Introduction to Metaheuristics
Evolutionary Search

DrC. Alejandro Piad Morffis

Review: What can we do

Key ideas in local search:

- Estimate gradients sampling from a neighborhood function
- Select best solutions to exploit gradients
- Accept random bad solutions to explore

How to avoid local optima:

- Accept bad solutions
- Forbid or penalize seen solutions
- Restart frequently

How to balance exploration/exploitation:

- Via a tradeoff parameter (e.g., probability to select bad solutions)

Why optimization is (still) hard

What is the single hardest problem in metaheuristic design (so far)?

Why optimization is (still) hard

What is the single hardest problem in metaheuristic design (so far)?

How to balance exploration and exploitation!

Why is this hard (so far)?

Why optimization is (still) hard

What is the single hardest problem in metaheuristic design (so far)?

How to balance exploration and exploitation!

Why is this hard (so far)?

Because every iteration, we must either explore or exploit.

Can we do both at the same time?

Going beyond single-solution

What if we have multiple solutions at the same time?

Not just sample multiple solutions, but actually keep multiple solutions as part of the current search state.

What is would the advantage be?

Going beyond single-solution

What if we have multiple solutions at the same time?

Not just sample multiple solutions, but actually keep multiple solutions as part of the current search state.

What is would the advantage be?

- We can get a more accurate idea of the function landscape
- We can explore multiple attraction basins at the same time
- Some solutions can move in exploration and others in exploitation steps

Going beyond single-solution

What is the simplest way to implement this?

Going beyond single-solution

What is the simplest way to implement this?

- start with a random population $x_1^{(0)}, \ldots, x_n^{(0)}$
- while not finished:
  - for each $x_k^{(i)}$ do
    - sample one or more $x_k^{(i+1)}$
    - apply a local search strategy
- return x^*

What are we missing here?

Going beyond single-solution

What is the simplest way to implement this?

- start with a random population $x_1^{(0)}, \ldots, x_n^{(0)}$
- while not finished:
  - for each $x_k^{(i)}$ do
    - sample one or more $x_k^{(i+1)}$
    - apply a local search strategy
- return x^*

What are we missing here?

This is just the same as running N local searches in parallel. There is no knowledge transfer between different solutions.

How do we make different solutions interact?

Enter evolutionary search

Where in local search you use what you've learned about the function?

Enter evolutionary search

Where in local search you use what you've learned about the function?

Every time you generate a new solution.

How do we make that knowledge more global?

Enter evolutionary search

Where in local search you use what you've learned about the function?

Every time you generate a new solution.

How do we make that knowledge more global?

Generate a new solution from more than one existing solution!

Can we find a reason to expect this might work?

Evolutionary search

Making the paradigm explicit

- initialize population of size N (and evaluate)
- while not stop:
  - generate M offsprings (recombination or crossover)
  - evaluate the offsprings
  - choose N out of $M + N$ (selection)
- return best ever

Design space

Selection strategies:

- Top best
- Tournament
- Probabilistic
- ...

Recombination strategies:

- Point-based
- Uniform selection
- Ad-hoc given the problem
- ...

Analysis

How good is this strategy?

- How does it avoid local optima?
- How does it exploit attraction basins?
- How does it discover new attraction basins?

Analysis

How good is this strategy?

- How does it avoid local optima?
- How does it exploit attraction basins?
- How does it discover new attraction basins?

What's wrong with exploration?

Analysis

How good is this strategy?

- How does it avoid local optima?
- How does it exploit attraction basins?
- How does it discover new attraction basins?

What's wrong with exploration?

Exploration by crossover means you can only generate solutions in the **convex hull** of the current population. (Why?)

How do we fix that?

Genetic Algorithms

- initialize population of size N (and evaluate)
- while not stop:
  - generate M offsprings (crossover)
  - **modify each new solution (mutation)**
  - evaluate the offsprings
  - choose N out of $M + N$ (selection)
- return best ever

Mutation strategies:

- Flip a random bit
- Swap random components
- Add a random small perturbation
- ...

Variants for continuous

Differential Evolution: An evolutionary algorithm for continuos optimization where genetic operators are vector operations.

Core step (generation of new solutions):

- for each x:
  - pick 3 vectors a, b, c
  - generate a new vector $y = a + F(b - c)$
  - for each component y_i:
    - if $U(0,1) < CR$ replace $y_i = x_i$
  - randomly perturbate one y_i (mutation)
  - if $f(y) < f(x)$ select it, else keep x

Why any of this works

What is the core hypothesis in evolutionary algorithms?

Why any of this works

What is the core hypothesis in evolutionary algorithms?

The building blocks hypothesis: The fitness of a solution is (at least in part) determined by the presence of certain building blocks.

This assumes that:

- There is a way to split a solution into meaningful components.
- The overall fitness depends in a predictable way from some quantifiable quality of each component.
- Components that appear in good solutions tend to have a higher quality.

Thus reusing components from good solutions will lead to more good solutions.

The representation issue

The building blocks hypothesis, if true, assumes a specific representation.

In general, there is no intrinsically best representation. Each problem requires careful design.

Some common representations:

- Bit strings
- Real-valued vectors
- Permutations
- ...

The big problem: guaranteeing feasibility.

Genetic Programming

What happens when representations are too complex? Use programs.

- Solutions are ASTs
- Crossover means swapping some subtree by another
- Mutation means changing a tree

Example: Analytic regression. Finding a mathematical formula that best fits a set of observations.

Genetic Programming

What happens when representations are too complex? Use programs.

- Solutions are ASTs
- Crossover means swapping some subtree by another
- Mutation means changing a tree

Example: Analytic regression. Finding a mathematical formula that best fits a set of observations.

Big problems:

- How do you guarantee all changes give valid programs?
- How do you guarantee you explore all possible programs?

Detaching optimization from evaluation

The representation problem is due to using the same representation for optimization and evaluation

Detaching optimization from evaluation

The representation problem is due to using the same representation for optimization and evaluation

Instead of operating on the actual object you are looking for (e.g., a helicopter design), operate on an abstract **genotpypical** representation, and then build a **phenotypical** representation for evaluation.

Detaching optimization from evaluation

The representation problem is due to using the same representation for optimization and evaluation

Instead of operating on the actual object you are looking for (e.g., a helicopter design), operate on an abstract **genotpypical** representation, and then build a **phenotypical** representation for evaluation.

You need to program a "constructor" that computes the the phenotype from the genotype.

Example: Grammatical evolution The phenotypical constructor is a context-free grammar. (We will get back to this later...)

Summary

Evolutionary computation draws inspiration from biological evolution but it is **not** an accurate simulation of biological phenomena (it doesn't need nor want to be).

The main hypothesis is that good components (genes) correspond to good solutions (phenotypes).

The fundamental issue is finding a meaningful representation, where fitness depends from **(mostly) independent** components.

Time to practice!