Evolutionary Algorithms
Project INF421

Gabriel Pereira de Carvalho

February 14, 2024

École polytechnique

Table of contents

Task 1: Individuals and benchmark functions

Individual class: Fully utilising memory

- Represent individual as array of integers
- Each integer represents 32 bits

```
def __init__(self, size):
"""
Constructor for new Individual
"""
# Number of integers necessary to represent all xi's
necessary_integers = (size + 31) // 32
self.size = size
self.bits = [0] * necessary_integers
```

Methods for Individual class

- Operations to update bit values in the population require bit manipulation

```python
def get(self, idx):
    """
    Get bit at index idx
    """
    test_bit = self.bits[idx // 32] & (1 << (idx % 32))
    return 1 if test_bit > 0 else 0

def set(self, idx):
    """
    Set bit at index idx to 1
    """
    self.bits[idx // 32] |= (1 << (idx % 32))
```

(1+1) EA implementation

- We use the same function to generate random initial and population and to perform *standard bit mutation*
- Termination condition: we know that $111\cdots1$ is the unique global optimum for all the benchmark funtions

```
def EvolutionaryAlgorithm(f, n):
    """
    (1+1) Evolutionary Algorithm
    """
    t = 0
    Pt = generateRandomOffspring(Individual(n), 0.5)

    while Pt.count() < n:
        y = generateRandomOffspring(Pt, 1 / n)
        if f(y) > f(Pt):  # pick solution that maximizes f
            Pt = y
        t += 1

    return Pt
```

Task 2: Runtime analysis for `OneMax` and `LeadingOnes`

Fitness Levels

Definition

We consider a **fitness-based partition** of the state space $E = \bigcup_{i \in [0..n]} A_i$ where

$$\forall i \in [0..n] \quad A_i = \{x \in E \quad | \quad f(x) = i\}.$$

Loop invariant: in the $(1+1)$ EA $f(P^t) \geq f(P^{t-1}) \quad \forall t \geq 1$.

Upper bound for the run time

Let T be a real random variable representing the run time.

Our strategy is to compute $\forall i \in [0..n-1]$ the probability p_i of leaving level A_i. Then, the expected number of iterations to leave level A_i is $\frac{1}{p_i}$.

Thus, we have the upper bound

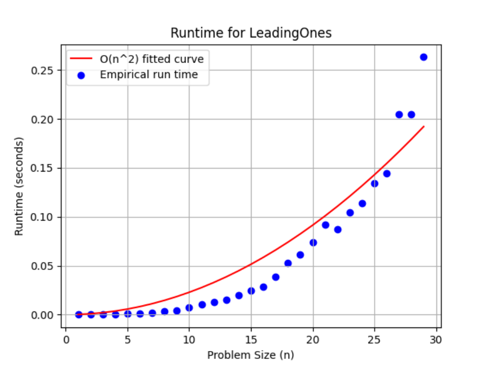
$$\mathbb{E}[T] \leq \sum_{i=1}^{n-1} \frac{1}{p_i}$$

for the expected run time.

Run time analysis `LeadingOnes`

$$p_i = \frac{1}{n}\left(\frac{n-1}{n}\right)^i \implies \sum_{i=0}^{n-1} \frac{1}{p_i} \leq n \sum_{i=0}^{n-1} \left(1 + \frac{1}{n-1}\right)^{n-1} \leq en^2$$

Thus, we have $O(n^2)$ time complexity.



Run time analysis `OneMax`

Let $\mathscr{P}(m,i,j)$ denote the probabilty of leaving level A_i and arriving at level A_j in an iteration for a problem size of m bits.

$$\mathscr{P}(m,0,j) = \frac{\binom{m}{j} p^j (1-p)^{m-j}}{\sum_{k=0}^{m} \binom{m}{k} p^k (1-p)^{m-k}} \tag{1}$$

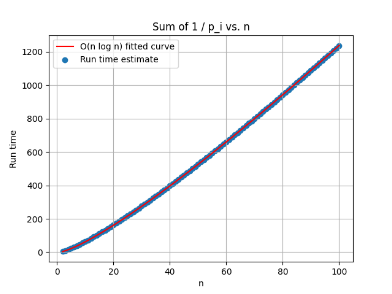
Now, we formulate our recurrence $\forall i \geq 1$.

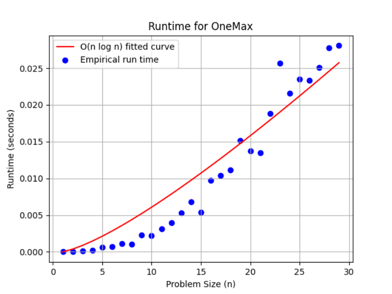
$$\mathscr{P}(m,i,j) = p\mathscr{P}(m-1,i-1,j) + (1-p)\mathscr{P}(m-1,i-1,j-1) \tag{2}$$

To calculate p_i using our coefficients, we have

$$p_i = \sum_{j=i+1}^{n} \mathscr{P}(n,i,j). \tag{3}$$

Run time analysis OneMax





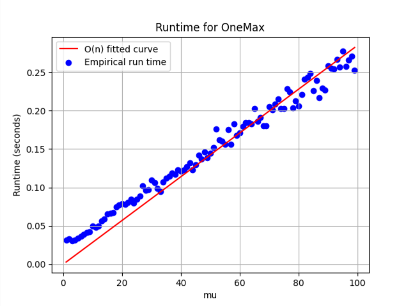
$(\mu+1)$ EA implementation

- Use a priority queue to update population in $O(\log(\mu))$.
- Each time we compute a fitness, update maximum.

```
while True:

if(most_fit_individual[1].count() == n):
    return most_fit_individual[1]

offspring = generateRandomOffspring(random.choice(Pt)[1], 1 / n)
fitness = f(offspring)
pair_fitness_offspring = (fitness, offspring)

if(fitness > most_fit_individual[0]):
    most_fit_individual = pair_fitness_offspring

heapq.heappushpop(Pt, pair_fitness_offspring)
t += 1
```

$(\mu+1)$ EA results

- We conclude that run time is strictly increasing function of μ.
- Increasing population size did not improve run time.
- The optimal value is $\mu = 1$.



Task 3: `Jump_k` function and the $(1+1)$ EA

- Distance between local optimum and global optimum is increasing in k.
- Probability of leaving plateau decreases exponentially with k.

$$\mathfrak{p} = p^k(1-p)^{n-k} \tag{4}$$

- Expected run time increases exponentially with k.

$$\frac{1}{\mathfrak{p}} = \frac{1}{p^k(1-p)^{n-k}} = \frac{n^n}{(n-1)^{n-k}} \tag{5}$$

Task 4: Implementing the $(\mu + \lambda)$ GA

$(\mu+\lambda)$ GA implementation

- Use a priority queue to update population in $O(\lambda \log(\mu))$.
- Each time we compute a fitness, update maximum.

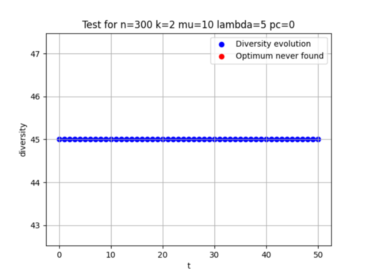
```
branch_decider = random.uniform(0, 1)
if(branch_decider < pc): #recombination with probability pc
        individual1 = random.choice(Pt)[1]
        individual2 = random.choice(Pt)[1]
        new_individual = Individual(n)
        for i in range(n):
            #chose between bit in individual1 and individual2
            bit_value = random.choice([individual1.get(i),
                                       individual2.get(i)])
            if(bit_value == 1): #if 1, set bit in offspring
                new_individual.set(i)
    else: #else we do normal EA iteration
        offspring.append(
            generateRandomOffspring(random.choice(Pt)[1], 1 / n)
        )
```

Task 5: Empirical runtime analysis for `Jump_k`

Conclusions

- Without crossover $p_c = 0$, solution is never found.
- With crossover $p_c > 0$, diversity becomes strictly decreasing.
- The bigger the initial diversity, the faster we find a solution.
- Bigger values of μ give bigger diversities $\implies$ faster runtime.
- Bigger values of λ find the solution faster.
- Big values of k still pose a problem, solution is never found.

Plots varying p_c



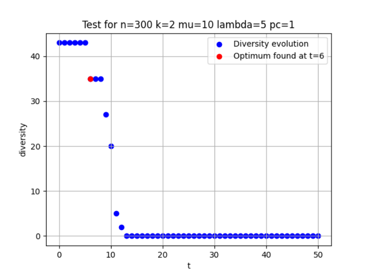


Figure 1: $p_c = 0$ and $p_c = 1$

Plots varying μ

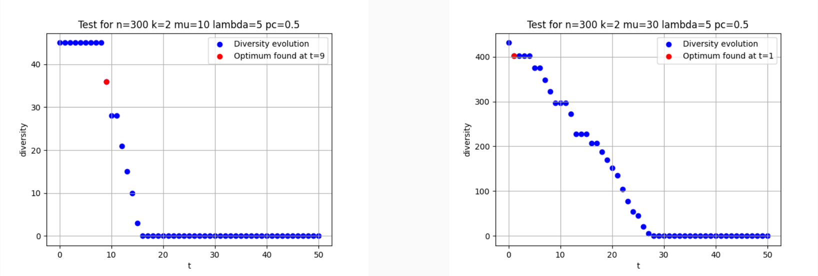


Figure 2: $\mu = 10$ and $\mu = 30$

Plots varying λ

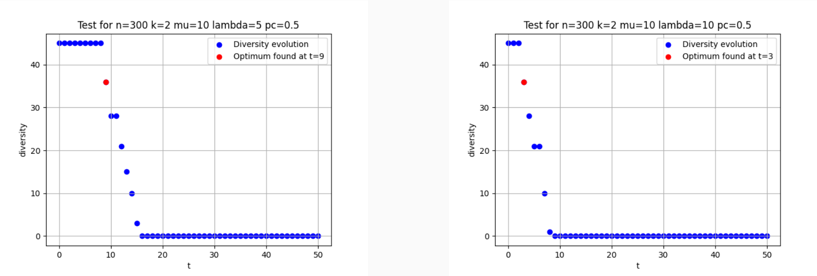
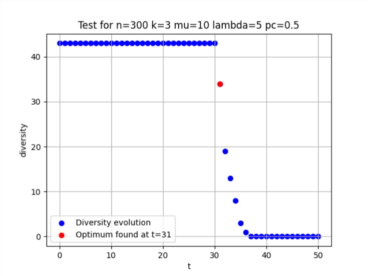


Figure 3: $\lambda = \frac{1}{2}\mu = 5$ and $\lambda = \mu = 10$

Plots varying k



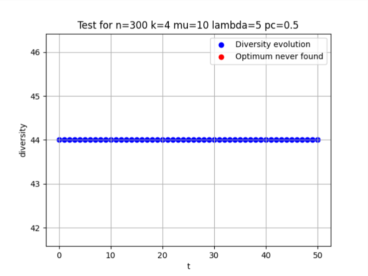


Figure 4: $k = 3$ and $k = 4$

Thank you for your attention!

Project's GitHub Repository