INF 421 PROJECT

Evolutionary Algorithms
Professeur Martin Krejca

26 décembre 2023

Gabriel Pereira de Carvalho

INTRODUCTION

This report presents a solution to the programming project **Evolutionary Algorithms** for the course INF421 : Design and Analysis of Algorithms at École Polytechnique. Each task is developed in a section of the report which also contains the code implemented using the Nim programming language.

TASK 1

Write code that allows to use individuals as well as the three functions `OneMax`, `LeadingOnes`, and `Jump_k` . For individuals, do not use libraries but implement a data type that fully utilizes the memory. That is, do not store each bit value of an individual in a byte but in an actual bit.

Individual data type

An individual refers to a potential solution $x = (x_1, ..., x_n) \in \{0,1\}^n$. We wish to implement a data type that stores each $x_i \in \{0,1\}$ using a single bit. Since the smallest addressable unit of memory is a byte, this means that a basic type such as `bool` cannot be used to represent x_i, because a `bool` would take up 1 byte of memory.

The `C++` standard library provides a data structure that compactly stores bits, the `bitset` which is analog to the data type we wish to implement. We take inspiration on the `C++ bitset` in order to implement our own `Individual` class. Let's discuss the member functions we wish to implement :

- a method `int get(int idx)` which returns the value of the bit at the index `idx`.
- a method `void set(int idx)` which updates the value of the bit at the index `idx` to 1. If the value is already 1, no change occurs.
- a method `void reset(int idx)` which updates the value of the bit at the index `idx` to 0. If the value is already 0, no change occurs.
- a method `void flip(int idx)` which flips the value of the bit at the index `idx`.
- a method `int count()` which counts the number of bits equal to 1.

Code (implemented in Nim)

• Individual.nim

```
TO COME
```

• BenchmarkFunctions.nim

```
import Individual

# Benchmark functions

proc OneMax(individual: Individual): int =
    #[ returns the number of 1s of the input ]#
    return count(individual)

proc LeadingOnes(individual: Individual): int =
    #[ returns the length of the longest consecutive prefix of 1s ]#
    let n = individual.size
    result = 0
    for i in countup(1, n):
        var prefixProduct = 1
        for j in countup(1, i):
            prefixProduct = prefixProduct*get(individual, j)
        result += prefixProduct
    return result

const k = 50 #we set k value as a constant
proc JumpK(individual: Individual): int =
    #[ analog to OneMax but penalises individuals with number of ones in n-k
   +1,...,n-1 ]#
    let n = individual.size
    let OneMax_x = OneMax(individual)
    if OneMax_x <= n - k or OneMax_x == n:
        return k + OneMax_x
    return n - OneMax_x
```

TASK 2

TASK 3

TASK 4

TASK 5