Local search

1 WORK DURING THE LAB

1. Implement the algorithm **Random Hill-Climbing** for the *knapsack problem*.
2. Test the algorithm for different parameter settings.

Points for the work during the lab: **25p**

2 ASSIGNMENT A2

1. Implement one of the following two algorithms to solve the *knapsack problem*.
   a. **Steepest Ascent Hill-Climbing**
   b. **Next Ascent Hill-Climbing**
2. Perform experiments for knapsack instances of size 20 and 200.

Deadline to submit A2: **Lab 3**

Points for A2: **25p**

3 REQUIREMENTS

1. Source code (notebook) needs to be documented.
2. Algorithms have to be tested for several parameter values (sufficient to clearly determine performance).
3. Experiments must be performed for all available problem instances and results compared for diferent parameter settings.
4. Results of the experiments need to be saved in output files, indicating solution quality, parameter values used, number of runs.
5. A report should capture the following: problem definition, algorithm used (name, steps/pseudocode), parameter setting, comparative results of experiments, discussion of results.

4 RANDOM HILL-CLIMBING (RHC)

1. Choose a string at random. Call this string *best–evaluated.*
2. Choose a locus at random to flip. If the flip leads to an equal or higher fitness, then set *best–evaluated* to the resulting string.
3. Go to step 2 until an optimum string has been found or until a maximum number of evaluations have been performed.
4. Return the current value of *best–evaluated*.

5 STEEPEST ASCENT HILL-CLIMBING (SAHC)

1. Choose a string at random. Call this string *current–hilltop*.
2. Going from left to right, systematically flip each bit in the string, one at a time, recording the fitnesses of the resulting one–bit mutants.
3. If any of the resulting one–bit mutants give a fitness increase, then set *current–hilltop* to the one–bit mutant giving the highest fitness increase. (Ties are decided at random.)
4. If there is no fitness increase, then save *current–hilltop* and go to step 1. Otherwise, go to step 2 with the new *current–hilltop*.
5. When a set number of function evaluations has been performed (here, each bit flip in step 2 is followed by a function evaluation), return the highest hilltop that was found.

6 NEXT ASCENT HILL-CLIMBING (NAHC)

1. Choose a string at random. Call this string *current–hilltop*.
2. For i from 1 to l (where l is the length of the string), flip bit i; if this results in a fitness increase, keep the new string, otherwise flip bit i; back. As soon as a fitness increase is found, set *current–hilltop* to that increased–fitness string without evaluating any more bit flips of the original string. Go to step 2 with the new *current–hilltop*, but continue mutating the new string starting immediately after the bit position at which the previous fitness increase was found.
3. If no increases in fitness were found, save *current–hilltop* and go to step 1.
4. When a set number of function evaluations has been performed, return the highest hilltop that was found.

***HC algorithms from: *M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1999.*