

7. Genetic Algorithms

7.2 The Knapsack Problem

Fall 2010

Instructor: Dr. Masoud Yaghini

Outline

- Problem Definition
- Representations
- Fitness Function
- Handling of Constraints
- Population
- Parent Selection Mechanism
- Variation Operators
- Survivor Selection
- Initialization and Termination
- References

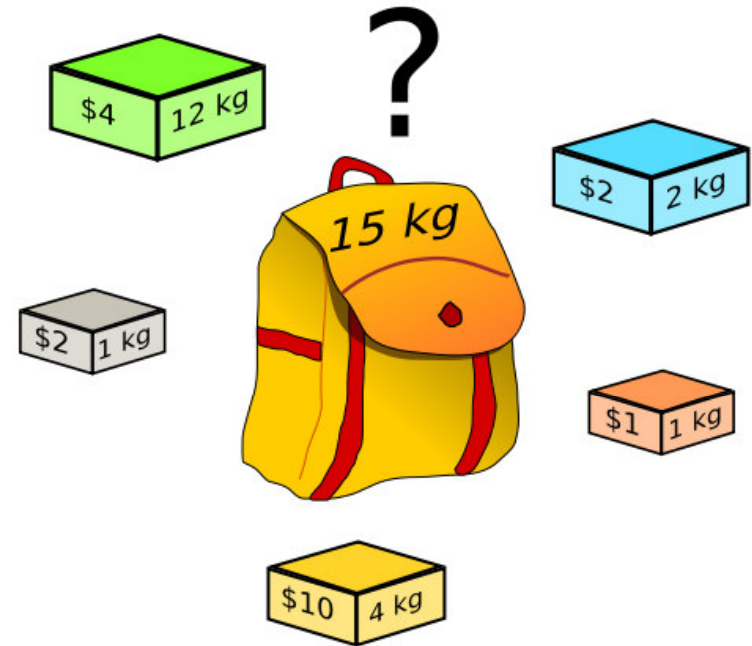
The image features a large green rectangular area on the left side. A white semi-circle is cut out from the right edge of this green area. To the right of the white semi-circle, the text "Problem Definition" is written in a dark blue, bold, sans-serif font. Below the text, a dark blue horizontal bar with rounded ends extends from the right edge of the green area towards the right side of the image.

Problem Definition

Example: The Knapsack problem

- There are n items:
 - Each item i has a weight w_i
 - Each item i has a value v_i
- The knapsack has a limited capacity of W units.
- We can take one of each item at most

$$\begin{aligned} \max \quad & \sum_i v_i * x_i \quad i = 1, 2, \dots, n \\ \text{subject to} \quad & \sum_i w_i * x_i \leq W \\ & x_i \in \{0, 1\} \end{aligned}$$



Example: The Knapsack problem

- **Example:**

- **Item:** 1 2 3 4 5 6 7
- **Benefit:** 5 8 3 2 7 9 4
- **Weight:** 7 8 4 10 4 6 4

- **Knapsack holds a maximum of 22 pounds**

- **Fill it to get the maximum benefit**

- **The problem description:**

- Maximize $\sum_i v_i$

- While $\sum_i w_i \leq W$



Representations



Representations

- Candidate solutions (**individuals**) exist in phenotype space
- They are encoded in chromosomes, which exist in genotype space
- Chromosomes contain **genes**, which are usually in fixed positions called and have a value (**allele**)

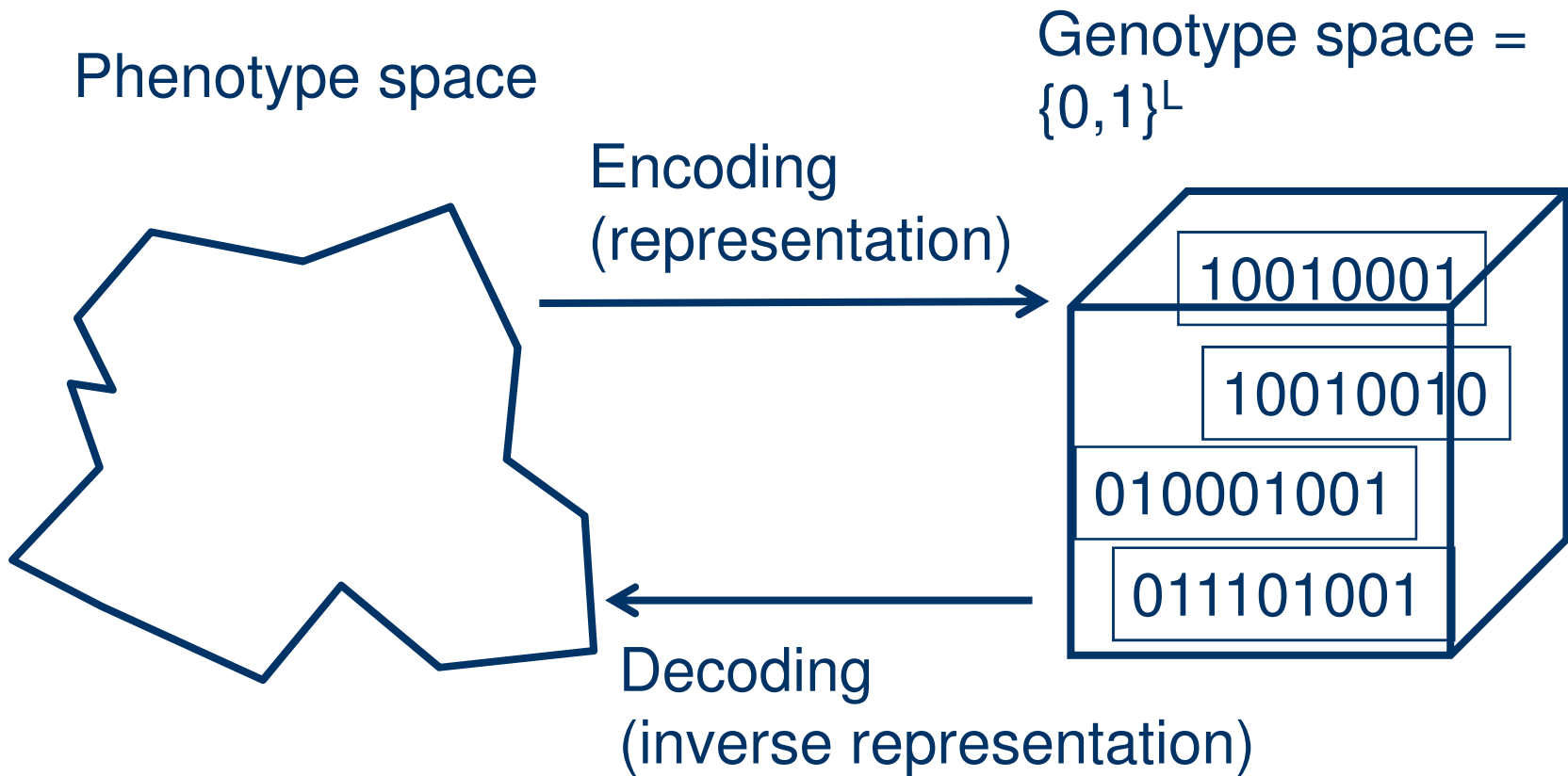
Representations

- For example, given an optimization problem on integers:
 - The given **set of integers** would form the set of phenotypes
 - They can be represented by binary code
 - **18** would be seen as a phenotype, and 10010 as a genotype representing it
- In order to find the **global optimum**, every feasible solution must be represented in genotype space

Representations

- A solution (**a good phenotype**) is obtained by decoding the best genotype after **termination**
- Coding can be done in two different ways:
 - **Encoding:**
 - the mapping from the phenotype to the genotype space
 - phenotype=> genotype
 - **Decoding:**
 - the inverse mapping from genotypes to phenotypes
 - genotype=> phenotype

Binary Representation



Knapsack Example: Representations

- Solutions take the form of a string of 1's and 0's
- Where
 - 0 = don't take the item in a given positions
 - 1 = take the item in a given positions
- Solutions: Also known as strings of genes called **Chromosomes**
- Example chromosomes:
 - 1100100 \Rightarrow items {1,2,5} included in sack
 - 0010000 \Rightarrow items {3} included in sack
 - 0001100 \Rightarrow items {4,5} included in sack
 - 0100001 \Rightarrow items {2,7} included in sack
- The genotype space G is the set of all strings with size 2^n



Fitness Function

Fitness Function

- **Fitness function** represents the requirements that the population should adapt to
- It defines what improvement means
 - i.e, **quality function** or **objective function**
- Assigns a single real-valued fitness to each phenotype which forms the basis for selection
- Typically we talk about fitness being maximised
 - Some problems may be best posed as minimisation problems, but conversion is easy

Knapsack Example: Fitness Function

- The fitness function as the total benefit, which is the sum of
 - the gene values in a string solution \mathbf{x} their representative benefit coefficient

$$Fitness = \sum_i v_i : \left(\sum_i w_i \leq W \right)$$

Genetic Algorithms: Part 2

Knapsack Example: Solution 1

Item	1	2	3	4	5	6	7
Solution	1	1	0	0	1	0	0
Benefit	5	8	3	2	7	9	4
Weight	7	8	4	10	4	6	4

- Fitness: $5 + 8 + 7 = 20$
- Weight: $7 + 8 + 4 = 19 \leq 22$

The slide features a green background on the left side, which contains a white semi-circle. The title 'Handling of Constraints' is centered within this white area. A dark blue horizontal bar with rounded ends extends from the green area towards the right side of the slide.

Handling of Constraints

Genetic Algorithms: Part 2

Knapsack Example: Solution 2 overweighted

Item	1	2	3	4	5	6	7
Solution	0	1	0	1	0	1	0
Benefit	5	8	3	2	7	9	4
Weight	7	8	4	10	4	6	4

- Weight: $8 + 10 + 6 = 24 > 22$

The trouble with constraints and EAs

- Standard reproduction operators are **blind** to constraints.
 - **Recombining** two feasible individuals can result in infeasible new individuals.
 - **Mutating** a feasible individual can result in an infeasible new individual.

Handling of Constraints

- **Constraint handling:**
 - Eliminating infeasible candidates
 - Penalizing functions
 - Repairing infeasible candidates

Eliminating of Infeasible Candidates

- Additional version of penalty approach (i.e., the most severe penalty: death penalty)
- Disadvantages:
 1. For some problems the probability of generating a feasible solution is relatively small and the algorithm spends a significant amount of time evaluating illegal individuals.
 2. In this approach non-feasible solutions do not contribute to the gene-pool of any population

Penalizing Functions

- Generating potential solutions without considering the constraints and then to penalize them by decreasing the "goodness" of the evaluation function.
- A variety of possible penalty functions which can be applied
 - assign a constant as a penalty measure
 - assign a penalty measure depend on the degree of violation: the larger violation is, the greater penalty is imposed
 - the growth of the penalty can be logarithmic, linear, quadratic, exponential, etc.

Knapsack Example: if overweight

- Penalize:

$$Fitness = \begin{cases} \sum_i v_i : \left(\sum_i w_i \leq W \right) \\ W - \sum_i w_i : (otherwise) \end{cases}$$

Genetic Algorithms: Part 2

Knapsack Example: Solution 2 overweighted

Item	1	2	3	4	5	6	7
Solution	0	1	0	1	0	1	0
Benefit	5	8	3	2	7	9	4
Weight	7	8	4	10	4	6	4

- Fitness (Benefit): $22 - (8 + 10 + 6) = -2$
- Weight: $8 + 10 + 6 = 24 > 22$

Repair Algorithms

- Special **repair algorithms** to "correct" any infeasible solutions so generated.
- **Disadvantages:**
 1. Such repair algorithms might be computationally intensive to run and the resulting algorithm must be tailored to the particular application.
 2. Moreover, for some problems the process of correcting a solution may be as difficult as solving the original problem.

Knapsack Example: if overweight

- Repair:
 - When creating solution we read from left to right along binary string,
 - We first check to see if including the item would break our capacity constraint
 - We interpret it as meaning include this item, IF it does not take us over the weight constraint
 - We do not add the right of the current position to the solution
 - This makes the mapping from genotype to phenotype space many-to-one.



Population



Population

- Population holds (representations of) **possible solutions**
- Usually has a fixed size and is a **multiset** of genotypes
- Selection operators usually take whole population into account
 - i.e., parent selection mechanisms are relative to current generation

Population

- **Diversity of a population** refers to the difference solutions
 - The number of fitness's / phenotypes / genotypes present
- **Population size** may be around 50, but for difficult problems it can be larger
 - Too few chromosomes \Rightarrow the GA won't have the diversity needed to find a good solution
 - Too many \Rightarrow the GA will be much slower, without much improvement in the quality of the solution

Knapsack Example: Population

- We will work with a **population size of 50**
- We will create **same number of offspring** as we have members our initial population (50)



Parent Selection Mechanism



Parent Selection Mechanism

- An individual is a **parent** if it has been selected to create offspring
- In GA, **parent selection** is usually probabilistic:
 - high quality individuals (solutions) more likely to become parents than low quality
 - but not guaranteed
 - worst in current population usually has non-zero probability of becoming a parent
- This stochastic nature can aid escape from local optima

Knapsack Example: Parents Selection

- We will use a **tournament** for selecting the parents
 - where each time we pick two members of the population at random (with replacement), and the one with the highest fitness value



Variation Operators



Variation Operators

- **Variation operators** are to generate new candidate solutions
- Usually divided into two types according to their number of inputs:
 - number of inputs = 1 : **mutation operators**
 - number of inputs ≥ 2 : **Recombination operators**
 - number of inputs = 2 typically called **crossover**
- There has been much debate about relative importance of recombination and mutation
 - Nowadays most GAs use both
 - Choice of particular variation operators depends upon genotype representation used.

Mutation

- **Mutation** is **unary variation operator** (it applies to one object as input)
- Acts on one genotype and delivers another, the child or offspring of it
- A mutation operator is **stochastic**
- Nature of the mutation operator depends upon the genotype representation
 - For example: flipping one or several bits with a given (small) probability.

Recombination

- **Crossover** is a **binary variation operator** (it applies to **two objects** as input) is called
- It merges information from **two parent** genotypes into one or two offspring genotypes
- Similar to mutation recombination is a **stochastic** operator
 - Choice of what information to merge is stochastic

Recombination

- The principle behind recombination is simple,
 - by mating two individuals with different but desirable features, we can produce an offspring that combines both of two features
- **Most offspring** may be worse, or the same as the parents
- Hope is that some are better by combining elements of genotypes that lead to good traits

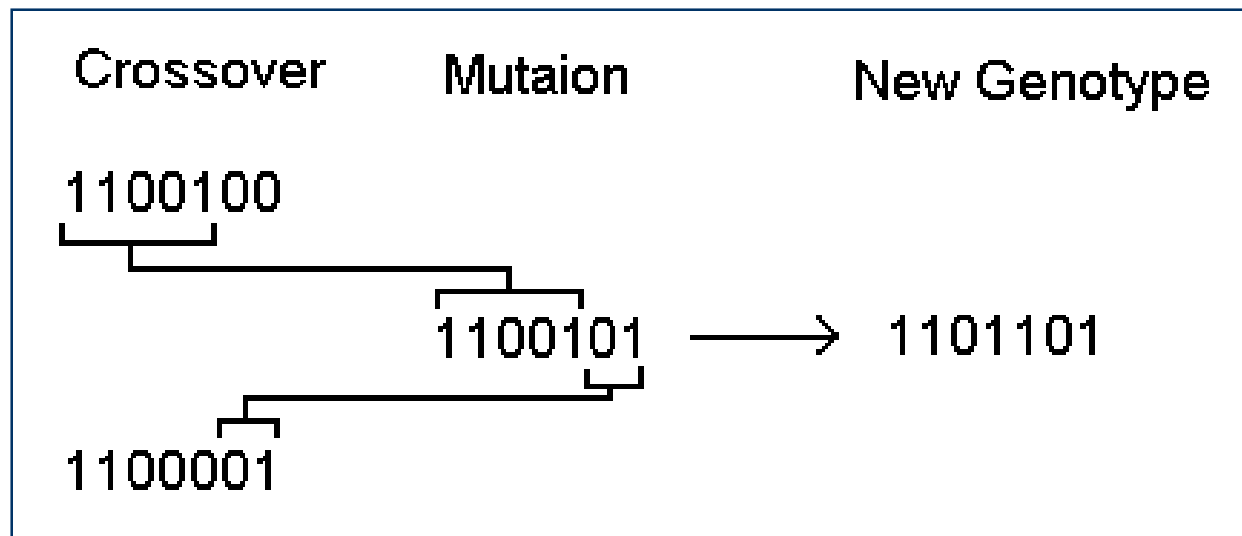
Knapsack Example: Recombination

- A suitable recombination operator is **one-point crossover**
- We will apply crossover
 - with 70% probability and
 - for other 30% we will make copies of the parents
- We align two parents for crossing over and pick a random point along their length
- The two offspring are created by exchanging the tails of the parents at that point

Knapsack Example: Mutation Operator

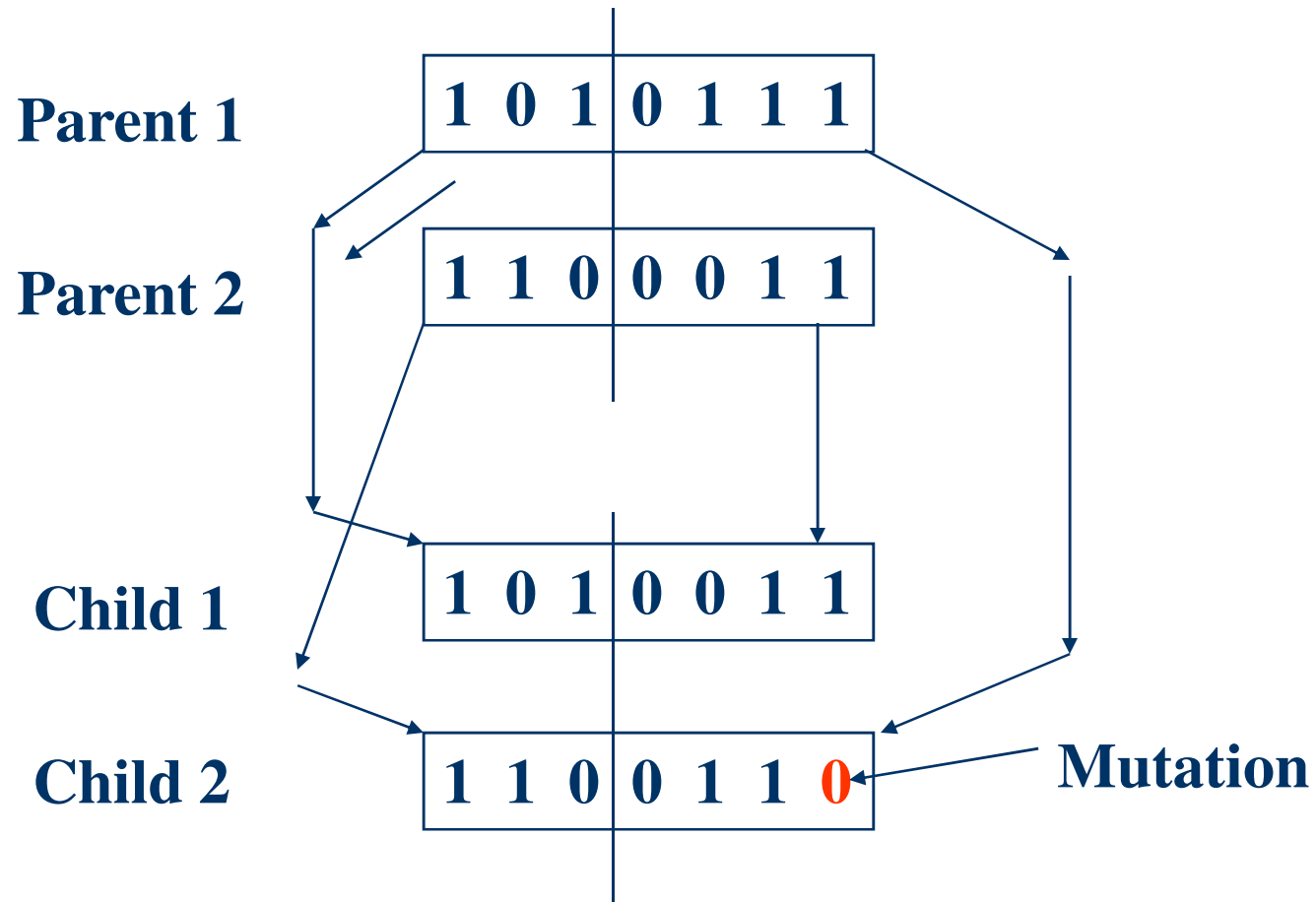
- A suitable mutation operator is so-called **bit-flipping**
- **Mutation rate:**
 - In each position we invert the value with a small probability $[0, 1)$
- We define a **mutation rate** of $P_m = 1/n$,
 - i.e. that on average 1 gene per recombination mutated
 - n : number of genes in a chromosome

Knapsack Example: Crossover & Mutation Operator



Genetic Algorithms: Part 2

Knapsack Example: Crossover & Mutation Operator



The image features a large green shape on the left side, which has a white semi-circular cutout. The text "Survivor Selection" is positioned within this white area. A dark blue horizontal bar with rounded ends extends from the green shape towards the right side of the image.

Survivor Selection

Survivor Selection (Replacement)

- **Survivor selection mechanism (replacement)** is called after created the offspring of the selected parents
- Most GAs use fixed population size so need a way of going from (parents + offspring) to next generation
- Survivor selection often is **deterministic**
 - **Fitness based**: e.g., rank parents+offspring and take best
 - **Age based**: make as many offspring as parents and delete all parents

Knapsack Example: Survivor Selection

- We will use a **generational scheme** for survivor selection
 - In this scheme, all the population in each iteration are discarded and replaced by their offspring

The slide features a green background on the left side, which contains a white semi-circular cutout. The title text is positioned within this white area. A dark blue horizontal bar with rounded ends extends from the green area towards the right side of the slide.

Initialization and Termination

Initialization

- **Initialization** usually done at **random**
- The first population is created by randomly generated individuals
- We can use **problem-specific heuristics**, to seed an initial population with higher fitness
- Need to ensure even spread and mixture of possible allele values

Termination

- **Termination condition** checked every generation
- Reaching some (known/hoped for) fitness
 - GAs are stochastic and there are no guarantees to reach a specific fitness
- Therefore we need another condition

Termination

- Options for certainly stops:
 - Reaching maximum allowed **CPU time** elapses
 - Reaching some maximum allowed **number of generations**
 - Reaching some specified **number of generations without fitness improvement**
 - Reaching **population convergence**

Convergence

- **Gene convergence:**
 - when individuals (e.g. 95% of them) have the same value for that gene
- **Chromosomes convergence:**
 - When **chromosomes (all genes)** have converged
- **Population convergence:**
 - average fitness approaches best fitness

Gene & Chromosomes Convergence

- Example 1: Gene convergence
 - 100% of 4th gene is 1
- Example 2: Chromosomes convergence
 - 75% of genes is same

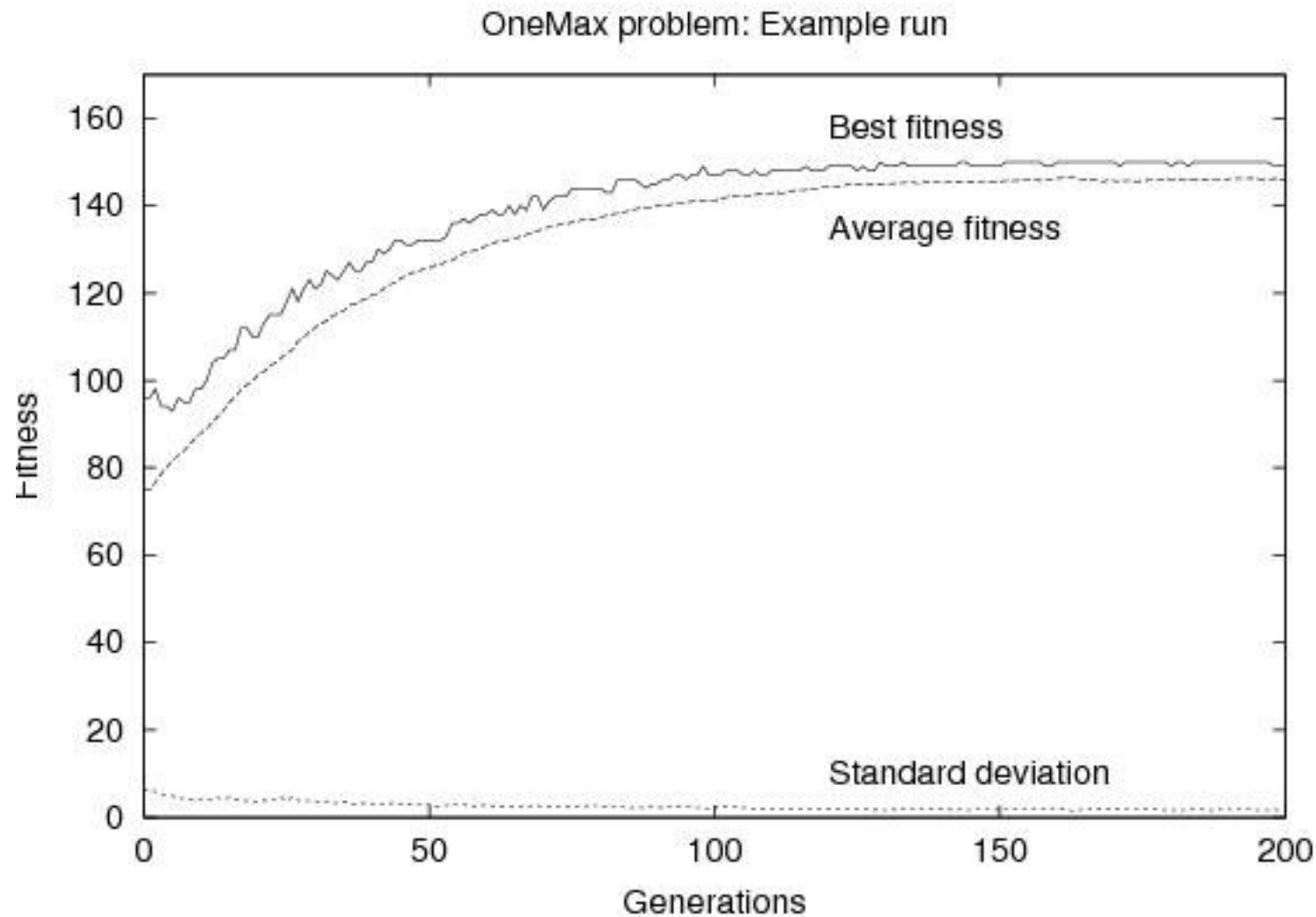
Example 1:

i1: 01010
i2: 10010
i3: 00010
i4: 11010

Example 2:

i1: 11111
i2: 11111
i3: 11010
i4: 11111

Population Convergence



Knapsack Example: Initialization & Termination

- **Initialization:**

- We all do initialization by random choice of 0 and 1 in each position of our initial population

- **Termination:**

- We will run our algorithm until no improvement in the fitness of the best number of the population has been observed for **25 generations**

Knapsack Example: Summary

Representation: Binary strings of length n

Recombination: One-point crossover

Recombination probability: 70%

Mutation: Each value inverted with independent probability P_m

Mutation probability P_m : $1/n$ (Average 1 gene per recombination mutated)

Parent Selection: Best out of random 2 (Tournament)

Knapsack Example: Summary

Survivor Selection: Replace all (Generational)

Population Size: 50

Number of offspring: 50

Initialization: Random

Termination Condition: No improvement in last 25 generations

Note: this **only one possible** set of operators and parameters!



References



References

- Eiben and Smith. **Introduction to Evolutionary Computing**, Springer-Verlag, New York, 2003.
- J. Drezo A. Petrowski, P. Siarry E. Taillard, **Metaheuristics for Hard Optimization**, Springer-Verlag, 2006.



The End

