Artificial Intelligence and Knowledge Engineering Laboratory

Assignment 1. Genetic Algorithms (version 2021)

Authors: P. Myszkowski, M. Przewoźniczek, M. Piasecki, A. Zawisza, W. Walentynowicz, P. Dolata

Deadline: 4th laboratory classes
Assignment Objectives

Getting familiar with genetic algorithms (GA) metaheuristics in practice through individual implementation.

Subtasks

- Get familiar with genetic algorithms metaheuristics
- Analyse the problems to solve (specified in the *Problem* section).
- Build a genetic algorithm. Define: an individual, a fitness function, a crossover operator, a mutation operator and a selection operator.
- Implement a genetic algorithm in any preferable object-oriented programming language.
- Test the influence of:
    - different parameters on genetic algorithm efficiency and effectiveness, including: probability of mutation and crossover, two different selection operators, population size and number of generations.
    - the problem sizes.
- Create a report in which you shortly explain all your decisions and provide analysis of the obtained results, your observations and findings.
- Provide visualisation of the best solutions found by the algorithm: in each population or the final ones.
- Create graphs presenting the changes of fitness function value during algorithm run and the populations (fitness value of best individual, population average, standard deviation or the worst individual).
- Compare your genetic algorithm to other, non-evolutionary optimization method (choose one: brute force, gradient method, non-gradient method, random search, hill climbing). Note the result quality, execution time, number of fitness function evaluations.
- Present and shortly discuss the most interesting results.
- The report should contain all of the above points.

Introductory information

Genetic algorithm is a method which imitates the natural evolution process through selection pressure and natural selection. In order to use GA, a potential solution (individual), the way it changes (mutation), mating (crossover) and fitness function must be defined. The work schema of a genetic algorithm might be presented as follows:

```
begin
  t:=0;
  initialise(pop0);
  evaluate(pop0);
  while (not stop_condition) do
  begin
    popt+1 := selection(popt);
    popt+1 := crossover(popt+1);
    popt+1 := mutation(popt+1);
    evaluate(popt+1);
    t:=t+1;
  end
end
```

At the beginning, the method initializes (usually at random) a population of solutions (individuals). Next, all individuals are evaluated (the fitness function value is computed for every individual). Then, the stop condition is checked (it might be based on the quality of the best found solution, computation time, number of fitness function evaluations etc.). The individuals for the next population are chosen (using roulette wheel method or tournament method) before crossover and mutation. Finally, all individuals are rated and the next iteration is started by check of stop criteria.

Problem – PCB autorouting

Introduction

Genetic algorithms are applied to design path layouts for printed circuit boards (PCBs)[1] – a problem known as *autorouting*. Your task will be to build a GA that solves a problem of finding an optimal layout for physical connections on a single-layer PCB, given:

- closed, connected subset of a plane, referred to as a *board* or a *PCB*,
- ordered set of soldering points,
- function that assigns each point from the set ***P*** its position on the board given by a pair of Cartesian coordinates,
- ordered set of planned structural connections ***S*** between points from the set ***P***, given by pairs of points that should be physically connected.

The task is to design a network of physical connections in such a way, that any two soldering points are connected if and only if there is a planned structural connection between them. Assume the following:

- the PCB is a single-layer rectangle of given dimensions,
- soldering points can only have integral coordinates,
- paths can only be laid on the board along a square grid of size 1,
- paths between two different points **MUST NOT** intersect (since intersecting physical paths connect all the points they comprise).

[1] For instance: https://en.wikipedia.org/wiki/Printed_circuit_board

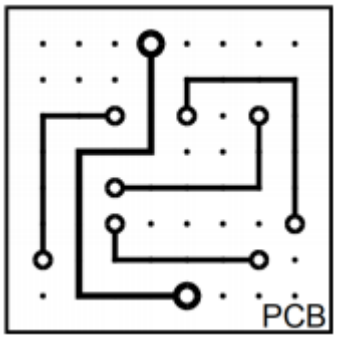

Figure 1. Example PCB design.

Coding
Each specimen shall represent a solution comprising all required physical connections (paths). Each such connection consists of a sequence of segments of variable length. These segments can be horizontal or vertical, of integral length. In order to read a path consisting of such segments, before coding you will need to denote which point of a pair is a starting point. A chromosome has a hierarchical structure, and representation of a path is variable length – you need to remember that when designing crossover and mutation operators!

Operators
The crossover operator has to account for the variable count of segments in a path. For this reason, designing a crossover operator working on the level of single paths would lead to problems with the proposed chromosome representation. We suggest basing the crossover operator on exchanging whole paths between the parents.
The mutation operator should operate on the path level – this means it should modify a path between two points. An example operator could modify the length of a segment, remove or create a new segment etc. However, it **should not** alter the orientation of a segment (horizontal-vertical). Mutation can also modify only a fragment of a segment.

Fitness function
The fitness function (or *objective function*) should minimize the overall length of all paths and their segment counts. Beyond that, it should penalize specimens that do not meet the constraints. Remember that in this task, the goal is to *minimize* the objective function – bear that in mind when implementing the roulette wheel selection operator.
Example feature set for the fitness function could contain:

- number of intersections,
- total path length,
- total segment count,
- number of segments that lay outside of the board,
- total length of the paths laying outside of the board.

Each such feature should be associated with a weight to denote the importance of each element in the fitness function (which should be a weighted sum of the features).

Implementation tips
Your solution will be a computer program. Consider including the following functionalities:

- data uploading (a problem will be given as a text file),
- evaluation of a solution in terms of a PCB design,
- GA parameters management,

- GA progress management,
- storing of the solution and genetic operators,
- population management (selection, initialization),
- progress logging (to external file).

Consider whether you need a shallow or deep copy for constructing the specimens. Select the best collection for storing the chromosome.

PCB instances

There are two sets of problems: test problems, which are fully described to help you check validity of your GA implementation, and research problems, which you should carry experiments on to prepare the report. You should generate outputs for each of these problems and include the relevant tables and graphs in your report.

You should annotate each result/graph by stating the configuration parameters used:

- population size,
- number of iterations,
- crossover probability,
- mutation probability,
- chosen operators (selection, crossover, mutation),
- any additional parameters relevant to your implementation.

We recommend presenting the results in a form of a table similar to the one below:

instance	Genetic algorithm [10x]				Other method [N]			
	best	worst	average	std	best	worst	average	Std
xyz2								
…								

Where:

- Genetic algorithm [10x] – denotes 10-fold evaluation of the GA (calculate the statistics over all the repetitions),
- Other method [N] – for a fair comparison with a reference method of your choice, run it for N iterations, where N is the number of solutions evaluated by the 10-fold evaluation of the GA,
- best – the best found solution,
- worst – the worst found solution,
- average – the average solution,
- std – standard deviation of the averaged solutions.

Caution: to evaluate the genetic algorithm, you should select the best specimen from each of the 10 repetitions, creating a set of 10 specimens. Then select the best and worst specimens and calculate the average and standard deviation *of those 10 best specimens*.

Task grading

Basic program and fitness function (max 2 points)

- implementation of the data loader and fitness function (1 point)
- implementation of the other optimization method (1 point)

Grading:
Your code will be subject to grading based on the validity of the implementation and clarity of the design (class structure). You will be expected to run both methods to show correct results. You can implement the data loader prior to the class.

GA operators (max 4 points)

- implementation of 2 selection operators – roulette wheel and tournament (2 pts)
- implementation of a chosen crossover operator (1 point)
- implementation of a chosen mutation operator (1 point)

Grading:
Validity of the operators implementation will be asserted by running the entire GA at submission time and checking whether it behaves according to expectations. For this reason we suggest preparing the skeleton of the GA prior to these classes. Alternatively, you can construct a suite of unit tests to prove correctness of the operators – in this case you can complete the GA implementation right after the classes.
If your code turns out impossible to check and verify its correctness (e.g. does not compile), you will only receive 50% of the points.

Experiments (max 4 points)

- examining the influence of parameters (1.25 points):
    - population size
    - iterations count
    - tournament size
- comparing the selection operators – roulette vs tournament (1 point)
- examining the influence of operators (1 point):
    - crossover probability
    - mutation probability
- comparing the genetic algorithm with a chosen method (0.75 points)

Grading:
You will receive points basing on the report you submit. Research should be carried according to the directions given in this instruction (GA repetition count, level of difficulty of the problem). The report should contain a summary of your research in the form of a table of results and graphs, as well as your observations and conclusions. Each piece of research should be named and have a statement of purpose (what is the objective of this experiment?). Do not forget about including the configuration parameters – a description of the experiment should allow reproducing it by the reader. Due to the time cost of running the GA, you will likely need to write the report after the classes. You should expect receiving questions about your results, implementation, theory etc.

References

1. Zbigniew Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs. Springer, 1996 (*Accessible from the WUS&T university network:* http://springerlink.com/).
2. S.N. Sivanandam, S.N. Deepa Introduction to Genetic Algorithms. Springer 2008 (*Accessible from the WUS&T university network:* http://springerlink.com/)
3. Man K.F., Tang K.S and Kwong S. Genetic Algorithms. Concepts and Designs Springer, 1999 (*Accessible from the WUS&T university network: http://link.springer.com/book/10.1007/978-1-4471-0577-0*)