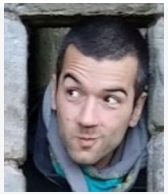
LEVERHULME TRUST

University of St Andrews

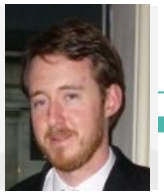
A constraint-based tool for generating benchmark instances

Nguyen Dang
University of St Andrews, UK
nttd@st-andrews.ac.uk


Ozgur Agkun


Joan Espasa


Ian Miguel


Peter Nightingale


Andras Salamon


Patrick Spracklen


Christopher Stone

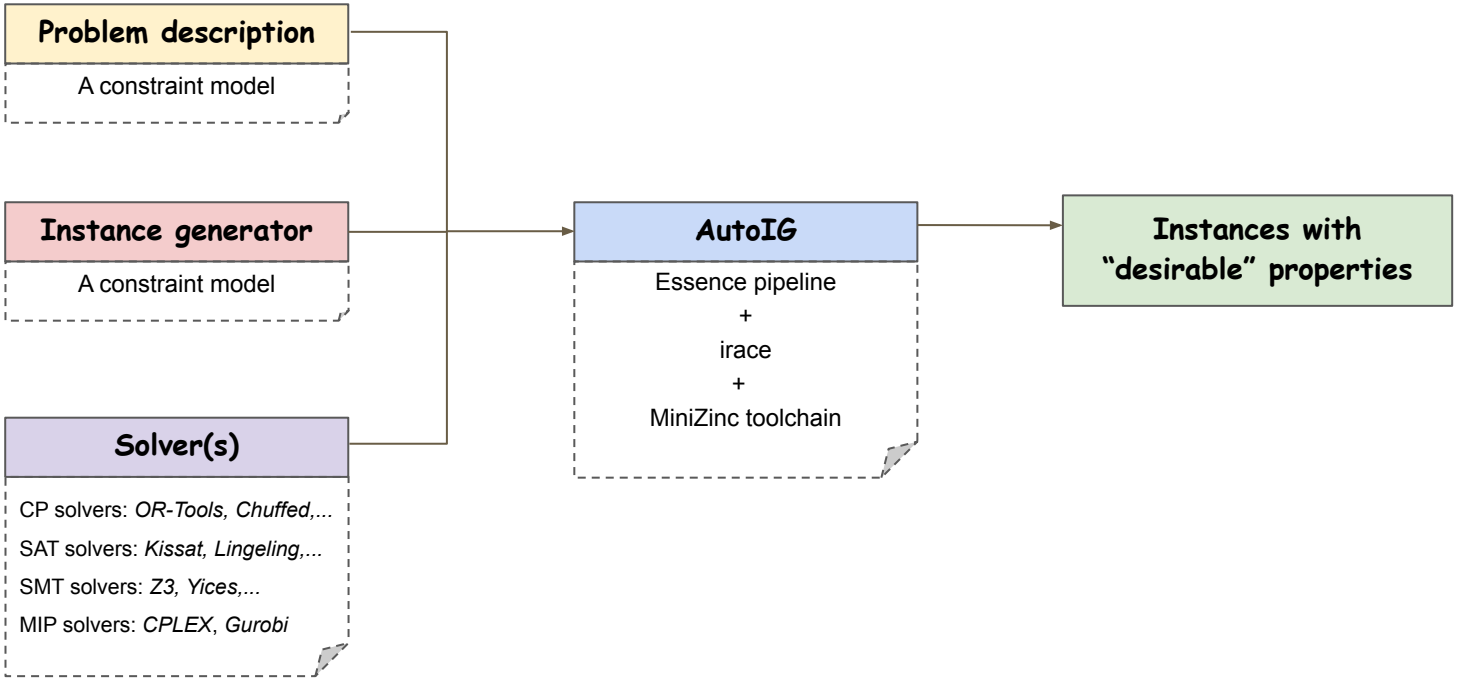
- Access to a wide range of instances is often needed when developing algorithms or comparing existing ones.
    - Testing, debugging, and evaluating algorithm performance.
    - Tuning algorithm parameters.
    - Gaining insights into strength and weakness of algorithms.
    - Building a portfolio of algorithms with complementary strengths.

- Access to a wide range of instances is often needed when developing algorithms or comparing existing ones.
    - Testing, debugging, and evaluating algorithm performance.
    - Tuning algorithm parameters.
    - Gaining insights into strength and weakness of algorithms.
    - Building a portfolio of algorithms with complementary strengths.
- Standard benchmark instance libraries are very useful, but the re-use of the same libraries over a long period of time is not always ideal.
    - Potential bias and overfitting issues.
    - Some old benchmarks are no longer challenging.

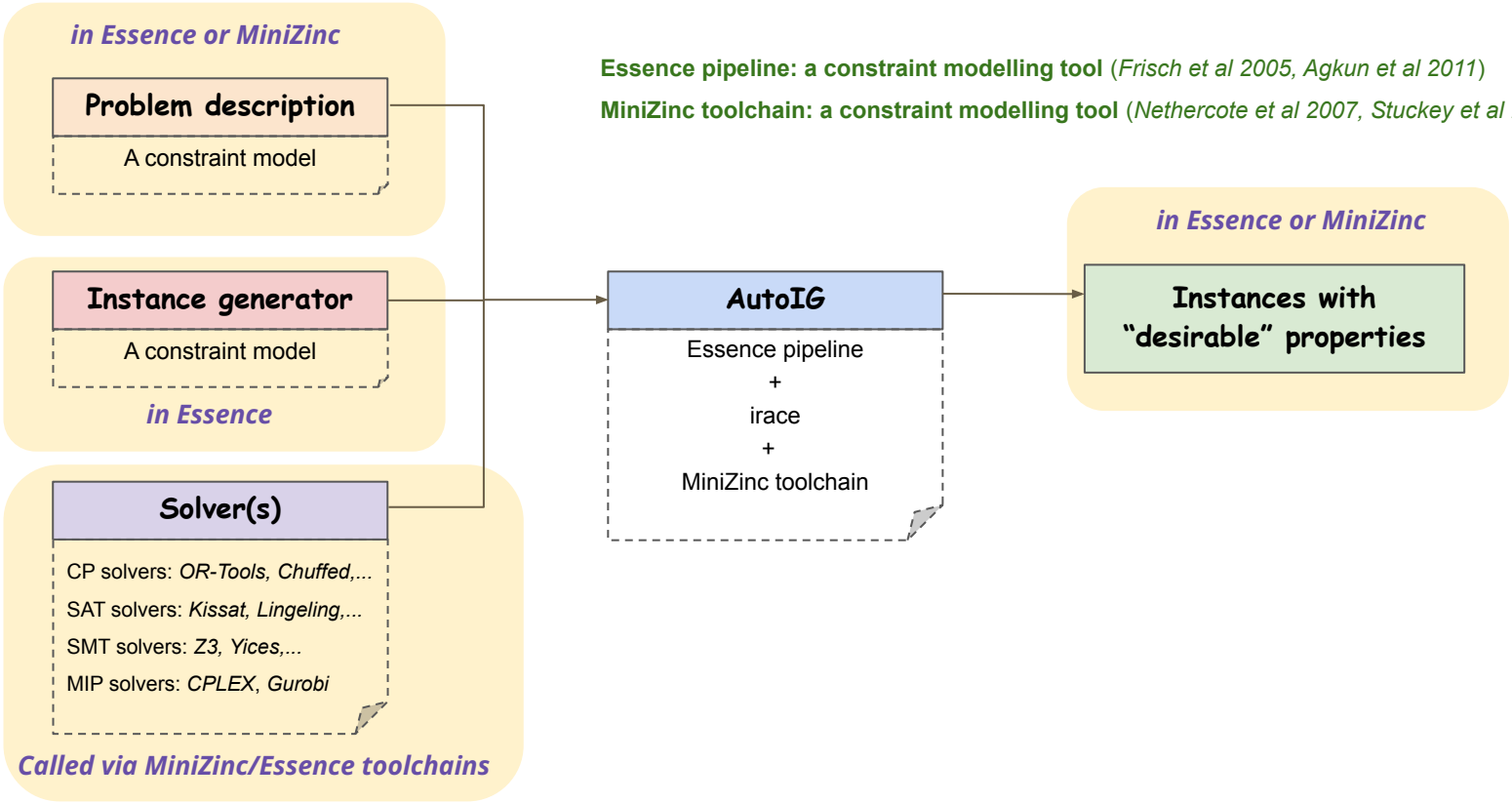
- Benchmark instance generation:
  - Nurse Rostering problems (*Vanhoucke and Maenhout, 2009*)
  - Knapsack problems (*Pisinger, 2005; Smith-Miles, Christiansen & Muñoz, 2021; Jooken, Leyman & De Causmaecker, 2022*)
  - Travelling Salesman Problems (*van Hemert 2006, Bossek et al 2019*)
  - AI Planning (*Torralba, Seipp & Sievers 2021*)
  - …

- Instance Space Analysis:
  - Machine Learning tasks
    - Classification (*Muñoz, Villanova, Baatar & Smith-Miles, 2018*), Regression (*Muñoz et al 2021*), Clustering (*Fernandes, Lorena & Smith-Miles 2021*), …
  - Combinatorial optimisation problems
    - Personnel scheduling (*Kletzander, Musliu & Smith-Miles, 2021*), Bin packing (*Liu, Smith-Miles, & Costa 2020*), Course timetabling (*Coster et al 2021*), Knapsack (*Smith-Miles, Christiansen & Muñoz, 2021*), …

AutoIG: a constraint-based instance generation tool



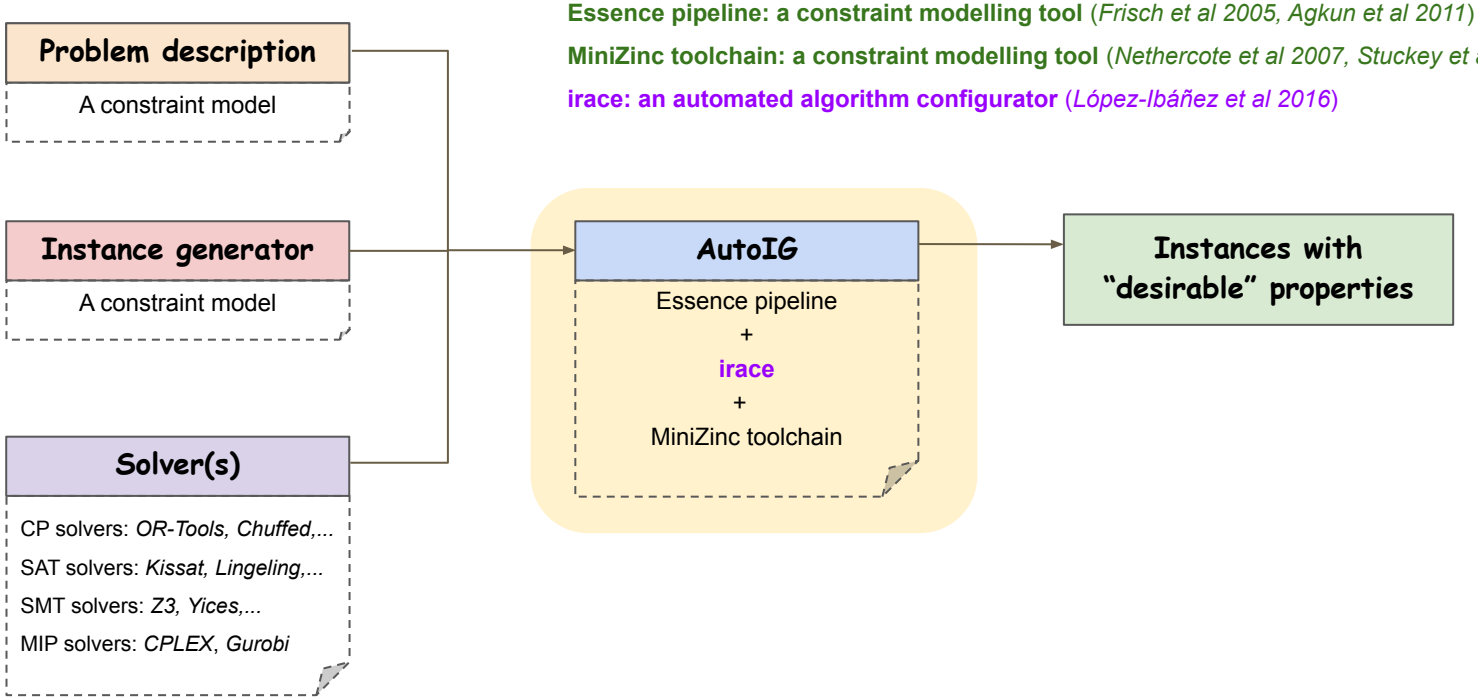
AutoIG: a constraint-based instance generation tool



Essence pipeline: a constraint modelling tool (*Frisch et al 2005, Agkun et al 2011*)

MiniZinc toolchain: a constraint modelling tool (*Nethercote et al 2007, Stuckey et al 2014*)

AutoIG: a constraint-based instance generation tool

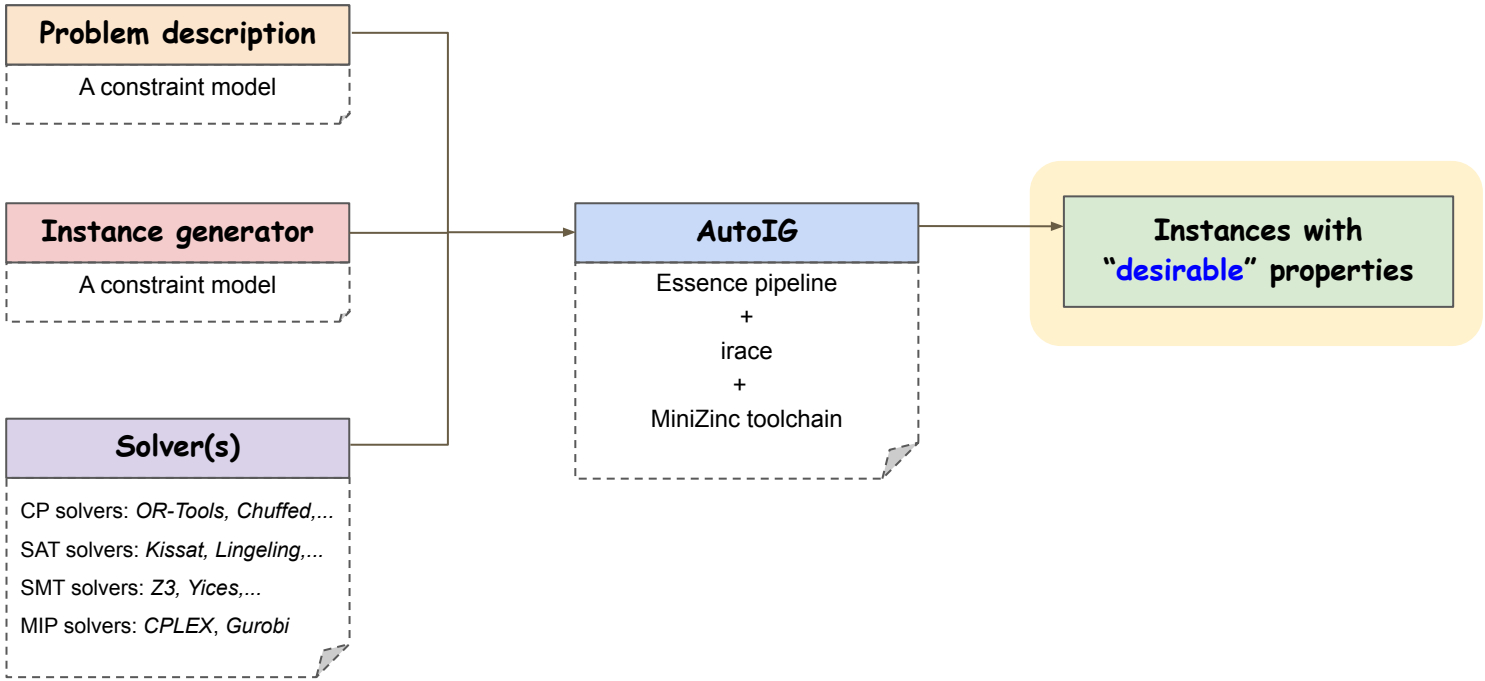


Essence pipeline: a constraint modelling tool (*Frisch et al 2005, Agkun et al 2011*)

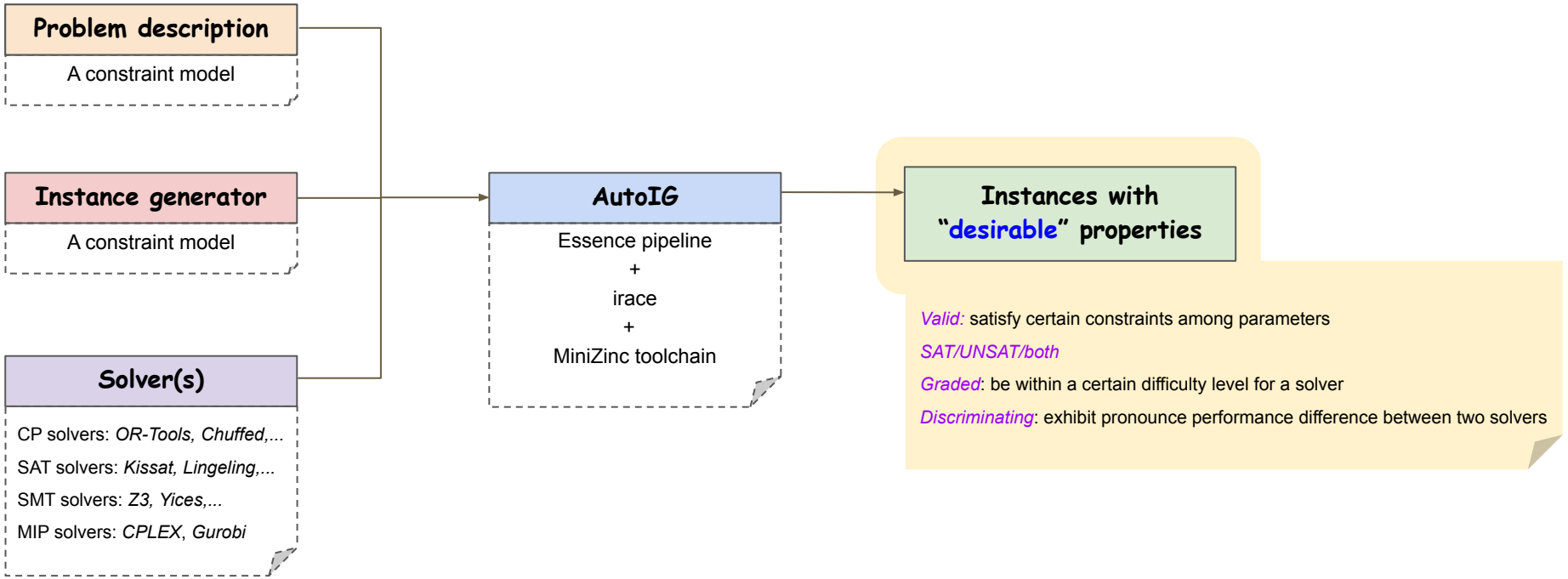
MiniZinc toolchain: a constraint modelling tool (*Nethercote et al 2007, Stuckey et al 2014*)

irace: an automated algorithm configurator (*López-Ibáñez et al 2016*)

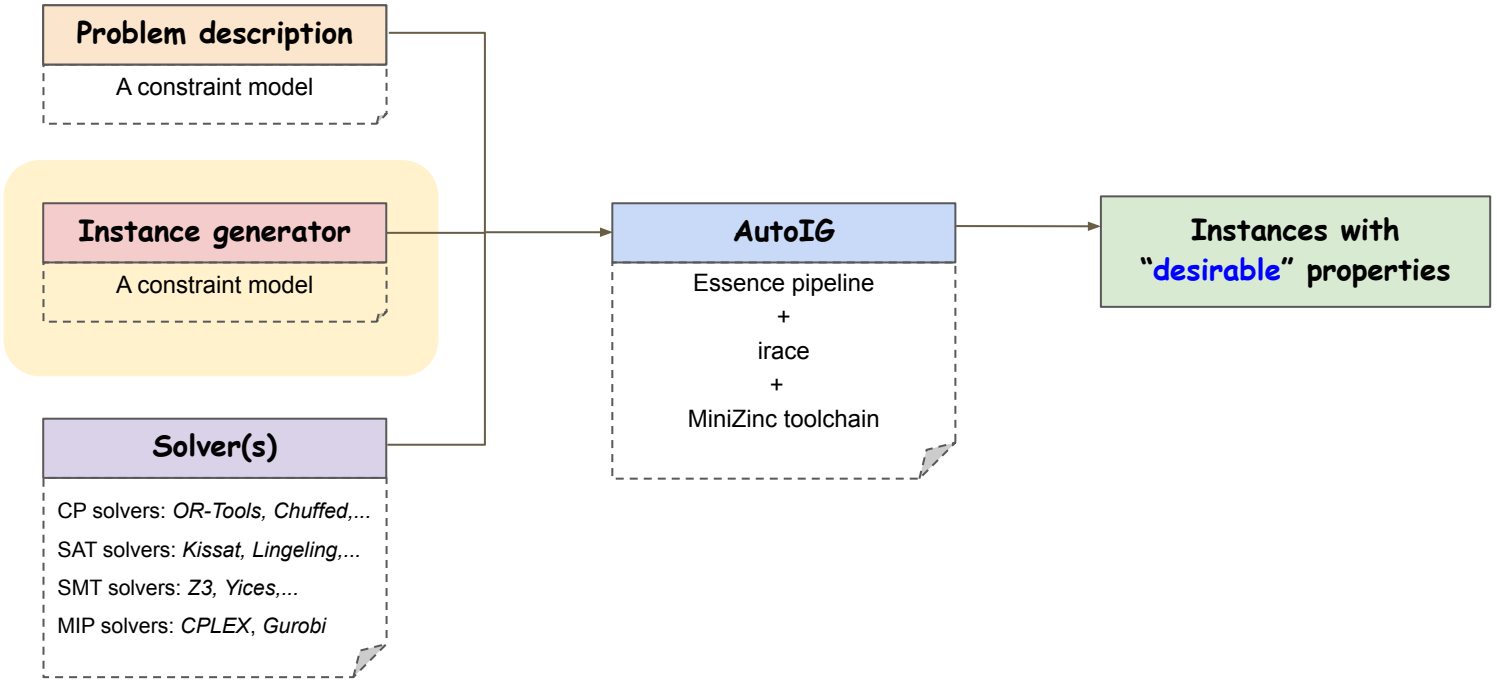
AutoIG: a constraint-based instance generation tool



AutoIG: a constraint-based instance generation tool



AutoIG: a constraint-based instance generation tool



Declarative instance generator

- *Users describe the generator and its parameters as a constraint model* (in Essence)
- *Instances are generated by a constraint solver* (minion*: Gent, Jefferson & Miguel 2006*)
  - According the parameter setting of the generator
  - Satisfying **validity constraints** among *instance parameters*

Declarative instance generator

- *Users describe the generator and its parameters as a constraint model* (in Essence)
- *Instances are generated by a constraint solver* (minion*: Gent, Jefferson & Miguel 2006*)
  - According the parameter setting of the generator
  - Satisfying **validity constraints** among *instance parameters*

A nurse rostering generator model

Parameters:
- number of nurses
- ratio of nurses for each skill
- min/max/average of daily demand

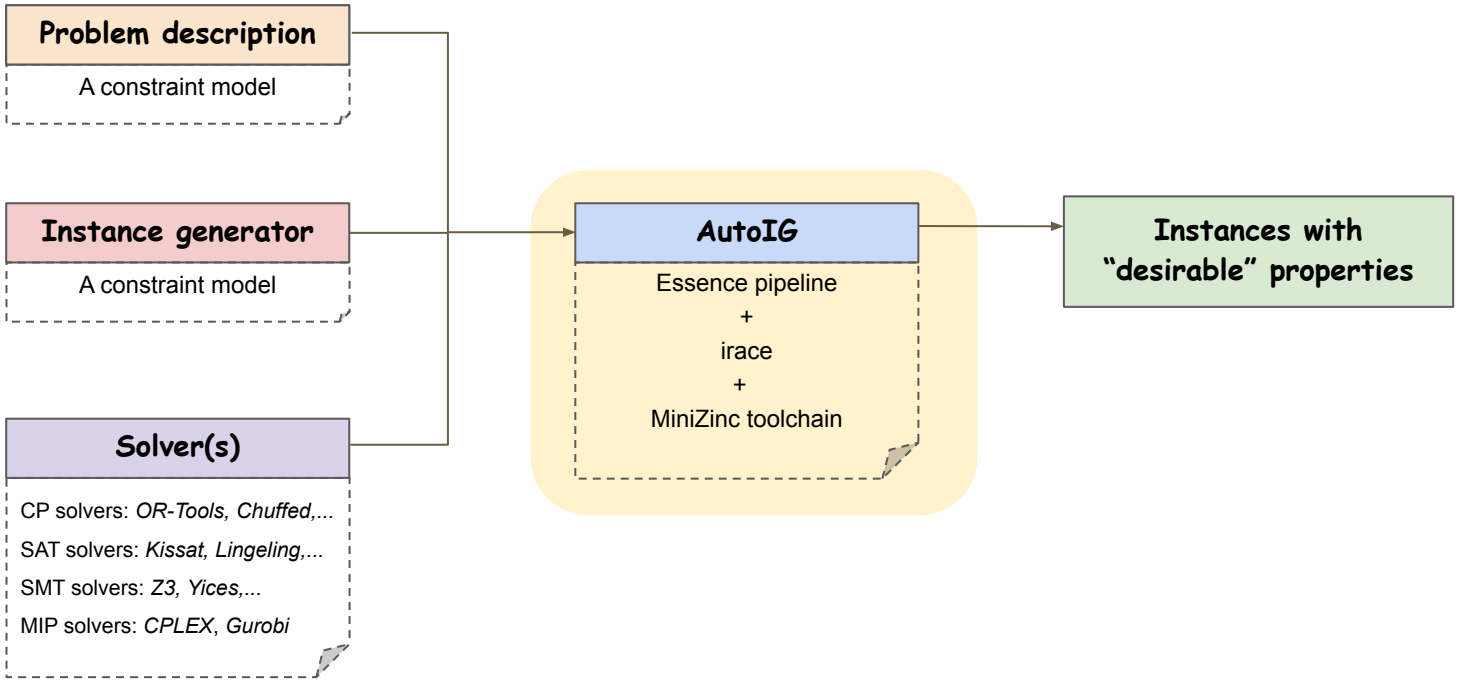
Decision variables (~ instance parameters)
- a set of nurses & their skills & preference
- demand for each day, shift and skill

Constraints (~ instance validity)
- patterns on nurse preference: day shifts vs night shifts, weekdays vs weekends,...
- patterns on daily demand: head nurses vs nurses vs trainees

tuned by irace

solved by a constraint solver

AutoIG: a constraint-based instance generation tool

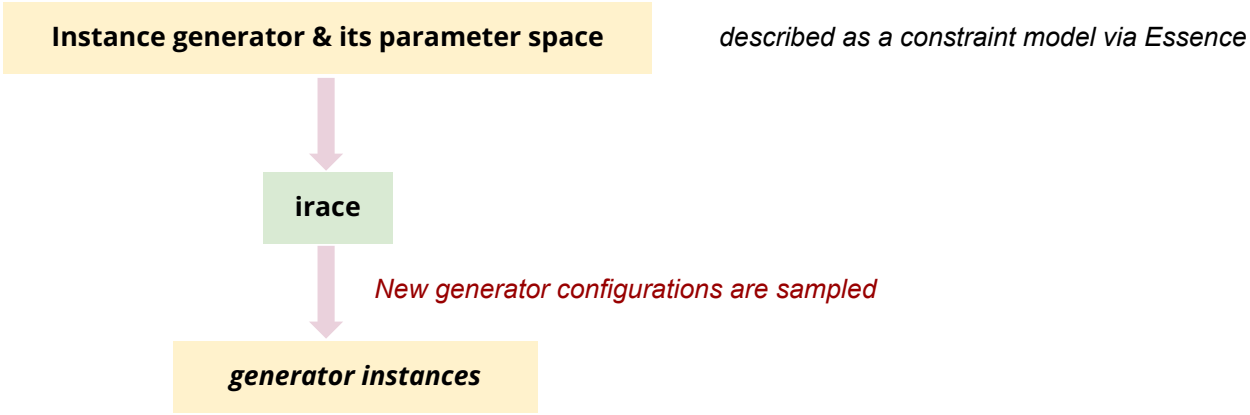


AutoIG instance generation process

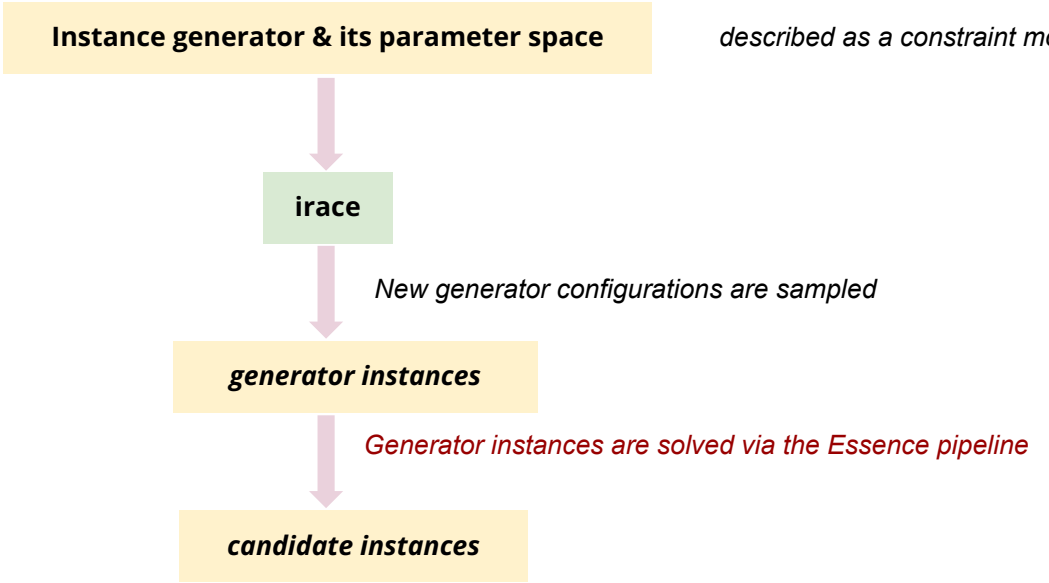
Instance generator & its parameter space

described as a constraint model via Essence

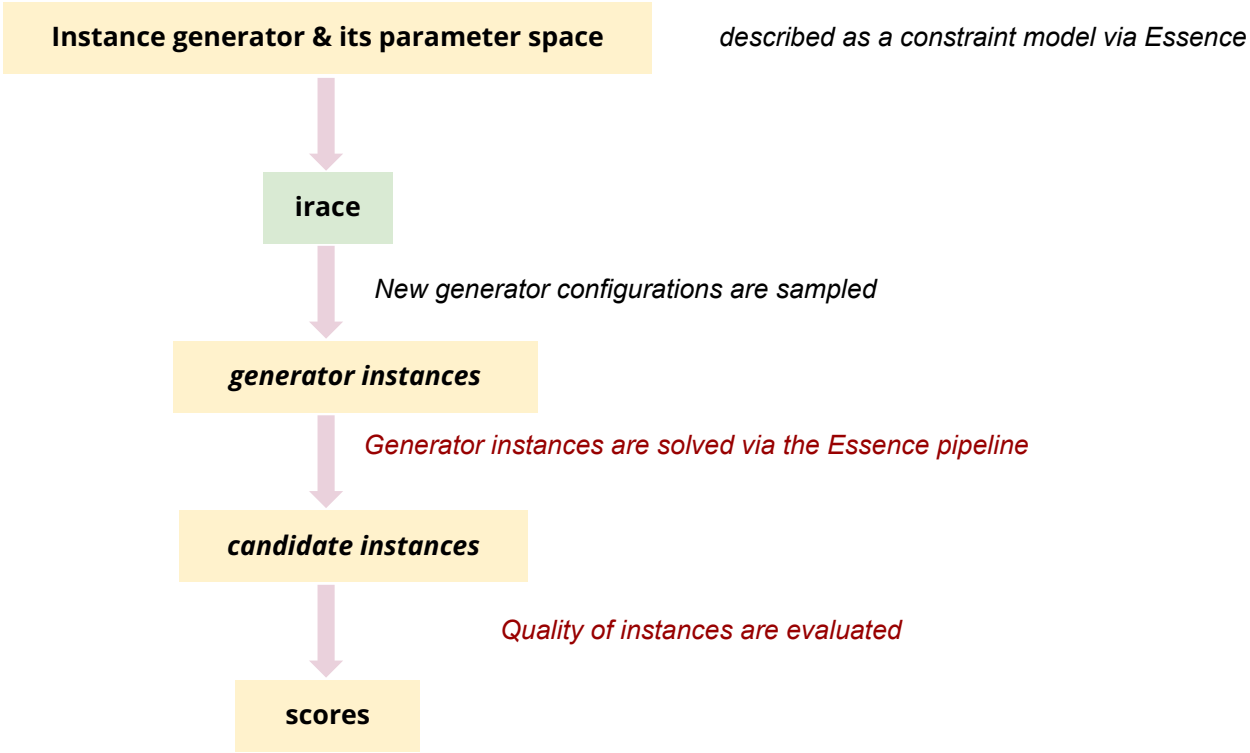
AutoIG instance generation process



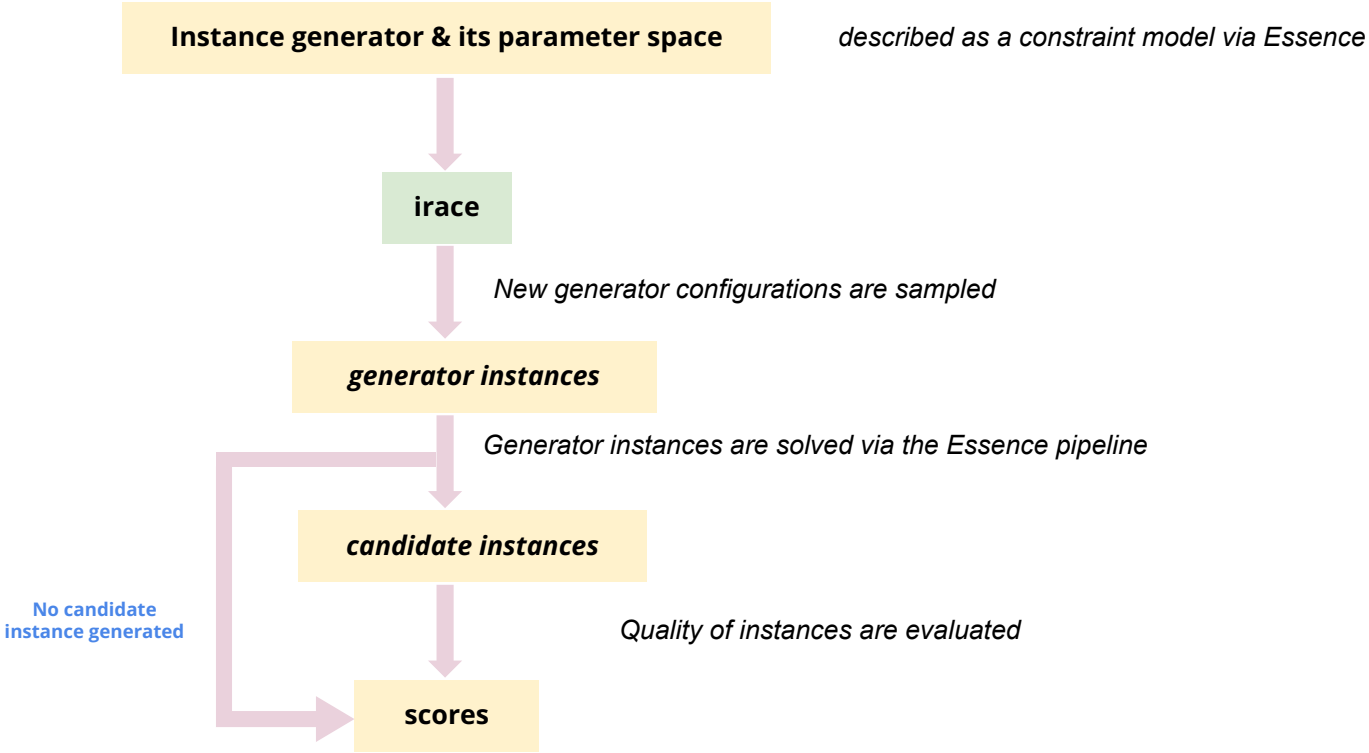
AutoIG instance generation process



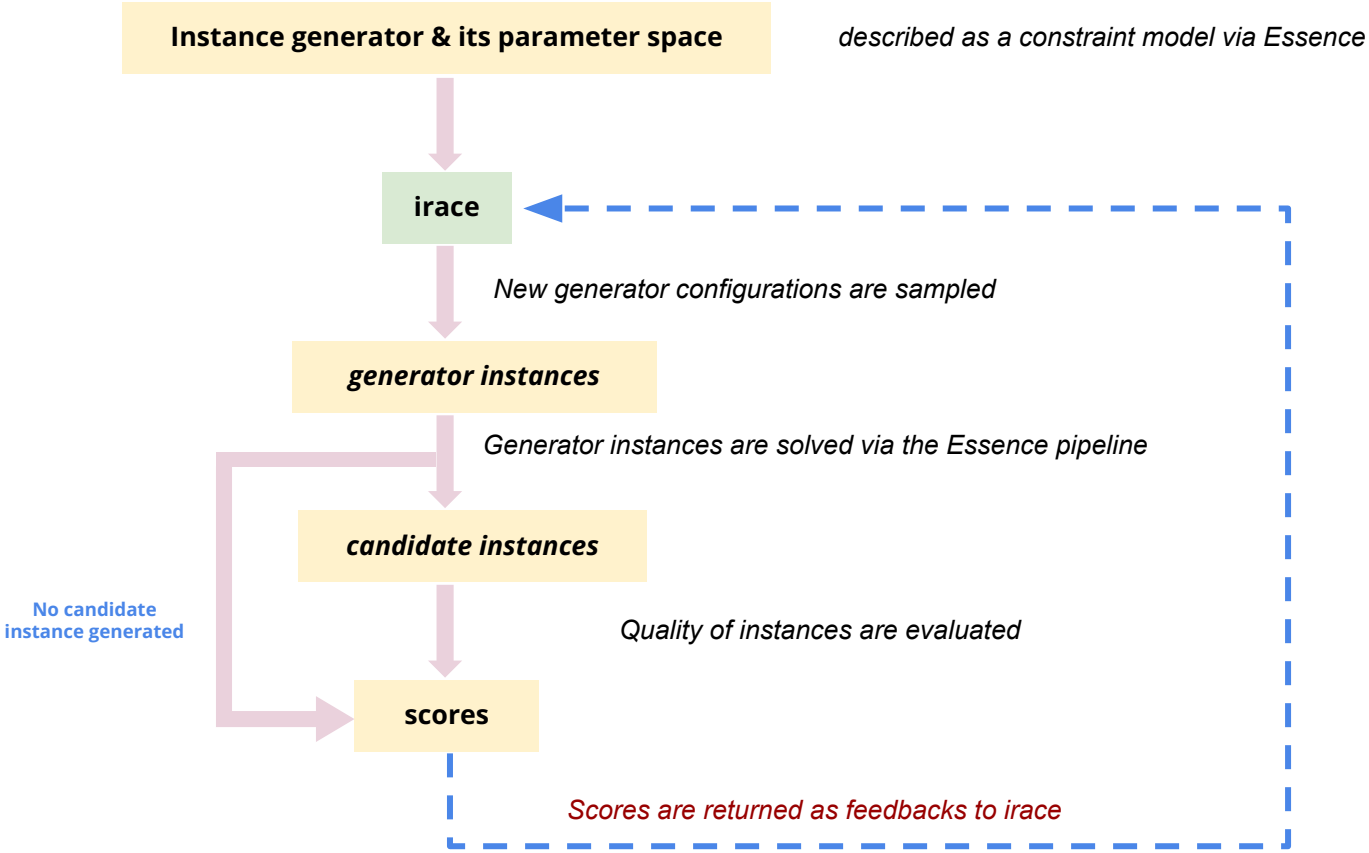
AutoIG instance generation process



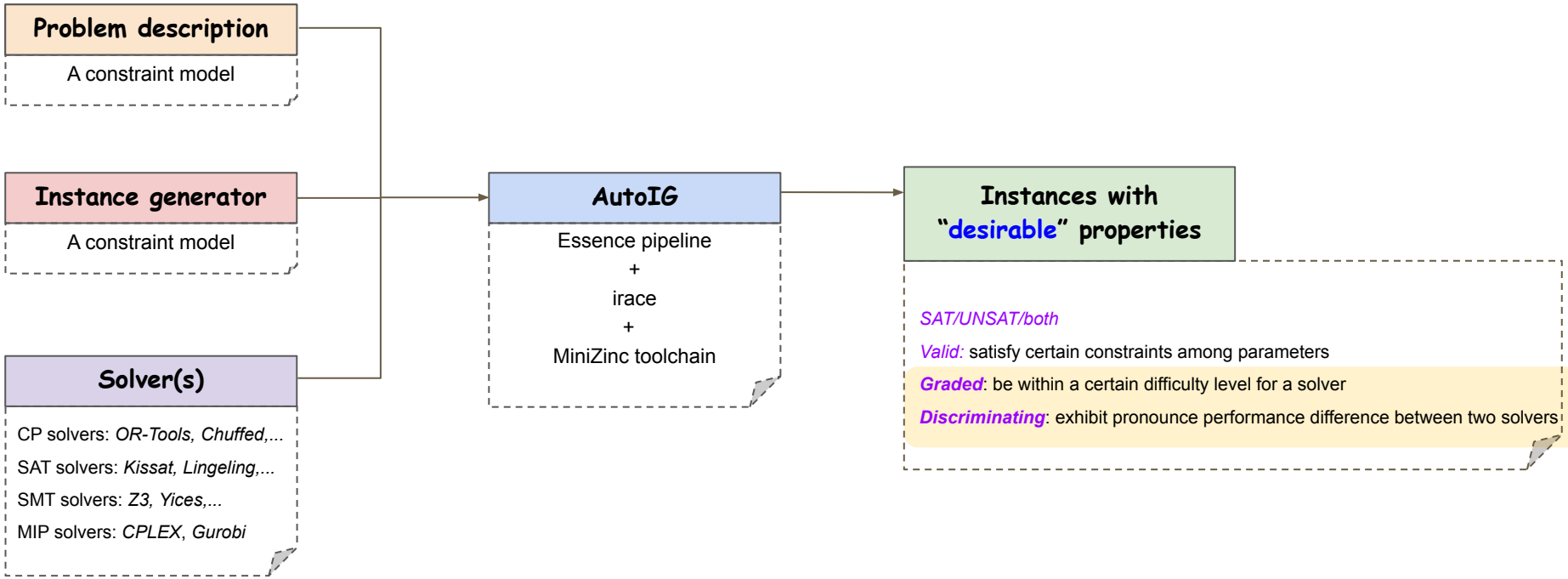
AutoIG instance generation process



AutoIG instance generation process



AutoIG: a constraint-based instance generation tool



- ***Graded* instances** (for a single solver)
  - non-trivial but solvable by a solver
    Akgün, Dang, Miguel, Salamon, Stone. *Instance generation via instance generators*. CP2019
- ***Discriminating instances*** (for a pair of solvers)
  - exhibit pronounce difference in performance between two solvers.
    Akgun, Dang, Miguel, Salamon, Spracklen & Stone. *Discriminating instance generation from abstract specifications: A case study with CP and MIP*. *CPAIOR2020*

Graded instances:

- *valid* instances that can be solved within [n_1, n_2] seconds, where n_1 and n_2 are pre-defined.
- example application: *generate instances with varying degrees of difficulty*

Graded instances:

- *valid* instances that can be solved within [n_1, n_2] seconds, where n_1 and n_2 are pre-defined.
- example application: *generate instances with varying degrees of difficulty*
  - Athanor: a high-level constraint-based local search solver
    - Attieh, Dang, Jefferson, Miguel & Nightingale. *Athanor: high-level local search over abstract constraint specifications in Essence*, IJCAI 2019
    - small instances for testing & debugging.
    - non-trivial instances for gaining insights on how the solver works, and for improving it.
    - a large instance set for parameter tuning.

Graded instances:

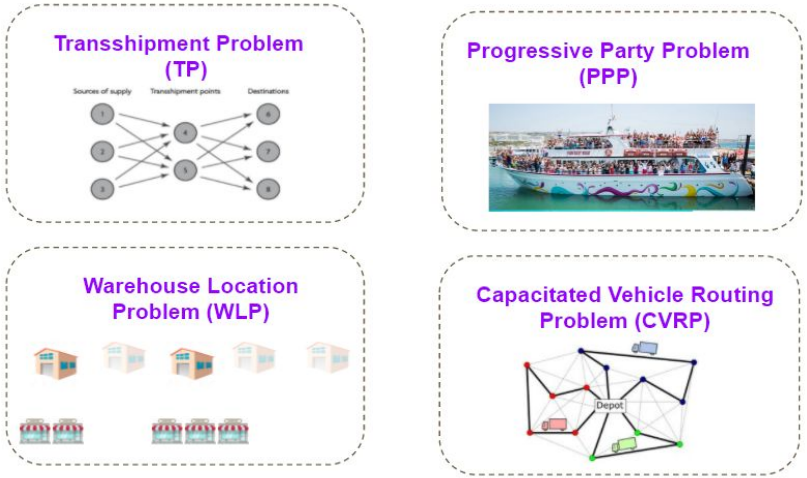
- *valid* instances that can be solved within [n_1, n_2] seconds, where n_1 and n_2 are pre-defined.
- example application: *generate instances with varying degrees of difficulty*
  - Automated generation of streamliners for constraint models
    - Spracklen, Dang, Agkun & Miguel. *Automatic streamlining for constrained optimisation,* CP2019
    - *Spracklen, Dang, Agkun & Miguel. Towards Portfolios of Streamlined Constraint Models: A Case Study with the Balanced Academic Curriculum Problem,* ModRef 2020
    - streamliners: constraints added to a model to speed up the solving process.
    - automated searching and training a robust portfolio of streamliners
      - needs a large set of *non-trivial,* but *not too difficult* instances for the training
      - plus a set of more challenging instances for evaluation.

Discriminating instances:

- *valid* instances that are:
  - *easy for one solver (the favoured solver)* while being *difficult for another solver* (the base solver).
- example application: *gaining insights into strengths and weakness of each solver*

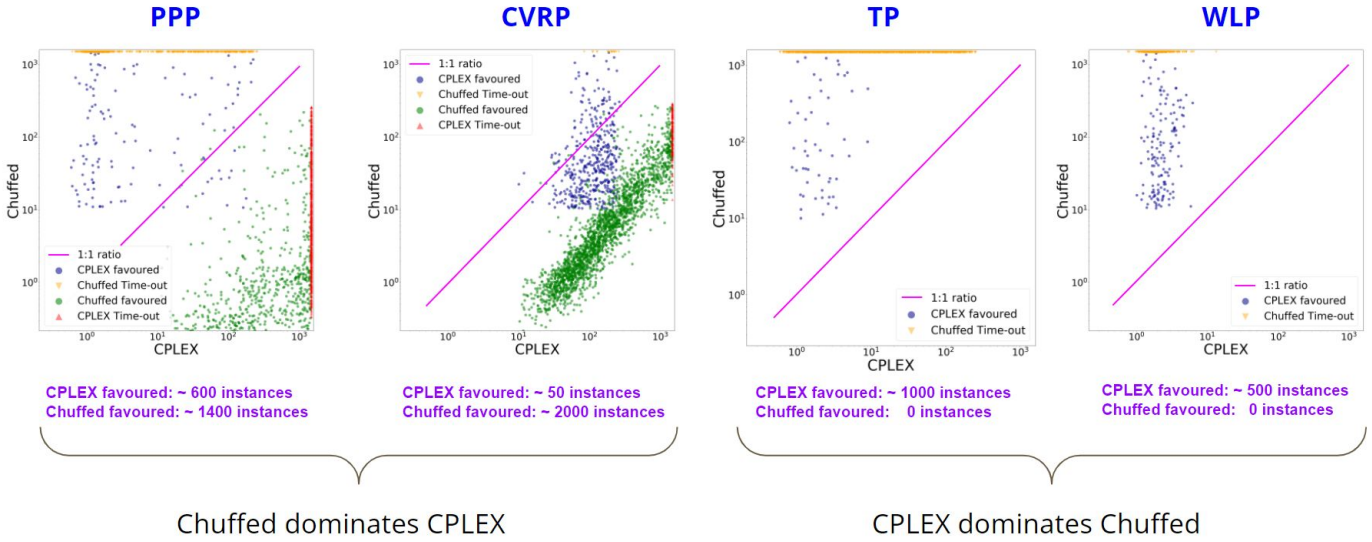
Discriminating instances:

- *valid* instances that are:
  - *easy for one solver (the favoured solver)* while being *difficult for another solver* (the base solver).
- example application: *gaining insights into strengths and weakness of each solver*
  - case study (*Akgun, Dang, Miguel, Salamon, Spracklen & Stone, 2020*):
    - **chuffed** (CP solver) and **CPLEX** (MIP solver)



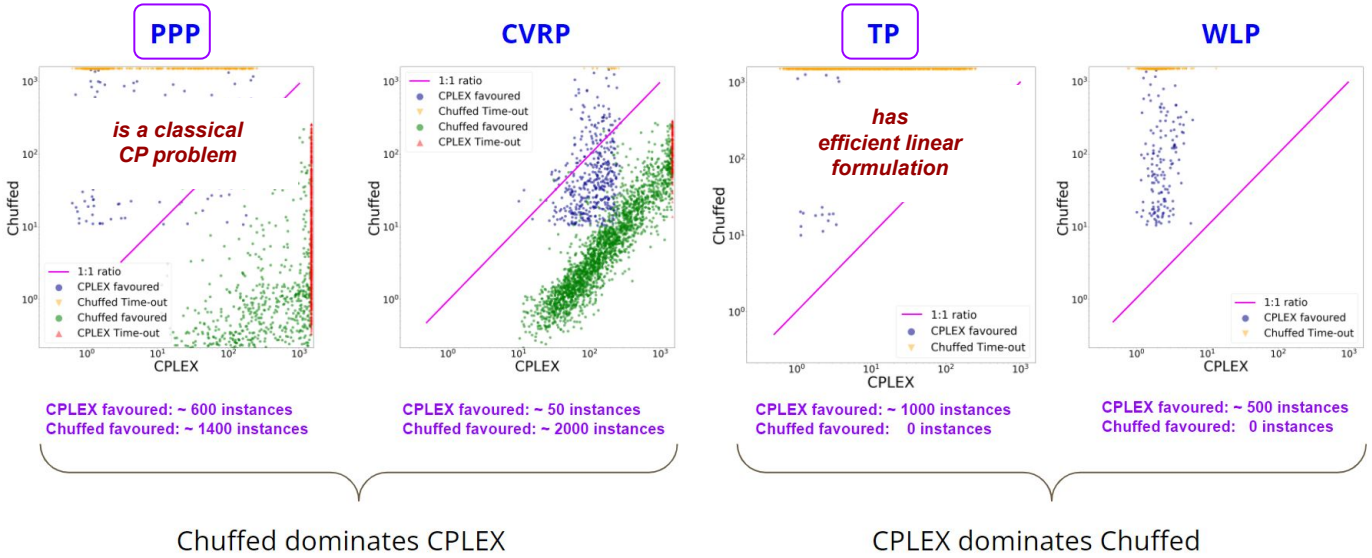
Discriminating instances:

- *valid* instances that are:
  - *easy for one solver (the favoured solver)* while being *difficult for another solver* (the base solver).
- example application: *gaining insights into strengths and weakness of each solver*
  - case study (*Akgun, Dang, Miguel, Salamon, Spracklen & Stone, 2020*):
    - **chuffed** (CP solver) and **CPLEX** (MIP solver)



Discriminating instances:

- *valid* instances that are:
  - *easy for one solver (the favoured solver)* while being *difficult for another solver* (the base solver).
- example application: *gaining insights into strengths and weakness of each solver*
  - case study (*Akgun, Dang, Miguel, Salamon, Spracklen & Stone, 2020*):
    - **chuffed** (CP solver) and **CPLEX** (MIP solver)



A study on the MiniZinc Challenges

- MiniZinc Challenges: https://www.minizinc.org/challenge.html
  - an annual competition series (2008-present) for benchmarking constraint solving technologies
  - *various solving paradigms*: CP, SAT, SMT, MIP & hybrid (via *MiniZinc* backends).

A study on the MiniZinc Challenges

- MiniZinc Challenges: https://www.minizinc.org/challenge.html
  - an annual competition series (2008-present) for benchmarking constraint solving technologies
  - *various solving paradigms*: CP, SAT, SMT, MIP & hybrid (via *MiniZinc* backends).
  - Benchmark instances:
    - 100 ***new*** instances each year (20 problems, 5 instances/problem)
    - Desirable properties:
      - not too easy but solvable by at least one participating solver.
      - not inadvertently favouring one solver over another.

A study on the MiniZinc Challenges

- MiniZinc Challenges: https://www.minizinc.org/challenge.html
  - an annual competition series (2008-present) for benchmarking constraint solving technologies
  - *various solving paradigms*: CP, SAT, SMT, MIP & hybrid (via *MiniZinc* backends).
  - Benchmark instances:
    - 100 ***new*** instances each year (20 problems, 5 instances/problem)
    - Desirable properties:
      - not too easy but solvable by at least one participating solver.
      - not inadvertently favouring one solver over another.
  - Sources:
    - Benchmark libraries
    - Submitted by participants

"In order to collect good benchmarks each entrant is strongly encouraged to submit one or two MiniZinc 2.3.1 models, making use of only the global constraints included in the MiniZinc 2.3.1 library, as well as some (preferably ***20***) instance files for each model. *The instances should range from easy (about a minute) to hard (about 15 minutes) if possible*. In addition, the submitter should provide one "toy" instance for testing purposes."

A study on the MiniZinc Challenges

- Dang, Akgün, Espasa, Miguel, and Nightingale (2022) *A Framework for Generating Informative Benchmark Instances*, CP2022.
- We use AutoIG for:
  - Automatically generating a large number of benchmark instances with the desired properties.
  - Gaining more detailed insights into solver performance than just a ranking
    - seeing how solver performance varies across different problems.
    - revealing cases where a solver is weak or even faulty.
    - revealing parts of the instance space where a generally weak solver performs well relative to others.

A study on the MiniZinc Challenges

(selected) Problems:

- *Multi-Agent Collaborative Construction problem* (MACC, *Lam et al 2020*): a multi-agent planning problem
- *Carpet Cutting problem* (*Schutt, Stuckey & Verden, 2011*): a packing problem
- *Mario problem*: a routing problem
- *Resource Availability Cost Problem* (RACP, *Kreter et al 2018*): a scheduling problem
- *Lot-sizing problem* (*Houndji et al 2014, Ullah & Parveen 2010*): a production scheduling problem

A study on the MiniZinc Challenges

(selected) Problems:

- *Multi-Agent Collaborative Construction problem* (MACC, *Lam et al 2020*): a multi-agent planning problem
- *Carpet Cutting problem* (*Schutt, Stuckey & Verden, 2011*): a packing problem
- *Mario problem*: a routing problem
- *Resource Availability Cost Problem* (RACP, *Kreter et al 2018*): a scheduling problem
- *Lot-sizing problem* (*Houndji et al 2014, Ullah & Parveen 2010*): a production scheduling problem

(selected) Solvers:

- *OR-Tools* (*Google*): a hybrid solver (CP + SAT + linear programming) 1st place
- *Picat-SAT* (*Zhou & Kjellerstrand 2017*): a SAT compiler, with *Kissat* (*Biere et al 2021*) as underlying solver. 2nd place
- *Chuffed* (*Chu et al 2018*): a learning CP solver non-participants
- *Yuck* (*Marte 2021*): a local-search constraint solver 1st place in local search category but very low overall ranking (last/second-to-last)

A study on the MiniZinc Challenges

Experiment 1: generate instances with desired properties using gradedness criteria.

Desired properties:

- Not too easy but solvable by at least one participating solver
- not inadvertently favouring one solver over another

A study on the MiniZinc Challenges

Experiment 1: generate instances with desired properties using gradedness criteria.

Desired properties:

- Not too easy but solvable by at least one participating solver
- not inadvertently favouring one solver over another

Our method (for a given problem):

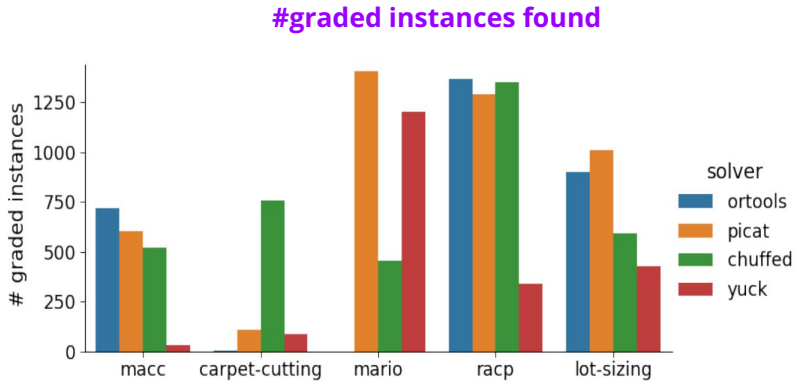
- For each solver *k*, apply AutoIG to find a set of *valid & graded* instances S_k
- Combined instance set S: randomly select 50 instances from each S_k
- Evaluate all solvers on all instances in S.

graded = solved within [10s, 1200s]

- decision problem: a solution found or unsatisfiability proved
- optimisation problem:
  - unsatisfiability/optimality proved (non local-search)
  - optimal solution found (local-search)

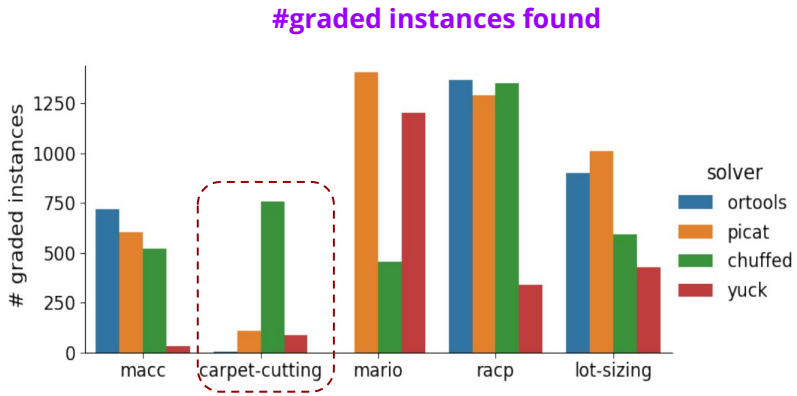
A study on the MiniZinc Challenges

Experiment 1: generate instances with desired properties using gradedness criteria.



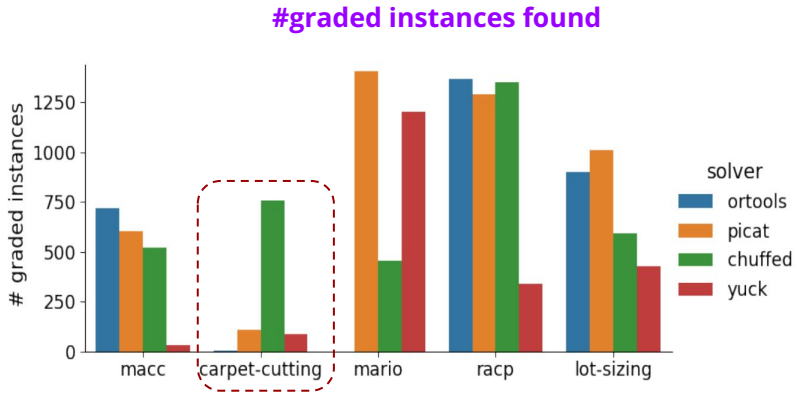
A study on the MiniZinc Challenges

Experiment 1: generate instances with desired properties using gradedness criteria.



A study on the MiniZinc Challenges

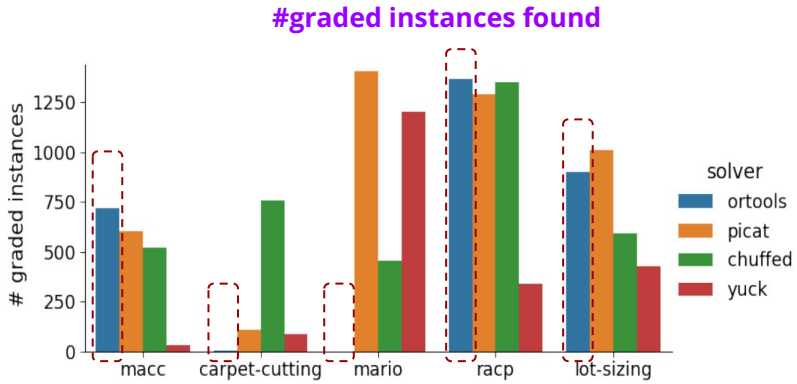
Experiment 1: generate instances with desired properties using gradedness criteria.



Conjecture 1: Solver performance can vary significantly when solving instances drawn from the same instance space.

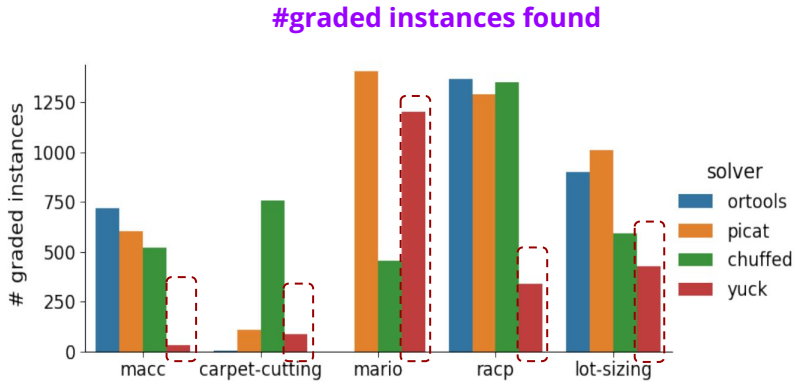
A study on the MiniZinc Challenges

Experiment 1: generate instances with desired properties using gradedness criteria.



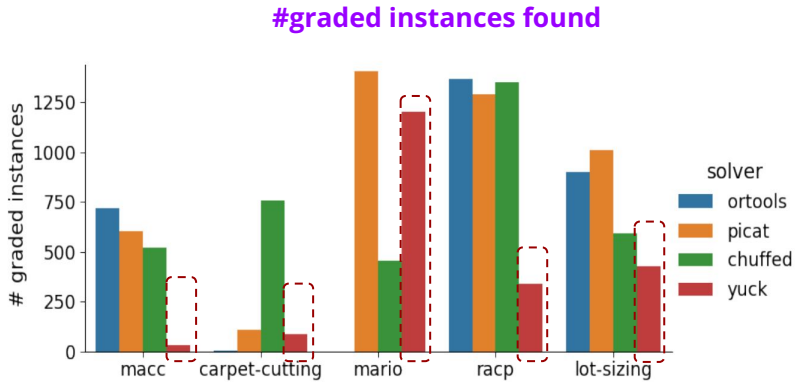
A study on the MiniZinc Challenges

Experiment 1: generate instances with desired properties using gradedness criteria.



A study on the MiniZinc Challenges

Experiment 1: generate instances with desired properties using gradedness criteria.

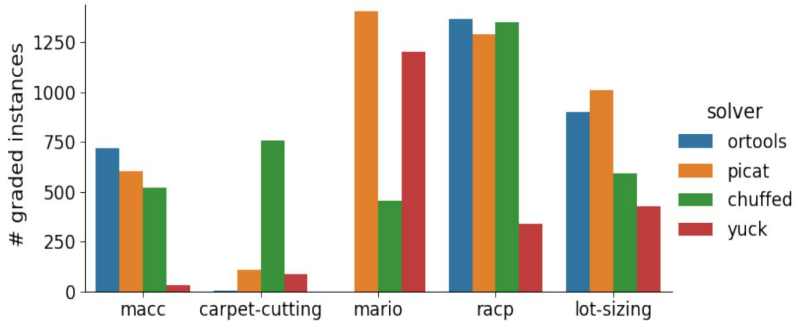


Conjecture 2: For the same solver, some problems are easier/more challenging than others.

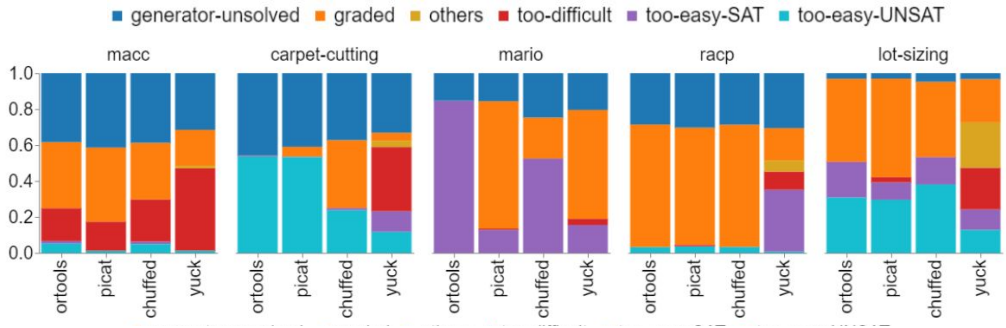
A study on the MiniZinc Challenges

Experiment 1: generate instances with desired properties using gradedness criteria.

#graded instances found



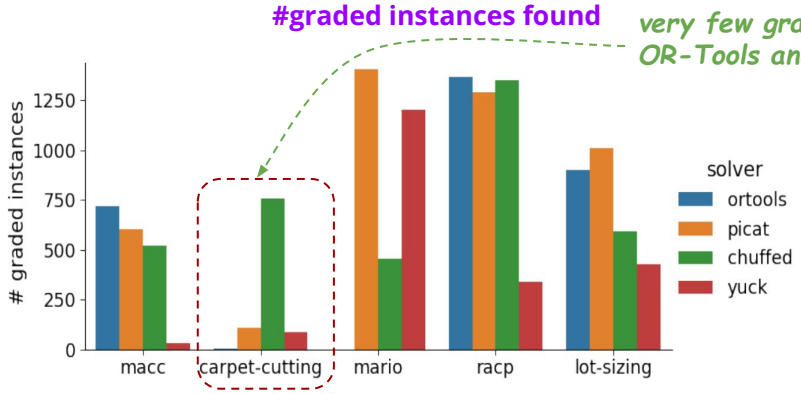
Run status frequency

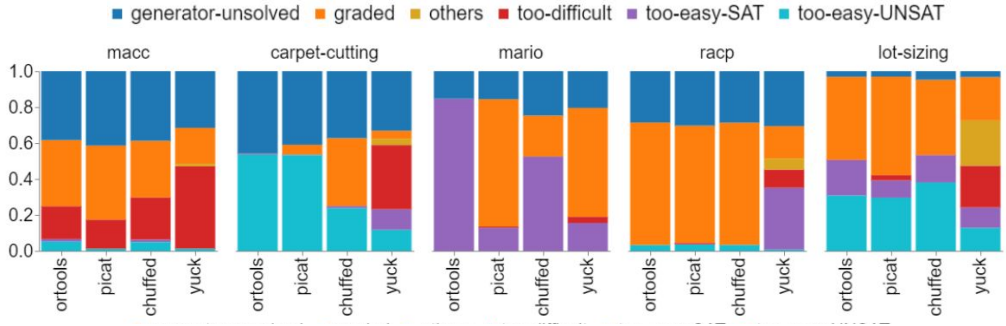


Conjecture 1: Solver performance can vary significantly when solving instances drawn from the same instance space.

A study on the MiniZinc Challenges

Experiment 1: generate instances with desired properties using gradedness criteria.

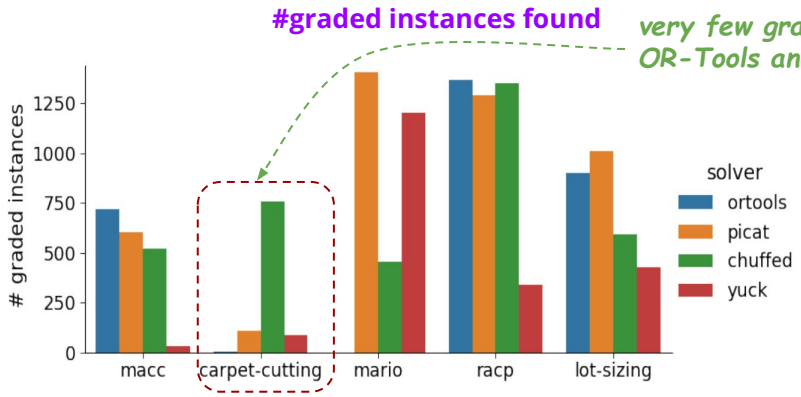


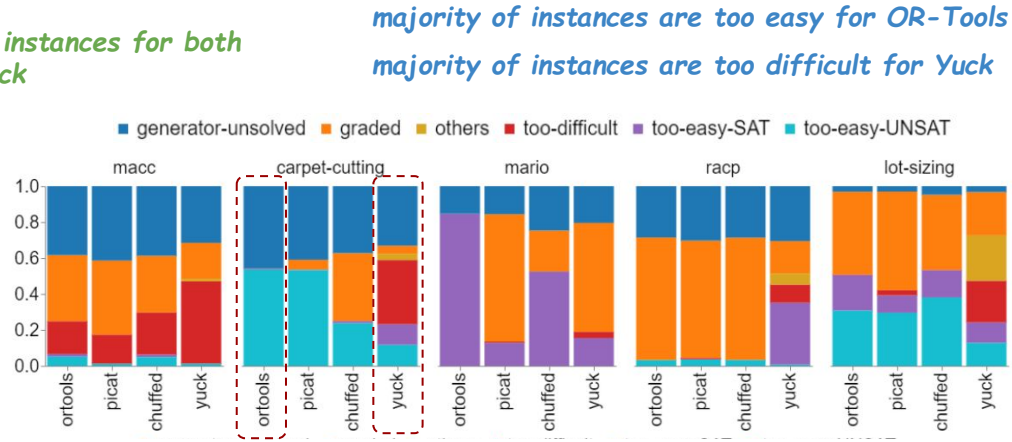


Conjecture 1: Solver performance can vary significantly when solving instances drawn from the same instance space.

A study on the MiniZinc Challenges

Experiment 1: generate instances with desired properties using gradedness criteria.

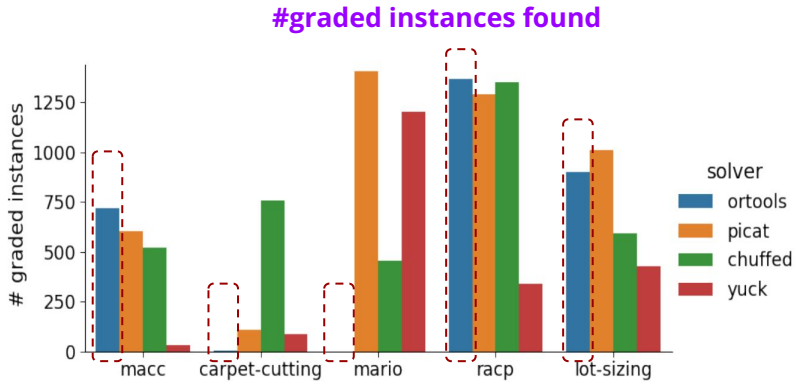




Conjecture 1: Solver performance can vary significantly when solving instances drawn from the same instance space.

A study on the MiniZinc Challenges

Experiment 1: generate instances with desired properties using gradedness criteria.

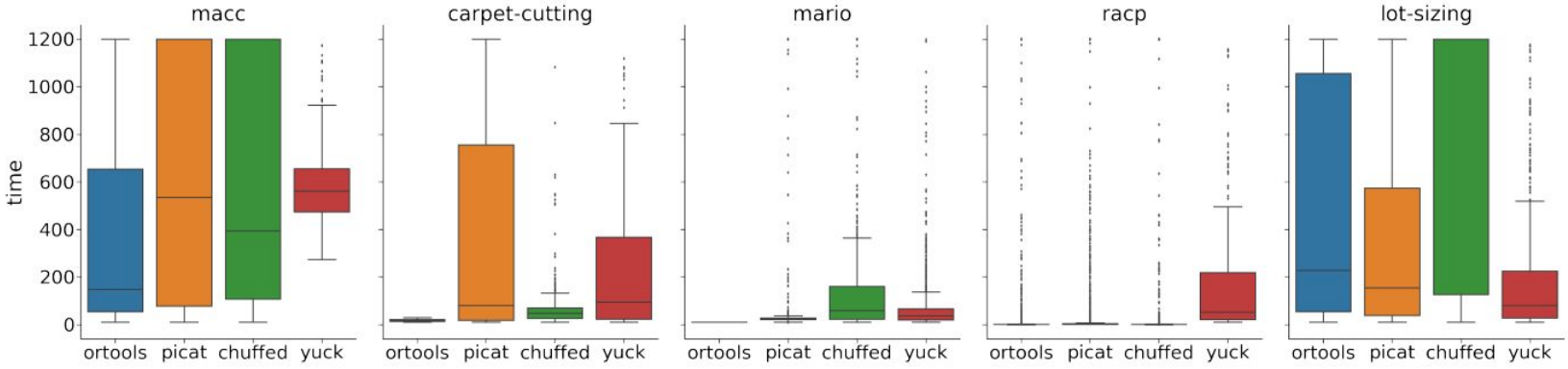


Conjecture 2: For the same solver, some problems are easier/more challenging than others.

A study on the MiniZinc Challenges

Experiment 1: generate instances with desired properties using gradedness criteria.

Solving time distribution on graded instances



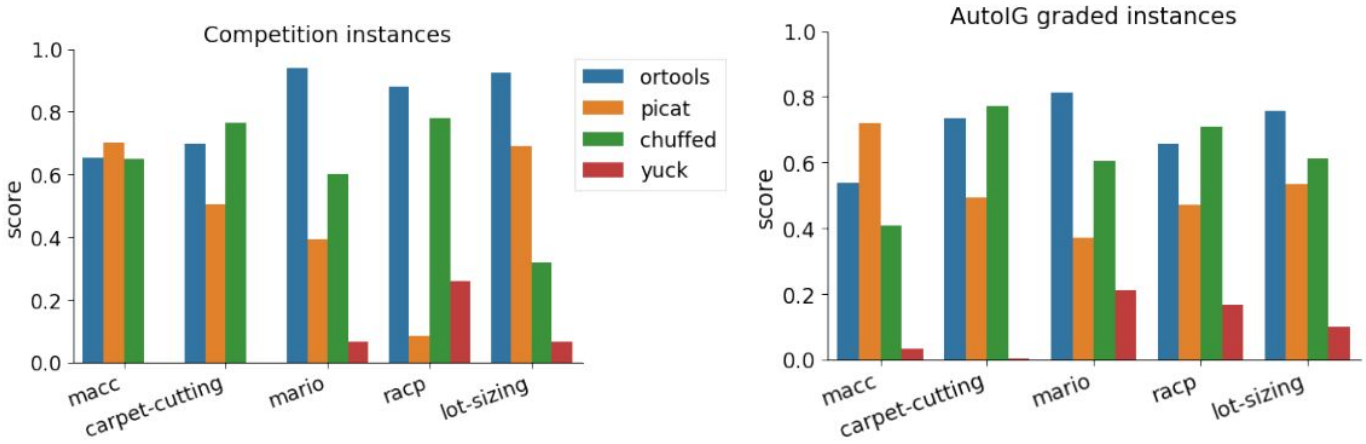
Carpet-cutting, Mario and RACP are mostly easy for OR-Tools (but AutoIG did find challenging RACP instances!)

MACC and Lot-Sizing exhibit a good diversity of difficulties for OR-Tools (and for other solvers)

A study on the MiniZinc Challenges

Experiment 1: generate instances with desired properties using gradedness criteria.

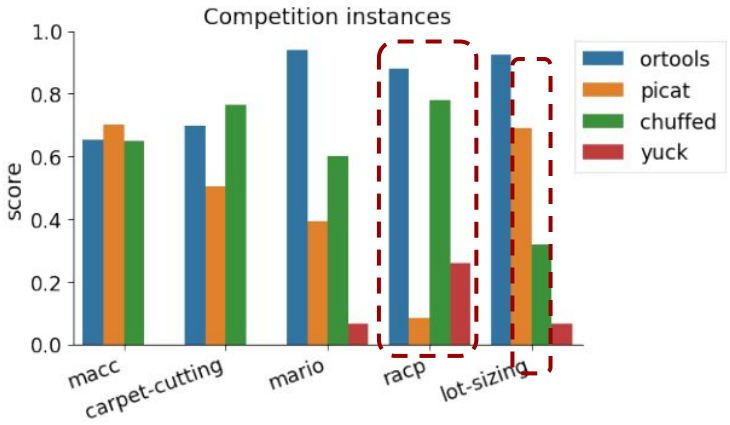
MiniZinc scores on the combined graded instance set

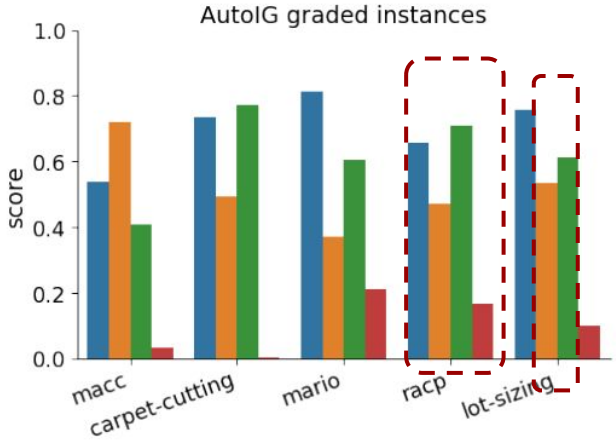


A study on the MiniZinc Challenges

Experiment 1: generate instances with desired properties using gradedness criteria.

MiniZinc scores on the combined graded instance set





The overall rankings are quite similar!

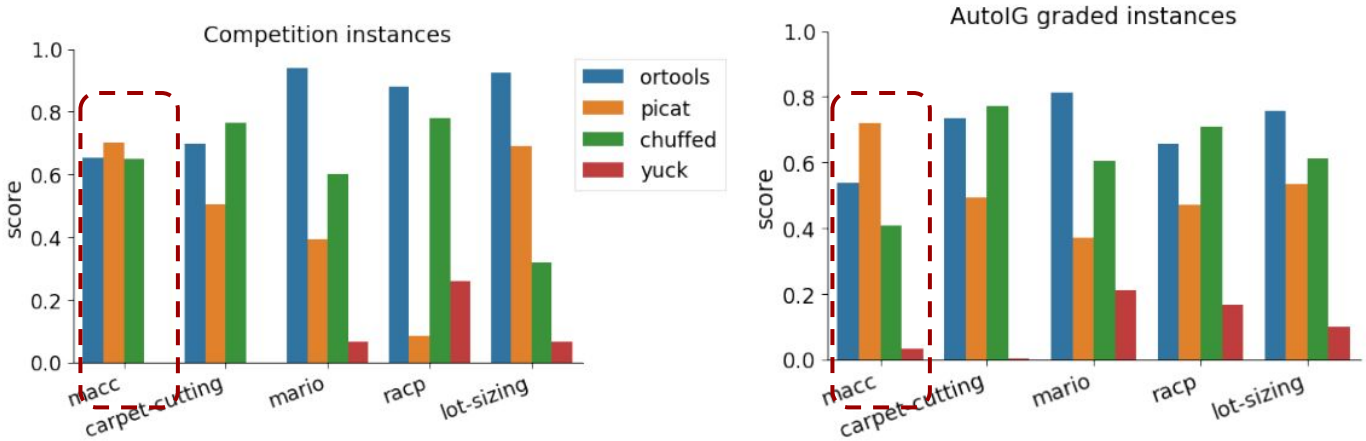
Some rankings are swapped:

- *RACP:*
  OR-Tools ↔ Chuffed
  Picat ↔ Yuck
- *Lot-sizing: Picat ↔ Chuffed*

A study on the MiniZinc Challenges

Experiment 1: generate instances with desired properties using gradedness criteria.

MiniZinc scores on the combined graded instance set



Yuck is completely dominated by all other solvers

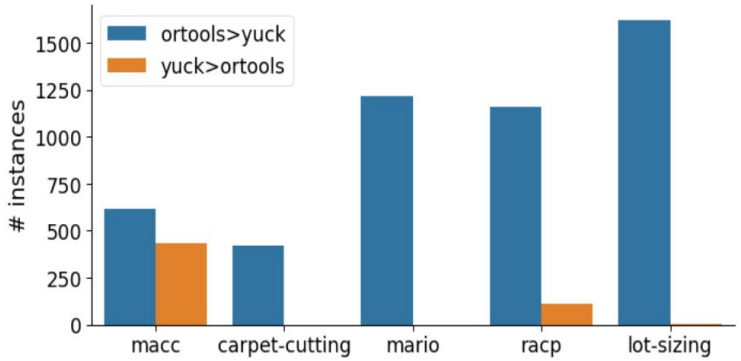
Not really!

Next experiment: generating discriminating instances for OR-Tools vs Yuck

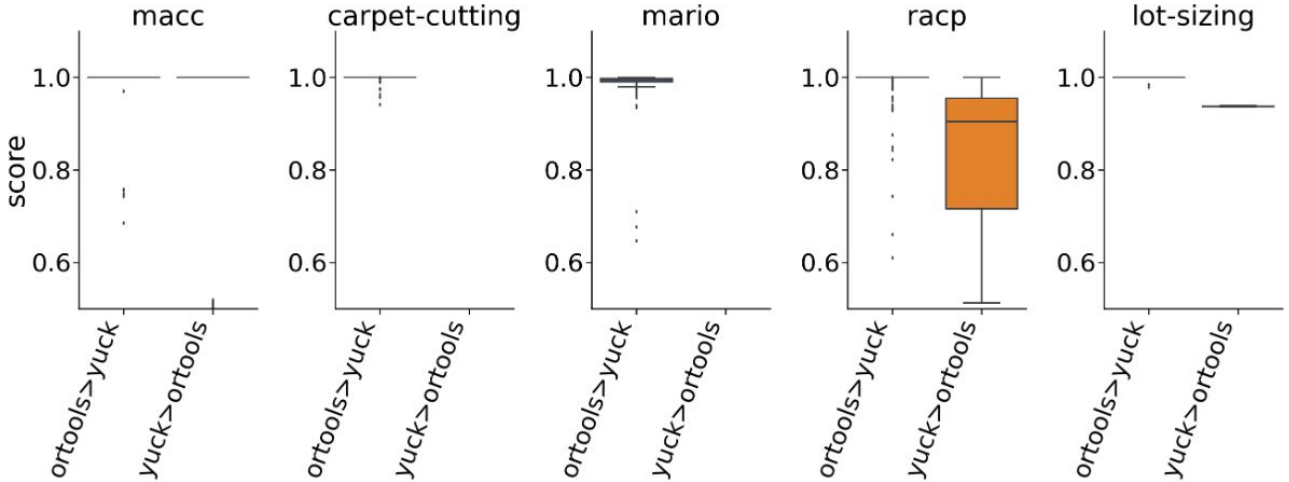
A study on the MiniZinc Challenges

Experiment 2: generate discriminating instances for OR-Tools and Yuck

#discriminating instances



Score distribution of the favoured solver on discriminating instances



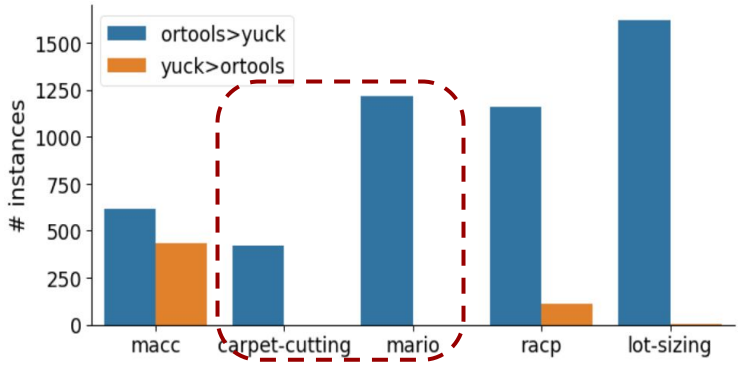
score ∈ (0.5, 1]
score=1: maximum discriminating power

A study on the MiniZinc Challenges

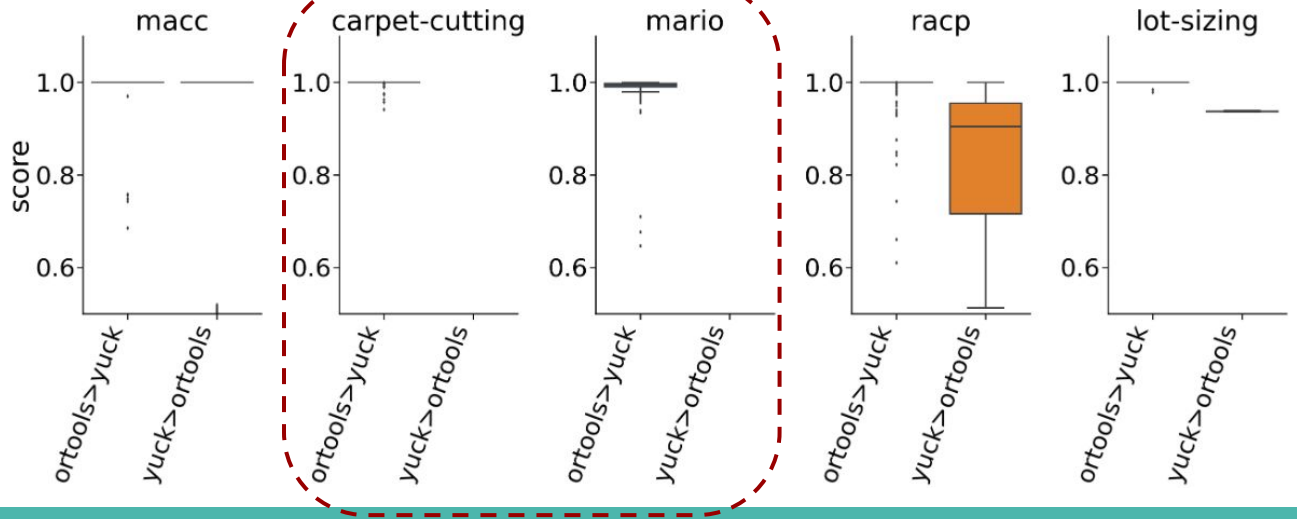
Experiment 2: generate discriminating instances for OR-Tools and Yuck

Carpet-cutting & Mario:
Yuck is completely dominated by OR-Tools

#discriminating instances



Score distribution of the favoured solver on discriminating instances



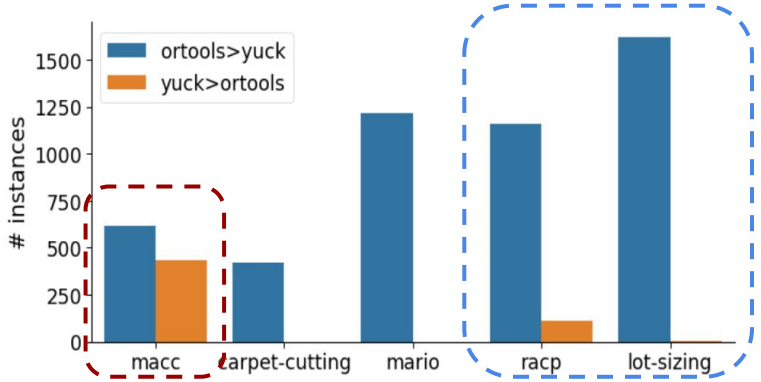
score ∈ (0.5, 1]
score=1: maximum discriminating power

A study on the MiniZinc Challenges

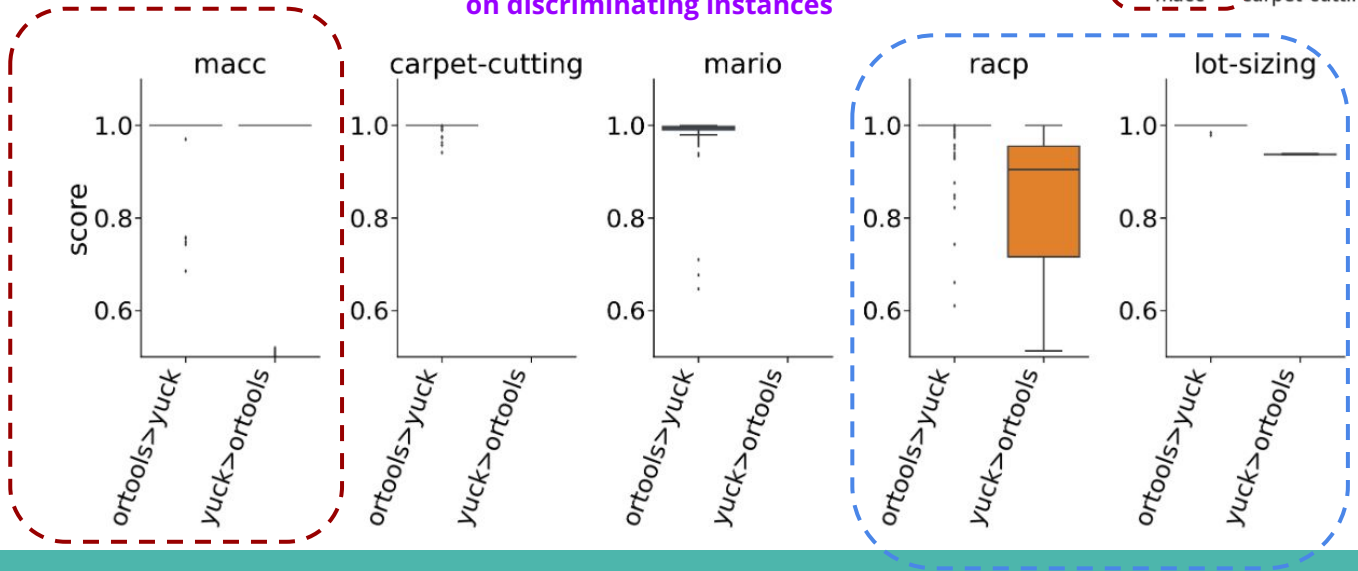
Experiment 2: generate discriminating instances for OR-Tools and Yuck

MACC, RACP & Lot-sizing:
Yuck can offer good complementary strengths to OR-Tools

#discriminating instances



Score distribution of the favoured solver on discriminating instances

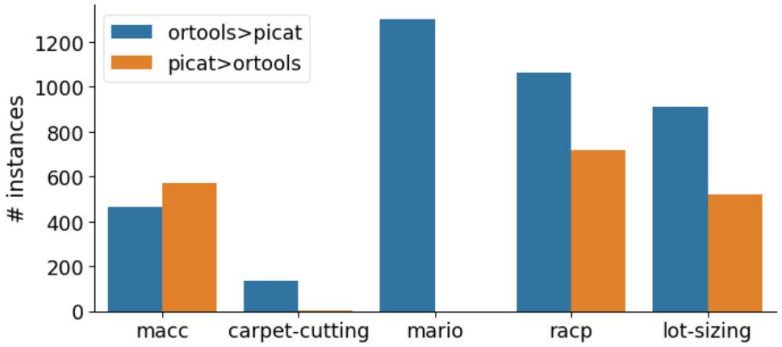


score ∈ (0.5, 1]
score=1: maximum discriminating power

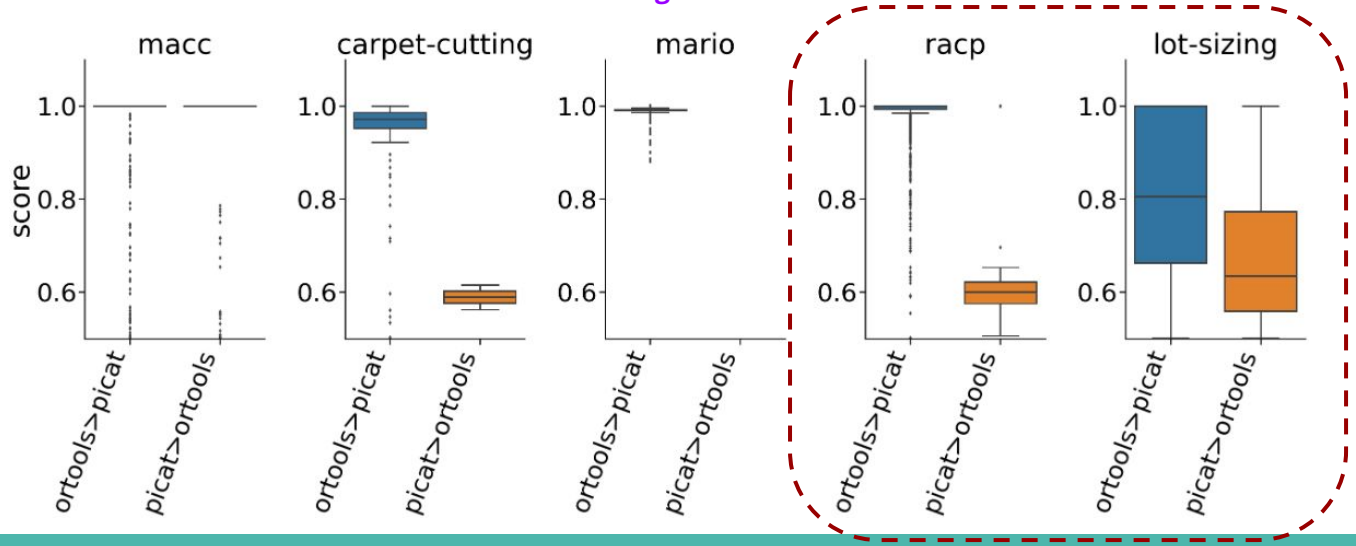
A study on the MiniZinc Challenges

Experiment 3: generate discriminating instances for OR-Tools and Picat

#discriminating instances



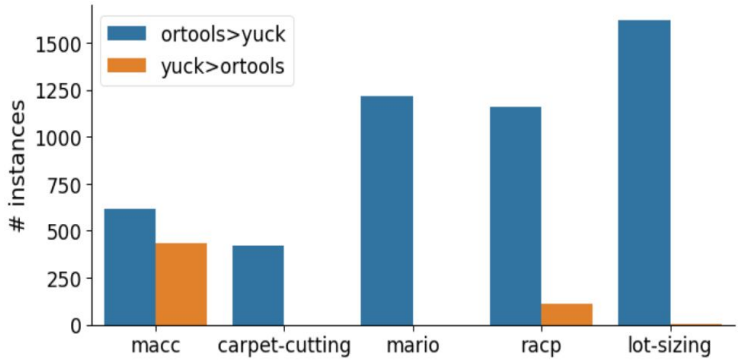
Score distribution of the favoured solver on discriminating instances



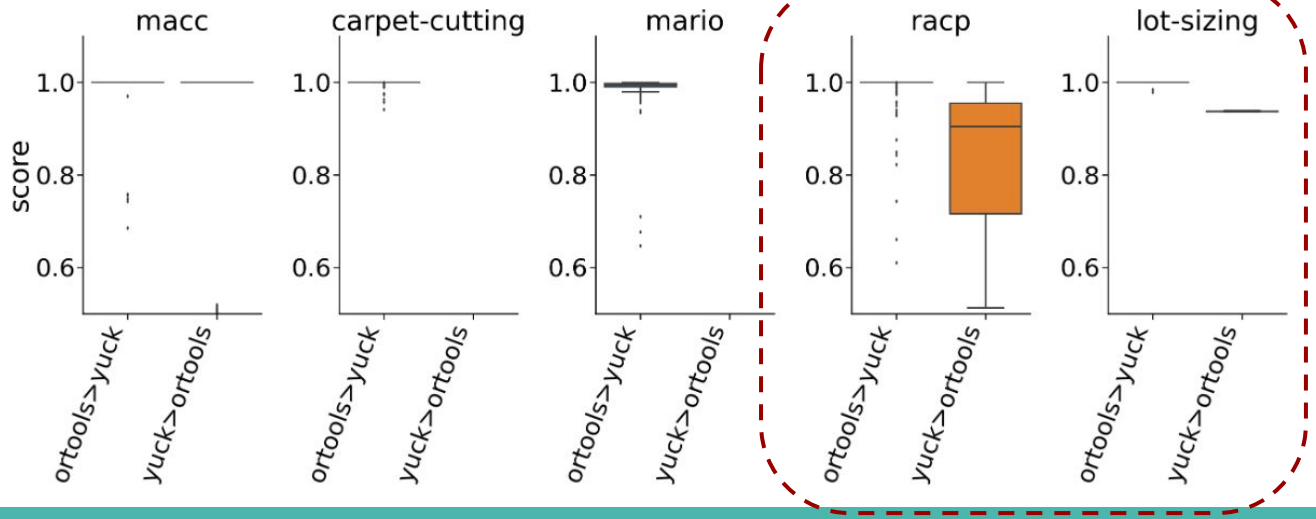
A study on the MiniZinc Challenges

Experiment 2: generate discriminating instances for OR-Tools and Yuck

#discriminating instances



Score distribution of the favoured solver on discriminating instances



score ∈ (0.5, 1]
score=1: maximum discriminating power

AutoIG: https://github.com/stacs-cp/AutoIG

Future works

- Improving the diversity of instances generated.
  - Diversity in term of solver performance.
  - Diversity in term of instance features.
- Visualisation and post analysis on the generated instances.
- Generating instances that are close to real-world data.
- (cross-domain) instance features