# Exercise 13

## Task 1

Comprehension questions on Hill Climbing:

a) What are the advantages and disadvantages of randomized hill climbing in comparison to the gradient method in the optimization of a continuous function?

b) What are the strengths and weaknesses of the gradient method and of randomized hill climbing when searching for global and local optima?

c) What kind of problems occur when optimizing a function with the randomized hill climbing algorithm if you choose the newly proposed solution candidates from neighborhoods that are too small or too large?

d) How can you ensure whether you have found the global optimum with your hill climbing algorithm?

## Task 2

You are given a graph $G = (V, E)$. In the **vertex cover problem**, we are looking for a smallest possible set of vertices $U \subseteq V$ so that every edge of the graph is incident to a vertex from $U$. See also: `http://en.wikipedia.org/wiki/Vertex_cover`

a) Propose a heuristic for obtaining a good starting solution.

b) Define a possible encoding of the optimization problem for a genetic algorithm.

c) Specify possible modification steps, such as mutations and recombination. What kind of problems occur and how can you deal with them?

## Task 3

Consider the following **assignment problem**: You are given $n$ workers, who have to perform $n$ jobs. Each worker is to be assigned exactly one job, in such a way that the total cost is minimized. The costs for job $i$, performed by worker $j$, are given by the entries $c_{ij}$ in a cost matrix.

a) Define an encoding of the optimization problem suitable for application in a genetic algorithm.

b) Specify possible modification steps, such as mutations and recombination. What kind of problems occur and how can you deal with them?

c) Specify possible neighborhoods which you could use e.g. in the hill climbing method.

## Task 4

In the **knapsack problem** you have the task of packing valuable items without exceeding a specified total weight $G$. The weights of the $n$ objects are given by $w_i$ and the values by $v_i$, $i = 1, \ldots, n$. Your goal is to maximize the total value of the packed items.

a) Devise a heuristic strategy to solve the problem. Does your strategy always give an optimal solution? Why (not)?

b) Propose an encoding of the optimization problem for application in a genetic algorithm.

c) What are possible operations for mutations and crossover in the algorithm?

## Task 5

In the **exact cover problem**, $n$ subsets $U_i$ of a base set $M$ are given and you have to determine whether there is a choice of subsets $U_i$ such that each element of $M$ is contained in exactly one of the chosen sets. (I.e, the union of the chosen subsets should be $M$, and no element should occur in two or more of the chosen sets.)

The optimization variant of this problem is as follows: Find a choice of subsets $U_i$ such that each element of $M$ is contained in one of the chosen sets, and the sum of the cardinalities of the selected subsets is as small as possible.

a) Devise a heuristic strategy to find an approximate solution.

b) Give an encoding of the optimization problem for application in a genetic algorithm, a suitable fitness function and possible operations for mutations and crossover.

## Task 6 (*optional)

Write a program which simulates a population of size $p$ consisting of individuals that are chromosomes (bitstrings) of length $n$. In each iteration, cross two randomly selected individuals by selecting each bit randomly from one of the parents, and then mutate each bit with probability $0.01$. Finally, replace the weakest individual in the population by the newly generated one, where the fitness of an individual is simply given by the number of ones its representation contains.

Simulate a number of evolution cycles with different population sizes (e.g. $p = 5, 50, 150, 500$) and different chromosome sizes (e.g. $n = 100, 500$). What do you observe?

## Task 7 (*optional)

Write a simulation program that displays 20 random pairs of coordinates in an $x$-$y$ diagram and then finds an approximation of the shortest salesman tour with a genetic algorithm.

# Solutions to Exercise 13

## Solution to Task 1

a) Hill climbing is simple to implement, as there is no need to calculate the gradient or the step size and therefore it does not require much computation time to propose a new solution candidate. The disadvantage of hill climbing is that a random point is selected from the neighborhood and the acceptance probability of the proposed solution can become very small. In the gradient method a good direction is chosen automatically and then the step size is adapted. This means information about the shape of the function is used, so that the algorithm does not scrape around in the search space, but rather the function is minimized in a targeted way.

b) The gradient method is geared towards successively improving an existing approximate solution and will therefore move into the nearest local optimum. The gradient method is not designed for searching for global optima. With the hill climbing method, local and global optima can be searched for by the neighborhoods from which a random solution is proposed being defined to be correspondingly narrow or wide.

c) If the proposed solution candidates are taken from too narrow a neighborhood, then hill climbing fails to escape from local optima. An additional problem is too slow convergence, since the step sizes are too short. If the neighborhoods chosen are too large, the convergence into local and global optima is difficult, because the local space around an optimum is not accurately searched enough, since new solution candidates are proposed from too distant neighborhoods instead.

d) One could run hill climbing from many different starting point in order to verify whether hill climbing converges to the same solution in all cases.

## Solutions to Task 2

a) Successively choose a vertex which contains the most edges not yet connected, and include it with the set $U$. Continue like this until all edges are covered.

b) One possible encoding would be a binary vector, where each element stands for a vertex. The entry 1 means that the vertex is selected and the entry 0 means that the vertex is not selected.

   A suitable fitness function would be the number of zeros in the vector. The more zeros are included in the vector, the fewer vertices are used in order to cover all edges. However, this only makes sense for individuals which represent valid solution candidates.

   A fitness function which allows invalid solution candidates would need to have a corresponding penalty term: e.g. the number of zeros minus $c$ times the number of edges not covered, for some constant $c > 0$.

c) We can use the standard mutations and recombination procedures here, i.e. one-/two-point or uniform crossover and point or position mutations.

However, most of these operations will lead to invalid solutions quite often. We could either (1) keep invalid solutions but penalize them in our fitness function (see b)), or (2) we could first repair and then intensify them. That is, we first add vertices until all edges are covered (e.g. by the heuristic suggested in a)), and then try to leave out superfluous vertices. (A third possibility for dealing with invalid solutions would be to simply throw all of them away and generate enough offspring in each generation to make sure there are enough valid solutions to continue. However, usually the other two approaches work better.)

Note that point mutations will not be very useful in a genetic algorithm that only allows valid solutions, because a good solution will almost always turn into an invalid one or a worse one. Also the other operations will quite often produce worse or invalid solutions. As outlined above, these issues can be dealt with appropriate repair and intensification procedures.

## Solutions to Task 3

a) A less effective encoding would be a binary $n \times n$-matrix, where a 1 in row $i$ and column $j$ means that job $i$ is performed by worker $j$. The matrix could then be assembled row by row to form a vector.

A better type of encoding would be a vector of length $n$, with each entry corresponding to one worker. Each job is also given a number between 1 and $n$, which is written into the vector at the position corresponding to the worker who will perform the job. A valid vector therefore contains each number exactly once, i.e. can be viewed as a permutation (as in the TSP example discussed in the lecture).

b) A suitable mutation is the position mutation. It corresponds to two workers switching jobs and will never create invalid solutions.

In the case of (e.g. two-point) crossover, invalid solutions will arise because typically the two children will no longer be permutations, i.e. they contain unassigned jobs and jobs assigned to two workers. A possible repair mechanism would be to assign jobs of the second kind to the worker which has lower cost for the job in question, freeing up the other worker. The unassigned jobs can then be assigned to the free workers, e.g. in increasing order of cost or by some other heuristic. As an intensification strategy, one could then possibly try to save costs by swapping pairs of jobs.

c) A neighborhood could e.g. be defined as all solutions obtained by permuting 2, 3 or more jobs. Permuting two jobs corresponds to a position mutation. Permuting three jobs can be seen as two position mutations which coincide in one of the positions.

## Solutions to Task 4

a) A reasonable greedy heuristic is to sort the items according to the ratio of value to weight and successively pack the item with the largest possible value/weight ratio until no more items can be added. The procedure is not guaranteed to be optimal, because it could be the case that only very heavy and valuable items are left over, one of which could have been added if a very small item with slightly higher value/weight ratio (but lower actual value) had been left out.

b) A suitable encoding would be a binary vector, where a 1 entry means that the item is packed and a 0 entry means that the item is not packed. (In view of the above heuristic, it might be useful to sort the vector positions according to the value/weight ratio of the corresponding items.)

A suitable fitness function for a genetic algorithm in which only valid solution are allowed is the total value of the packed items. Otherwise, a penalty term would have to be added, which penalizes the maximum weight being exceeded. For example: Fitness is equal to total value of the packed items minus a factor c times excess weight.

c) A suitable mutation would be a position mutation at two points in the genome with different values. That is, the inclusion of one item is replaced by the inclusion of another item.

In the case of a recombination, e.g. a one-point, two-point or uniform crossover, a portion of the items from one parent is replaced by a portion of the items from the other parent. Typically this will create one valid and one invalid solution as offspring. We could either use a repair and intensify approach similar to Task 2 (i.e. drop excess items and greedily add items if there is space left), or just accept these offspring as they are and penalize restriction-violating ones as outlined in b).

If we allow invalid solution candidates, the weighting factor c of the penalty term should be dynamically adjusted. If there are too many invalid candidates in the population, c should be increased, if there are very few, we might lower it. In that spirit, we should start out with a moderate value of $c$ and increase it towards the end of the algorithm in order to obtain good solutions which satisfy and exhaust the weight constraint.

## Solutions to Task 5

a) Successively select the subset $U_i$ which has the smallest intersection of elements with the elements selected so far. Sets with greater cardinality should be preferred, because this means the sets of lower cardinality remain to *fill in the gaps* later in the process.

b) A suitable encoding would be a binary vector, where each element stands for a subset $U_i$. The entry 1 means the corresponding subset is selected and the entry 0 means that the subset is not selected. With regard to the heuristic strategy one could represent the sets in the vector ranked according to their cardinality .

A suitable fitness function in the case of only valid solution candidates would be the negative sum of the cardinalities of the selected sets.

A point mutation then almost always leads to a solution that is either worse or invalid, since a subset with elements is removed or added. Similarly, a non-trivial position mutation (i.e. one between positions with values 0 and 1) frequently leads to invalid solutions. Also most non-trivial recombinations will likely lead to invalid solutions, since e.g. in the case of a one or two-point crossover, some sets with elements will be removed and other sets with other elements will be added. In order to obtain good and valid solutions again, one would have to first repair the invalid solutions and then intensify them. That is, first add sets until a valid solution emerges and then try to remove redundant sets.

An additional intensification strategy to improve valid solution would be the following: Try to remove one set at a time and to replace it by a better one. One could also remove two or more sets and then use the heuristic from a) to *fill up* again until all elements are covered.

## Solutions to Task 6

See R code:

- The fitness of the best individual is monotone increasing.
- A population size that is too small or too big is bad:
  *Too small population without mutations:* Positions in the genome in which a 0 occurs in all individuals of the population are preserved forever!
  *Too large population, without mutations:* It takes a very long time for improvements to be established, since at most one individual is improved in each generation. That is, a too large gene pool slows down convergence.
- Mutations can either slow down, accelerate or prevent convergence, or even enable it, depending on the population size. In a small population in which each individual has a 0 at a particular position in the genome, the value at this position can only mutate into an advantageous 1 by a mutation. Mutations therefore help to exhaust the entire solution space. On the other hand, mutations can prevent the optimal genome, with only 1s, being reached. With a high mutation rate, good genomes with very many 1s at the positions in the genome are mutated into poorer 0s again at many positions.

## Solutions to Task 7

See R code.