Exercise: Simple Stochastic Algorithms for the Knapsack Problem

Introduction to Optimization lecture
at Ecole Centrale Paris / ESSEC Business School

Dimo Brockhoff
firstname.lastname@inria.fr

November 21, 2016

Abstract

In this exercise, we will implement two simple (randomized) search heuristics for the 0-1-knapsack problem in order to get acquainted to stochastic algorithms and to show how easy their application can be.

1 Implementing Simple Stochastic Search Algorithms

In the lecture, we have seen a few basic stochastic search algorithms of which we will implement the randomized local search (RLS) and the so-called (1+1)-EA here. While RLS boils down to a local search which randomly selects from the 1-bit-flip neighborhood ("first improvement"), the (1+1)-EA is using a standard bit-flip operator which flips each bit with probability of $1/n$ with n being the number of bits in each solution. In order to keep the implementation simple, please follow the instructions below.

a) Implement a basic 1-bit flip mutation operator which takes a solution and flips a single bit, uniformly chosen at random, in each solution. Assume thereby that a solution is coded as an array of bits (i.e., each bit can be either 0 or 1, false or true, ..., best use a `numpy.array`).

b) Use the implemented 1-bit flip operator to code the most basic randomized local search (RLS). It starts with a randomly sampled search point, always keeps a single solution with the best-so-far function value, and replaces it by a randomly chosen neighbor of the 1-bit-flip neighborhood if and only if this is better than the current search point. Note that, because the knapsack problem is a constrained problem, we have to deal with potentially infeasible solutions. The possibly easiest way to do this here is to implement a "repair" method which takes any infeasible solution and makes it feasible by removing items until the item selection fits again into the knapsack (the items are thereby removed greedily in the order of their profit/weight ratios).

c) Do you think that relying solely on 1-bit flips is a reasonable operator for the knapsack problem? What do you expect will happen in particular in the end of the optimization when you only use 1-bit flips? What do you suggest to circumvent this behavior?

d) Implement now the standard bit-flip operator which takes a solution and flips each bit with probability $1/n$ with n being the length of the solution's bitstring.

e) Similar to the randomized local search from above, implement the most basic evolutionary algorithm, the so-called (1+1)-EA which always keeps the best-so-far solution and replaces it by a newly sampled solution (with the standard bitflip operator) if and only if it is better than the current search point. Again, better here is meant after a potential repair step to make the new solution feasible.

f) Test your two algorithms now for some of the knapsack instances from `http://researchers.lille.inria.fr/~brockhof/optimizationSaclay/knapsackinstances/` by using the code of the previous exercise(s) for evaluating the objective and constraint functions. In particular, compare the number of function evaluations to reach the optimal objective function value. What do you observe? Can you explain the difference between the two algorithms?