COVERAGE

COVER: NODE PLACEMENT OPTIMIZATION

In the point coverage problem, the subject to be covered to be some particular space points the representing the sensor field (discrete, e.g., the vertices of a grid).

They could represent or model some physical targets in the sensor field (e.g., the missile launchers in a battlefield).

In order to cover these points, sensor nodes can be deterministically placed or randomly deployed in the sensor field.

In a random network deployment, it is possible that some nodes cannot cover even a single target point.

On the other hand, if the sensor field is friendly reachable and the network size is not too large, deterministic node placement is to place the nodes only at the desired locations so that each sensor node can cover at least one target.

DETERMINISTIC NODE PLACEMENT

The objective of a deterministic node placement can be summarized as to answer the following question:

What are the optimal locations (*among the available locations*) *to place the minimum number of sensor nodes* (*or minimum network cost*) while achieving the required coverage.

The problem of placing the least number of sensors to cover all target points is a variant of the canonical set-covering problem.

NODE PLACEMENT AS THE SET-COVERING PROBLEM

Assume known locations of the targets (i.e., the space points) to be covered (sites).

The problem of placing the least number of sensors to cover all discrete targets can be equated to the canonic *set-covering problem*.

- X denotes the finite set of targets with known locations.
- A sensor node can only be placed at one of the available sites and can cover at least one target if it is placed at this site.
- Furthermore, all targets can be covered if all of these available sites are occupied by sensors.
- F denotes a family of subsets of X,
- S ∈ F are elements of F.
- The cardinality of F is the number of available sites to place sensors.
- Every element of F corresponds to the set of targets that can be covered if a sensor is placed at one site. We say that S covers some targets. The set-covering problem is to find a minimum-size subset C ⊆ F whose elements cover all targets of X, that is,

$$X = \bigcup_{S \in \mathcal{C}} S.$$

We say that any C satisfying the above equation covers X or that C is a set-cover of X.
The size of C is defined as the number of sets it contains, rather than the number of individual elements in these sets.

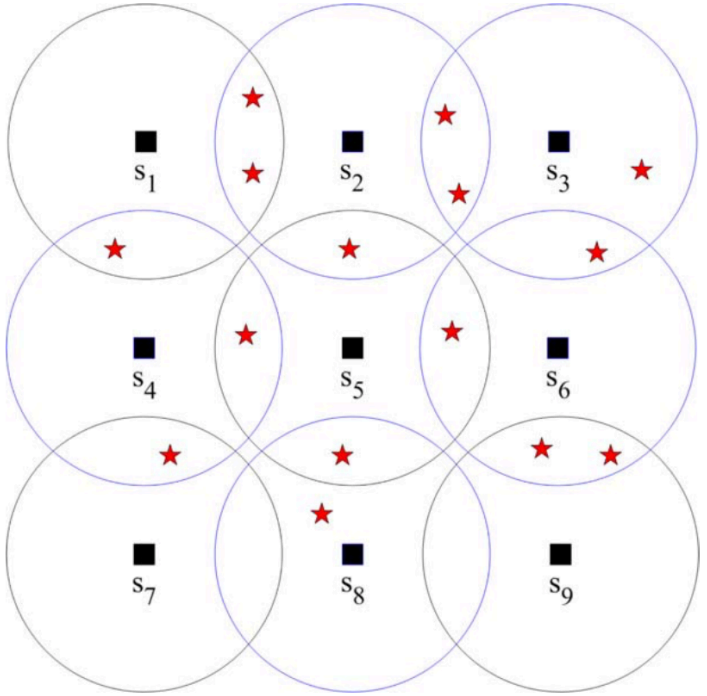


Fig. 4.1 (Color online) An example of the set-covering problem. There are 15 targets (*the red stars*) to be covered. Sensors can put only at nine available sites (*the black squares*), i.e., $s_1, \ldots, s_9$. The greedy algorithm produces a cover of size 5 by selecting the set s_2, s_6, s_4, s_8, and s_3 in sequence

MODELING NODE PLACEMENT

The problem of placing the least number of sensor nodes to cover all targets is an optimization problem.

Many optimization techniques can be used to model and solve the node placement problem.

Suppose there are I sites and J targets. We use subscript i to index a site and j to index a target. Let x_i denote the indicator function of whether a sensor is placed at the site i, i.e.,

$$x_i = \begin{cases} 1 & \text{if a sensor is placed at site } i, \\ 0 & \text{otherwise.} \end{cases}$$

Let δ_{ij} denote an indicator function of whether a target j can be covered by a sensor located at the site i, i.e.,

$$\delta_{ij} = \begin{cases} 1 & \text{if target } j \text{ is covered by a sensor located at site } i, \\ 0 & \text{otherwise.} \end{cases}$$

The problem of placing the least number of sensors to cover all targets can be formulated as the following *integer linear programming* (ILP) problem:

MODELING THE NODE PLACEMENT PROBLEM

The problem of placing the least number of sensors to cover all targets can be formulated as the following *integer linear programming* (ILP) problem:

$$\text{Minimize:} \quad \sum_{i=1}^{I} x_i$$

$$\text{Subject to:} \quad \sum_{i=1}^{I} \delta_{ij} > 0, \quad j = 1, \ldots, J$$

$$x_i \in \{0, 1\}, \quad i = 1, \ldots, I.$$

The constraint is to guarantee that all targets are covered by at least one sensor.

MODELING THE NODE REPLACEMENT PROBLEM

A generalization of the ILP problem is to minimize the network cost if different sensor nodes have different costs and coverage capabilities.

Sometimes, we may have other constraints such as the distance between any pair of sensor nodes should not be less than a certain min distance.

For example, in the scenario of placing different types of sensors, suppose that we have A types of sensors, each type with cost c_a and coverage distance r_a , a = 1, . . . , A.

Normally, a larger coverage range corresponds to a larger cost.

Let D_a(i) denote the set of targets that can be covered by a type a sensor placed at site i, i.e.,

$$D_a(i) = \{j \mid d(i, j) \leq r_a\}, \quad i = 1, \ldots, I,$$

where $d(i, j)$ is the Euclidean distance between a site i and a target j. Again, we use $x_i^a = 1$ to denote that a type a sensor been placed at site i and $x_i^a = 0$ otherwise.

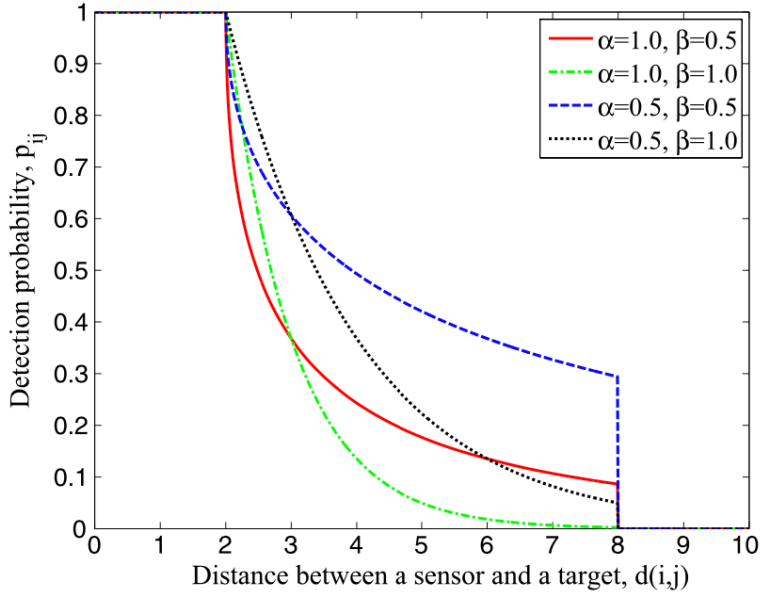
MODELING THE NODE REPLACEMENT PROBLEM

Furthermore, the coverage requirement is to cover each target with at least k sensors. The objective is to minimize the total sensor placement cost

Minimize: $$\sum_{i=1}^{I}\sum_{a=1}^{A} c_a x_i^a,$$

Subject to: $$\sum_{a=1}^{A}\sum_{j\in D_a(i)} x_i^a \geq k, \quad i = 1, \ldots, I$$

$$\sum_{a=1}^{A} x_i^a \leq 1, \quad i = 1, \ldots, I.$$



- The first constraint ensures that each target is covered by at least k sensors, and the second constraint is to ensure that each site can be occupied by at most one sensor.
- It is also possible to allow that some points are not covered by any sensor. This might be the case where we use grid approach to approximate area coverage.
- Hence, some other variants of sensor placement problems include
  - (a) maximizing the number of covered targets subject to a given number of sensors and
  - (b) minimizing the network cost subject to a minimum target coverage ratio. For directional coverage model, another objective is to optimize sensor orientational angles to maximize target coverage..

All these types of sensor placement problems are basically optimization problems and can be formulated as various mathematical programming problems.

APPROXIMATION ALGORITHMS

For small problem instances (for example, the available sites for placing sensors are less than 20), exhaustive search can be used to find the global optimum by trying every possible placement.

If we have I available sites, we can choose I sites each time for placing sensors and examine the cost of the placement.
The number of all possible placements that need to be examined

$$\sum_{i=1}^{I}\binom{I}{i} = 2^I - 1.$$

That is, the computation complexity for exhaustive search increases exponentially with the number of available sites.

Strategy: When the field has a symmetry (e.g., sensors are to be placed at vertices of a 100 × 100 grid), a simple approximation method is to divide the field into several congruent subfields, solve the optimization problem for one subfield, and extend the result to the rest subfields. However, such a division may not be optimal. Therefore, the solution to the whole field may not be globally optimal, even if the solution to a subfield is optimal.

SIMULATED ANNEALING BASED SENSOR PLACEMENT

APPROXIMATION SOLUTIONS: SIMULATED ANNEALING

Simulated annealing is a generic probabilistic heuristic for locating a good approximation to the global extremum of a given function.

- Simulated annealing has been applied to solve various node placement problems. The term simulated annealing derives from a roughly analogous physical process of heating and then slowly cooling a substance to obtain a strong crystalline structure . The basic idea of the simulated annealing method is as follows.
- At first, we start with an initial placement arrangement, with a corresponding placement cost
- We set the annealing initial temperature.
- The process consists of two loops.
- In the outer loop, the temperature is reduced in each step until it reaches a predefined threshold.
  - In the inner loop (at each given temperature), we slightly alter the placement arrangement (e.g., by randomly moving a sensor to a new site) and then compare the new cost value with the old one.
  - If the difference represents an improvement, we then accept this state, and then enter the next temperature.
  - Otherwise we accept this state probabilistically, otherwise we change the system state again.

For the node placement problem, the initial state is to place nodes to all available sites.
In each temperature cooling step, we remove one sensor node, and in each temperature cooling step, the best site to remove a sensor node is found by running the inner loop and using the cost function to determine the difference of the system value.

SIMULATED ANNEALING

- Mimics the physical annealing process.
- First procedure to simulate the annealing behaviour was developed by Metropolis in 1953 to generate sample states of a thermodynamic system.

V. Cerny, Thermodynamical approach to the traveling salesman problem : an

efficient simulation algorithm. J. of Optimization Theory and Applications, 45(1):41–51, 1985

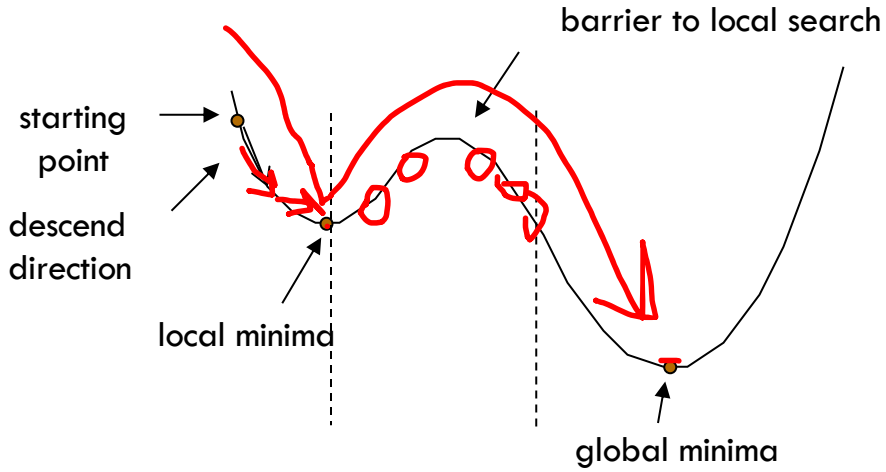
- Defined a set of conditions that, if met, ensure the random walk will sample from probability distribution at equilibrium.
- His algorithm simulates the change in energy of the material when it subjected to cooling process until it reaches a steady “frozen” state.
- He developed a formula for the probability of an increase in energy.

NATURAL VS SIMULATED

Optimization Problem	Physical System
solution x	current state of the solid
cost or objective value $f(x)$	energy of current state
control parameter T	temperature
optimal solution x_{opt}	ground state
simulated annealing	gradual cooling

SIMULATED ANNEALING

- SA is applied to solve optimization problems
- SA is a stochastic algorithm
- SA escapes from local optima by allowing worsening moves
- SA is a memoryless algorithm, the algorithm does not use any information gathered during the search
- SA is applied for both combinatorial and continuous optimization problems
- SA is simple and easy to implement.
- SA is motivated by the physical annealing process



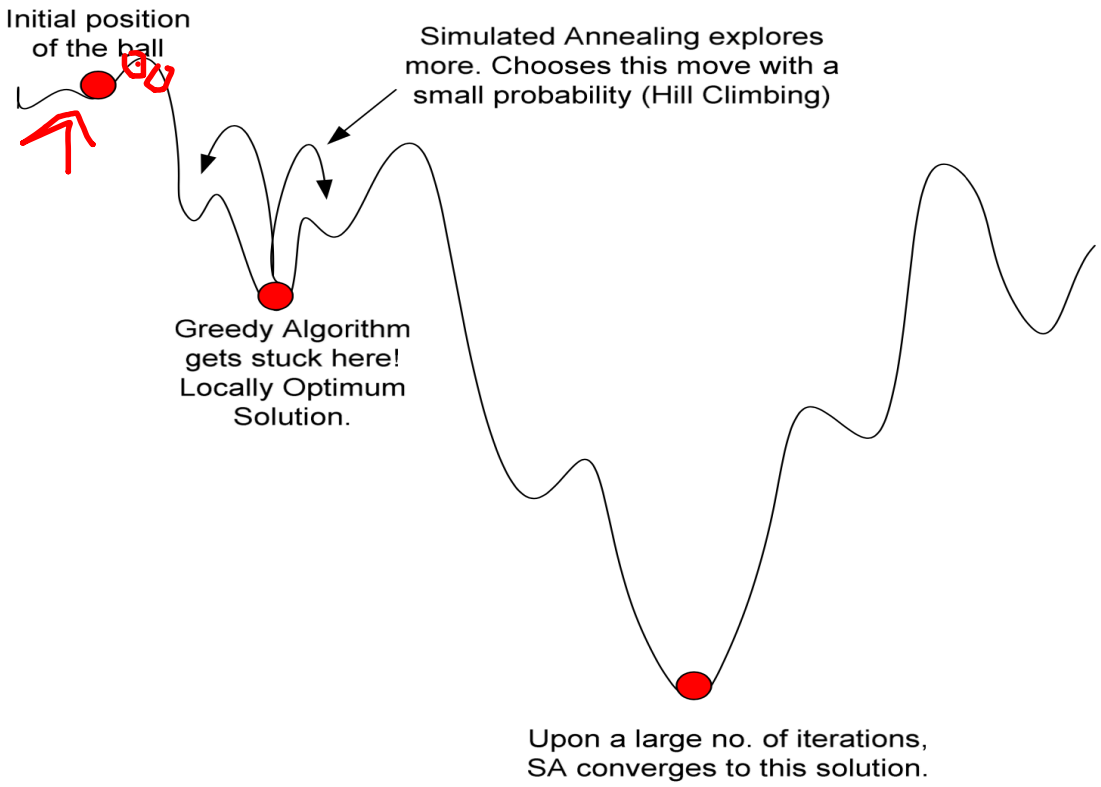
SOME APPLICATIONS

SA has demonstrated capability in problems, such as

- Traveling Salesman Problems
- Graph partitioning
- Matching prob.
- Quadratic Assignment
- Linear Arrangement
- Scheduling
- Machine Learning
- Routing
- Speech Recognition
- etc.

SIMULATED ANNEALING: HOW IT WORKS?

- The approach is part of neighborhood search approaches to move from current solution to new solutions.
- It also tries to escape from local optimum by allowing ***non-improving solutions*** but in a controlled manner.



SIMULATED ANNEALING ALGORITHM STRATEGY

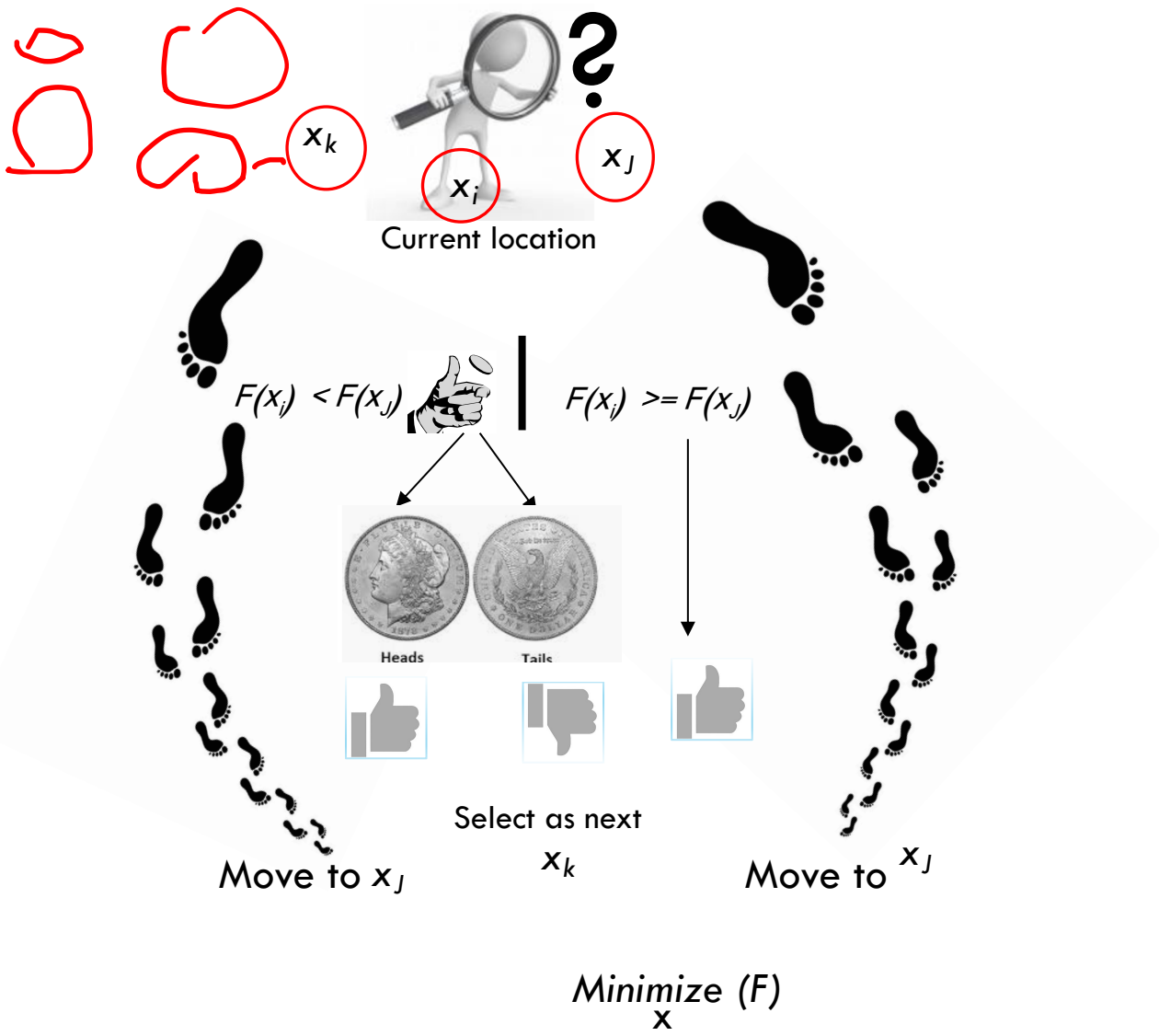
P(H/T) is 50/50

In our case temperature dependent!

Recall, in thermodynamics, a state at a temperature t has a probability of an increase in the energy magnitude ΔE given by:

$$P(\Delta E) = e^{-\Delta E / k \times t}$$

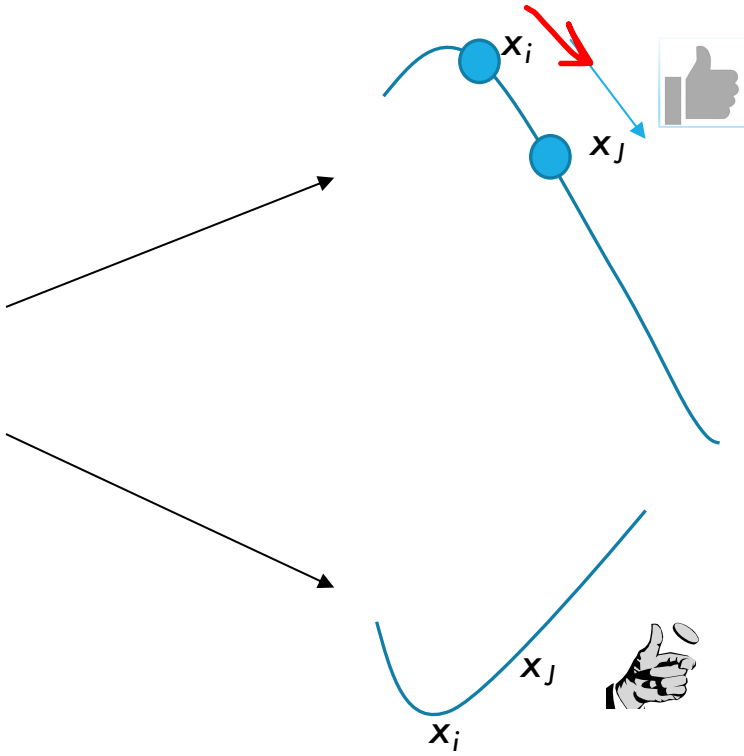
where k is the Boltzmann constant.



SIMULATED ANNEALING

The acceptance probability is defined as:

$$P = \begin{cases} 1 & if \Delta c \leq 0 \\ e^{-\Delta c / t} & if \Delta c > 0 \end{cases}$$



where:

Δc is the change in the solution cost,

t is the current temperature (control parameter, no physical role).

Notice that we dropped Boltzmann's constant

SA BASIC ALGORITHM

Set current solution to an initial solution s= s_0

Set current temperature to an initial temperature $t = t_o$

Select a temperature reduction function α

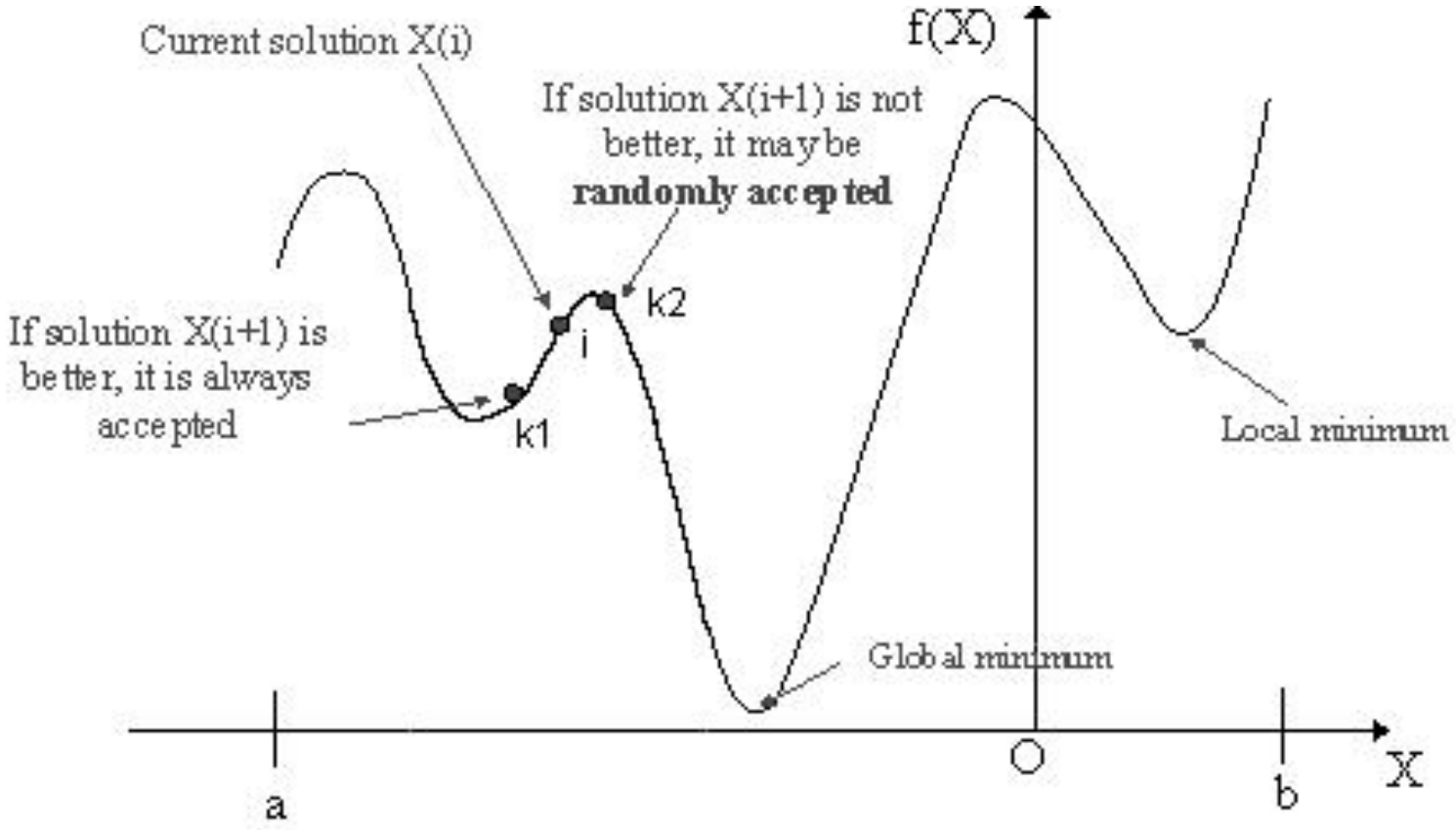
Repeat

- Repeat
  - Select a solution s_i from the neighborhood $N(s)$
  - Calculate change in cost according to the new solution ΔC
  - If ΔC <0 then accept new solution (it is improving) s=s_i
  - Else generate a random number x in the range (0,1)
    - If x < $e^{-\Delta c/t}$ then accept new solution s= s_i
- Until max no. of iterations for the temperature
- Decrease t using α

Until stopping conditions are satisfied

s is the final solution

SIMULATED ANNEALING



SIMULATED ANNEALING

SA **always accepts** better solutions,

SA can accept worse states according to some probability, the ***acceptance probability***,

The acceptance probability is chosen so as to ultimately move to a better solution in the search space,

The acceptance probability is dependent on:

- ✓ The current temperature,
- ✓ The cost difference.

STRATEGY

Higher temperature more attitude to accept bad moves

***i,e,* At high temperatures, explore parameter space**

At lower temperatures, tendency to reject bad moves restrict exploration- exploitation

ACCEPTANCE PROBABILITY OF WORSE SOLUTION

Change	Temperature	Acceptance Probability
0.2	0.1	0.1353
0.4	0.1	0.0183
0.6	0.1	0.0025
0.8	0.1	0.0003

Change	Temperature	Acceptance Probability
0.2	0.9	0.8007
0.4	0.9	0.6412
0.6	0.9	0.5134
0.8	0.9	0.4111

- As the temperature decreases, the probability of accepting a worse state decreases,
- At zero temperature, no worse states are accepted,

ANNEALING SCHEDULE

The value of the temperature is adjusted according to the ***annealing schedule***,

The annealing schedule is a mapping from time to temperature,

The annealing schedule is determined by:

- The initial temperature,
- The final temperature,
- The temperature decrement rule,
- Number of iterations at each temperature.

TEMPERATURE

The initial temperature:

- Must be **high enough** to allow a move to possibly any state in the search space,
- Must not be too hot [SA would behave as random search for a number of iterations],
- The maximum **change of the cost function** could be used as **a guide** to set this value,
- A good initial temperature is one that accepts about **60%** of the worse moves.

TEMPERATURE

The final temperature:

- Doesn't have to reach zero,
- Might take a very long time to reach zero when some decrement approaches are used,
- To stop the algorithm:
  - A reasonably low temperature,
  - The system is frozen, no better moves are generated neither any worse moves are getting accepted.

TEMPERATURE

Temperature decrement rules

- Linear

$$t = t - \alpha$$

- Geometric

$$t = t \times \alpha$$

- Slow decrease: [for example]

  - decreases the temperature very slowly.

$$t = t/(1+\beta t), \beta$$

TEMPERATURE-ITERATIONS

The number of iterations at each temperature:

- Enough iterations should be allowed at every temperature for the system to be stable at that temperature,
- The required number of iterations might be large for large size problems
- Usually, a constant value is used.
- In slow decrease rules, use one iteration per temperature
- We could dynamically change this value:
  - Allowing a small number of iterations at high temperatures,
  - Allowing a large number of iterations at low temperature to fully explore the local optimum.

CONVERGENCE OF SA

The behavior of SA Algorithm can be modeled using **Markov chains**.

- It has been shown that under **constant temperature** and as long as it is possible to find a sequence of exchanges that will transform any solution to another with **non−zero−probability**, the process **converges** independent of the starting solution.

In SA the temperature is **not constant**.

- The process is still a **Markov chain but non−homogeneous**. It converges if the temperature is reduced to zero slowly (with a specific logarithmic form) and the number of iterations at each temperature is large (exponential in terms of the problem size).

The convergence theory dictates a certain form of cooling schedule that is not practical to use.

- However, the results give SA a theoretical backing not always possible with heuristic methods.

SIMULATED ANNEALING

Advantages:

- Ease of use,
- Providing good solutions for a wide range of problems.

Disadvantages:

- A lot of computer time for many runs,
- A lot of tuneable parameters.

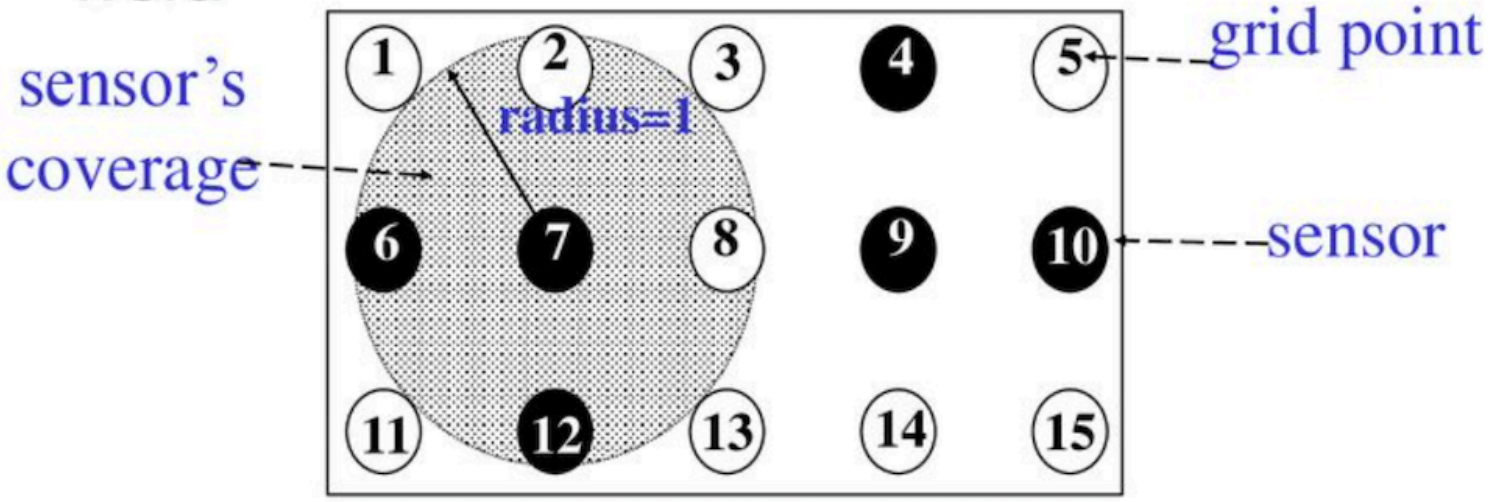
SIMULATED ANNEALING

Applied successfully to many applications:

- Non−linear function optimization,
- Travelling Salesman Problem (TSP),
- Scheduling Problems,
- FPGA placement,
- Task allocation,
- Network design,
- Graph colouring and partitioning,
- Many more …

Sensor Placement for Target Location

- A complete covered and discriminated sensor field



Distance Error

- Due to some resource limitations, a complete discriminated sensor field cannot be constructed.
- Positioning accuracy, therefore, becomes a major issue of the problem.
- Distance Error
  - The distance error of two indistinguishable grid points is defined as the Euclidean distance between them.

Objectives

- Sensor Placement For Target Location
  - Complete Discrimination
    - It is implied that the minimum Hamming distance of power vectors for any pair of grid points will be maximized.
  - High Discrimination
    - When complete discrimination is not possible.
    - It is implied that the maximum distance error will be minimized.

GIVEN PARAMETERS

$A:\{1,2,\ldots,m\}$:Index set of the location in the sensor field.

$B:\{i \mid i \in A\}$:Index set of the sensor's candidate locations.

r_k :Detection radius of sensor located at k, $k \in B$

d_{ij} :Euclidean distance between location i and j, $i, j \in A$

c_k :The cost of sensor allocated at location k, $k \in B$

G :Total cost limitation

DECISION VARIABLES

y_k : 1, if an sensor is allocated at location k, $k \in B$.

$v_i = (v_{i1}, v_{i2}, \ldots, v_{ik})$: A power vector of location i, where v_{ik} is 1 if the target at location i can be detected by the sensor at location k and 0 otherwise.

OBJECTIVE FUNCTION

$$Z_{IP1} = \min_{v} \max_{(i,j)} \frac{d_{ij}}{1 + K \sum_{\forall k \in B} (v_{ik} - v_{jk})^2}$$

(K is a big number.)

CONSTRAINTS

$$v_{ik} d_{ik} \leq y_k r_k \qquad \forall i \in A, k \in B, i \neq k \qquad (1)$$

$$\frac{d_{ik}}{r_k} > y_k - v_{ik} \qquad \forall i \in A, k \in B, i \neq k \qquad (2)$$

$$v_{ik} = y_k \qquad \forall i \in A, k \in B, i = k \qquad (3)$$

$$\sum_{\forall k \in B} c_k y_k \leq G \qquad (4)$$

$$\sum_{\forall k \in B} v_{ik} \geq 1 \qquad \forall i \in A \qquad (5)$$

CONSTRAINTS

y_k	v_{ik}	Conditions	Need	Constraint (1)	Constraint (2)	Notes	Physical meanings
0	0	Don't care	Y	Y	Y		No sensor, no power vector.
0	1	Don't care	N	N	Y	$d_{ik} \le 0 \cdots\cdots (1)$ $\frac{d_{ik}}{r_k} > -1 \cdots\cdots (2)$	No sensor, but a corresponding power vector presents.
1	0	$d_{ik} > r_k$	Y	Y	Y	$r_k \ge 0 \cdots\cdots (1)$ $\frac{d_{ik}}{r_k} > 1 \cdots\cdots (2)$	Out of the detection range.
1	0	$d_{ik} \le r_k$	N	Y	N		
1	1	$d_{ik} > r_k$	N	N	Y	$d_{ik} \le r_k \cdots\cdots (1)$ $\frac{d_{ik}}{r_k} > 0 \cdots\cdots (2)$	Within the detection range.
1	1	$d_{ik} \le r_k$	Y	Y	Y		

Notes: $v_{ik} d_{ik} \le y_k r_k \cdots\cdots (1)$ $\frac{d_{ik}}{r_k} > y_k - v_{ik} \cdots\cdots (2)$

THE ENERGY FUNCTION

The energy, ***E***, is defined as follows:

$$E = \max_{(i,j)} \frac{d_{ij}}{1 + K \sum_{\forall k \in B} (v_{ik} - v_{jk})^2}$$

ALGORITHM

1. Deploy sensors on all grid points // **Initial guess**
2. repeat until $t<t_f$
3. repeat **r** times
4. If the cost is still higher than the cost limitation then
5. ***Discard*** a sensor randomly
6. Decide the new configuration can be accepted or not
7. If accepted then goto step 10
8. ***Move*** a sensor to a new grid point randomly
9. Decide the new configuration can be accepted or not
10. If the best deployment is the desired solution then stop
11. **t=t*0.75, r=r*1.3** // **Cooling function**

DECISION FOR NEW DEPLOYMENT

a. Generate a random number $p, 0 < p < 1$

b. If $\sum_{\forall k \in B} v_{ik} \geq 1, \forall i \in A,$ then // Complete covered

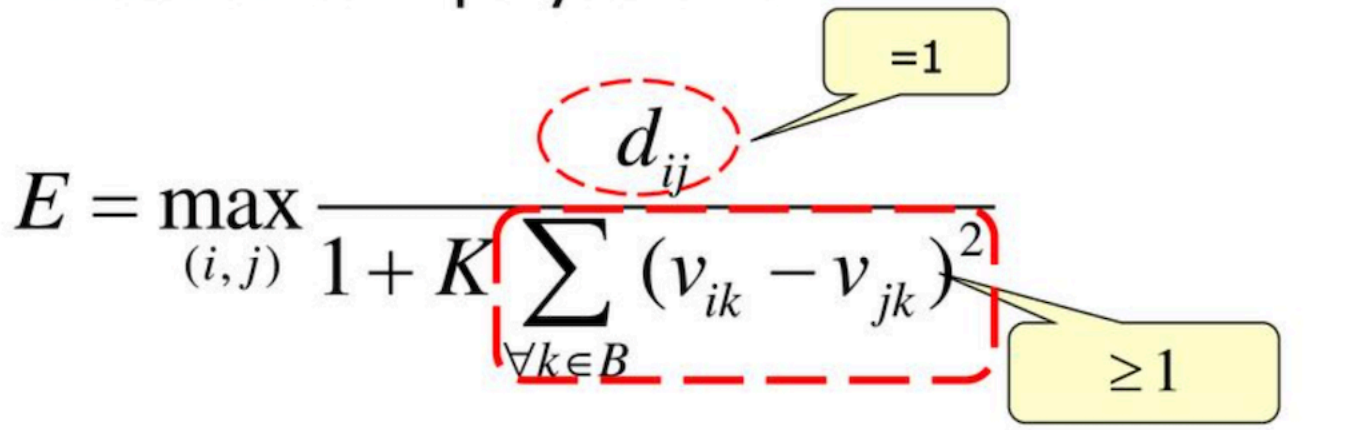
Evaluate the energy differential ΔE between the two deployments

If $\exp(-\Delta E/t) > p$ then

Accept the new deployment and save the best one

ENERGY FOR THE DESIRED SOLUTION

The energy for a complete coverage and discrimination deployment is



$$\Rightarrow \quad E \leq 1/(1+K)$$

PARAMETERS

Initial $r_0=5n$, $t_0=0.1$

$\alpha=0.75$, $t_{n+1}=\alpha\, t_n$

$\beta=1.3$, $r(t_{n+1})=\beta\, r(t_n)$

$t_f=1/300$

APPROXIMATION SOLUTIONS: GENETIC ALGORITHM

Genetic algorithms are a class of probabilistic optimization algorithms to search or to approximate solutions for optimization problems.

Inspired by biological evolutionary processes, genetic algorithms model and apply biological inheritance, mutation, selection, and crossover (recombination) in the search of global optimal solution.

Genetic algorithm has been widely used to solve the combinatorial and nonlinear optimization problems with complex constraints and has also been applied to find solutions to various sensor placement problems.

The computation of genetic algorithm is an iterative process towards achieving the global optimality.

During the iterations, candidate solutions are retained and ranked according to their quality. A fitness value is used to screen out unqualified solutions.

Genetic operations of crossover, mutation, translocation, inversion, addition, and deletion are then performed on those qualified solutions to create new candidate solutions of the next generation.

The above process is carried out repeatedly until certain stopping or convergence condition is met For simplicity, a maximum number of iterations can be chosen to be the stopping condition.

The variation difference of the fitness values between two adjacent generations may also serve as a good indication for convergence.

NEXT lecture