

Metaheuristics

Stefano Benedettini

Andrea Roli

Artificial Intelligence LM 2010

Contents

1	Supplementary Material	1
2	Metaheuristics Warm-up	2
2.1	01-Knapsack	2
2.2	SLS for Knapsack	2
3	First Assignment	2
3.1	Longest Common Subsequence Problem	3
3.2	Error Correcting Code Design Problem	3
3.3	Low Autocorrelation Binary Sequence Problem	3

Topics

- Stochastic Local Search (SLS)
- Application to Knapsack
- **First assignment**

1 Supplementary Material

Attached to the exercise archive you will find the following **Python** modules:

localsearch: contains several basic implementations of local search techniques

knapsack: contains a class to generate Knapsack instances and utility functions

lcs: useful code to parse a Longest Common Subsequence instance

2 Metaheuristics Warm-up

2.1 01-Knapsack

The Knapsack problem is a well studied problem in OR. Given a set of objects $I = 1, \dots, n$ with weights w_i and profits c_i and a knapsack with constant capacity C , find a subset of items $\mathcal{I} \subseteq I$ that fits into the knapsack and maximizes the profit. Formally:

$$\begin{aligned} & \max \sum_{k \in \mathcal{I}} c_k \\ \text{s.t. } & \sum_{k \in \mathcal{I}} w_k \leq C \end{aligned}$$

2.2 SLS for Knapsack

Knapsack is best solved by exact techniques, for instance Branch-and-Bound, nevertheless is a fairly easy playground for trying stochastic local search techniques.

The objective here is to implement simple SLS algorithms for the Knapsack, experiment with various techniques and test different variants. Using the code provided by `localsearch` module, try to instantiate the following local searches:

1. First improvement
2. Best Improvement
3. One of: Variable Neighborhood Search or Variable Neighborhood Descent
4. Tabu Search

Also experiment with different neighborhood definition.

3 First Assignment

For the first assignment you are required to design a family of SLS algorithm for *one* of the following problems. You can work in groups of no more than 3 students.

Each student or group, let's call them working unit, has to deliver a SLS algorithm written in `Python` language. The algorithm should be parametrized in the sense that it should be possible for the user to tune its parameters in order to accommodate for different requirements or problem instances. For example, let's consider Simulated Annealing: the coefficient in the cooling schedule is an algorithm parameter that greatly alters algorithm behaviour. Another example is the tabu tenure for a Tabu Search. It is also possible to tune neighborhood size: a larger neighborhood is computationally expensive, but should lead to better solutions.

Mind that the objective *is not* to mimic an algorithm proposed in literature *nor* improve on the state of the art!

For all algorithm the user should also be able to provide a maximum run time.
For each problem, test instances and reference material are provided.

3.1 Longest Common Subsequence Problem

LCS is a classical string problem. Given an alphabet Σ and a set of string $\mathcal{S} = \{s_1, s_2, \dots, s_n\}$ (lengths can be different), find the longest string t such that t is *subsequence* of each s_i . A string s' is said to be a subsequence of a string s if s' can be obtained by erasing characters from s . For example, let $\Sigma = \{a, b, c, d\}$ and $s = aabccdbad$ then ε (the empty string), $acdb$, $aaad$, cdd and s itself are subsequences while $aaaa$, $cbab$ or $abcdc$ are not.

`lcs` module implements a parser and can be the starting point for an implementation. 80 random instances are provided. Since they are very large ones and `Python` is not that fast, i suggest you to trim them down. `Lcs.parse_file` function allows you to decide a maximum string length, which should be anywhere between 50 and 100 but no more.

Attached papers are very useful as they provide insights and many hints like greedy searches, upper and lower bounds.

3.2 Error Correcting Code Design Problem

ECC goal is to find a m binary strings, called code words, of length n such that the minimum Hamming distance d_H between the codewords is maximal. A solution is a $m \times n$ matrix C and the objective is to maximize:

$$d_{\min}(C) = \min\{d_H(c_i, c_j) \mid i \neq j\}$$

An instance is just a couple of integers (m, n) . Solve at least the following instances:

1. (24, 12) which has optimum 6
2. (32, 16) which has optimum 8
3. (40, 20) which has optimum 10

You can invent instances yourself, but remember to avoid (m, n) which correspond to Hamming codes.

3.3 Low Autocorrelation Binary Sequence Problem

LABS problem aims at finding a n -length sequence $s_0, s_1, \dots, s_n, s_i \in \{-1, 1\}$ in order to minimize the energy E defined as:

$$C_k = \sum_{i=1}^{n-k} s_i s_{i+k}$$

$$E = \sum_{k=1}^{n-1} C_k^2$$

In the definition of the C_k 's you have to consider every pair of elements that are $k - 1$ positions apart. For instance, if $k = 1$ consider all adjacent elements $s_1s_2, s_2s_3, \dots, s_{n-1}s_n$, if $k = 2$ consider the sequence $s_1s_3, s_2s_4, \dots, s_{n-2}s_n$, up to $k = n - 1$ where you only consider s_1s_n .

This problem is the fifth problem in CSPLib. Definition, results in literature and bibliography can be found here. On these site you can find tables that report optimum (or very good) energy values for different n : http://www.comp.nus.edu.sg/~stevenha/viz/results_labs.html.