Machine Learning to Accelerate Solving of Constraint Programs

Neil Yorke-Smith
n.yorke-smith@tudelft.nl

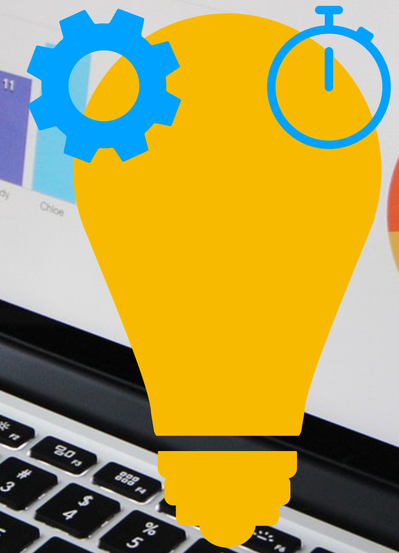
TAILOR
Foundation of Trustworthy AI:
Integrating Learning, Optimisation and Reasoning

TU Delft

STAR Lab

learning from data
(System 1)

inference over representations
(System 2)

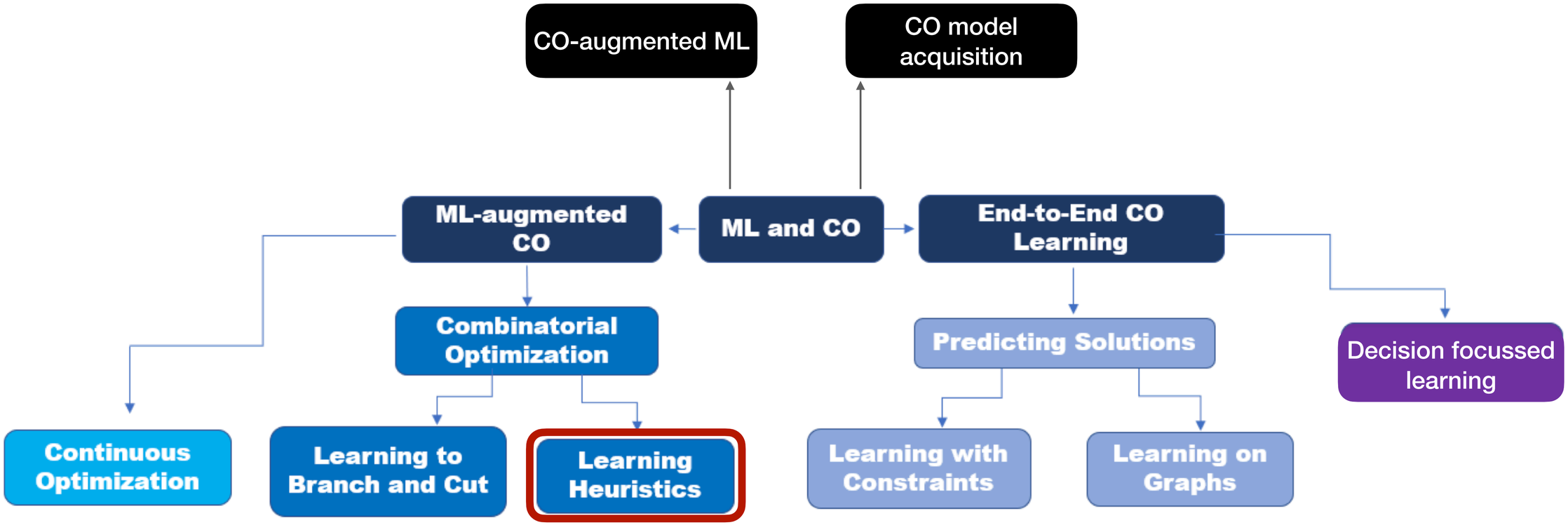


objectives

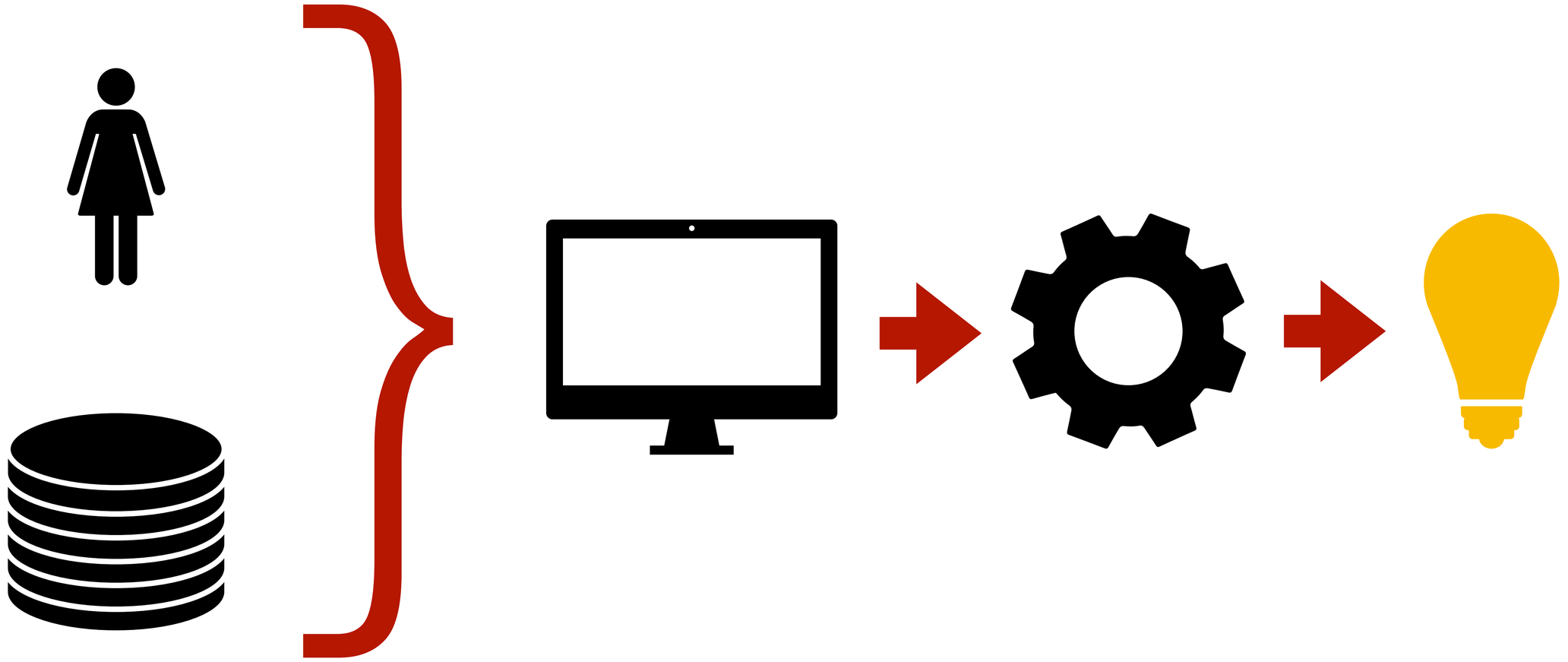
constraints

decisions

“Main challenge is two-way integration of learners and solvers.” — Geffner



James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck, Bryan Wilder:
End-to-End Constrained Optimization Learning: A Survey. IJCAI 2021: 4475-4482



1

ML to configure CP-SAT solver

2

ML to learn CP heuristics on-the-fly

3

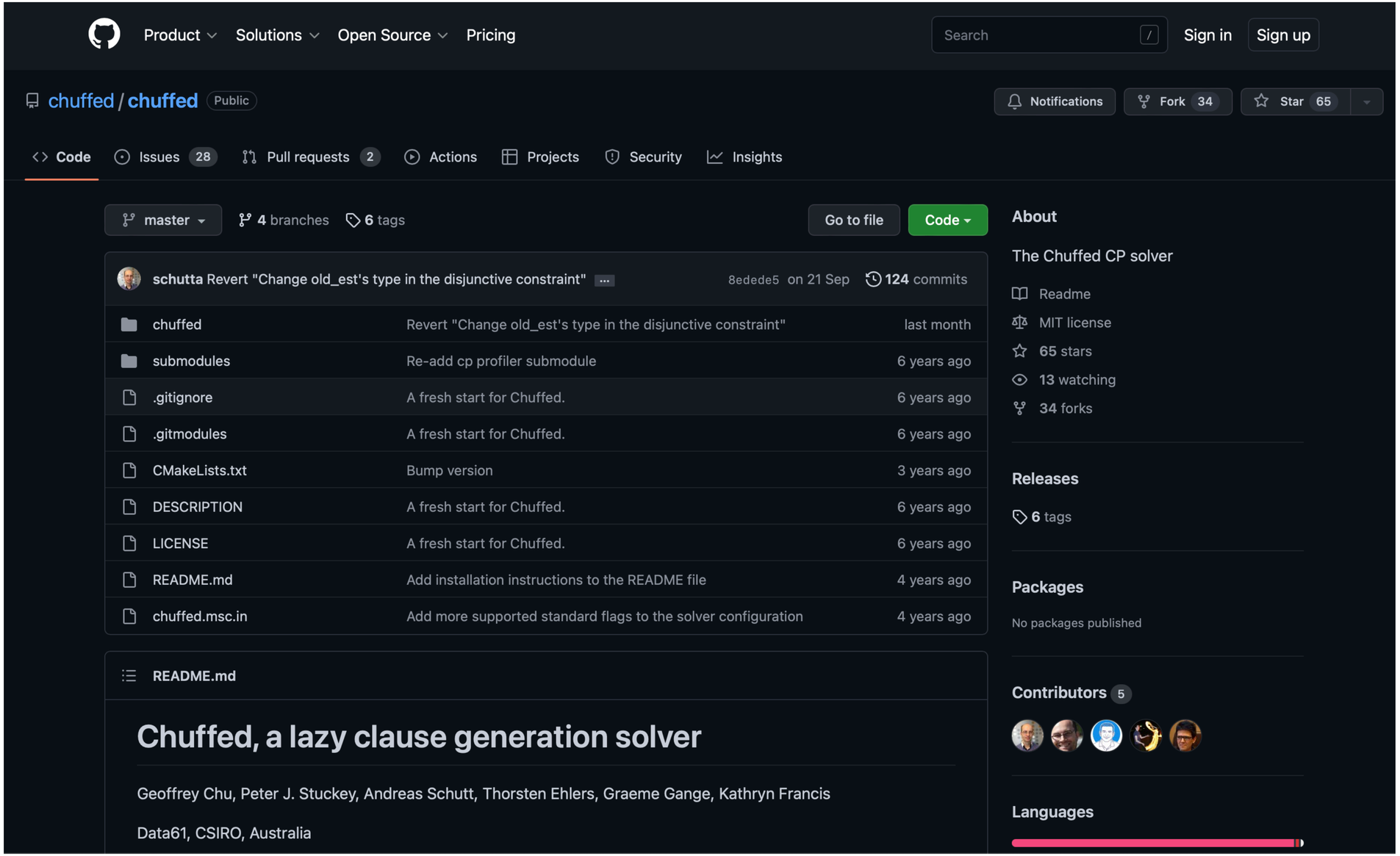
ML to embed forecasts within a CP model

Learning Variable Activity Initialisation for Lazy Clause Generation Solvers

Ronald van Driel, Emir Demirović, **Neil Yorke-Smith**
n.yorke-smith@tudelft.nl

TU Delft

improving the performance of Chuffed, a Lazy-Clause-Generation solver



improving the performance of Chuffed, a Lazy-Clause-Generation solver

improving the performance of Chuffed, a Lazy-Clause-Generation solver,
by influencing the LCG variable selection heuristic

improving the performance of Chuffed, a Lazy-Clause-Generation solver,
by using machine learning to predict unsatisfiable cores,
and using those predictions to initialise VSIDS variable weights

Why?

ML + OR

ML + CP

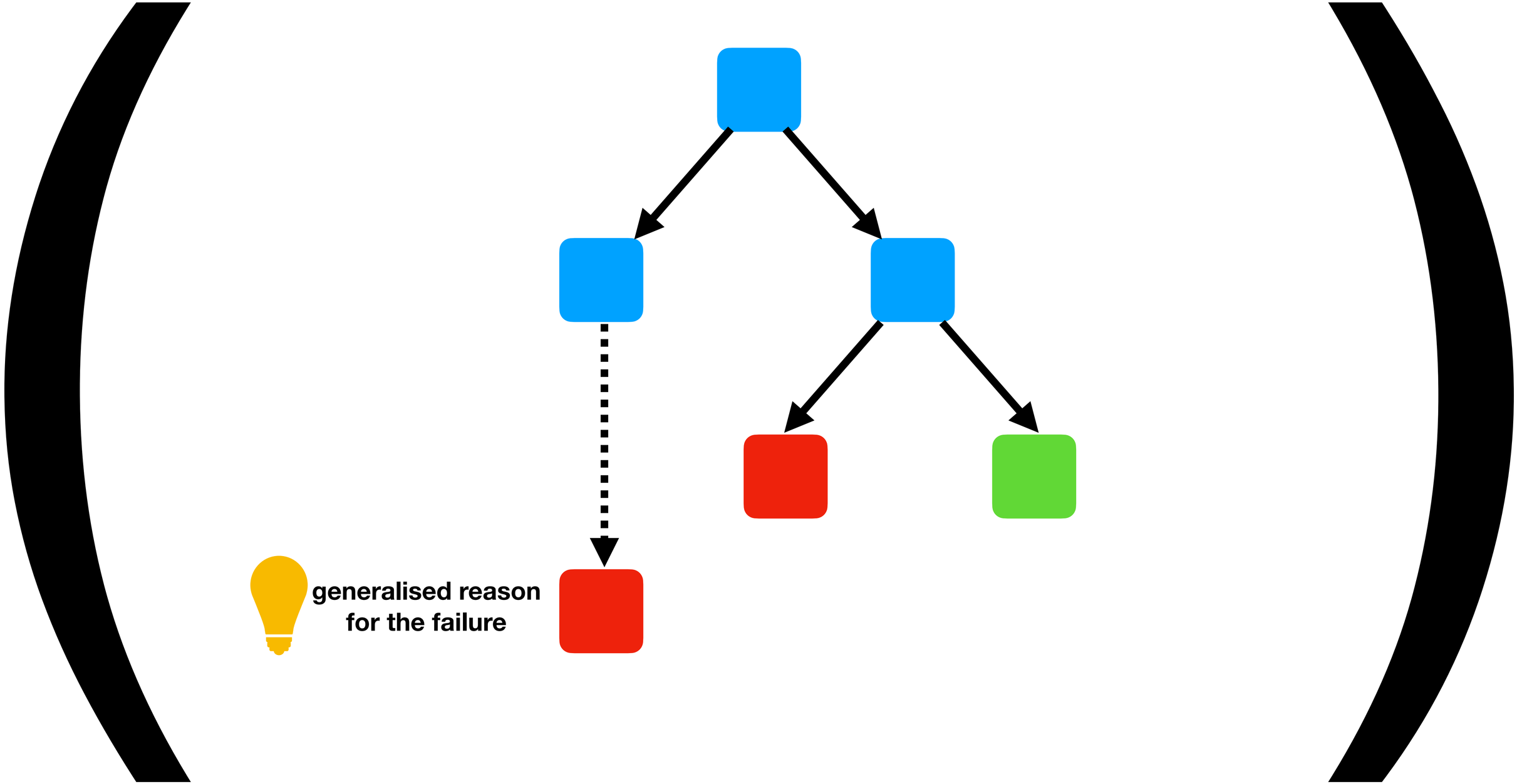
ML + SAT

SAT + CP

LCG

ML + LCG?

idea: can we use ML to accelerate a LCG solver?



Guiding High-Performance SAT Solvers with Unsat-Core Predictions

Daniel Selsam[1(✉)] and Nikolaj Bjørner[2]

[1] Stanford University, Stanford, CA 94305, USA
dselsam@cs.stanford.edu
[2] Microsoft Research, Redmond, WA 98052, USA

Abstract. The *NeuroSAT* neural network architecture was introduced in [37] for predicting properties of propositional formulae. When trained to predict the satisfiability of toy problems, it was shown to find solutions and unsatisfiable cores on its own. However, the authors saw "no obvious path" to using the architecture to improve the state-of-the-art. In this work, we train a simplified NeuroSAT architecture to directly predict the unsatisfiable cores of real problems. We modify several state-of-the-art SAT solvers to periodically replace their variable activity scores with NeuroSAT's prediction of how likely the variables are to appear in an unsatisfiable core. The modified MiniSat solves 10% more problems on SATCOMP-2018 within the standard 5,000 second timeout than the original does. The modified Glucose solves 11% more problems than the original, while the modified Z3 solves 6% more. The gains are even greater when the training is specialized for a specific distribution of problems; on a benchmark of hard problems from a scheduling domain, the modified Glucose solves 20% more problems than the original does within a one-hour timeout. Our results demonstrate that NeuroSAT can provide effective guidance to high-performance SAT solvers on real problems.

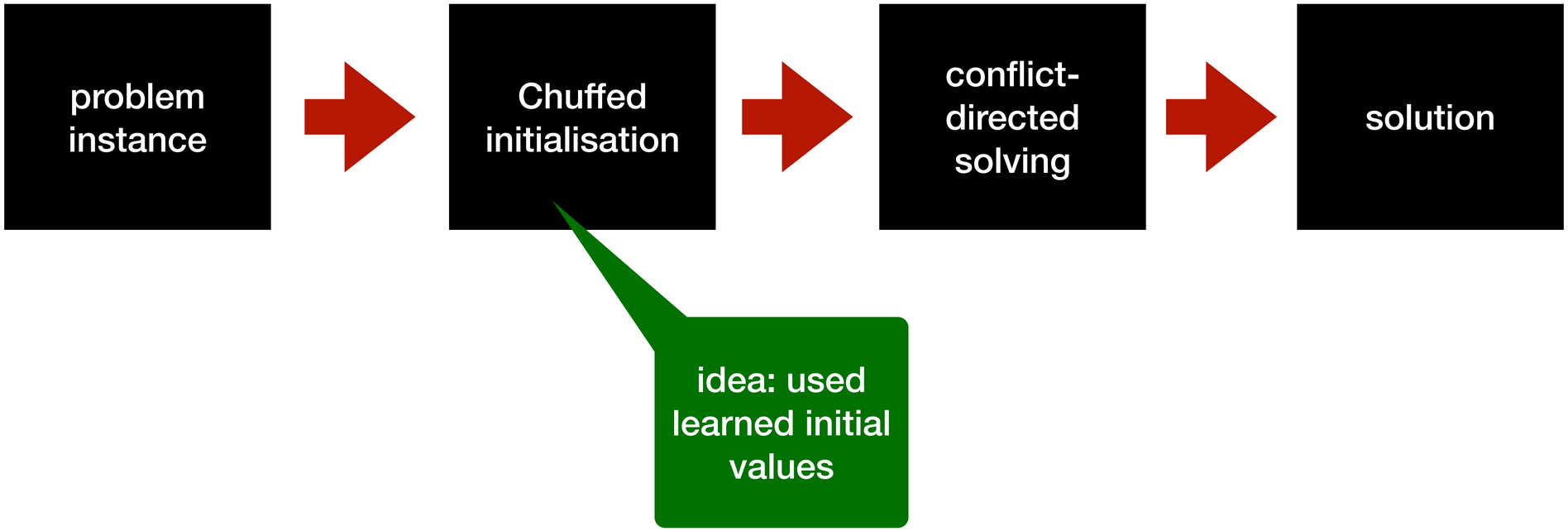
1 Introduction

Over the past decade, neural networks have dramatically advanced the state of the art on many important problems, most notably object recognition [22], speech recognition [13], and machine translation [45]. There have also been several attempts to apply neural networks to problems in discrete search, such as program synthesis [7,33], first-order theorem proving [17,27] and higher-order theorem proving [16,18,42,44]. More recently, [37] introduce a neural network architecture designed for satisfiability problems, called *NeuroSAT*, and show that when trained to predict satisfiability on toy problems, it learns to find solutions and unsatisfiable cores on its own. Moreover, the neural network is iterative, and the authors show that by running for many more iterations at test time, it can solve problems that are bigger and even from completely different domains

D. Selsam—This paper describes work performed while the first author was at Microsoft Research.

M. Janota and I. Lynce (Eds.): SAT 2019, LNCS 11628, pp. 336–353, 2019.
https://doi.org/10.1007/978-3-030-24258-9_24

What?



How?

train on unsatisfiable CP instances

predict unsat cores for satisfiable instances

initialise Chuffed's VSIDS scores based on predictions

Chuffed regular solving

Data

approach requires both satisfiable data and unsatisfiable data

no public unsatisfiable datasets

make unsat instances from MiniZinc benchmarks

extract unsat cores from each instance

How?

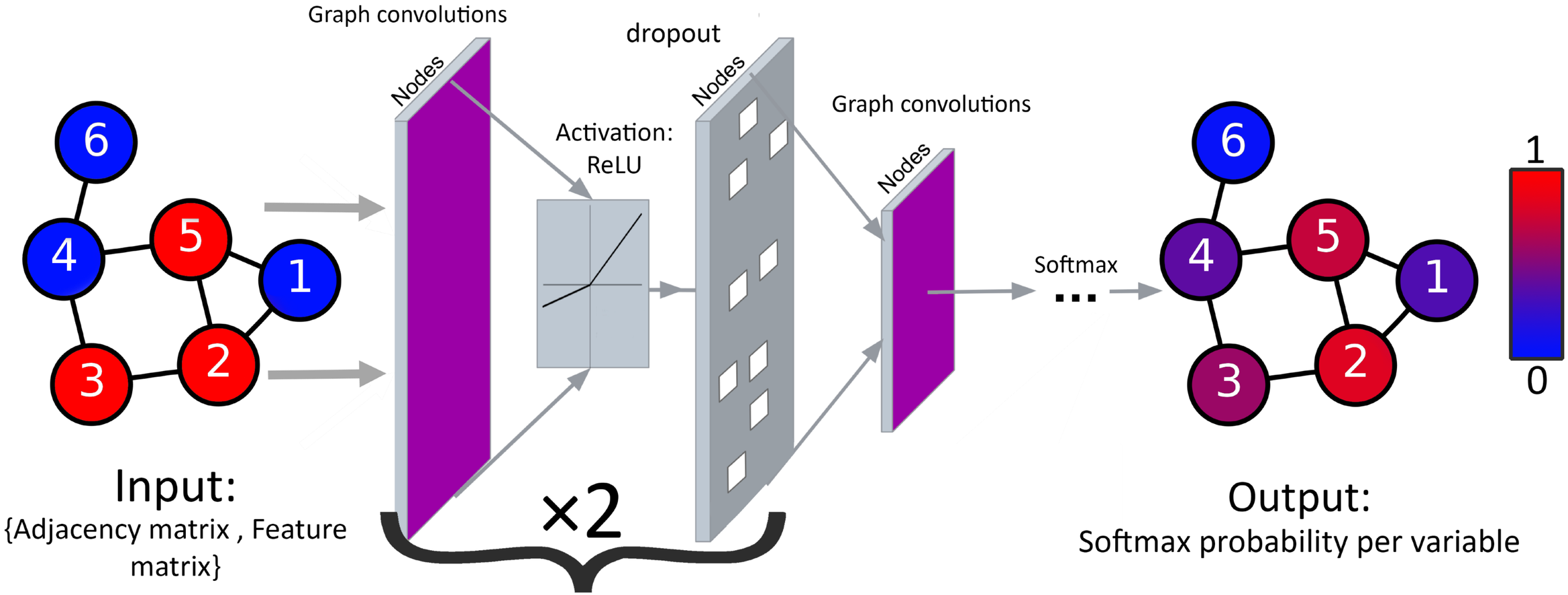
train on unsatisfiable CP instances

predict unsat cores for satisfiable instances

initialise Chuffed's VSIDS scores based on predictions

Chuffed regular solving

ML model



Input:
{Adjacency matrix , Feature matrix}

Output:
Softmax probability per variable

ML features

categorical features
if a variable is declared as a Boolean, integer, float or set

numerical features
min value, max value, range of the variable domain

if variable is part of an unsat core of the instance, the label is one, otherwise zero

use GCN output to initialise Chuffed's VSIDS scores

How?

train on unsatisfiable CP instances

predict unsat cores for satisfiable instances

initialise Chuffed's VSIDS scores based on predictions

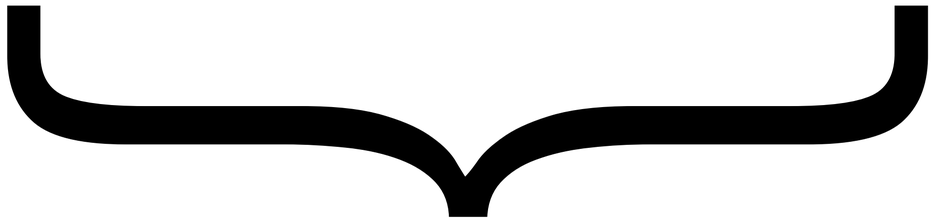
Chuffed regular solving

Experiments

baseline:
Chuffed0

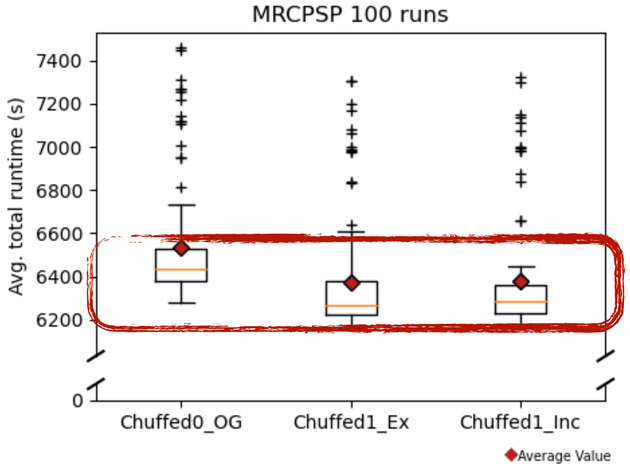
learned inits:
Chuffed1_Ex

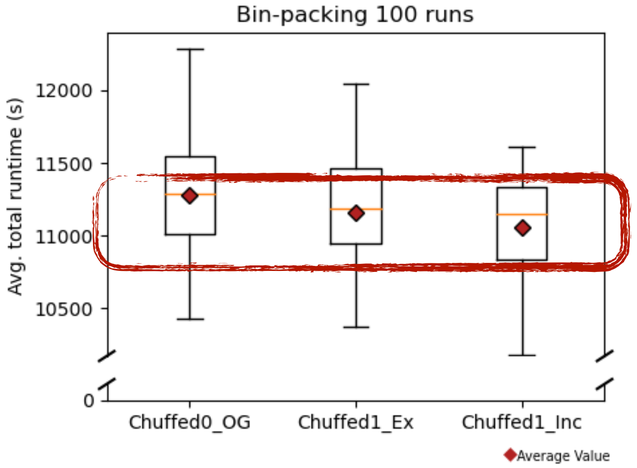
learned inits:
Chuffed1_Inc

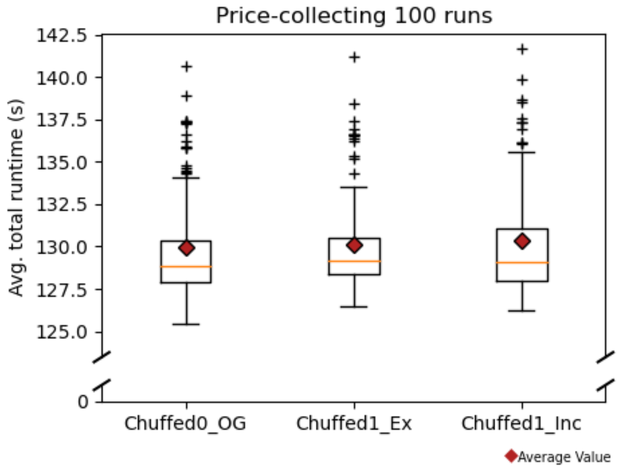


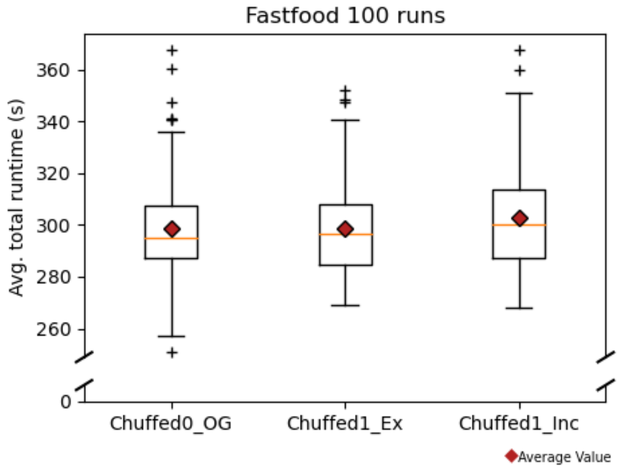
100 runs on 4 largest problem types

Results

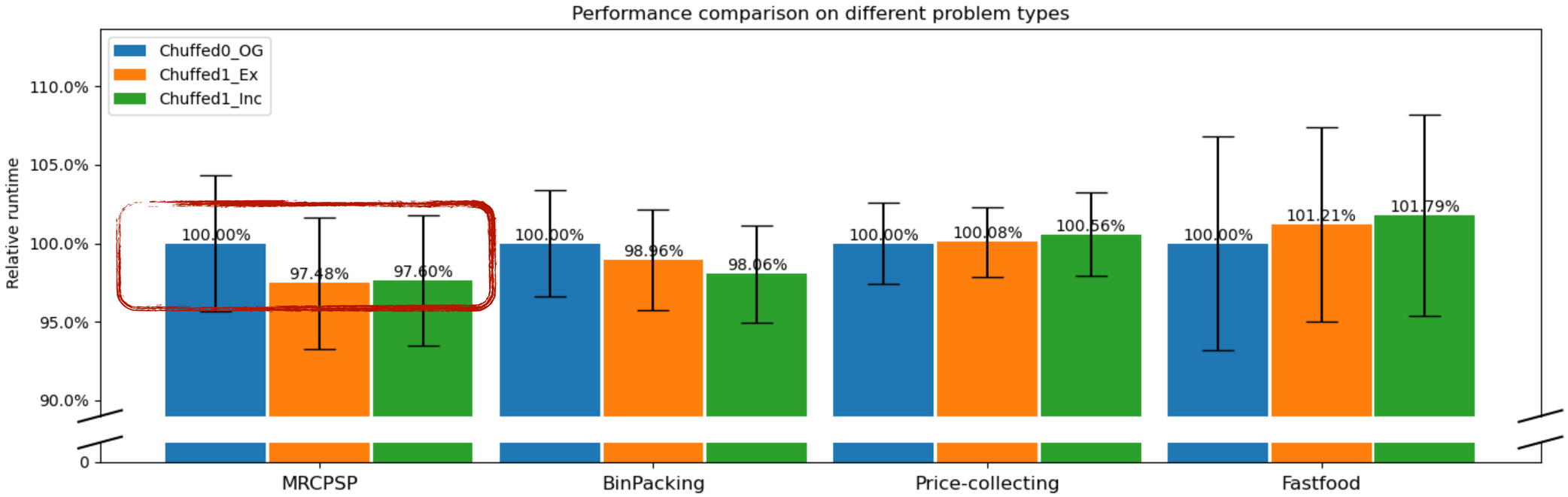








Chuffed + learned inits wins



improving the performance of Chuffed, a Lazy-Clause-Generation solver,
by using machine learning to predict unsatisfiable cores,
and using those predictions to initialise VSIDS variable weights

Lessons

promising results on MRCPSP and bin-packing: 1-2% speed increase

small difference between Excluding and Including → generalisation

no improvement on two other classes: not enough data?

compatible with other ML-improvements to LCG

What next?

- direct embedding
- more data
- other ML models
- periodic refocusing
- LCG parameter learning
- predicting no-goods

1

ML to configure CP-SAT solver

2

ML to learn CP heuristics on-the-fly

3

ML to embed forecasts within a CP model

Online Learning of Deeper Variable Ordering Heuristics for Constraint Optimisation

Floris Doolaard and **Neil Yorke-Smith**
n.yorke-smith@tudelft.nl

TU Delft

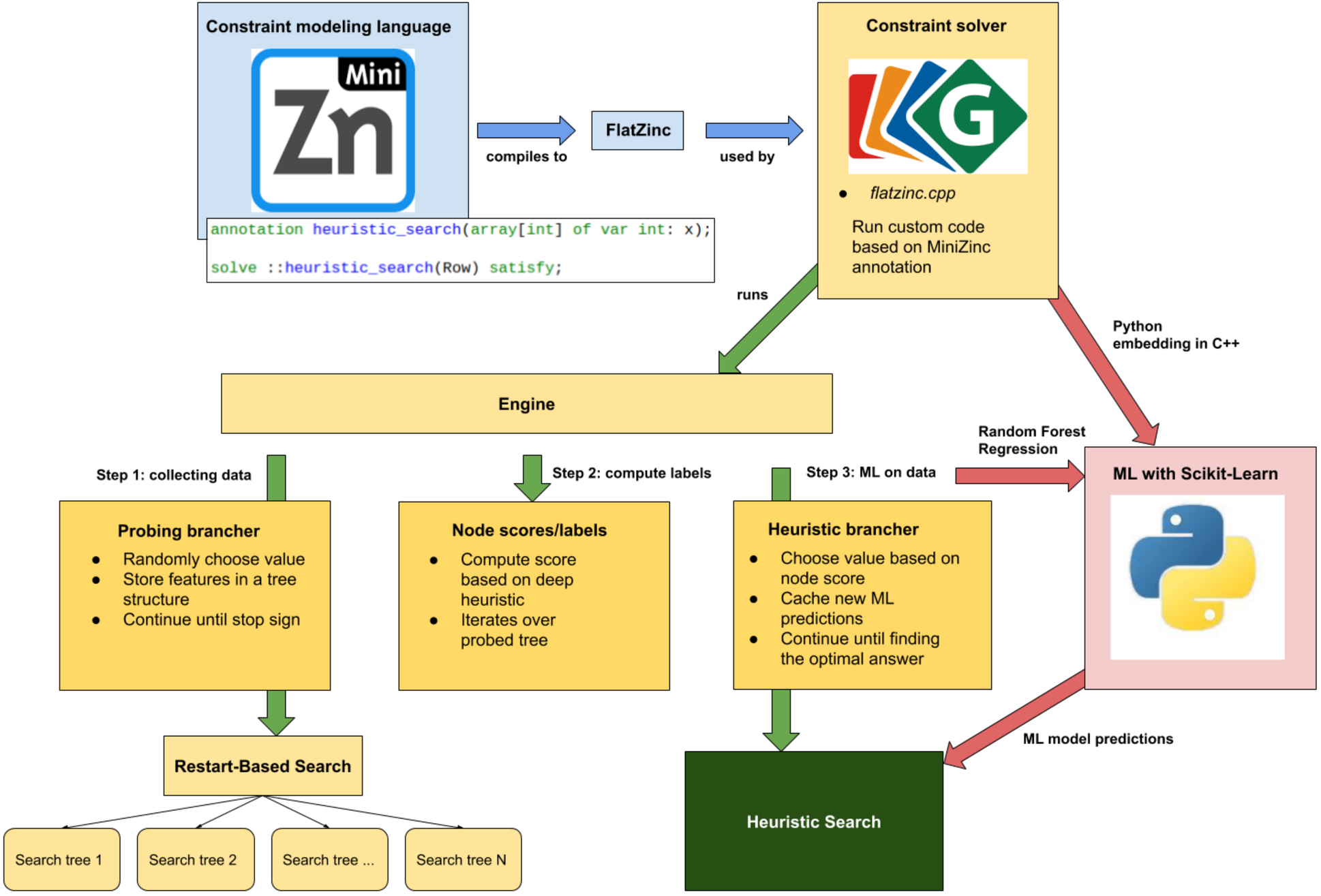
Ambition

Kristine Wook

one-shot learning of optimisation heuristics

learn instance-specific variable ordering for COPs

idea: probe, learn, solve



Learning Value Heuristics for Constraint Programming

Geoffrey Chu(✉) and Peter J. Stuckey

National ICT Australia, Victoria Laboratory, Department of Computing and Information Systems, University of Melbourne, Melbourne, Australia
{geoffrey.chu,pstuckey}@unimelb.edu.au

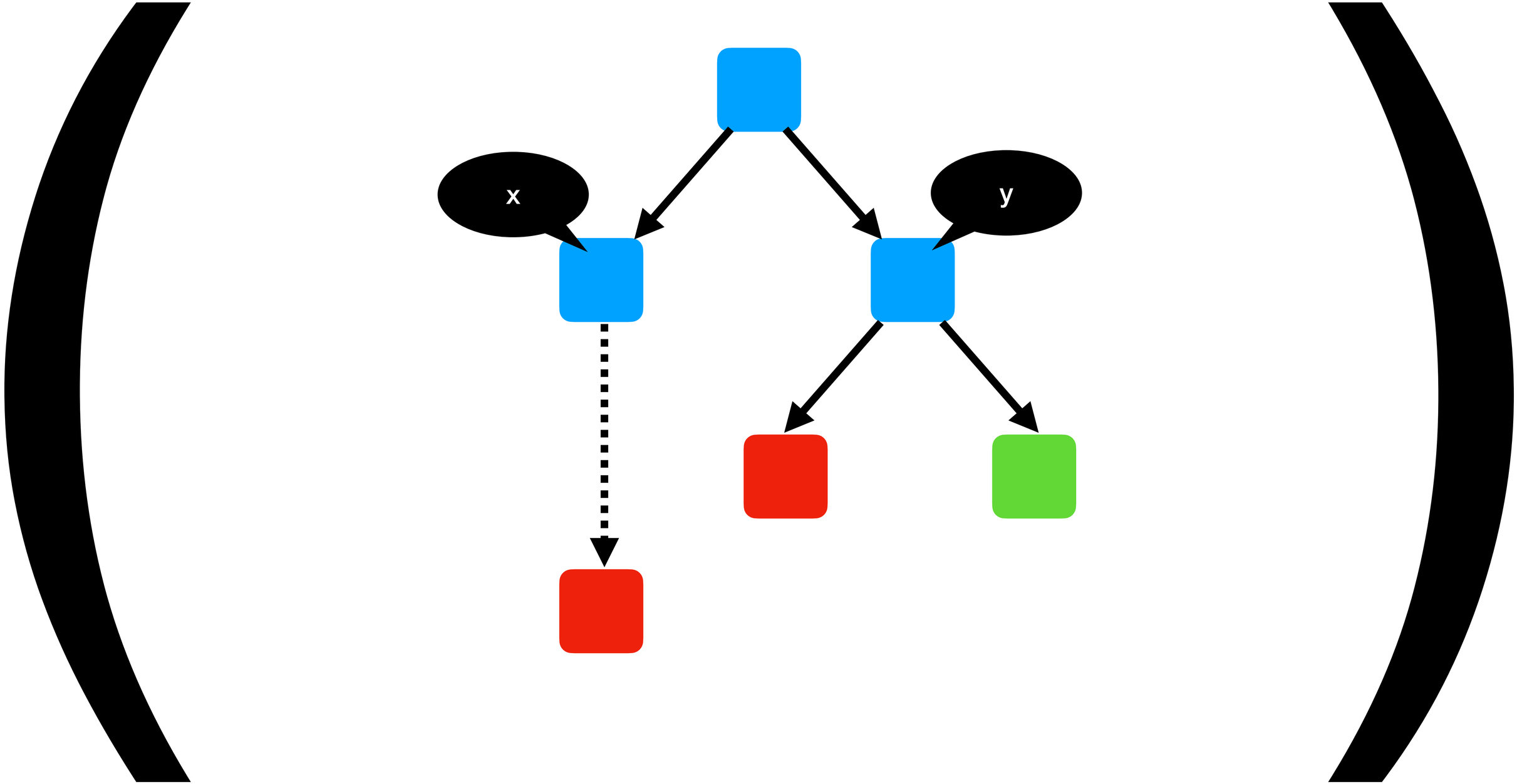
Abstract. Search heuristics are of paramount importance for finding good solutions to optimization problems quickly. Manually designing problem specific search heuristics is a time consuming process and requires expert knowledge from the user. Thus there is great interest in developing autonomous search heuristics which work well for a wide variety of problems. Various autonomous search heuristics already exist, such as first fail, domwdeg and impact based search. However, such heuristics are often more focused on the variable selection, i.e., picking important variables to branch on to make the search tree smaller, rather than the value selection, i.e., ordering the subtrees so that the good subtrees are explored first. In this paper, we define a framework for learning value heuristics, by combining a scoring function, feature selection, and machine learning algorithm. We demonstrate that we can learn value heuristics that perform better than random value heuristics, and for some problem classes, the learned heuristics are comparable in performance to manually designed value heuristics. We also show that value heuristics using features beyond a simple score can be valuable.

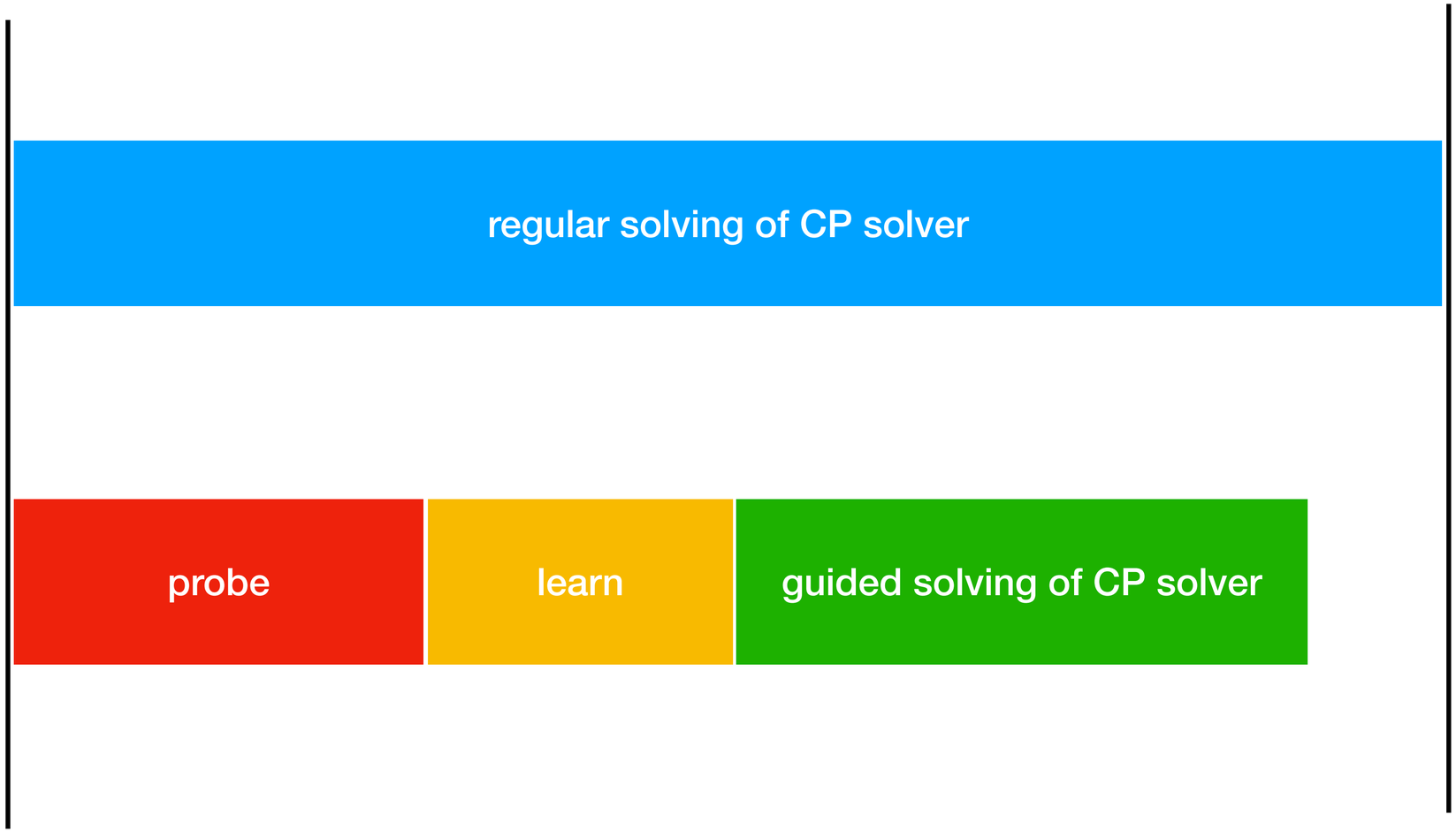
1 Introduction

Search heuristics are of paramount importance for finding good solutions to optimization problems quickly. Search heuristics can roughly be divided into two parts: the variable selection heuristic, which selects which variable to branch on, and the value heuristic, which determines which value is tried first. There has been significant research on autonomous search heuristics including: first fail [1], variable state independent decaying sum (VSIDS) [2], domain size divided by weighted degree (domwdeg) [3], impact based search [4], solution counting based search [5], and action[1] based search [6]. Most of these search heuristics concentrate on variable selection, as this is critical in reducing the size of the search tree, although some, in particular impact and action based search also generate value heuristics. Phase saving [7] if a value-only heuristic which reuses the last value of a Boolean variable (its phase) when it is reconsidered. In

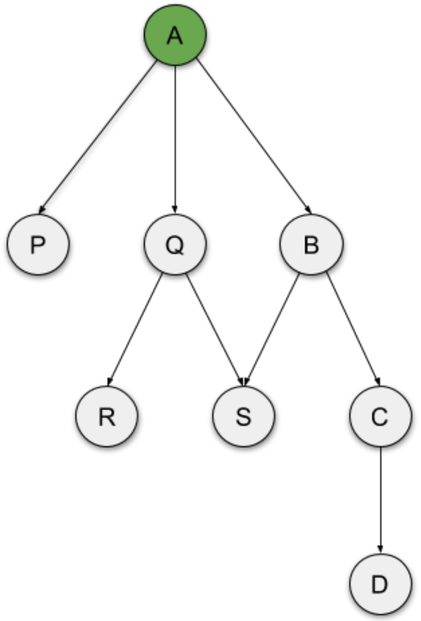
[1] It was originally called activity-based search, we use the alternate name to distinguish it from the long established activity-based search used in SAT, SMT and LCG solvers.

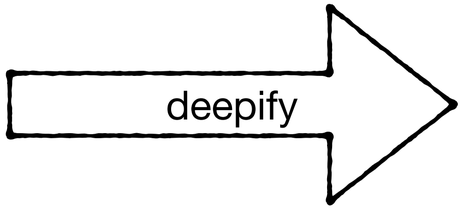
L. Michel (Ed.): CPAIOR 2015, LNCS 9075, pp. 108–123, 2015.
DOI: 10.1007/978-3-319-18008-3_8

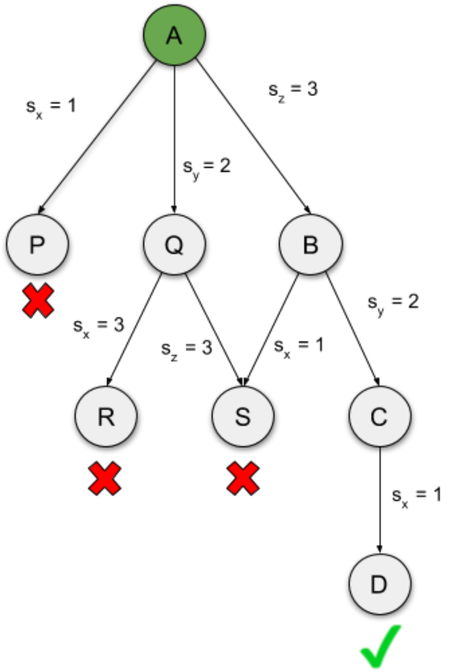


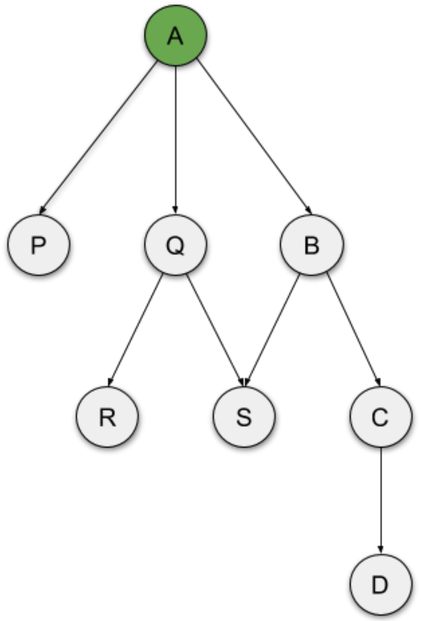


Idea

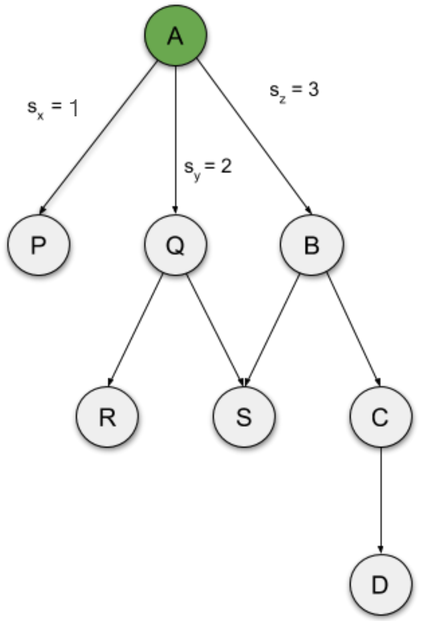




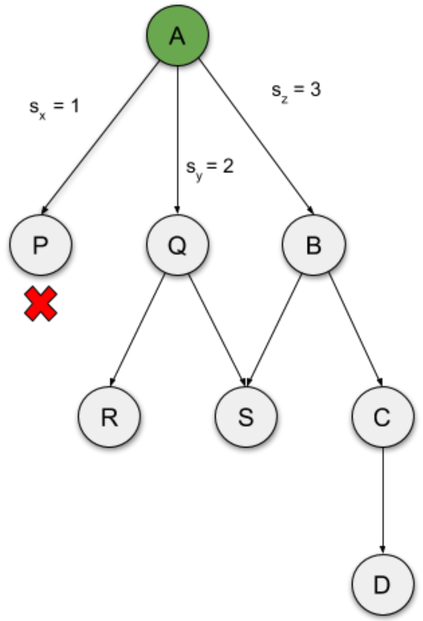




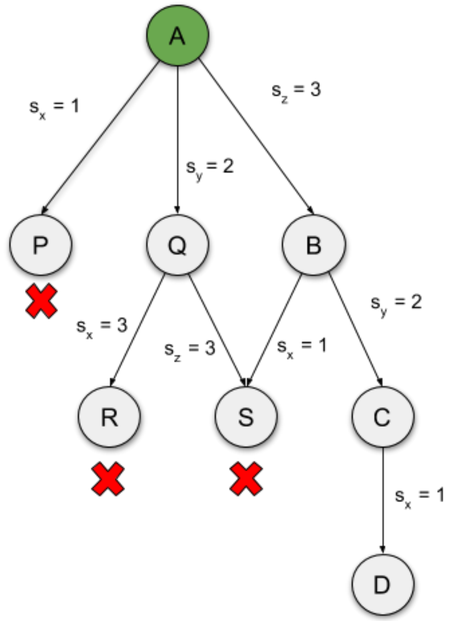
Variables: {x, y, z}
Pick variable with minimum score
Depth: d = 2



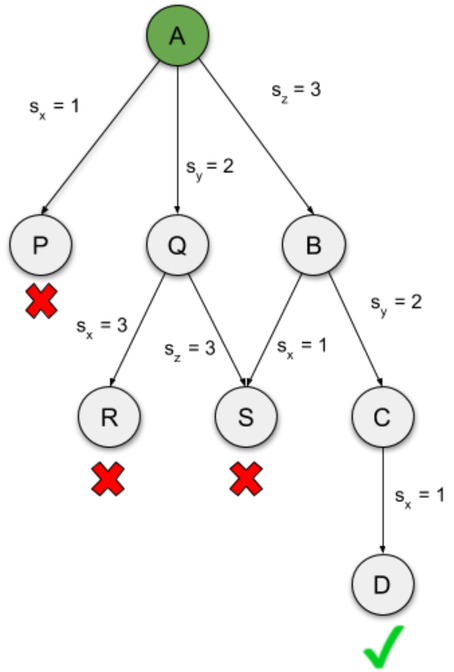
Variables: {x, y, z}
Pick variable with minimum score
Depth: d = 2



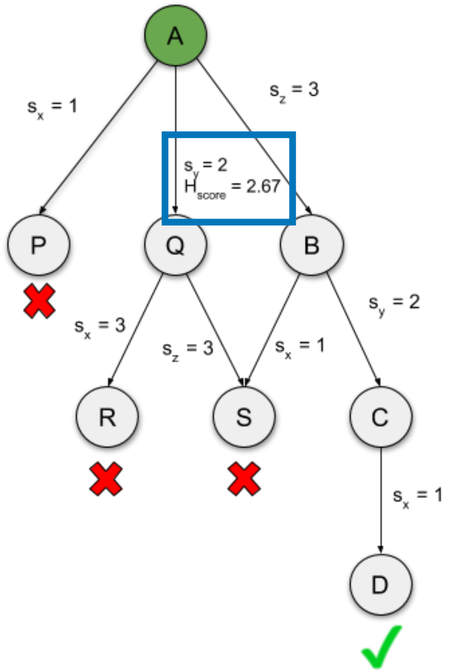
Variables: {x, y, z}
Pick variable with minimum score
Depth: d = 2



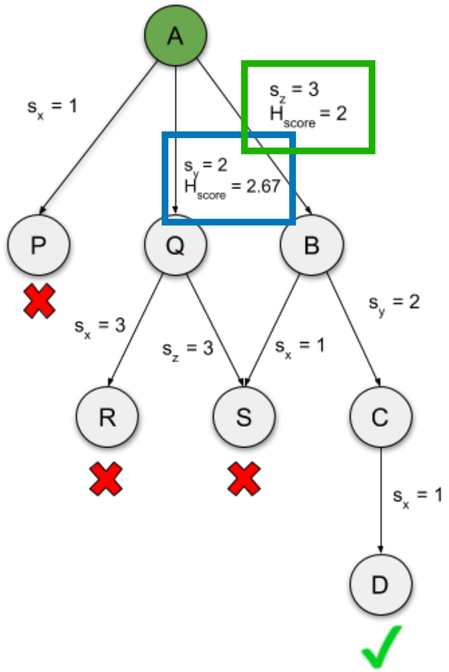
Variables: {x, y, z}
Pick variable with minimum score
Depth: d = 2



Variables: {x, y, z}
Pick variable with minimum score
Depth: d = 2

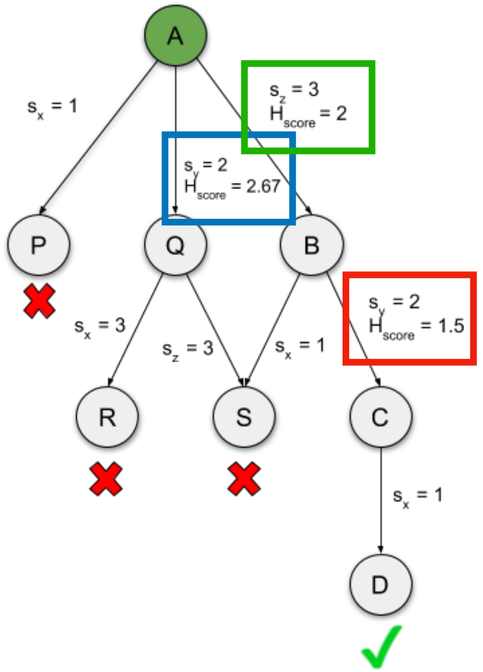


$H_{score} = (2 + 3 + 3) / 3 = 2.67$



$H_{score} = (2 + 3 + 3) / 3 = 2.67$

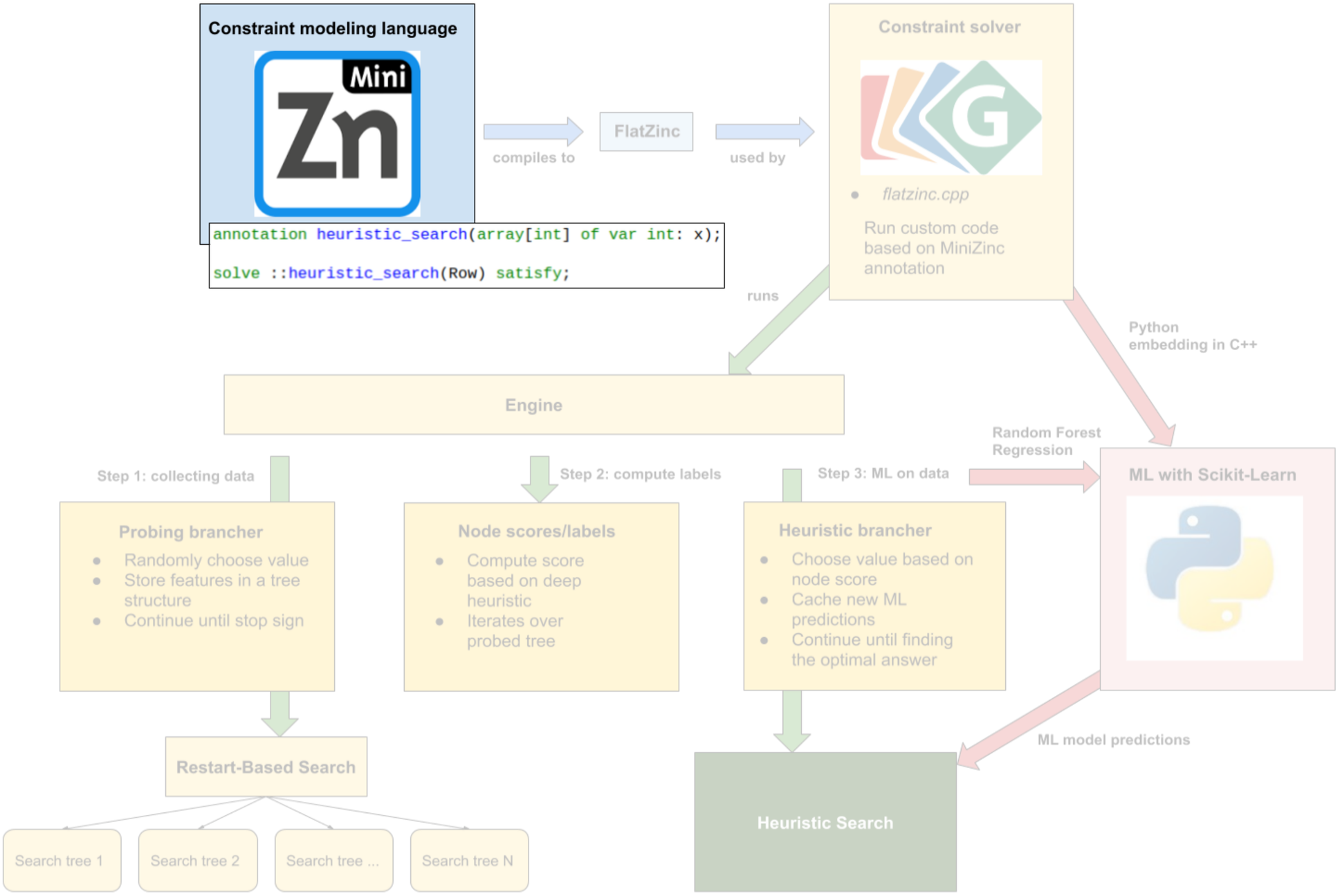
$H_{score} = (3 + 1 + 2) / 3 = 2$

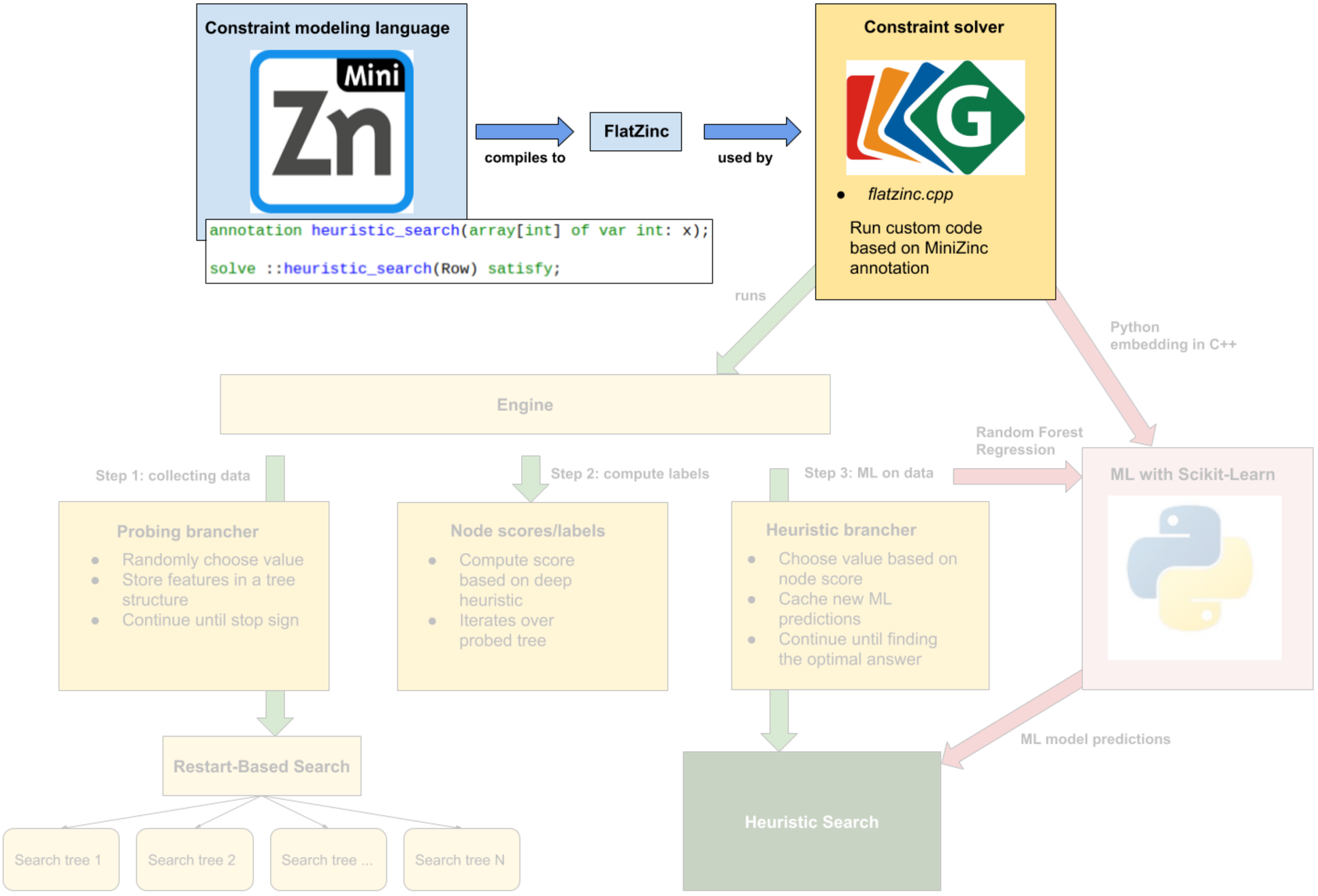


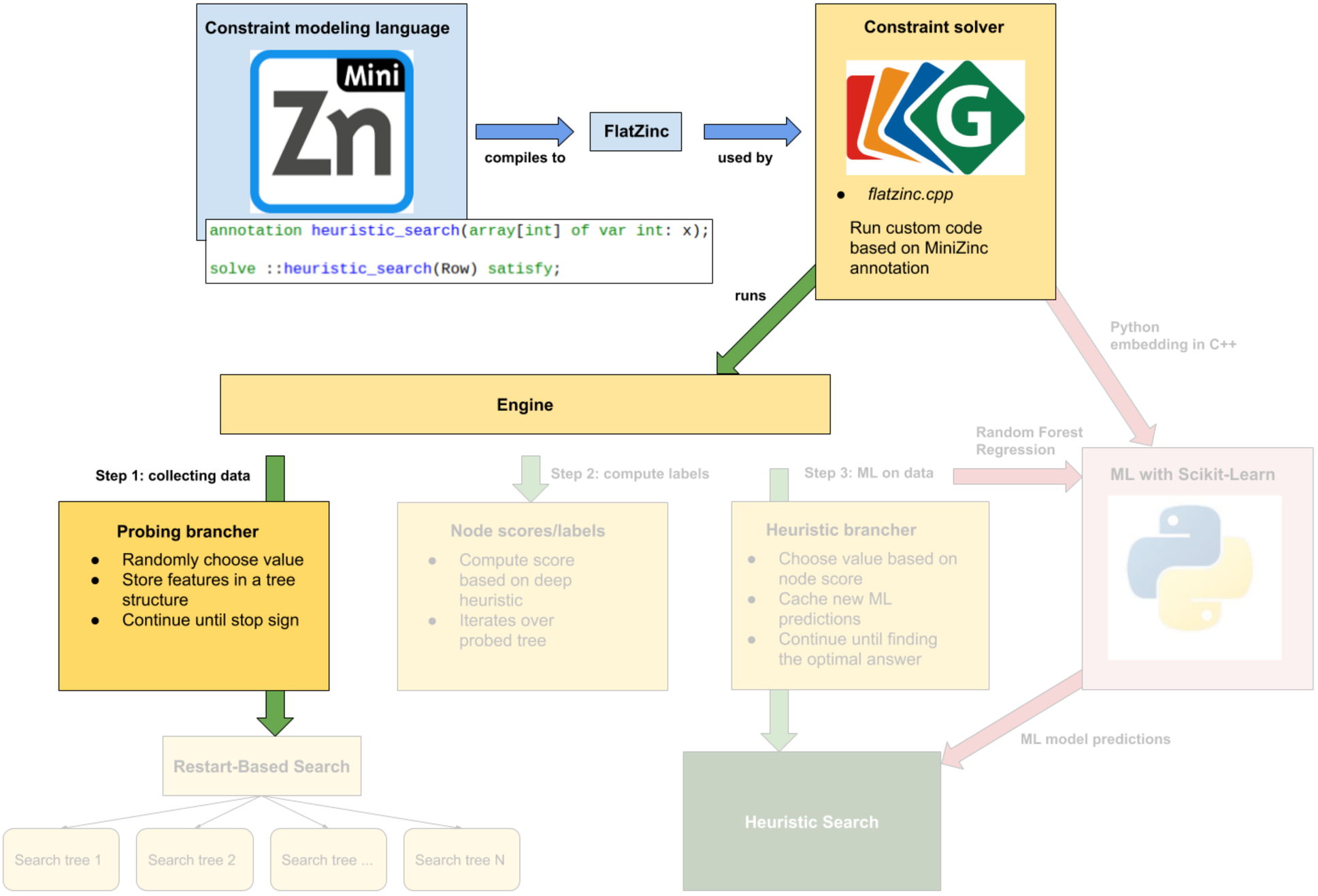
$H_{score} = (2 + 3 + 3) / 3 = 2.67$

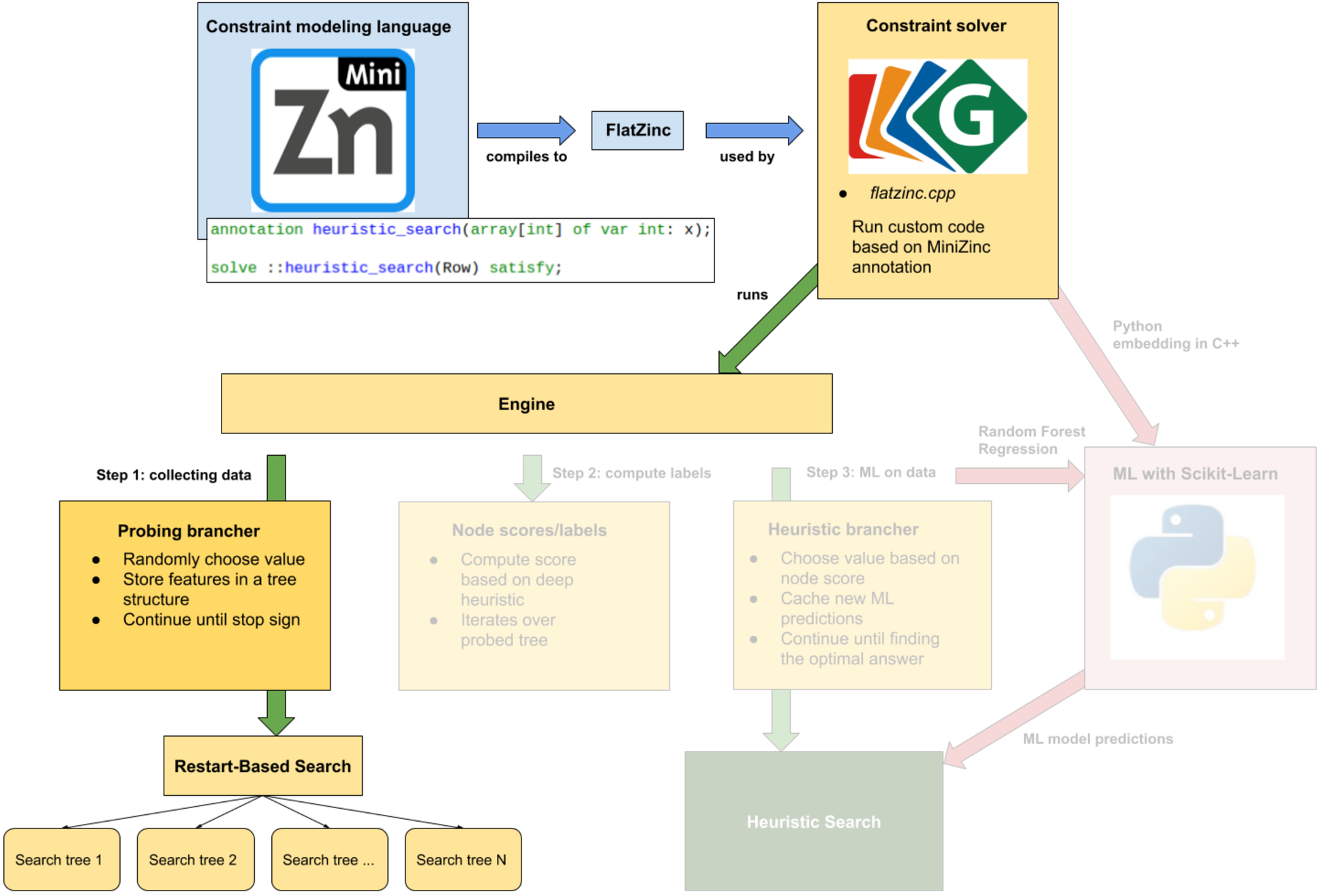
$H_{score} = (3 + 1 + 2) / 3 = 2$

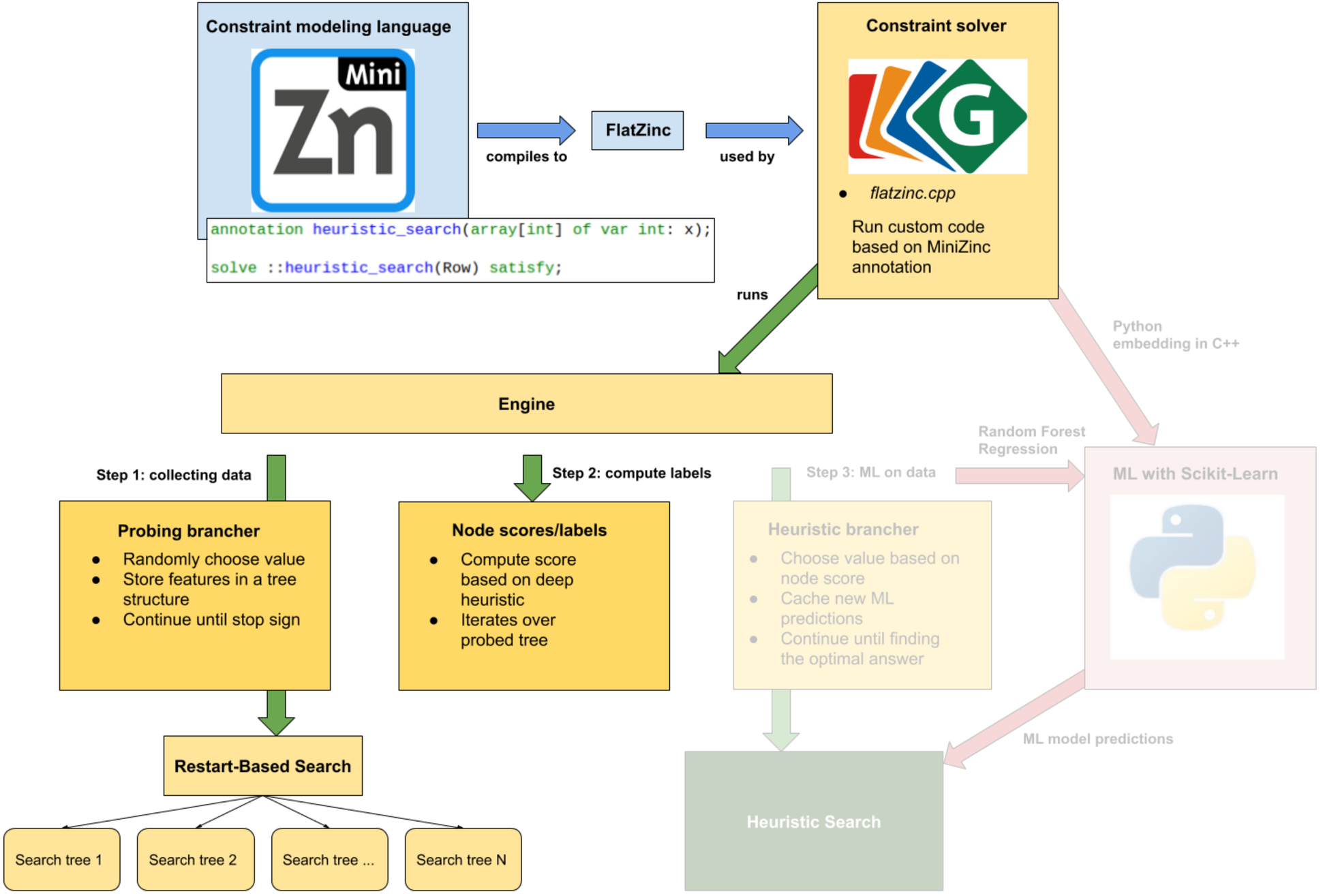
$H_{score} = (2 + 1) / 2 = 1.5$

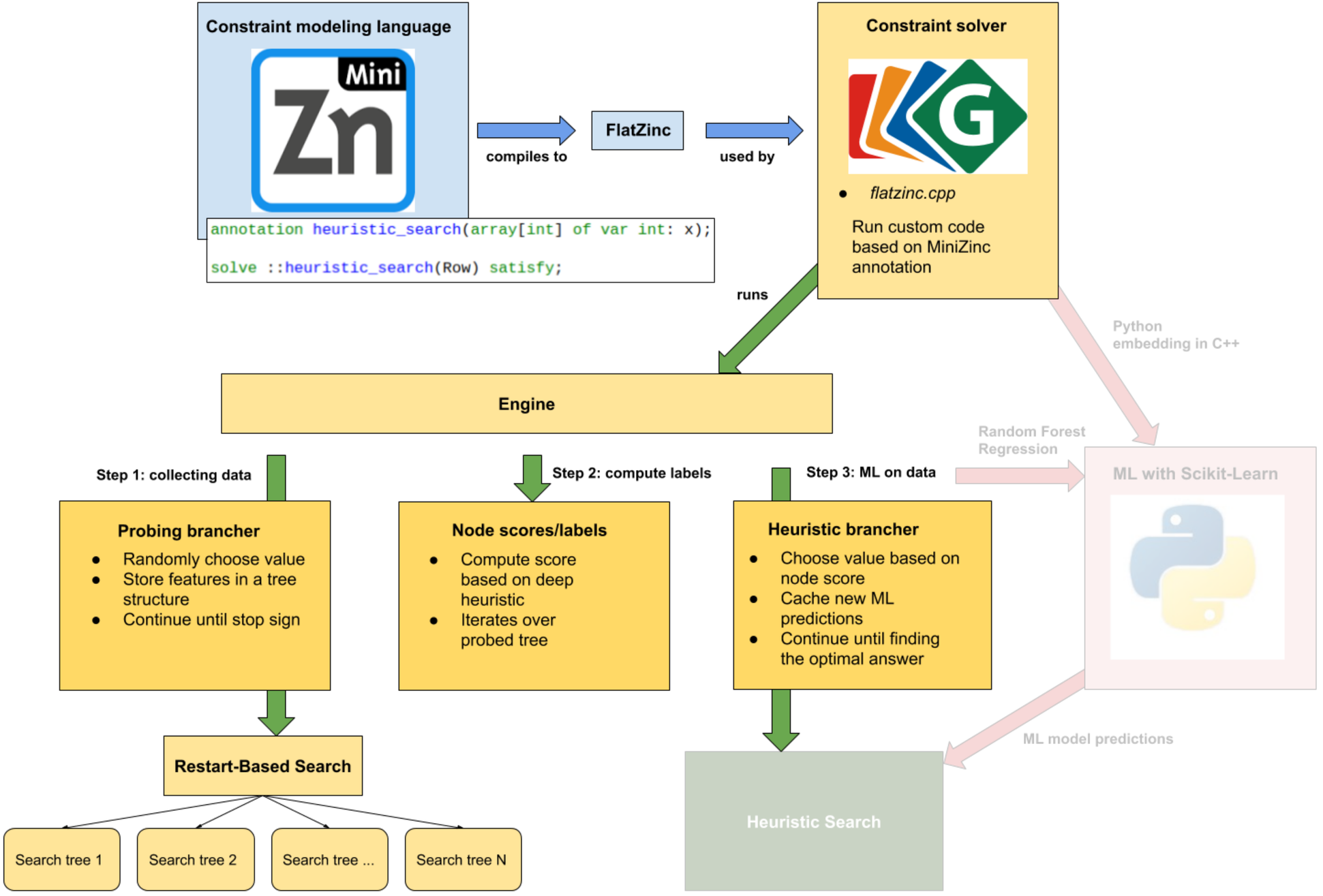


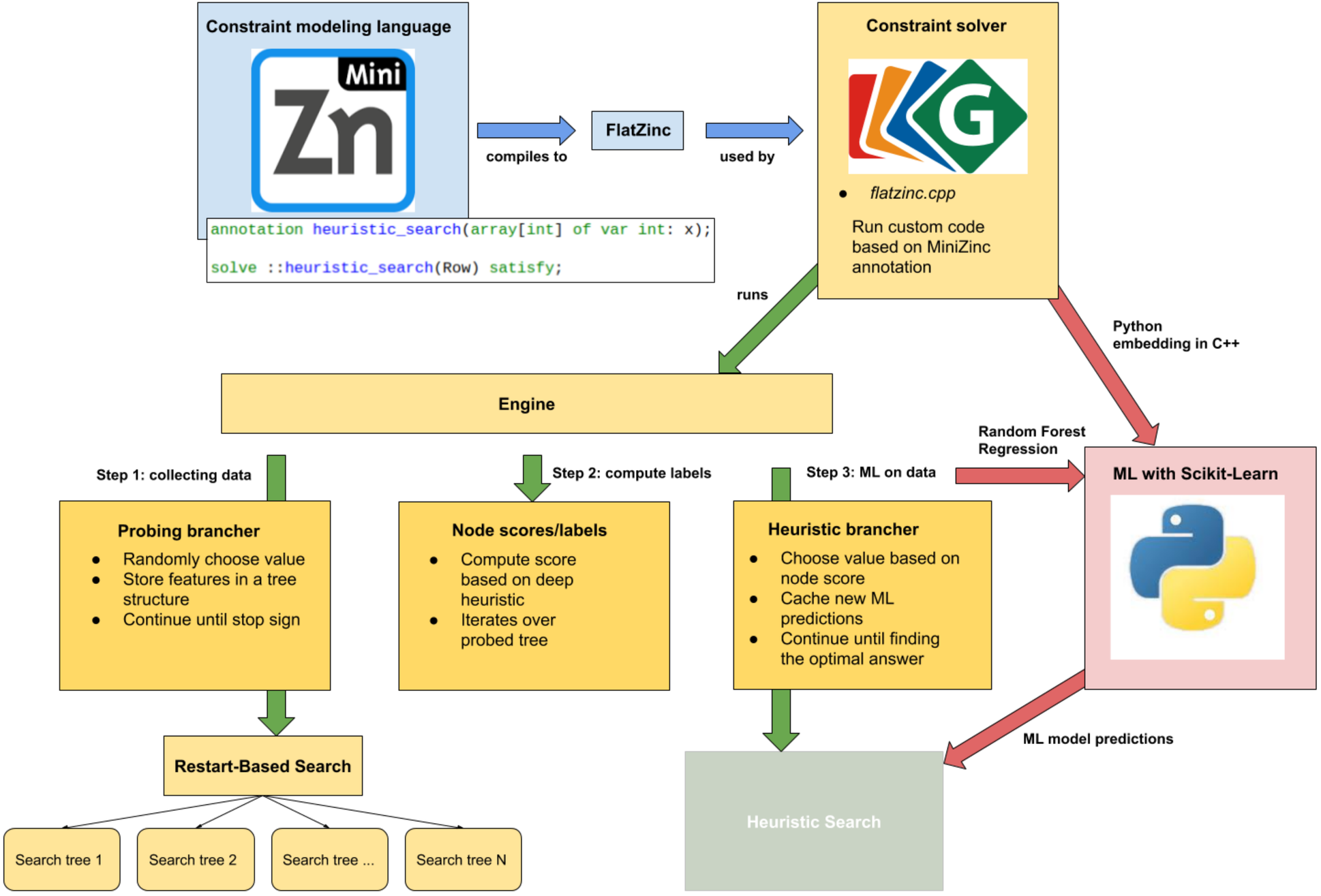


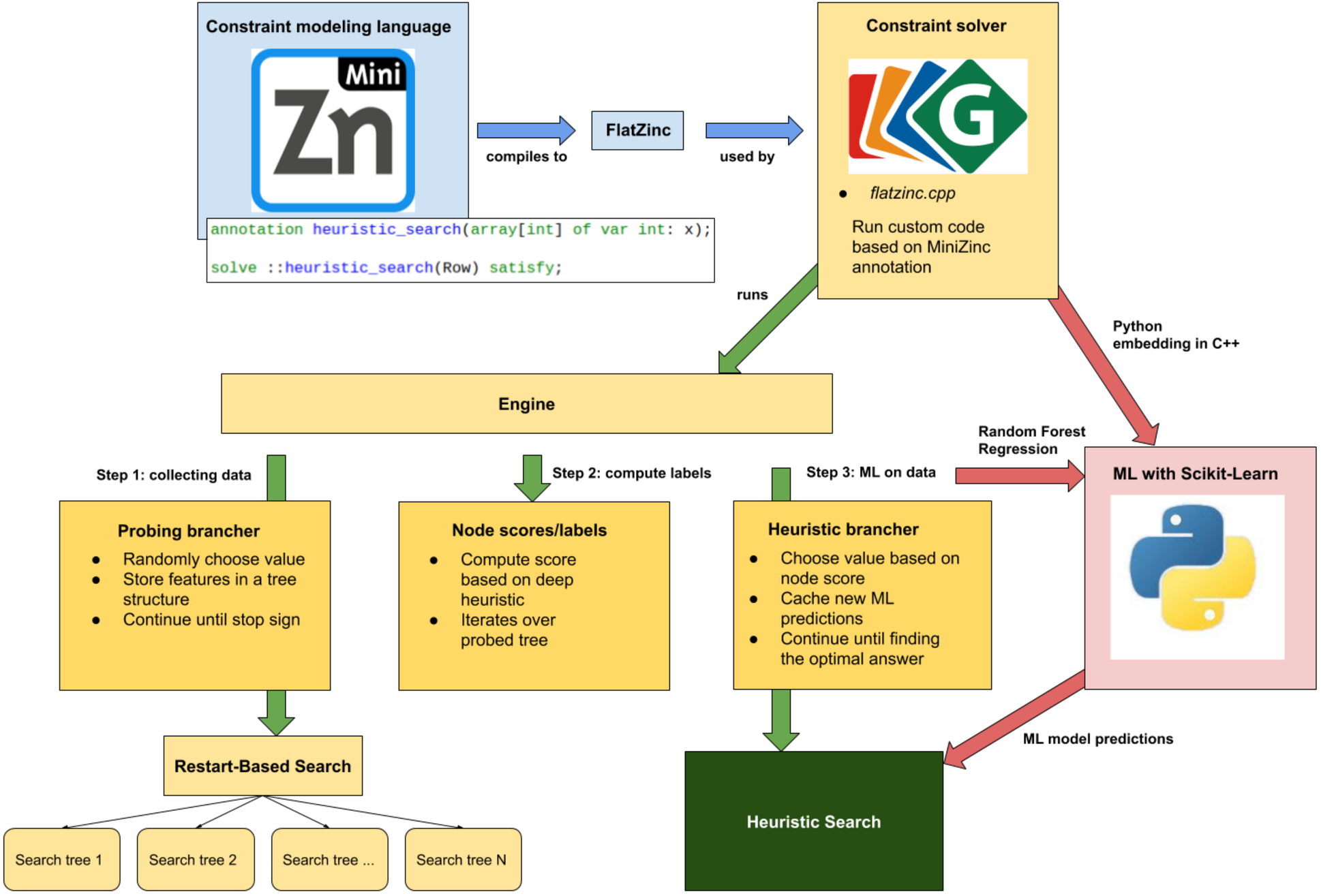


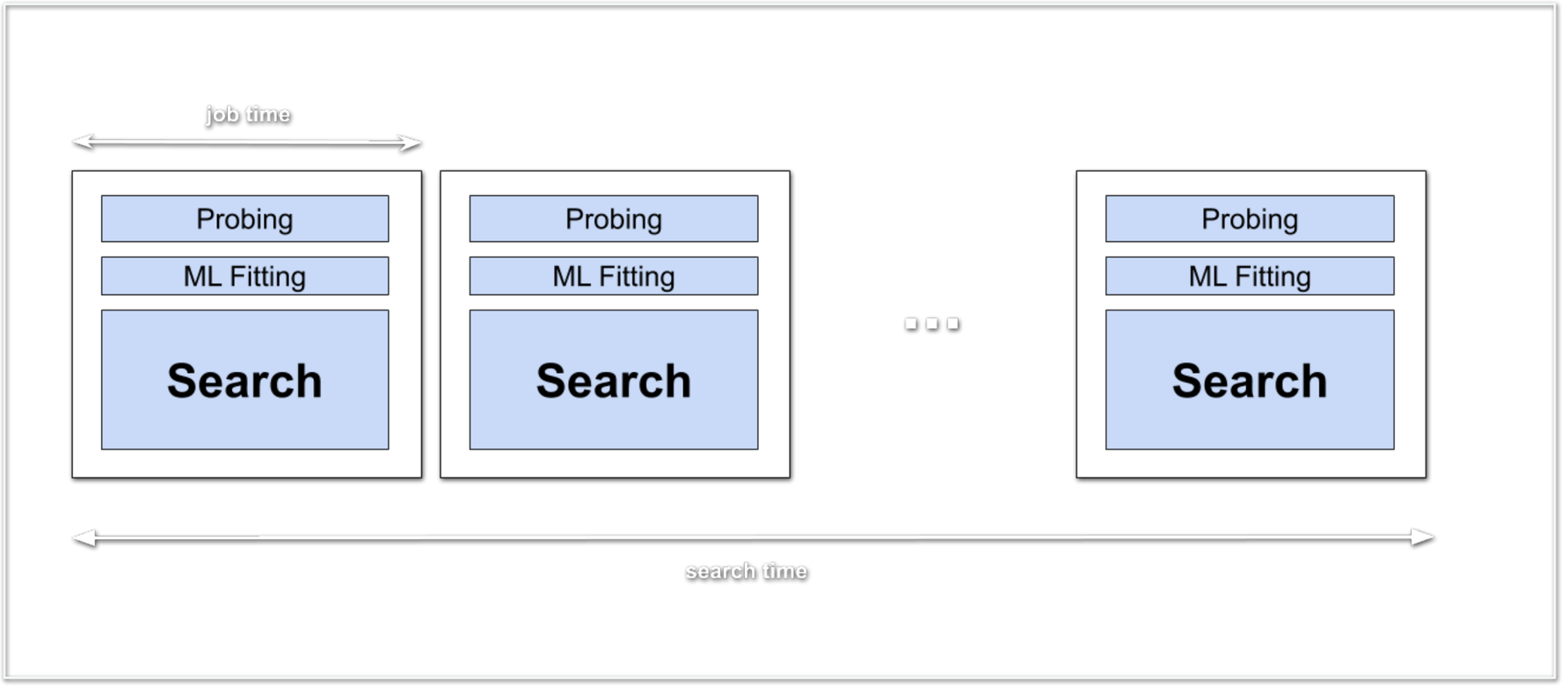


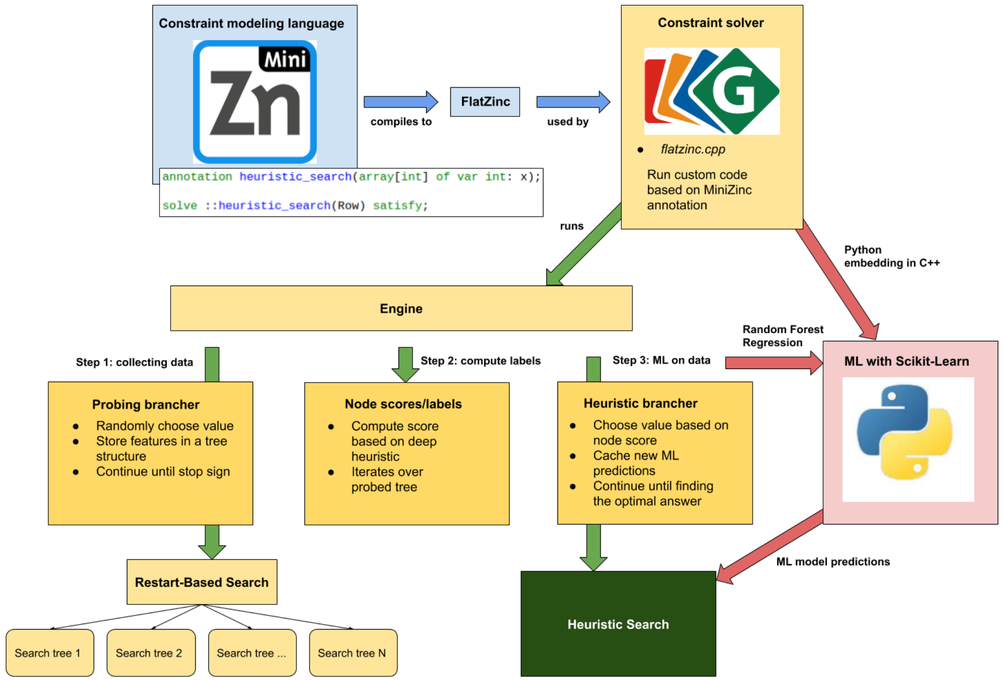












Three variable ordering heuristics

- smallest
- max regret
- anti first fail

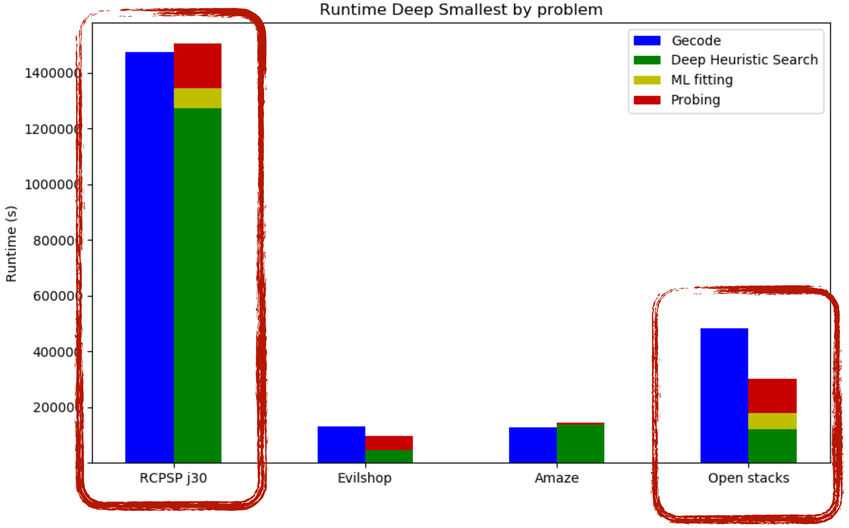
Four COPs from MiniZinc library

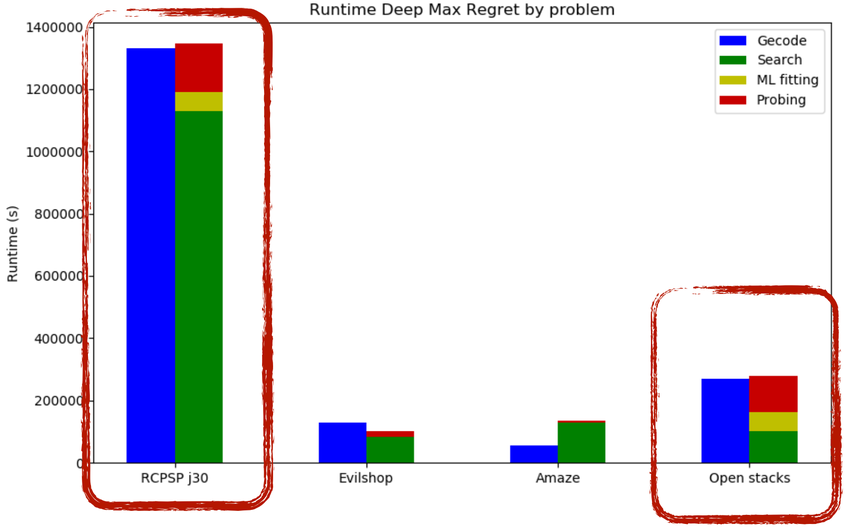
- RCPSP
- Amaze
- Evilshop
- Open Stacks

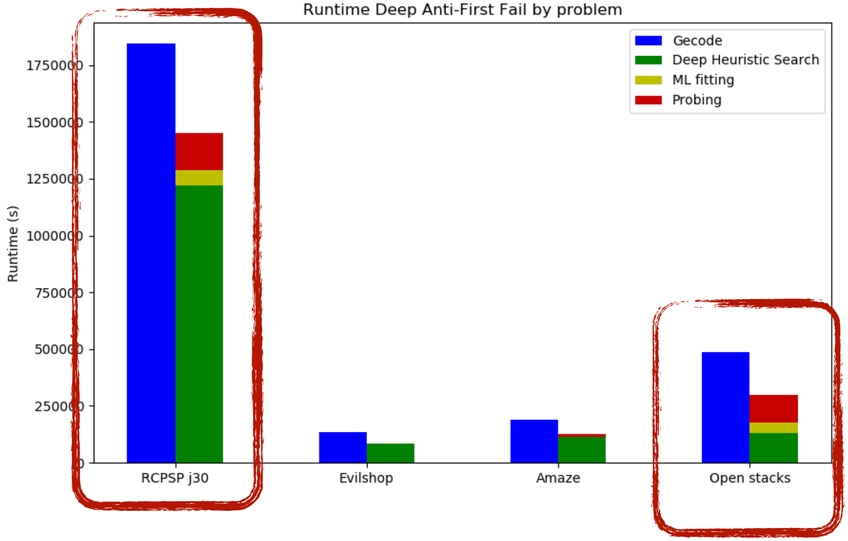
Experimental parameters

- Timeout: 4 hours
- Job time: 15 minutes
- Depth: d = 25

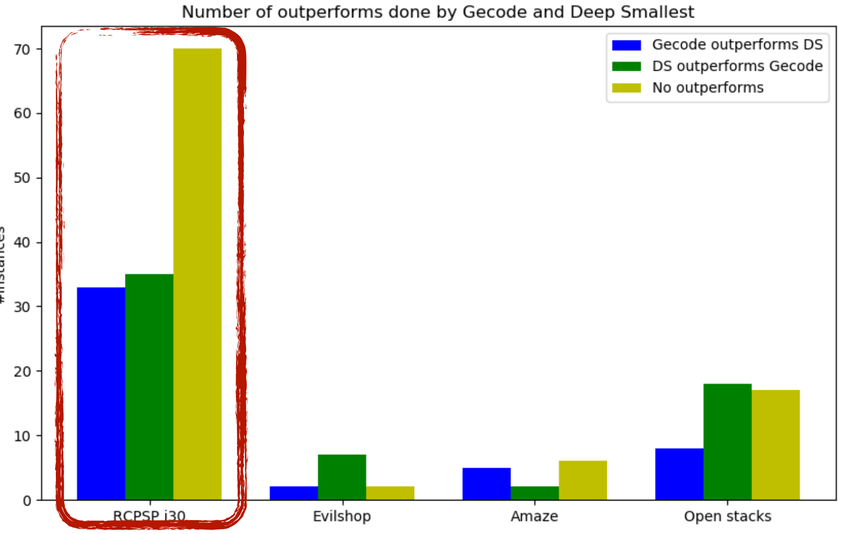
Results

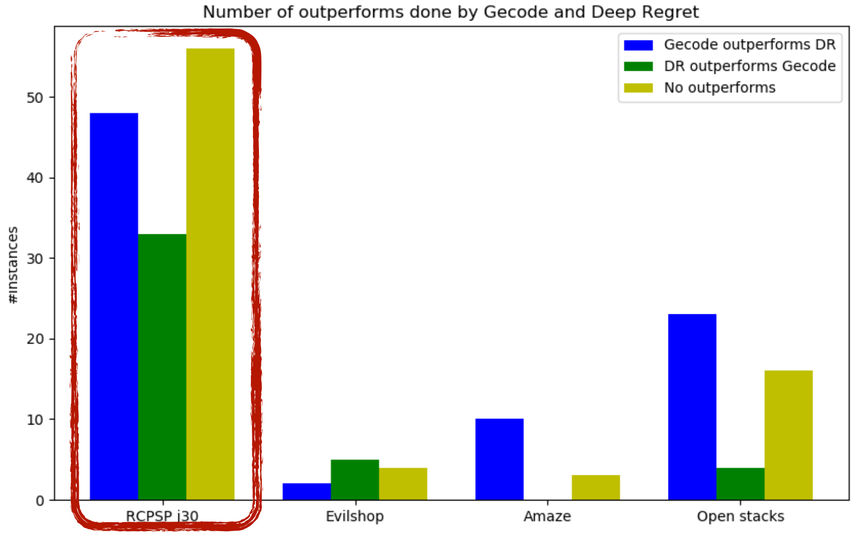


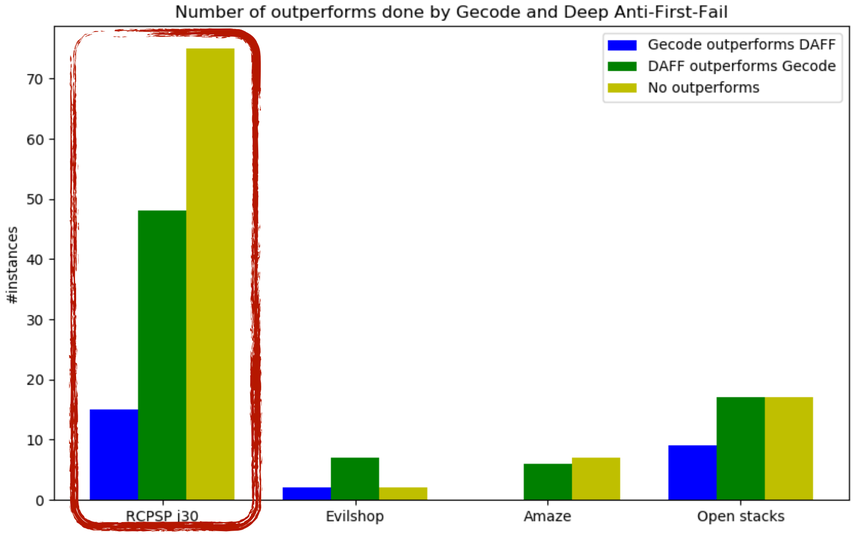




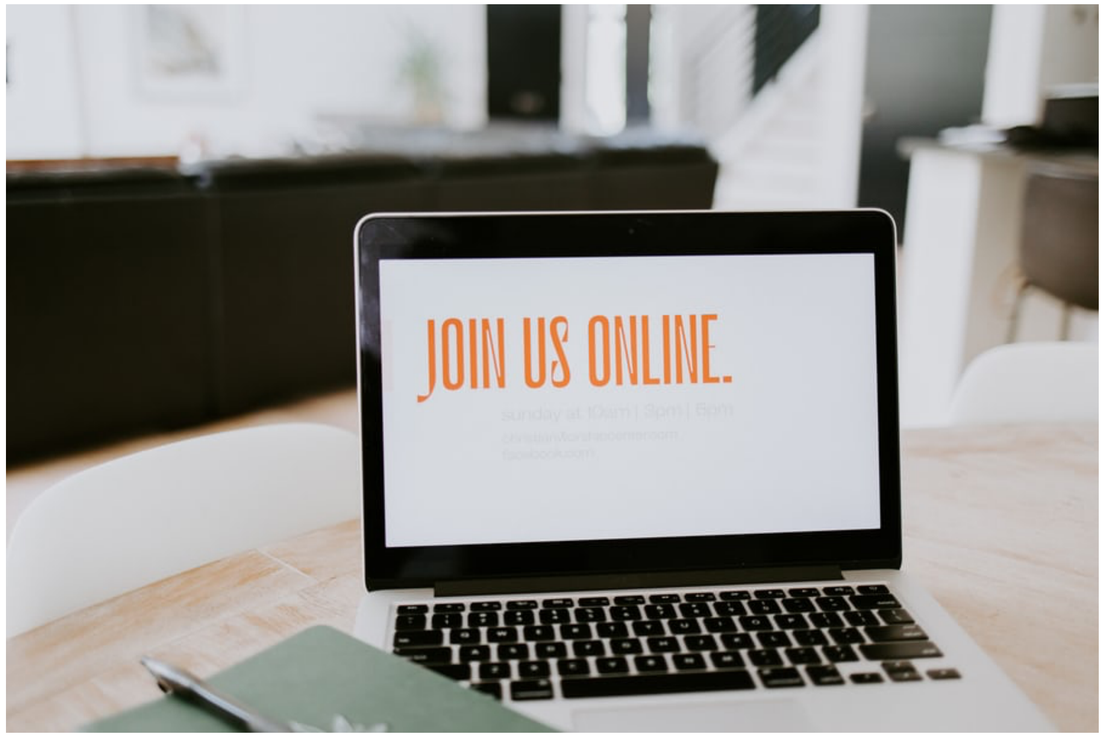
Results







Lessons



Samantha Borges

probe, approximate, restart-based search

promising results on 3 variable ordering heuristics

ablation study, variable+value ordering

1

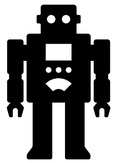
ML to configure CP-SAT solver

2

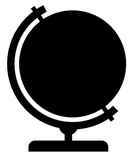
ML to learn CP heuristics on-the-fly

3

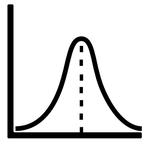
ML to embed forecasts within a CP model



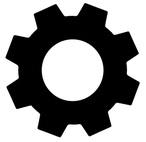
RL to learn MIP branching



learning surrogate simulation models



robust optimisation with learning



optimising GNNs using CO solvers

Machine Learning to Accelerate Solving of Constraint Programs

Neil Yorke-Smith
n.yorke-smith@tudelft.nl

TU Delft

STAR Lab