h = 6.2 k = 6.2 b=	h = 5.2 k = 5.2 b=	h = 4.2 k = 4.2 b=	h = 3.8 k = 3.8 b=	h = 3.4 k = 3.4 b=	h = 3 k = 3 b=
2	h = 7.6 k = 7.6 b=	h = 6.6 k = 6.6 b=	h = 5.6 k = 5.6 b=	h = 5.2 k = 5.2 b=	h = 4.8 k = 4.8 b=	h = 4.4 k = 4.4 b=	h = 4 k = 4 b=
1	h = 8 k = 8 b=	h = 7.0 k = 7.0 b=	h = 6.6 k = 6.6 b=	h = 6.2 k = 6.2 b=	h = 5.8 k = 5.8 b=	h = 5.4 k = 5.4 b=	h = 5 k = 5 b=
	1	2	3	4	5	6	7

State	k
(7,6)	0

D*: Example (4/16)

6	h = 6 k = 6 b=	h = 5 k = 5 b=	h = 4 k = 4 b=	h = 3 k = 3 b=	h = 2 k = 2 b=	h = 1 k = 1 b=(7,6)	**h = 0** **k = 0** **b=**
5	h = 6.4 k = 6.4 b=	h = 5.4 k = 5.4 b=	h = 4.4 k = 4.4 b=	h = 3.4 k = 3.4 b=	h = 2.4 k = 2.4 b=	h = 1.4 k = 1.4 b=(7,6)	h = 1 k = 1 b=(7,6)
4	h = 6.8 k = 6.8 b=	h = 5.8 k = 5.8 b=	h = 4.8 k = 4.8 b=	h = 3.8 k = 3.8 b=	h = 2.8 k = 2.8 b=	h = 2.4 k = 2.4 b=	h = 2 k = 2 b=
3	h = 7.2 k = 7.2 b=	h = 6.2 k = 6.2 b=	h = 5.2 k = 5.2 b=	h = 4.2 k = 4.2 b=	h = 3.8 k = 3.8 b=	h = 3.4 k = 3.4 b=	h = 3 k = 3 b=
2	h = 7.6 k = 7.6 b=	h = 6.6 k = 6.6 b=	h = 5.6 k = 5.6 b=	h = 5.2 k = 5.2 b=	h = 4.8 k = 4.8 b=	h = 4.4 k = 4.4 b=	h = 4 k = 4 b=
1	h = 8 k = 8 b=	h = 7.0 k = 7.0 b=	h = 6.6 k = 6.6 b=	h = 6.2 k = 6.2 b=	h = 5.8 k = 5.8 b=	h = 5.4 k = 5.4 b=	h = 5 k = 5 b=
	1	2	3	4	5	6	7

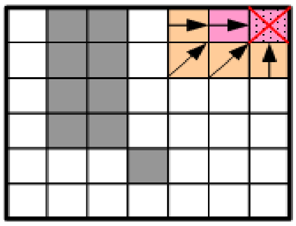
State	k
(6,6)	1
(7,5)	1
(6,5)	1.4



D*: Example (5/16)

6	h = 6 k = 6 b=	h = 5 k = 5 b=	h = 4 k = 4 b=	h = 3 k = 3 b=	h = 2 k = 2 b=(6,6)	h = 1 k = 1 b=(7,6)	~~h = 0~~ ~~k = 0~~ ~~b=~~
5	h = 6.4 k = 6.4 b=	h = 5.4 k = 5.4 b=	h = 4.4 k = 4.4 b=	h = 3.4 k = 3.4 b=	h = 2.4 k = 2.4 b=(6,6)	h = 1.4 k = 1.4 b=(7,6)	h = 1 k = 1 b=(7,6)
4	h = 6.8 k = 6.8 b=	h = 5.8 k = 5.8 b=	h = 4.8 k = 4.8 b=	h = 3.8 k = 3.8 b=	h = 2.8 k = 2.8 b=	h = 2.4 k = 2.4 b=	h = 2 k = 2 b=
3	h = 7.2 k = 7.2 b=	h = 6.2 k = 6.2 b=	h = 5.2 k = 5.2 b=	h = 4.2 k = 4.2 b=	h = 3.8 k = 3.8 b=	h = 3.4 k = 3.4 b=	h = 3 k = 3 b=
2	h = 7.6 k = 7.6 b=	h = 6.6 k = 6.6 b=	h = 5.6 k = 5.6 b=	h = 5.2 k = 5.2 b=	h = 4.8 k = 4.8 b=	h = 4.4 k = 4.4 b=	h = 4 k = 4 b=
1	h = 8 k = 8 b=	h = 7.0 k = 7.0 b=	h = 6.6 k = 6.6 b=	h = 6.2 k = 6.2 b=	h = 5.8 k = 5.8 b=	h = 5.4 k = 5.4 b=	h = 5 k = 5 b=
	1	2	3	4	5	6	7

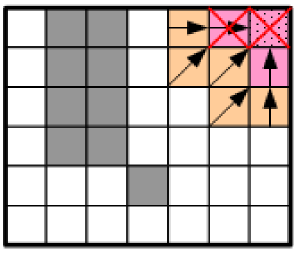
State	k
(7,5)	1
(6,5)	1.4
(5,6)	2
(5,5)	2.4



D*: Example (6/16)

6	h = 6 k = 6 b=	h = 5 k = 5 b=	h = 4 k = 4 b=	h = 3 k = 3 b=	h = 2 k = 2 b=(6,6)	h = 1 k = 1 b=(7,6)	h = 0 k = 0 b=
5	h = 6.4 k = 6.4 b=	h = 5.4 k = 5.4 b=	h = 4.4 k = 4.4 b=	h = 3.4 k = 3.4 b=	h = 2.4 k = 2.4 b=(6,6)	h = 1.4 k = 1.4 b=(7,6)	h = 1 k = 1 b=(7,6)
4	h = 6.8 k = 6.8 b=	h = 5.8 k = 5.8 b=	h = 4.8 k = 4.8 b=	h = 3.8 k = 3.8 b=	h = 2.8 k = 2.8 b=	h = 2.4 k = 2.4 b=(7,5)	h = 2 k = 2 b=(7,5)
3	h = 7.2 k = 7.2 b=	h = 6.2 k = 6.2 b=	h = 5.2 k = 5.2 b=	h = 4.2 k = 4.2 b=	h = 3.8 k = 3.8 b=	h = 3.4 k = 3.4 b=	h = 3 k = 3 b=
2	h = 7.6 k = 7.6 b=	h = 6.6 k = 6.6 b=	h = 5.6 k = 5.6 b=	h = 5.2 k = 5.2 b=	h = 4.8 k = 4.8 b=	h = 4.4 k = 4.4 b=	h = 4 k = 4 b=
1	h = 8 k = 8 b=	h = 7.0 k = 7.0 b=	h = 6.6 k = 6.6 b=	h = 6.2 k = 6.2 b=	h = 5.8 k = 5.8 b=	h = 5.4 k = 5.4 b=	h = 5 k = 5 b=
	1	2	3	4	5	6	7

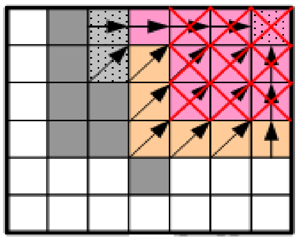
State	k
(6,5)	1.4
(5,6)	2
(7,4)	2
(6,4)	2.4
(5,5)	2.4



D*: Example (7/16)

6	h = 6 k = 6 b=	h = 5 k = 5 b=	h = 10003 k = 4 b=(4,6)	h = 3 k = 3 b=(5,6)	h = 2 k = 2 b=(6,6)	h = 1 k = 1 b=(7,6)	h = 0 k = 0 b=
5	h = 6.4 k = 6.4 b=	h = 5.4 k = 5.4 b=	h = 10003.4 k = 4.4 b=(4,6)	h = 3.4 k = 3.4 b=(5,6)	h = 2.4 k = 2.4 b=(6,6)	h = 1.4 k = 1.4 b=(7,6)	h = 1 k = 1 b=(7,6)
4	h = 6.8 k = 6.8 b=	h = 5.8 k = 5.8 b=	h = 4.8 k = 4.8 b=	h = 3.8 k = 3.8 b=(5,5)	h = 2.8 k = 2.8 b=(6,5)	h = 2.4 k = 2.4 b=(7,5)	h = 2 k = 2 b=(7,5)
3	h = 7.2 k = 7.2 b=	h = 6.2 k = 6.2 b=	h = 5.2 k = 5.2 b=	h = 4.2 k = 4.2 b=(5,4)	h = 3.8 k = 3.8 b=(6,4)	h = 3.4 k = 3.4 b=(7,4)	h = 3 k = 3 b=(7,4)
2	h = 7.6 k = 7.6 b=	h = 6.6 k = 6.6 b=	h = 5.6 k = 5.6 b=	h = 5.2 k = 5.2 b=	h = 4.8 k = 4.8 b=	h = 4.4 k = 4.4 b=	h = 4 k = 4 b=
1	h = 8 k = 8 b=	h = 7.0 k = 7.0 b=	h = 6.6 k = 6.6 b=	h = 6.2 k = 6.2 b=	h = 5.8 k = 5.8 b=	h = 5.4 k = 5.4 b=	h = 5 k = 5 b=
	1	2	3	4	5	6	7

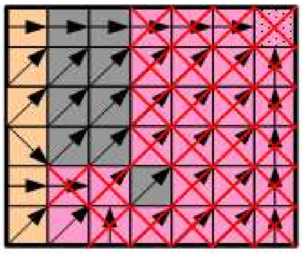
State	k
(7,3)	3
(6,3)	3.4
(4,5)	3.4
(5,3)	3.8
(4,4)	3.8
(3,6)	4
(4,3)	4.2
(3,5)	4.4



D*: Example (8/16)

6	h = 20004 k = 6 b=(2,6)	h = 10004 k = 5 b=(3,6)	h = 10003 k = 4 b=(4,6)	h = 3 k = 3 b=(5,6)	h = 2 k = 2 b=(6,6)	h = 1 k = 1 b=(7,6)	h = 0 k = 0 b=
5	h = 20004.4 k = 6.4 b=(2,6)	h = 10004.4 k = 5.4 b=(3,6)	h = 10003.4 k = 4.4 b=(4,6)	h = 3.4 k = 3.4 b=(5,6)	h = 2.4 k = 2.4 b=(6,6)	h = 1.4 k = 1.4 b=(7,6)	h = 1 k = 1 b=(7,6)
4	h = 20006.8 k = 6.8 b=(2,5)	h = 10004.8 k = 5.8 b=(3,5)	h = 10003.8 k = 4.8 b=(4,5)	h = 3.8 k = 3.8 b=(5,5)	h = 2.8 k = 2.8 b=(6,5)	h = 2.4 k = 2.4 b=(7,5)	h = 2 k = 2 b=(7,5)
3	h = 8.0 k = 8.0 b=(2,2)	h = 10005.2 k = 6.2 b=(3,4)	h = 10004.2 k = 5.2 b=(4,4)	h = 4.2 k = 4.2 b=(5,4)	h = 3.8 k = 3.8 b=(6,4)	h = 3.4 k = 3.4 b=(7,4)	h = 3 k = 3 b=(7,4)
2	h = 7.6 k = 7.6 b=(2,2)	h = 6.6 k = 6.6 b=(3,2)	h = 5.6 k = 5.6 b=(4,3)	h = 10004.2 k = 5.2 b=(5,3)	h = 4.8 k = 4.8 b=(6,3)	h = 4.4 k = 4.4 b=(7,3)	h = 4 k = 4 b=(7,3)
1	h = 8.0 k = 8.0 b=(2,2)	h = 7.0 k = 7.0 b=(3,2)	h = 6.6 k = 6.6 b=(3,2)	h = 6.2 k = 6.2 b=(5,2)	h = 5.8 k = 5.8 b=(6,2)	h = 5.4 k = 5.4 b=(7,2)	h = 5 k = 5 b=(7,2)
	1	2	3	4	5	6	7

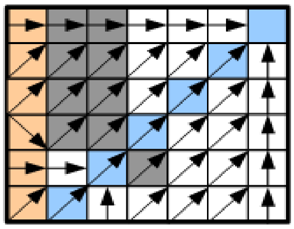
State	k
(1,6)	6
(1,5)	6.4
(1,4)	6.8
(1,2)	7.6
(1,3)	8.0
(1,1)	8.0



D*: Example (9/16)

6	h = 20004 k = 6 b=(2,6)	h = 10004 k = 5 b=(3,6)	h = 10003 k = 4 b=(4,6)	h = 3 k = 3 b=(5,6)	h = 2 k = 2 b=(6,6)	h = 1 k = 1 b=(7,6)	**h = 0** **k = 0** **b=**
5	h = 20004.4 k = 6.4 b=(2,6)	h = 10004.4 k = 5.4 b=(3,6)	h = 10003.4 k = 4.4 b=(4,6)	h = 3.4 k = 3.4 b=(5,6)	h = 2.4 k = 2.4 b=(6,6)	h = 1.4 k = 1.4 b=(7,6)	h = 1 k = 1 b=(7,6)
4	h = 20006.8 k = 6.8 b=(2,5)	h = 10004.8 k = 5.8 b=(3,5)	h = 10003.8 k = 4.8 b=(4,5)	h = 3.8 k = 3.8 b=(5,5)	h = 2.8 k = 2.8 b=(6,5)	h = 2.4 k = 2.4 b=(7,5)	h = 2 k = 2 b=(7,5)
3	h = 8.0 k = 8.0 b=(2,2)	h = 10005.2 k = 6.2 b=(3,4)	h = 10004.2 k = 5.2 b=(4,4)	h = 4.2 k = 4.2 b=(5,4)	h = 3.8 k = 3.8 b=(6,4)	h = 3.4 k = 3.4 b=(7,4)	h = 3 k = 3 b=(7,4)
2	h = 7.6 k = 7.6 b=(2,2)	h = 6.6 k = 6.6 b=(3,2)	h = 5.6 k = 5.6 b=(4,3)	h = 10004.2 k = 5.2 b=(5,3)	h = 4.8 k = 4.8 b=(6,3)	h = 4.4 k = 4.4 b=(7,3)	h = 4 k = 4 b=(7,3)
1	h = 8.0 k = 8.0 b=(2,2)	h = 7.0 k = 7.0 b=(3,2)	h = 6.6 k = 6.6 b=(3,2)	h = 6.2 k = 6.2 b=(5,2)	h = 5.8 k = 5.8 b=(6,2)	h = 5.4 k = 5.4 b=(7,2)	h = 5 k = 5 b=(7,2)
	1	2	3	4	5	6	7

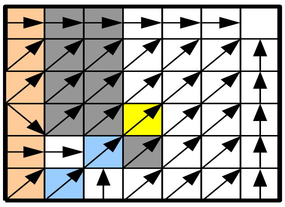
State	k
(1,6)	6
(1,5)	6.4
(1,4)	6.8
(1,2)	7.6
(1,3)	8.0
(1,1)	8.0



D*: Example (10/16)

6	h = 20004 k = 6 b=(2,6)	h = 10004 k = 5 b=(3,6)	h = 10003 k = 4 b=(4,6)	h = 3 k = 3 b=(5,6)	h = 2 k = 2 b=(6,6)	h = 1 k = 1 b=(7,6)	**h = 0** **k = 0** **b=**
5	h = 20004.4 k = 6.4 b=(2,6)	h = 10004.4 k = 5.4 b=(3,6)	h = 10003.4 k = 4.4 b=(4,6)	h = 3.4 k = 3.4 b=(5,6)	h = 2.4 k = 2.4 b=(6,6)	h = 1.4 k = 1.4 b=(7,6)	h = 1 k = 1 b=(7,6)
4	h = 20006.8 k = 6.8 b=(2,5)	h = 10004.8 k = 5.8 b=(3,5)	h = 10003.8 k = 4.8 b=(4,5)	h = 3.8 k = 3.8 b=(5,5)	h = 2.8 k = 2.8 b=(6,5)	h = 2.4 k = 2.4 b=(7,5)	h = 2 k = 2 b=(7,5)
3	h = 8.0 k = 8.0 b=(2,2)	h = 10005.2 k = 6.2 b=(3,4)	h = 10004.2 k = 5.2 b=(4,4)	h = 4.2 k = 4.2 b=(5,4)	h = 3.8 k = 3.8 b=(6,4)	h = 3.4 k = 3.4 b=(7,4)	h = 3 k = 3 b=(7,4)
2	h = 7.6 k = 7.6 b=(2,2)	h = 6.6 k = 6.6 b=(3,2)	= 5.6 k = 5.6 b=(4,3)	h = 10004.2 k = 5.2 b=(5,3)	h = 4.8 k = 4.8 b=(6,3)	h = 4.4 k = 4.4 b=(7,3)	h = 4 k = 4 b=(7,3)
1	h = 8.0 k = 8.0 b=(2,2)	= 7.0 k = 7.0 b=(3,2)	h = 6.6 k = 6.6 b=(3,2)	h = 6.2 k = 6.2 b=(5,2)	h = 5.8 k = 5.8 b=(6,2)	h = 5.4 k = 5.4 b=(7,2)	h = 5 k = 5 b=(7,2)
	1	2	3	4	5	6	7

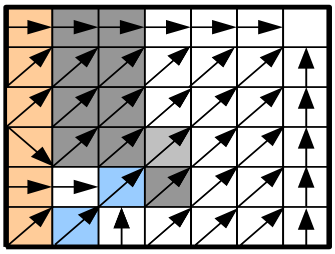
State	k
(1,6)	6
(1,5)	6.4
(1,4)	6.8
(1,2)	7.6
(1,3)	8.0
(1,1)	8.0



D*: Example (11/16)

6	h = 20004 k = 6 b=(2,6)	h = 10004 k = 5 b=(3,6)	h = 10003 k = 4 b=(4,6)	h = 3 k = 3 b=(5,6)	h = 2 k = 2 b=(6,6)	h = 1 k = 1 b=(7,6)	**h = 0** **k = 0** **b=**
5	h = 20004.4 k = 6.4 b=(2,6)	h = 10004.4 k = 5.4 b=(3,6)	h = 10003.4 k = 4.4 b=(4,6)	h = 3.4 k = 3.4 b=(5,6)	h = 2.4 k = 2.4 b=(6,6)	h = 1.4 k = 1.4 b=(7,6)	h = 1 k = 1 b=(7,6)
4	h = 20006.8 k = 6.8 b=(2,5)	h = 10004.8 k = 5.8 b=(3,5)	h = 10003.8 k = 4.8 b=(4,5)	h = 3.8 k = 3.8 b=(5,5)	h = 2.8 k = 2.8 b=(6,5)	h = 2.4 k = 2.4 b=(7,5)	h = 2 k = 2 b=(7,5)
3	h = 8.0 k = 8.0 b=(2,2)	h = 10005.2 k = 6.2 b=(3,4)	h = 10004.2 k = 5.2 b=(4,4)	[illegible]	h = 3.8 k = 3.8 b=(6,4)	h = 3.4 k = 3.4 b=(7,4)	h = 3 k = 3 b=(7,4)
2	h = 7.6 k = 7.6 b=(2,2)	h = 6.6 k = 6.6 b=(3,2)	= 5.6 k = 5.6 b=(4,3)	h = 10004.2 k = 5.2 b=(5,3)	h = 4.8 k = 4.8 b=(6,3)	h = 4.4 k = 4.4 b=(7,3)	h = 4 k = 4 b=(7,3)
1	h = 8.0 k = 8.0 b=(2,2)	= 7.0 k = 7.0 b=(3,2)	h = 6.6 k = 6.6 b=(3,2)	h = 6.2 k = 6.2 b=(5,2)	h = 5.8 k = 5.8 b=(6,2)	h = 5.4 k = 5.4 b=(7,2)	h = 5 k = 5 b=(7,2)
	1	2	3	4	5	6	7

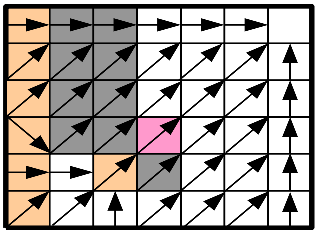
State	k
(4,3)	4.2
(1,6)	6
(1,5)	6.4
(1,4)	6.8
(1,2)	7.6
(1,3)	8.0
(1,1)	8.0



D*: Example (12/16)

6	h = 20004 k = 6 b=(2,6)	h = 10004 k = 5 b=(3,6)	h = 10003 k = 4 b=(4,6)	h = 3 k = 3 b=(5,6)	h = 2 k = 2 b=(6,6)	h = 1 k = 1 b=(7,6)	**h = 0** **k = 0** **b=**
5	h = 20004.4 k = 6.4 b=(2,6)	h = 10004.4 k = 5.4 b=(3,6)	h = 10003.4 k = 4.4 b=(4,6)	h = 3.4 k = 3.4 b=(5,6)	h = 2.4 k = 2.4 b=(6,6)	h = 1.4 k = 1.4 b=(7,6)	h = 1 k = 1 b=(7,6)
4	h = 20006.8 k = 6.8 b=(2,5)	h = 10004.8 k = 5.8 b=(3,5)	h = 10003.8 k = 4.8 b=(4,5)	h = 3.8 k = 3.8 b=(5,5)	h = 2.8 k = 2.8 b=(6,5)	h = 2.4 k = 2.4 b=(7,5)	h = 2 k = 2 b=(7,5)
3	h = 8.0 k = 8.0 b=(2,2)	h = 10005.2 k = 6.2 b=(3,4)	h = 10004.2 k = 5.2 b=(4,4)	h = 4.2 k = 4.2 b=(5,4)	h = 3.8 k = 3.8 b=(6,4)	h = 3.4 k = 3.4 b=(7,4)	h = 3 k = 3 b=(7,4)
2	h = 7.6 k = 7.6 b=(2,2)	h = 6.6 k = 6.6 b=(3,2)	h = 10004.6 k = 5.6 b=(4,3)	h = 10004.2 k = 5.2 b=(5,3)	h = 4.8 k = 4.8 b=(6,3)	h = 4.4 k = 4.4 b=(7,3)	h = 4 k = 4 b=(7,3)
1	h = 8.0 k = 8.0 b=(2,2)	h = 7.0 k = 7.0 b=(3,2)	h = 6.6 k = 6.6 b=(3,2)	h = 6.2 k = 6.2 b=(5,2)	h = 5.8 k = 5.8 b=(6,2)	h = 5.4 k = 5.4 b=(7,2)	h = 5 k = 5 b=(7,2)
	1	2	3	4	5	6	7

State	k
(3,2)	5.6
(1,6)	6
(1,5)	6.4
(1,4)	6.8
(1,2)	7.6
(1,3)	8.0
(1,1)	8.0

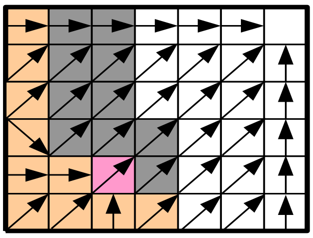


D*: Example (13/16)

6	h = 20004 k = 6 b=(2,6)	h = 10004 k = 5 b=(3,6)	h = 10003 k = 4 b=(4,6)	h = 3 k = 3 b=(5,6)	h = 2 k = 2 b=(6,6)	h = 1 k = 1 b=(7,6)	**h = 0** **k = 0** **b=**
5	h = 20004.4 k = 6.4 b=(2,6)	h = 10004.4 k = 5.4 b=(3,6)	h = 10003.4 k = 4.4 b=(4,6)	h = 3.4 k = 3.4 b=(5,6)	h = 2.4 k = 2.4 b=(6,6)	h = 1.4 k = 1.4 b=(7,6)	h = 1 k = 1 b=(7,6)
4	h = 20006.8 k = 6.8 b=(2,5)	h = 10004.8 k = 5.8 b=(3,5)	h = 10003.8 k = 4.8 b=(4,5)	h = 3.8 k = 3.8 b=(5,5)	h = 2.8 k = 2.8 b=(6,5)	h = 2.4 k = 2.4 b=(7,5)	h = 2 k = 2 b=(7,5)
3	h = 8.0 k = 8.0 b=(2,2)	h = 10005.2 k = 6.2 b=(3,4)	h = 10004.2 k = 5.2 b=(4,4)	h = 4.2 k = 4.2 b=(5,4)	h = 3.8 k = 3.8 b=(6,4)	h = 3.4 k = 3.4 b=(7,4)	h = 3 k = 3 b=(7,4)
2	h = 7.6 k = 7.6 b=(2,2)	h = 10005.6 k = 6.6 b=(3,2)	h = 10004.6 k = 5.6 b=(4,3)	h = 10004.2 k = 5.2 b=(5,3)	h = 4.8 k = 4.8 b=(6,3)	h = 4.4 k = 4.4 b=(7,3)	h = 4 k = 4 b=(7,3)
1	h = 8.0 k = 8.0 b=(2,2)	h = 10006 k = 7.0 b=(3,2)	h = 10005.6 k = 6.6 b=(3,2)	h = 6.2 k = 6.2 b=(5,2)	h = 5.8 k = 5.8 b=(6,2)	h = 5.4 k = 5.4 b=(7,2)	h = 5 k = 5 b=(7,2)
	1	2	3	4	5	6	7

State	k
(1,6)	6
(4,1)	6.2
(1,5)	6.4
(3,1)	6.6
(2,2)	6.6
(1,4)	6.8

State	k
(2,1)	7.0
(1,2)	7.6
(1,3)	8.0
(1,1)	8.0

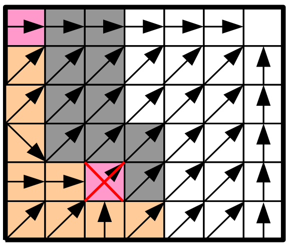


D*: Example (14/16)

6	h = 20004 k = 6 b=(2,6)	h = 10004 k = 5 b=(3,6)	h = 10003 k = 4 b=(4,6)	h = 3 k = 3 b=(5,6)	h = 2 k = 2 b=(6,6)	h = 1 k = 1 b=(7,6)	**h = 0** **k = 0** **b=**
5	h = 20004.4 k = 6.4 b=(2,6)	h = 10004.4 k = 5.4 b=(3,6)	h = 10003.4 k = 4.4 b=(4,6)	h = 3.4 k = 3.4 b=(5,6)	h = 2.4 k = 2.4 b=(6,6)	h = 1.4 k = 1.4 b=(7,6)	h = 1 k = 1 b=(7,6)
4	h = 20006.8 k = 6.8 b=(2,5)	h = 10004.8 k = 5.8 b=(3,5)	h = 10003.8 k = 4.8 b=(4,5)	h = 3.8 k = 3.8 b=(5,5)	h = 2.8 k = 2.8 b=(6,5)	h = 2.4 k = 2.4 b=(7,5)	h = 2 k = 2 b=(7,5)
3	h = 8.0 k = 8.0 b=(2,2)	h = 10005.2 k = 6.2 b=(3,4)	h = 10004.2 k = 5.2 b=(4,4)	h = 4.2 k = 4.2 b=(5,4)	h = 3.8 k = 3.8 b=(6,4)	h = 3.4 k = 3.4 b=(7,4)	h = 3 k = 3 b=(7,4)
2	h = 7.6 k = 7.6 b=(2,2)	h = 10005.6 k = 6.6 b=(3,2)	h = 10004.6 k = 5.6 b=(4,3)	h = 10004.2 k = 5.2 b=(5,3)	h = 4.8 k = 4.8 b=(6,3)	h = 4.4 k = 4.4 b=(7,3)	h = 4 k = 4 b=(7,3)
1	h = 8.0 k = 8.0 b=(2,2)	h = 10006 k = 7.0 b=(3,2)	h = 10005.6 k = 6.6 b=(3,2)	h = 6.2 k = 6.2 b=(5,2)	h = 5.8 k = 5.8 b=(6,2)	h = 5.4 k = 5.4 b=(7,2)	h = 5 k = 5 b=(7,2)
	1	2	3	4	5	6	7

State	k
(4,1)	6.2
(1,5)	6.4
(3,1)	6.6
(2,2)	6.6
(1,4)	6.8

State	k
(2,1)	7.0
(1,2)	7.6
(1,3)	8.0
(1,1)	8.0

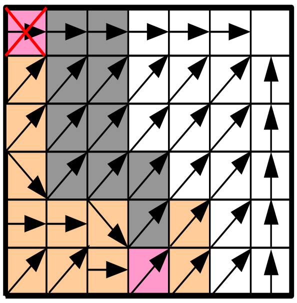


D*: Example (15/16)

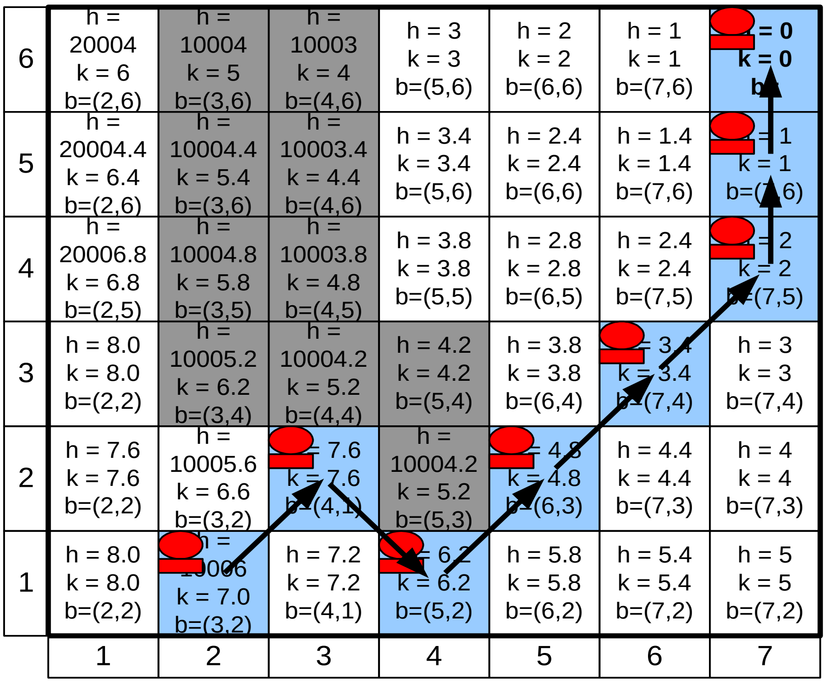
6	h = 20004 k = 6 b=(2,6)	h = 10004 k = 5 b=(3,6)	h = 10003 k = 4 b=(4,6)	h = 3 k = 3 b=(5,6)	h = 2 k = 2 b=(6,6)	h = 1 k = 1 b=(7,6)	**h = 0** **k = 0** **b=**
5	h = 20004.4 k = 6.4 b=(2,6)	h = 10004.4 k = 5.4 b=(3,6)	h = 10003.4 k = 4.4 b=(4,6)	h = 3.4 k = 3.4 b=(5,6)	h = 2.4 k = 2.4 b=(6,6)	h = 1.4 k = 1.4 b=(7,6)	h = 1 k = 1 b=(7,6)
4	h = 20006.8 k = 6.8 b=(2,5)	h = 10004.8 k = 5.8 b=(3,5)	h = 10003.8 k = 4.8 b=(4,5)	h = 3.8 k = 3.8 b=(5,5)	h = 2.8 k = 2.8 b=(6,5)	h = 2.4 k = 2.4 b=(7,5)	h = 2 k = 2 b=(7,5)
3	h = 8.0 k = 8.0 b=(2,2)	h = 10005.2 k = 6.2 b=(3,4)	h = 10004.2 k = 5.2 b=(4,4)	h = 4.2 k = 4.2 b=(5,4)	h = 3.8 k = 3.8 b=(6,4)	h = 3.4 k = 3.4 b=(7,4)	h = 3 k = 3 b=(7,4)
2	h = 7.6 k = 7.6 b=(2,2)	h = 10005.6 k = 6.6 b=(3,2)	h = 7.6 k = 7.6 b=(4,1)	h = 10004.2 k = 5.2 b=(5,3)	h = 4.8 k = 4.8 b=(6,3)	h = 4.4 k = 4.4 b=(7,3)	h = 4 k = 4 b=(7,3)
1	h = 8.0 k = 8.0 b=(2,2)	h = 10006 k = 7.0 b=(3,2)	h = 7.2 k = 7.2 b=(4,1)	h = 6.2 k = 6.2 b=(5,2)	h = 5.8 k = 5.8 b=(6,2)	h = 5.4 k = 5.4 b=(7,2)	h = 5 k = 5 b=(7,2)
	1	2	3	4	5	6	7

State	k
(5,2)	4.8
(5,1)	5.8
(3,2)	5.6
(1,5)	6.4
(3,1)	6.6
(2,2)	6.6
(1,4)	6.8

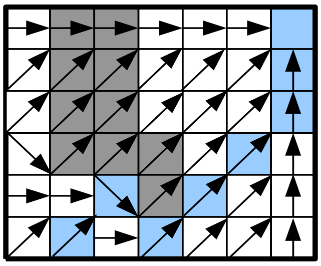
State	k
(2,1)	7.0
(1,2)	7.6
(1,3)	8.0
(1,1)	8.0



D*: Example (16/16)



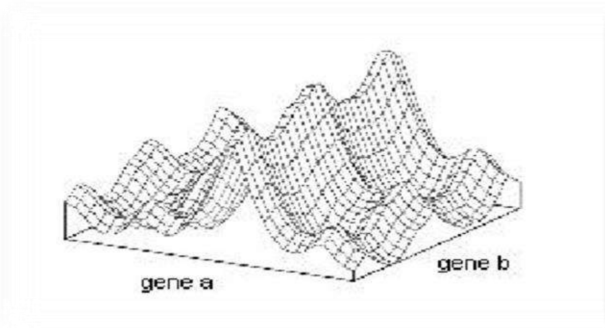
	1	2	3	4	5	6	7
6	h = 20004 k = 6 b=(2,6)	h = 10004 k = 5 b=(3,6)	h = 10003 k = 4 b=(4,6)	h = 3 k = 3 b=(5,6)	h = 2 k = 2 b=(6,6)	h = 1 k = 1 b=(7,6)	= 0 k = 0 b
5	h = 20004.4 k = 6.4 b=(2,6)	h = 10004.4 k = 5.4 b=(3,6)	h = 10003.4 k = 4.4 b=(4,6)	h = 3.4 k = 3.4 b=(5,6)	h = 2.4 k = 2.4 b=(6,6)	h = 1.4 k = 1.4 b=(7,6)	= 1 k = 1 b=(7,6)
4	h = 20006.8 k = 6.8 b=(2,5)	h = 10004.8 k = 5.8 b=(3,5)	h = 10003.8 k = 4.8 b=(4,5)	h = 3.8 k = 3.8 b=(5,5)	h = 2.8 k = 2.8 b=(6,5)	h = 2.4 k = 2.4 b=(7,5)	= 2 k = 2 b=(7,5)
3	h = 8.0 k = 8.0 b=(2,2)	h = 10005.2 k = 6.2 b=(3,4)	h = 10004.2 k = 5.2 b=(4,4)	h = 4.2 k = 4.2 b=(5,4)	h = 3.8 k = 3.8 b=(6,4)	= 3.4 k = 3.4 b=(7,4)	h = 3 k = 3 b=(7,4)
2	h = 7.6 k = 7.6 b=(2,2)	h = 10005.6 k = 6.6 b=(3,2)	= 7.6 k = 7.6 b=(4,1)	h = 10004.2 k = 5.2 b=(5,3)	= 4.8 k = 4.8 b=(6,3)	h = 4.4 k = 4.4 b=(7,3)	h = 4 k = 4 b=(7,3)
1	h = 8.0 k = 8.0 b=(2,2)	h = 10006 k = 7.0 b=(3,2)	h = 7.2 k = 7.2 b=(4,1)	= 6.2 k = 6.2 b=(5,2)	h = 5.8 k = 5.8 b=(6,2)	h = 5.4 k = 5.4 b=(7,2)	h = 5 k = 5 b=(7,2)

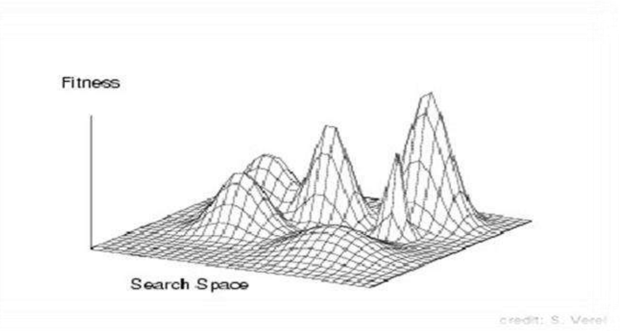


Landscapes is a tool for analyzing optimization problems.

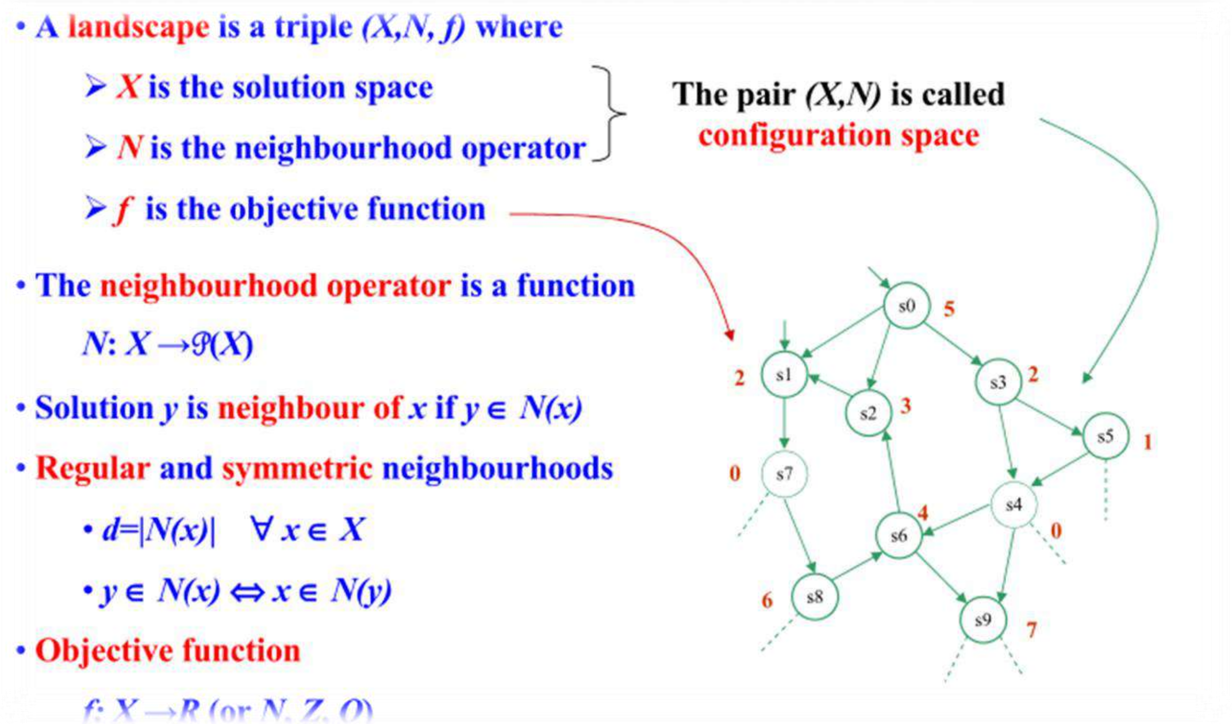
Central idea: study the search space to obtain information

- Better understanding of the problem “
- Predict algorithmic performance among the rare existing
- Improve search algorithms





credit: S. Verel



- A landscape is a triple (X,N,f) where
  - X is the solution space
  - N is the neighbourhood operator
  - f is the objective function

The pair (X,N) is called configuration space

- The neighbourhood operator is a function

  $N: X \rightarrow \mathcal{P}(X)$
- Solution y is neighbour of x if $y \in N(x)$
- Regular and symmetric neighbourhoods
  - $d=|N(x)| \quad \forall x \in X$
  - $y \in N(x) \Leftrightarrow x \in N(y)$
- Objective function

  $f: X \rightarrow R$ (or N, Z, Q)

Fitness Landscape

Components of fitness landscape:

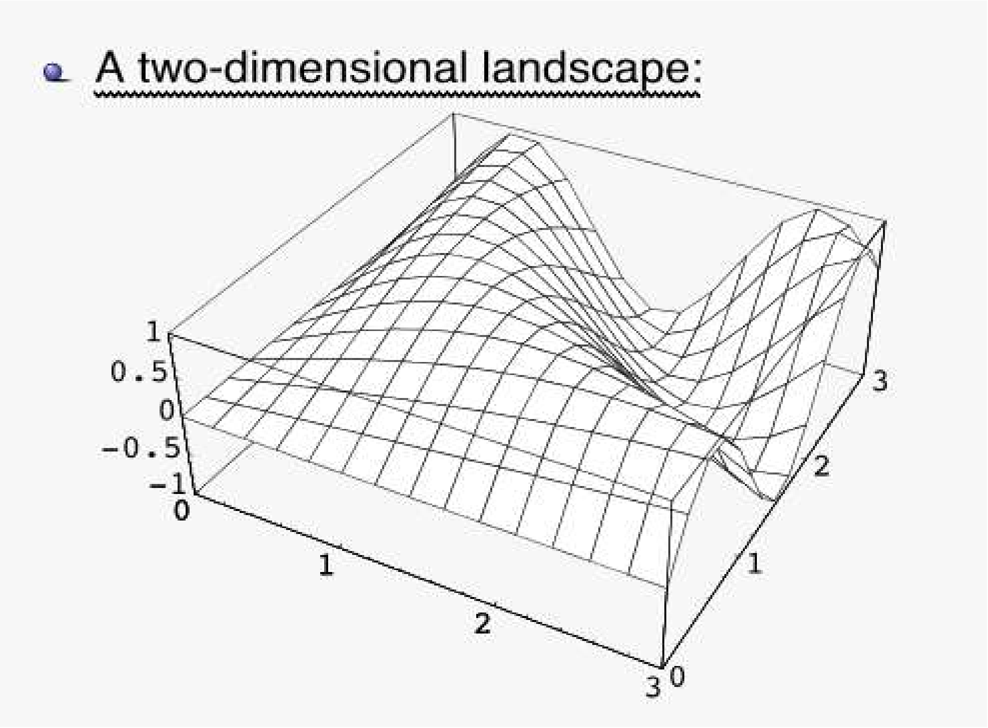
- Set S of admissible solutions
- Fitness function that assigns a real value to each solution
- Distance measure that defines distance between any two solutions in S

Example distance measures:

- Hamming distance for binary strings (num. mismatched bits).
- Euclidean distance metric for continuous vectors

Easy Problems, and Hard Problems:

- Easy: few peaks; smooth surfaces, no ridges/plateaus
- Hard: many peaks; jagged or discontinuous surfaces, plateaus



- A two-dimensional landscape:

Search Space

Given a combinatorial problem P, a search space associated to a mathematical formulation of P is defined by a couple (S,f).

where S is a finite set of configurations (or nodes or points) and f a cost function which associates a real number to each configurations of S.

For this structure two most common measures are the minimum and the maximum costs. In this case we have the combinatorial optimization problems.

Combinatorial optimization problems are often hard to solve since such problems may have huge and complex search landscape

Example: K-SAT

- An instance of the *K-SAT problem* consists of a set *V* of variables, a collection *C* of clauses over *V* such that each clause $c \in C$ has $|c| = K$
- The problem is to find a satisfying truth assignment for *C*
- The *search space for the 2-SAT* with $|V|=2$ is **(S,*f*)** where
  - **S={ (T,T), (T,F), (F,T), (F,F) }** and
  - **the cost function** for *2-SAT* computes only the number of satisfied clauses

$$f_{sat}(s) = \#SatisfiedClauses(F,s),\ s \in S$$

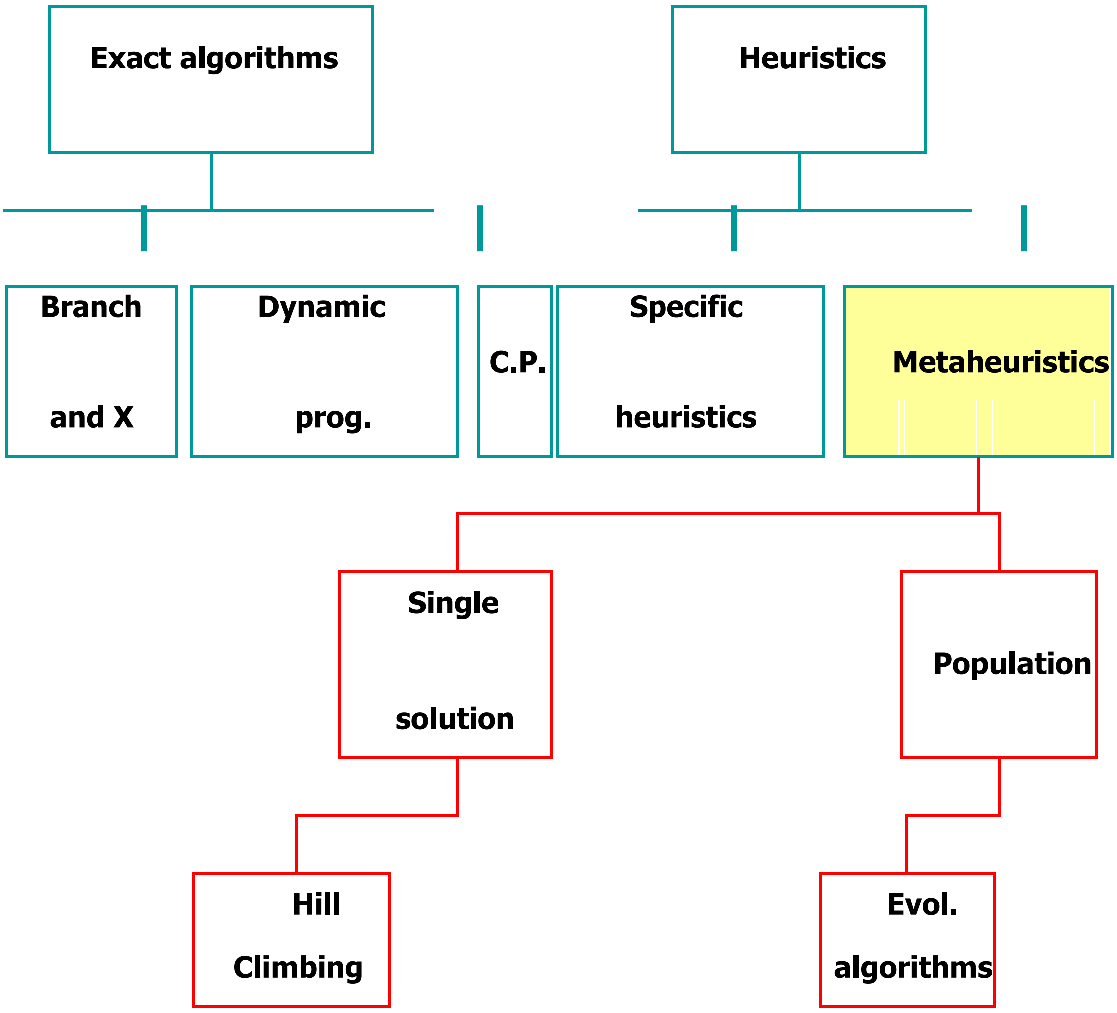
An Example of SEARCH SPACE

Let we consider $F = (A \vee \neg B) \wedge (\neg A \vee \neg B)$

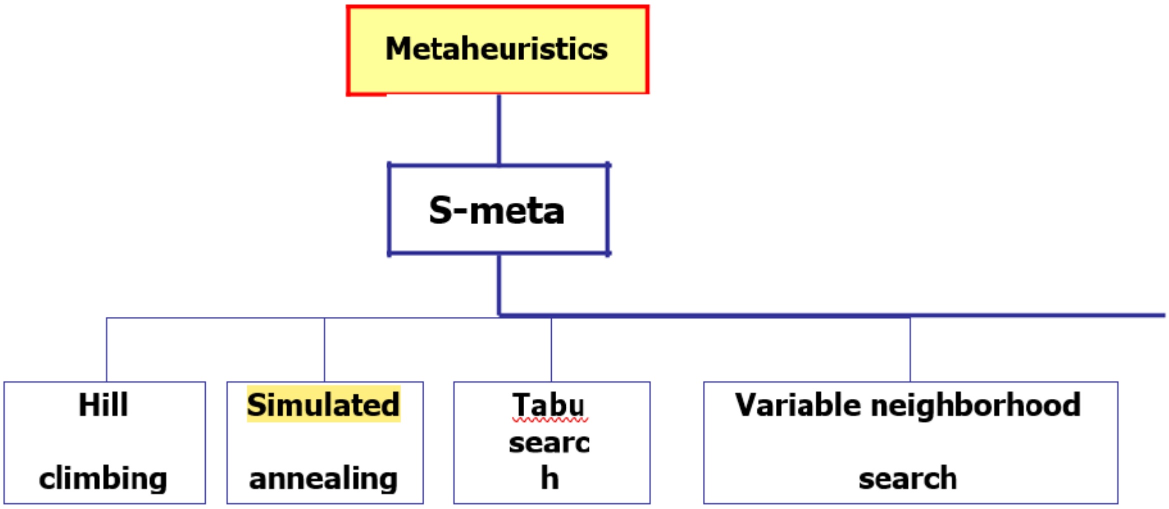
A	**B**	$f_{sat}(F,s)$
T	*T*	1
T	*F*	2
F	*T*	1
F	*F*	2

Lect 4: Some metaheuristics classification

Taxonomy (metaheuristics)



Taxonomy (Single solution-based metaheuristics)



Metaheuristics

S-meta

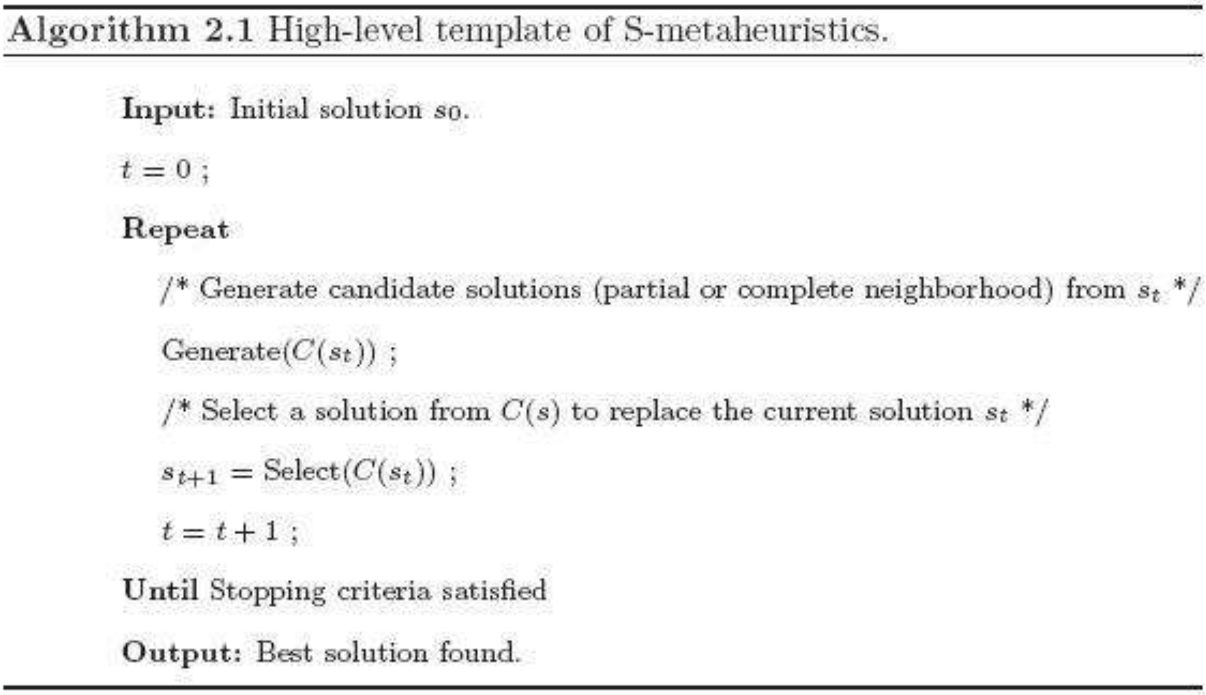
Hill climbing

Simulated annealing

Tabu search

Variable neighborhood search

High-level template of S-metaheuristics

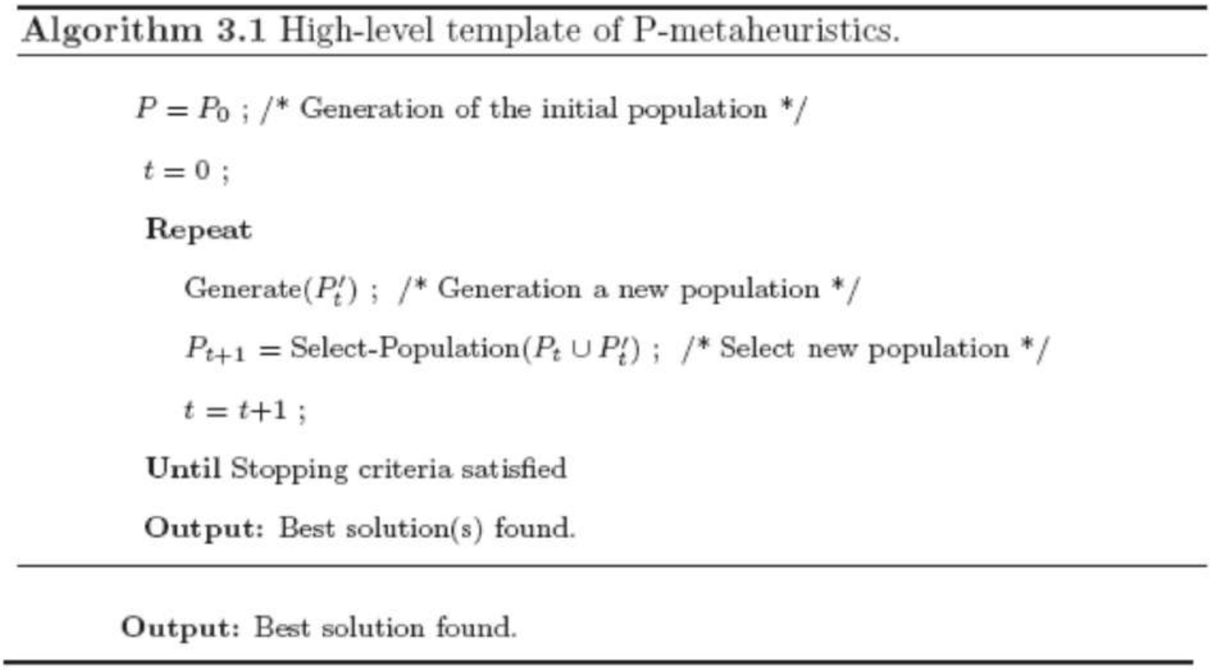


Algorithm 2.1 High-level template of S-metaheuristics.

```
Input: Initial solution $s_0$.
$t = 0$ ;
Repeat
    /* Generate candidate solutions (partial or complete neighborhood) from $s_t$ */
    Generate($C(s_t)$) ;
    /* Select a solution from $C(s)$ to replace the current solution $s_t$ */
    $s_{t+1}$ = Select($C(s_t)$) ;
    $t = t + 1$ ;
Until Stopping criteria satisfied
Output: Best solution found.
```

High-level template of P-metaheuristics

*



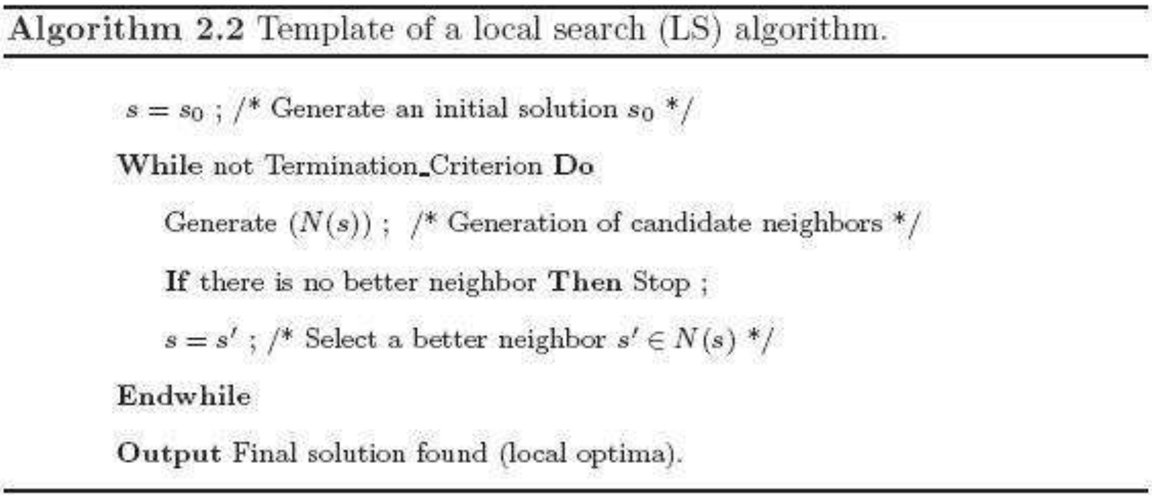
Algorithm 3.1 High-level template of P-metaheuristics.

```
$P = P_0$ ; /* Generation of the initial population */
$t = 0$ ;
Repeat
    Generate($P'_t$) ; /* Generation a new population */
    $P_{t+1}$ = Select-Population($P_t \cup P'_t$) ; /* Select new population */
    $t = t+1$ ;
Until Stopping criteria satisfied
Output: Best solution(s) found.
```

Output: Best solution found.

Local search (Hill-climbing)

- Oldest and simplest S-metaheuristic method
- Hill-climbing, descent, iterative improvement, and so on
- It starts at a given initial solution. At each iteration, the heuristic replaces the current solution by a neighbor that improves the objective function.
- The search stops when all candidate neighbors are worse than the current solution, meaning a local optimum is reached.
- LS may be seen as a descent walk in the graph representing the search space. This graph may be defined by G = (S, V), where S represents the set of all feasible solutions of the search space and V represents the neighborhood relation. In the graph G, an edge (i, j) will connect to any neighboring solutions si and sj . For a given solution s, the number of associated edges will be |N(s)| (number of neighbors).



Algorithm 2.2 Template of a local search (LS) algorithm.

```
$s = s_0$ ; /* Generate an initial solution $s_0$ */
While not Termination_Criterion Do
    Generate $(N(s))$ ;  /* Generation of candidate neighbors */
    If there is no better neighbor Then Stop ;
    $s = s'$ ; /* Select a better neighbor $s' \in N(s)$ */
Endwhile
Output Final solution found (local optima).
```

Selection of the Neighbor

Many strategies can be applied in the selection of a better neighbor, such as:

1. Best improvement (steepest descent): In this strategy, the best neighbor (i.e., neighbor that improves the most the cost function) is selected. This type of exploration may be time-consuming for large neighborhoods.
2. First improvement: This strategy consists in choosing the first improving neighbor that is better than the current solution. In the worst case (i.e., when no improvement is found), a complete evaluation of the neighborhood is performed.
3. Random selection: In this strategy, a random selection is applied to those neighbors improving the current solution.

Escaping from Local Optima

- In general, local search is a very easy method to design and implement and gives fairly good solutions very quickly.
- One of the main disadvantages of LS is that
  - it converges toward local optima. Moreover,
  - the algorithm can be very sensitive to the initial solution;
  - -the number of iterations performed may not be known in advance.
- Local search works well if
  - there are not too many local optima in the search space or
  - the quality of the different local optima is more or less similar.

As the main disadvantage of local search algorithms is the convergence toward local optima, many alternatives algorithms have been proposed to avoid becoming stuck at local optima.

- Iterating from different initial solutions: This strategy is applied in iterated local search, GRASP, and so forth.

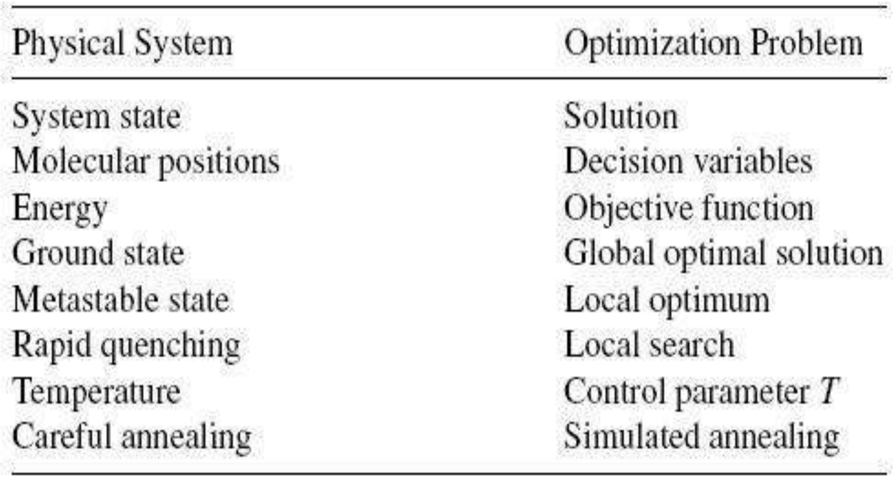
- Accepting nonimproving neighbors: Simulated annealing and tabu search are popular representative of this class of algorithms.
- Changing the neighborhood: this approach is used in variable neighborhood search strategies.
- Changing the objective function or the input data of the problem: the smoothing strategies, and the noising methods.

Simulated Annealing

- Mimics the physical annealing process (statistical mechanics).
- Material is heated and slowly cooled towards a strong crystalline structure (instead of metastable states).
- The first SA algorithm was developed in 1953 (Metropolis).
- Kirkpatrick et al. (1982) and V. Cerny applied SA to optimization problems:

– Kirkpatrick, S , Gelatt, C.D., Vecchi, M.P. 1983. “Optimization by Simulated Annealing”. Science, vol 220, No. 4598, pp 671-680

Analogy between the physical system and the optimization problem



Physical System	Optimization Problem
System state	Solution
Molecular positions	Decision variables
Energy	Objective function
Ground state	Global optimal solution
Metastable state	Local optimum
Rapid quenching	Local search
Temperature	Control parameter T
Careful annealing	Simulated annealing

- SA allows downwards steps.

- A move is selected at random and its acceptation is conditional (stochastic Boltzmann distribution)

$$P(\Delta E, T) = e^{-\frac{f(s')-f(s)}{T}}$$

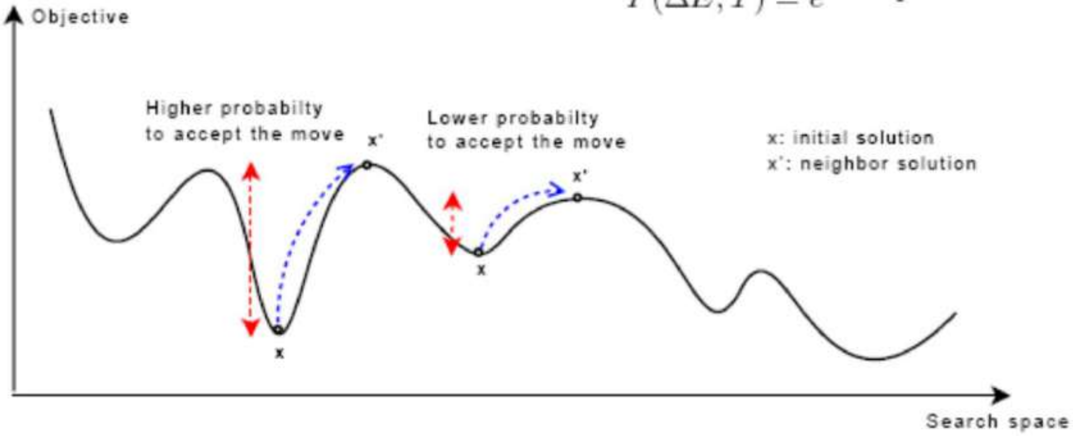
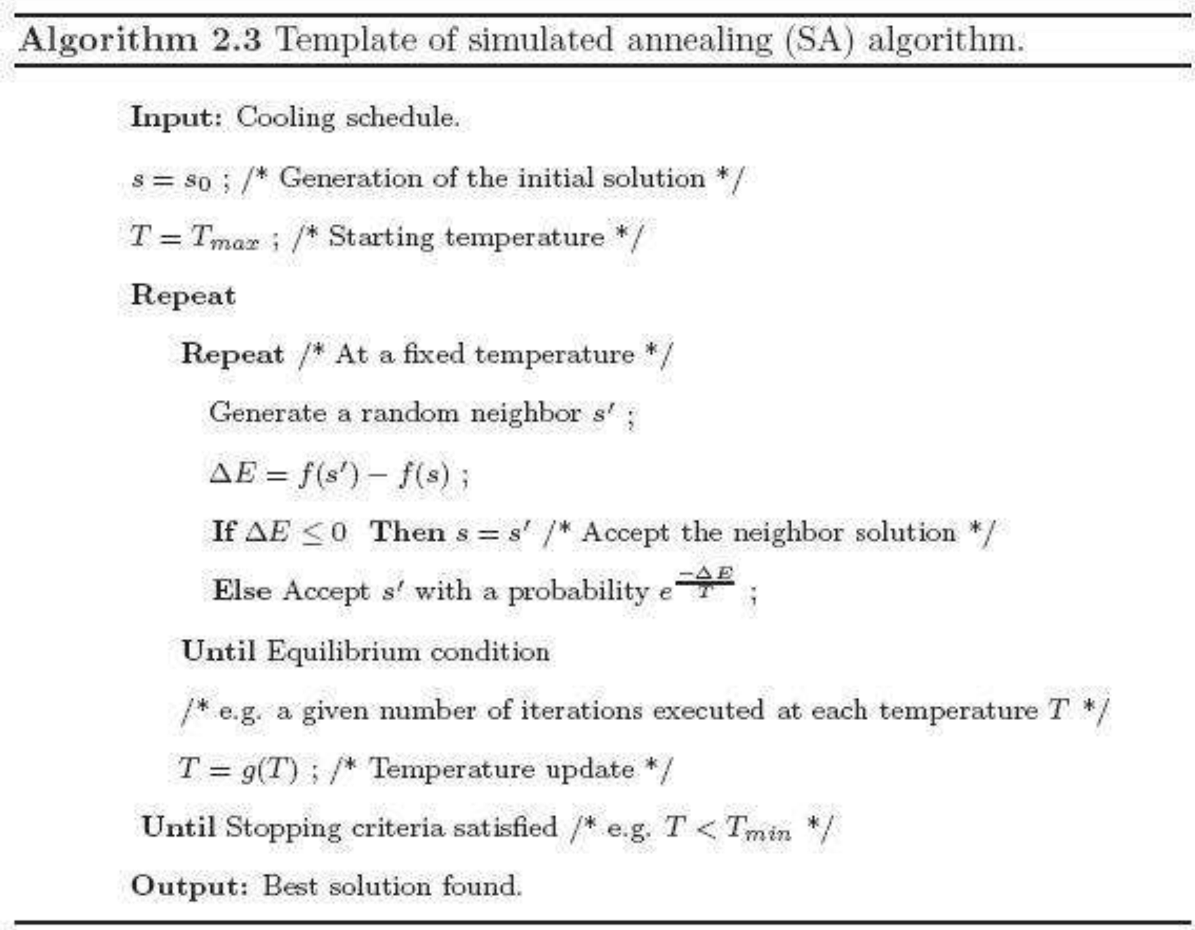


Fig. 2.25 Simulated annealing escaping from local optima. Higher is the temperature, more important is the probability of accepting a worst move. At a given temperature, lower is the increase of the objective function, more important is the probability of accepting the move. A better move is always accepted.

The Simulated Annealing Algorithm



Algorithm 2.3 Template of simulated annealing (SA) algorithm.

```
Input: Cooling schedule.
$s = s_0$ ; /* Generation of the initial solution */
$T = T_{max}$ ; /* Starting temperature */
Repeat
    Repeat /* At a fixed temperature */
        Generate a random neighbor $s'$ ;
        $\Delta E = f(s') - f(s)$ ;
        If $\Delta E \leq 0$ Then $s = s'$ /* Accept the neighbor solution */
        Else Accept $s'$ with a probability $e^{\frac{-\Delta E}{T}}$ ;
    Until Equilibrium condition
    /* e.g. a given number of iterations executed at each temperature $T$ */
    $T = g(T)$ ; /* Temperature update */
Until Stopping criteria satisfied /* e.g. $T < T_{min}$ */
Output: Best solution found.
```

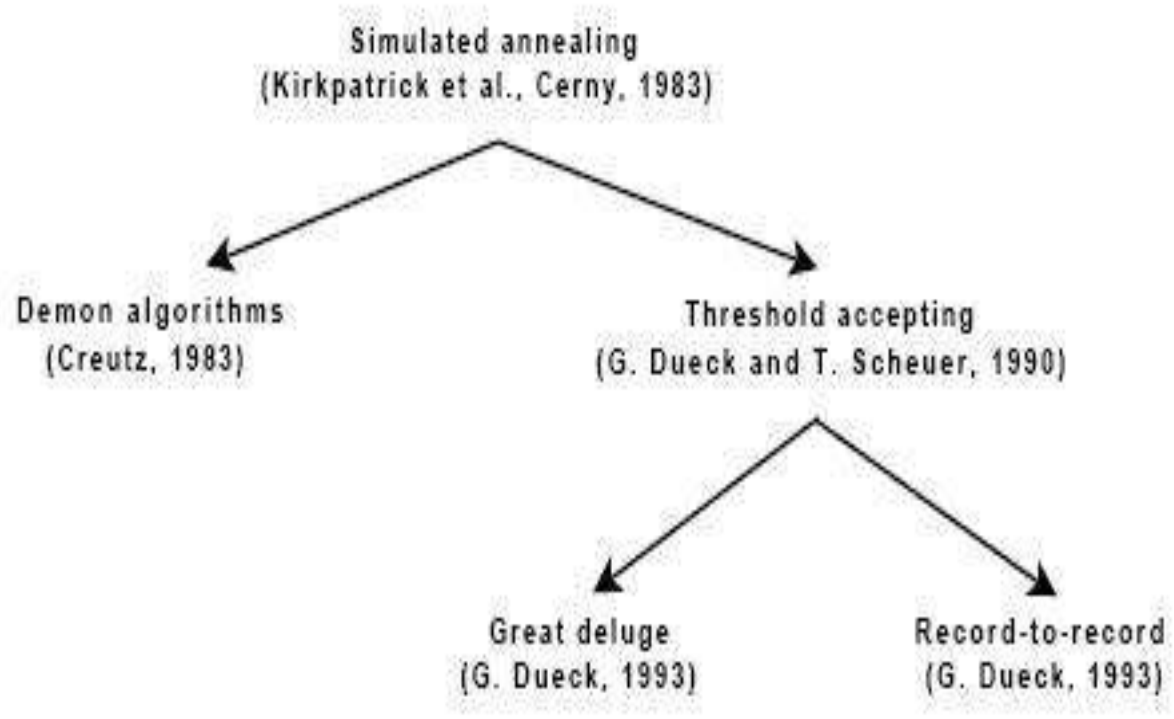
Travelling Salesman Example

- Suppose the T = 500max
- Initial trial solution: 1-2-3-4-5-6-7-1 Distance = 69
- Iteration 1: Reverse (3-4) 1-2-4-3-5-6-7-1 Distance = 65 (65-69=-4 <0) accepted
- Accept as new solution.
- Iteration 2: Reverse (3-5-6) 1-2-4-6-5-3-7-1 Distance = 64 (64-65=-1 <0) accepted
- Accept as new solution.
- Iteration 3: Reverse (3-7) 1-2-4-6-5-7-3-1 Distance = 66
- Since the new distance is "worse," accept as new solution with some probability
- Continue for some fixed number of iterations, or until temperature function falls below a given threshold.
- 66-64=2≤0 no then taking the probability for the solution
- $\alpha = e^{-\Delta E / T}$
- $\alpha = e^{-2/500} = 0.99$
- →0.99>0 yes then accept as new solution
- Temperature Update using:
- T = αT
- T=0.9*500
- T=450 (new temperature)

then repeat the loop (with 3 iterations) until reach to the goal with low distance and temperature or number of iterations which dete

SA Family: Similar methods

Other similar methods of simulated annealing have been proposed in the literature, such as threshold accepting, great deluge algorithm, record-to-record travel, and demon algorithms (Fig. bellow). The main objective in the design of those SA-inspired algorithms is to speed up the search of the SA algorithm without sacrificing the quality of solutions.

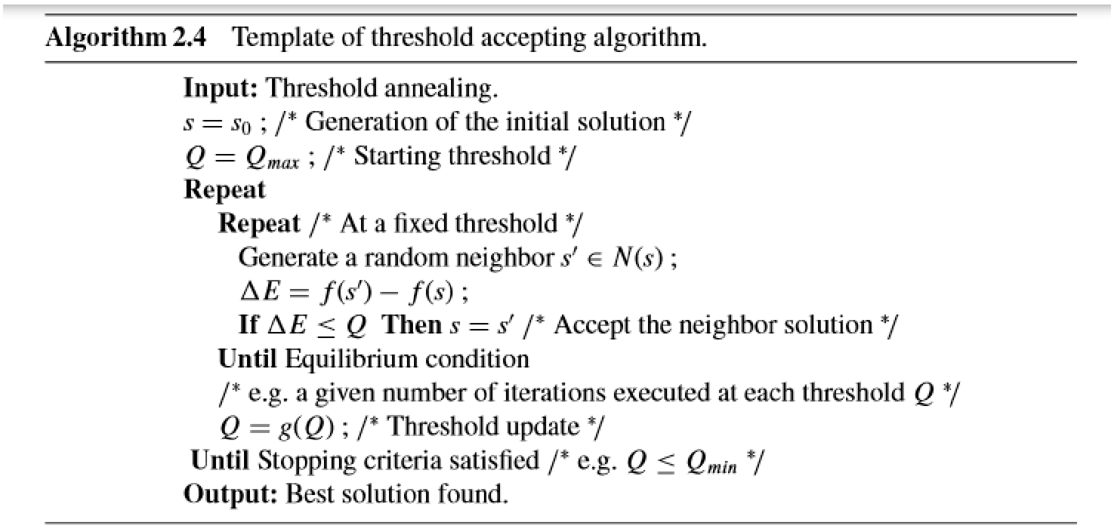


Threshold Accepting

Threshold accepting may be viewed as the deterministic variant of simulated annealing. TA escapes from local optima by accepting solutions that are not worse than the current solution by more than a given threshold Q. A deterministic acceptance function is defined as follows:

$$P_i(\Delta(s,s')) = \begin{cases} 1 & \text{if } Q_i \geq \Delta(s,s') \\ 0 & \text{otherwise} \end{cases}$$

Where Qi is the threshold value at iteration I and Δ(s,s′) is the change in the evaluation function between the current solution s and the neighbor solution s′. The threshold parameter in TA operates somewhat like the temperature in simulated annealing. Algorithm 2.4 describes the template of the TA algorithm. The number of generated neighbors at each iteration is fixed a priori. The threshold Q is updated following any annealing schedule.



Algorithm 2.4 Template of threshold accepting algorithm.

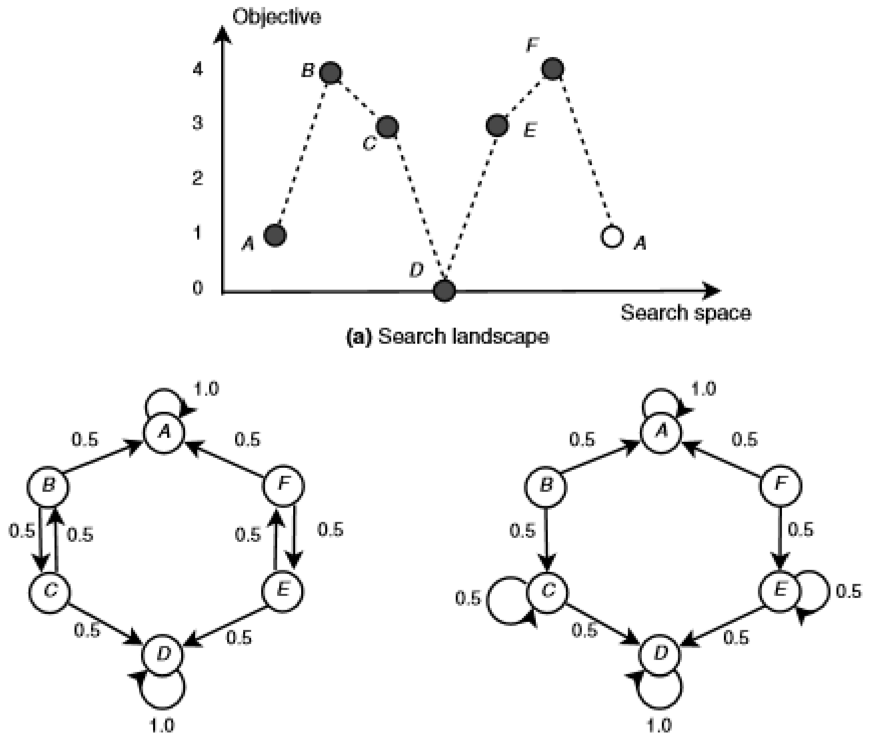
```
Input: Threshold annealing.
$s = s_0$ ; /* Generation of the initial solution */
$Q = Q_{max}$ ; /* Starting threshold */
Repeat
   Repeat /* At a fixed threshold */
     Generate a random neighbor $s' \in N(s)$ ;
     $\Delta E = f(s') - f(s)$ ;
     If $\Delta E \leq Q$ Then $s = s'$ /* Accept the neighbor solution */
   Until Equilibrium condition
   /* e.g. a given number of iterations executed at each threshold $Q$ */
   $Q = g(Q)$ ; /* Threshold update */
Until Stopping criteria satisfied /* e.g. $Q \leq Q_{min}$ */
Output: Best solution found.
```

- **TA is a <u>fast algorithm</u> compared to SA because the generation of random number and exponential functions consume a significant amount of computational time.**
- **The literature reports some <u>performance improvements compared</u> to the simulated annealing algorithm in solving combinatorial optimization problems such as the traveling salesman problem.**
- **In terms of <u>asymptotic convergence</u>, the theoretical properties of TA are similar to those of the SA algorithm.**

- **The threshold Q is updated according to an annealing schedule. It must be set as a deterministic non-increasing step function in the number of iterations i. The threshold decreases at each iteration and then reaches the value of 0 after a given number of iterations.**

Example Non-monotone threshold schedule. This simple example will illustrate that the optimal threshold schedule to solve a given problem may be non-monotone. Let us consider the one-dimensional landscape of a problem shown in Fig. 2.27. The search space is composed of six solutions, and each solution has two neighbors. The number of

iterations of the algorithm is fixed to M. The optimal schedule is defined as the one that maximizes the probability of finding the global optimum solution starting from any random initial solution. For many values of M, it has been shown that the optimal threshold schedule is non-monotone with negative thresholds. For instance, for M = 9, the optimal schedule is (0,−3,3,−1,−3,3,3,−1) and the probability to find the global optimum is 0.8372.



(a) Search landscape

(b) Transition probabilities for $T=2$

(c) Transition probabilities for $T=0$

FIGURE 2.27 Nonmonotone optimal threshold schedule [390]. (a) Represents the search landscape of the problem. (b) (Resp. (c)) represents the transition probabilities for $T = 2 (T = 0)$.

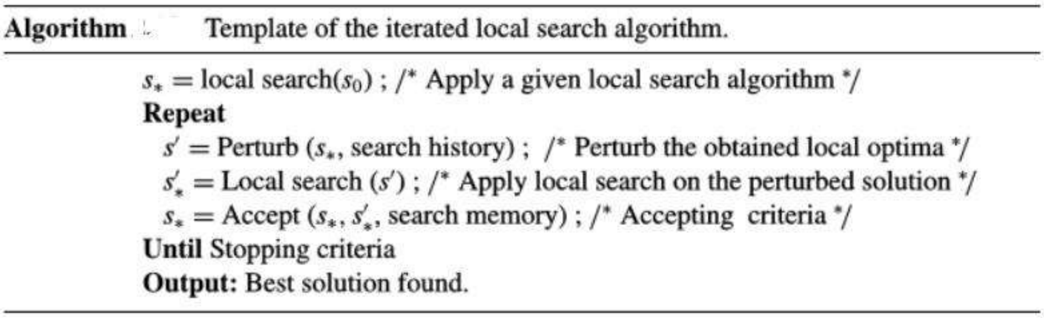
Threshold Accepting (TA) is a local search method and was first described by Dueck

Iterated local search

The quality of the local optima obtained by a local search method depends on the initial solution.

- As we can generate local optima with high variability, iterated local search may be used to improve the quality of successive local optima.

First, a local search is applied to an initial solution. Then, at each iteration, a perturbation of the obtained local optima is carried out. Finally, a local search is applied to the perturbed solution. The generated solution is accepted as the new current solution under some conditions. This process iterates until a given stopping criterion. Algorithm bellow describes the ILS algorithm



Algorithm Template of the iterated local search algorithm.

```
$s_*$ = local search($s_0$) ; /* Apply a given local search algorithm */
Repeat
  $s'$ = Perturb ($s_*$, search history) ;  /* Perturb the obtained local optima */
  $s'_*$ = Local search ($s'$) ; /* Apply local search on the perturbed solution */
  $s_*$ = Accept ($s_*$, $s'_*$, search memory) ; /* Accepting  criteria */
Until Stopping criteria
Output: Best solution found.
```

Three basic elements compose an ILS (Fig. bellow):

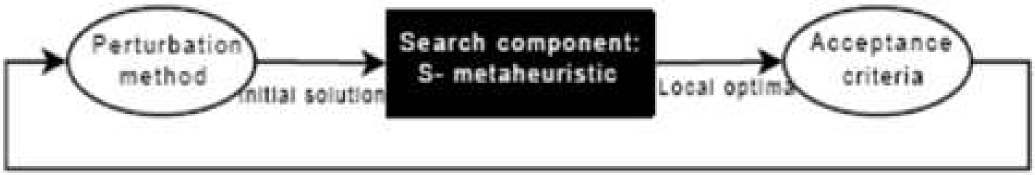
• Local search: Any S-metaheuristic (deterministic or stochastic) can be used in the ILS framework such as a simple descent algorithm, a tabu search, or simulated annealing. The search procedure is treated as a black box (Fig. bellow). In the literature, P-metaheuristics are not considered as candidates in the search procedure as they manipulate populations. However, some population-based metaheuristics integrate the concept of perturbation of the (sub)population to encourage the search diversification.

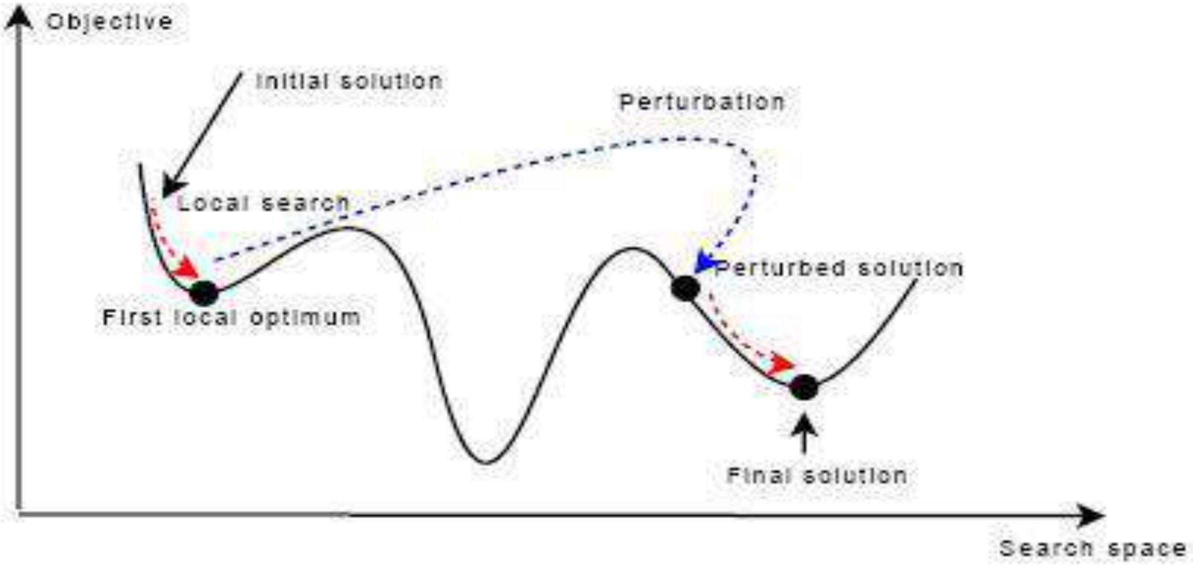
• Perturbation Method. The perturbation operator may be seen as a large random move of the current solution. The perturbation method should keep some part of the solution and perturb strongly another part of the solution to move hopefully to another basin of attraction.

• Acceptance criteria. The acceptance criterion defines the conditions the new local optima must satisfy to replace the current solution.

Once the S-metaheuristic involved in the ILS framework is specified, the design of ILS will depend mainly on the used perturbation method and the acceptance criterion.

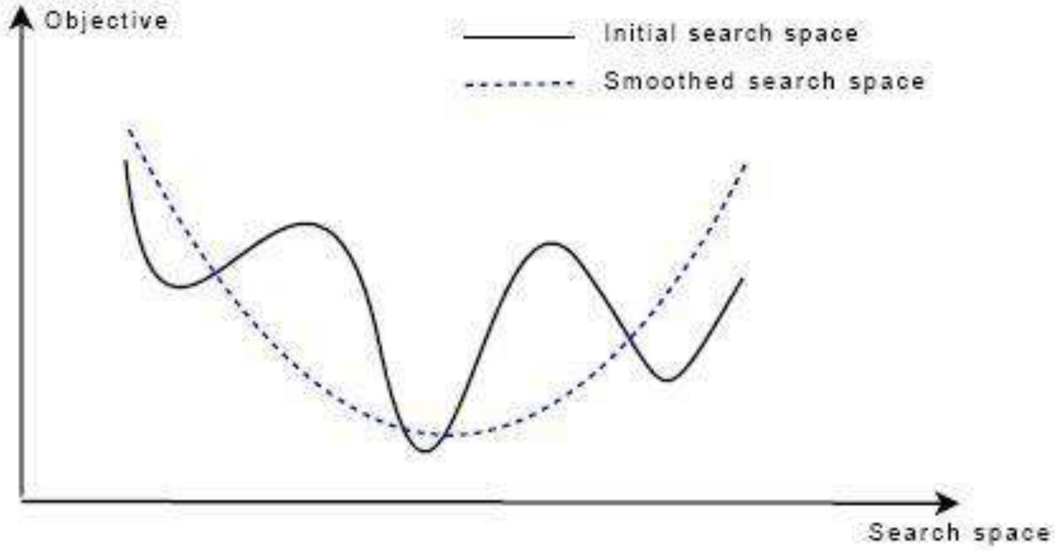
Many different designs may be defined according to various choices for implementing the perturbation method and the acceptance criterion.





Smoothing algorithm : Let’s have a dream

- Change the landscape of the target optimization problem
- Reduce the number of the local optima to smooth the landscape



Successive Smoothing

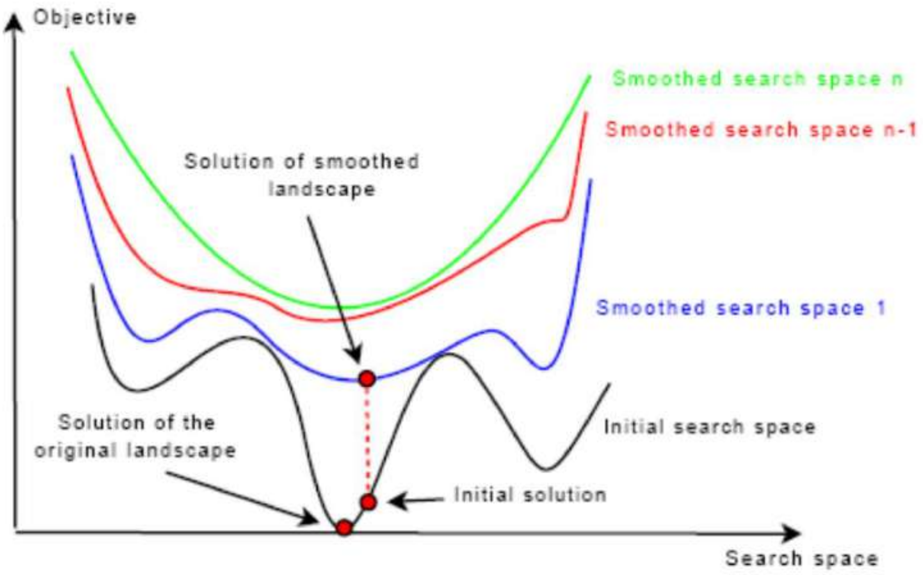
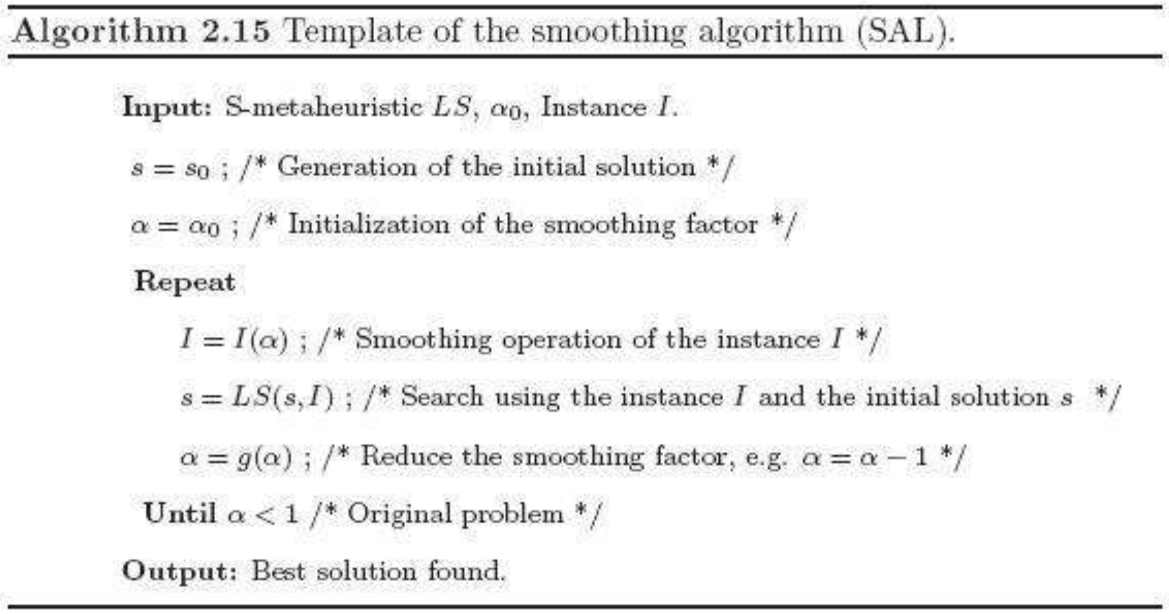


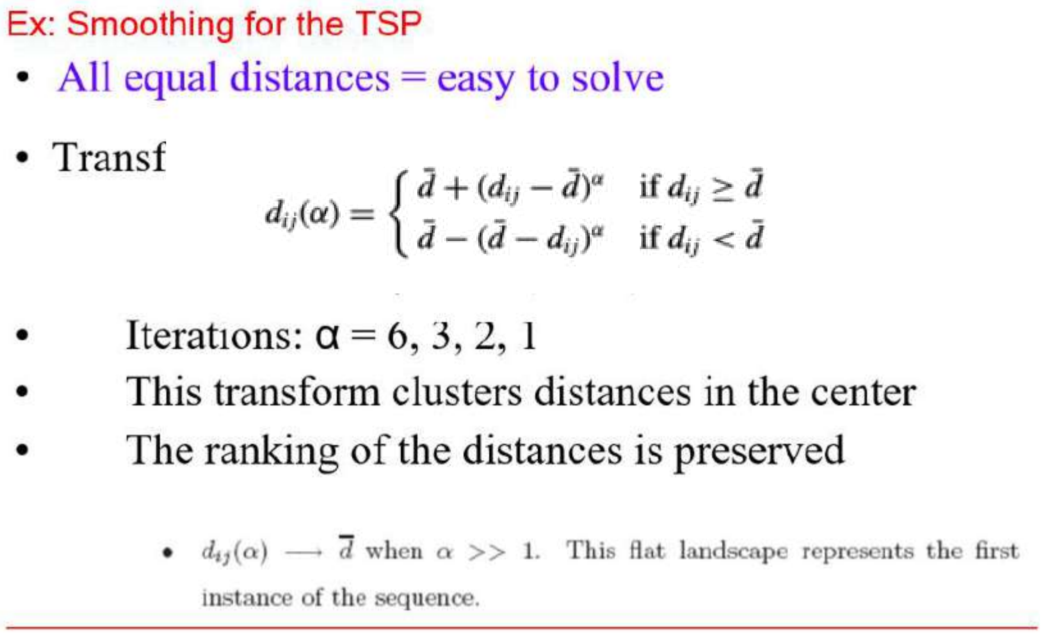
Fig. 2.34 Successive smoothing of the landscape. The solution found at the step i



Algorithm 2.15 Template of the smoothing algorithm (SAL).

```
Input: S-metaheuristic $LS$, $\alpha_0$, Instance $I$.
$s = s_0$ ; /* Generation of the initial solution */
$\alpha = \alpha_0$ ; /* Initialization of the smoothing factor */
Repeat
    $I = I(\alpha)$ ; /* Smoothing operation of the instance $I$ */
    $s = LS(s, I)$ ; /* Search using the instance $I$ and the initial solution $s$ */
    $\alpha = g(\alpha)$ ; /* Reduce the smoothing factor, e.g. $\alpha = \alpha - 1$ */
Until $\alpha < 1$ /* Original problem */
Output: Best solution found.
```

Ex: Smoothing for the TSP



Ex: Smoothing for the TSP

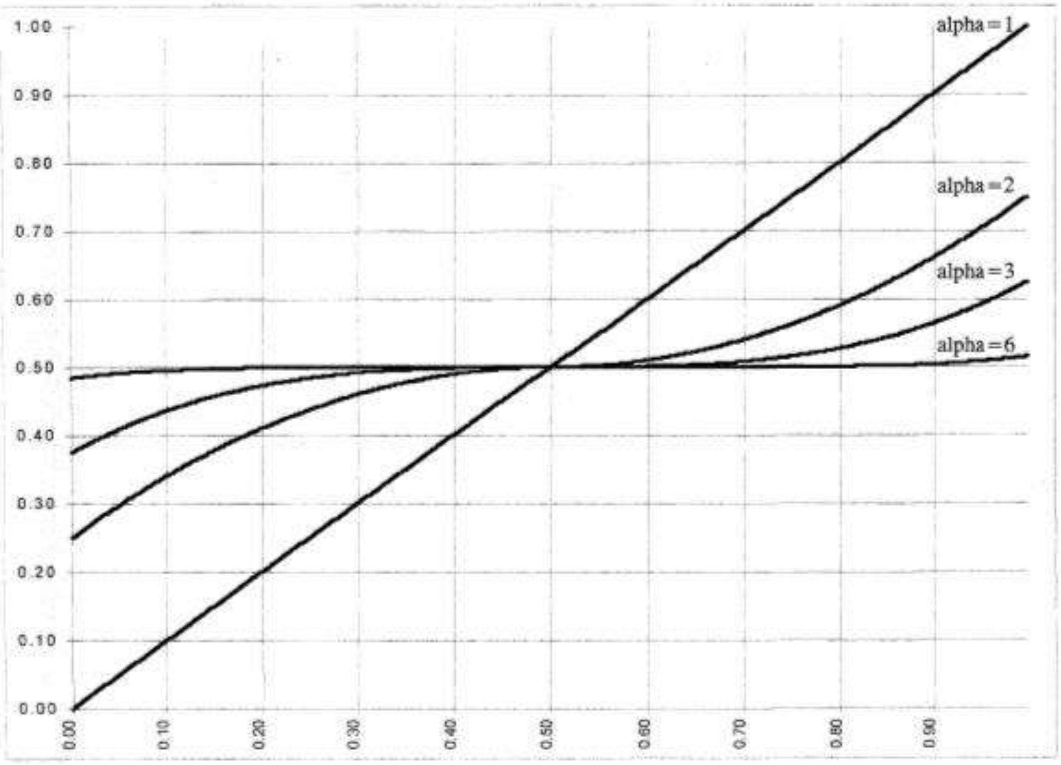
- All equal distances = easy to solve
- Transf

$$d_{ij}(\alpha) = \begin{cases} \bar{d} + (d_{ij} - \bar{d})^{\alpha} & \text{if } d_{ij} \geq \bar{d} \\ \bar{d} - (\bar{d} - d_{ij})^{\alpha} & \text{if } d_{ij} < \bar{d} \end{cases}$$

- Iterations: $\alpha = 6, 3, 2, 1$
- This transform clusters distances in the center
- The ranking of the distances is preserved

  - $d_{ij}(\alpha) \longrightarrow \bar{d}$ when $\alpha >> 1$. This flat landscape represents the first instance of the sequence.

Smoothing Transformation



Variable neighborhood search

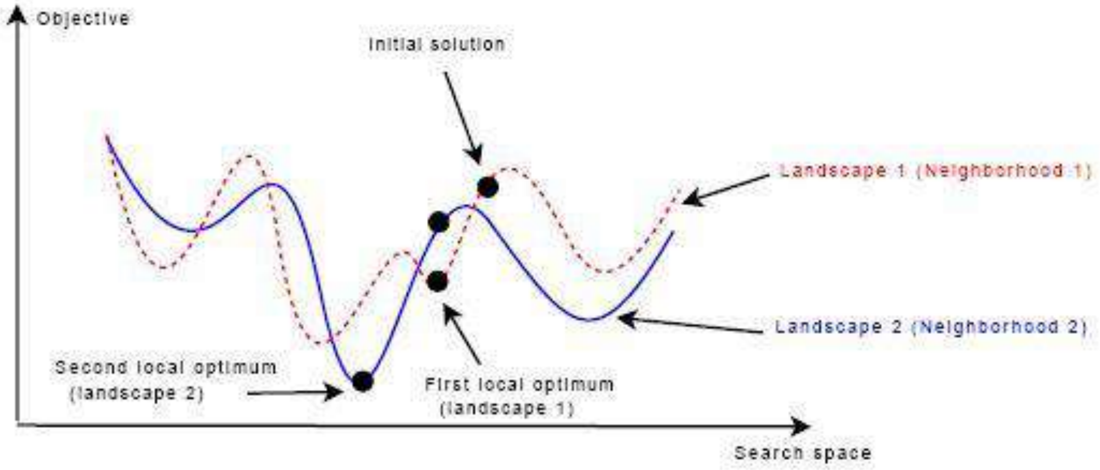
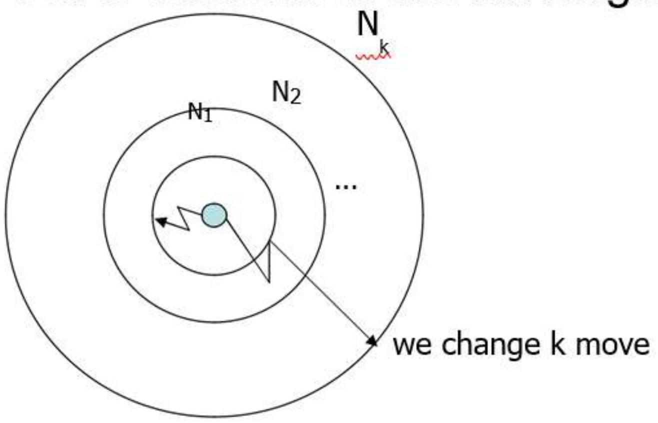


Fig. 2.30 Variable neighborhood search using two neighborhoods. The first local optimum is obtained according to the neighborhood 1. According to the neighborhood 2, the second local optimum is obtained from the first local optimum.

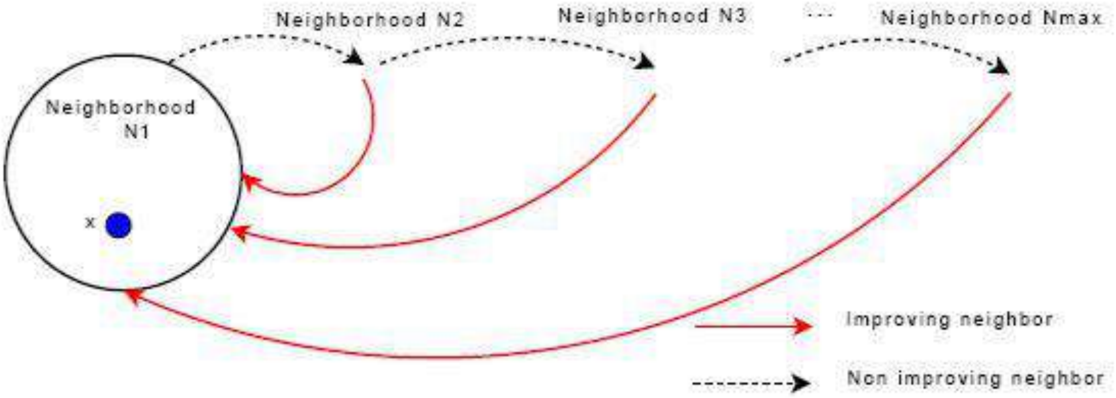
- Different neighborhoods
  - Ex: N_k ➪ Neighborhood in k distance
- Order of exploration
  - Ex: $N_k(x)$ ➪ Set of solutions in the kth neighborhood of x.

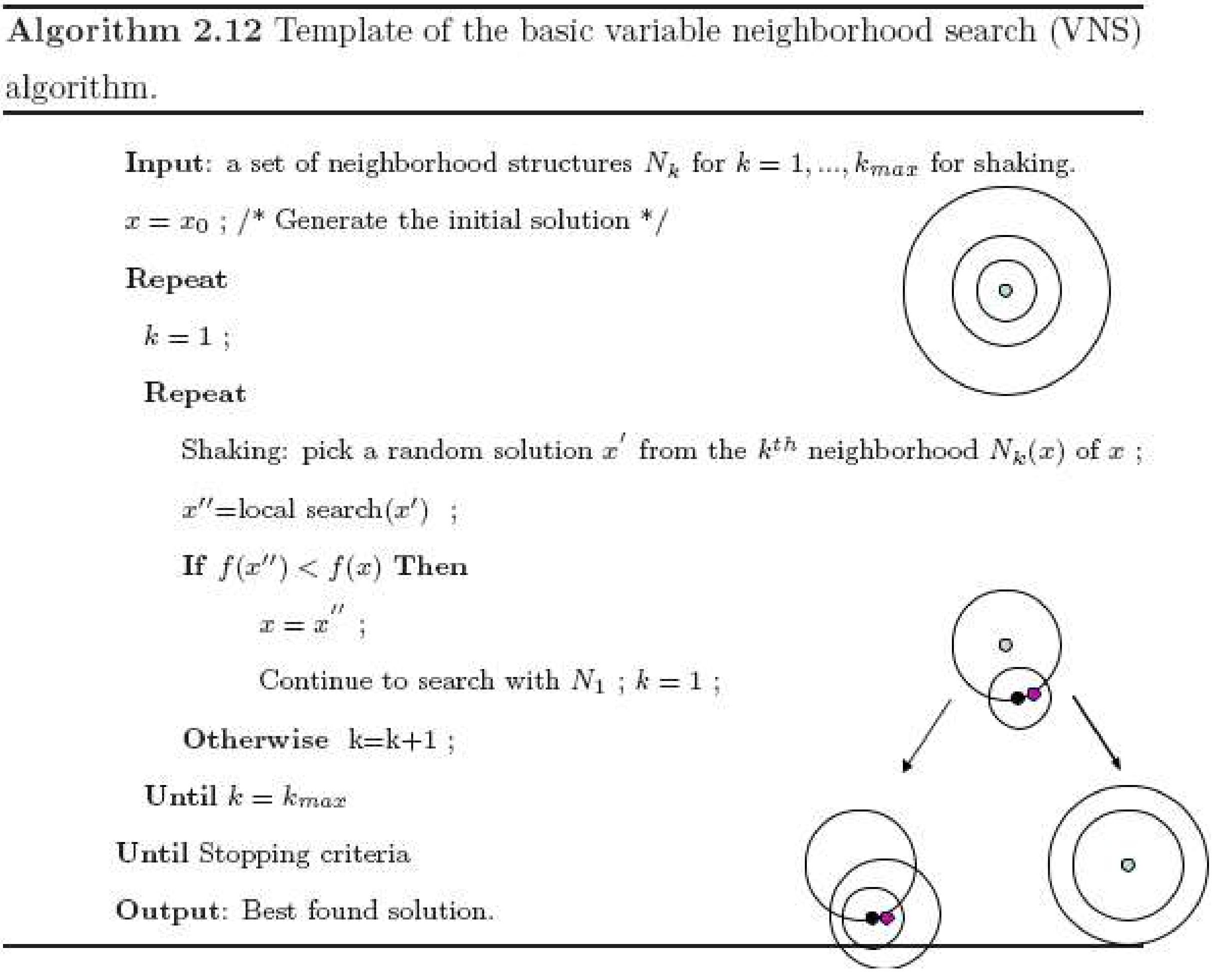


VND: Variable Neighbourhood descent

- **VND (*Variable Neighborhood Descent*)** changes the neighbourhoods N_k each time a local optimum is reached.
- It ends when there is no improve with the all neighborhoods

The final solution provided by the algorithm should be a local optimum with respect to all k_{max} neighborhoods





Algorithm 2.12 Template of the basic variable neighborhood search (VNS) algorithm.

```
Input: a set of neighborhood structures $N_k$ for $k = 1, ..., k_{max}$ for shaking.
$x = x_0$ ; /* Generate the initial solution */
Repeat
  $k = 1$ ;
  Repeat
    Shaking: pick a random solution $x'$ from the $k^{th}$ neighborhood $N_k(x)$ of $x$ ;
    $x''$=local search($x'$) ;
    If $f(x'') < f(x)$ Then
        $x = x''$ ;
        Continue to search with $N_1$ ; $k = 1$ ;
    Otherwise k=k+1 ;
  Until $k = k_{max}$
Until Stopping criteria
Output: Best found solution.
```

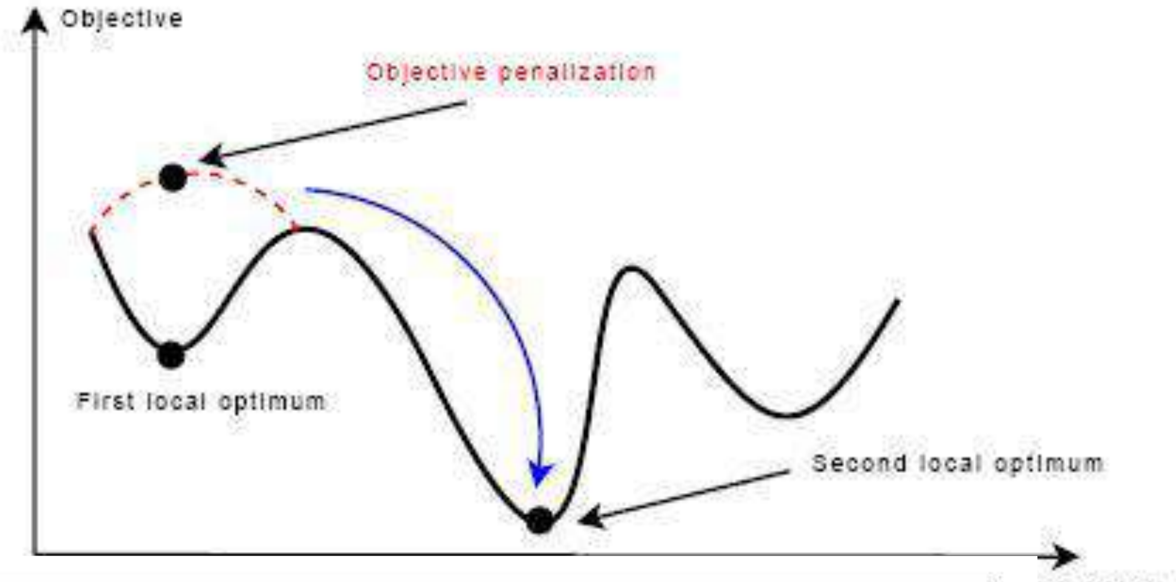
Algorithm 2.13 Template of the general variable neighborhood search (GVNS) algorithm.

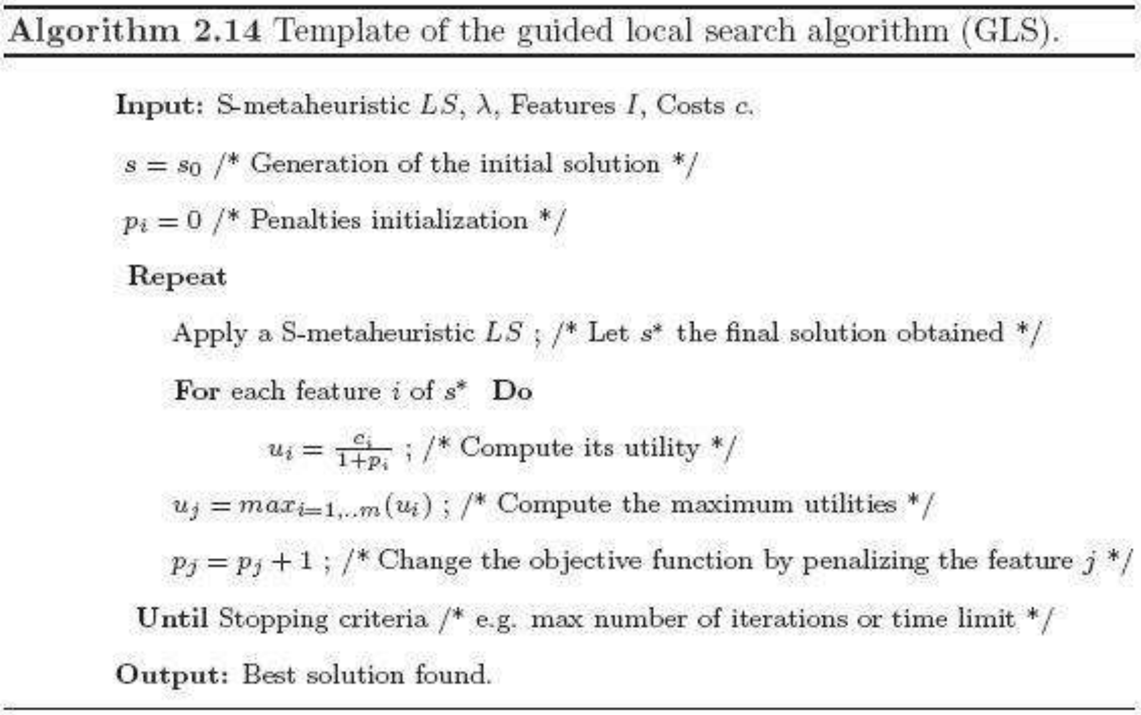
```
Input: a set of neighborhood structures $N_k$ for $k = 1, ..., k_{max}$ for shaking.
            a set of neighborhood structures $N_l$ for $k = 1, ..., l_{max}$ for local search.
$x = x_0$ ; /* Generate the initial solution */
Repeat
   For k=1 To $k_{max}$ Do
   Shaking: pick a random solution $x^{'}$ from the $k^{th}$ neighborhood $N_k(x)$ of $x$ ;
   Local search by VND ;
   For l=1 To $l_{max}$ Do
       Find the best neighbor $x^{''}$ of $x^{'}$ in $N_l(x^{'})$ ;
       If $f(x^{''}) < f(x^{'})$ Then $x^{'} = x^{''}$ ; l=1 ;
       Otherwise l=l+1 ;
       Move or not:
       If local optimum is better than $x$ Then
          $x = x^{''}$ ;
          Continue to search with $N_1$ $(k = 1)$ ;
       Otherwise k=k+1 ;
Until Stopping criteria
Output: Best found solution.
```

Guided local search

- Dynamic changing of the objective function according to the generated local optima

- The features of the local optima are used to change the objective function





Algorithm 2.14 Template of the guided local search algorithm (GLS).

```
Input: S-metaheuristic $LS$, $\lambda$, Features $I$, Costs $c$.
$s = s_0$ /* Generation of the initial solution */
$p_i = 0$ /* Penalties initialization */
Repeat
    Apply a S-metaheuristic $LS$ ; /* Let $s^*$ the final solution obtained */
    For each feature $i$ of $s^*$ Do
        $u_i = \frac{c_i}{1+p_i}$ ; /* Compute its utility */
    $u_j = max_{i=1,..m}(u_i)$ ; /* Compute the maximum utilities */
    $p_j = p_j + 1$ ; /* Change the objective function by penalizing the feature $j$ */
Until Stopping criteria /* e.g. max number of iterations or time limit */
Output: Best solution found.
```

Guided local search: Features

- Identification of suitable features for a given optimization problem:
  - – Routing problems: edges (cost: distance, travel time)
  - – Assignment problems: pair (i,k), i: object, k: location, cost=assignment cost of i on k

– Satisfiability (SAT): constraint (cost: constraint violation

Tabu search

Main characteristics

- Proposed independently by Glover (1986) and Hansen (1986)
- It behaves like Hill Climbing algorithm

- But it accepts non-improving solutions in order to escape from local optima (where all the neighbouring solutions are non-improving)

Deterministic algorithm

- After exploring the neighbouring solutions, we accept the best one even if it decreases the cost function.

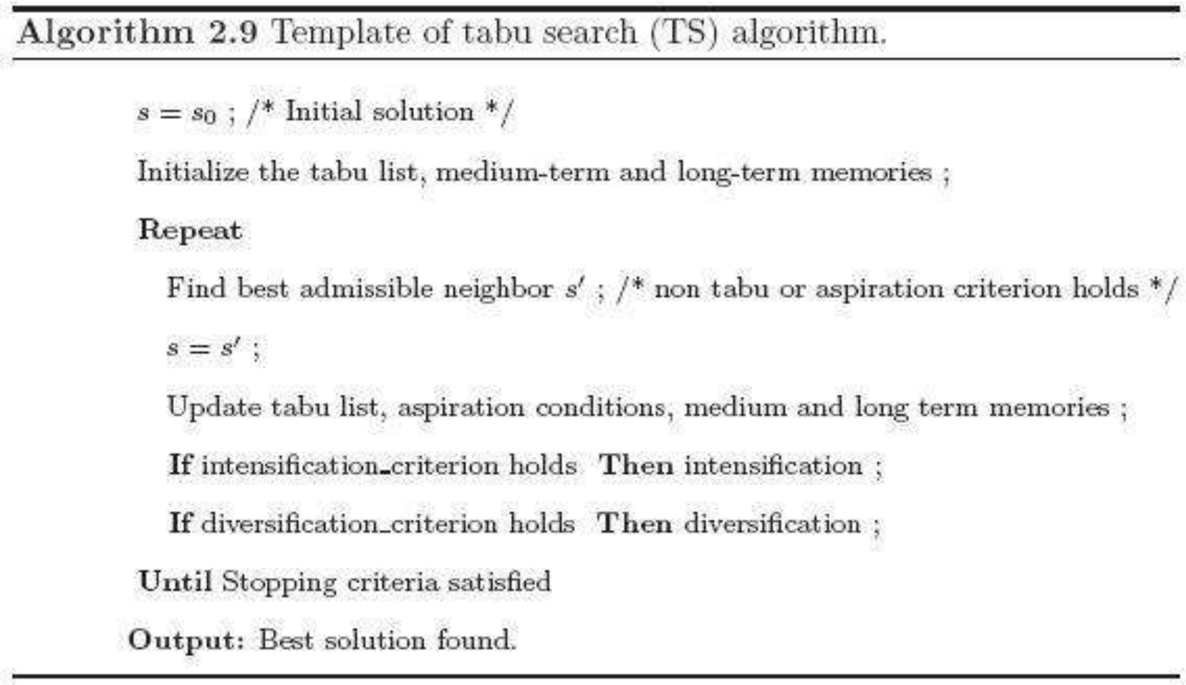
  → A worse move may be accepted.

- Three goals in the use of memory:

  - – Preventing the search from revisiting previously visited solutions (tabu list).

  - – Exploring the unvisited areas of the solution space (diversification).

  - – Exploiting the elite (best found) solutions (intensification).



Algorithm 2.9 Template of tabu search (TS) algorithm.

```
$s = s_0$ ; /* Initial solution */
Initialize the tabu list, medium-term and long-term memories ;
Repeat
    Find best admissible neighbor $s'$ ; /* non tabu or aspiration criterion holds */
    $s = s'$ ;
    Update tabu list, aspiration conditions, medium and long term memories ;
    If intensification_criterion holds Then intensification ;
    If diversification_criterion holds Then diversification ;
Until Stopping criteria satisfied
Output: Best solution found.
```

Tabu Search

- is a metaheuristic that guides a local search procedure to explore the solution space beyond local optimality.
- One of the features of tabu search is to avoid bad solutions which have already been explored i.e use of memory to guide the search by using tabu list to record recent searches.
- Essentially, Tabu search makes some moves illegal by maintaining a list of 'tabu' moves.

For example, if A is a neighbor of B in the TSP then B is a

neighbor of A. But if you have already chosen B over A, there might not be any reason to search A again.

1. Tabu search is a local search strategy with a flexible memory structure. Simulated annealing has no memory structure (it relies on randomization) Tabu search has two prominent features:

- Adaptive memory

- Reasonable exploration strategies

2. Main features: always move to the best available neighborhood solution point, even if it is worse than the current solution point.

3. intensification and diversification

- Intensify: To intensify the search is to search the local area (portion of feasible region) more thoroughly.

- Diversify: To diversify the search is to force the search away from the current solution (to unexplored areas of feasible region).

- Length of Tabu List: The length of the list signifies the balance between intensify/diversify.

5. Long term memory

Travelling Salesman Example

Initial trial solution: 1-2-3-4-5-6-7-1 Distance = 69 Tabu

- list : Null
- Iteration 1: Reverse 3-4
- Deleted links: 2-3 and 4-5
- Added links : 2-4 and 3-5
- Tabu List: (2-4), (3-5)
- New trials Solution: 1-2-4-3-5-6-7-1 Distance = 65
- Iteration 2: Reverse: 3-5-6
- Delete links: 4-3 and 6-7
- Add links: 4-6 and 3-7
- Tabu List: 2-4, 3-5, 4-6, 3-7
- New Solution: 1-2-4-6-5-3-7-1 Distance = 64

- Advantage:
- This method is easy to integrate with other methods.
- Disadvantage:
- Tabu search approach is to climb the hill in the steepest
- direction and stop at top and then climb downwards to
- search for another hill to climb. The drawback is that a lot of
- iterations are spent climbing each hill rather than searching
- for tallest hill.

GRASP algorithm (Greedy Randomized Adaptive Search Procedure)

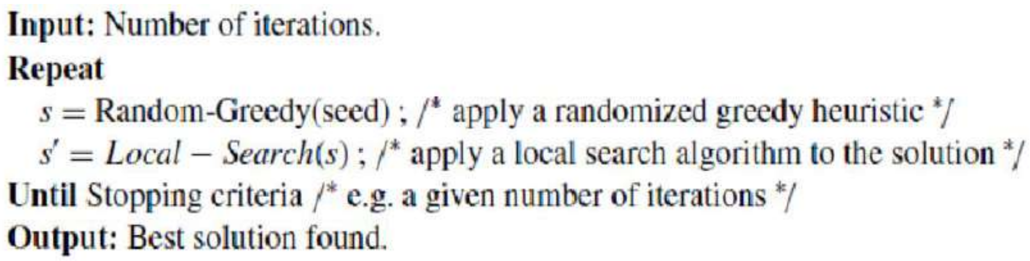
The GRASP metaheuristic is an iterative greedy heuristic to solve combinatorial optimization problems. It was introduced in 1989. Each iteration of the GRASP algorithm contains two steps: construction and local search. In the construction step, a feasible solution is built using a randomized greedy algorithm, while in the Improving step a local search heuristic is applied from the constructed solution.

The GRASP metaheuristic has two phases:

- Constructive phase: constructs a good initial solution *x*.
- Improving phase: applied to the initial solution *x*.

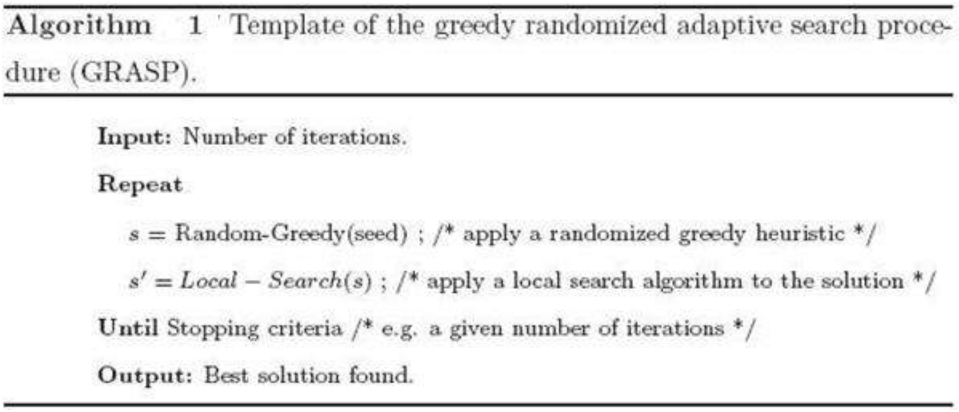
Both phases are repeated until the stopping criterion is met.

Greedy Randomized Adaptive Search Procedure



```
Input: Number of iterations.
Repeat
    $s$ = Random-Greedy(seed) ; /* apply a randomized greedy heuristic */
    $s' = Local - Search(s)$ ; /* apply a local search algorithm to the solution */
Until Stopping criteria /* e.g. a given number of iterations */
Output: Best solution found.
```

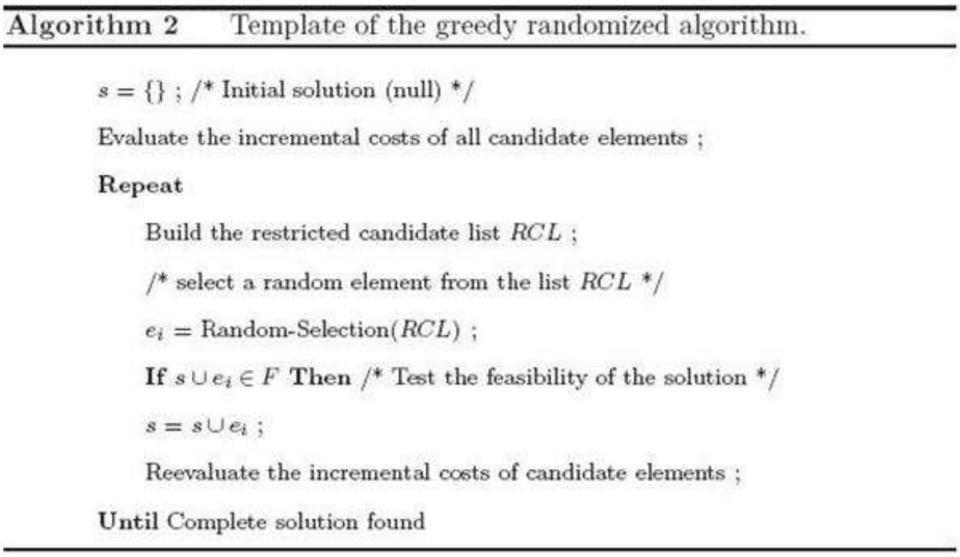
GRASP (Greedy Randomized Adaptive Search Procedure)



Algorithm 1 Template of the greedy randomized adaptive search procedure (GRASP).

```
Input: Number of iterations.
Repeat
    $s$ = Random-Greedy(seed) ; /* apply a randomized greedy heuristic */
    $s' = Local - Search(s)$ ; /* apply a local search algorithm to the solution */
Until Stopping criteria /* e.g. a given number of iterations */
Output: Best solution found.
```

Greedy randomized algorithm



Algorithm 2 Template of the greedy randomized algorithm.

```
$s = \{\}$ ; /* Initial solution (null) */
Evaluate the incremental costs of all candidate elements ;
**Repeat**
    Build the restricted candidate list $RCL$ ;
    /* select a random element from the list $RCL$ */
    $e_i$ = Random-Selection($RCL$) ;
    **If** $s \cup e_i \in F$ **Then** /* Test the feasibility of the solution */
    $s = s \cup e_i$ ;
    Reevaluate the incremental costs of candidate elements ;
**Until** Complete solution found
```

Population solution metaheuristics

Population-based metaheuristics (P-metaheuristics) share many common concepts. They could be viewed as an iterative improvement in a population of solutions.

First, the population is initialized. Then, a new population of solutions is generated. Finally, this new population is integrated into the current one using some selection pro-cedures. The search process is stopped when a given condition is satisfied (stopping criterion).

Algorithms such as evolutionary algorithms (EAs), scatter search (SS), estimation of distribution algorithms (EDAs), particle swarm optimization (PSO), bee colony (BC), and artificial immune systems (AISs) belong to this class of metaheuristics

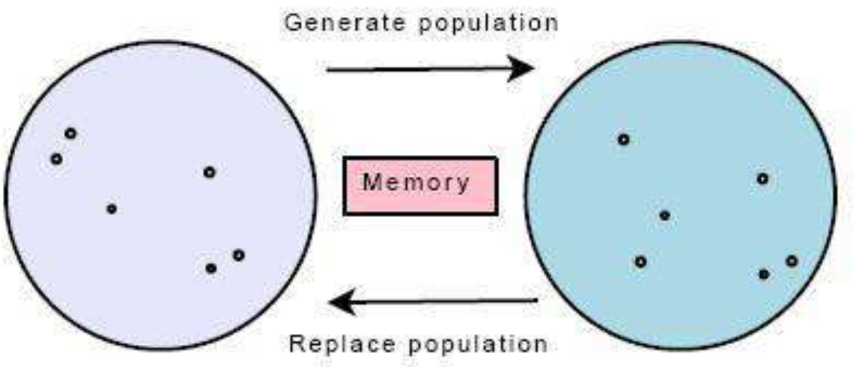
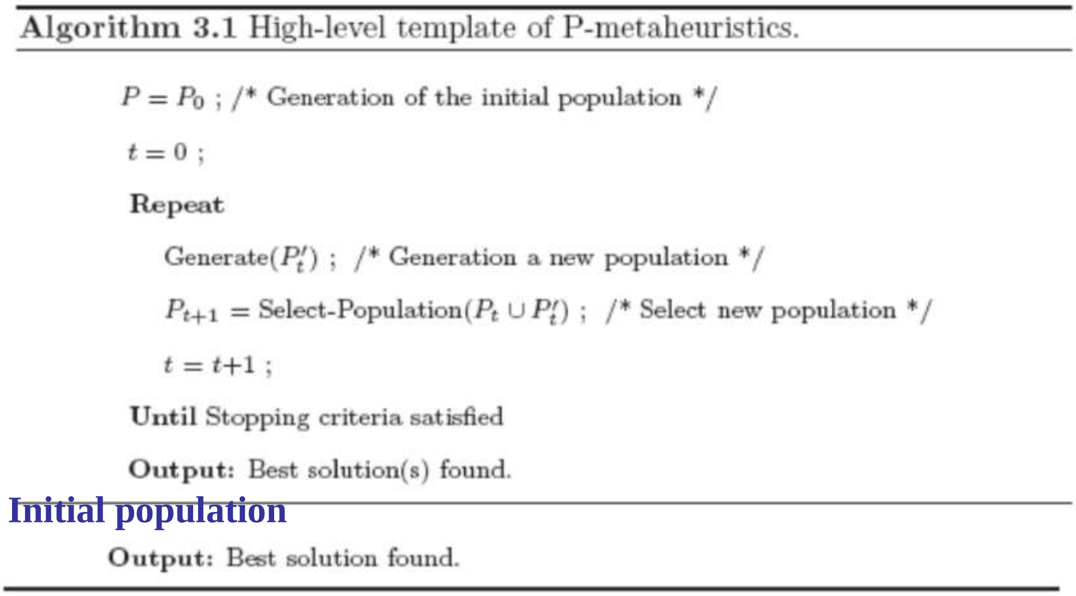


FIGURE 3.1 Main principles of P-metaheuristics.



Algorithm 3.1 High-level template of P-metaheuristics.

```
$P = P_0$ ; /* Generation of the initial population */
$t = 0$ ;
Repeat
    Generate($P'_t$) ;  /* Generation a new population */
    $P_{t+1}$ = Select-Population($P_t \cup P'_t$) ;  /* Select new population */
    $t = t+1$ ;
Until Stopping criteria satisfied
Output: Best solution(s) found.
```

Initial population

Output: Best solution found.

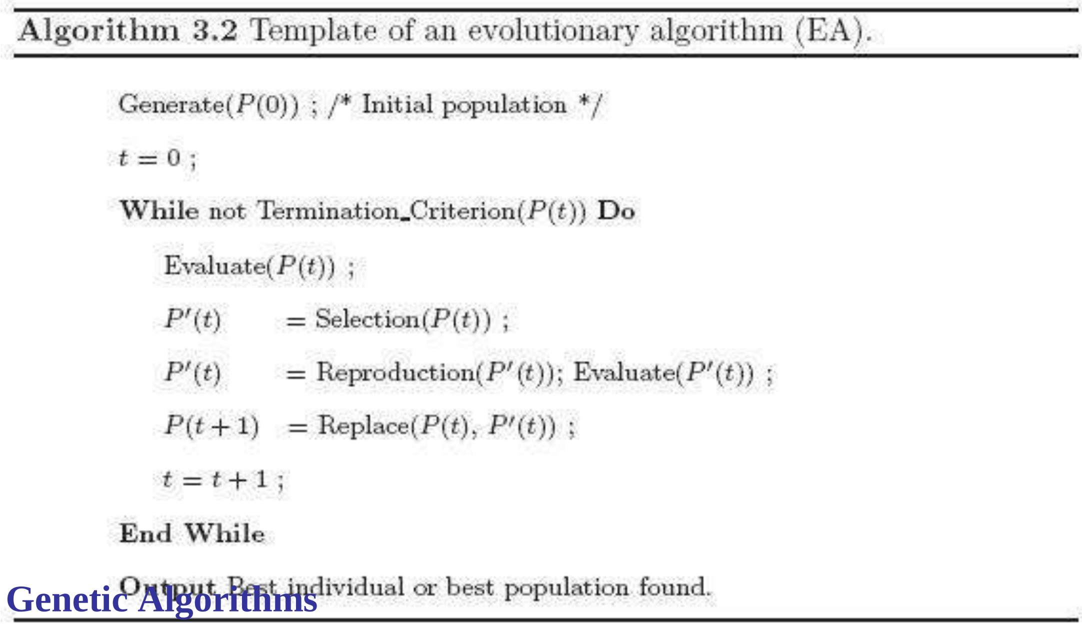
- Random generation

  - Pseudo-random
  - Quasi-random
- Sequential diversification
- Parallel diversification
- Heuristic initialization

Stopping criteria

- Static procedure: known a priori
  - Number of iterations
  - CPU time
  - Number of evaluations
- Adaptive procedure
  - Number of iterations without improvements
  - Diversity of the population
  - Optimal or satisfactory solution is reached

Evolutionary Algorithm procedure



Algorithm 3.2 Template of an evolutionary algorithm (EA).

```
Generate($P(0)$) ; /* Initial population */
$t = 0$ ;
While not Termination_Criterion($P(t)$) Do
    Evaluate($P(t)$) ;
    $P'(t)$     = Selection($P(t)$) ;
    $P'(t)$     = Reproduction($P'(t)$); Evaluate($P'(t)$) ;
    $P(t+1)$   = Replace($P(t)$, $P'(t)$) ;
    $t = t + 1$ ;
End While
Output Best individual or best population found.
```

Genetic Algorithms

Memetic algorithm

- Meme - an idea, behavior, style, or usage that spreads from person to person within a culture
- Population-based approach for heuristic search in optimization problems.
- They combine local search heuristics with crossover operators to mimic cultural evolution. Viewed by some researchers as Hybrid Genetic Algorithms.
- Other researchers known it as **Genetic Local Search**.
- “the basic unit of cultural transmission, or imitation”

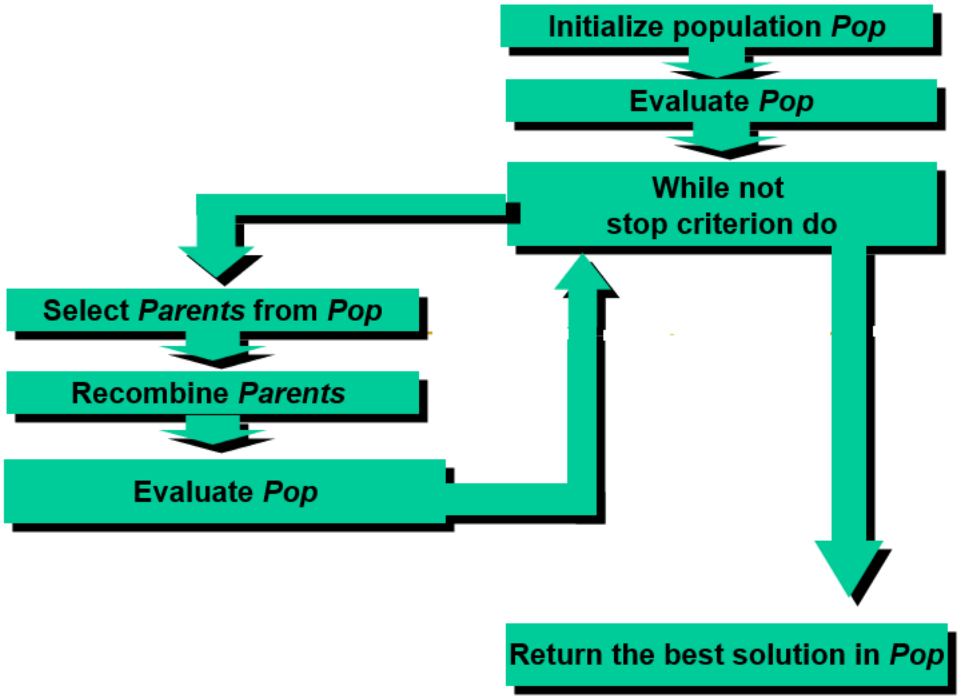
Genes and Memes, where they are similar

- Genes propagate biologically from chromosome to chromosome
- Memes propagate from brain to brain via imitation
- Survival of fittest in meme
- Genes are pre-decided
- Genes are (almost always) static through generations, memes can be changed!

- Memes allow improvement
- New heuristics to search problems
- We use this property to improve algorithms

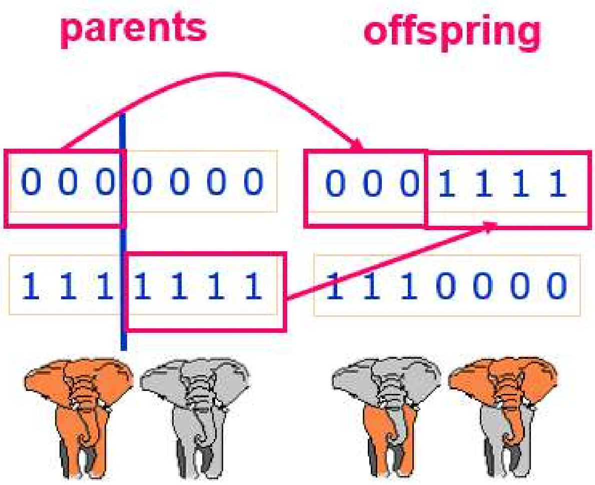
Genetic Algorithm

- solves (typically optimization) problems by combining features of candidate solutions to create new populations of candidate solutions.
- applicable to complex optimization problems, e.g., when it is hard to tackle the problem via an exact search approach.


Initialize population *Pop*
Evaluate *Pop*
While not stop criterion do
Select *Parents* from *Pop*
Recombine *Parents*
Evaluate *Pop*
Return the best solution in *Pop*

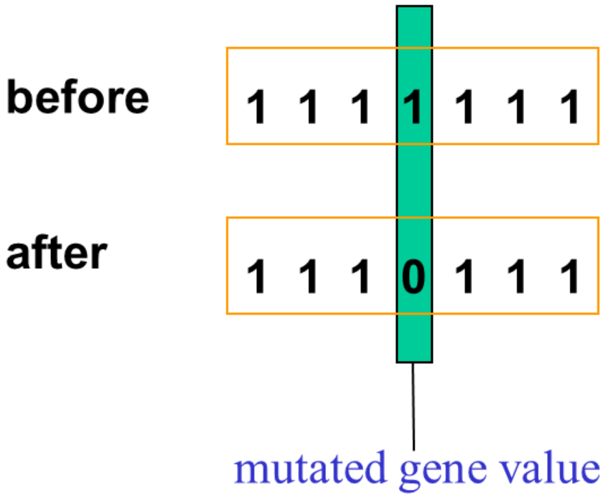
Crossover

- Purpose: to combine features of feasible solutions already visited in order to provide new potential candidate solutions with better objective function value.
-
- Mechanism that restarts the search by "exploring" the space "between" solutions.



Mutation

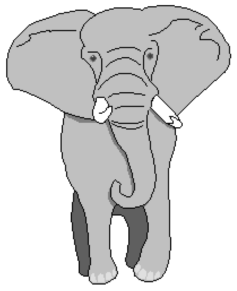
- Purpose: to introduce new characteristics in the population by random modifications.
- Explores the "neighborhood" of a solution.



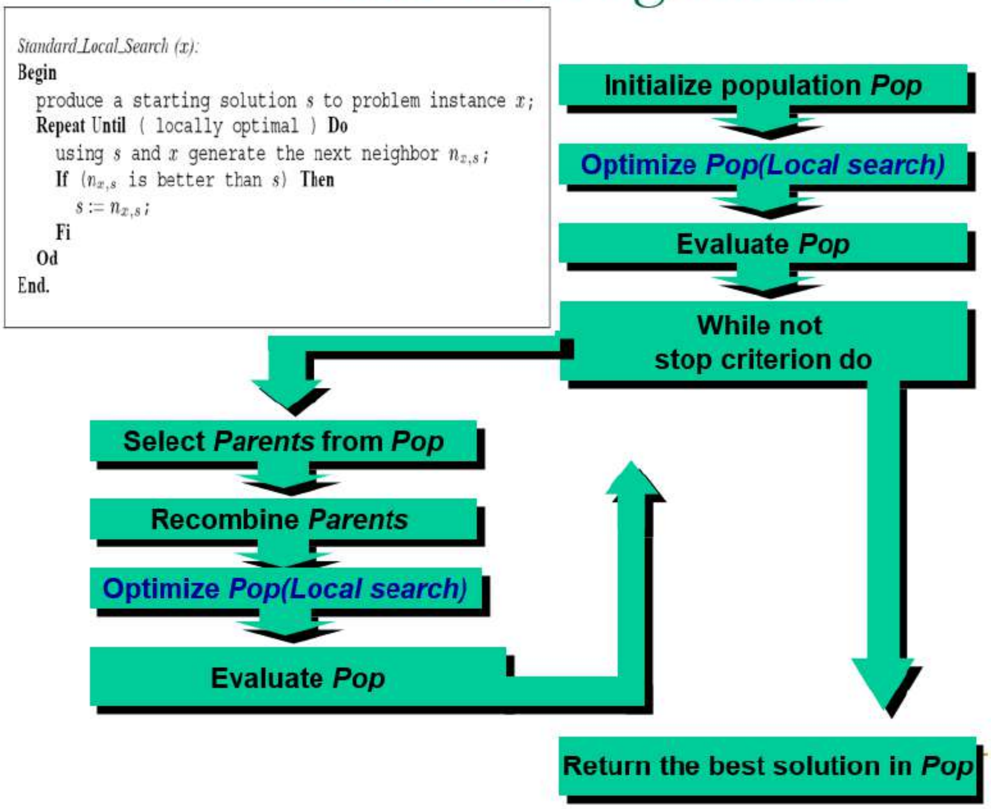
before 1 1 1 1 1 1 1

after 1 1 1 0 1 1 1

mutated gene value



Memetic Algorithm



```
Standard_Local_Search (x):
Begin
  produce a starting solution s to problem instance x;
  Repeat Until ( locally optimal ) Do
    using s and x generate the next neighbor $n_{x,s}$;
    If ($n_{x,s}$ is better than s) Then
      $s := n_{x,s}$;
    Fi
  Od
End.
```

Initialize population *Pop*

Optimize *Pop(Local search)*

Evaluate *Pop*

While not stop criterion do

Select *Parents* from *Pop*

Recombine *Parents*

Optimize *Pop(Local search)*

Evaluate *Pop*

Return the best solution in *Pop*

15: Cultural Algorithms

Cultural algorithms (CAs) are special variants of evolutionary algorithms. CAs have been introduced by R. G. Reynolds in 1994. They are computational models of cultural evolution based upon principles of human social evolution. They employ a model of cultural change within an optimization problem, where culture might be symbolically represented and transmitted between successive populations. The main principle behind this process is to preserve beliefs that are socially accepted and discard unacceptable beliefs.

Cultural algorithms contain two main elements, a **population spac**e at the microevolutionary level and a **belief space** at the macroevolutionary level (Fig. 3.28). The two elements interact by means of a vote–inherit–promote or VIP protocol. This enables the individuals to alter the belief space and allows the belief space to influence the ways in which individuals evolve. The population space at the microevolutionary level may be carried out by EAs. At each generation, the knowledge acquired by the search of the population (e.g., best solutions of the population) can be memorized in the belief space in many forms such as logic-and rule-based models, schemata, graphical models, semantic networks, and version spaces [559] among others to model the macroevolutionary process of a cultural algorithm.

The belief space is divided into distinct categories that represent different domains of knowledge that the population has acquired on the search space:

normative knowledge (i.e., a collection of desirable value ranges for some decision variables of the individuals in the population),
domain specific knowledge (i.e., information about the domain of the problem CA is applied to),
situational knowledge, Specific examples of important events - e.g. successful/unsuccessful solutions
temporal knowledge (i.e., information of important events about the search), and spatial **knowledge** (i.e., information about the landscape of the tackled optimization problem).
Algorithm 3.9 illustrates the template of a cultural algorithm.
As such, cultural algorithms represent a P-metaheuristic based on hybrid evolutionary systems that integrate evolutionary search and symbolic reasoning.

They are particularly useful for problems whose solutions require extensive domain knowledge (e.g., constrained optimization problems) and dynamic environments (e.g., dynamic optimization problems).

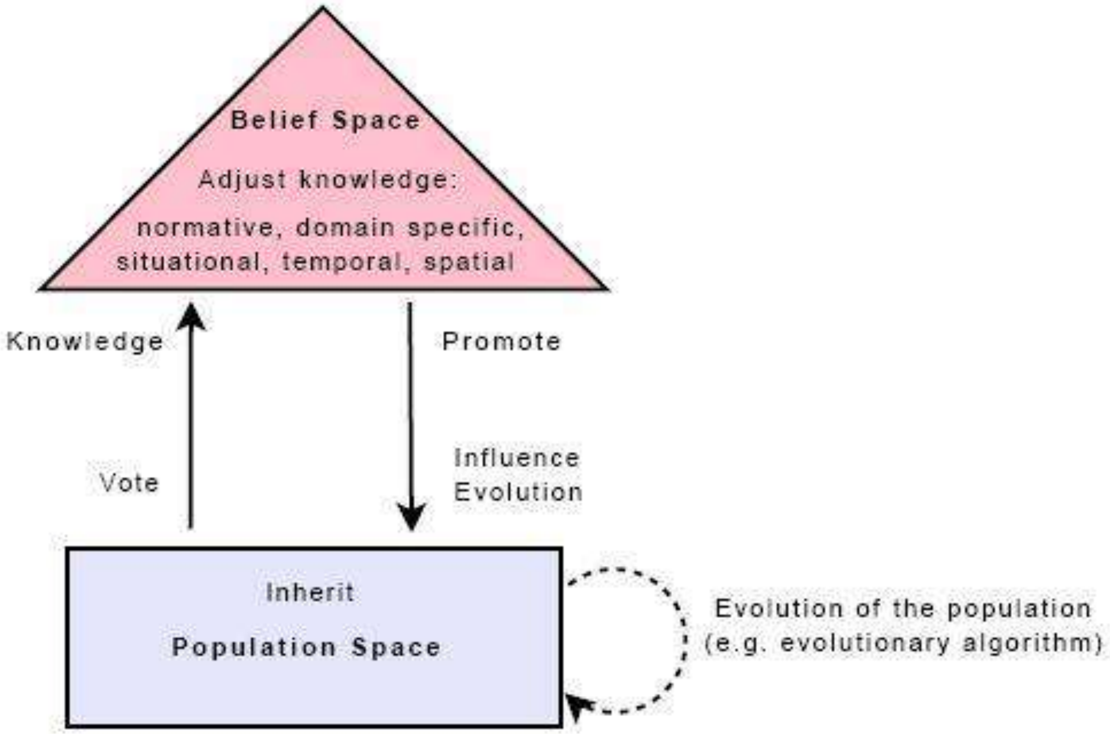
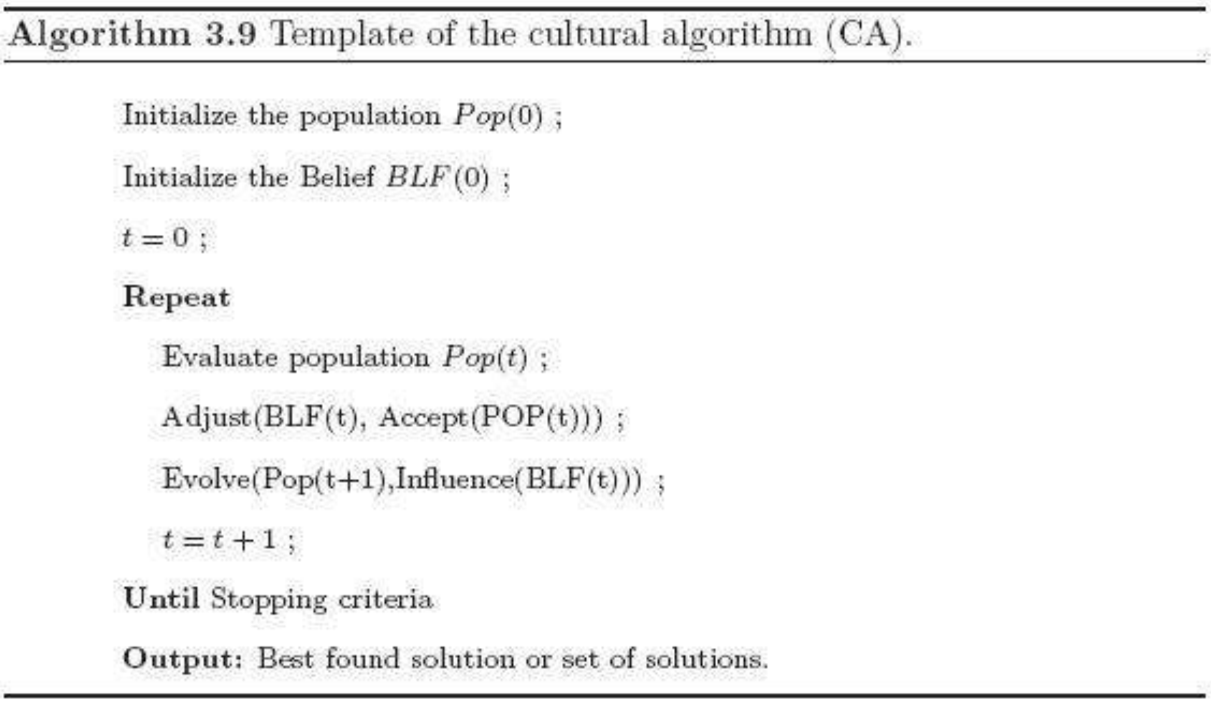

Belief Space
Adjust knowledge:
normative, domain specific,
situational, temporal, spatial
Knowledge
Vote
Promote
Influence
Evolution
Inherit
Population Space
Evolution of the population
(e.g. evolutionary algorithm)

Fig. 3.28 Search components of cultural algorithms.



Algorithm 3.9 Template of the cultural algorithm (CA).

```
Initialize the population $Pop(0)$ ;
Initialize the Belief $BLF(0)$ ;
$t = 0$ ;
**Repeat**
    Evaluate population $Pop(t)$ ;
    Adjust(BLF(t), Accept(POP(t))) ;
    Evolve(Pop(t+1),Influence(BLF(t))) ;
    $t = t + 1$ ;
**Until** Stopping criteria
**Output:** Best found solution or set of solutions.
```