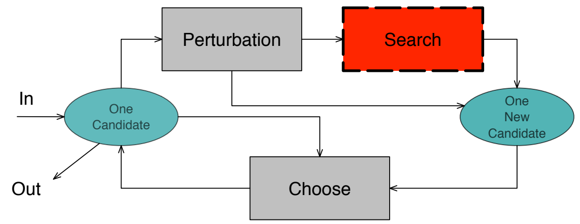
4

Evolutionary Algorithms

when biology meets computer science

Single State Stochastic



Evolution by Natural Selection

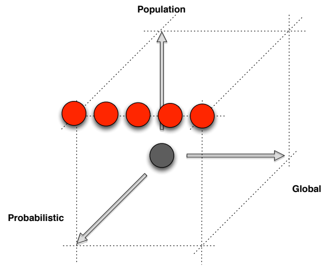
a population business

Selection

Reproduction with Variation

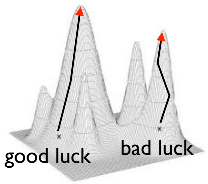
Evolutionary Algorithms

paralell stochastic search procedures guided by an objective function



Paralell Hill-Climbing

```
def paralell_hc(population_size, problem,max_iter):
    population = random_pop(problem, population_size)
    population = fitness_pop(population)
    for i in range(max_iter):
        for index,individual in enumerate(population):
            candidate,fit_candidate = individual
            # look for the first best
            for pos in range(len(candidate)):
                next_neighbor = neighbor(candidate,pos)
                cost_next_neighbor = fitness(next_neighbor)
                if cost_next_neighbor >= fit_candidate: # maximization
                    candidate = next_neighbor
                    fit_candidate = cost_next_neighbor
                    break
            population[index] = [candidate,fit_candidate]
    population.sort(key = operator.itemgetter(1))
    return population
```



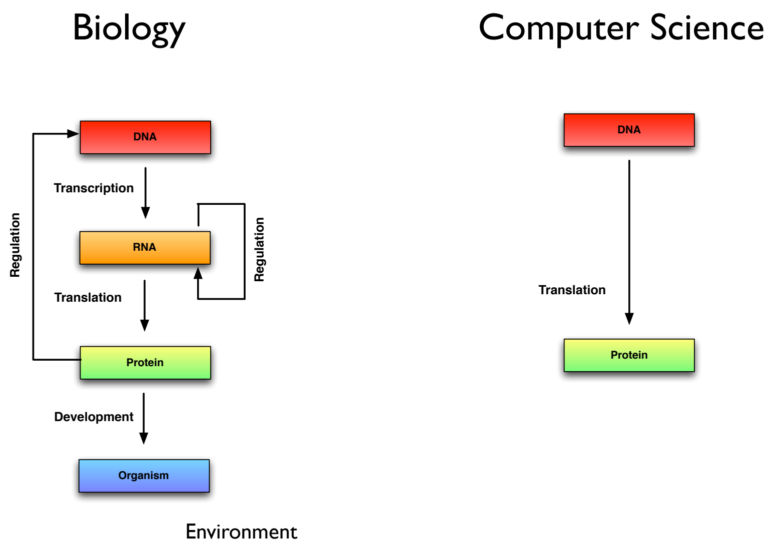
No Selection

No Reproduction

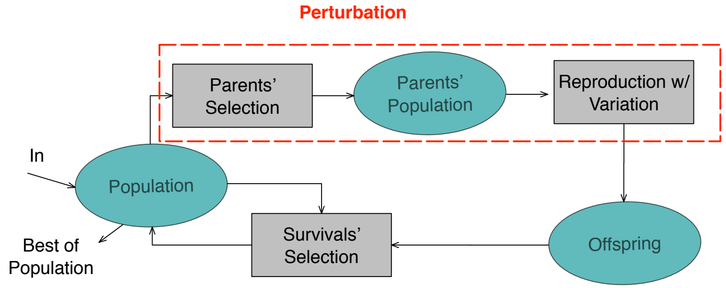
Evolutionary Algorithms

Adaptive and global stochastic search procedures in a space of candidate solutions, guided by an objective function

Inspiration



Model



Algorithm

Algorithm 1: Simple Evolutionary Algorithm

Input : NumGener,Problem
Output: Best

Population ← `RandomPopulation`(Problem)
Population ← `EvalPopulation`(Population)
foreach $gener_i \in NumGener$ **do**
  Mates ← `SelectParents`(Population)
  Offspring ← `Variation`(Mates)
  Offspring ← `EvalPopulation`(Offspring)
  Population ← `SelectSurvivors`(Population, Offspring)
return BestIndividual*(*Population*)*

Questions to be solved

Representation

Variation

Selection

Parents

Survivals

Example

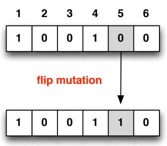
Numbers of João Brandão

Representation

Binary

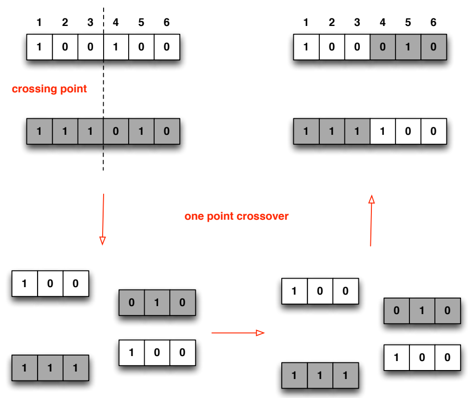
1	2	3	4	5	6
1	1	0	1	1	0

Variation



Mutation

Variation



Crossover

Variation

Probabilities

Selection

Survivals

Generational

Elitism

Selection

Parents

Roulette Wheel

Tournament

Implementation

Evolutionary Algorithm

```python
def sea(numb_generations,size_pop, size_cromo, prob_mut,prob_cross,selection,recombination,
mutation,survivors, fit_func, phenotype,elite,t_size):
    if int(size_pop * elite) == 0:
        print('No Elitism will be used!!!')
    #  initialize population: [...,indiv,....] with indiv = (cromo,fit)
    populacao = [(gera_indiv(size_cromo),0) for j in range(size_pop)]
    # evaluate population
    populacao = [[indiv[0], fit_func(indiv[0])] for indiv in populacao]
    populacao.sort(key=itemgetter(1), reverse = True) # Maximizing!
    for i in range(numb_generations):
    # work with a copy of the population
        aux_populacao = deepcopy(populacao)
        # select parents
        mate_pool = selection(aux_populacao, t_size)
    # Variation
    # ------ Crossover
        progenitores = []
        for i in  range(0,size_pop-1,2):
            cromo_1= mate_pool[i][0]
            cromo_2 = mate_pool[i+1][0]
            filhos = recombination(cromo_1,cromo_2, prob_cross)
            progenitores.extend(filhos)
        # ------ Mutation
        descendentes = []
        for cromo,fit in progenitores:
            novo_cromo = cromo
            novo_cromo = mutation(cromo,prob_mut)
            descendentes.append((novo_cromo,0))
        descendentes = [ [indiv[0], fit_func(indiv[0])] for indiv in descendentes]
        descendentes.sort(key=itemgetter(1), reverse = True)
        # New population
        populacao = survivors(populacao,descendentes,elite)
        # Evaluate and sort
        populacao = [[indiv[0], fit_func(indiv[0])] for indiv in populacao]
        populacao.sort(key=itemgetter(1), reverse = True) # Maximizing
    print("Individual: %s\nSize: %s\nFitness: %4.2f\nViolations:%d" % (phenotype(populacao[0][0]),
len(phenotype(populacao[0][0])), populacao[0][1],viola(phenotype(populacao[0][0]), size_cromo)))
    return 0
```


Fitness

```python
def merito(indiv):
    return evaluate(phenotype(indiv), len(indiv))

def phenotype(indiv):
    fen = [ i+1 for i in range(len(indiv)) if indiv[i] == 1]
    return fen

def evaluate(indiv, comp):
    alfa = 1.5
    beta = -1
    return alfa * len(indiv) + beta * viola(indiv,comp)
```

```python
def viola(indiv,comp):
    # Count violations
    v=0
    for elem in indiv:
        limite= min(elem-1,comp - elem)
        vi=0
        for j in range(1,limite+1):
            if ((elem - j) in indiv) and ((elem+j) in indiv):
                vi+=1
        v+=vi
    return v
```


Parents' Selection

Tournament

```python
def tournament_selection(population,t_size):
    size= len(population)
    mate_pool = []
    for i in range(size):
        winner = tournament(population,t_size)
        mate_pool.append(winner)
    return mate_pool

def tournament(population,size):
    """Maximization Problem.Deterministic"""
    pool = sample(population, size)
    pool.sort(key=itemgetter(1), reverse=True)
    return pool[0]
```


Variation

Crossover

```python
def one_point_cross(cromo_1, cromo_2,prob_cross):
    # in: cromo, out: indiv
    value = random()
    if value < prob_cross:
        pos = randint(0,len(cromo_1))
        f1 = cromo_1[0:pos] + cromo_2[pos:]
        f2 = cromo_2[0:pos] + cromo_1[pos:]
        return [(f1,0),(f2,0)]
    else:
        return [(cromo_1,0),(cromo_2,0)]
```

Variation

Mutation

```
def muta_bin(cromo,prob_muta):
    for i in range(len(cromo)):
        cromo[i] = muta_bin_gene(cromo[i],prob_muta)
    return cromo

#mutation: gene
def muta_bin_gene(gene, prob_muta):
    g = gene
    value = random()
    if value < prob_muta:
        g ^= 1
    return g
```


Survivals' Selection

Elitism

```
def survivors_elitism(parents,offspring,elite):
    """ Assunption: no size problems"""
    size = len(parents)
    comp_elite = int(size* elite)
    new_population = parents[:comp_elite] + offspring[:size - comp_elite]
    return new_population
```


Best

populacao.sort(key=itemgetter(1), reverse = True)

Lessons?