Biased Random-key Genetic Algorithms

Research Seminar Artificial Intelligence and Machine Learning

Philip Naumann

philip.naumann@fu-berlin.de

@ PhilipNaumann.com

June 03, 2021

Freie Universität Berlin

For further reference have a look at the tutorial presentation of *Mauricio G. C. Resende*:

Mauricio G. C. Resende. *Biased random-key genetic algorithms: A tutorial*. CLAIO/SBPO 2012

What will be discussed?

What will be discussed?

- (Discrete) optimization/minimization
- Evolutionary Algorithm (EA)
- Biased Random-key Genetic Algorithm (BRKGA)
  - **Genotype**
  - **Phenotype**
  - **Decoder**
  - Biased **recombination**
  - **Mutant generation** and elitist **selection**
- How to solve problems with BRKGA

Brief Overview

Evolutionary Algorithms — General overview: (Eiben and Smith 2015)

- **Genetic Algorithms** (GAs) (1960–1970)
  - Idea: use existing **building blocks** to create new solutions (*recombination*)
- **Random-key GAs** (RKGAs) (Bean 1994)
  - Idea: same as in GAs, but optimize in a fixed space ($[0, 1]$) and **decode solutions**
- **Biased Random-key GAs** (BRKGAs) Overview: (Gonçalves and Resende 2011)
  - Idea: same as in RKGAs, but **favor good solutions**
- ...

EA Terminology

- A **genotype** G is the *internal* representation of a solution (e.g. *binary*)
- A **phenotype** P is the *actual* representation of a solution (e.g. *natural number*)
- A **population** $\mathcal{P}$ is the current set of (*genotype*) solutions
- **Evaluation** is the *rating of solutions* based on their *fitness* f
- **Selection** is the strategy how *parent* solutions are picked from $\mathcal{P}$
- **Recombination** is the *combination of parents* to create *offsprings* (**exploitation**)
- **Mutation** is (controlled) *adding of noise* to offsprings (**exploration**)

Why to use BRKGA?

Why to use BRKGA?

For solving **(non-differentiable) combinatorial** problems!

- Routing problems (e.g. Traveling salesman)
- Scheduling problems (e.g. Job shop scheduling)
- Allocation problems (e.g. Knapsack)
- Graph & set problems (e.g. Matching, minimum spanning tree, set cover)
- ...

Evolutionary operators (*recombination*, *mutation*) are hard to define in combinatorial settings due to the importance of a **meaningful ordering**—risk to create **infeasible** or **redundant** solutions.

Not fixed to combinatorial problems, but most powerful for these.

Optimization

Given an **objective function** (*single-objective*)

$$f : X \to \mathbb{R}$$

We want to find a **solution** $x \in X$ s.t. (for *minimzation*)

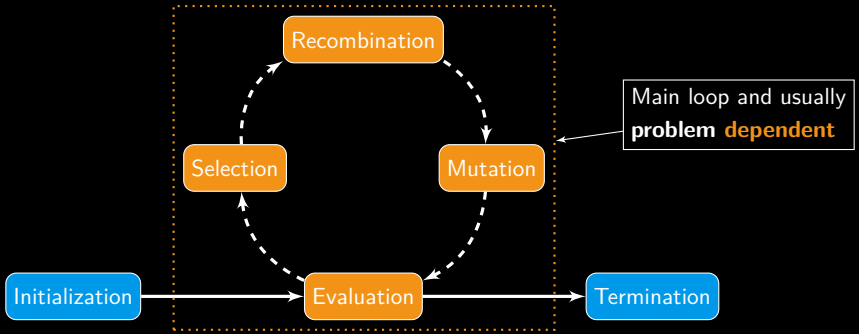
$$f(x) \leq f(y) \quad \forall \quad y \in X$$

❗ There are *no conditions* to f, so it can be

- a *black-box*
- *non-convex*
- *non-differentiable* etc.

How does it work?

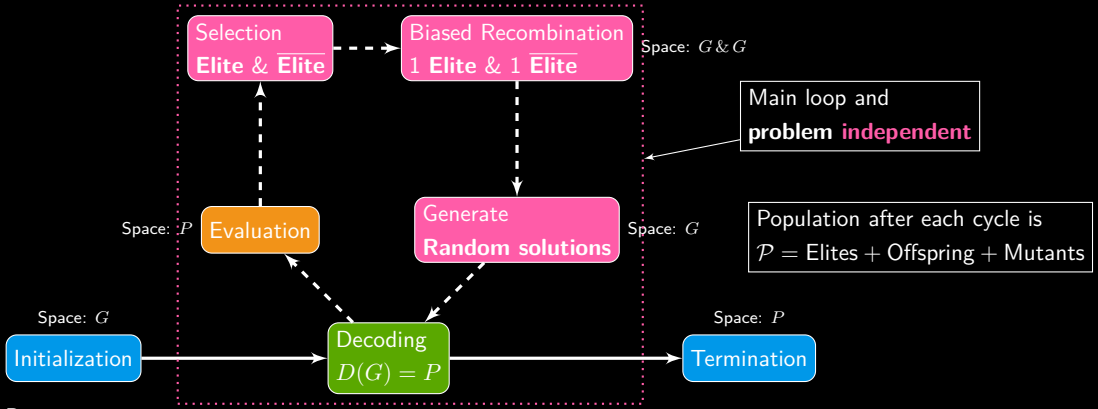
The General EA Cycle



The recipe:

- Solution $x \in X$
- Population $\mathcal{P} = \{x_i\} \subseteq X$
- Evaluation (`fitness`) $f : X \to \mathbb{R}$
- Operators `selection`, `recombination`, `mutation`, ...

The BRKGA Cycle



Parameters:

- Number of **elites** to select
- Number of **offspring** to generate (through *recombination*)
- Number of **mutants** to generate (*randomly generated*)
- **Biased probability** towards elites

The **genotype** G is the ***internal representation*** of a **solution**

$$G = [v_1, v_2, \ldots, v_N] \in [0, 1]^N$$

Random key

A solution is *encoded* as an N-dimensional vector G of random keys $v_i \in [0, 1] \subset \mathbb{R}$

This representation is **problem *independent***!

The **phenotype** P is the ***actual representation*** of the **solution**

$$P = [a_1, a_2, \ldots, a_K] \in X$$

Actual value

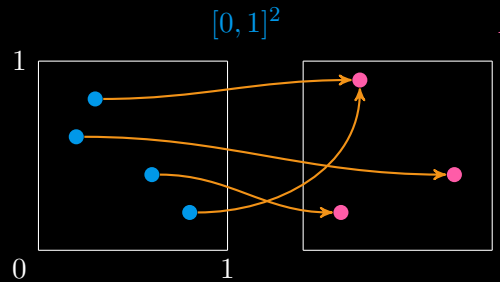
A solution is *decoded* as an K-dimensional vector P of problem values $a_i \in X_i$

This representation is **problem *specific***! Note that it's possible that $K \neq N$.

The **decoder** D is a surjective ***mapping function*** from G to P

$$D(G) = P$$

$$D : [0,1]^N \twoheadrightarrow X$$



Decoding from one space into another

Allows to *deterministically* derive P from G in order to evaluate $f(P)$

Should be able to reach *all feasible* solutions in P!

Examples for Decoders

$$G = [\underbrace{0.23, 0.65, 0.98}_{\text{Order}}, \underbrace{0.17, 0.55, 0.99}_{\text{Value}}]$$

- **Argsort**, for finding an optimal order

$$\texttt{argsort}(G) = [4, 1, 5, 2, 3, 6] = P$$

- (*Linear*) **interpolation** with $\texttt{Interp} : [0, 1] \to [0, 100]$

$$\texttt{Interp}(G) = [23, 65, 98, 17, 55, 99] = P$$

- **Combinations**, e.g. `argsort` and `Interp` (for an *allocation* or *scheduling* problem)

$$P = [1, 2, 3, 17, 55, 99] \Rightarrow \langle (1, 17), (2, 55), (3, 99) \rangle$$

Elitist Selection & Mutant Generation

Selection by elitist selection

Partition the population $\mathcal{P}$ into **elite** $\mathcal{P}_E$ and **non-elite** $\mathcal{P}_{\overline{E}}$ solutions based on f. For example, in a population of 100 solutions, the top 10 solutions according to f represent the elite population:

$$\mathcal{P}_E = \{G_i \forall\, G_i \in \mathcal{P} \mid f(G_i) \leq f(G_{10} \in \mathcal{P}_E)\}$$

$$\mathcal{P}_{\overline{E}} = \mathcal{P} - \mathcal{P}_E$$

Exploration by mutants

For the **exploration**, **add random solutions** (the so-called **mutants**):

$$M \in \mathcal{U}[0,1]^N$$

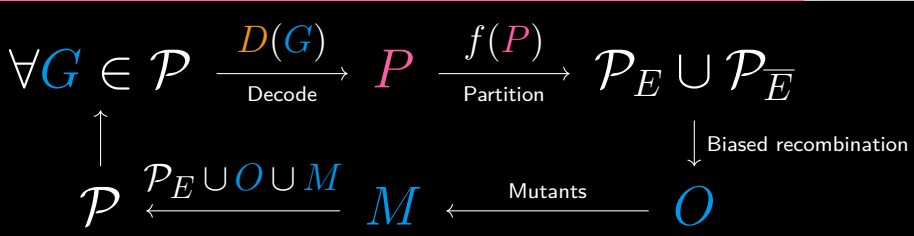
Actually the same in the **initilization**.

Biased Recombination: Exploitation of Good Solutions

- Create **offspring** solutions O by combining $G_E \in \mathcal{P}_E$ and $G_{\overline{E}} \in \mathcal{P}_{\overline{E}}$.
- Idea: bias towards the "good" solution to augment the *uniform recombination*.
- For example, if random number $r \leq p = 0.7$, then favor elite G_E's value.

$$
\begin{array}{rcl}
G_E & = & [\ v_1 \quad v_2 \quad v_3 \quad v_4 \quad v_5\ ] \\
G_{\overline{E}} & = & [\ v_1 \quad v_2 \quad v_3 \quad v_4 \quad v_5\ ] \\
r & = & 0.32 \quad 0.10 \quad 0.61 \quad 0.94 \quad 0.13 \\
 & & \downarrow{\leq}0.7 \quad \downarrow{\leq}0.7 \quad \downarrow{\leq}0.7 \quad \downarrow{>}0.7 \quad \downarrow{\leq}0.7 \\
O & = & [\ v_1 \quad v_2 \quad v_3 \quad v_4 \quad v_5\ ]
\end{array}
$$

The Main Cycle



1. Create genotype **solutions** as random-key vectors G
2. **Decode** G to get $P = D(G)$
3. **Evaluate** the objective (goal) on $f(P)$
4. Rank solutions according to $f(P)$ and use the best ones $\mathcal{P}_E$ to breed **offspring** O (through *biased recombination*)
5. Add **mutants** M to the population $\mathcal{P}$ to *maintain diversity* and *exploration* of the search space

Example: The 4-Queens Problem

Problem

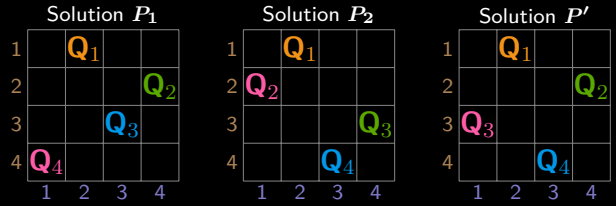
Place $n = 4$ **queens** $\mathbf{Q}_i$ so that there is no other in its *row*, *column* or *diagonal*.

Formulation

- Value a_i of $P = [a_1, a_2, a_3, a_4]$ provides the **column** of queen $\mathbf{Q}_i$ in **row** i.
- The **decoder** D uses **argsort** to map the i-th queen to the a_i-th column.
- The **objective function** $f : P \to \mathbb{N}_0$ provides the **total number of conflicts** (on the *diagonal*).

Note, that with this formulation of the decoder we already prohibit a conflict of two queens appearing in the same row or column!

Solution



$$G_1 = [0.8, 0.2, 0.6, 0.3] \xrightarrow[D(G_1)]{\texttt{argsort}} P_1 = [2, 4, 3, 1] \quad f(P_1) = 1 \quad \text{(best, \textbf{elite})}$$

$$G_2 = [0.4, 0.3, 0.9, 0.5] \xrightarrow[D(G_2)]{\texttt{argsort}} P_2 = [2, 1, 4, 3] \quad f(P_2) = 4 \quad \text{(worst, \textbf{non-elite})}$$

Biased recombination:

$$G' = [\underbrace{0.8}_{G_1}, \underbrace{0.2}_{G_1}, \underbrace{0.9}_{G_2}, \underbrace{0.3}_{G_1}] \xrightarrow[D(G')]{\texttt{argsort}} P' = [2, 4, 1, 3] \quad f(P') = 0 \quad \text{(optimal)}$$

Advantages & Disadvantages

👍 Advantages

Biggest advantage!

- **Problem independent**
- No conditions on the objective function
- Makes use of local knowledge (recombination)
- Maintains diversity in the population
- Easy to extend (e.g. parallelization)

👎 Disadvantages

- Not trivial to define an *efficient* decoder
- Can get stuck and not escape
- Parameters can have a big impact
- Can be slow for (very) complex problems (no gradient info)

Most important takeaways

BRKGAs

- are problem independent
- are powerful for combinatorial problems
- but only as powerful as the decoder function
- make no assumptions on the objective function

References

Bean, James C. 1994. "Genetic Algorithms and Random Keys for Sequencing and Optimization." *ORSA Journal on Computing* 6 (2): 154–60. https://doi.org/10.1287/ijoc.6.2.154.

Eiben, A. E., and J. E. Smith. 2015. *Introduction to Evolutionary Computing*. Natural Computing Series. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-44874-8.

Gonçalves, José Fernando, and Mauricio G. C. Resende. 2011. "Biased Random-Key Genetic Algorithms for Combinatorial Optimization." *Journal of Heuristics* 17 (5): 487–525. https://doi.org/10.1007/s10732-010-9143-1.