Automated Streamliner Selection via Automated Algorithm Configuration and Selection

Patrick Spracklen

Nguyen Dang

Özgür Akgün

Ian Miguel

University of St Andrews

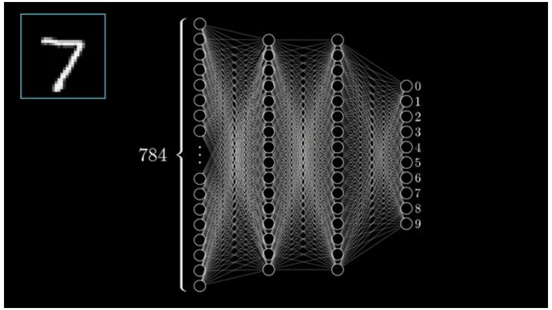
Automated Streamliner Selection

via Automated Algorithm Configuration and Selection

Automated Algorithm Configuration

Algorithm Parameters

Almost every algorithm has its own parameters that can be tuned!

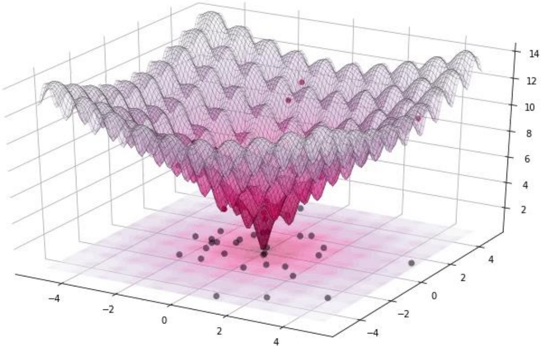


Deep Learning

#hidden layers, #hidden nodes

activation function

learning rate

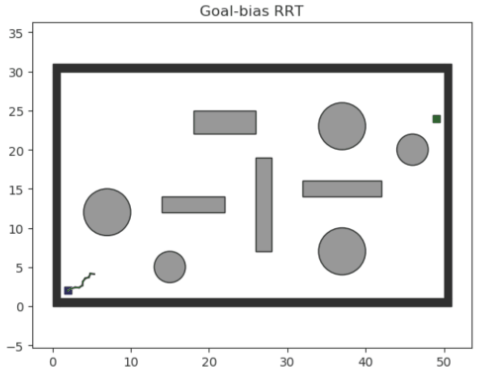


Evolutionary Algorithms

mutation rate

crossover probability

population size



AI Planning

choices of heuristics in greedy best first search

Automated Algorithm Configuration

General-purpose techniques to configure algorithm parameters automatically

algorithm **A** with configuration space $\boldsymbol{C}$
instance set $\boldsymbol{I}$
performance metric $\boldsymbol{p}$

→ algorithm configurator →

a configuration $c^* \in C$ such that
$\mathbf{f}(\boldsymbol{c}^*) = \frac{\mathbf{1}}{|\boldsymbol{I}|}\sum_{\boldsymbol{i}\in\boldsymbol{I}} \boldsymbol{p}(\boldsymbol{c}, \boldsymbol{i})$ is optimised

Automated Algorithm Configuration

General-purpose techniques to solve the algorithm configuration problem **automatically**

algorithm **A** with configuration space $\boldsymbol{C}$
instance set $\boldsymbol{I}$
performance metric $\boldsymbol{p}$

→ algorithm configurator →

a configuration $c^* \in C$ such that
$\mathbf{f}(\boldsymbol{c}^*) = \frac{\mathbf{1}}{|\boldsymbol{I}|}\sum_{\boldsymbol{i}\in \boldsymbol{I}} \boldsymbol{p}(\boldsymbol{c}, \boldsymbol{i})$ is optimised

Key components

- ❑ **A black-box optimisation algorithm**
  - Local search algorithm: ParamILS (*Hutter et al 2007, 2009)*
  - Genetic algorithm: GGA, GGA++ (*Tierney et al 2009, Ansotegui at al 2015)*
  - Estimation of distribution algorithm: irace (*López-Ibáñez et al 2011, 2016)*
  - Bayesian optimization: SMAC, SMAC3 (*Hutter et all 2011, Lindauer et al 2022)*
  - Golden section search algorithm: GPS (*Pushak & Hoos, 2022)*

Automated Algorithm Configuration

General-purpose techniques to solve the algorithm configuration problem **automatically**

algorithm **A** with configuration space $\boldsymbol{C}$
instance set $\boldsymbol{I}$
performance metric $\boldsymbol{p}$

→ algorithm configurator →

a configuration $c^* \in C$ such that
$\mathbf{f}(\boldsymbol{c}^*) = \frac{\mathbf{1}}{|\boldsymbol{I}|}\sum_{\boldsymbol{i}\in\boldsymbol{I}} \boldsymbol{p}(\boldsymbol{c}, \boldsymbol{i})$ is optimised

Key components

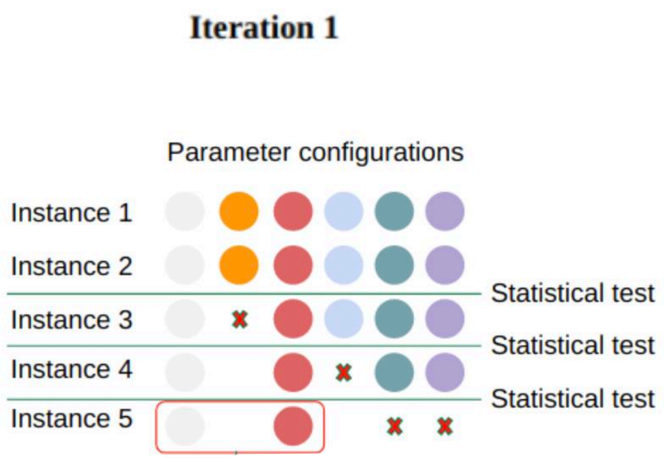
- **A black-box optimisation algorithm**
  - Local search algorithm: ParamILS (*Hutter et al 2007, 2009)*
  - Genetic algorithm: GGA, GGA++ (*Tierney et al 2009, Ansotegui at al 2015)*
  - Estimation of distribution algorithm: irace (*López-Ibáñez et al 2011, 2016)*
  - Bayesian optimization: SMAC, SMAC3 (*Hutter et all 2011, Lindauer et al 2022)*
  - Golden section search algorithm: GPS (*Pushak & Hoos, 2022)*

- **Special tricks to reduce the cost of evaluating each configuration on *all* instances**
  - racing
  - adaptive capping (when performance metric is runtime)

racing

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

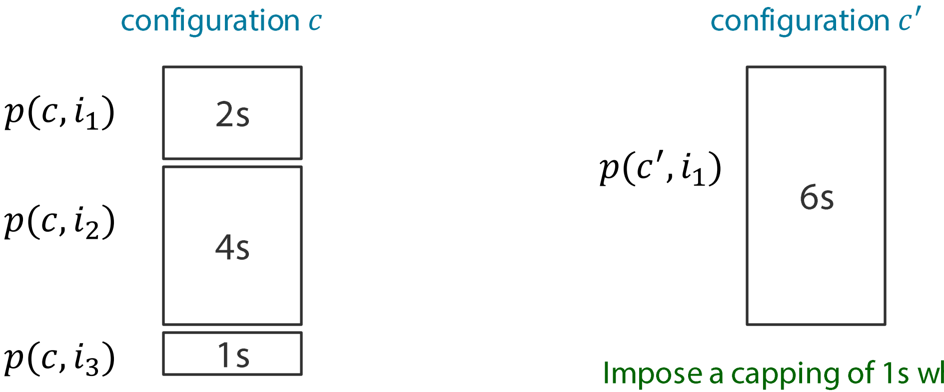
Iterated racing



López-Ibáñez, Dubois-Lacoste, Cáceres, Birattari, Stützle (2016)
The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives.

capping

time limit of each run: 3600s



Impose a capping of 1s when running c' on i_2

Automated Algorithm Selection

Automated Algorithm Selection

- In many cases, there is often no single algorithm that performs best on ***all*** problem instances
- **Automated Algorithm Selection**

  given a set of (complementary) algorithms, predict the best algorithm for a given problem instance

  (based on instance features)

Automated Algorithm Selection

- In many cases, there is often no single algorithm that performs best on ***all*** problem instances
- Automated Algorithm Selection

  given a set of (complementary) algorithms, predict the best algorithm for a given problem instance

  (based on instance features)

- **Effective automated algorithm selection recipe**
  - informative instance features
  - a representative (and sufficiently large) training instance set
  - suitable ML models (+ extra tricks)

Automated Algorithm Selection

SATzilla

Nudelman, Devkar, Shoham, Leyton-Brown, Hoos (2004) "*SATzilla: An Algorithm Portfolio for SAT*". SAT competition

Xu, Hutter, Hoos, Leyton-Brown (2008) "*SATzilla: portfolio-based algorithm selection for SAT*". JAIR

Xu, Hutter, Hoos, Leyton-Brown (2009) "*SATzilla2009: an Automatic Algorithm Portfolio for SAT*". SAT competition

Xu, Hutter, Shen, Hoos, Leyton-Brown (2012) "*SATzilla2012: Improved algorithm selection based on cost-sensitive classification models*". Proceedings of SAT Challenge.

- won several medals at SAT competitions 2007, 2009 & 2012
- informative SAT features:
  - syntactic features
  - probing features
- a representative (and sufficiently large) training instance set
  - several thousands of instances from previous SAT competitions
- suitable ML models (+ extra tricks)
  - empirical hardness models: regression models to predict algorithm performance
  - cost-sensitive pairwise classification models with random forests

Automated Algorithm Selection

SATzilla

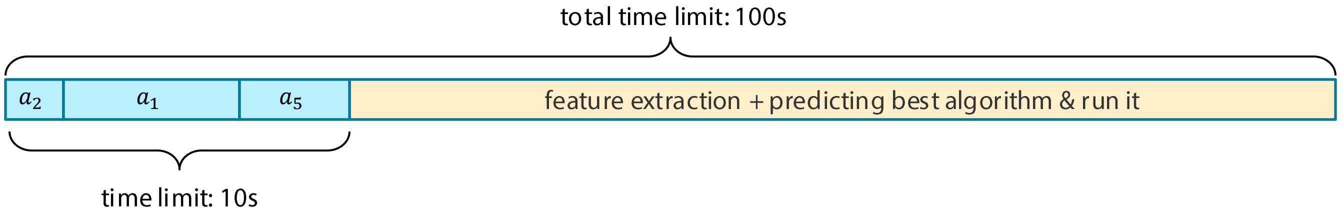
Nudelman, Devkar, Shoham, Leyton-Brown, Hoos (2004) "*SATzilla: An Algorithm Portfolio for SAT*". SAT competition

Xu, Hutter, Hoos, Leyton-Brown (2008) "*SATzilla: portfolio-based algorithm selection for SAT*". JAIR

Xu, Hutter, Hoos, Leyton-Brown (2009) "*SATzilla2009: an Automatic Algorithm Portfolio for SAT*". SAT competition

Xu, Hutter, Shen, Hoos, Leyton-Brown (2012) "*SATzilla2012: Improved algorithm selection based on cost-sensitive classification models*". Proceedings of SAT Challenge.

- suitable ML models (+ extra tricks)
  - trick: pre-solving (static) algorithm schedule (runtime scenarios)
    - some instances can be solved very quickly by a subset of algorithms
    - feature extraction can be computationally expensive



Automated Algorithm Selection

- **Automatically choose a suitable ML model and tricks:**

  Lindauer, Hoos, Hutter, Schaub (2015) "*AutoFolio: An automatically configured algorithm selector*". JAIR

  https://github.com/automl/AutoFolio

  Automated Algorithm Configuration for Algorithm Selection

  - **Feature preprocessing methods**

    PCA
    standardisation/normalisation
    data imputation

  - **Use pre-solving schedule?**

    percentage of time for pre-solving schedule

  - **Prediction model**

    clustering / regression / (cost-sensitive) pairwise classification
    random forest / neural networks / XGBosst/ etc
    hyper-parameter values for the chosen ML model.

Per-instance Automated Algorithm Configuration

Per-instance Algorithm Configuration

- **Automated algorithm configuration**
  - given: a (large) algorithm configuration space, a set of problem instances
  - objective: search for the best overall algorithm configuration on the given instance set

    (in the hope that this configuration will also work well for unseen instances)
- **Automated algorithm selection**
  - given: a set of (complementary) algorithms, a set of problem instances
  - objective: predict the best algorithm for any given (unseen) instance
- **Per-instance algorithm configuration**
  - given: a (large) algorithm configuration space, a set of problem instances
  - objective: predict the best *algorithm configuration* for any given (unseen) instance

Per-instance Algorithm Configuration

- **Per-instance algorithm configuration**
  - given: a (large) algorithm configuration space, a set of instances
  - objective: predict the best algorithm configuration for any given (unseen) instance

Step 1: build a set of algorithm configurations with *complementary strengths*

Step 2: apply automated algorithm selection on that set

Per-instance Algorithm Configuration

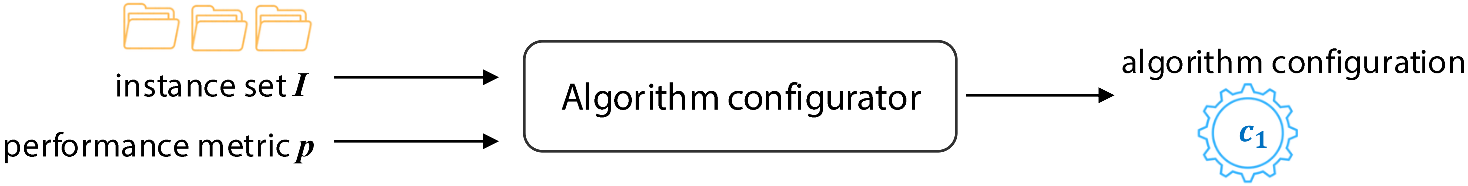
- **Per-instance algorithm configuration**
  - given: a (large) algorithm configuration space, a set of instances
  - objective: predict the best algorithm configuration for any given (unseen) instance

Step 1: build a set of algorithm configurations with *complementary strengths*

Step 2: apply automated algorithm selection on that set

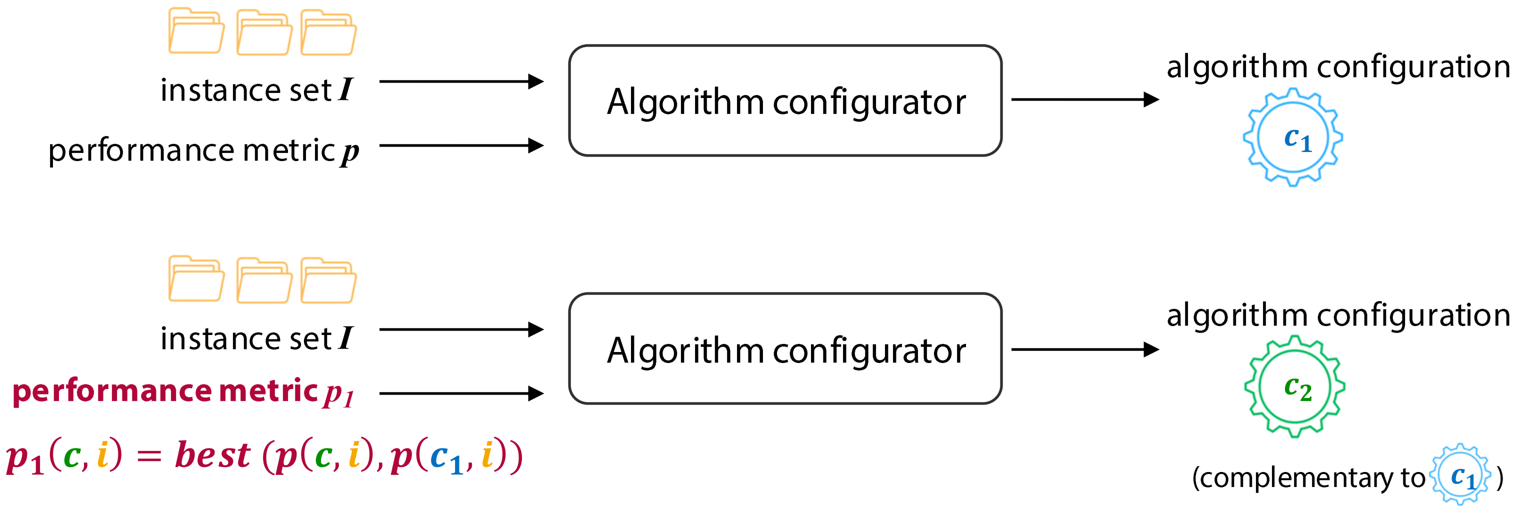
Per-instance Algorithm Configuration

- **Step 1: the Hydra approach**
  - Xu, Hoos, Leyton-Brown (2010) *Hydra: Automatically configuring algorithms for portfolio-based selection. AAAI*
  - given: a (large) algorithm configuration space, a set of problem instances
  - objective: build a set of algorithm configurations with *complementary strengths*



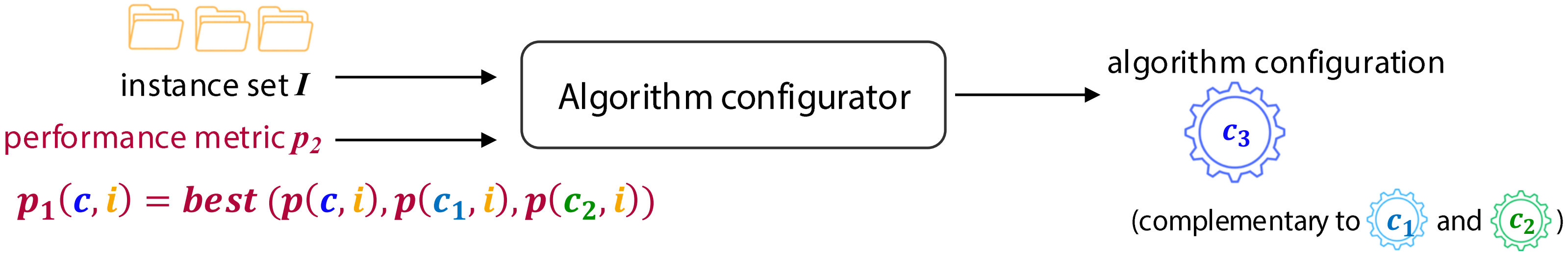
Per-instance Algorithm Configuration

- **Step 1: the Hydra approach**
  - Xu, Hoos, Leyton-Brown (2010) *Hydra: Automatically configuring algorithms for portfolio-based selection. AAAI*
  - given: a (large) algorithm configuration space, a set of problem instances
  - objective: build a set of algorithm configurations with *complementary strengths*



Per-instance Algorithm Configuration

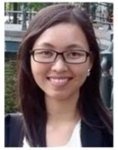
- **Step 1: the Hydra approach**
  - Xu, Hoos, Leyton-Brown (2010) *Hydra: Automatically configuring algorithms for portfolio-based selection. AAAI*
  - given: a (large) algorithm configuration space, a set of problem instances
  - objective: build a set of algorithm configurations with *complementary strengths*



$$p_1(c, i) = best\ (p(c, i), p(c_1, i), p(c_2, i))$$

Automated Streamliner Selection via Automated Algorithm Configuration and Selection

Patrick Spracklen



Nguyen Dang

Özgür Akgün

Ian Miguel

Automated Streamlining for Constrained Optimisation. CP 2019

Towards Portfolios of Streamlined Constraint Models: A Case Study with the Balanced Academic Curriculum Problem. ModRef 2020

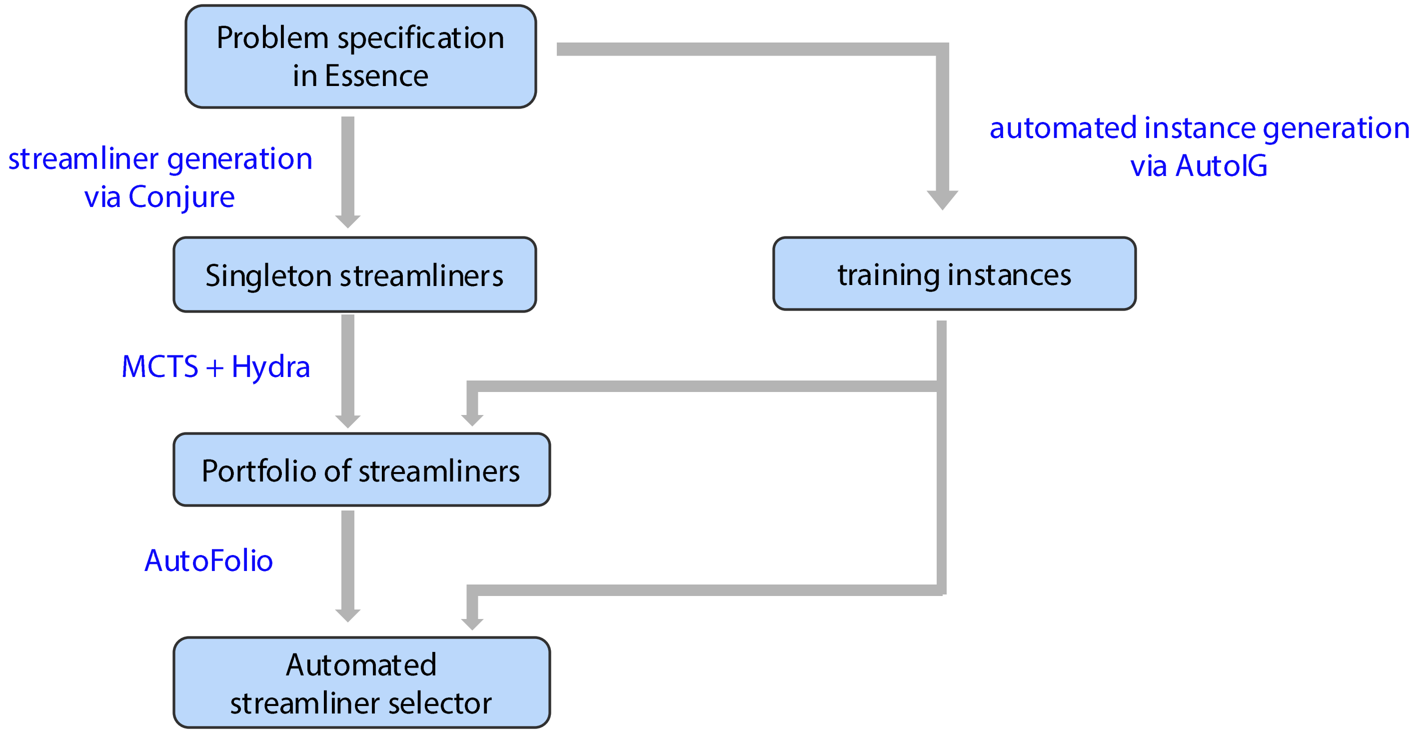
Automated streamliner portfolios for constraint satisfaction problems. Artificial Intelligence Journal (2023)

Automated Streamliner Selection

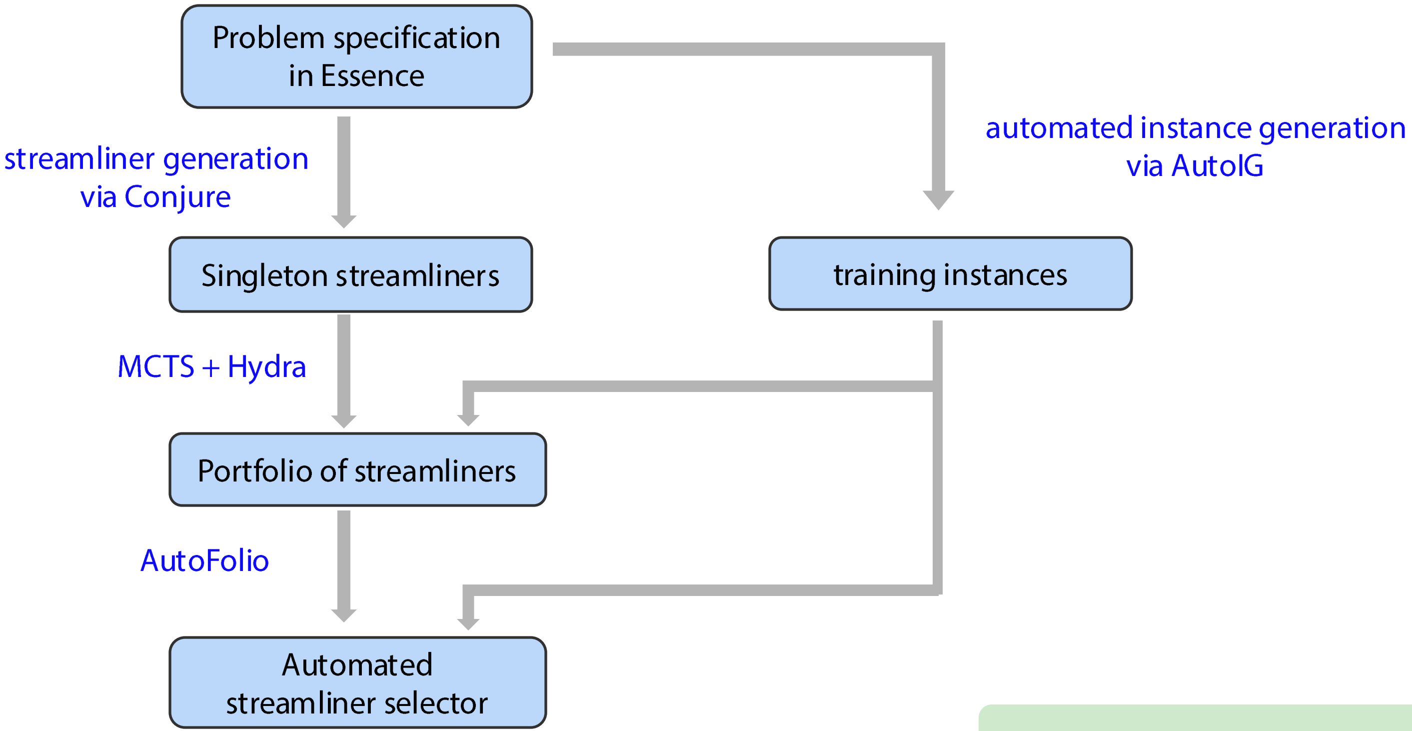
Streamliners: uninferred constraints added to a constraint model to reduce the search space.

- first proposed in: Carla Gomes and Meinolf Sellmann (2004) *Streamlined constraint reasoning*. CP
- not guaranteed to be sound
- but if chosen correctly, can offer significant speedup in solving time

Automated Streamliner Selection

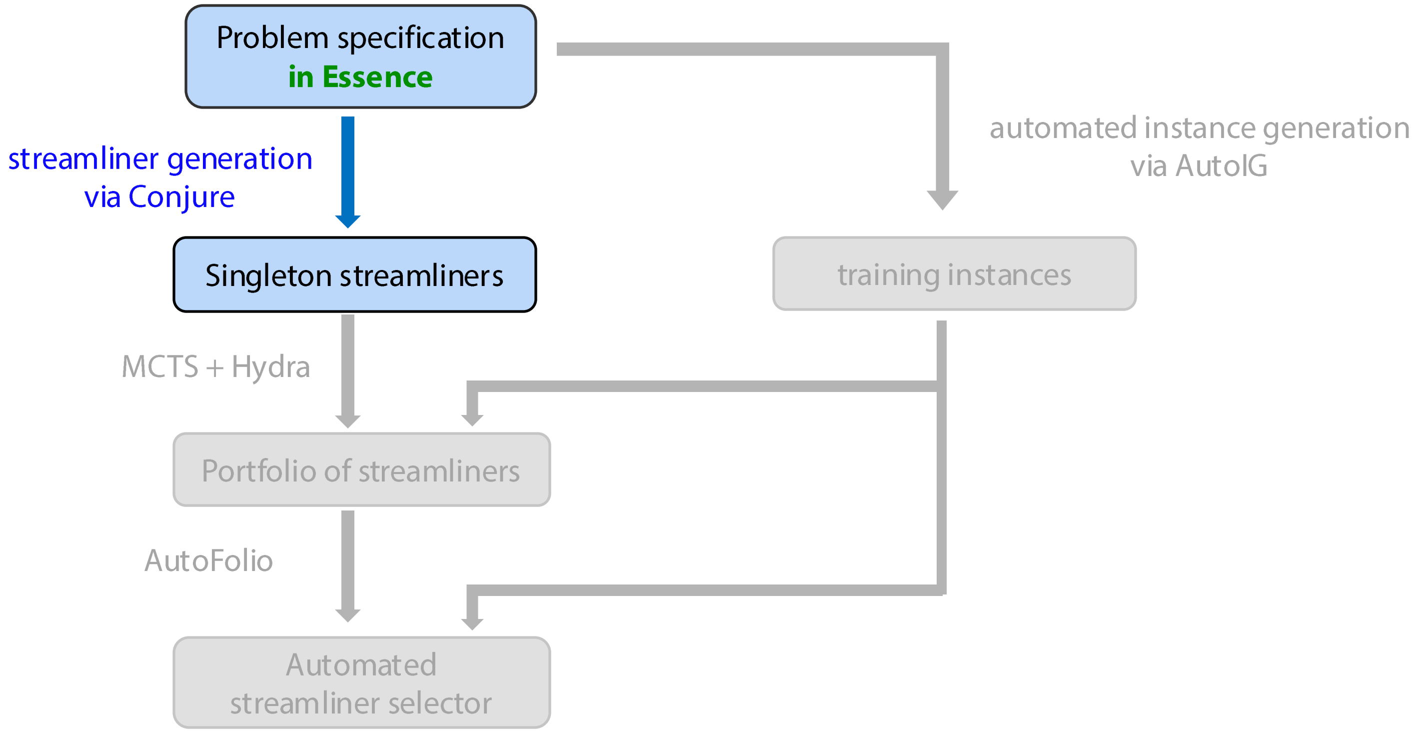


Automated Streamliner Selection



for constraint satisfaction problems

Automated Streamliner Selection



Automated Streamliner Generation

Essence

Frisch, Harvey, Jefferson, Martínez-Hernández, Miguel (2008)
Essence: A constraint language for specifying combinatorial problems. Constraints.

- an abstract constraint specification language
- supports several abstract types: set, multiset, function, partition, relation, …
  and *arbitrary nesting* of such types

Social Golfers Problem:

In a golf club there are a number of golfers who wish to play together in **g** groups of size **s**.

Find a schedule of play for **w** days such that no pair of golfers play together more than once

Mon	ABCD	EFGH	IJKL	MNOP	QRST
Tue	AEIM	BJOQ	CHNT	DGLS	FKPR
Wed	AGKO	BIPT	CFMS	DHJR	ELNQ
Thu	AHLP	BKNS	CEOR	DFIQ	GJMT
Fri	AFJN	BLMR	CGPQ	DEKT	HIOS

20 golfers, 5 groups, 5 days

Source: https://mathworld.wolfram.com/SocialGolferProblem.html

Automated Streamliner Generation

Essence

Frisch, Harvey, Jefferson, Martínez-Hernández, Miguel (2008)
Essence: A constraint language for specifying combinatorial problems. Constraints.

- an abstract constraint specification language
- supports several abstract types: set, multiset, function, partition, relation, …
  and *arbitrary nesting* of such types

Social Golfers Problem:

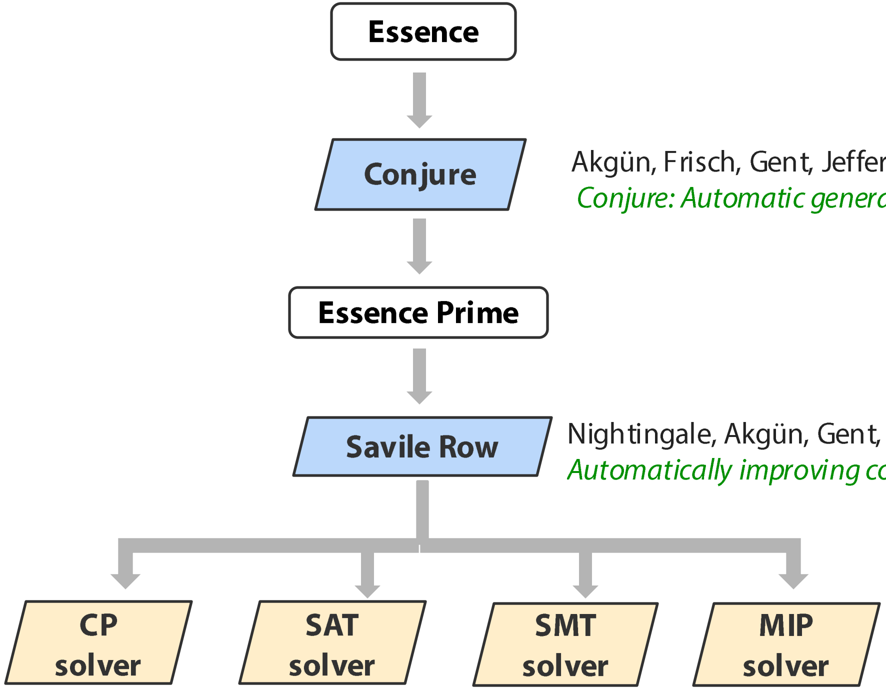
In a golf club there are a number of golfers who wish to play together in **g** groups of size **s**.

Find a schedule of play for **w** days such that no pair of golfers play together more than once

```
language Essence 1.3
given w, g, s : int(1..)
letting Golfers be new type of size g * s
find sched : set (size w) of
             partition (regular, numParts g, partSize s)
             from Golfers
such that
  forAll g1, g2 : Golfers, g1 < g2 .
    (sum week in sched . toInt(together({g1, g2}, week))) <= 1
```

Automated Streamliner Generation

Essence pipeline



Akgün, Frisch, Gent, Jefferson, Miguel, Nightingale (2022) *Conjure: Automatic generation of constraint models from problem specifications*. AIJ

Nightingale, Akgün, Gent, Jefferson, Miguel, Spracklen (2017) *Automatically improving constraint models in Savile Row*. AIJ

Automated Streamliner Generation

Streamliner generation from an Essence specification

- We define a set of rules to generate streamliners from the types of the decision variables in an Essence constraint model.

- First-order rules: constraints that directly reduce the domain of a decision variable

  - integer variables:
    - only allow odd/even values
    - restrict domain to the lower/upper half
  - function variables:
    - enforce that the function is monotonically increasing/decreasing
    - enforce that the function is commutative
  - partition variables
    - make it quasi-regular: size of each partition must be roughly equal

  …

Automated Streamliner Generation

Streamliner generation from an Essence specification

- We define a set of rules to generate streamliners from the types of the decision variables in an Essence constraint model.

- Higher-order rules: take another rule and apply it to a variable with nested domains
  - examples:
    - set of integers:
      - *approximately half* of the integers must be *odd*
    - set of functions:
      - *at least one* function must be *monotonically increasing*

Automated Streamliner Generation

Streamliner generation from an Essence specification

- We define a set of rules to generate streamliners from the types of the decision variables in an Essence constraint model.

- Higher-order rules: take another rule and apply it to a variable with nested domains
  - examples:
    - set of integers:
      - *approximately half* of the integers must be *odd*
    - set of functions:
      - *at least one* function must be *monotonically increasing*

softness parameter

Automated Streamliner Generation

Streamliner generation from an Essence specification

- Given a problem written in Essence, we can generate a large set of candidate streamliners

(from CSPLib)

timetabling	`$ Balanced Academic Curriculum Problem (BACP)` `find curr : function (total) Course --> Period`
combinatorial design	`$ Balanced Incomplete Block Designs (BIBD)` `find bibd : relation of (Obj * Block)`
testing	`$ Covering Array` `find CA: matrix indexed by [int(1..k), int(1..b)] of int(1..g)`

coding theory	`$ Equidistant Frequency Permutation Arrays (EFPA)`
	`letting String be domain function (total) Index --> Character`
	`find c : set (size numCodeWords) of String`

telecommunication	`$ Fixed Length Error Correcting Codes (FLECC)`
	`letting String be domain function (total) Index --> Character`
	`find c : set (size numOfCodeWords) of String`

network flow	`$ Transshipment`
	`find amountWT : function (W, T) --> int(1..max(range(stock)))`
	`find amountTC : function (T, C) --> int(1..max(range(demand))`

scheduling	`$ Tail Assignment` `find route : function (total) Plane --> function int(1..n_flights) --> Flight`
combinatorial design	`$ Social Golfers` `find sched : set (size w) of partition (regular, numParts g, partSize s) from Golfers`

transportation	`$ Vessel Loading`
	`find west, east : function (total) Container --> X,`
	`north, south: function (total) Container --> Y`

Automated Streamliner Generation

Streamliner generation from an Essence specification

- Given a problem written in Essence, we can generate a large set of candidate streamliners

Problem	#Candidate Streamliners
BACP	108
BIBD	200
CoveringArray	64
Car Sequencing	36
EFPA	312
FLECC	144
Transshipment	68
Tail Assignment	336
Social Golfers	260
Vessel Loading	208

Automated Streamliner Generation

Streamliner generation from an Essence specification

- Given a problem written in Essence, we can generate a large set of candidate streamliners

- Streamliners can also be **combined**

  Example:

  - integer variables:
    - only allow odd/even values
    - restrict domain to the lower/upper half

  → combination: must be odd with domain restricted to the lower half

Problem	#Candidate Streamliners
BACP	108
BIBD	200
CoveringArray	64
Car Sequencing	36
EFPA	312
FLECC	144
Transshipment	68
Tail Assignment	336
Social Golfers	260
Vessel Loading	208

Automated Streamliner Generation

Streamliner generation from an Essence specification

- Given a problem written in Essence, we can generate a large set of candidate streamliners

- Streamliners can also be **combined**

  Example:

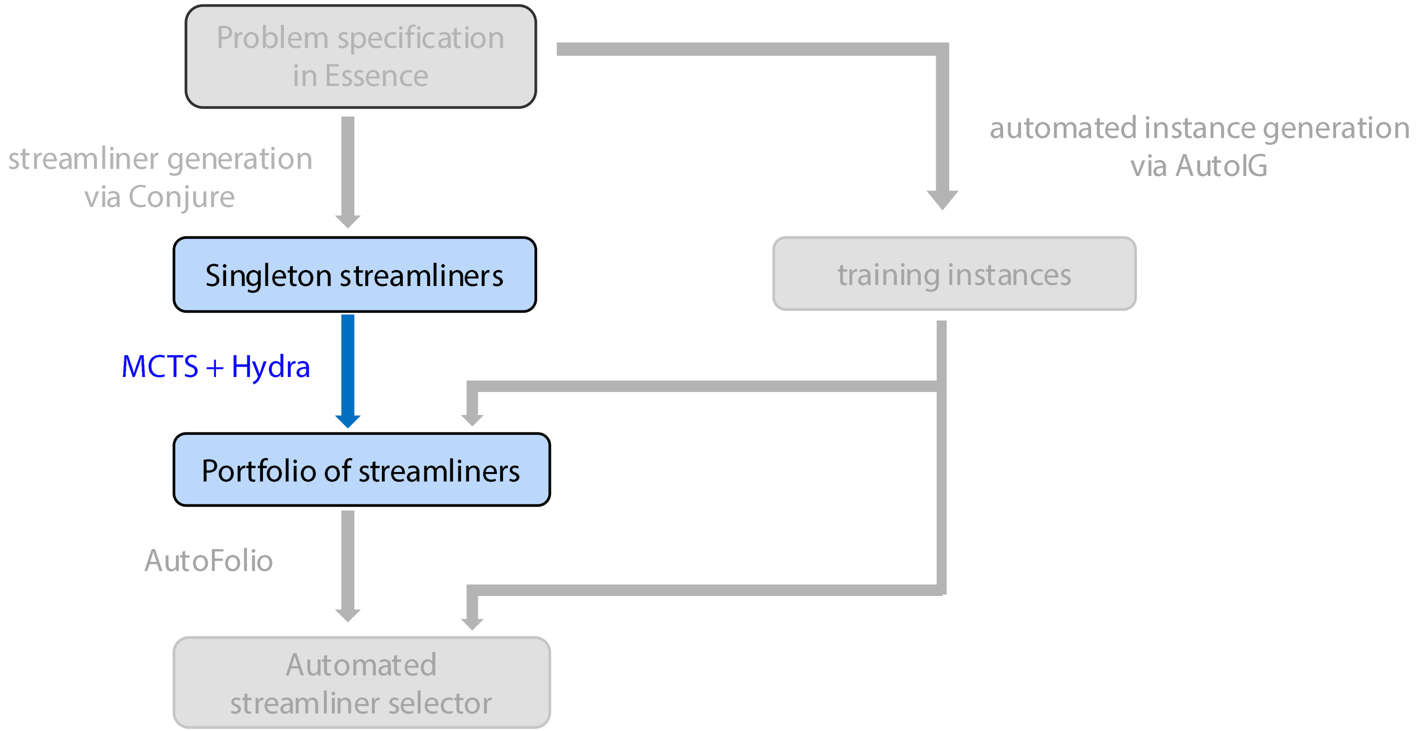
  - integer variables:
    - only allow odd/even values
    - restrict domain to the lower/upper half

  → combination: must be odd with domain restricted to the lower half

Problem	#Candidate Streamliners
BACP	108
BIBD	200
CoveringArray	64
Car Sequencing	36
EFPA	312
FLECC	144
Transshipment	68
Tail Assignment	336
Social Golfers	260
Vessel Loading	208

given a problem instance, which streamliner (combination) should we use?

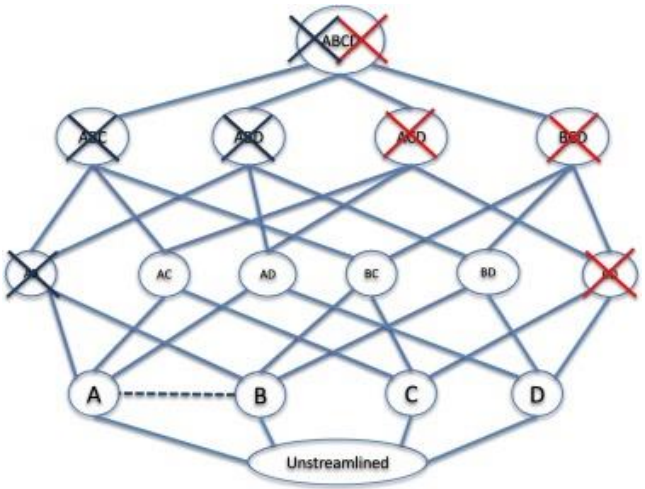
Automated Streamliner Selection



Automated Streamliner Generation & Selection

Monte Carlo Tree Search (MCTS) to search in the streamliner combination space

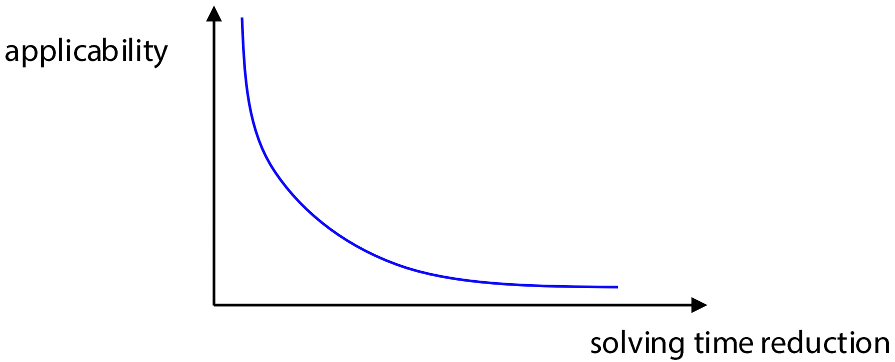
- the search space forms a lattice.
- pruning: if a streamliner combination returns UNSAT, its supersets will also return UNSAT.



Automated Streamliner Generation & Selection

Monte Carlo Tree Search (MCTS) to search in the streamliner combination space

- performance of a streamliner combination:
  - **applicability**: percentage of training instances solved
  - **solving time reduction**: average reduction in solving time across the solved instances



Automated Streamliner Generation & Selection

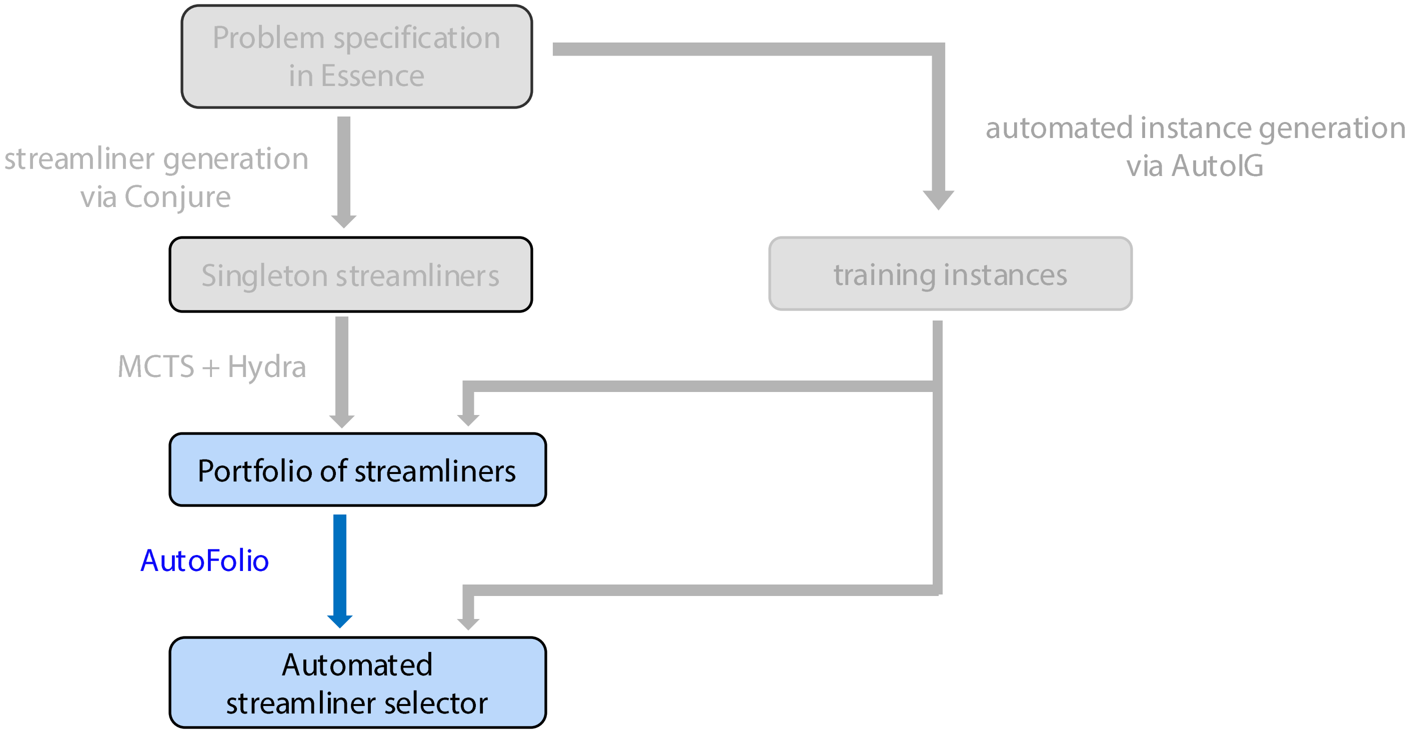
Monte Carlo Tree Search (MCTS) to search in the streamliner combination space

- performance of a streamliner combination:
  - applicability: percentage of training instances solved
  - solving time reduction: average reduction in solving time across the solved instances
- **Muti-objective MCTS**:

  Wang and Sebag (2013) *Hypervolume indicator and dominance reward based multi-objective monte-carlo tree search*. Machine learning.

  - returns a set of streamliner combinations in the Pareto front

Automated Streamliner Selection

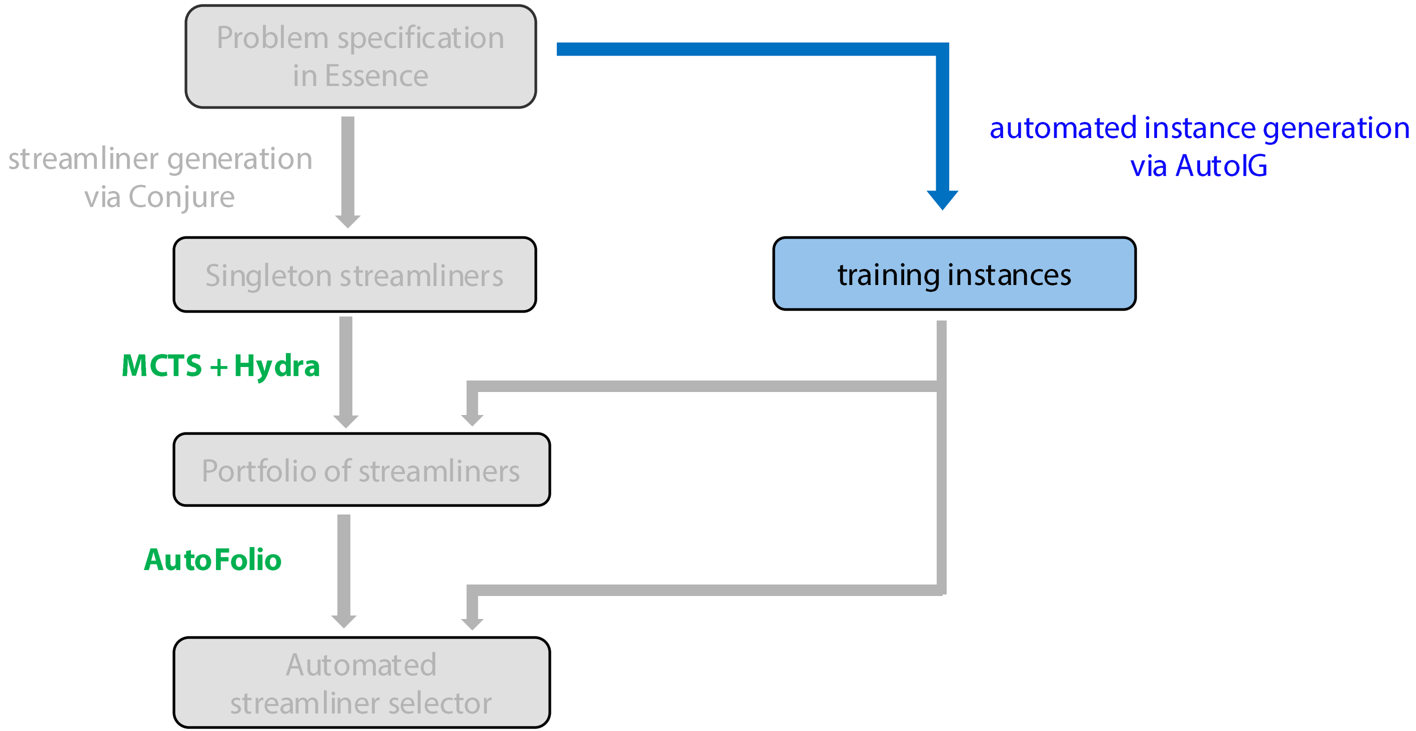


Automated Streamliner Selection

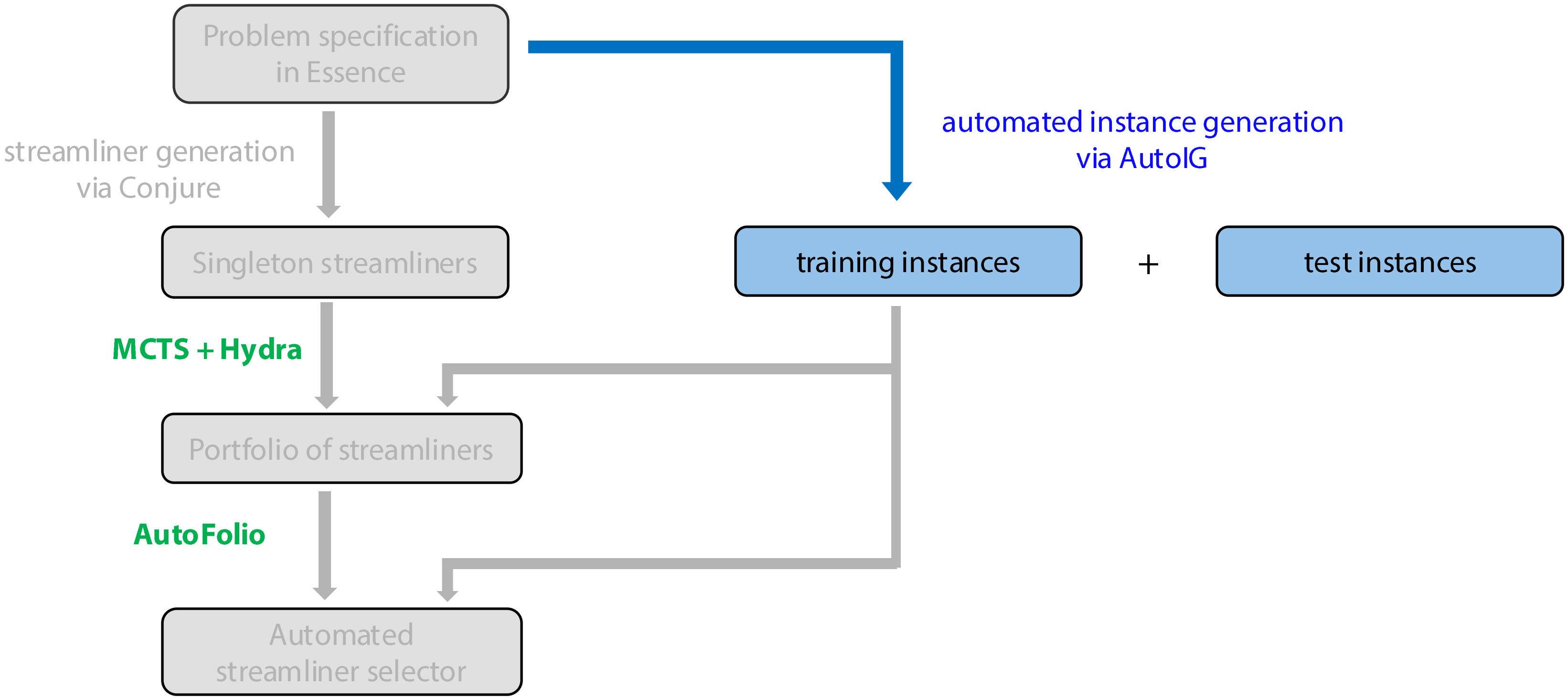
Automated streamliner selection with AutoFolio

- fzn2feat as instance features
  - Amadini, Gabbrielli, Mauro (2014) *An enhanced features extractor for a portfolio of constraint solvers*. SAC

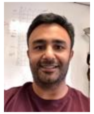
Automated Streamliner Selection



Automated Streamliner Selection



Automated Instance Generation



Özgür Akgün

Nguyen Dang

Joan Espasa

Ian Miguel

Peter Nightingale

Andras Salamon

Patrick Spracklen

Christopher Stone

LEVERHULME TRUST

Instance generation via instance generators. CP'19

Discriminating instance generation from abstract specifications: A case study with CP and MIP. CPAIOR'20

A Framework for Generating Informative Benchmark Instances. CP'22

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

Automated Instance Generation

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

AutoIG: a constraint-based automated instance generation tool

- Instances satisfy certain validity constraints
- Instances with certain properties regarding solvability
  - SAT, UNSAT, or both
  - **graded**: at a certain level of difficulty for a solver
  - **discriminating**: easy for one solver, difficult for another solver

Automated Instance Generation

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

AutoIG: a constraint-based automated instance generation tool

- Instances satisfy certain validity constraints
- Instances with certain properties regarding solvability
  - SAT, UNSAT, or both
  - graded: at a certain level of difficulty for a solver
  - discriminating: easy for one solver, difficult for another solver
- AutoIG supports generating instances in both Essence and MiniZinc

Experiments

Training instances (generated by AutoIG):

- SAT instances
- For MCTS:
  - "easy" instances: solved within [10s, 300s]
- For AutoFolio:
  - same instances as in MCTS
  - plus a small number of "hard" instances: solved within [300s, 3600s]

Test instances (generated by AutoIG):

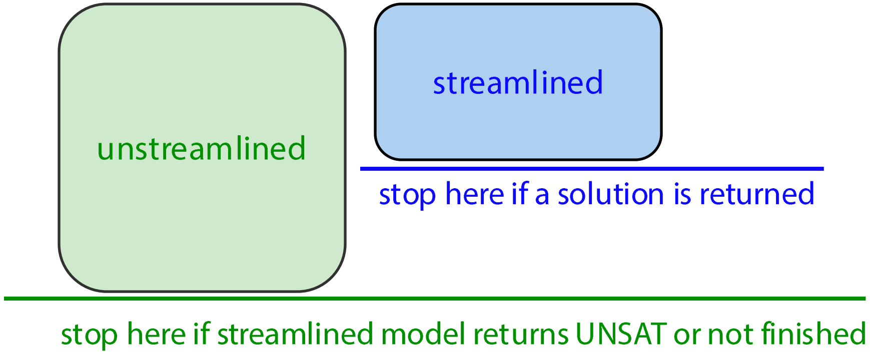
- SAT instances
- "hard": solved within [300s, 3600s]

Experiments

- 10 problems
- 2 solvers:
  - Chuffed: by Chu, Stuckey, Schutt, Ehlers, Gange, and Francis
  - Lingeling: by Armin Biere

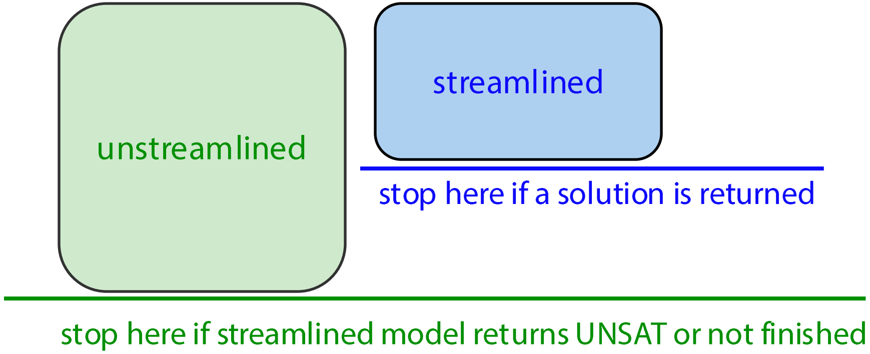
Experiments

- **A practical setting:** the selected streamlined model is run alongside the unstreamlined one



Experiments

- A practical setting: the selected streamlined model is run alongside the unstreamlined one



- **Performance metric:**

$$\textit{speedup} = \frac{\textit{runtime (unstreamlined)}}{\textit{runtime (streamlined)}}$$

$1 <$ speedup ≤ 2: gain in wall time but not CPU time

speedup > 2: gain in CPU time

for each test instance, pick the best streamliner in the portfolio

Solver	Problem	# Instances	Oracle	SBS	ApplicFirst	ReducFirst	Autofolio
Chuffed	BACP	16	53.47	1.47	2.48	4.71	**46.56**
	BIBD	59	2.25	1.13	1.15	1.04	**1.71**
	CarSequencing	52	8.77	1.91	1.88	2.19	**6.77**
	CoveringArray	46	3.36	2.20	1.26	1.26	**3.20**
	EFPA	121	4.86	1.02	1.93	1.79	**2.53**
	FLECC	192	3.95	2.18	2.02	1.68	**2.24**
	SocialGolfersProblem	19	2.53	1.28	1.00	1.00	**2.53**
	TailAssignment	35	3.20	**3.20**	**3.20**	1.21	**3.20**
	Transshipment	216	16.21	2.77	2.89	2.93	**5.39**
	VesselLoading	322	4.72	1.64	1.21	1.02	**2.12**
Lingeling	BACP	15	5.92	2.20	2.20	1.88	**4.91**
	BIBD	25	2.26	1.25	**1.39**	1.04	1.30
	CarSequencing	69	3.32	1.06	1.34	1.19	**2.95**
	CoveringArray	34	16.65	2.19	1.63	1.63	**10.81**
	EFPA	158	1.39	1.00	1.18	1.03	**1.20**
	FLECC	166	5.89	1.62	1.79	1.42	**3.39**
	SocialGolfersProblem	17	2.23	1.14	1.09	1.09	**1.89**
	TailAssignment	36	2.97	**2.95**	**2.95**	1.18	**2.95**
	Transshipment	68	12.42	3.59	3.55	3.60	**5.25**
	VesselLoading	78	2.51	1.29	1.11	1.80	**2.34**

the best overall streamliner
from the portfolio

Solver	Problem	# Instances	Oracle	SBS	ApplicFirst	ReducFirst	Autofolio
Chuffed	BACP	16	53.47	1.47	2.48	4.71	**46.56**
	BIBD	59	2.25	1.13	1.15	1.04	**1.71**
	CarSequencing	52	8.77	1.91	1.88	2.19	**6.77**
	CoveringArray	46	3.36	2.20	1.26	1.26	**3.20**
	EFPA	121	4.86	1.02	1.93	1.79	**2.53**
	FLECC	192	3.95	2.18	2.02	1.68	**2.24**
	SocialGolfersProblem	19	2.53	1.28	1.00	1.00	**2.53**
	TailAssignment	35	3.20	**3.20**	**3.20**	1.21	**3.20**
	Transshipment	216	16.21	2.77	2.89	2.93	**5.39**
	VesselLoading	322	4.72	1.64	1.21	1.02	**2.12**
Lingeling	BACP	15	5.92	2.20	2.20	1.88	**4.91**
	BIBD	25	2.26	1.25	**1.39**	1.04	1.30
	CarSequencing	69	3.32	1.06	1.34	1.19	**2.95**
	CoveringArray	34	16.65	2.19	1.63	1.63	**10.81**
	EFPA	158	1.39	1.00	1.18	1.03	**1.20**
	FLECC	166	5.89	1.62	1.79	1.42	**3.39**
	SocialGolfersProblem	17	2.23	1.14	1.09	1.09	**1.89**
	TailAssignment	36	2.97	**2.95**	**2.95**	1.18	**2.95**
	Transshipment	68	12.42	3.59	3.55	3.60	**5.25**
	VesselLoading	78	2.51	1.29	1.11	1.80	**2.34**

streamliners (from the portfolio) selected and applied sequentially based on their applicability / solving time reduction

Solver	Problem	# Instances	Oracle	SBS	ApplicFirst	ReducFirst	Autofolio
Chuffed	BACP	16	53.47	1.47	2.48	4.71	**46.56**
	BIBD	59	2.25	1.13	1.15	1.04	**1.71**
	CarSequencing	52	8.77	1.91	1.88	2.19	**6.77**
	CoveringArray	46	3.36	2.20	1.26	1.26	**3.20**
	EFPA	121	4.86	1.02	1.93	1.79	**2.53**
	FLECC	192	3.95	2.18	2.02	1.68	**2.24**
	SocialGolfersProblem	19	2.53	1.28	1.00	1.00	**2.53**
	TailAssignment	35	3.20	**3.20**	**3.20**	1.21	**3.20**
	Transshipment	216	16.21	2.77	2.89	2.93	**5.39**
	VesselLoading	322	4.72	1.64	1.21	1.02	**2.12**
Lingeling	BACP	15	5.92	2.20	2.20	1.88	**4.91**
	BIBD	25	2.26	1.25	**1.39**	1.04	1.30
	CarSequencing	69	3.32	1.06	1.34	1.19	**2.95**
	CoveringArray	34	16.65	2.19	1.63	1.63	**10.81**
	EFPA	158	1.39	1.00	1.18	1.03	**1.20**
	FLECC	166	5.89	1.62	1.79	1.42	**3.39**
	SocialGolfersProblem	17	2.23	1.14	1.09	1.09	**1.89**
	TailAssignment	36	2.97	**2.95**	**2.95**	1.18	**2.95**
	Transshipment	68	12.42	3.59	3.55	3.60	**5.25**
	VesselLoading	78	2.51	1.29	1.11	1.80	**2.34**

automated algorithm selection

Solver	Problem	# Instances	Oracle	SBS	ApplicFirst	ReducFirst	Autofolio
Chuffed	BACP	16	53.47	1.47	2.48	4.71	**46.56**
	BIBD	59	2.25	1.13	1.15	1.04	**1.71**
	CarSequencing	52	8.77	1.91	1.88	2.19	**6.77**
	CoveringArray	46	3.36	2.20	1.26	1.26	**3.20**
	EFPA	121	4.86	1.02	1.93	1.79	**2.53**
	FLECC	192	3.95	2.18	2.02	1.68	**2.24**
	SocialGolfersProblem	19	2.53	1.28	1.00	1.00	**2.53**
	TailAssignment	35	3.20	**3.20**	**3.20**	1.21	**3.20**
	Transshipment	216	16.21	2.77	2.89	2.93	**5.39**
	VesselLoading	322	4.72	1.64	1.21	1.02	**2.12**
Lingeling	BACP	15	5.92	2.20	2.20	1.88	**4.91**
	BIBD	25	2.26	1.25	**1.39**	1.04	1.30
	CarSequencing	69	3.32	1.06	1.34	1.19	**2.95**
	CoveringArray	34	16.65	2.19	1.63	1.63	**10.81**
	EFPA	158	1.39	1.00	1.18	1.03	**1.20**
	FLECC	166	5.89	1.62	1.79	1.42	**3.39**
	SocialGolfersProblem	17	2.23	1.14	1.09	1.09	**1.89**
	TailAssignment	36	2.97	**2.95**	**2.95**	1.18	**2.95**
	Transshipment	68	12.42	3.59	3.55	3.60	**5.25**
	VesselLoading	78	2.51	1.29	1.11	1.80	**2.34**

Summary

- It works, in most cases ☺

  But there's definitely room for improvement

What's next?

- More fine-grained streamliner generation (softness parameters)

- More cost-effective streamliner search for improved generalisation
  - racing & adaptive capping

- More informative and cost-effective instance features
  - **<u>Pellegrino</u>**, Akgün, Dang, Kiziltan, and Miguel. Transformer-based Feature Learning for Algorithm Selection in Combinatorial Optimisation. CP B (JMS 745) – Tuesday 12:00

What's next?

- Leveraging context information during the streamliner generation process
  - Using no-goods to identify promising streamliners

    Yazicilar, Akgun, and Miguel (2024) *Automated nogood-filtered fine-grained streamlining: a case study on covering arrays*.
  - LLMs instead of rule-based streamliner generation

    Voboril, Ramaswamy, and Szeider (2024) *Generating streamlining constraints with large language models*.

- Learning across similar problems

What's next?

- Automated streamliners for optimisation problems
  - Voboril, Ramaswamy, and Szeider. "*Balancing Latin Rectangles with LLM-generated Streamliners*"
    CP 2025 Application track – Tuesday 14:30 – JMS 745
  - Spracklen, Dang, Akgün, and Miguel. "Automatic streamlining for constrained optimisation". CP 2019