Evolutionary Algorithms - Project Report

EL KHADIR Bachir
Ecole Polytechnique

May 1, 2014

Abstract

This paper presents the result ... I choose *Python* as a programming langage.

1 Implementation of the GA algorithm

The EA class implmenets the $(1+(\lambda,\lambda))$ GA in a generic form. The constructor is:

```
def __init__(self, fitness, mutation=None, crossover=None)
```

It requires:

- a fitness function to evaluate individuals,
- a mutation operator, by default, this operator is used:

```
def default_mutation(l, x):
        bits_to_change         = random.sample(range(len(x)), l)
        x_mut                                          = list(x)
        for b in bits_to_change:
                        x_mut[b] = random.choice([0, 1])
        return x_mut
```

- a crossover operator

```
def default_crossover(c, x, xx):
                parent                                  = [x, xx]
                parent_choice = bernoulli.rvs(c, size=len(x))
                return [ parent[p][i] for i, p in enumerate(parent_choice) ]
```

Then, the *run* function /* bla bla bla */

```
def run(self, n, x_init, offspring_size=5, n_generations=10, p=None, c=None, self_adapt=Fals
```

1.1 The One-Max problem

The implmentation of the one-max problem is straight forward. We start with a random vector in $0, 1^n$. Default mutation and crossover operator work fine. For the fitness function, we sum up all the bits in x (there is a builtin function *sum* provided with *Python*). The code is as follow:

```
ea_algo = ga.EA(fitness=sum)
x_init          = np.random.random_integers(0, 1, size=n)
best_x          = ea_algo.run(n, x_init, offspring_size=5, n_generations=100)
```

Self adaptative rule

1.2 The Maximum Matching

We represent an undirected graph like suggested in the **** Feuille de projet ****. For example:

```
vertices = range(n)
edges = [(i, i+1) if i+1 < n else (n-1, 0) for i in range(n)]
```

The function reponsible for calculating the degree of a vertex v in ine the subgraph consisting of the edges of M:

```
def deg(M):
    deg_m = np.zeros(m)
    for i, (e, f) in enumerate(edges):
        if M[i]:
            deg_m[e] += 1
            deg_m[f] += 1
    return sum([max(0, d-1) for d in deg_m])
```

And the fitness function:

```
def fitness(M):
    return sum(M) - m * deg(M)
```

Here is the result (edges of the best matching are coloured in red):