Evolutionary Algorithms

Piero P. Bonissone

GE Corporate Research & Development

Bonissone@crd.ge.com

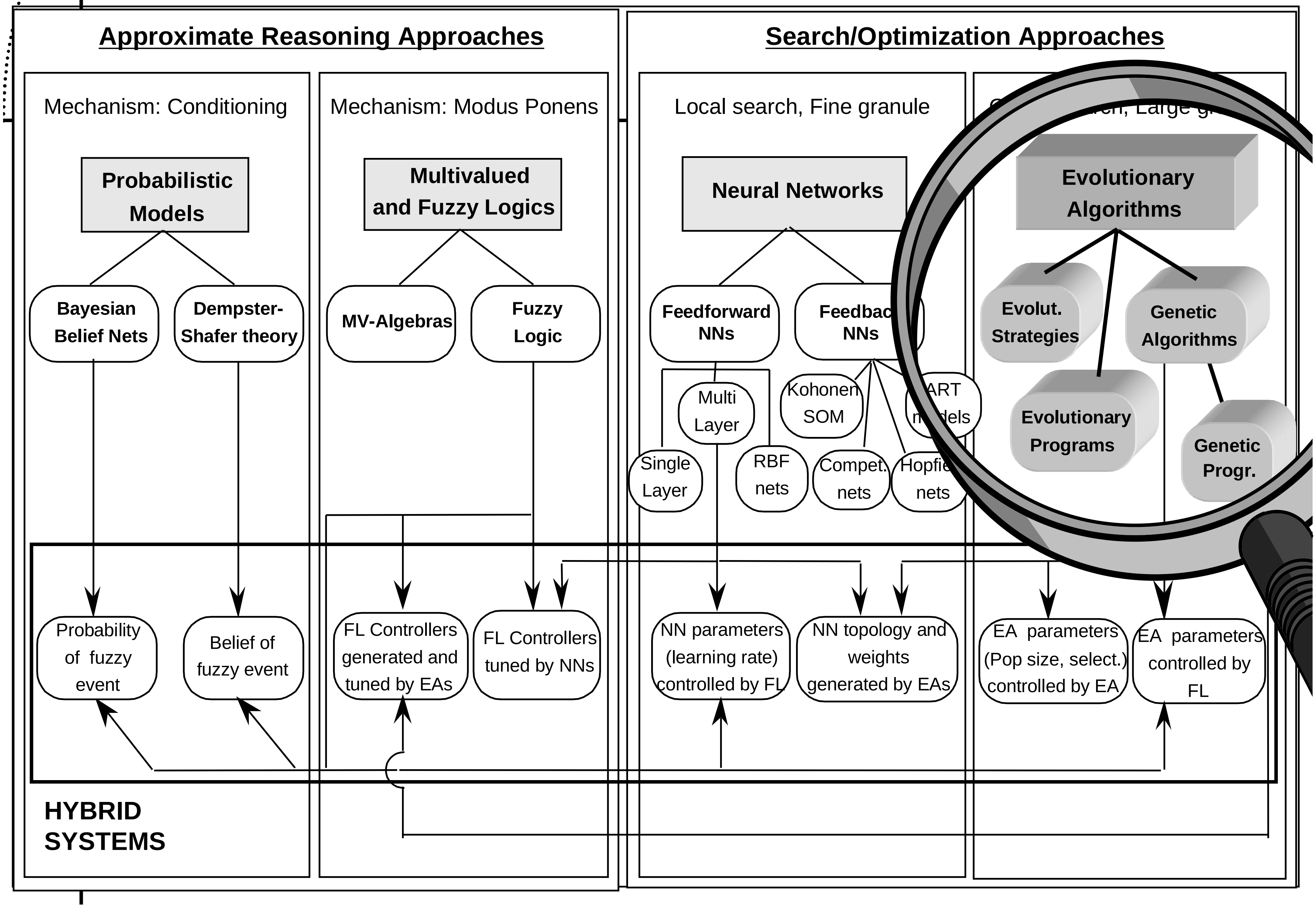
EA Review - Outline

- Soft Computing
  - Definition
  - SC Components & Hybrid Systems
- Evolutionary Algorithms
  - Derivative Free
  - Components (ES, EP, GA, GP)
- Genetic Algorithms
  - General Characteristics
  - Representation
  - Evaluation & Constraints
  - Operators
  - Components Summary
  - Process Cycle
  - Functional Optimization Example
  - Evolution Stages

SC Definition and EA

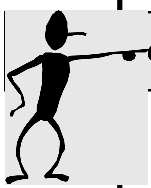
- **Soft Computing (SC): the symbiotic use of many emerging problem-solving disciplines.**
- **According to Prof. Zadeh:**
  - *"...in contrast to traditional hard computing, soft computing exploits the tolerance for imprecision, uncertainty, and partial truth to achieve tractability, robustness, low solution-cost, and better rapport with reality"*
- **Evolutionary Algorithms is one of Soft Computing four main components:**
  - ***Approximate Reasoning:***
    - **Probabilistic Reasoning, Fuzzy Logic**
  - ***Search:***
    - **Neural Networks, Evolutionary Algorithms**

[redacted] **Evolutionary Algorithms** **Hybrid SC Systems**



EA Review - Outline

- Soft Computing
  - Definition
  - SC Components & Hybrid Systems



- Evolutionary Algorithms
  - Derivative Free
  - Components (ES, EP, GA, GP)
- Genetic Algorithms
  - General Characteristics
  - Representation
  - Evaluation & Constraints
  - Operators
  - Components Summary
  - Process Cycle
  - Functional Optimization Example
  - Evolution Stages

Evolutionary Algorithms (EA)

- **EA are part of the Derivative-Free Optimization and Search Methods:**
  - Evolutionary Algorithms
  - Simulated annealing (SA)
  - Random search
  - Downhill simplex search
  - Tabu search

- **EA consists of**
  - **Evolutionary Strategies (ES)**
  - **Evolutionary Programming (EP)**
  - **Genetic Algorithms (GA)**
  - **Genetic Programming (GP)**

Evolutionary Algorithms (EA) Characteristics

- Most Evolutionary Algorithms can be described by

$$\boxed{x[t+1] = s(v(x[t]))}$$

  - $x[t]$ **: the population at time** t **under representation** x
  - v **: is the variation operator(s)**
  - s **: is the selection operator**

- EA exhibit an *adaptive behavior* that allows them to handle non-linear, high dimensional problems without requiring differentiability or explicit knowledge of the problem structure.

- EA are very robust to time-varying behavior, even though they may exhibit low speed of convergence.

Evolutionary Algorithms: ES

Evolutionary Strategies (ES)

- **Originally proposed for the optimization of continuous functions**
- **(μ , λ)-ES and (μ + λ)-ES**
  - A population of μ parents generate λ offspring
  - Best μ offspring are selected in the next generation
  - (μ , λ)-ES: parents are **excluded** from selection
  - (μ + λ)-ES: parents are **included** in selection
- **Started as (1+1)-ES** *(Reschenberg)* **and evolved to (μ + λ)-ES** *(Schwefel)*
- **Started with Mutation only (with individual mutation operator) and later added a recombination operator**
- **Focus on behavior of individuals**

Evolutionary Algorithms: EP

Evolutionary Programming (EP)

- **Originally proposed for sequence predictiom and optimal gaming strategies**
- **Currently focused on continuous parameter optimization and training of NNs**
- **Could be considered a special case of $(\mu + \mu)$-ES without recombination operator**
- **Focus on behavior of species (hence no crossover)**
- **Proposed by** *Larry Fogel (1963)*

Evolutionary Algorithms: GA

Genetic Algorithms (GA)

- **Perform a randomized search in solution space using a <u>genotypic</u> rather than a <u>phenotypic</u>**
- **Each solution is encoded as a chromosome in a population (a binary, integer, or real-valued string)**
  - Each string's element represents a particular feature of the solution
- **The string is evaluated by a fitness function to determine the solution's quality**
  - Better-fit solutions survive and produce offspring
  - Less-fit solutions are culled from the population
- Strings are evolved using mutation & recombination operators.
- New individuals created by these operators form next generation of solutions
- Started by *Holland (1962; 1975)*

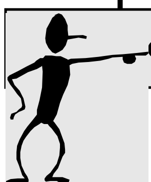
Evolutionary Algorithms: GP

Genetic Programming (GP)

- **A special case of Genetic Algorithms**
  - Chromosomes have a **hierarchical** rather than a **linear** structure
  - Their sizes are not predefined
  - Individuals are tree-structured programs
  - Modified operators are applied to sub-trees or single nodes
- Proposed by *Koza (1992)*

EA Review - Outline

- Soft Computing
  - Definition
  - SC Components & Hybrid Systems
- Evolutionary Algorithms
  - Derivative Free
  - Components (ES, EP, GA, GP)



- Genetic Algorithms
  - General Characteristics
  - Representation
  - Evaluation & Constraints
  - Operators
  - Components Summary
  - Process Cycle
  - Functional Optimization Example
  - Evolution Stages

GAs General Characteristics

- No gradient information required
- No restrictions on structure of evaluation function (could be a black box)
- Resulting search is global
- Local Optima are avoided by hyperplane sampling in Hamming space (crossovers) plus random perturbations (mutations)
- Potential for massive parallelism
- They can be hybridized with conventional optimization methods

Genetic Algorithms: Description & Representation

- **What are They?**

GAs are a new programming paradigm used to solve NP-hard problems by performing a randomized search in the solution space.

- **Encoding:**

GAs *encode* the solution to a given problem in a binary (or real-valued) string. Each string's element represents a particular feature in the solution.

Genetic Algorithms: Evaluation & Constraints

- **Evaluation:**

The string (solution) is *evaluated* by a fitness function to determine the solution's quality: good solutions survive and have off-springs, while bad solutions are discontinued.

- **Constraints:**

Solution's constraints can be modeled by *penalties* in the fitness function or encoded directly in the solution *data structures*.

Genetic Algorithms: Operators

- **Operators:**

To improve current solutions, the string is modified by two basic type of operators: Cross-over and Mutations.

  - **Cross-over are (sometime) deterministic operators that capture the best features of two parents and pass it to a new off-spring string.**
  - **Mutations are probabilistic operators that try to introduce needed solutions features in populations of solutions that lack such feature.**

Genetic Algorithms: Components Summary

1) Encoding Technique

- (Chromosome Structure)

2) Evaluation or Fitness Function

- (The Environment)

3) Initialization Procedure

- (Creation)

4) Genetic Operators

- (Mutation, Recombination or Crossover)

FOCUS

5) Parameter Setting

- (practice and art)

Genetic Algorithms: Process Cycle

Initialization:

- generate a random population of individuals

Solution Evaluation:

- Evaluate and score all individuals in population by using the fitness function

Reproduction:

- Select parents and generate offsprings using cross-over operator
- Recombine population according to probability of crossover
- Apply mutation according to probability of mutation

Genetic Algorithms: Example of Binary Encoding, Crossover, Mutation

Binary encoding

$$(11, 6, 9) \Rightarrow \overbrace{1011 \;\; 0110 \;\; \underbrace{1001}_{\text{Gene}}}^{\text{Chromosome}}$$

Crossover

$$\begin{array}{ccc|ccccc} 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{array} \Rightarrow \begin{array}{ccc|ccccc} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \end{array}$$

Crossover point

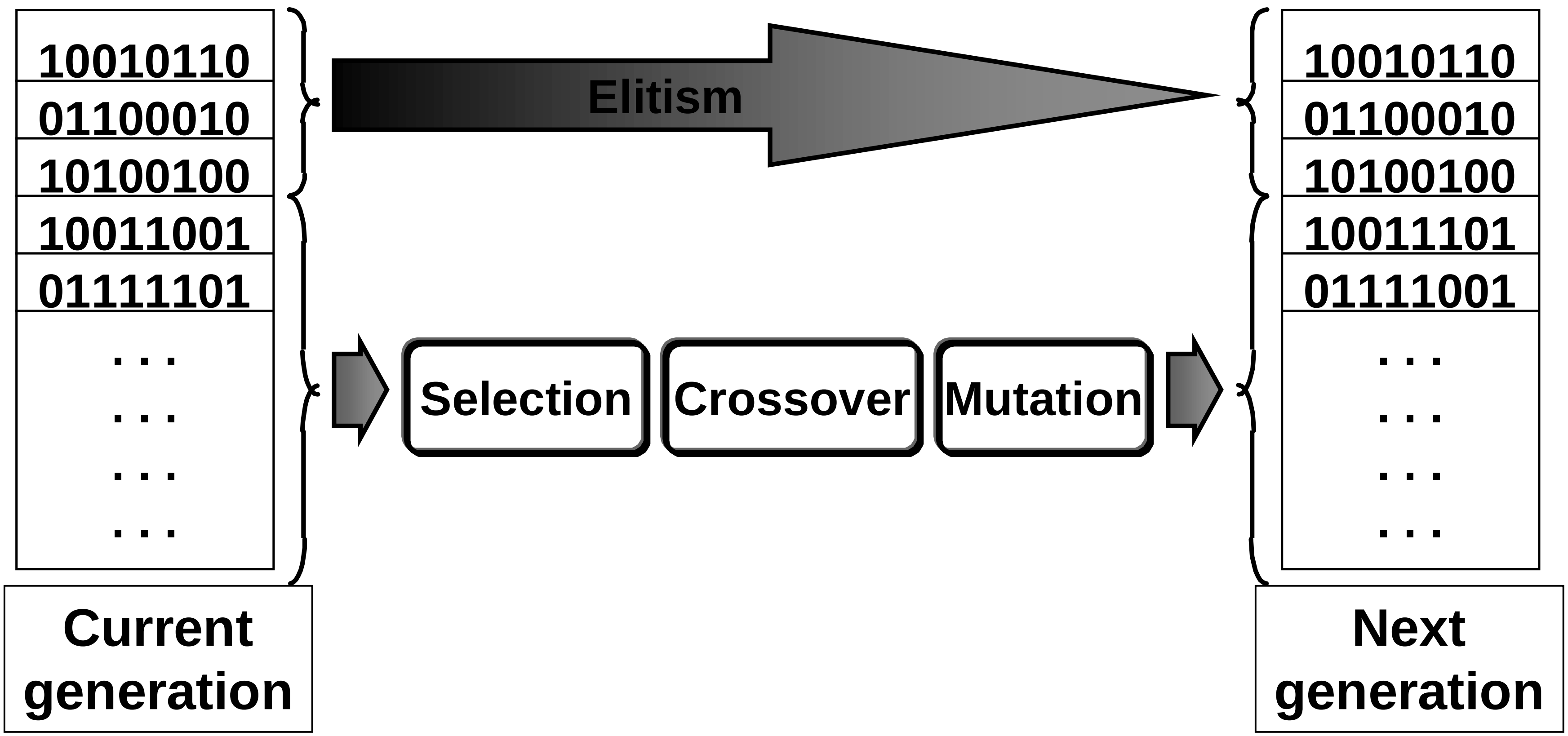
Mutation

$$1\ 0\ 0\ 1\ 1\ 1\ 1\ 0 \Rightarrow 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0$$

Mutation bit

Genetic Algorithms: Example of Process

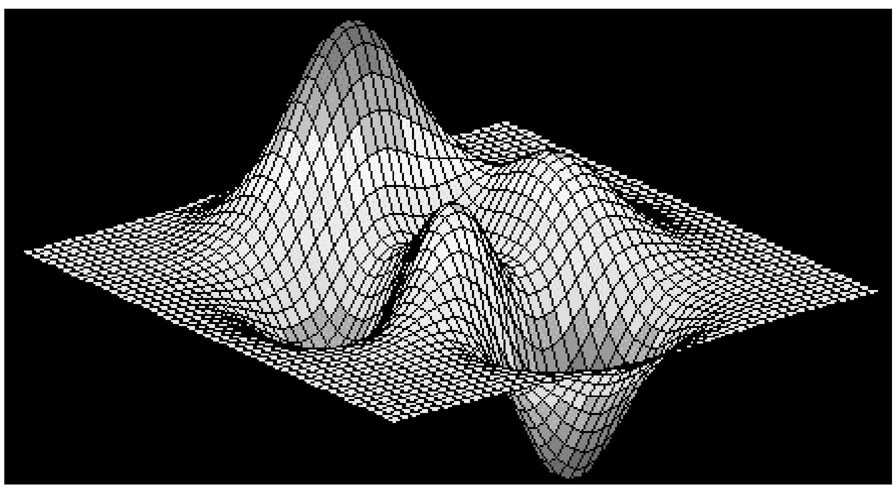
Flowchart

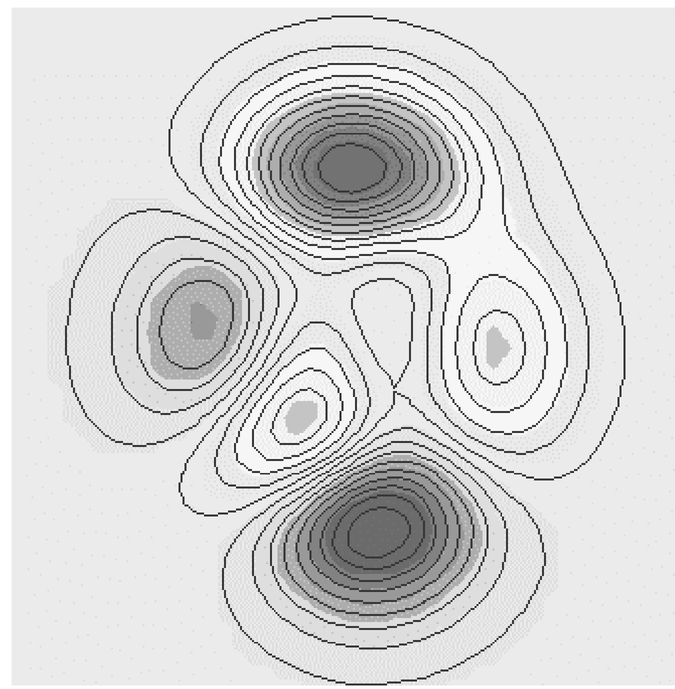


Genetic Algorithms: Functional Optimization Example

Example: Find the max. of the “peaks” function

z = f(x, y) = 3*(1-x)^2*exp(-(x^2) - (y+1)^2) - 10*(x/5 - x^3 - y^5)*exp(-x^2-y^2) -1/3*exp(-(x+1)^2 - y^2).





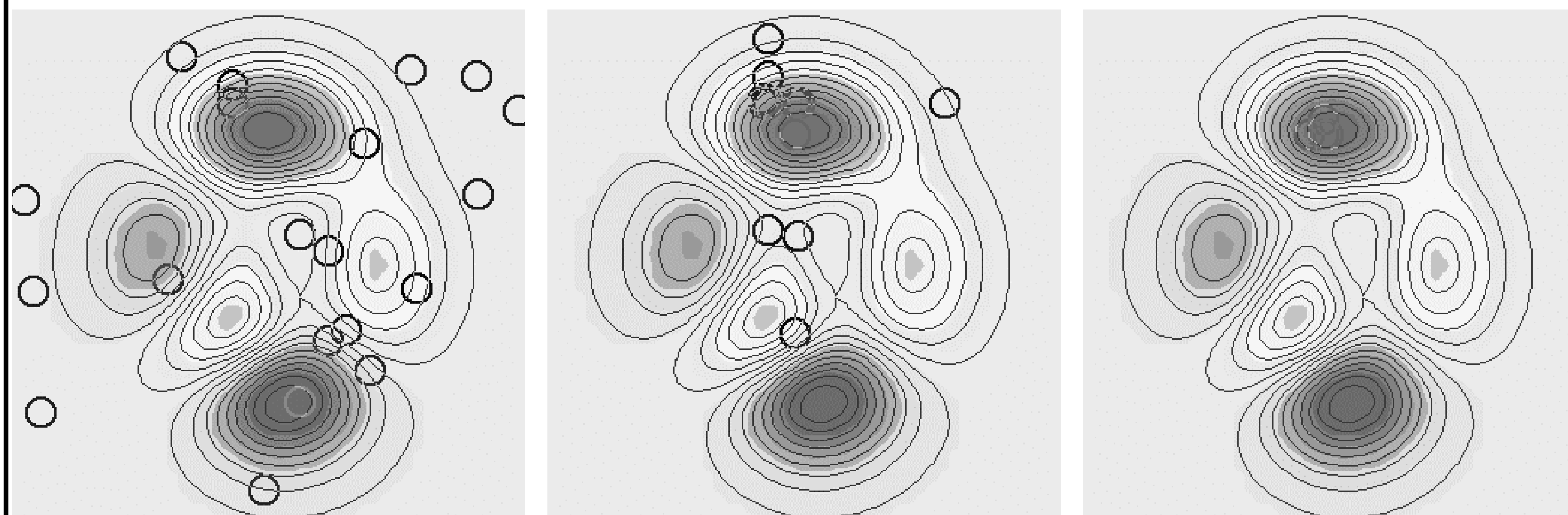
Genetic Algorithms: Functional Optimization Example

Derivatives of the “peaks” function

- **dz/dx = -6*(1-x)*exp(-x^2-(y+1)^2) - 6*(1-x)^2*x*exp(-x^2-(y+1)^2) - 10*(1/5-3*x^2)*exp(-x^2-y^2) + 20*(1/5*x-x^3-y^5)*x*exp(-x^2-y^2) - 1/3*(-2*x-2)*exp(-(x+1)^2-y^2)**
- **dz/dy = 3*(1-x)^2*(-2*y-2)*exp(-x^2-(y+1)^2) + 50*y^4*exp(-x^2-y^2) + 20*(1/5*x-x^3-y^5)*y*exp(-x^2-y^2) + 2/3*y*exp(-(x+1)^2-y^2)**
- **d(dz/dx)/dx = 36*x*exp(-x^2-(y+1)^2) - 18*x^2*exp(-x^2-(y+1)^2) - 24*x^3*exp(-x^2-(y+1)^2) + 12*x^4*exp(-x^2-(y+1)^2) + 72*x*exp(-x^2-y^2) - 148*x^3*exp(-x^2-y^2) - 20*y^5*exp(-x^2-y^2) + 40*x^5*exp(-x^2-y^2) + 40*x^2*exp(-x^2-y^2)*y^5 - 2/3*exp(-(x+1)^2-y^2) - 4/3*exp(-(x+1)^2-y^2)*x^2 -8/3*exp(-(x+1)^2-y^2)*x**
- **d(dz/dy)/dy = -6*(1-x)^2*exp(-x^2-(y+1)^2) + 3*(1-x)^2*(-2*y-2)^2*exp(-x^2-(y+1)^2) + 200*y^3*exp(-x^2-y^2)-200*y^5*exp(-x^2-y^2) + 20*(1/5*x-x^3-y^5)*exp(-x^2-y^2) - 40*(1/5*x-x^3-y^5)*y^2*exp(-x^2-y^2) + 2/3*exp(-(x+1)^2-y^2)-4/3*y^2*exp(-(x+1)^2-y^2)**

Genetic Algorithms: Functional Optimization Example

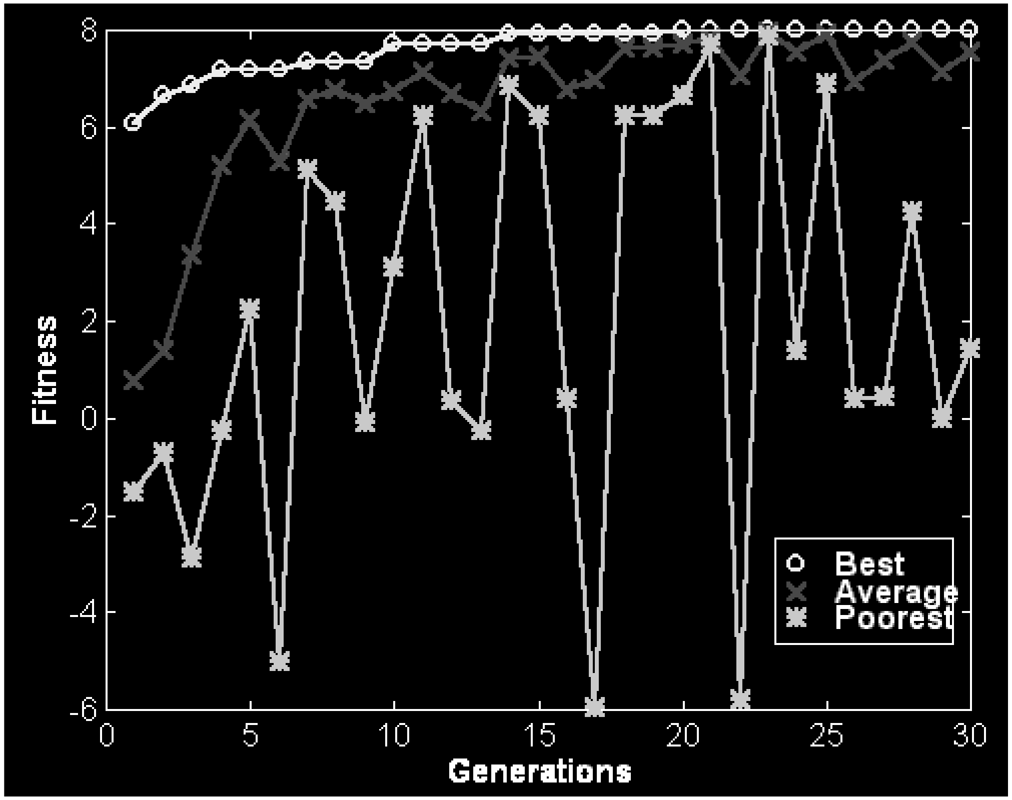
GA process:



Initial population | 5th generation | 10th generation

GAs Performance Measurements

Performance profile



GAs Evolution Stages

- ***<u>Exploration</u>*** **: GAs generate enough candidates to have enough diversity (building blocks) for good solution**
  - Increased Population Size
  - More forgiving parents selection
  - Stronger mutation
- ***<u>Transition</u>*** **: GAs refine their performance by performing a down-selection & managing their resources during exploitation**
- ***<u>Exploitation</u>*** **: GAs refine their solutions by focusing on good individuals and enforcing more stringent solution quality requirements**
  - Decreased Population Size
  - Stricter parents selection
  - Weaker mutation

Appendix

Derivative-Free Optimization

• Other Derivative-Free Optimization & Search Methods besides EA:

- Simulated annealing (SA)
- Random search
- Downhill simplex search
- Tabu search

• We will explore Simulated Annealing (SA)

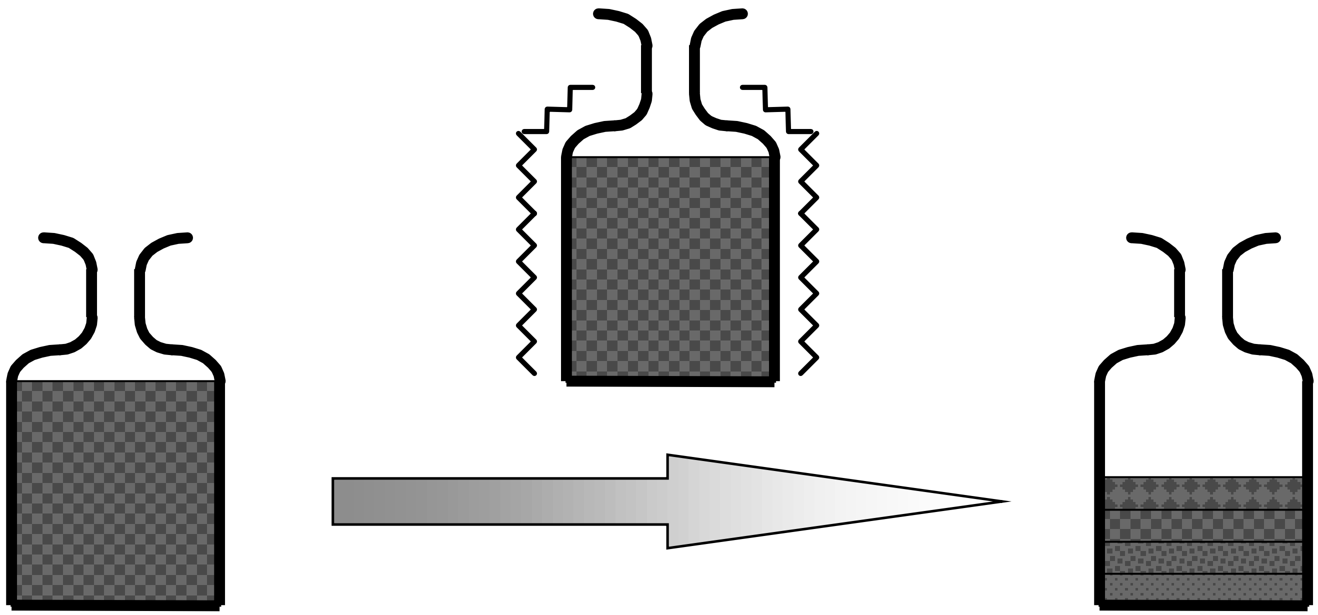
Simulated Annealing: SA

Simulated Annealing (SA)

- Stochastic Hill-climbing algorithm based on the analogy with the physical process of annealing:
  - a lattice structure of a solid is achieved by heating up the solid to its melting point
  - and then slowly cooling until it solidifies to a low-energy state
- SA works as follows:
  - Randomly picks feasible solutions,
  - Improving on a solution by always accepting better-cost neighbors if they are selected
  - Allowing for a stochastically guided acceptance of worse-cost neighbors
  - Gradually decreasing the probability of accepting worse-cost neighbors (*gradual cooling*)

Simulated Annealing

Analogy



Simulated Annealing: SA

Simulated Annealing (SA)

- SA can be seen as a single-individual EA in which crossovers are disabled and only mutations are used
- This is also a global search strategy and can work in very high-dimensional searches given enough computational resources.
- Proposed by *Kirkpatrick* (*1983*)

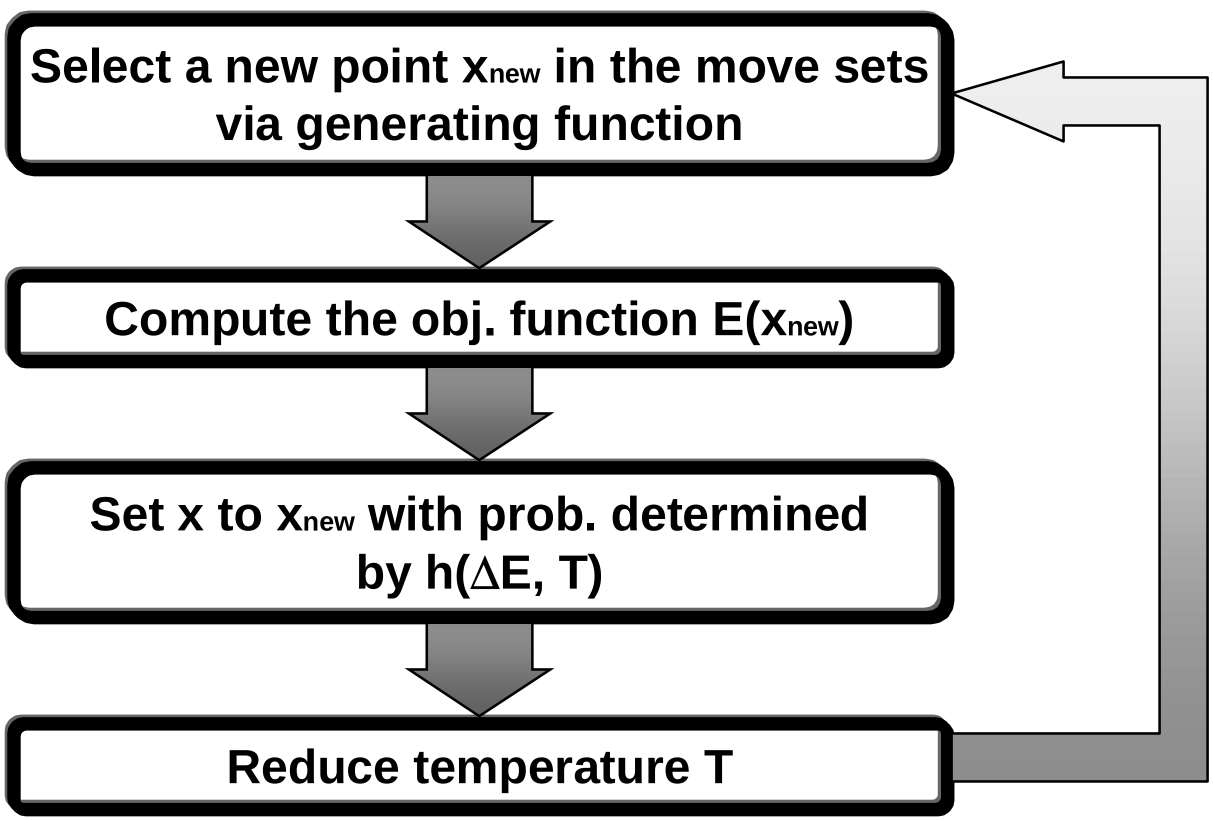
Simulated Annealing

Terminology:

- **<u>Objective function E(x)</u>:**
  - function to be optimized
- **<u>Move set</u>:**
  - set of next points to explore
- **<u>Generating function</u>:**
  - to select next point
- **<u>Acceptance function h(ΔE, T)</u>:**
  - to determine if the selected point should be accept or not. Usually $h(\Delta E, T) = 1/(1+\exp(\Delta E/(cT))$.
- **<u>Annealing (cooling) schedule</u>:**
  - schedule for reducing the temperature T

Simulated Annealing

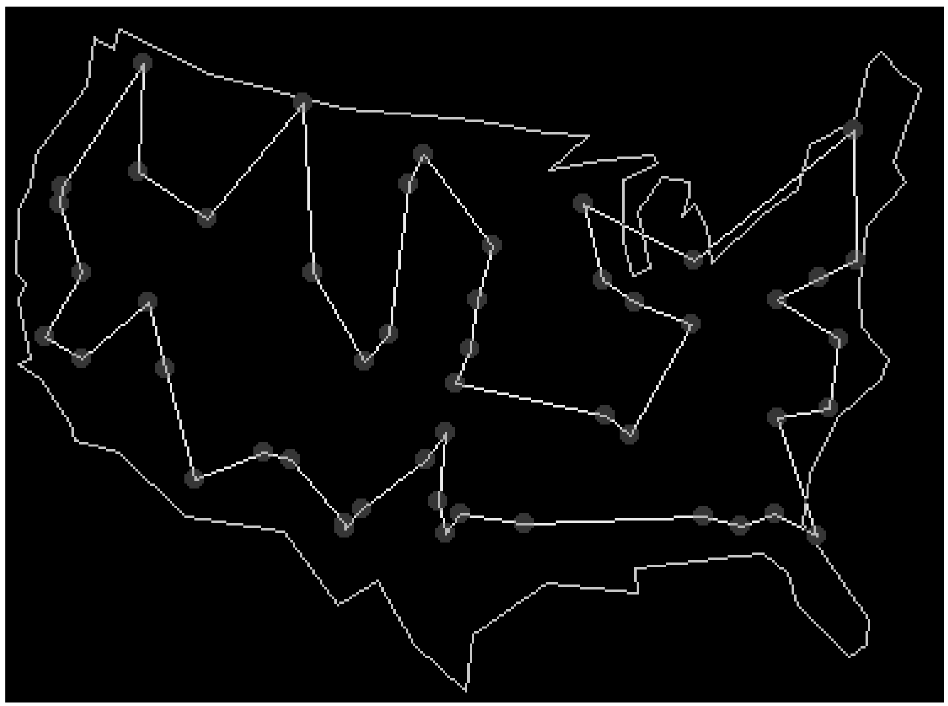
Flowchart



Simulated Annealing

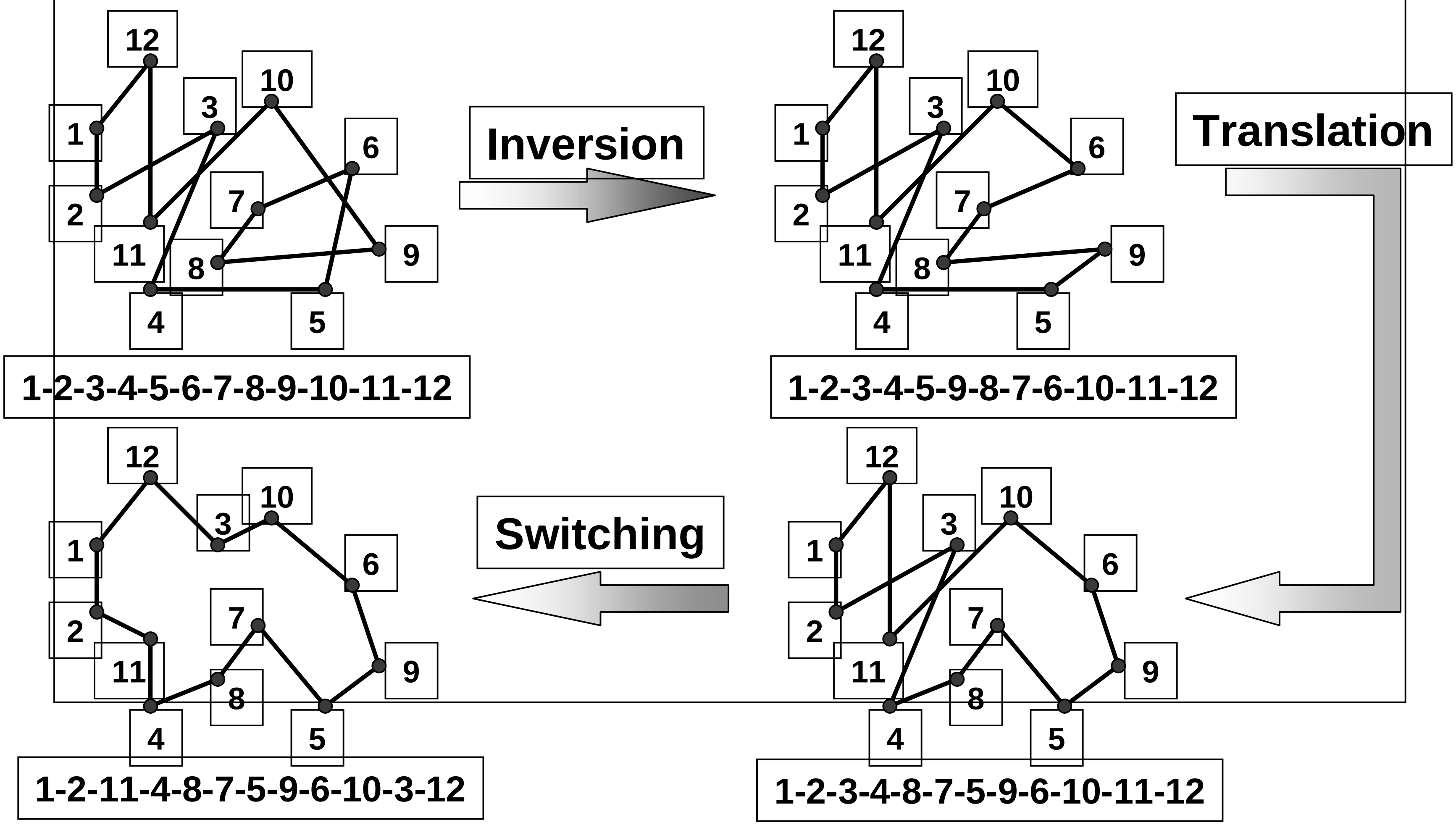
Example: Travel Salesperson Problem (TSP)

How to transverse n cities once and only once with a minimal total distance?



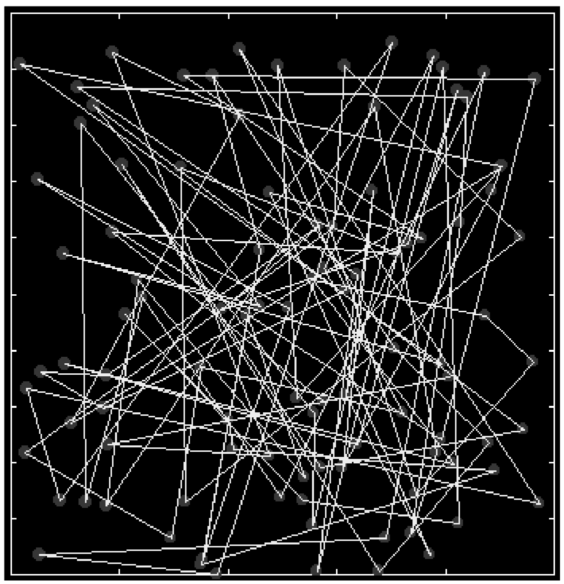
Simulated Annealing

Move sets for TSP

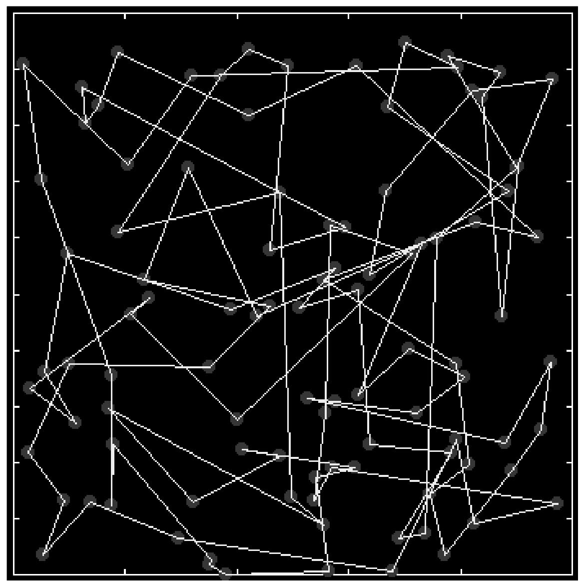


Simulated Annealing

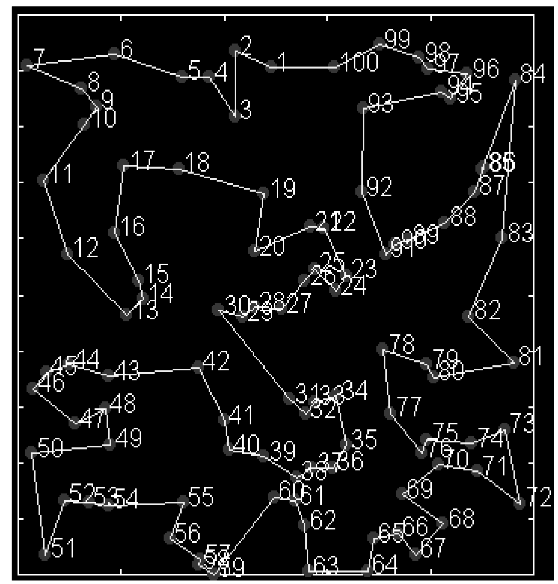
A 100-city TSP using SA



Initial random path



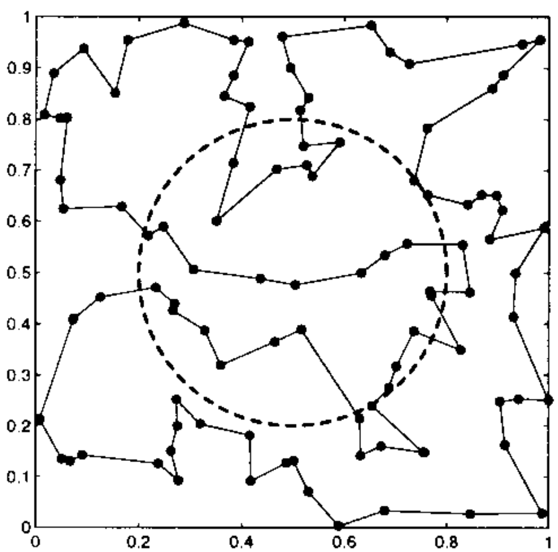
During SA process



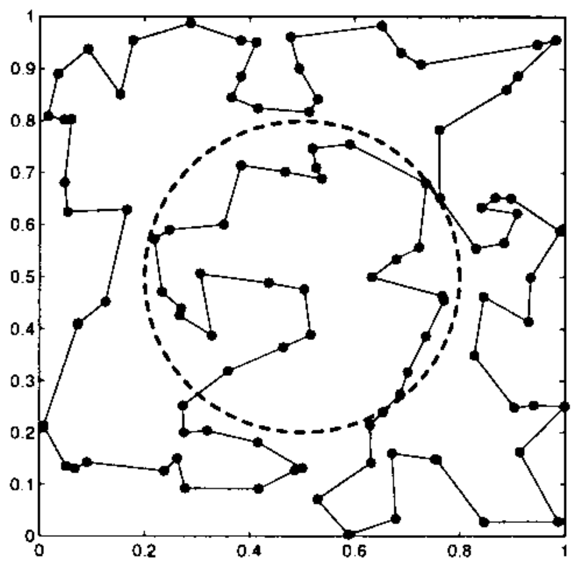
Final path

Simulated Annealing

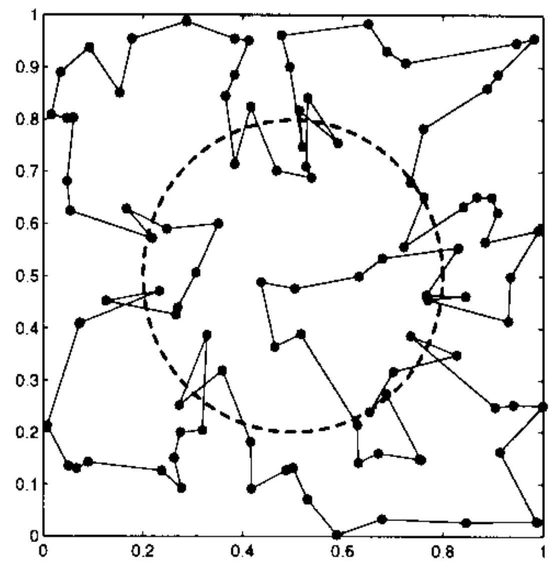
100-city TSP with penalities when crossing the circle



Penalty = 0



Penalty = 0.5



Penalty = -0.3

Tabu Search

- Usually applied to combinatorial optimization problems
  - Retains information of best solutions detected and “path” to reach previous solutions.
  - Memory of itinerary used to restrict search transitions in the neighborhood of a current solution and discourage cycling among recently visited solutions.
  - Restrictions are relaxed when a solution has some preferred characteristics
  - Search selects the best solution from a set of feasible solutions that are subject to restrictions and relaxations.
  - Update list maintaining restrictions and relaxations.
  - Search terminated by stopping criteria
- Requires extensive parameter tuning
- List maintenance/update overhead could be large
- Proposed by Glover (1986)