COMP 5660/6660 - Evolutionary Computing - Lecture Slides

Daniel Tauritz, PhD

Auburn University

August 27, 2021

Terminology

- Many computational problems can be formulated as **generate-and-test** search problems

Terminology

- Many computational problems can be formulated as **generate-and-test** search problems
- A **search space** contains the set of all possible solutions

Terminology

- Many computational problems can be formulated as **generate-and-test** search problems
- A **search space** contains the set of all possible solutions
- A **search space generator** is *complete* if it can generate the entire search space

Terminology

- Many computational problems can be formulated as **generate-and-test** search problems
- A **search space** contains the set of all possible solutions
- A **search space generator** is *complete* if it can generate the entire search space
- An **objective function** tests the quality of a solution

Terminology

- Many computational problems can be formulated as **generate-and-test** search problems
- A **search space** contains the set of all possible solutions
- A **search space generator** is *complete* if it can generate the entire search space
- An **objective function** tests the quality of a solution
- A **heuristic** is a problem-dependent rule-of-thumb

Terminology

- Many computational problems can be formulated as **generate-and-test** search problems
- A **search space** contains the set of all possible solutions
- A **search space generator** is *complete* if it can generate the entire search space
- An **objective function** tests the quality of a solution
- A **heuristic** is a problem-dependent rule-of-thumb
- A **meta-heuristic** determines the sampling order over a search space with the goal to find a near-optimal solution (or set of solutions)

Terminology

- Many computational problems can be formulated as **generate-and-test** search problems
- A **search space** contains the set of all possible solutions
- A **search space generator** is *complete* if it can generate the entire search space
- An **objective function** tests the quality of a solution
- A **heuristic** is a problem-dependent rule-of-thumb
- A **meta-heuristic** determines the sampling order over a search space with the goal to find a near-optimal solution (or set of solutions)
- A **hyper-heuristic** is a meta-heuristic for a space of programs

Algorithmic Toolbox

- A **Black-Box Search Algorithm (BBSA)** is a meta-heuristic which iteratively generates trial solutions employing solely the information gained from previous trial solutions, but no explicit problem knowledge

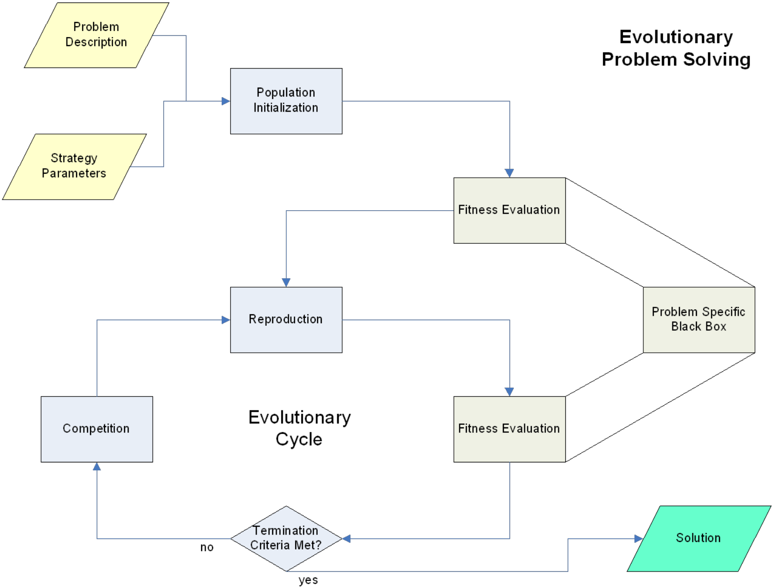
Algorithmic Toolbox

- A **Black-Box Search Algorithm (BBSA)** is a meta-heuristic which iteratively generates trial solutions employing solely the information gained from previous trial solutions, but no explicit problem knowledge
- **Evolutionary Algorithms (EAs)** can be described as a class of *stochastic*, *population-based* BBSAs inspired by *Evolution Theory*, *Genetics*, and *Population Dynamics*

Algorithmic Toolbox

- A **Black-Box Search Algorithm (BBSA)** is a meta-heuristic which iteratively generates trial solutions employing solely the information gained from previous trial solutions, but no explicit problem knowledge
- **Evolutionary Algorithms (EAs)** can be described as a class of *stochastic*, *population-based* BBSAs inspired by *Evolution Theory*, *Genetics*, and *Population Dynamics*

Evolutionary Cycle



Genospace versus phenospace

Genospace versus phenospace

Let F be the decoder function from G (genospace) to P (phenospace) and x^* be the global optimum.

Genospace versus phenospace

Let F be the decoder function from G (genospace) to P (phenospace) and x^* be the global optimum.

- $F : G \rightarrow P$ is *surjective* if $\forall p \in P \exists g \in G : F(g) = p$

Genospace versus phenospace

Let F be the decoder function from G (genospace) to P (phenospace) and x^* be the global optimum.

- $F : G \rightarrow P$ is *surjective* if $\forall p \in P \exists g \in G : F(g) = p$
- $F : G \rightarrow P$ is *injective* if $\forall g_1, g_2 \in G (F(g_1) = F(g_2)) \Rightarrow (g_1 = g_2)$

Genospace versus phenospace

Let F be the decoder function from G (genospace) to P (phenospace) and x^* be the global optimum.

- $F : G \to P$ is *surjective* if $\forall p \in P \exists g \in G : F(g) = p$
- $F : G \to P$ is *injective* if $\forall g_1, g_2 \in G (F(g_1) = F(g_2)) \Rightarrow (g_1 = g_2)$
- $F : G \to P$ is *bijective* if F is surjective and injective

Genospace versus phenospace

Let F be the decoder function from G (genospace) to P (phenospace) and x^* be the global optimum.

- $F : G \to P$ is *surjective* if $\forall p \in P \exists g \in G : F(g) = p$
- $F : G \to P$ is *injective* if $\forall g_1, g_2 \in G(F(g_1) = F(g_2)) \Rightarrow (g_1 = g_2)$
- $F : G \to P$ is *bijective* if F is surjective and injective
- If F is not surjective and $x^* \notin F(G)$, then the EA cannot find the global optimum. Therefore one should think twice before choosing a non-surjective decoder function if one cannot guarantee that the global optimum is still reachable.

Genospace versus phenospace

Let F be the decoder function from G (genospace) to P (phenospace) and x^* be the global optimum.

- $F : G \rightarrow P$ is *surjective* if $\forall p \in P \exists g \in G : F(g) = p$
- $F : G \rightarrow P$ is *injective* if $\forall g_1, g_2 \in G(F(g_1) = F(g_2)) \Rightarrow (g_1 = g_2)$
- $F : G \rightarrow P$ is *bijective* if F is surjective and injective
- If F is not surjective and $x^* \notin F(G)$, then the EA cannot find the global optimum. Therefore one should think twice before choosing a non-surjective decoder function if one cannot guarantee that the global optimum is still reachable.
- F does not need to be injective, but realize there is less to search if F is injective so there should be sufficient compensation, such as limiting F(G) to valid solutions in a constraint satisfaction problem.

The 0-1 Knapsack Problem

Given a set of n items with values v_i and cost c_i, select a subset that maximises value while not exceeding the capacity limit C_{max}.

The 0-1 Knapsack Problem

Given a set of n items with values v_i and cost c_i, select a subset that maximises value while not exceeding the capacity limit C_{max}.We consider two cases:

1. $fitness(p) = \sum_{i=1}^{n}(v_i \cdot g_i)$

The 0-1 Knapsack Problem

Given a set of n items with values v_i and cost c_i, select a subset that maximises value while not exceeding the capacity limit C_{max}.We consider two cases:

1. $fitness(p) = \sum_{i=1}^{n}(v_i \cdot g_i)$
2. Modify $fitness(p)$ to exclude items that would exceed C_{max}