Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Summer Term 2017

Project 1 – "Nature-Inspired Algorithms"

https://hpi.de/en/friedrich/teaching/ss17/natinsalg.html

In this project we will look at some toy problems and test the run time of some simple nature-inspired search heuristics on these problems. We will look at optimization problems which are defined on *bit strings* (chains of 1s and 0s). One parameter is n, the length of the bit strings, which does not change during the optimization process. We write the search space (bit strings of length n) as $\{0,1\}^n$. For any $x \in \{0,1\}^n$ we write the number of 1s in x as $|x|$. Here are four functions, which assign a value to any bit string.

$$\begin{aligned}
\textsc{OneMax} : \{0,1\}^n \to \mathbb{R}, x &\mapsto |x|; \\
\textsc{LeadingOnes} : \{0,1\}^n \to \mathbb{R}, x &\mapsto \text{number of 1s before the first 0 in } x; \\
\forall k < n : \textsc{Jump}_k : \{0,1\}^n \to \mathbb{R}, x &\mapsto \begin{cases} |x|, & \text{if } |x| < n-k; \\ n-k, & \text{if } n-k \leq |x| < n; \\ n, & \text{if } |x| = n; \end{cases} \\
\textsc{BinVal} : \{0,1\}^n \to \mathbb{R}, x &\mapsto x \text{ interpreted as binary number.}
\end{aligned}$$

These problems are frequently considered on which one can test search heuristics. We add a fifth function, for $k < n$, $\textsc{RoyalRoads}_k$, which is defined on all n divisible by k. We divide the bit positions in n/k subsequences of length k. $\textsc{RoyalRoads}_k(x)$ is the number of subsequences of x, in which all k bits are set to 1 (so $\textsc{RoyalRoads}_k(x)$ is a number between 0 and n/k). For each of the functions, the goal is to find a bit string with maximal value.

We will consider three different "evolutionary" algorithms. All three algorithms have in common that they start with a random solution (a bit string in which each bit is randomly 1 or 0); in each iteration they create a new population of solutions out of which a parent could be selected for the next generation. For the first two algorithms, this population will only consist of a single solution.

The first approach is called **RLS** (Random Local Search), a simple algorithm to improving a solution: in each iteration we generate a new solution by flipping exactly

one bit in the old solution. The following gives pseudo-code for RLS.

Algorithm 1: RLS

```
Choose x uniformly at random from {0,1}^n;
while stopping criterion not met do
    y ← x;
    Choose i uniformly at random from {1,...,n};
    y_i ← (1 − y_i);
    if f(y) ≥ f(x) then x ← y;
```

For now, we will stop when the optimum is found (for all functions given above we know the value of the optimum beforehand).

Now we get to an evolutionary algorithm (**EA**), the so-called (1+1)-EA (it maintains one solution and compares it with one new solution, thus 1+1). It works almost like RLS, but instead of flipping exactly one bit, it flips each bit with probability $1/n$. Thus, in expectation, it flips exactly one bit, but it might be more (or less). This operation we call *mutation*. The pseudo-code is as follows.

Algorithm 2: (1+1)-EA

```
Choose x uniformly at random from {0,1}^n;
while stopping criterion not met do
    y ← x;
    foreach i ∈ {1,...,n} do
        With probability 1/n, y_i ← (1 − y_i);
    if f(y) ≥ f(x) then x ← y;
```

The third algorithm selects a new parent for subsequent generations out of a population of offspring. This is the $(1+\lambda)$ EA, in which we decide on a population size $\lambda \geq 1$ and replace the current search point by uniformly at random choosing the best (fittest) element of the set composed of the union of the offspring population

and the current point.

Algorithm 3: (1+λ)-EA

```
1 Choose x uniformly at random from {0,1}^n;
2 while stopping criterion not met do
3     P ← ∅;
4     foreach j ∈ {1,...,λ} do
5         P[j] ← x;
6         foreach i ∈ {1,...,n} do
7             With probability 1/n, P[j]_i ← (1 − P[j]_i);
8     y ← arg max_{z∈P∪{x}} f(z)// breaking any ties at random
9     if f(y) ≥ f(x) then x ← y;
```

Assignment. Implement all three algorithms, and for each algorithm a variant where $\geq$ is replaced by $>$ (a small but significant change). Also implement all five test functions. Evaluate all six algorithms on all five test functions as follows. We use $k = 3$ for JUMP and $k = 5$ for ROYALROADS. For any given n you let the algorithm run 10 ten times and determine the number of iterations until reaching the optimum. From these 10 runs you compute the average. This you do for $n = 25$ and then in steps of size 25 up until your computer does not want to go any further. For the (1+λ) EA, experiment with different population sizes such as 2, 10, and 50. Does using a population help on any of the functions?

The results of the project are submitted via Moodle:

`https://hpi.de/friedrich/moodle`

Each team submits two files. The below example assumes a team with three members where `LastnameX` denotes the last (family) name of a team member.

(a) A zip file named `LastnameA-LastnameB-LastnameC.zip` that contains your code;

(b) A pdf file, named `LastnameA-LastnameB-LastnameC.pdf` in which you show a graph for each of the algorithm variant you plot the run time on all the five test functions in dependence on n (experimentally determined as detailed above). For each plot you write what the asymptotic run time seems to be (using O-notation in dependence on n). Write a text of about half a page in which you discuss your findings.