Metaheuristics Lesson

Johann Dreo

Contents

Chapter 1

Introduction

Key concepts

Metaheuristics are mathematical optimization algorithms solving $\arg\min_{x \in X} f(x)$.

Synonyms:

- search heuristics,
- evolutionary algorithms,
- stochastic local search.

The general approach is to only look at the solutions, by trial and error, without further information on its structure. Hence the problem is often labelled as "black-box".

Link to NP-hardness/curse of dimensionality: easy to evaluate, hard to solve. Easy to evaluate = fast, but not as fast as the algorithm itself. Hard to solve, but not impossible.

1.1 Hard optimization

Metaheuristics are algorithms which aim at solving "hard" mathematical optimization problems.

A mathematical optimization problem is defined by a "solution" x and an "objective function" $f : x \mapsto \mathbb{R}$:

$$\arg\min_{x \in X} f(x)$$

One can consider using $\arg\max$ without loss of genericity[1]. Usually, the set

[1]In the metaheuristics literature, $\arg\max$ is often assumed for evolutionary algorithms, whether $\arg\min$ is often assumed for local search or simulated annealing.

X is defined intentionally and constraints on x are managed separately. For example, using a function $g : x \mapsto \{0, 1\}$:

$$\operatorname*{arg\,min}_{x \in [0,1]^n} f(x), \quad \text{s.t.} \quad g(x) = 0$$

Chapter 2

Algorithmics

Key concepts

Those algorithms are randomized and iteratives (hence stochastics) and manipulates a sample (synonym population) of solutions (s. individual) to the problem, each one being associated with its quality (s. cost, fitness).

Thus, algorithms have a main loop, and articulate functions that manipulates the sample (called "operators").

Main design problem: exploitation/exploration compromise (s. intensification/diversification). Main design goal: raise the abstraction level. Main design tools: learning (s. memory) + heuristics (s. bias).

Forget metaphors and use mathematical descriptions.

Seek a compromise between complexity, performances and explainability.

The is no better "method". Difference between model and instance, for problem and algorithm. No Free Lunch Theorem. But there is a "better algorithm instances on a given problem instances set".

The better you understand it, the better the algorithm will be.

Chapter 3

Problem modelization

Key concepts

Way to assess the quality: fitness function. Way to model a solution: encoding.

3.1 Main models

Key concepts

Encoding:

- continuous (s. numeric),
- discrete metric (integers),
- combinatorial (graph, permutation).

Fitness:

- mono-objective,
- multi-modal,
- multi-objectives.

3.2 Constraints management

Key concepts

Main constraints management tools for operators: - penalization, - reparation, - generation.

Chapter 4

Performance evaluation

4.1 What is performance

Key concepts

Main performances axis:

- time,
- quality,
- probability.

Additional performance axis:

- robustness,
- stability.

Golden rule: the output of a metaheuristic is a distribution, not a solution.

4.2 Empirical evaluation

Key concepts

Proof-reality gap is huge, thus empirical performance evaluation is gold standard.

Empirical evaluation = scientific method.

Basic rules of thumb:

- randomized algorithms => repetition of runs,
- sensitivity to parameters => design of experiments,
- use statistical tools,
- design experiments to answer a single question,
- test one thing at a time.

4.3 Useful statistical tools

Key concepts

Statistical tests:

- classical null hypothesis: test equality of distributions.
- beware of p-value.

How many runs?

- not always “as many as possible”,
- maybe “as many as needed”,
- generally: 15 (min for non-parametric tests) – 20 (min for parametric-gaussian tests).

Use robust estimators: median instead of mean, Inter Quartile Range instead of standard deviation.

4.4 Expected Empirical Cumulative Distribution Functions

Key concepts

On Run Time: ERT-ECDF.

$$ERTECDF(\{X_0,\dots,X_i,\dots,X_r\},\delta,f,t) := \#\{x_t \in X_t | f(x_t^*) >= \delta\}$$

$$\delta \in \left[0, \max_{x\in\mathcal{X}}(f(x))\right]$$

$$X_i := \left\{\left\{x_0^0,\dots,x_i^j,\dots,x_p^u | p \in [1,\infty[\right\} | u \in [0,\infty[\right\} \in \mathcal{X}$$

with p the sample size, r the number of runs, u the number of iterations, t the number of calls to the objective function.

The number of calls to the objective function is a good estimator of time because it dominates all other times.

The dual of the ERT-ECDF can be easily computed for quality (EQT-ECDF).

3D ERT/EQT-ECDF may be useful for terminal comparison.

4.5 Other tools

Key concepts

Convergence curves: do not forget the golden rule and show distributions:

- quantile boxes,
- violin plots,
- histograms.

Chapter 5

Algorithm Design

5.1 Neighborhood

Key concepts

Convergence definition(s):

- strong,
- weak.

Neighborhood: subset of solutions atteinable after an atomic transformation:

- ergodicity,
- quasi-ergodicity.

Relationship to metric space in the continuous domain.

5.2 Structure of problem/algorithms

Key concepts

Structure of problems to exploit:

- locality (basin of attraction),
- separability,
- gradient,
- funnels.

Structure with which to capture those structures:

- implicit,
- explicit,
- direct.

Silver rule: choose the algorithmic template that adhere the most to the problem model.

- taking constraints into account,
- iterate between problem/algorithm models.

5.3 Grammar of algorithms

Key concepts

Parameter setting < tuning < control.

Portfolio approaches. Example: numeric low dimensions => Nelder-Mead Search is sufficient.

Algorithm selection.

Algorithms are templates in which operators are interchangeable.

Most generic way of thinking about algorithms: grammar-based algorithm selection with parameters. Example: modular CMA-ES.

Parameter setting tools:

- ParamILS,
- SPO,
- i-race.

Design tools:

- ParadisEO,
- IOH profiler
- jMetal,

- Jenetics,
- ECJ,
- DEAP,
- HeuristicLab.

5.4 Landscape-aware algorithms

Key concepts

Fitness landscapes: structure of problems as seen by an algorithm. Features: tool that measure one aspect of a fitness landscape.

We can observe landscapes, and learn which algorithm instance solves it better. Examples: SAT, TSP, BB.

Toward automated solver design.