INSTITUTO
DE INFORMÁTICA
UFRGS

AUTOMATIC ALGORITHM CONFIGURATION – METHODS

Marcus Ritt

May 2017

Introduction and Overview

Goals

- Brief overview on automatic algorithm configuration (AAC)
- Panorama of the methods related to AAC
- Main research lines

The problem

- Search space: (heuristic) algorithms solving a problem.
- Objective: find an algorithm of good performance.

Exact problem is ***undecidable***.

- Solution: some (truly meta) heuristics which select an algorithm.

Most of the routine part of finding good algorithms can be done better automatically

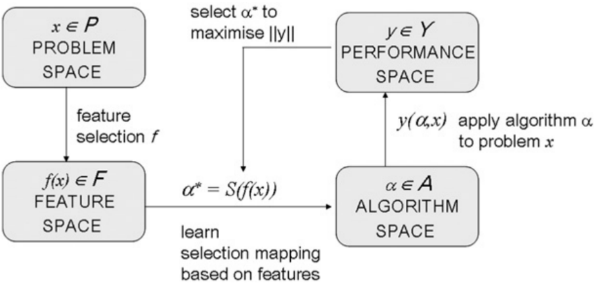
Humans

- usually work iteratively, by trial and error, with small-scale experiments,
- are slow,
- are biased,
- get easily fooled by complex interactions,
- get easily bored.

Let humans do the ***creative part***! (For now.)

Rice (1976): The algorithm selection problem

- Four spaces: problems $\mathcal{P}$, features $\mathcal{F}$, algorithms $\mathcal{A}$, performance $\mathcal{Y}$.
- Feature selection: $f : \mathcal{P} \rightarrow \mathcal{F}$
- Evaluation: $y : \mathcal{A} \times \mathcal{P} \rightarrow \mathcal{Y}$
- Algorithm selection: $S : \mathcal{F} \rightarrow \mathcal{A}$
- Goal: $y(S(f(x)), x) = \max_{a \in \mathcal{A}} ||y(a, x)||$.



Source: Smith-Miles and Lopes (2012)

- Rice's formulation

$$y(S(f(x)), x) = \max_{a \in \mathcal{A}} ||y(a, x)||$$

is ***instance-based*** (online selection).

- Note: y can be an expected value, e.g. over seeds.
- For (offline) ***algorithm configuration***:

$$y(a) = \max_{a \in \mathcal{A}} ||y(a)||$$

- Now: y is a summary statistic over instances, too.

The Fastest and Shortest Algorithm for All Well-Defined Problems[1]

Where's the problem?

Marcus Hutter

IDSIA, Galleria 2, CH-6928 Manno-Lugano, Switzerland
marcus@idsia.ch http://www.idsia.ch/~marcus

Key Words

Acceleration, Computational Complexity, Algorithmic Information Theory, Kolmogorov Complexity, Blum's Speed-up Theorem, Levin Search.

- What is the ***search space*** $\mathcal{A}$? How do we represent elements $x \in \mathcal{A}$?
- What is the (exploitable) ***structure*** of the search space?
- How do we ***evaluate*** (cheaply!) an algorithm, or compare two algorithms?

The search space

- Parameters: ***algorithm tuning or calibration***.
  Typically: Numerical (real, integer), ordinal, categorical.
- Some selected algorithms: ***portfolio methods***.
  Typically: online.
- Large class of algorithms: ***algorithm configuration or design***.
  Typically: Syntax trees, grammars.

These categories are ***blurry***.

Search space structure

- What is the ***distance*** of two algorithms?
- What is the ***neighbor*** of an algorithm?
- What is the ***recombination*** of two algorithms?
- Is there any ***fitness-distance correlation***?

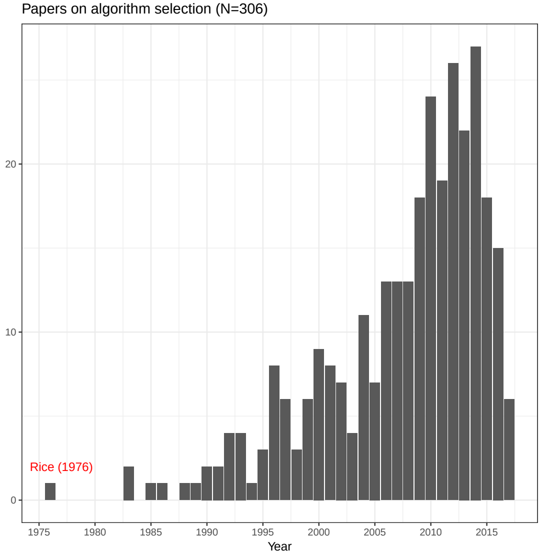
Without structure: can't do better than ***random search*** (no free lunch!).

What is good performance?

- Theoretical minimum: ***time*** and solution ***quality***.
- Trade-off: ***fast***, ***anytime***, ***best*** possible.
- Problems: ***instance-dependent*** often ***stochastic*** (= seed-dependent).
- Ideally: Empirical performance models.

Performance evaluation is the
bottleneck and ***drives algorithms***

Publications on algorithm selection



Source: Kotte (2016)

Related concepts

- Programming by optimization (Hoos 2012; Hoos 2014)
  - Try to ***avoid design choices***
  - Make them ***explicit***
- Hyper-heuristics (Cowling, Kendall, and Soubeiga 2000)
  - Typically ***online***.
- Automatic algorithm configuration (Birattari 2005)

Features, hardness and performance models

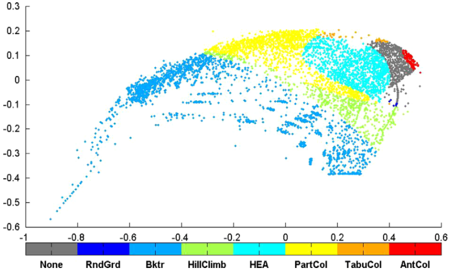
Feature extraction: what?

- ***Goals***: reduce data, explain observations, simplify models.
- ***Problem-independent***: fitness landscape analysis (e.g. ruggedness, fitness-distance correlation), landmarks.
- ***Problem-dependent***: e.g. density in SAT problems, triangle inequality for distances.

Correlation with performance
usually depends on the algorithm

Hardness models

- Techniques: instance classification via ***machine learning***
  - e.g. random trees, Bayesian classifiers, decision trees, neural networks.
- Example: ***Graph coloring*** (Smith-Miles et al. 2014)
  - 18 features, 8 algorithms, GA evolves feature selections
  - Fitness: success of Naive Bayesian classifier to predict being best after projection to 2D with PCA
  - Classifier: Support vector machine



Source: Smith-Miles et al. (2014)

Empirical performance models

- ***Assumption***: Practical hardness $\Leftrightarrow$ Empirical performance.
- Solves Rice (1976)'s problem: select algorithm of best predicted performance.
- ***Techniques***: again by ***machine learning*** from examples
- According to Hutter et al. (2014): ***random forests*** are state of the art $\rightsquigarrow$ SMAC

Search techniques

- ***Blackbox function*** $f : D \to \mathbb{R}$, f random variable
- Find $\max_{x \in D} E[f(d)]$ (or other summary statistic).
- Commonly: $D = \mathbb{R}^n$

- Relation to Rice (1976)'s model: for a fixed instance x, function $f(a) = y(a, x)$ is the ***black box***.
- For algorithm configuration: $D = \mathcal{A} = A_1 \times \cdots \times A_n$ for n parameters.

Model-free versus model-based

- ***Model-free***: random search, grid search, direct (pattern) search, genetic programming
  Examples: ParamILS, Mesh adaptive direct search, Gender-based genetic algorithm
- ***Model-based***: racing (F-Race), surrogates, estimation of distribution
  Examples: Sequential Parameter Optimization (SPO), Sequential Model-Based Algorithm Configuration (SMAC), REVAC, Bonesa.

Most are ***essentially Blackbox*** (i.e. no or light assumptions on algorithmic structure)

Mesh adaptive direct search (Audet and Orban 2006)

- Adaptive mesh refinement, with a current mesh size Δ
- Repeatedly:
  - ***Search***: global search on the mesh. On success, coarsen mesh if too fine, continue.
  - ***Poll***: local search for an improving point. On success, continue.
  - ***Refine***: refine mesh.
- OPAL: Python-implementation of MADS applied to algorithm tuning (Audet, Dang, and Orban 2010).

- Genetic algorithms applied to ***evolution of algorithms***.
- Traditional responses in GP:
  - Representation: Syntax trees, often homogeneous (=expressions); grammars.
  - Initialization: grow trees.
  - Crossover: choose random subtrees (biased to internal), exchange them.
  - Mutation: substitute some subtree but a new, random one; change a node.
  - Folk wisdom: larger populations are better (Poli, Langdon, and PcPhee 2008).
- Essentially ***same difficulties*** as generic problem: representation, modification, evaluation.

Gender-based genetic algorithm (Ansótegui, Sellmann, and Tierney 2009)

- ***Individuals***: variable trees of parameters (bounded numerical or categorical), either of competing or non-competing gender.
- ***Selection***: Top 10% competing mate with random non-competing individual.
- ***Crossover***: Uniform, tree-based, with higher correlation of variables in subtrees.
- ***Mutation***: With fixed probability 0.1 uniform for categorical, Gaussian for numerical parameters.
- ***Evaluation***: Racing on N random instances, N increasing over runtime.
- Claim: better than ParamILS.

Surrogate methods (Booker et al. 1999)

- ***Other names***: metamodels, response surface models, approximation model, cheap models.
- Substitute costly evaluation of f by a surrogate function s.
- Surrogate s is a ***simplified model***.
- Repeatedly: find a minimizer x of s, evaluate $f(x)$, update surrogate s.
- Sequential Model-Based Algorithm Configuration (SMAC) (Hutter, Hoos, and Leyton-Brown 2011) is a surrogate method using random forests.

Relevance Estimation and Value Calibration (Nannen and Eiben 2007)

- An example of an ***estimation of distribution*** algorithm.
- ***Steady-state*** evolution of a population of parameter settings: recombine n best, replace oldest.
- ***Recombination***: uniform scanning.
- ***Mutation***: independent for each parameter. Sort values of all parents, substitute value in child x by $U[l, u]$, where l is the value of the h-th predecessor and u the value of the h-th successor of x. $(h \approx n/10)$
- REVAC maximizes the entropy of the marginal distributions of each parameter.

Conclusion

Conclusions and Questions

- There's ***nothing new*** under the sun.
- But we find a ***lot of different names*** for similar ideas.
- There's ***nothing specific*** to meta-heuristics (model, search space structure).
- How can we ***evaluate more aggressively***?

References I

- Ansótegui, Carlos, Meinolf Sellmann, and Kevin Tierney (2009). “A gender-based genetic algorithm for the automatic configuration of algorithms”. In: ***Principles and Practice of Constraint Programming***. Ed. by I. P. Gent. Vol. 5732. LNCS. Heidelberg, Germany: Springer, pp. 142–157. DOI: 10.1007/978-3-642-04244-7.
- Audet, C., C.-K. Dang, and D. Orban (2010). “Software Automatic Tuning: From Concepts to State-of-the-Art Results”. In: ed. by K. Naono et al. Springer. Chap. Algorithmic parameter optimization of the DFO method with the OPAL framework, pp. 255–274.

Audet, C. and D. Orban (2006). "Finding optimal algorithmic parameters using derivative-free optimization". In: ***SIAM J. Opt.*** 17.3, pp. 642–664. DOI: dx.doi.org/10.1137/040620886.

Birattari, Mauro (2005). ***Tuning Metaheuristics – A machine learning perpective***. Vol. 197. Studies in Computational Intelligence. Springer.

Booker, A. J. et al. (1999). "A rigorous framework for optimization of expensive functions by surrogates". In: ***Structural optimization*** 17.1, pp. 1–13. DOI: 1007/BF01197708.

References III

Cowling, P., G. Kendall, and E. Soubeiga (2000). "A hyperheuristic approach to scheduling a sales summit". In: ***Proceedings of the Third International Conference on the Practice And Theory of Automated Timetabling***, pp. 176–190.

Hoos, Holger H. (2012). "Programming by optimization". In: ***Comm. ACM*** 55.22. DOI: 10.1145/2076450.2076469.

– (2014). ***Programming by Optimisation (PbO)***. URL: http://www.prog-by-opt.net (visited on 11/27/2014).

Hutter, F., H. H. Hoos, and K. Leyton-Brown (2011). "Sequential model-based optimization for general algorithm configuration". In: ***Learning and Intelligent Optimization, 5th International Conference, LION 5***. Ed. by C. A. Coello Coello. Vol. 6683. LNCS. Springer, Heidelberg, Germany, pp. 507–523. DOI: 10.1007/978-3-642-25566-3_40.

Hutter, F. et al. (2014). "Algorithm runtime prediction: Methods and evaluation". In: ***Artif. Intell.*** 206, pp. 79–111.

Kotte, Lars (2016). ***Algorithm Selection literature summary***. URL: http://larskotthoff.github.io/assurvey.

References V

Nannen, Volker and A. E. Eiben (2007). "Relevance estimation and value calibration of evolutionary algorithm parameters". In: *Proc. 20th Int. Joint Conf. Art. Intell.* Hyderabad, pp. 975–980. URL: http://ijcai.org/papers07/Papers/IJCAI07-157.pdf.

Poli, Riccardo, William B. Langdon, and Nicholas F. PcPhee (2008). *A Field Guide to Genetic Programming*. Lulu.

Rice, J. R. (1976). "The algorithm selection problem". In: *Adv. Comput.* 15, pp. 65–118.

Smith-Miles, Kate and Leo Lopes (2012). "Measuring instance difficulty for combinatorial optimization problems". In: *Comput. Oper. Res.* 39.5, pp. 875–889. DOI: 10.1016/j.cor.2011.07.006.

References VI

Smith-Miles, Kate et al. (2014). "Towards objective measures of algorithm performance across instance space". In: *Comput. Oper. Res.* 45, pp. 12–24. DOI: 10.1016/j.cor.2013.11.015.