Heuristic Optimization Methods

Genetic Algorithms

Agenda

- Short overview of Local Search and Local Search-based Metaheuristics
- Introduction to Genetic Algorithms

Local Search (1)

- Basic Idea: Improve the current solution
- Start with some solution
- Find a set of solutions (called neighbors) that are ”close” to the current solution
- If one of these neighbors are better than the current solution, move to that solution
- Repeat until no improvements can be made

Local Search (2)

- Variations
  - Best Improvement (always select the best neighbor)
  - First Improvement (select the first improving neighbor)
  - Random Descent (select neighbors at random)
  - Random Walk (move to neighbors at random)
- Problem: Gets stuck in a local optimum
  - (except Random Walk, which isn't a good method anyway)

Local Search Based Metaheuristics (1)

- Main goal
  - To avoid getting stuck in local optima
- Additional goals
  - Explore a larger part of the search space
  - Attempt to find the global (not just a local) optimum
  - Give a reasonable alternative to exact methods (especially for large/hard problem instances, and where the solution time is important)

Local Search Based Metaheuristics (2)

- The different methods employ very different techniques in order to escape local optima
  - Simulated Annealing relies on controlled random movement
  - Tabu Search relies on memory structures, recording enough information to prevent looping between solutions

Local Search Based Metaheuristics (3)

- The different methods employ very different techniques in order explore a larger part of the search space
  - Simulated Annealing relies on controlled random movement
  - Tabu Search relies on memory structures, recording enough information to guide the search to different areas of the search space (e.g., frequency based diversification)

Local Search Based Metaheuristics (4)

- Which method is better?
- Depends on your needs
  - SA is easier to implement?
  - SA is easier to use/understand?
  - TS is more flexible and robust?
  - TS requires a better understanding of the problem?
  - TS requires more ”tuning”?
  - TS produces better overall results?

Genetic Algorithms

- We have now studied many Metaheuristics based on the idea of a Local Search
- It is time to look at methods that are based on different mechanisms
- The first such method will be the Genetic Algorithm

The Genetic Algorithm

- Directed search algorithms based on the mechanics of biological evolution
- Developed by John Holland, University of Michigan (1970's)
  - To understand the adaptive processes of natural systems
  - To design artificial systems software that retains the robustness of natural systems

Genetic Algorithms

- Provide efficient, effective techniques for optimization and machine learning applications
- Widely-used today in business, scientific and engineering circles

Genetic Algorithms (GA)

- Function Optimization
- AI (Games,Pattern recognition ...)
- OR after a while
- Basic idea:
  - intelligent exploration of the search space based on random search
  - analogies from biology

GA - Analogies with biology

- Representation of complex objects by a vector of simple components
- Chromosomes
- Selective breeding
- Darwinistic evolution

- Classical GA: Binary encoding

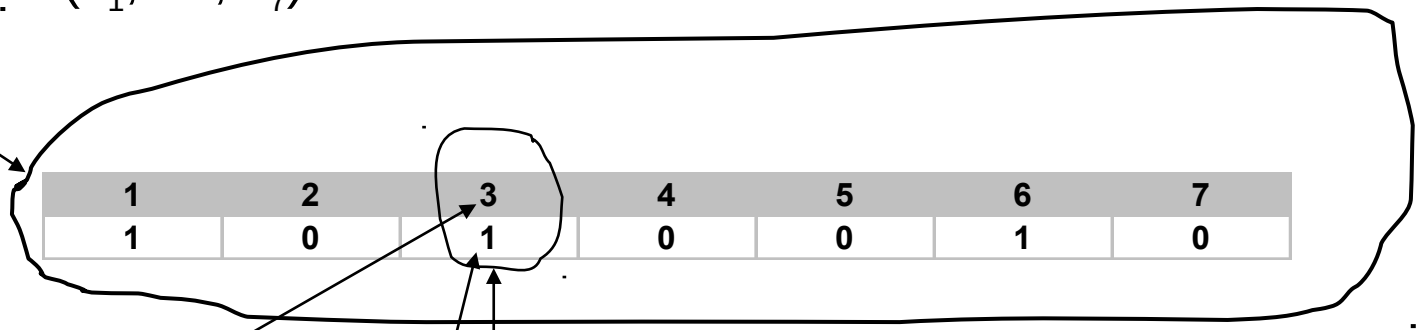
Components of a GA

A problem to solve, and ...

- Encoding technique (*gene, chromosome*)
- Initialization procedure *(creation)*
- Evaluation function *(environment)*
- Selection of parents *(reproduction)*
- Genetic operators *(mutation, recombination)*
- Parameter settings *(practice and art)*

Classical GA: Binary Chromosomes

Chromosome, component vector, vector, string, solution, individual $\mathbf{x}=(x_1, \ldots, x_7)$



Gene, Component, Variable, x_3

Locus, position

Allele, value

$x_3 \in \{0,1\}$

Alleles, domain

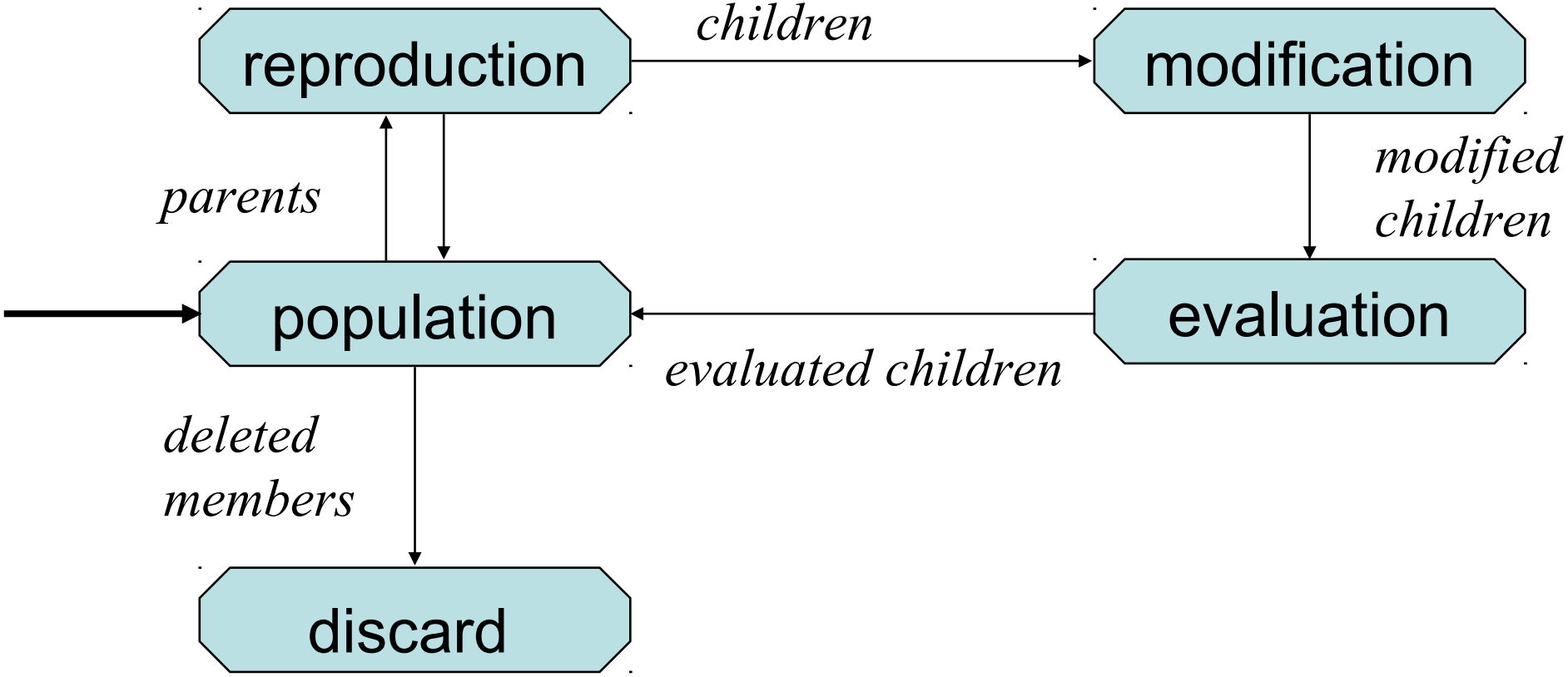
Genotype, Phenotype, Population

- Genotype
  - chromosome
  - Coding of chromosomes
  - coded string, set of coded strings
- Phenotype
  - The physical expression
  - Properties of a set of solutions
- Population – a set of solutions

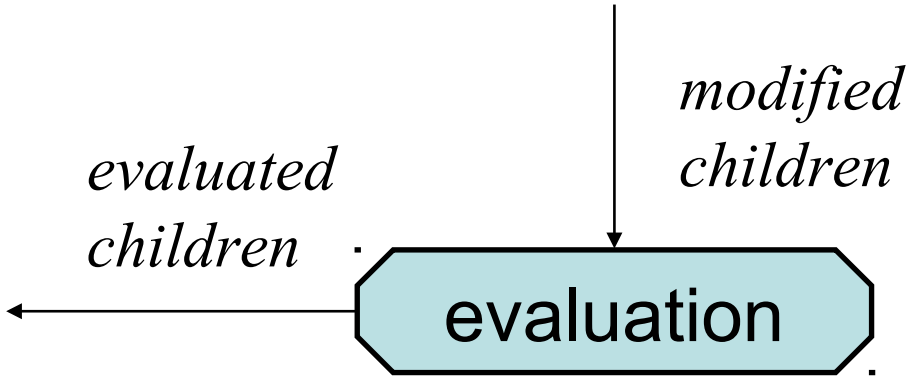
Genetic Algorithm

```
1: Choose an initial population of chromosomes
2: while stopping criterion not met do
3:    while sufficient offspring has not been created do
4:       if condition for crossover is satisfied then
5:          Select parent chromosomes
6:          Choose crossover parameters
7:          Perform crossover
8:       end if
9:       if condition for mutation is satisfied then
10:         Choose mutation points
11:         Perform mutation
12:      end if
13:      Evaluate fitness of offspring
14:   end while
15: end while
```

The GA Cycle of Reproduction



Evaluation



- The evaluator decodes a chromosome and assigns it a fitness measure
- The evaluator is the only link between a classical GA and the problem it is solving

Evaluation of Individuals

- Adaptability – ”fitness”
- Relates to the objective function value for a DOP
- Fitness is maximized
- Used in selection (”*Survival of the fittest*”)
- Often *normalized*

$$f : S \rightarrow [0,1]$$

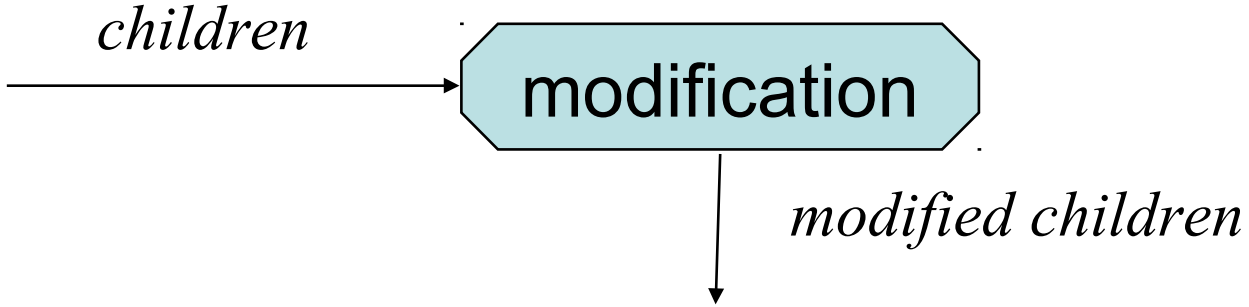
Genetic Operators

- Manipulates chromosomes/solutions
- Mutation: Unary operator
  - Inversions
- Crossover: Binary operator

GA - Evolution

- N generations of populations
- For every step in the evolution
  - Selection of individuals for genetic operations
  - Creation of new individuals (reproduction)
  - Mutation
  - Selection of individuals to survive
- Fixed population size M

Chromosome Modification



- Modifications are stochastically triggered
- Operator types are:
  - Mutation
  - Crossover (recombination)

GA - Mutation

1	2	3	4	5	6	7
1	0	1	0	0	1	0

1	2	3	4	5	6	7
1	0	1	1	0	1	0

Mutation: Local Modification

Before: (1 0 1 1 0 1 1 0)

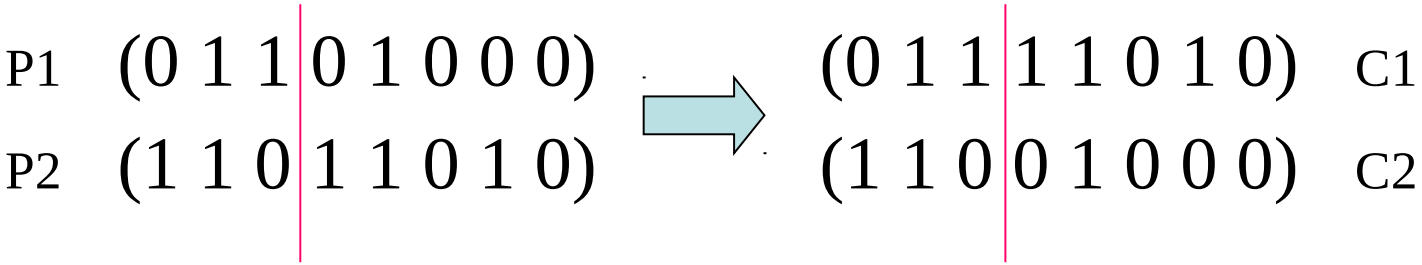
After: (1 0 1 0 0 1 1 0)

Before: (1.38 -69.4 326.44 0.1)

After: (1.38 -67.5 326.44 0.1)

- Causes movement in the search space (local or global)
- Restores lost information to the population

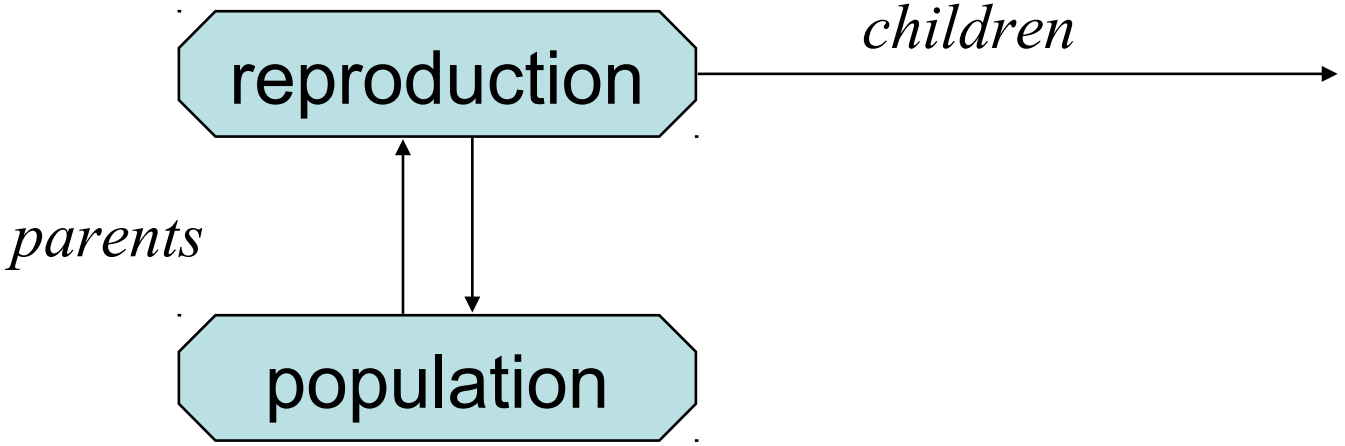
Crossover: Recombination



Crossover is a critical feature of genetic algorithms:

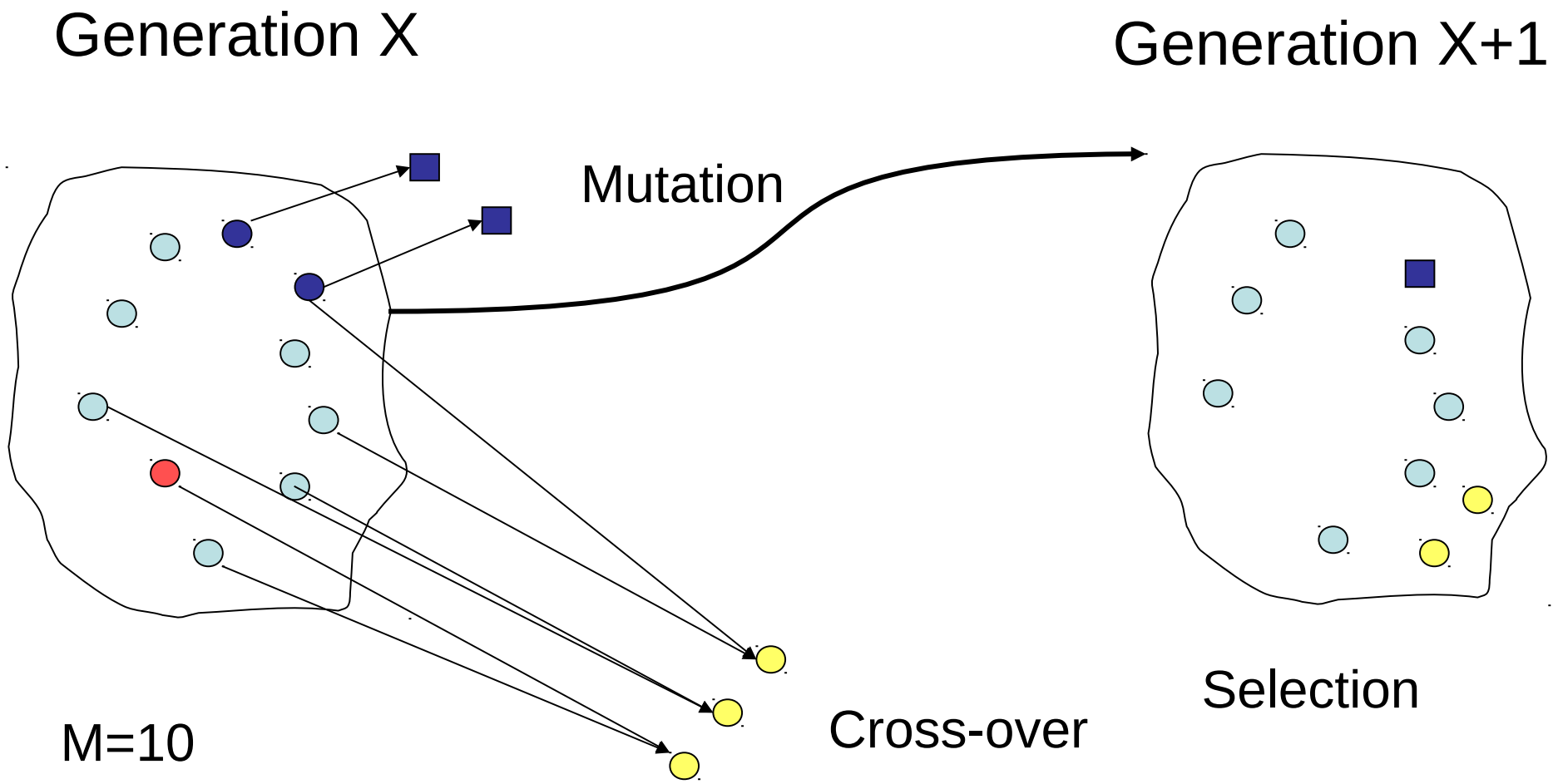
- It greatly accelerates search early in evolution of a population
- It leads to effective combination of schemata (subsolutions on different chromosomes)

Reproduction



Parents are selected at random with selection chances biased in relation to chromosome evaluations

GA - Evolution



Population

Chromosomes could be:

- Bit strings (0101 ... 1100)
- Real numbers (43.2 -33.1 ... 0.0 89.2)
- Permutations of element (E11 E3 E7 ... E1 E15)
- Lists of rules (R1 R2 R3 ... R22 R23)
- Program elements (genetic programming)
- ... any data structure ...

Classical GA: Binary chromosomes

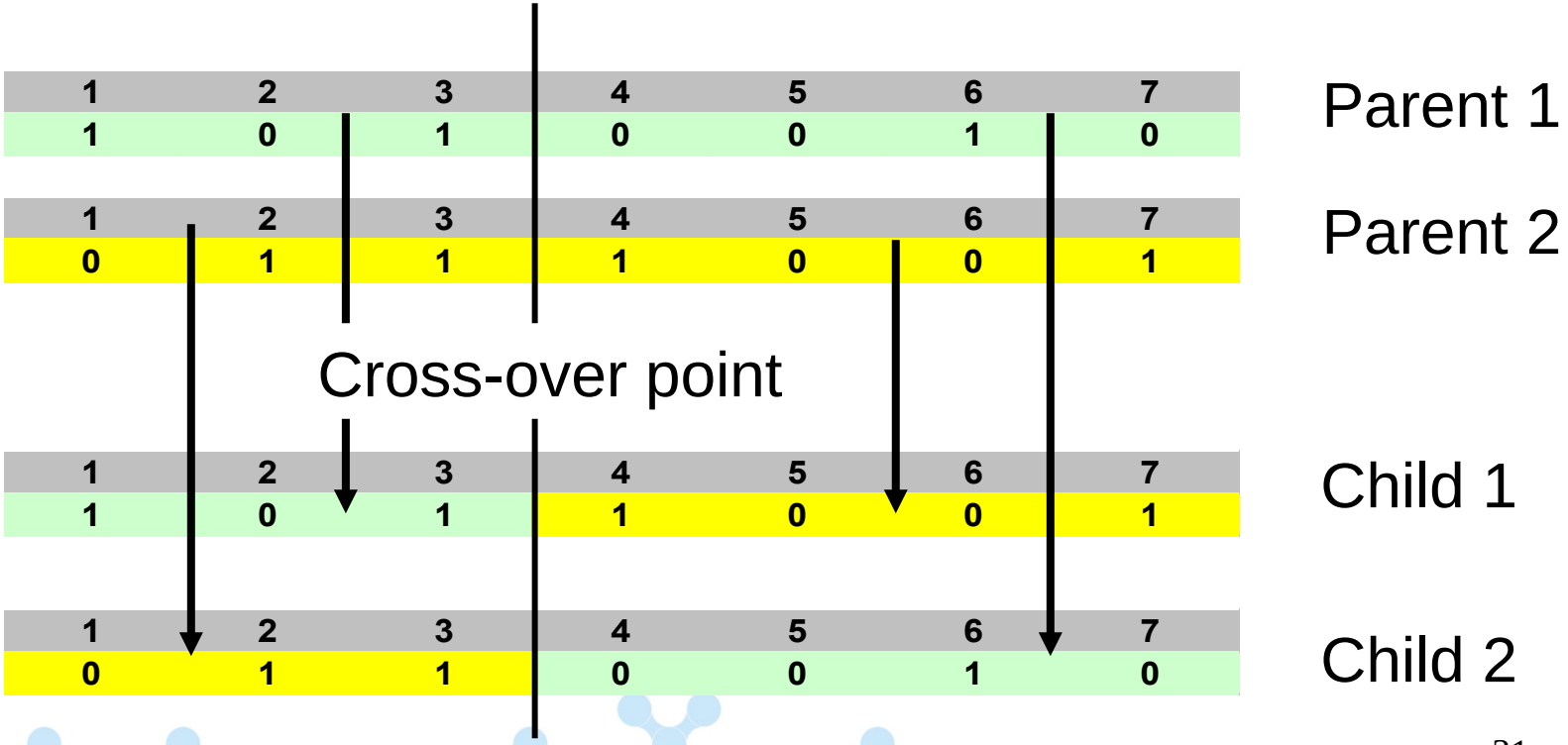
1	2	3	4	5	6	7
1	0	1	0	0	1	0

1	2	3	4	5	6	7
0	1	1	1	0	0	1

- Functional optimization
  - Chromosome corresponds to a binary encoding of a real number - min/max of an arbitrary function
- COP, TSP as an example
  - Binary encoding of a solution
  - Often better with a more direct representation (e.g. sequence representation)

GA - Classical Crossover (1-point)

- One parent is selected based on *fitness*
- The other parent is selected randomly
- Random choice of cross-over point



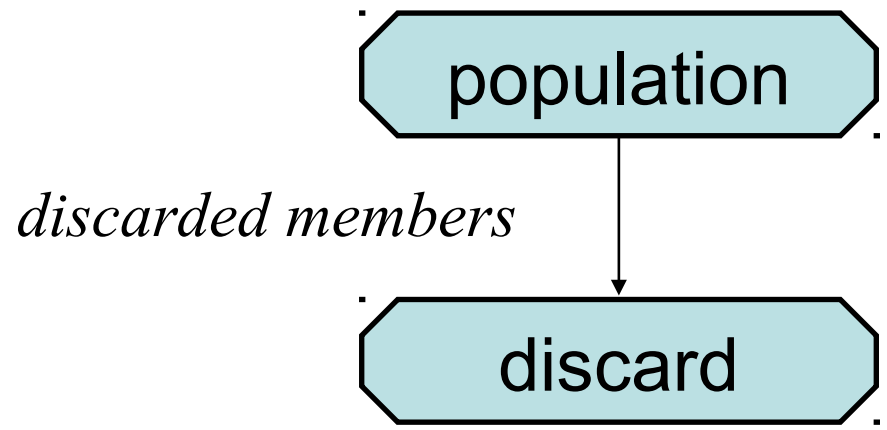
GA – Classical Crossover

- Arbitrary (or worst) individual in the population is changed with one of the two offspring (e.g. the best)
- Reproduce as long as you want
- Can be regarded as a sequence of almost equal populations
- Alternatively:
  - One parent selected according to fitness
  - Crossover until (at least) M offspring are created
  - The new population consists of the offspring
- Lots of other possibilities ...
- Basic GA with classical crossover and mutation often works well

GA – Standard Reproduction Plan

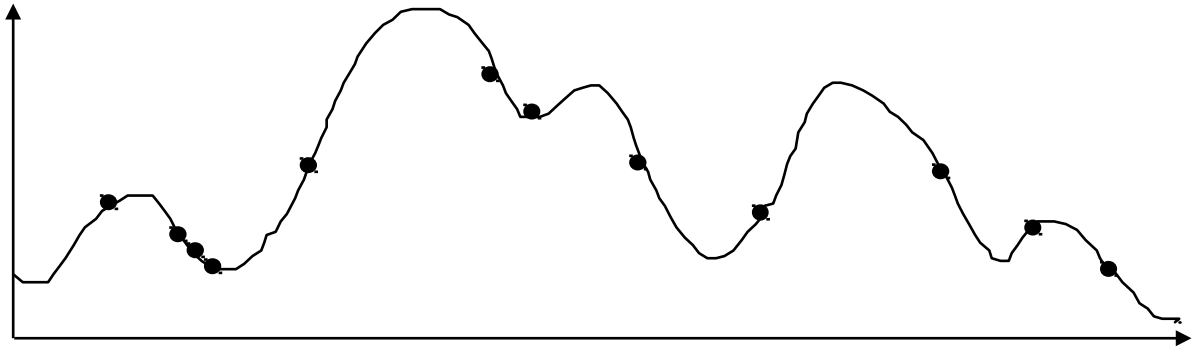
- Fixed population size
- Standard cross-over
  - One parent selected according to fitness
  - The other selected randomly
  - Random cross-over point
  - A random individual is exchanged with one of the offspring
- Mutation
  - A certain probability that an individual mutate
  - Random choice of which gene to mutate
  - Standard: mutation of offspring

Deletion

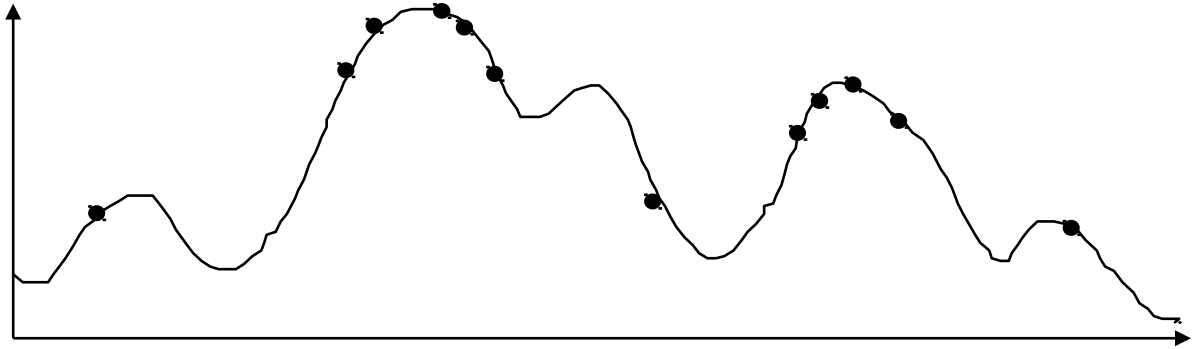


- *Generational* GA:
  entire populations replaced each iteration
- *Steady-state* GA:
  a few members replaced each generation

An Abstract Example



Distribution of Individuals in Generation 0



Distribution of Individuals in Generation N

A Simple Example

The Traveling Salesman Problem:

Find a tour of a given set of cities so that

- each city is visited only once
- the total distance traveled is minimized

Representation

Representation is an ordered list of city numbers known as an *order-based* GA.

1) London	3) Dunedin	5) Beijing	7) Tokyo
2) Venice	4) Singapore	6) Phoenix	8) Victoria

City List 1 (3 5 7 2 1 6 4 8)

City List 2 (2 5 7 6 8 1 3 4)

Crossover

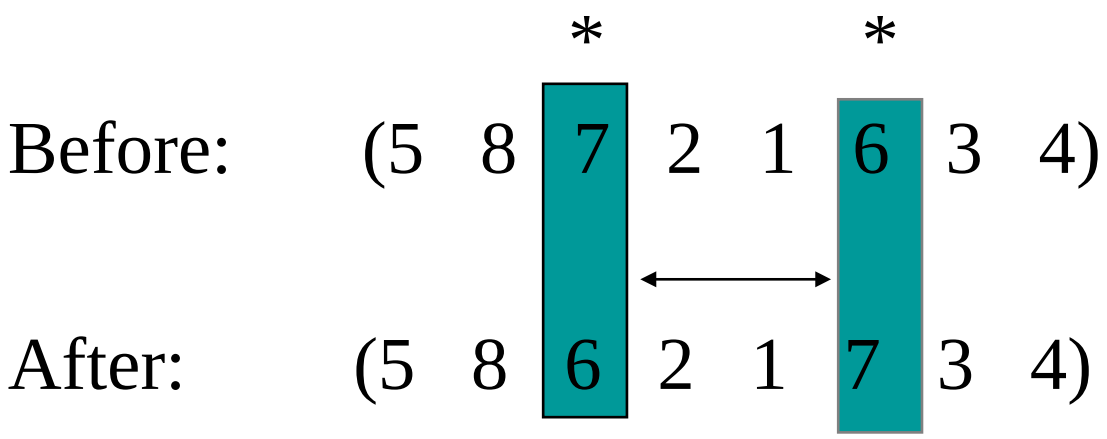
Crossover combines inversion and recombination:

			*			*		
Parent1	(3	5	7	2	1	6	4	8)
Parent2	(2	5	7	6	8	1	3	4)
Child	(5	8	7	2	1	6	3	4)

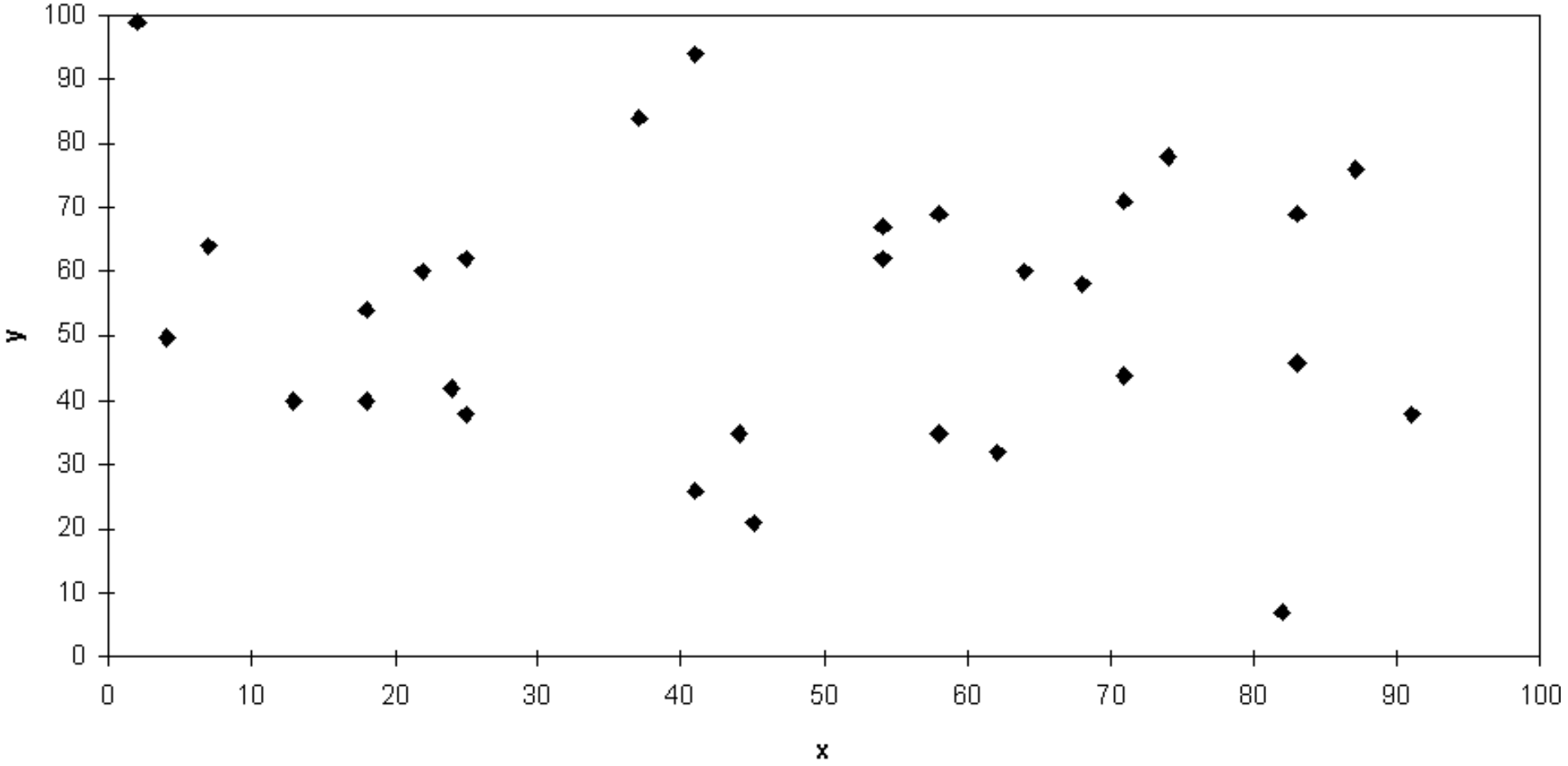
This operator is called order-based crossover.

Mutation

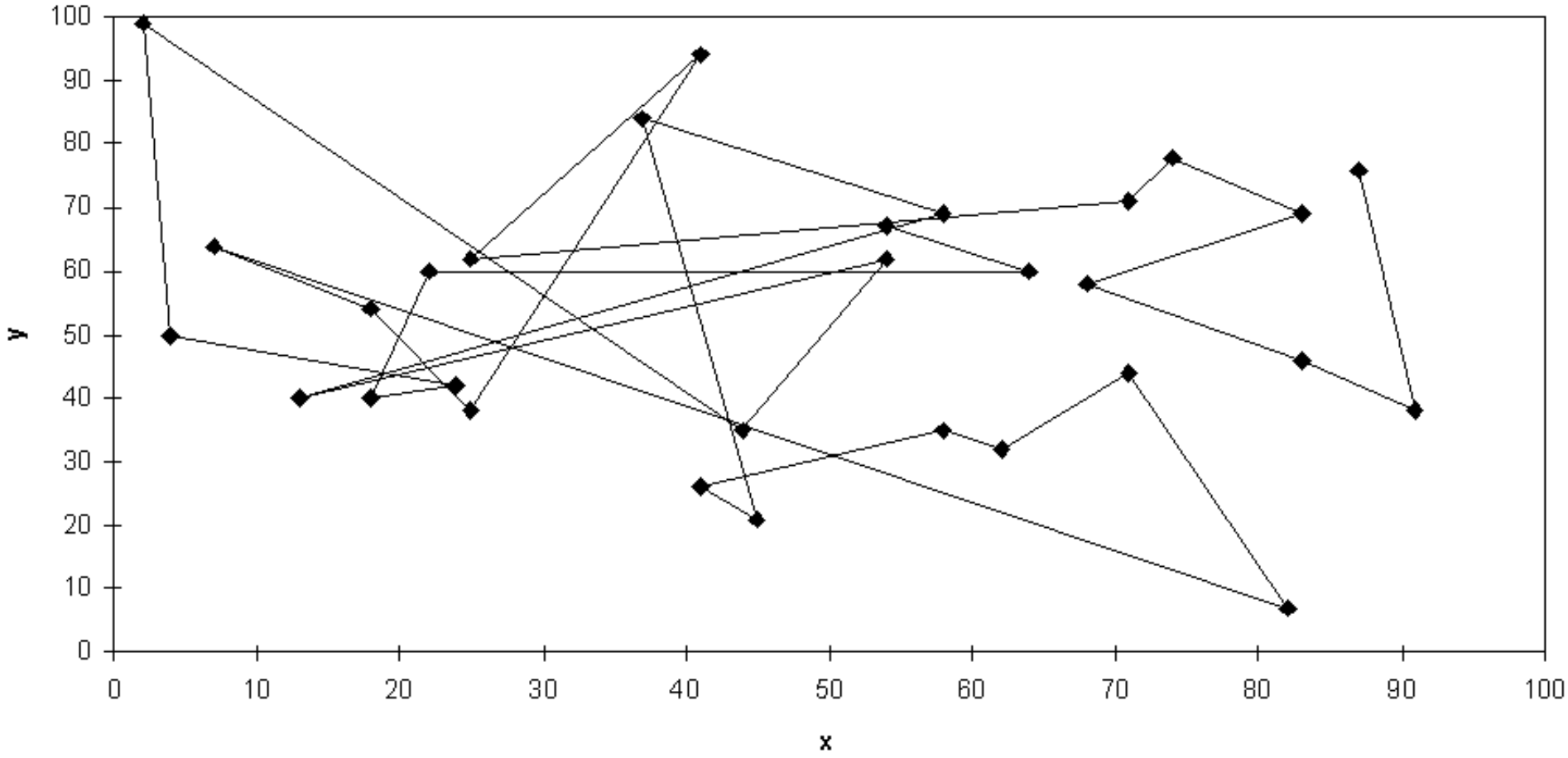
Mutation involves reordering of the list:



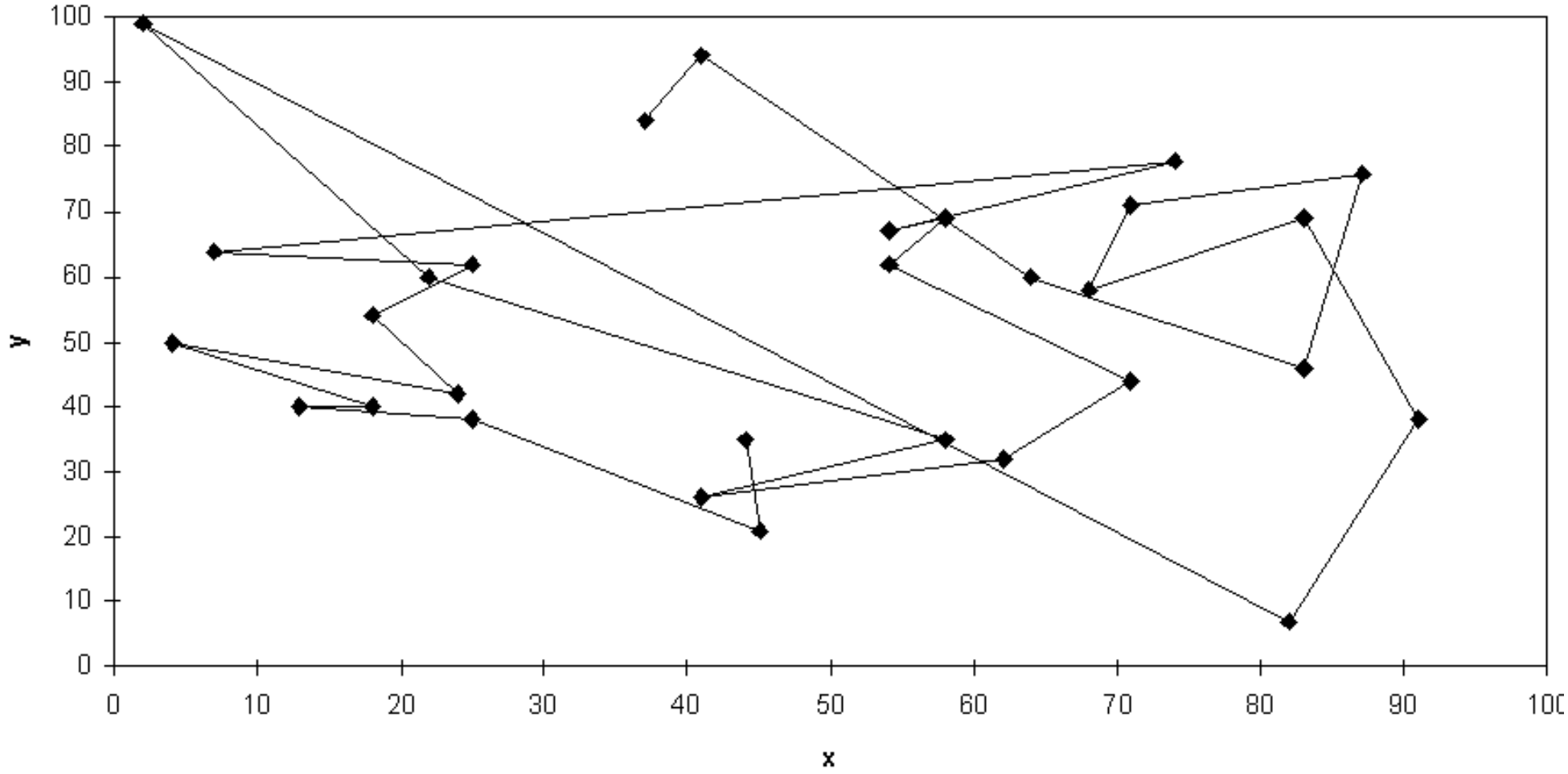
TSP Example: 30 Cities



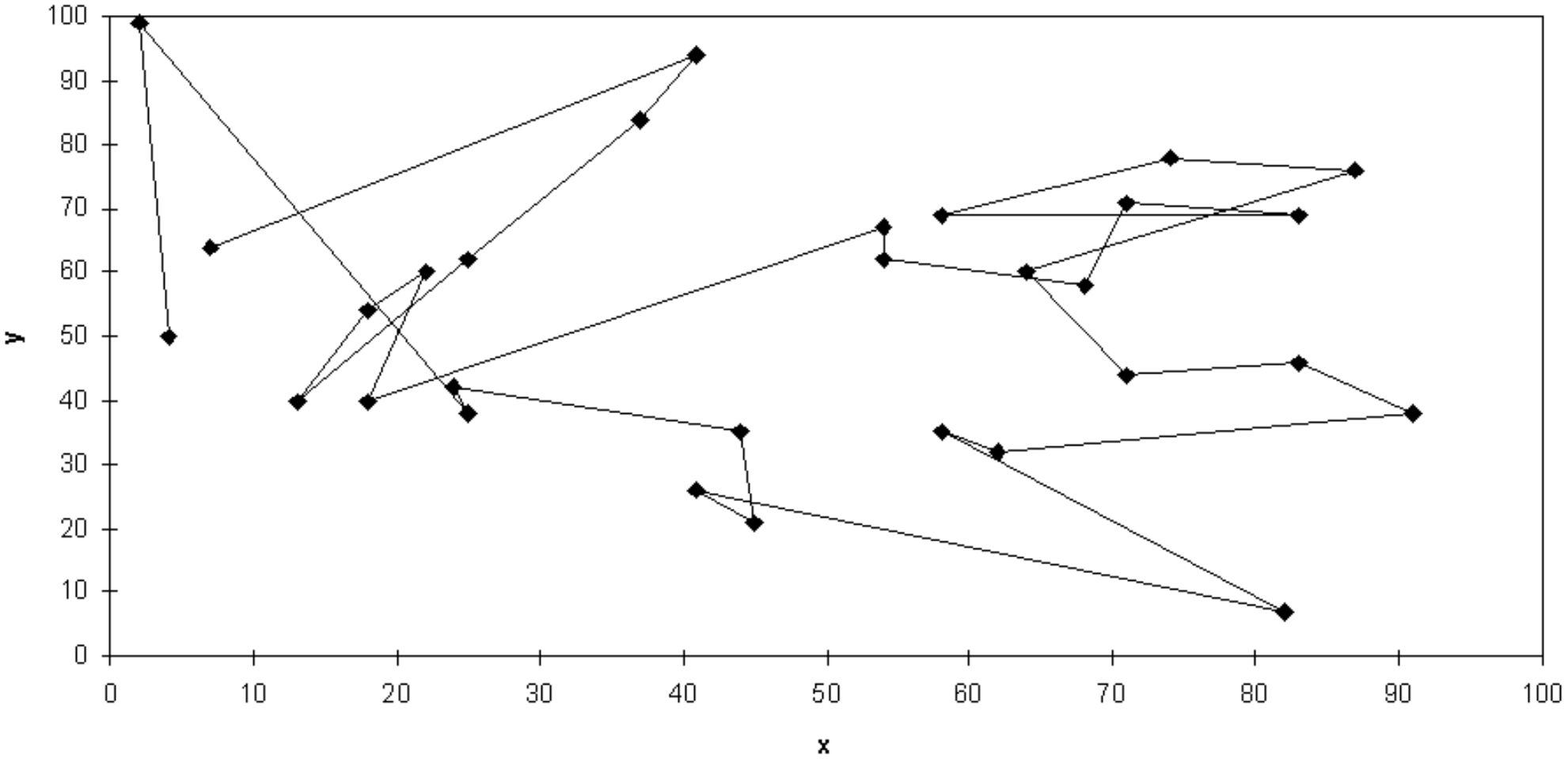
Solution $_i$ (Distance = 941)



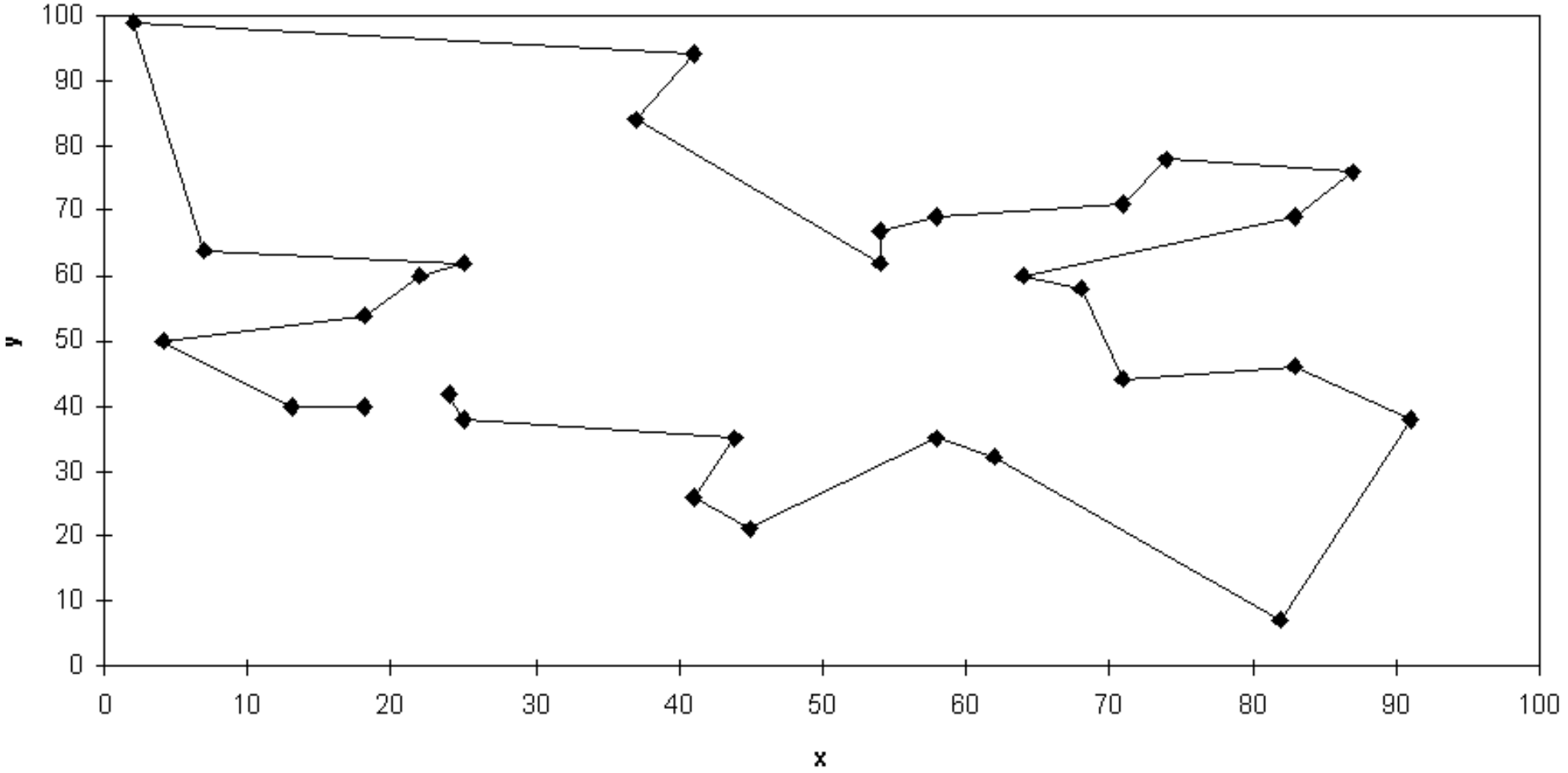
Solution $_j$ (Distance = 800)



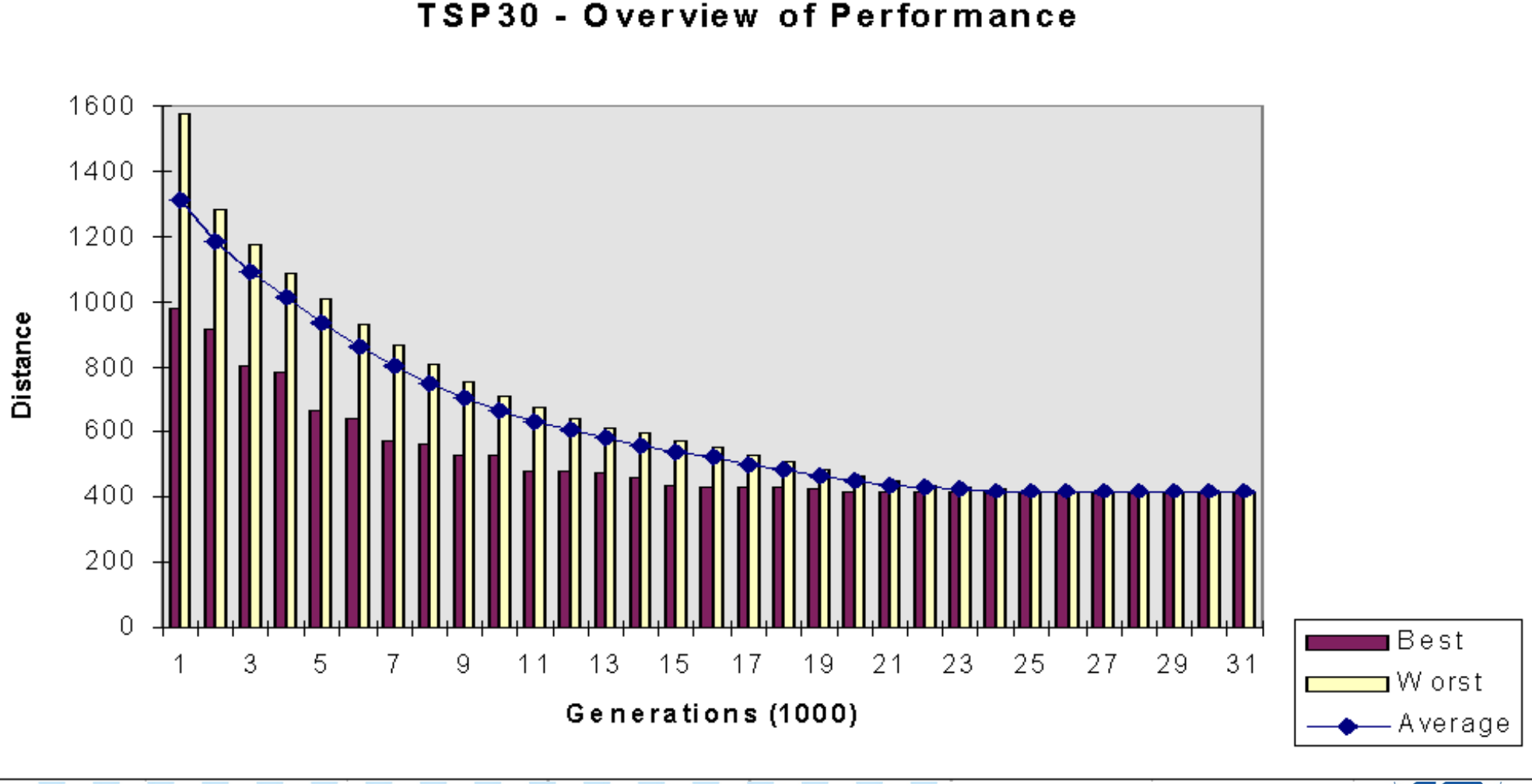
Solution $_k$ (Distance = 652)



Best Solution (Distance = 420)



Overview of Performance



Theoretical Analysis of GA (Holland)

- Schema: subsets of chromosomes that are similar
- The same alleles in certain locations

1	2	3	4	5	6	7
1	0	1	0	0	1	0

1	2	3	4	5	6	7
0	1	1	1	0	0	1

1	2	3	4	5	6	7
*	*	1	*	0	*	*

- A given chromosome can be in many schema

Schema - Fitness

- Every time we evaluate the fitness of a chromosome we collect information about average fitness to each of the schemata
- Theoretically: population can contain ($M*2^n$) schemata
- In practice: overlap

GA – Inherently Parallel

- Many schemata are evaluated in parallel
- Under reasonable assumptions: $O(M^3)$ schemata
- When using the genetic operators, the schemata present in the population will increase or decrease in numbers according to their relative fitness
- Schema theorem

Length and Order for Schemata

1	2	3	4	5	6	7
*	*	1	*	0	*	*

- *Length*: distance between first and last defined position
- *Order*: the number of defined positions
- Example:
  - *Length*: 2
  - *Order*: 2

Fitness Ratio

- The relation between the average fitness of a schema and the average fitness of the population
- $F(S)$ is the fitness of schema S
- $\bar{F}(t)$ is the average fitness of the population at time t
- Let $f(S,t)$ be the fitness ratio for schema S at time t

Schema Theorem (1)

- In a GA with standard reproduction plan where the probabilities for a 1-point crossover and a mutation are P_c og P_m, respectively, and schema S with order $k(S)$ and length $l(S)$ has a fitness-ratio $f(S,t)$ in generation t, then the expected number of copies of schema S in generation $t+1$ is limited by:

$$E[N(S,t+1)] \geq f(S,t)N(S,t)\left[1 - P_c\frac{l(S)}{n-1} - P_m k(S)\right]$$

Schema Theorem (2)

- The theorem states that short, low-order, above average schemata receive exponentially increasing trials in subsequent generations of a GA
  - Such schemata are called *building blocks*
- This is the basis of the Building Block Hypothesis:
  - Combining short, low-order, above average schemata yields high order schemata that also demonstrate above average fitnesses
  - This is the fundamental theorem of GAs, showing how GAs explore similarities as a basis for the search procedure

Later Development of Theory

- Schema theorems for
  - Uniform choice of parents in a crossover
  - Choice of both parents based on fitness
- Exact expressions in the schema theorem
- Analysis by using Walsh-functions (from signal analysis)
- Generalization of schema
  - design of operators

GA - Extensions and Modifications

- Many possibilities for variations
- A lot of literature, chaotic
- Unclear terminology
- Modifications regarding:
  - Population
  - Encoding
  - Operators
  - Hybridization, parallellization

GA: Population Size

- Small populations: undercoverage
- Large population: computationally demanding
- Optimal size increases exponentielly with the string-length in binary encodings
- A size of 30 can often work
  - OK with 10-100
- Between N and 2N (Alander)

GA: Initial Population

- Usually: random strings
- Alternative: seed with good solutions
  - faster convergence
  - premature convergence
- Sophisticated statistical methods
  - Latin hypercubes
- Often problems with infeasibility

GA: Population Updates

- Generation gap
  - Replace the whole population each iteration
- Steady state
  - Add and remove one individual each generation
  - Only use part of the population for reproduction
  - The offspring can replace
    - Parents
    - Worst member of population
    - Randomly selected individuals (doubtful if this works better)
- Avoid duplicates
- Uncertain if the best solution so far will survive
- Elitism – e.g. have a small set of ”queens”
- Selectiv death

GA: Fitness

- The objective function value is rarely suitable
- Naïve goal gives convergence to identical individuals
  - Premature convergence
- Scaling
  - Limited competition in early generations
  - Increase competition over time

$$f(x) = \alpha g(x) + \beta$$

GA: Fitness/Selection

- Use ranking instead of objective function value
- Tournament selection
  - Random choice of groups
  - The best in the group advances to reproduction

GA: Operators

- Mutation – upholds diversity
  - Choice of mutation rate not critical
- Crossover – often effective
  - Late in the search: crossover has smaller effect
  - Selective choice of crossover point
- N-point crossover
  - 2-points has given better performance
  - 8-point crossover has given best results

GA: Generalized Crossover

- Bit-string specifies which genes to use

1	2	3	4	5	6	7
1	0	1	0	0	1	0

Parent 1

1	2	3	4	5	6	7
1	1	1	0	0	1	0

Mask

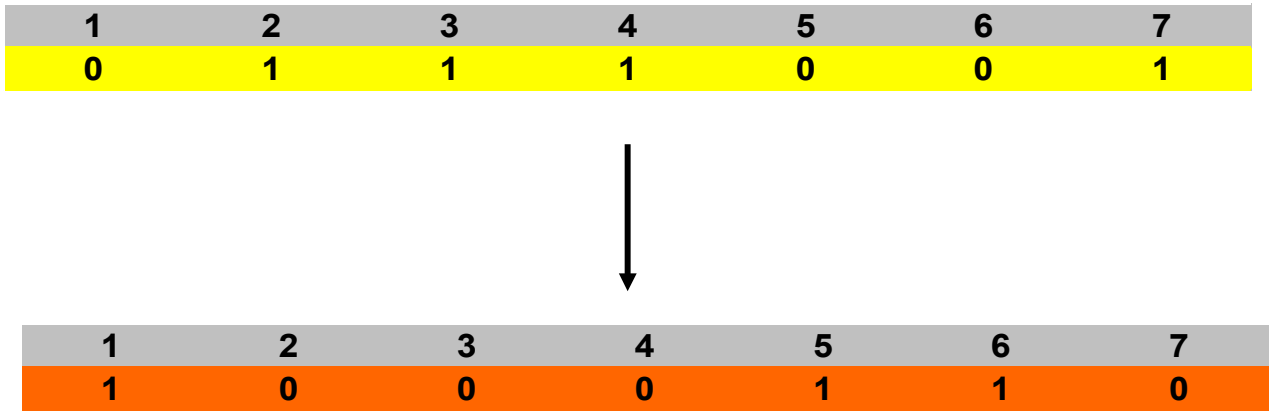
1	2	3	4	5	6	7
0	1	1	1	0	0	1

Parent 2

1	2	3	4	5	6	7
1	0	1	1	0	1	1

Child

GA: Inversion



GA: Hybridization and Parallelization

- GAs strengths and weaknesses:
  - Domain independence
- Hybridization
  - Seed good individuals in the initial population
  - Combine with other Metaheuristics to improve some solutions
- Parallelization
  - Fitness-evaluation
  - Sub-populations
  - The Island Model

Issues for GA Practitioners

- Basic implementation issues:
  - Representation
  - Population size, mutation rate, ...
  - Selection, deletion policies
  - Crossover, mutation operators
- Termination Criteria
- Performance, scalability
- Solution is only as good as the evaluation function (often hardest part)

Benefits of Genetic Algorithms

- Concept is easy to understand
- Modular, separate from application
- Supports multi-objective optimization
- Good for “noisy” environments
- Always an answer; answer gets better with time
- Inherently parallel; easily distributed

Benefits of Genetic Algorithms (cont.)

- Many ways to speed up and improve a GA-based application as knowledge about the problem domain is gained
- Easy to exploit previous or alternate solutions
- Flexible building blocks for hybrid applications
- Substantial history and range of use

When to Use a GA

- Alternate methods are too slow or overly complicated
- Need an exploratory tool to examine new approaches
- Problem is similar to one that has already been successfully solved by using a GA
- Want to hybridize with an existing method
- Benefits of the GA technology meet key problem requirements

Some GA Application Types

Domain	Application Types
Control	gas pipeline, pole balancing, missile evasion, pursuit
Design	semiconductor layout, aircraft design, keyboard configuration, communication networks
Scheduling	manufacturing, facility scheduling, resource allocation
Robotics	trajectory planning
Machine Learning	designing neural networks, improving classification algorithms, classifier systems
Signal Processing	filter design
Game Playing	poker, checkers, prisoner's dilemma
Combinatorial Optimization	set covering, travelling salesman, routing, bin packing, graph colouring and partitioning

GA: Overview

- Important characteristics:
  - Population av solutions
  - Domaine independence – encoding
  - Structure is not exploited
  - Inherent parallell – schema, vocabulary
  - Robust
  - Good mechanisms for intensification
  - Lacking in diversification

Genetic Algorithms

- Bit-string encoding is inappropriate for many combinatorial problems. In particular, crossover may lead to infeasible or meaningless solutions.
- Pure GAs are usually not powerful enough to solve hard combinatorial problems.
- Hybrid GAs use some form of local search as mutation operator to overcome this.

Memetic Algorithms (1)

- Basically, a Memetic Algorithm is a GA with Local Search as improvement mechanism
  - Also known under different names
  - An example of hybridization
- A meme is a unit of cultural information transferable from one mind to another
  - Sounds like gene: the unit carrying inherited information

Memetic Algorithms (2)

- The experience is that GAs do not necessarily perform well in some problem domains
- Using Local Search in addition to the population mechanisms proves to be an improvement
- In a sense this elevates the population search to a search among locally optimal solutions, rather than among any solution in the solution space

Summary of Lecture

- Local Search
  - Short summary
- Genetic Algorithms
  - Population based Metaheuristic
  - Based on genetics:
    - Mutation
    - Combination of chromosomes from parents
  - Hybridization: Memetic Algorithm

Topics for the next Lecture

- Scatter Search (SS)
  - For Local Search based Metaheuristics:
    - SA based on ideas from nature
    - TS based on problem-solving and learning
  - For population based Metaheuristics:
    - GA based on ideas from nature
    - SS based on problem-solving and learning
  - Nature works, but usually very slowly
  - Being clever is better than emulating nature?