On the Automatic Configuration of Algorithms

Marcelo de Souza
marcelo.desouza@udesc.br

Santa Catarina State University, Brazil
Federal University of Rio Grande do Sul, Brazil

Doctoral Consortium
24th European Conference on Artificial Intelligence (ECAI 2020)
Santiago de Compostela, Spain

Parameterized algorithms

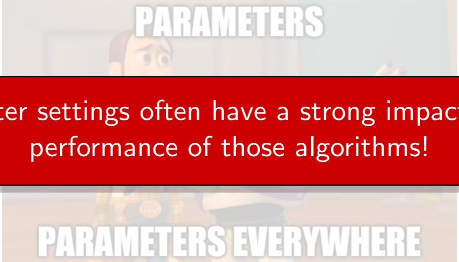
Many algorithms expose parameters to control their internal behavior.

- **Exact solvers:** CPLEX (63), Gurobi (25), SCIP (200+).
- **Heuristic solvers:** ACOTSP (11), HHBQP (14), AACFS (41).
- **Machine learning:** Weka (768).
- **Compilers:** GCC (367).

Parameterized algorithms

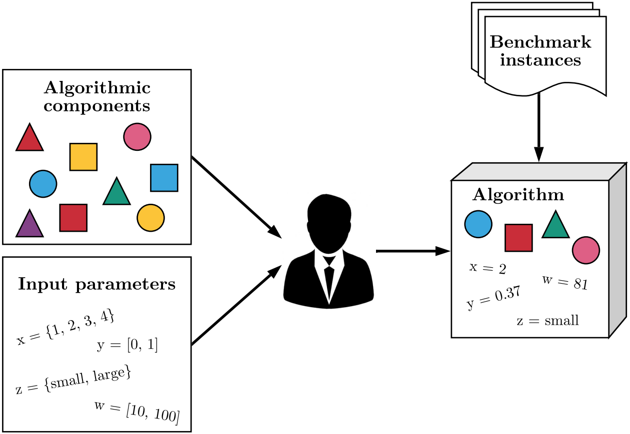
Many algorithms expose parameters to control their internal behavior.

- **Exact solvers:** CPLEX (63), Gurobi (25), SCIP (200+).
- **Heuristic solvers:** ACOTSP (11), HHBQP (14), AACFS (41).
- **Machine learning:** Weka (768).
- **Compilers:** GCC (367).

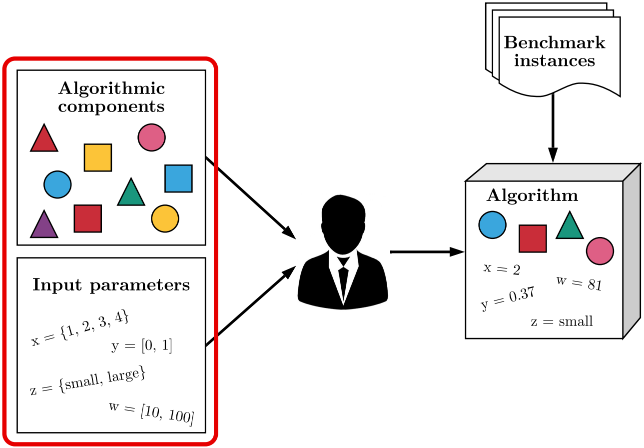


Parameter settings often have a strong impact on the performance of those algorithms!

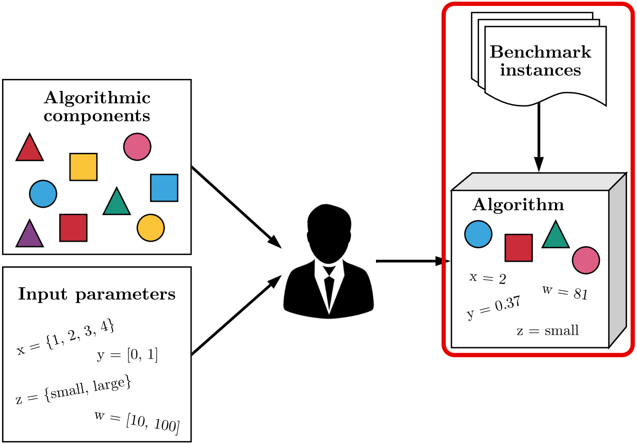
Algorithm configuration



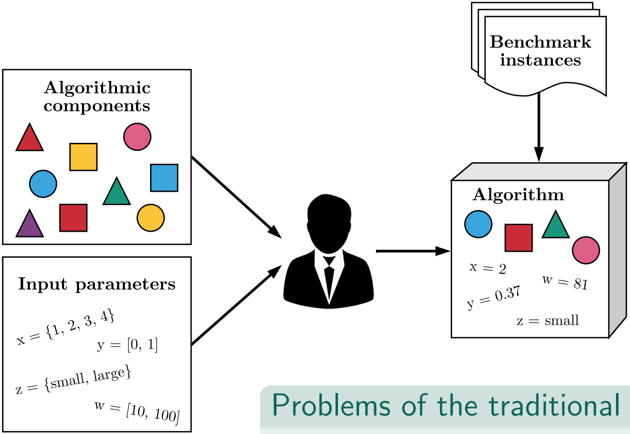
Algorithm configuration



Algorithm configuration



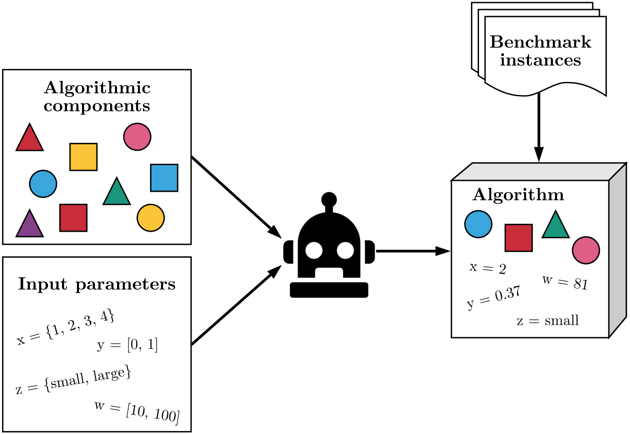
Algorithm configuration



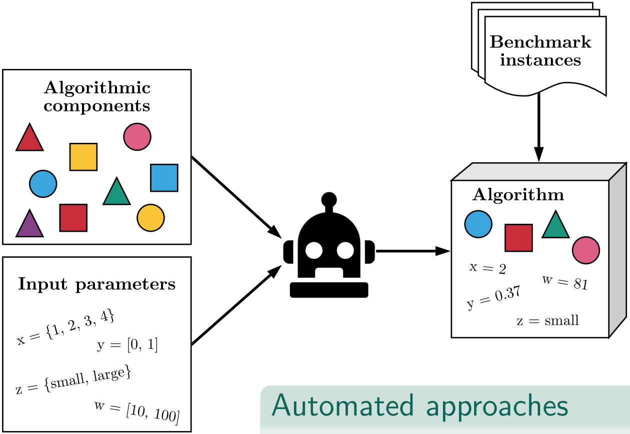
Problems of the traditional approach

- Exploration of limited design alternatives,
- Human bias,
- Non-reproducible.

Automatic algorithm configuration



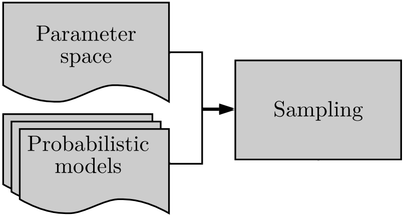
Automatic algorithm configuration



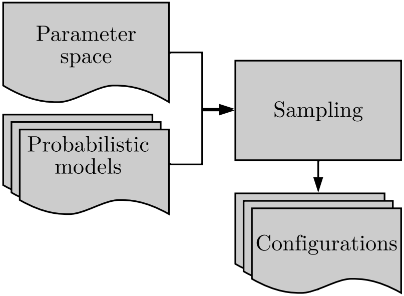
Automated approaches

- Heuristic search: **ParamILS** [1], **GGA** [2].
- Model-based methods: **SMAC** [3], **GGA++** [4].
- Racing methods: **iterated F-race** [5], **irace** [6].

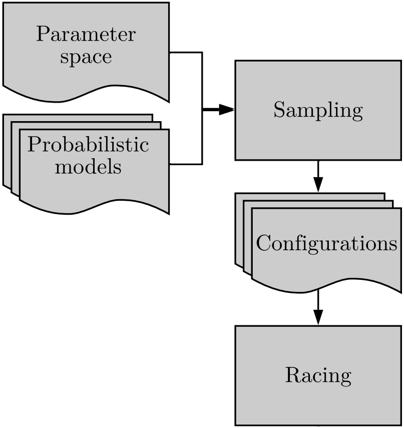
Automatic algorithm configuration with irace



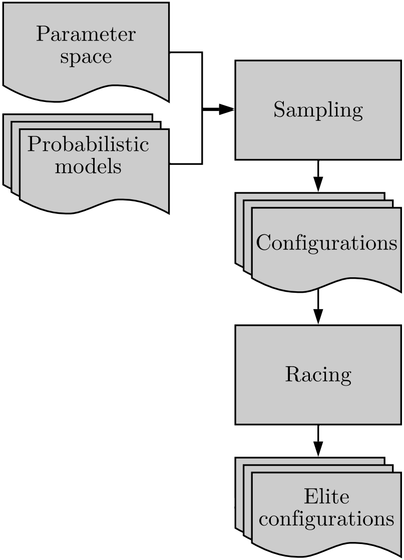
Automatic algorithm configuration with irace



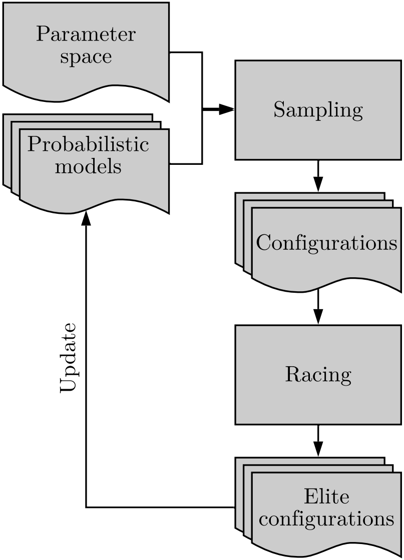
Automatic algorithm configuration with irace



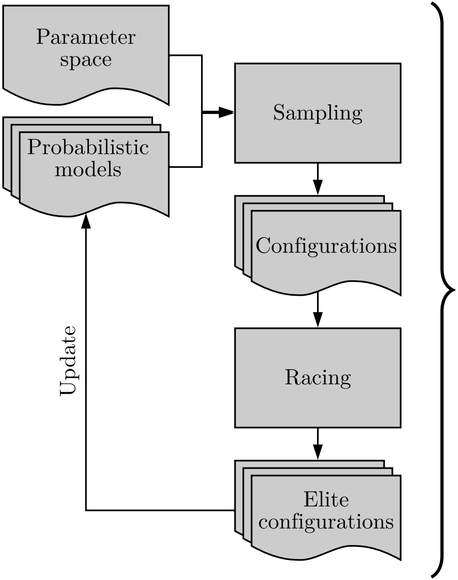
Automatic algorithm configuration with irace



Automatic algorithm configuration with irace

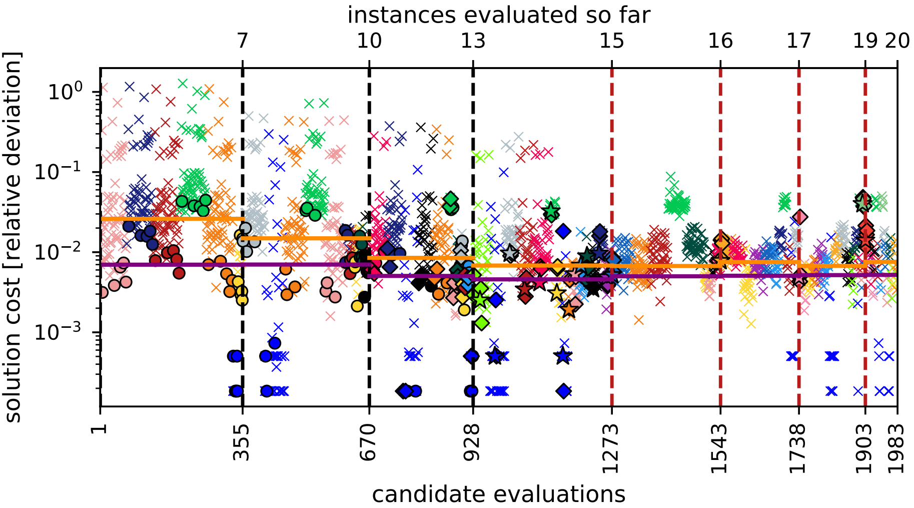


Problem 1: how do we analyze the configuration?

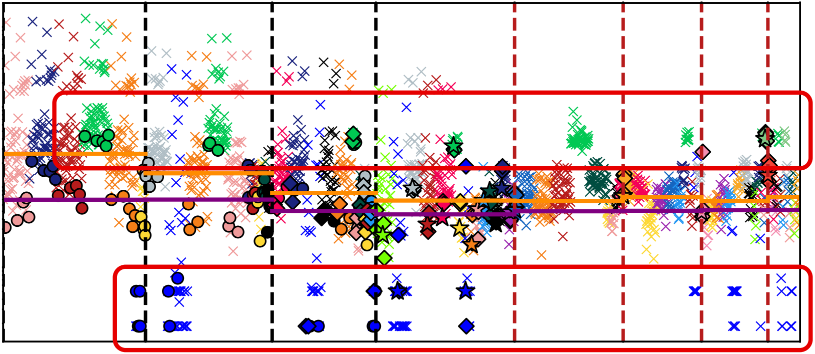


Hard to analyze an execution;
often used as a black box.

Configuration Analysis Tools (cat)

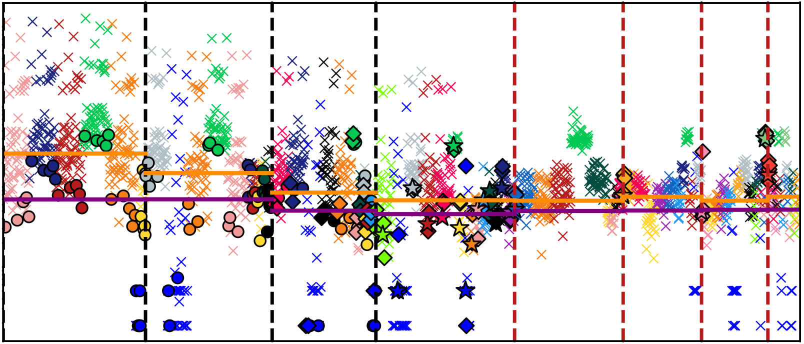


Configuration Analysis Tools (cat)



Too easy and too hard instances do not contribute to differentiate configurations!

Configuration Analysis Tools (cat)

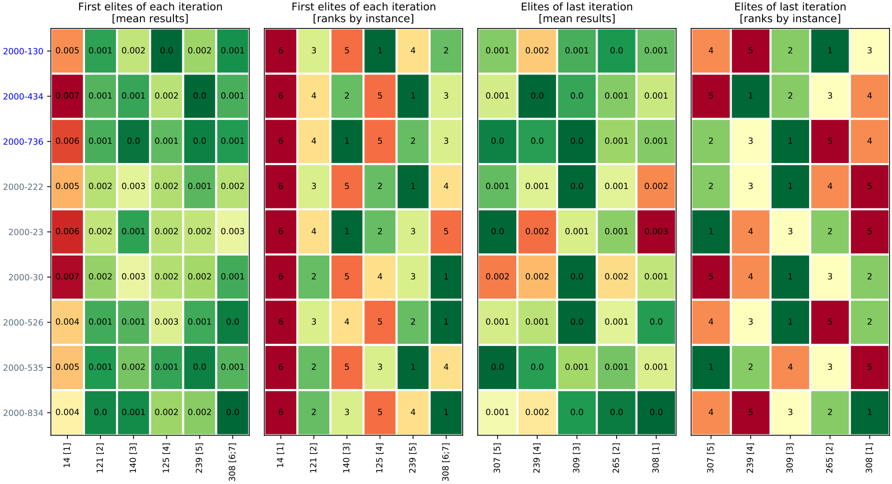


Performance does not improve and soft restart is made!

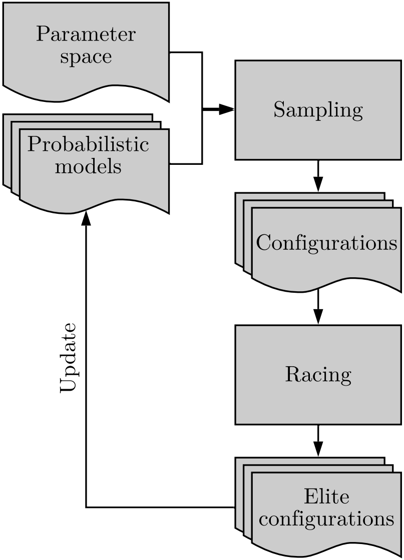
Configuration Analysis Tools (cat)

- **cat** is available at https://github.com/souzamarcelo/cat.
  - Source code, additional plots and features, examples.
- Next steps:
  - Apply cat to analyze several configuration scenarios.
  - List common mistakes in the design of configuration scenarios and how to identify them using cat.
  - **New visualizations!**

Configuration Analysis Tools (cat)



Problem 2: how do we speed up the process?



Racing is very **time consuming**: many configurations executed on several instances.

Capping methods for optimization scenarios

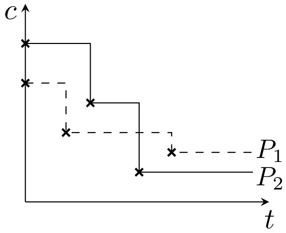
- **What if we use the results of previously seen executions to evaluate the new ones?** Then we can stop early poorly performing executions and **save time**!
  - Similar approaches were previously applied to configure decision algorithms [1; 7], but they are not suitable for optimization scenarios.

Capping methods for optimization scenarios

- **What if we use the results of previously seen executions to evaluate the new ones?** Then we can stop early poorly performing executions and **save time**!
  - Similar approaches were previously applied to configure decision algorithms [1; 7], but they are not suitable for optimization scenarios.
- General idea:
  - A performance profile is a function $P(t) = c$, where c is the cost of the best found solution after executing the algorithm for a time t.
  - Given the executions of elite configurations on the current instance:
    - We aggregate the corresponding profiles $\rightarrow$ **performance envelope**.
    - Use the envelope to evaluate (and maybe cap) the current execution.

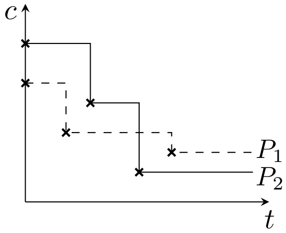
Profile-based envelope

The envelope is also a performance profile:



Profile-based envelope

The envelope is also a performance profile:

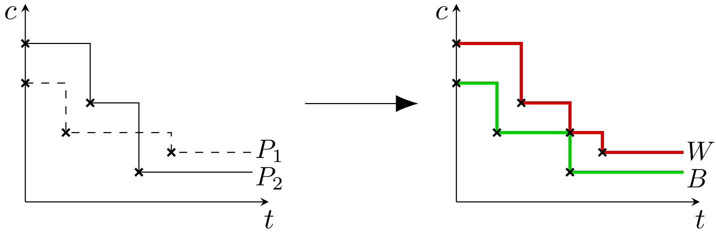


Aggregation functions:

- Worst: $W(P_1, \ldots, P_n)(t) = \max\{P_1(t), \ldots, P_n(t)\}$.
- Best: $B(P_1, \ldots, P_n)(t) = \min\{P_1(t), \ldots, P_n(t)\}$.

Profile-based envelope

The envelope is also a performance profile:

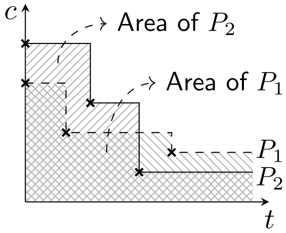


Aggregation functions:

- Worst: $W(P_1, \ldots, P_n)(t) = \max\{P_1(t), \ldots, P_n(t)\}$.
- Best: $B(P_1, \ldots, P_n)(t) = \min\{P_1(t), \ldots, P_n(t)\}$.

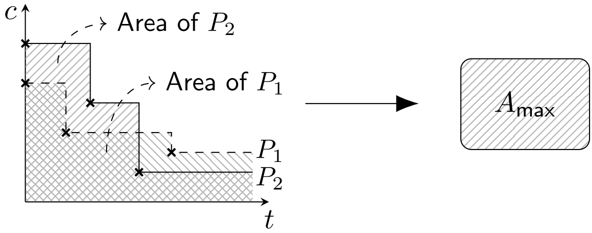
Area-based envelope

The envelope is a maximum allowed area:



Area-based envelope

The envelope is a maximum allowed area:



Aggregation functions:

- Worst: $W(P_1, \ldots, P_n) = \max \{A_{P_1}, \ldots, A_{P_n}\}$.
- Best: $B(P_1, \ldots, P_n) = \min \{A_{P_1}, \ldots, A_{P_n}\}$.

Capping methods for optimization scenarios

Mean effort savings and mean deviation for each capping method.

Cap.	ACOTSP		HEACOL		TSBPP		HHBQP	
	sav.	r. dev.	sav.	r. dev.	sav.	r. dev.	sav.	a. dev.
-	-	**0.33**	-	**4.14**	-	1.31	-	49.72
PW	59.7	0.37	61.3	4.22	22.6	**1.25**	44.3	65.16
PB	**77.7**	0.52	**74.8**	4.48	38.1	1.27	**74.9**	58.38
AW	26.8	0.35	27.2	4.18	12.4	1.28	17.3	**46.97**
AB	52.7	0.38	47.0	4.18	**41.4**	1.35	65.9	68.56

- ACOTSP: ant colony optimization for the travelling salesperson problem.
- HEACOL: hybrid evolutionary algorithm for the graph coloring.
- TSBPP: tabu search for the bin packing problem.
- HHBQP: hybrid heuristic for the unconstrained binary quadratic programming.

Capping methods for optimization scenarios

Mean effort savings and mean deviation for each capping method.

Cap.	ACOTSP		HEACOL		TSBPP		HHBQP	
	sav.	r. dev.	sav.	r. dev.	sav.	r. dev.	sav.	a. dev.
-	-	**0.33**	-	**4.14**	-	1.31	-	49.72
PW	59.7	0.37	61.3	4.22	22.6	**1.25**	44.3	65.16
PB	**77.7**	0.52	**74.8**	4.48	38.1	1.27	**74.9**	58.38
AW	26.8	0.35	27.2	4.18	12.4	1.28	17.3	**46.97**
AB	52.7	0.38	47.0	4.18	**41.4**	1.35	65.9	68.56

We save some time: from 12% to 77%.

Capping methods for optimization scenarios

Mean effort savings and mean deviation for each capping method.

Cap.	ACOTSP		HEACOL		TSBPP		HHBQP	
	sav.	r. dev.	sav.	r. dev.	sav.	r. dev.	sav.	a. dev.
-	-	**0.33**	-	**4.14**	-	1.31	-	49.72
PW	59.7	0.37	61.3	4.22	22.6	**1.25**	44.3	65.16
PB	**77.7**	0.52	**74.8**	4.48	38.1	1.27	**74.9**	58.38
AW	26.8	0.35	27.2	4.18	12.4	1.28	17.3	**46.97**
AB	52.7	0.38	47.0	4.18	**41.4**	1.35	65.9	68.56

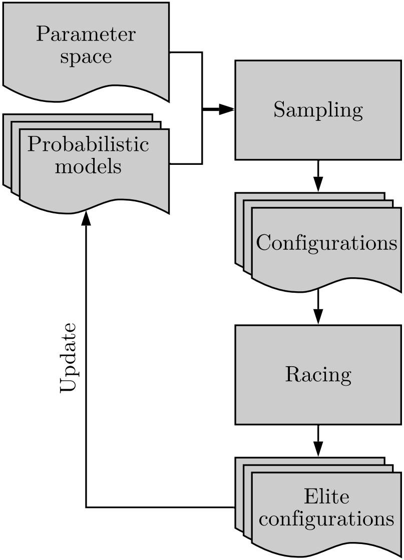
We save some time: from 12% to 77%.

The resulting configurations have comparable quality.

Capping methods for optimization scenarios

- The capping methods are implemented in the **capopt** package.
- **capopt** is available at `https://capopt.github.io`.
  - Source code, all details, quick start, experimental data.
- Next steps:
  - New methods: model-based and adaptive approaches.
  - Experiments: apply capping for scenarios with a budget on the configuration time, allowing irace to use the saved time to further explore the parameter space.

Problem 3: how do we avoid premature convergence?



Premature convergence: new configurations are very similar to those already evaluated.

Convergence detection

Current convergence detection method:

- After the sampling phase...
  - Compute the distance between elites and each offspring configuration;
  - If any distance is less than a threshold, convergence is detected and the associated probabilistic models are restarted.

Convergence detection

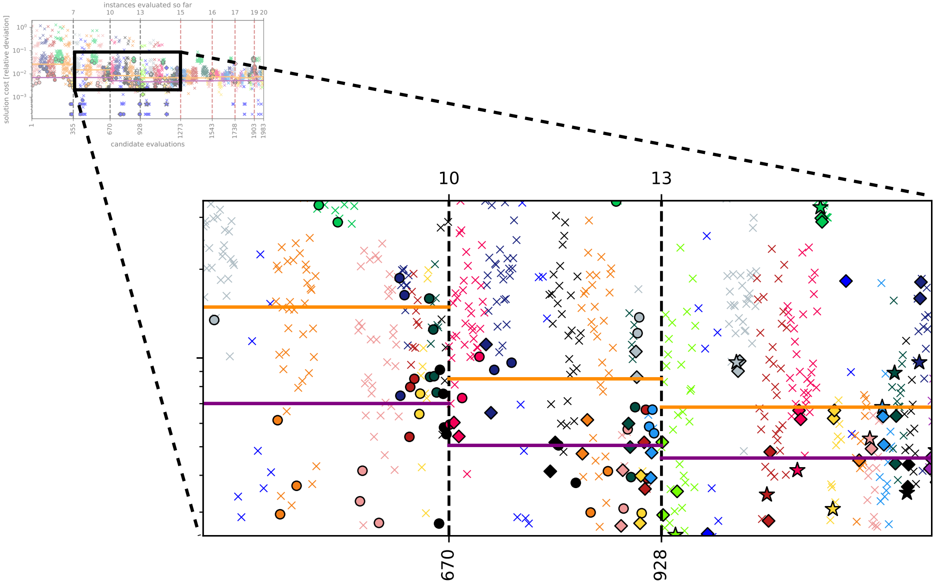
Current convergence detection method:

- After the sampling phase...
  - Compute the distance between elites and each offspring configuration;
  - If any distance is less than a threshold, convergence is detected and the associated probabilistic models are restarted.

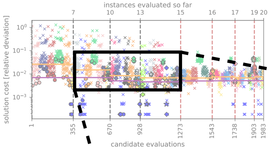
However, it often fails to detect convergence:

- Very similar configurations (with almost the same performance) may still have distance greater than the threshold.
- **What if we also consider the performance of the configurations?**

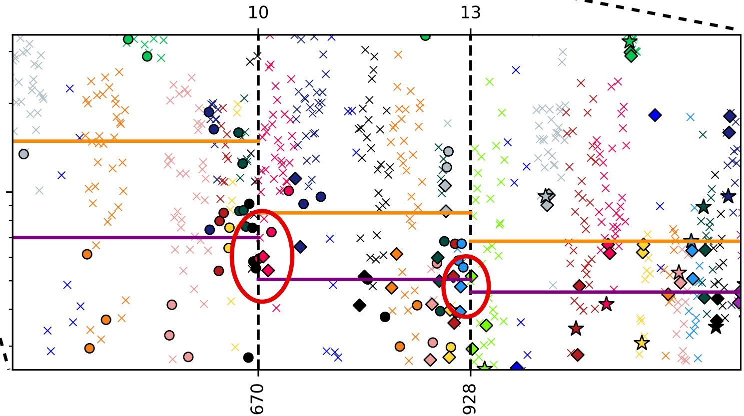
Convergence detection



Convergence detection



Convergence: when the difference in the performances is less than a threshold.



Convergence detection

Preliminary tests:

- This basic approach is able to identify convergence in cases when the default method does not detect.
- Both methods can (and should) be combined for better results.

Next steps:

- Extend our experiments and analyze the best values for the threshold.
- Detect convergence based on the evolution of the probabilistic models (e.g. how the model parameters change over the iterations).

Summary

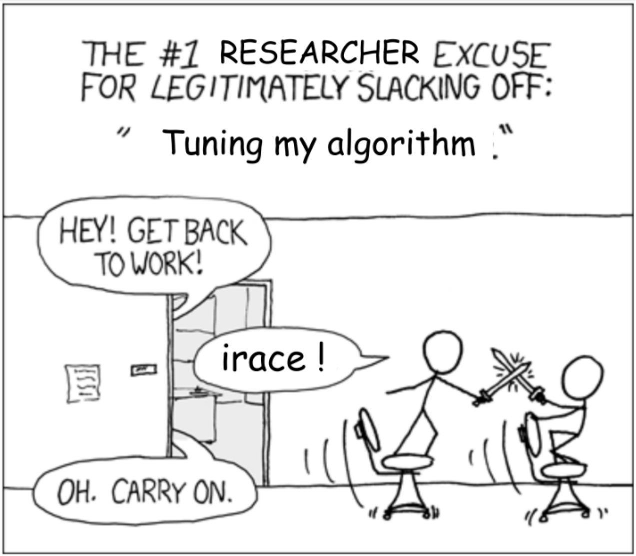
Main contributions:

- Visualizations (cat): https://github.com/souzamarcelo/cat.
- Capping (capopt): https://capopt.github.io.
- Good results so far!

Do you work with automatic algorithm configuration (irace)?

- Try using cat to analyze the results!
- Apply capopt to speed up the configuration!
- Share with me your experience, suggestions and ideas for collaboration!

Thank you!



Coppied from Manuel López-Ibáñez!

References

[1] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, October 2009.

[2] Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In Ian P. Gent, editor, *Principles and Practice of Constraint Programming, CP 2009*, volume 5732 of *Lecture Notes in Computer Science*, pages 142–157. Springer, Heidelberg, Germany, 2009.

[3] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In Carlos A. Coello Coello, editor, *Learning and Intelligent Optimization, 5th International Conference, LION 5*, volume 6683 of *Lecture Notes in Computer Science*, pages 507–523. Springer, Heidelberg, Germany, 2011.

[4] Carlos Ansótegui, Yuri Malitsky, Horst Samulowitz, Meinolf Sellmann, and Kevin Tierney. Model-based genetic algorithms for algorithm configuration. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI-15)*, pages 733–739. IJCAI/AAAI Press, Menlo Park, CA, 2015.

[5] Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle. F-race and iterated F-race: An overview. In Thomas Bartz-Beielstein, Marco Chiarandini, Luís Paquete, and Mike Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336. Springer, Berlin, Germany, 2010.

[6] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[7] Leslie Pérez Cáceres, Manuel López-Ibáñez, Holger H. Hoos, and Thomas Stützle. An experimental study of adaptive capping in irace. In Roberto Battiti, Dmitri E. Kvasov, and Yaroslav D. Sergeyev, editors, *Learning and Intelligent Optimization, 11th International Conference, LION 11*, volume 10556 of *Lecture Notes in Computer Science*, pages 235–250. Springer, Cham, Switzerland, 2017.