Univerzitet u Novom Sadu Fakultet tehničkih nauka

Dokumentacija za projektni zadatak

Studenti: Viktorija Avanesova, SV 87/2024

Roman Avanesov, SV 88/2024

Predmet: Nelinearno programiranje i evolutivni algoritmi

Broj projektnog zadatka: 2

Tema projektnog zadatka: Genetski algoritam, problem putujućeg trgovca

1. **Problem Description**

The task is to solve the 0/1 knapsack problem. Initially, a set of items is given. Each item is described by its weight ​ and value ​ ​. There is a knapsack with a maximum capacity . The goal is to select a subset of items such that the total value of the selected items is maximized and the total weight does not exceed .  
Implement a solution to this problem using a genetic algorithm.

1. **Input**

A genetic algorithm approximately solves an optimization problem through evolutionary operations on a population of binary chromosomes, where each chromosome represents a particular selection of items.

**Main Steps of the Genetic Algorithm**

1. **Initialization** - create a random initial population of bit‑vector chromosomes.
2. **Fitness Evaluation** - for each chromosome, compute its fitness as the total value of the items packed into the knapsack, or 0 if the total weight exceeds .
3. **Selection** - use tournament selection to choose parent chromosomes for producing offspring.
4. **Mutation -** introduce small random changes (bit flips) to offspring chromosomes to maintain genetic diversity.
5. **Replacement** - use the offspring to form the new population, and repeat the process until convergence.

**3. Implementation**

**3.1. Input Data**  
The input file data\_knapsack01.txt is given. The first line specifies the knapsack’s maximum weight . From the second line onward, each line describes a single item by its weight and value.

**3.2. Program Structure**

**3.2.1. Function load\_data**  
Parses data\_knapsack01.txt, extracting the capacity and the arrays of weights and values.

**3.2.2. Function fitness**  
Computes the fitness of a chromosome: if the total weight of the selected items exceeds , return 0; otherwise return the total value.

**3.2.3. Function tournament\_selection**  
Implements tournament selection of parents. For each parent to select, choose random indices, pick the best among them, and repeat until you have as many parents as the original population size.

**3.2.4. Functions one\_point\_crossover and two\_point\_crossover**  
Two crossover operators:

* 1. **One‑point crossover** – split each parent pair at a single random point and swap tails.
  2. **Two‑point crossover** – choose two cut points and exchange the middle segments.

**3.2.5. Function mutate**  
Bit‑flip mutation operator: randomly flip bits in a chromosome with a low probability.

**3.2.6. Function run\_ga**

The run\_ga function implements the main loop of the genetic algorithm. It begins by initializing a population of candidate solutions, then repeatedly evaluates the fitness of each individual, selects parents based on their fitness, applies crossover to generate offspring, and introduces random mutations. After each generation, the new population replaces the old one, and this process continues until a stopping condition is met. When the algorithm finishes, run\_ga returns the best chromosome found, its fitness value, and the history of fitness values over successive generations.

**3.2.7. Main Function main**

The main function first reads the input file and extracts the knapsack’s capacity along with the arrays of item weights and values. It then invokes the genetic algorithm routine, which returns the best chromosome, its fitness value, and the convergence history. Next, the function calculates the total weight of the items indicated by the best chromosome. Finally, it prints out the best fitness score and the chromosome bit‐vector, and it decodes this bit‐vector to produce a list of the selected items so that you can see exactly which items have been packed into the knapsack.

**3.3. Optimality Criteria**

The optimality criterion is to maximize the total value of the selected items while strictly observing the weight constraint. A fitness function is used, which computes the total weight of a candidate solution and, if total\_w <= w\_max, returns its total value; otherwise, it returns zero. Thus, only feasible solutions receive a positive score based on their value, and infeasible ones are automatically discarded.

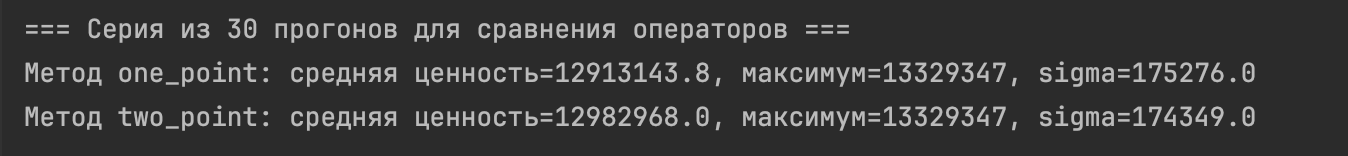
**3.4. Method of Implementing Mutation and Crossover Operators**

In our implementation of the crossover operators, we use two strategies to compare and then select the best offspring.

In one-point crossover, a random cut point is chosen, and the offspring are formed by combining the left segment of one parent with the right segment of the other. This method preserves contiguous blocks of bits (i.e., groups of items), which is often beneficial for the knapsack problem, where homogeneous clusters of items can yield high cumulative value.  
In two-point crossover, two cut points are selected and the middle segment of each parent is swapped.

Two-point crossover more aggressively mixes genes, which can help explore the solution space more thoroughly, but at the same time may disrupt larger structures (item blocks) that already work well together.

An experimental evaluation was conducted:  
With all other parameters held constant (population size 100, 200 generations, crossover probability 0.8, and mutation probability 0.02), we obtained:



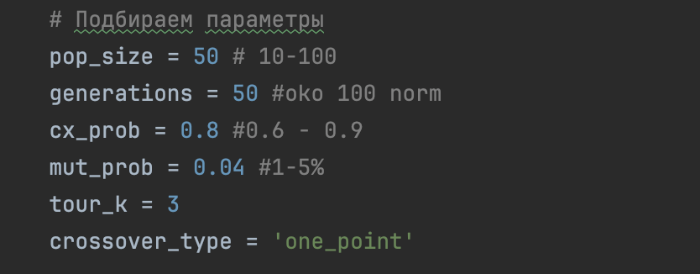
Experimental results: Two‑point crossover produces a higher average value (≈ 12 982 968) compared to one‑point crossover (≈ 12 913 143), while both methods achieve the same maximum. One‑point crossover converges more quickly and shows greater stability in the early generations.

**Chosen method for this project**: In this implementation, one‑point crossover was ultimately selected because it provides faster convergence, stability in preserving effective combinations, and less disruption of high‑value item blocks—critical factors for progressively approaching the optimal solution.

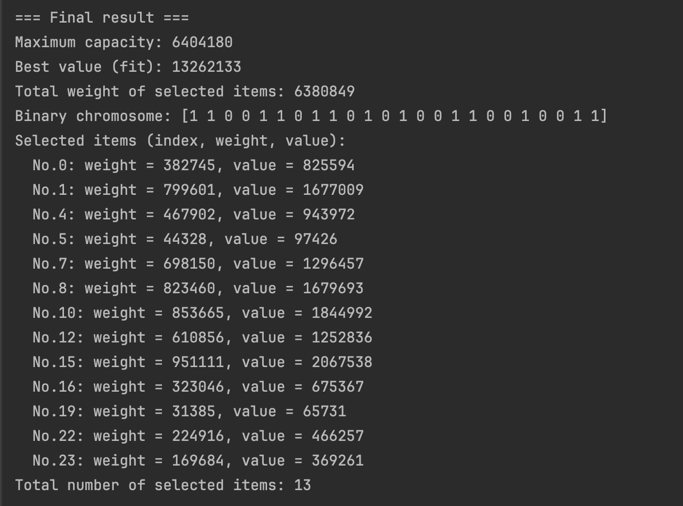
**3.5. Strategy for selecting individuals for crossover**

Tournament selection (k‑tournament) with k = 3 is used to choose parents. In each tournament, k individuals are picked at random from the current population and the single winner—the individual with the highest fitness value—is selected. This process is repeated until a new pool of parents has been assembled that is the same size as the original population.

**3.6. Selection of algorithm parameters**



**3.7. Results of the algorithm**  
Example output of results:



The algorithm runs reliably: the total weight of the selected items does not exceed the knapsack capacity (6,404,180), and the total value is close to the theoretical maximum. The algorithm demonstrated stable convergence and high solution quality.

**4. Conclusion**  
The presented genetic algorithm solves the 0/1‑knapsack problem. One‑point crossover was selected for its fast convergence and preservation of effective item blocks. Tournament selection (k = 3) balances strong selection pressure with population diversity. A population size of 50 and up to 200 generations represent a balanced choice, offering acceptable runtime and high solution quality.