Lab 2

Dr. Donald Davendra
CS471 - Optimization

October 12, 2016

1 Blind Algorithm - Random Walk

The first attempts to search for optimal values of the *utility* function within the group of enumerative algorithms, whose aim was to calculate the objective value in all possible points and then comparing the obtained value for the *minimum* or *maximum*.

In essence, this means generating a user-defined number of *random points* on the search space and define a function which then selects an individual with a minimum value. This algorithm has no other ties to the development of individuals towards global extreme that is found on the quality of previous values.

input : $iterations$: maximum number of iterations,
arg_{best}: the best solution found
$fitness_0$: initial setting to a large value
output: arg_{best}: the best solution found

```
1 for i ← 0 to iterations do
2 |   arg = rand_i
3 |   /* where rand_i is a random number between the defined interval     */
4 |   fitness = F_cost(arg)
5 |   if fitness < fitness_0 then
6 |   |   fitness_0 = fitness
7 |   |   arg_best = arg
8 |   end
9 end
```

Algorithm 1: Blind Search

2 Local Search

The method is based on the *initial solution* under consideration (usually randomly generated), which generates a set of *neighbouring* solutions. In order to find the best solution among those neighbours, if the found one is better than the default solution, we take it as

new current best solution and we again generate neighbours to it and look amongst them for "improved" solutions. This procedure is repeated until we decide that between neighbouring solutions there are no longer any improving solutions.

Stochasticity of this procedure is just a random selection of initial solutions, the following optimisation algorithm used is strictly deterministic. The method of local search (LS) is a (stochastic) algorithm with the following parameters:

$$LS = (M, x_0, N, f)$$

where:

- M is the solution space
- x_0 is the initial solution. If x_0 is determined randomly, then the method is stochastic local search algorithm.
- $\sigma \subseteq M \cdot M$ is a binary relation on M defining neighbouring solutions.
- $N(x, \sigma) = \{y \in M | (y, x) \in \sigma\}$ is the set of solutions which are neighbour of $x, x \in M$
- f is the objective function, the optimum seeking $f : M \to R$

The following pseudocode describes the local search process. Boolean variable τ serves as the termination criterion, which is `true` if a better solution is found in the neighborhood and `false` otherwise. x_0 is the initial (pseudo-random) solution.

```
   input  : x_0: initial random solution
            x*: best solution found
            τ: boolean variable
   output: x*: best solution found
 1 x* = x        /* initialize best solution to the random solution        */
 2 τ = true      /* initialize τ to true                                    */
 4 while τ = true do
 5 |   τ = false
 6 |   generate N(x*, σ)
 7 |   find x ∈ N(x*, σ) so that f(x_loc) ≤ f(x), for each x ∈ N(x*, σ))
 8 |   if f(x) < f(x*) then
 9 |   |   x* = x_loc;
10 |   |   τ = true;
11 |   end
12 end
```

Algorithm 2: Local Search

3 Iterative Local Search

There are several ways to reduce the likelihood of deadlock in a local optimum, namely:

- enlarge the set of neighbours.
- repeat the method of local search for several different (pseudo-randomly generated) initial solutions and record the best solution.
- to admit even steps, after which there is a deterioration in the value of objective function, thereby allowing the "turn" to another area of space solution.

All these methods offer hope of obtaining a better solution, but require longer calculation time. The second method, known as the repeated local search can be formally written as follows:

$$RLS = (M, x_0, N, f, t_{max})$$

where:

- M is the solution space
- x_0 is the initial solution. If x_0 is determined randomly, then the method is stochastic local search algorithm.
- $\sigma \subseteq M \cdot M$ is a binary relation on M defining neighbouring solutions.
- $N(x, \sigma) = \{y \in M | (y, x) \in \sigma\}$ is the set of solutions which are neighbour of $x, x \in M$
- f is the objective function, the optimum seeking $f : M \to R$
- t_{max} parameter specifies the chosen number of iterations the algorithm performs a local search and criteria for termination.

The following pseudocode describes the repeated local search process. Boolean variable τ serves as the termination criterion, which is `true` if a better solution is found in the neighborhood and `false` otherwise. x_0 is the initial (pseudo-random) solution, and t_{max} is the maximum number of iterations.

```
input  : x_0t: initial random solution
         x*: best global solution
         x*_t: best global solution in each iteration
         τ: boolean variable
         t_max: maximum number of iterations
output: x*: best solution found
1  x*_t = x_0t        /* initialize iterative best solution                  */
2  x* = x_0t          /* initialize global best solution                     */
3  τ = true           /* initialize τ to true                                */
4  t = 1              /* initialize t to 1                                   */
6  while t ≤ t_max do
8      while τ = true do
9          τ = false
10         generate N(x*, σ)
11         find x ∈ N(x*, σ) so that f(x_loc) ≤ f(x), for each x ∈ N(x*, σ))
12         if f(x_loc) < f(x*_t) then
13             x*_t = x_loc;
14             τ = true;
15         end
16     end
17     if f(x*) < f(x*_t) then
18         x* = x*_t;
19                 /* update the best solution after t interactions       */
20     end
21     t = t + 1      /* update the t counter                              */
22     x*_t = x_0t    /* (randomly) choose a new initial solution x_0t     */
23 end
```

Algorithm 3: Repeated Local Search

4 Experimentation

The student is required to modify the code from Lab 1 and add the three described algorithms. For each algorithm, 30 iterations for each problem is required for 10, 20 and 30 dimensions. Compute statistical analysis on the obtained results for average, standard deviation, range, median and time.

Submission

The student must submit the following separate files to canvas:

1. source codes for the problems
2. a LaTeX typeset report on the results and its analysis

The report must contain an introduction in the algorithms, the full experimentation results in tabular format and condensed results with statistical analysis compared with what was obtained in Lab 1.

The files must be submitted through Canvas by midnight October 24, 2016. The penalty for late submission is 10% for 1 day, 20% for 2 day, after which it will be zero. The grading rubric is given in Table 1.

Table 1: Grading rubric

File	Aspects	Points
`Code`	Compiles and executes	35
	Explanation	15
`Report`	Results	25
	Analysis	25