Problem Solving Using Search

IT426: Artificial Intelligence

Information Technology Department

SEARCH

Genetic Algorithm

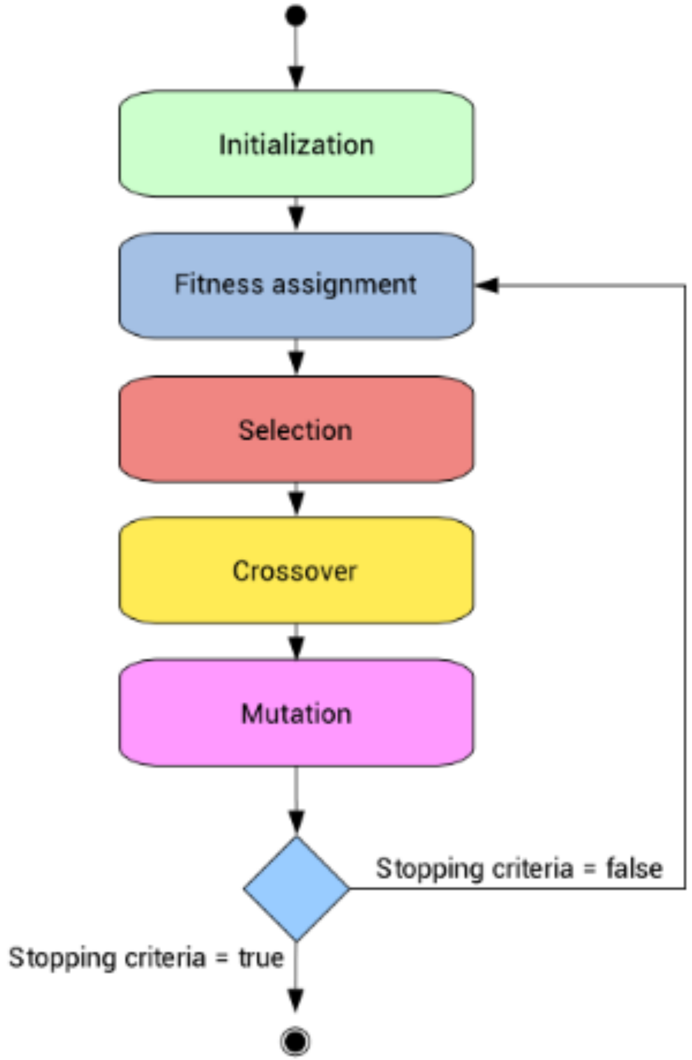
Stochastic search: Genetic Algorithms (GAs)

- GAs emulate ideas from genetics and natural selection and can search potentially large spaces.
- Successors in this case are generated by combining two parent states rather than modifying a single state.
- Genetic algorithms starts with a set of k randomly generated states called <u>Population</u>
- Each state or individual is represented as a string over a finite alphabet. It is also called <u>chromosome</u>.

Genetic Algorithms (GAs)

Before we can apply Genetic Algorithm to a problem, we need to answer:

- How is an individual represented?
- What is the fitness function?
- How are individuals selected?
- How do individuals reproduce?



https://www.neuraldesigner.com/blog/genetic_algorithms_for_feature_selection

Genetic Algorithms (GAs)

maximization

- Each state is rated by the evaluation function called *fitness function*.
- Fitness function should return higher values for better states.
- For reproduction, individuals are *selected* with a probability which is directly proportional to the fitness score.
- For each pair to be mated, a *crossover point* is randomly chosen from the positions in the string.
- The *offspring* themselves are created by *crossing over* the parent strings at the crossover point.
- *Mutation* is performed randomly with a small independent probability.

Simple Genetic Algorithms (SGA)

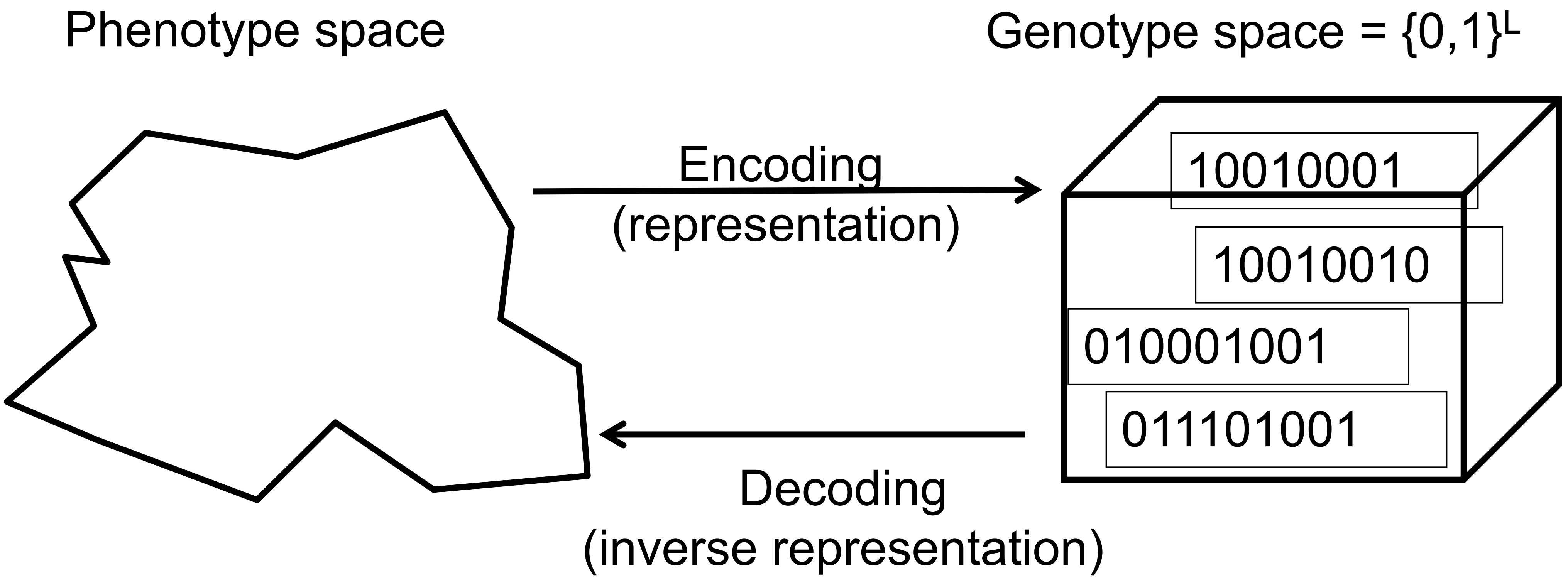
- Holland's original GA is now known as the simple genetic algorithm (SGA)

- Other GAs use different:
  - Representations
  - Mutations
  - Crossovers
  - Selection mechanisms

SGA Techniques - Summary

Representation	Binary strings
Recombination	N-point or uniform
Mutation	Bitwise bit-flipping with fixed probability
Parent selection	Fitness-Proportionate
Survivor selection	All children replace parents
Speciality	Emphasis on crossover

SGA Representation

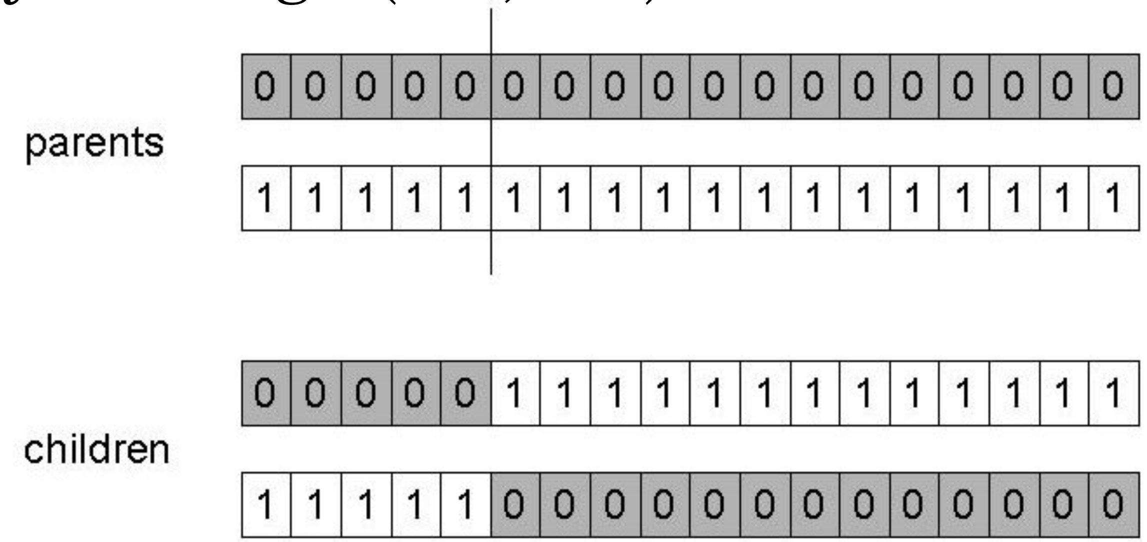


SGA Reproduction Cycle

1. **Select** parents for the mating pool
   - (size of mating pool = population size)
2. **Shuffle** the mating pool
3. For each <u>consecutive pair</u>, apply **crossover** with probability p_c
   - Otherwise copy parents
4. For each <u>offspring</u>, apply **mutation**
   - (bit-flip with probability p_m independently for each bit)
5. **Replace** the whole population with the resulting offspring

SGA Operators: 1-point Crossover

- Choose a random point on the two parents
- Split parents at this crossover point
- Create children by exchanging tails
- P_c typically in range (0.6, 0.9)



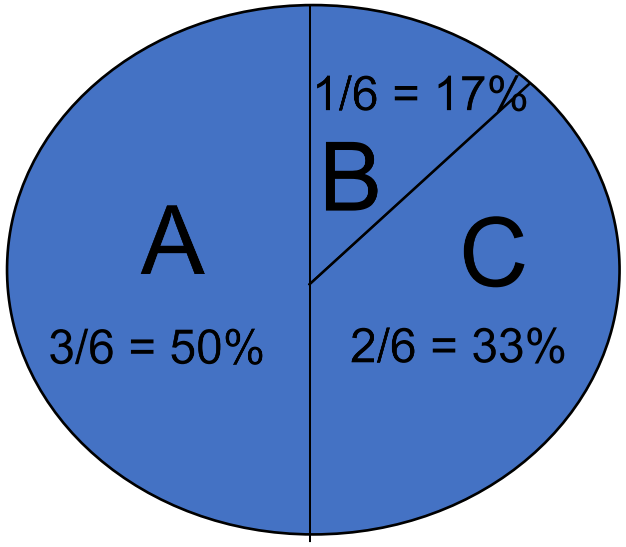
SGA Operators: Mutation

- Alter each gene independently with a probability p_m
- p_m is called the mutation rate
  - Typically between 1/pop_size and 1/ chromosome_length

parent	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
child	0	1	0	0	1	0	1	1	0	0	0	1	0	1	1	0	0	1

SGA Operators: Selection

- Main idea: better individuals get higher chance
  - Chances proportional to fitness
  - Implementation: roulette wheel technique
    - Assign to each individual a part of the roulette wheel
    - Spin the wheel n times to select n individuals



fitness(A) = 3

fitness(B) = 1

fitness(C) = 2

GA Example (1)

- Simple problem: max x^2 over $\{0,1,\ldots,31\}$
- GA approach:
  - Representation: binary code, e.g. 01101 $\leftrightarrow$ 13
  - Population size: 4
  - 1-point xover, bitwise mutation
  - Roulette wheel selection
  - Random initialization
- We show one generational cycle done by hand

X^2 Example: Selection

جامعة الملك سعود
King Saud University

4 times this

rounding

String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

King Saud University

X^2 Example: Crossover

String no.	Mating pool	Crossover point	Offspring after xover	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 | 1	4	0 1 1 0 0	12	144
2	1 1 0 0 | 0	4	1 1 0 0 1	25	625
2	1 1 | 0 0 0	2	1 1 0 1 1	27	729
4	1 0 | 0 1 1	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

got higher

X^2 Example: mutation

String no.	Offspring after xover	Offspring after mutation	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	1 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 1 0 0	18	324
Sum				2354
Average				588.5
Max				729

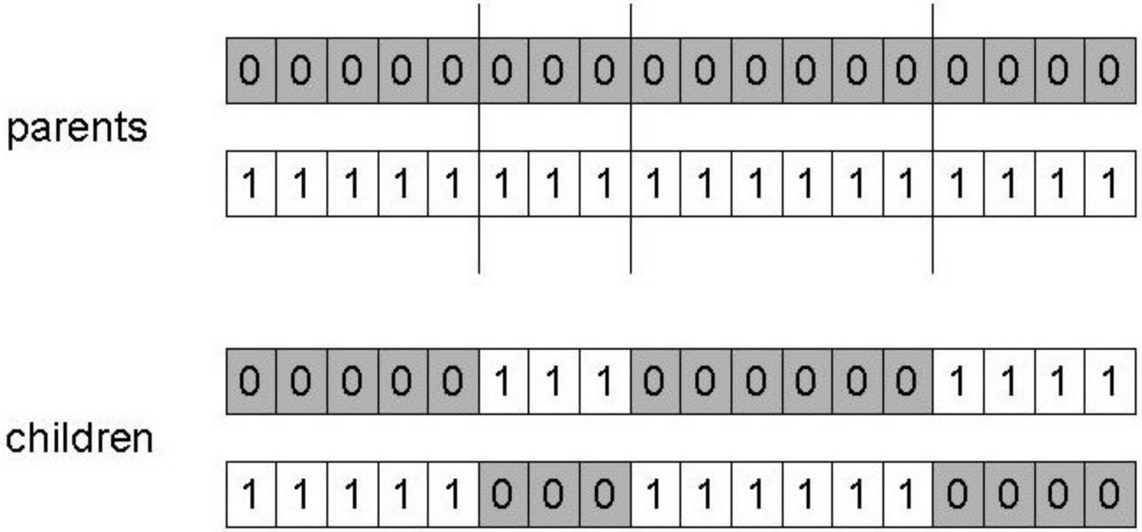
got higher

Summary of SGA Process

- **Select** the **initial population** (usually randomly).
- **Select percent probability** of the following:
  1. Crossover (often 0.6-0.8)
  2. Mutation (often about .001).
- **Calculate** the **fitness** value for each population member.
- **Normalize** fitness values and use to determine probabilities for reproduction.
- **Reproduce** new generation with the same number of members, using probabilities from 3.
- **Pair off** strings to cross over randomly.
- **Select** crossing sites (often 2) randomly for each pair.
- **Mutate** on a bit-by-bit basis. then replace the whole population with the new offsprings (bye parents)
- If more generations, go to step 3.
- If completed, stop and output results.

n-Point Crossover

- Choose n random crossover points
- Split along those points
- Glue parts, alternating between parents
- Generalization of 1 point (still some positional bias)



Uniform Crossover

- Assign 'heads' to one parent, 'tails' to the other
- Flip a coin for each gene of the first child
- Make an inverse copy of the gene for the second child
- Inheritance is independent of position

{0.3,0.6, 0.2, 0.1, 0.6, 0.1, 0.7, 0.9, 0.2, 0.3, 0.4, 0.7, 0.4, 0.6, 0.7,0.3, 0.2, 0.9}

parents	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
children	0	1	0	0	1	0	1	1	0	0	0	1	0	1	1	0	0	1
	1	0	1	1	0	0	0	0	1	1	1	0	1	0	0	1	1	0

Uniform Crossover

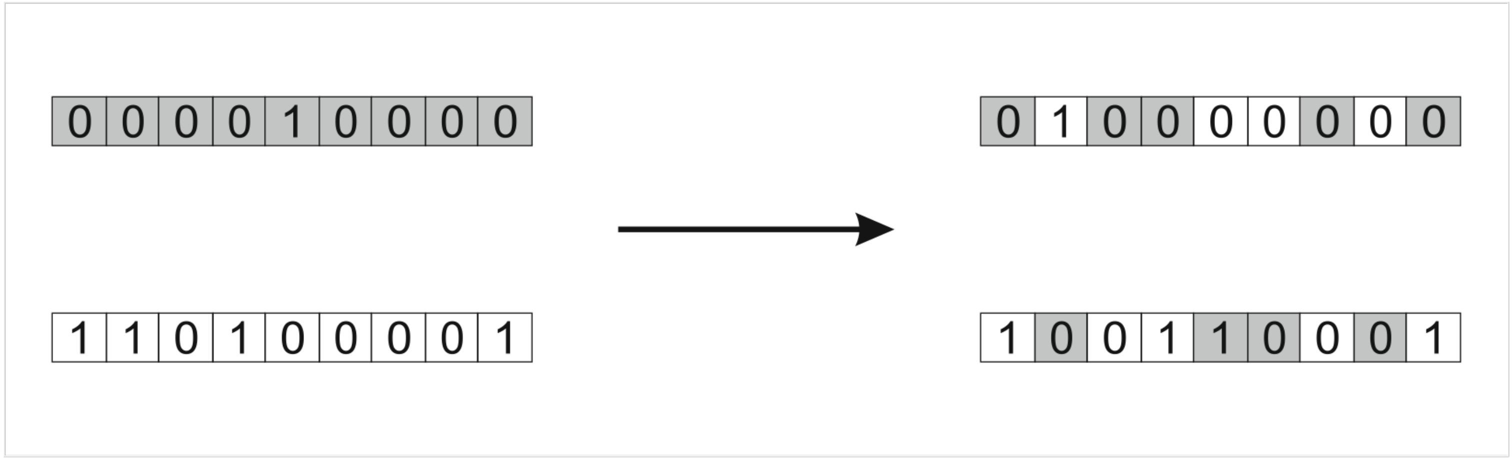


Fig. 4.3. Uniform crossover. The array [0.3, 0.6, 0.1, 0.4, 0.8, 0.7, 0.3, 0.5, 0.3] of random numbers and $p = 0.5$ were used to decide inheritance for this example.

c

either do a crossover or not,
if its bigger we dont crossover

Crossover OR mutation?

Answer: depends on the problem, but

- In general, it is good to have both
- both have another role
- Mutation-only-Evolutionary Algorithm (EA) is possible, xover-only-EA would not work

xover would teleport you to places yet you cant dig deeper/exploit

Crossover OR mutation?

Exploration: Discovering promising areas in the search space, i.e. gaining information on the problem

Exploitation: Optimising within a promising area, i.e. using information
There is co-operation AND competition between them

- **Crossover is explorative**, it makes a *big* jump to an area somewhere "in between" two (parent) areas
- **Mutation is exploitative**, it creates random *small* diversions, thereby staying near (in the area of) the parent

Crossover OR mutation?

- Only crossover can combine information from two parents
- Only mutation can introduce new information (alleles)
- Crossover does not change the allele frequencies of the population
- To hit the optimum you often need a ‘lucky’ mutation

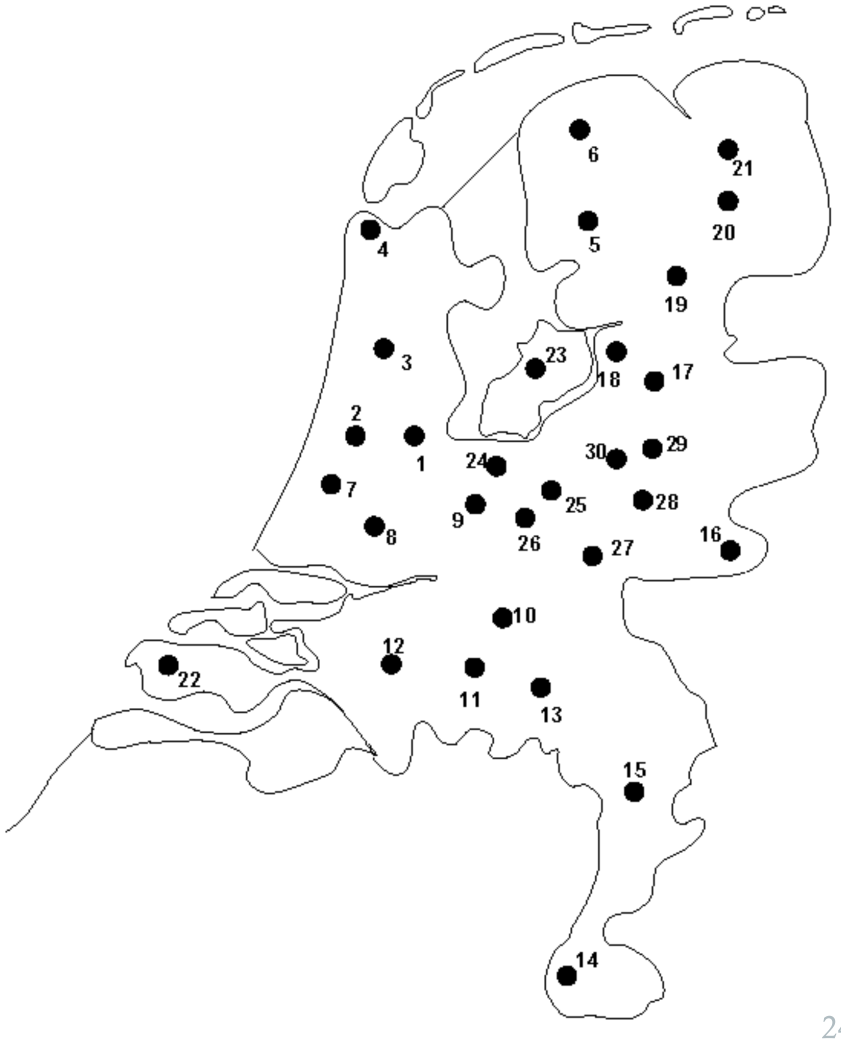
Permutation representation: TSP example

- **Problem:**
  - Given n cities
  - Find a complete tour with minimal length

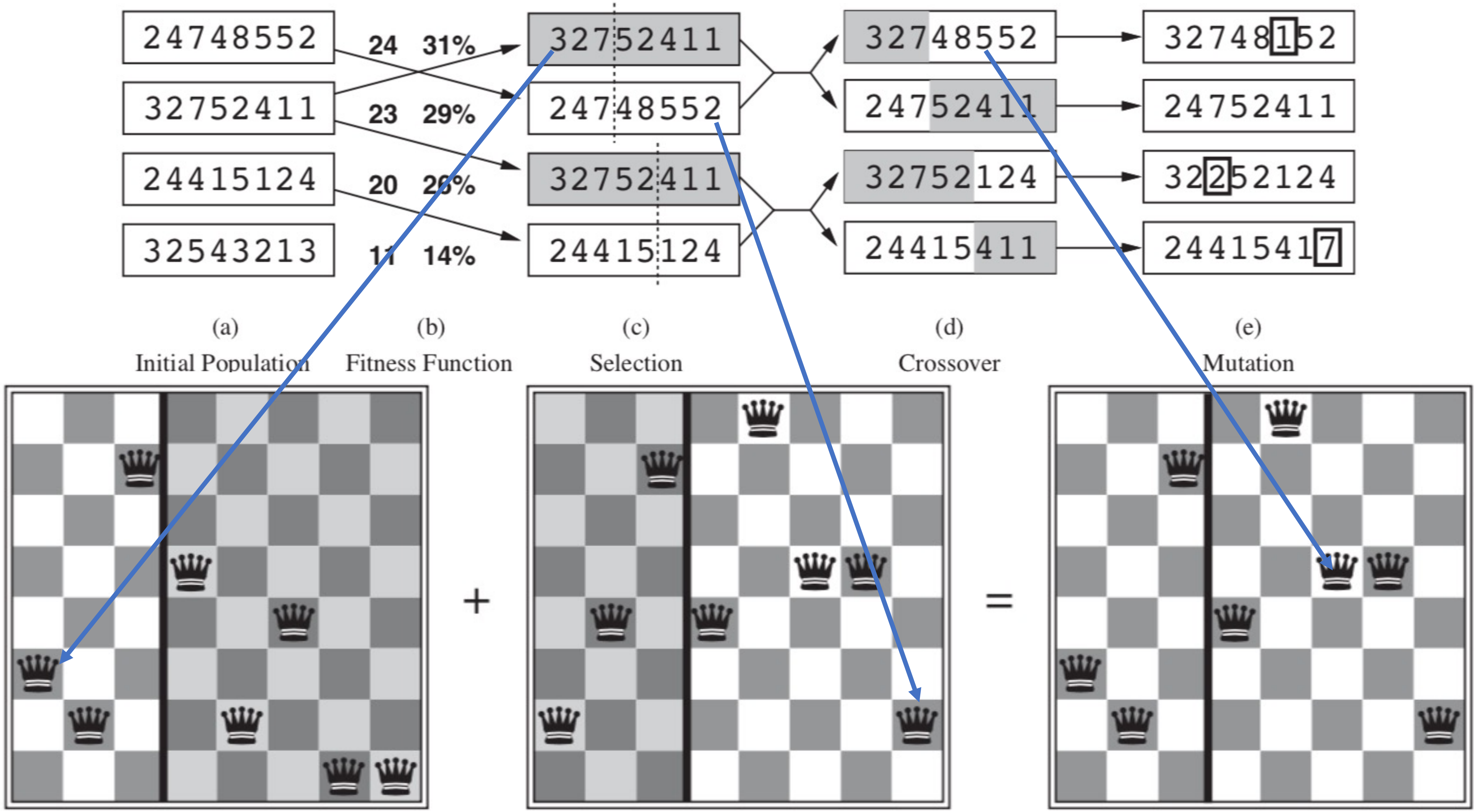
- **Encoding:**
  - Label the cities 1, 2, … , n
  - One complete tour is one permutation

    (e.g. for n =4 [1,2,3,4], [3,4,2,1] are OK)

- **Search space** is BIG:
  - for 30 cities there are $30! \approx 10^{32}$ possible tours



GA Example (2)



Artificial Intelligence A Modern Approach, Third Edition, 127

When to Use a GA

genetic algorithm

- Alternate solutions 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 solution

Advantages of GA

- Perform a “global” search in the search space
  1. Work with a *population of individuals (candidate solutions), rather* than with just one candidate solution at a time)
  2. Avoid the use of “greedy” heuristics (e.g., start at a given city and visit one city at a time, choosing the nearest city at each step)

- Easy to implement

Benefits of GA

- Concept is easy to understand
- Modular, separate from application
- Supports multi-objective optimization
- Good for "noisy" environments by adding a mechanism, uses population does not return a single sol
- Always an answer; answer gets better with time also for local and simulated. not like greedy
- Inherently parallel; easily distributed

Disadvantages of GA

- Do not offer any guarantee of finding the optimal solution, nor any lower bound on the quality of the solutions to be found
- Are computationally expensive in large-scale problems
- Have several parameters (crossover probability, mutation probabilities, population size, number of generations, etc.) whose “optimization” is not a trivial task might face in the project

Issues for GA Practitioners

Choosing basic implementation issues:

- Representation
- Population size, mutation rate, ...
- Selection, deletion policies
- Crossover, mutation operators
- Termination Criteria

Solution is only as good as the evaluation function (often hardest part)

f(x)
fitness, objective

Related Reading

- 15 Real-World Applications of Genetic Algorithm
  - https://www.brainz.org/15-real-world-applications-genetic-algorithms/