OJMO

Open Journal of Mathematical Optimization

Mark Turner, Thorsten Koch, Felipe Serrano & Michael Winkler

Adaptive Cut Selection in Mixed-Integer Linear Programming

Volume 4 (2023), article no. 5 (28 pages)

https://doi.org/10.5802/ojmo.25

Article submitted on February 24, 2022, revised on February 23, 2023,
accepted on June 29, 2023.

© The author(s), 2023.

This article is licensed under the
CREATIVE COMMONS ATTRIBUTION 4.0 INTERNATIONAL LICENSE.
http://creativecommons.org/licenses/by/4.0/

CENTRE MERSENNE

Open Journal of Mathematical Optimization is a member of the
Centre Mersenne for Open Scientific Publishing
http://www.centre-mersenne.org/
e-ISSN: 2777-5860

Adaptive Cut Selection in Mixed-Integer Linear Programming

Mark Turner
Chair of Software and Algorithms for Discrete Optimization, Institute of Mathematics, Technische Universität Berlin, Straße des 17. Juni 135, 10623 Berlin, Germany
Zuse Institute Berlin, Department of Mathematical Optimization, Takustr. 7, 14195 Berlin
turner@zib.de

Thorsten Koch
Chair of Software and Algorithms for Discrete Optimization, Institute of Mathematics, Technische Universität Berlin, Straße des 17. Juni 135, 10623 Berlin, Germany
Zuse Institute Berlin, Department of Mathematical Optimization, Takustr. 7, 14195 Berlin
koch@zib.de

Felipe Serrano
Cardinal Operations GmbH, Englerallee 19 14195 Berlin, Germany
Zuse Institute Berlin, Department of Mathematical Optimization, Takustr. 7, 14195 Berlin
serrano@zib.de

Michael Winkler
Gurobi GmbH, Ulmenstr. 37-39, 60325 Frankfurt am Main, Germany
Zuse Institute Berlin, Department of Mathematical Optimization, Takustr. 7, 14195 Berlin
winkler@gurobi.com

Abstract

Cutting plane selection is a subroutine used in all modern mixed-integer linear programming solvers with the goal of selecting a subset of generated cuts that induce optimal solver performance. These solvers have millions of parameter combinations, and so are excellent candidates for parameter tuning. Cut selection scoring rules are usually weighted sums of different measurements, where the weights are parameters. We present a parametric family of mixed-integer linear programs together with infinitely many family-wide valid cuts. Some of these cuts can induce integer optimal solutions directly after being applied, while others fail to do so even if an infinite amount are applied. We show for a specific cut selection rule, that any finite grid search of the parameter space will always miss all parameter values, which select integer optimal inducing cuts in an infinite amount of our problems. We propose a variation on the design of existing graph convolutional neural networks, adapting them to learn cut selection rule parameters. We present a reinforcement learning framework for selecting cuts, and train our design using said framework over MIPLIB 2017 and a neural network verification data set. Our framework and design show that adaptive cut selection does substantially improve performance over a diverse set of instances, but that finding a single function describing such a rule is difficult. Code for reproducing all experiments is available at `https://github.com/Opt-Mucca/Adaptive-Cutsel-MILP`.

Digital Object Identifier 10.5802/ojmo.25

2020 Mathematics Subject Classification 90C11.

Keywords Mixed-Integer Linear Programming, Cutting Plane Selection, Instance-Dependent Learning.

Acknowledgments We would like to thank the anonymous reviewers for their extensive feedback, which truly improved the quality of this work. The work for this article has been conducted in the Research Campus MODAL funded by the German Federal Ministry of Education and Research (BMBF) (fund numbers 05M14ZAM, 05M20ZBM). The described research activities are funded by the Federal Ministry for Economic Affairs and Energy within the project UNSEEN (ID: 03EI1004-C).

1 Introduction

A Mixed-Integer Linear Program (MILP) is an optimisation problem that is classically defined as:

$$\operatorname*{argmin}_{\mathbf{x}}\{\mathbf{c}^\intercal\mathbf{x} \mid \mathbf{A}\mathbf{x} \leq \mathbf{b},\ \ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u},\ \ \mathbf{x} \in \mathbb{Z}^{|\mathcal{J}|} \times \mathbb{R}^{n-|\mathcal{J}|}\} \tag{1}$$

© Mark Turner & Thorsten Koch & Felipe Serrano & Michael Winkler;
licensed under Creative Commons License Attribution 4.0 International

Here, $\mathbf{c} \in \mathbb{R}^n$ is the objective coefficient vector, $\mathbf{A} \in \mathbb{R}^{m \times n}$ is the constraint matrix, $\mathbf{b} \in \mathbb{R}^m$ is the right hand side constraint vector, $\mathbf{l}, \mathbf{u} \in \mathbb{R}^n \cup \{-\infty, \infty\}^n$ are the lower and upper variable bound vectors, and $\mathcal{J} \subseteq \{1, \dots, n\}$ is the set of indices of integer variables.

One of the main techniques for solving MILPs is the branch-and-cut algorithm, see [1] for an introduction. Generating cutting planes, abbreviated as *cuts*, is a major part of this algorithm, and is one of the most powerful techniques for quickly solving MILPs to optimality, see [3]. A cut is an inequality that does not remove any feasible solutions of (1) when added to the formulation. We restrict ourselves to linear cuts in this paper, and denote a cut as $\boldsymbol{\alpha} = (\alpha_0, \dots, \alpha_n) \in \mathbb{R}^{n+1}$, and denote the set of feasible solutions as $\mathcal{I}_{\mathcal{X}}$, to formally define a cut in (2).

$$\sum_{i=1}^{n} \alpha_i x_i \leq \alpha_0, \quad \forall\, x \in \mathcal{I}_{\mathcal{X}}, \quad \text{where } \mathbf{x} = (x_1, \dots, x_n) \tag{2}$$

The purpose of cuts is to tighten the linear programming (LP) relaxation of (1), where the LP relaxation is obtained by removing all integrality requirements. Commonly, cuts are found that separate the current feasible solution to the LP relaxation, referred to as $\mathbf{x}^{LP}$, from the tightened relaxation, and for this reason algorithms that find cuts are often called *separators*. This property is defined as follows:

$$\sum_{i=1}^{n} \alpha_i x_i^{LP} > \alpha_0, \quad \text{where } \mathbf{x}^{LP} = (x_1^{LP}, \dots, x_n^{LP}) \tag{3}$$

Within modern MILP solvers, the cut aspect of the branch-and-cut algorithm is divided into cut generation and cut selection subproblems. The goal of cut generation is finding cuts that both tighten the LP relaxation at the current node and improve overall solver performance. The cut selection subproblem is then concerned with deciding which of the generated cuts to add to the formulation (1). That is, given the set of generated cuts $\mathcal{S}' = \{\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_{|\mathcal{S}'|}\}$, find a subset $\mathcal{S} \subseteq \mathcal{S}'$ to add to the formulation (1).

We focus on the cut selection subproblem in this paper, where we motivate the need for instance-dependent cut selection rules as opposed to fixed rules, and introduce a reinforcement learning (RL) framework for learning parameters of such a rule. The cut selection subproblem is important, as adding either all or none of the generated cuts to the LP usually results in poor solver performance. This is due to the large computational burden of solving larger LPs at each node when all cuts are added, and the large increase in nodes needed to solve MILPs when no cuts are added. For a summary on MILPs we refer readers to [1], for cutting planes [23], for cut selection [30], and for reinforcement learning [28].

The rest of the paper is organised as follows. In Section 2, we summarise existing literature on learning cut selection. In Section 3, with an expanded proof in Appendix A, we motivate the need for adaptive cut selection by showing worst case performance of fixed cut selection rules. This section was inspired by [6], which proved complexity results for fixed branching rules. In Section 4 we summarise how cut selection is performed in the MILP solver SCIP [9]. In Section 5 we show how to formulate cut selection as a Markov decision process, and phrase cut selection as a reinforcement learning problem. This section was motivated by [15], which presented variable selection as a Markov decision process as well as experimental results of an imitation learning approach. Finally, in Section 6, we present a thorough computational experiment on learning cut selector parameters that improve root node performance, and study the generalisation of these parameters to the larger solving process. All experiments are done over MIPLIB 2017 [16] and a neural network verification data set [24] using the MILP solver SCIP version 8.0.1 [9].

2 Related Work

Several authors have proposed cut selection rules and performed several computational studies. The thesis [1] presents a linear weighted sum cut selection rule, which drastically reduces solution time to optimality by selecting a reduced number of good cuts. This cut selection rule and algorithm, see [9], can still be considered the basis of what we use in this paper. A more in-depth guide to cutting plane management is given in [30]. Here, a large variety of cut measures are summarised and additional computational results given that show how a reduced subset of good cuts can drastically improve solution time. A further computational study, focusing on cut selection strategies for zero-half cuts, is presented in [4]. They hypothesise that generating a large amount of cuts followed by heuristic selection strategy is more effective than generating a few deep cuts. Note that the solver and cut selection algorithms used in [1], [4], and [30] are different. More recently, [11] summarises the

current state of separators and cut selection in the literature, and poses questions aimed to better develop the science of cut selection. The final remark of the paper ponders whether machine learning can be used to answer some of the posed questions.

Recently, the intersection of mixed-integer programming and machine learning has received a lot of attention, specifically when it comes to branching, see [6, 15, 24] for examples. To the best of our knowledge, however, there are currently only four publications on the intersection of cut selection and machine learning. Firstly, [7] shows how cut selection parameter spaces can be partitioned into regions, such that the highest ranking cut is invariant to parameter changes within the regions. These results are extended to the class of Chvàtal-Gomory cuts applied at the root, with a sample complexity guarantee of learning cut selection parameters w.r.t. the resultant branch and bound tree size. Secondly, [29] presents a reinforcement learning approach using evolutionary strategies for ranking Gomory cuts via neural networks. They show that their method outperforms standard measures, e.g. max violation, and generalises to larger problem sizes within the same class. Thirdly, [8] train a neural network to rank linear cuts by expected objective value improvement when applied to a semi-definite relaxation. Their experiments show that substantial computational time can be saved when using their approximation, and that the gap after each cut selection round is very similar to that found when using the true objective value improvement. Most recently, [19] proposes a multiple instance learning approach for cut selection. They learn a scoring function parameterised as a neural network, which takes as input an aggregated feature vector over a bag of cuts. Their features are mostly composed of measures normally used to score cuts, e.g. norm violation. Cross entropy loss is used to train their network by labelling the bags of cuts before training starts.

Our contribution to the literature is three-fold. First, we provide motivation for instance-dependent cut selection by proving the existence of a family of parametric MILPs together with an infinite amount of family-wide valid cuts. Some of these cuts can induce integer optimal solutions directly after being applied, while others fail to do so even if an infinite amount are applied. Using a basic cut selection strategy and a pure cutting plane approach, we show that any finite grid search of the cut selector's parameter space, will miss all parameter values, which select integer optimal inducing cuts in an infinite amount of our instances. An interactive version of this constructive proof is provided in Mathematica® [31], and instance creation algorithms are provided using SCIP's Python API [9, 22]. Second, we introduce a RL framework for learning instance-dependent cut selection rules, and present results on learning parameters to SCIP's default cut selection rule [1] over MIPLIB 2017 [16] and a neural network verification data set [24]. Third and finally, we implemented a new cut selector plugin, which is available from SCIP 8.0 [9], and enables users to include their own cut selection algorithms in the larger MILP solving process.

3 Motivating Adaptive Cut Selection

This section introduces a simplified cut scoring rule, and discusses how the parameters for such a rule are traditionally set in solvers. A theorem is then introduced that motivates the need for adaptive cut scoring rules, and is proven in Appendix A using a simulated pure cutting plane approach.

Consider the following simplified version of SCIP's default cut scoring rule (see Section 4 for the default scoring rule):

$$\texttt{simple\_cut\_score}(\lambda, \boldsymbol{\alpha}, \mathbf{c}) := \lambda * \texttt{isp}(\boldsymbol{\alpha}) + (1-\lambda) * \texttt{obp}(\boldsymbol{\alpha}, \mathbf{c}), \quad \lambda \in [0,1], \boldsymbol{\alpha} \in \mathbb{R}^{n+1}, \mathbf{c} \in \mathbb{R}^n \tag{4}$$

Using the general MILP definition given in (1), we define the cut measures integer support (`isp`) and objective parallelism (`obp`) as follows:

$$\texttt{isp}(\boldsymbol{\alpha}) := \frac{\sum_{i \in \mathcal{J}} \texttt{nonzero}(\alpha_i)}{\sum_{i=1}^{n} \texttt{nonzero}(\alpha_i)}, \quad \text{where} \quad \texttt{nonzero}(\alpha_i) = \begin{cases} 0 & \text{if } \alpha_i = 0 \\ 1 & \text{otherwise} \end{cases} \tag{5}$$

$$\texttt{obp}(\boldsymbol{\alpha}, \mathbf{c}) := \left| \frac{\sum_{i=1}^{n} \alpha_i c_i}{\sqrt{\sum_{i=1}^{n} \alpha_i^2} \sqrt{\sum_{i=1}^{n} c_i^2}} \right| \tag{6}$$

We now introduce Theorem 1, which refers to the λ parameter in (4).

Theorem 1. *Given a finite discretisation of λ, an infinite family of MILP instances together with an infinite amount of family-wide valid cuts can be constructed. Using a pure cutting plane approach and applying a single cut per selection round, the infinite family of instances do not solve to optimality for any value in the discretisation, but do solve to optimality for an infinite amount of alternative λ values.*

The general purpose of Theorem 1 is to motivate the need for instance-dependent parameters in the cut selection subroutine. The typical approach for finding the best choice of cut selector parameters, see previous SCIP computational studies [1, 9, 14], is to perform a parameter sweep, most often a grid search. A grid search, however, leaves regions unexplored in the parameter space. In our simplified cut scoring rule (4), we have a single parameter, namely λ, and these unexplored regions are simply intervals. We define Λ, the set of values in the finite grid search of λ, as follows:

$$\Lambda := \{\lambda_1, \dots, \lambda_{|\Lambda|}\}, \text{ where } 0 \leq \lambda_i < \lambda_{i+1} \leq 1 \quad \forall\, i \in \{1, \dots, n-1\}, \quad |\Lambda| \in \mathbb{N}$$

The set of unexplored intervals in the parameter space, denoted $\widetilde{\Lambda}$, is then defined as:

$$\widetilde{\Lambda} := \{[0, \lambda_1) \cup (\lambda_1, \lambda_2) \cup \dots \cup (\lambda_{n-1}, \lambda_n) \cup (\lambda_{|\Lambda|}, 1]\}$$

Our goal is to show that for any Λ we can construct an infinite family of MILP instances from Theorem 1. Together with our infinite amount of family-wide valid cuts and specific cut selection rule, we will show that the solving process does not finitely terminate for any choice of λ outside of an interval $(\lambda_{lb}, \lambda_{ub}) \subset \widetilde{\Lambda}$. In effect, this shows that using the same fixed λ value over all problems in a MILP solver could result in incredibly poor performance for many problems. This is somewhat expected, as a fixed parameter cannot be expected to perform well on all possible instances, and moreover, cut selection is only a small subroutine in the much larger MILP solving process. Additionally, the instance space of MILPs is non-uniform, and good performance over certain problems may be highly desirable as they occur more frequently in practice. Nevertheless, Theorem 1 provides important motivation for adaptive cut selection. See Appendix A for a complete proof.

4 Cut Selection in SCIP

Until now we have motivated adaptive cut selection in a theoretical manner, by simulating poor performance of fixed cut selector rules in a pure cutting approach. Using this motivation, we now present results of how parameters of a cut selection scoring rule can be learnt, and made to adapt with the input instance. We begin with an introduction to cut selection in SCIP [9].

The official SCIP cut scoring rule (7) that has been used since SCIP 6.0 is defined as:

$$\begin{aligned} \texttt{cut\_score}(\boldsymbol{\lambda}, \boldsymbol{\alpha}, \mathbf{c}, \mathbf{x}^{LP}, \widehat{\mathbf{x}}) := \lambda_1 * \texttt{dcd}(\boldsymbol{\alpha}, \mathbf{x}^{LP}, \widehat{\mathbf{x}}) + \lambda_2 * \texttt{eff}(\boldsymbol{\alpha}, \mathbf{x}^{LP}) + \lambda_3 * \texttt{isp}(\boldsymbol{\alpha}) + \lambda_4 * \texttt{obp}(\boldsymbol{\alpha}, \mathbf{c}) \\ \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 = 1, \quad \lambda_i \geq 0 \quad \forall\, i \in \{1, 2, 3, 4\}, \quad \boldsymbol{\lambda} = [\lambda_1, \lambda_2, \lambda_3, \lambda_4] \end{aligned} \tag{7}$$

The measures integer support (`isp`) and objective parallelism (`obp`) are defined in (5) and (6). Using the general MILP definition (1), letting $\mathbf{x}^{LP}$ be the LP optimal solution of the current relaxation, and $\widehat{\mathbf{x}}$ be the current best incumbent solution, we define the cut measures directed cutoff distance (`dcd`) and efficacy (`eff`) as follows:

$$\texttt{dcd}(\boldsymbol{\alpha}, \mathbf{x}^{LP}, \widehat{\mathbf{x}}) := \frac{\sum_{i=1}^{n} \alpha_i x_i^{LP} - \alpha_0}{|\sum_{i=1}^{n} \alpha_i y_i|}, \text{ where } \quad y = \frac{\widehat{\mathbf{x}} - \mathbf{x}^{LP}}{\|\widehat{\mathbf{x}} - \mathbf{x}^{LP}\|} \tag{8}$$

$$\texttt{eff}(\boldsymbol{\alpha}, \mathbf{x}^{LP}) := \frac{\sum_{i=1}^{n} \alpha_i x_i^{LP} - \alpha_0}{\sqrt{\alpha_1^2 + \dots + \alpha_n^2}} \tag{9}$$

We note that in SCIP the cut selector does not control how many times it itself is called, which candidate cuts are provided, nor the maximum amount of cuts we can apply each round. We reiterate that each call to the selection subroutine is called an *iteration* or *round*. Algorithm 1 gives an outline of the SCIP cut selection rule.

The SCIP cut selector rule in Algorithm 1 still follows the major principles presented in [1]. Cuts are greedily added by the largest score according to the scoring rule (7). After a cut is added, all other candidate cuts that are deemed too parallel to the added cut are filtered out and can no longer be added to the formulation this round. Forced cuts, which are always added to the formulation, prefilter all candidate cuts for parallelism, and are most commonly one-dimensional cuts or user defined cuts. We note that Algorithm 1 is a summarised version of the true algorithm, and has abstracted some procedures. Certain parameters have also been removed for the sake simplicity, such as those which determine when two cuts are too parallel. We further note that $\boldsymbol{\lambda} = \{0.0, 1.0, 0.1, 0.1\}$ as of SCIP 8.0.

Motivated by work from this paper, users can now define their own cut selection algorithms and include them in SCIP for all versions since SCIP 8.0 [9]. Users can do this with a single function interface, bypassing

Algorithm 1: SCIP Default Cut Selector (Summarised)

Input : $cuts \in \mathbb{R}^{s_1 \times n}$, $forced\_cuts \in \mathbb{R}^{s_2 \times n}$, $max\_cuts \in \mathbb{Z}_{\geq 0}$, $(s_1, s_2) \in \mathbb{Z}^2_{\geq 0}$
Return : Sorted array of selected cuts, the amount of cuts selected

$n\_cuts \leftarrow s_1$ `// Size of cuts array`
for forced_cut *in* forced_cuts **do**
  $cuts$, $n\_cuts \leftarrow$ remove cuts from $cuts$ too parallel to $forced\_cut$
end
$n\_selected\_cuts \leftarrow 0$
$selected\_cuts \leftarrow \emptyset$
while n_cuts > *0 and* max_cuts > n_selected_cuts **do**
  `// Scoring done with (7). If no primal, efficacy replaces cutoff distance`
  $best\_cut \leftarrow$ select highest scoring cut remaining in $cuts$
  $selected\_cuts \leftarrow selected\_cuts \cup best\_cut$
  $n\_selected\_cuts \leftarrow n\_selected\_cuts + 1$
  $cuts$, $n\_cuts \leftarrow$ remove cuts from $cuts$ too parallel to $best\_cut$
end
return forced_cuts ∪ selected_cuts, s_2 + n_selected_cuts

the previous need to modify SCIP source code. For example, users can introduce a cut selection rule with an entirely new scoring rule that replaces (7), or introduce a new filtering mechanism that is not based exclusively on parallelism. We hope that this leads to additional research about cut selection algorithms in modern MILP solvers.

5 Problem Representation and Solution Architecture

We now present our approach for learning cut selector parameters for MILPs. In Subsection 5.1 we describe our encoding of a general MILP instance into a bipartite graph. Subsection 5.2 introduces a framework for posing cut selection parameter choices as a RL problem, with Subsection 5.3 describing the graph convolutional neural network architecture used as our policy network. Subsection 5.4 outlines the training method to update our policy network.

5.1 Problem representation as a graph

The current standard for deep learning representation of a general MILP instance is the constraint-variable bipartite graph as described in [15]. Some extensions to this design have been proposed, see [12], as well as alternative non graph embeddings, see [16] and [27]. We use the embedding as introduced in [15] and the accompanying graph convolutional neural network (GCNN) design, albeit with the removal of all LP solution specific features and a different interpretation of the output. The construction process for the bipartite graph can be seen in Figure 1.

The bipartite graph representation can be written as $G = \{\mathbf{V}, \mathbf{C}, \mathbf{E}\} \in \mathcal{G}$, where $\mathcal{G}$ is the set of all bipartite graph representations of MILP instances. $\mathbf{V} \in \mathbb{R}^{n \times 7}$ is the feature matrix of nodes on one side of the graph, which correspond one-to-one with the variables (columns) in the MILP. $\mathbf{C} \in \mathbb{R}^{m \times 7}$ is the feature matrix of nodes on the other side, and correspond one-to-one with the constraints (rows) in the MILP. An edge $(i, j) \in \mathbf{E}$ exists when the variable represented by x_i has non-zero coefficient in constraint cons_j, where $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$. We abuse notation slightly and say that $\mathbf{E} \in \mathbb{R}^{m \times n \times 1}$, where $\mathbf{E}$ is the edge feature tensor. Note that we do not extend our MILP representation after every round of cuts is added due to using single-step learning, see Subsection 5.2. The representation is extendable however to multi-step learning, where the added cuts could become constraints. The exact set of features can be seen in Table 1.

5.2 Reinforcement Learning Framework

We formulate our problem as a single step Markov decision process. The initial state of our environment is $s_0 = G^0 = G$. An agent takes an action $a_0 \in \mathbb{R}^4$, resulting in an instant reward $\mathtt{r}(s_0, a_0) \in \mathbb{R}$, and deterministically

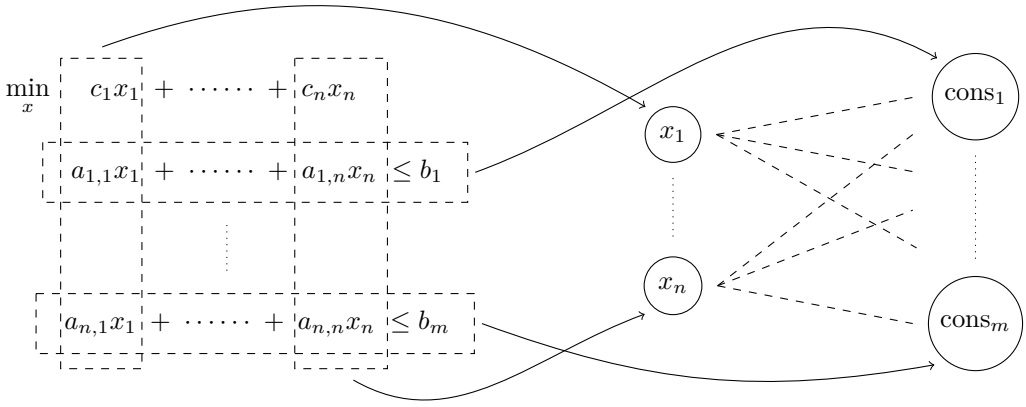

Figure 1 A visualisation of the variable-constraint bipartite graph construction from a MILP.

transitions to a terminal state $s_1 = G^{N_r}$, $N_r \in \mathbb{Z}$. The action taken, a_0, is dictated by a policy $\pi_\theta(a_0|s_0)$ that maps any initial state to a distribution over our action space, i.e. $a_0 \sim \pi_\theta(\,\cdot\,|s_0)$.

The MILP solver in this framework is our environment, and the cut selector our agent. Let N_r be the number of paired separation and cut selection rounds we wish to apply, and $G^i \in \mathcal{G}$ be the bipartite graph representation of $G \in \mathcal{G}$ after i rounds have been applied. The action $a_0 \in \mathbb{R}^4$ is the choice of cut selector parameters $\{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}$ followed by N_r paired separation rounds. Applying action a_0 to state s_0 results in a deterministic transition to $s_1 = G^{N_r}$, defined by the function $\mathtt{f} : \mathcal{G} \times \mathbb{R}^4 \to \mathcal{G}$.

The baseline function, $\mathtt{b}(s_0) : \mathcal{G} \to \mathbb{R}$, maps an initial state s_0 to the primal-dual difference of the LP solution of $\mathtt{f}(s_0, a') \in \mathcal{G}$, where the solver is run with standard cut selector parameters, $a' \in \mathbb{R}^4$, and some pre-loaded primal solution. The primal-dual difference in this experiment can be thought of as a strict dual bound improvement, as the pre-loaded primal cannot be improved upon without a provable optimal solution itself. The pre-loaded primal also serves to make directed cutoff distance active from the beginning of the solving process. We do note that this is different to the normal solve process and introduces some bias, most notably for directed cutoff distance. Let $\mathtt{g}_{a_0}(s_0)$ be the primal-dual difference of the LP solution of $\mathtt{f}(s_0, a_0)$ if a_0 are the cut selector parameter values used. The reward $\mathtt{r}(s_0, a_0)$ can then be defined as:

$$\mathtt{r}(s_0, a_0) := \frac{\mathtt{b}(s_0) - \mathtt{g}_{a_0}(s_0)}{|\mathtt{b}(s_0)| + 10^{-8}}$$

Let $(s_0, a_0, s_1) \in \mathcal{G} \times \mathbb{R}^4 \times \mathcal{G}$ be a trajectory, also called a roll out in the literature. The goal of reinforcement learning is to maximise the expected reward over all trajectories. That is, we want to find θ that parameterises:

$$\underset{\theta}{\operatorname{argmax}} \mathop{\mathbb{E}}_{(s_0,a_0,s_1)\sim\pi_\theta} [\mathtt{r}(s_0, a_0, s_1)] \;=\; \underset{\theta}{\operatorname{argmax}} \int_{s_1 \in \mathcal{G}} \int_{(s_0,a_0)\in \mathtt{f}^{-1}(s_1)} p(s_0)\pi_\theta(a_0|s_0)\mathtt{r}(s_0, a_0)\, \mathrm{d}s_0 \mathrm{d}a_0 \mathrm{d}s_1 \tag{10}$$

Here, $p(s_0)$ is the density function on instances $s \in \mathcal{G}$ evaluated at $s = s_0$. The pre-image $\mathtt{f}^{-1}(s_1) : \mathcal{G} \to \mathbb{R}^4 \times \mathcal{G}$ is defined as:

$$\mathtt{f}^{-1}(\mathcal{G}') := \{(s_0, a_0) \in \mathcal{G} \times \mathbb{R}^4 \mid \mathtt{f}(s_0, a_0) \in \mathcal{G}'\}, \quad \mathcal{G}' \subseteq \mathcal{G}$$

Table 1 Feature descriptions of variable (column) feature matrix $\mathbf{V}$, constraint (row) feature matrix $\mathbf{C}$, and edge feature tensor $\mathbf{E}$.

Tensor	Features	Value Range
$\mathbf{V}$	Normalised objective coefficient	$[-1, 1]$
	Normalised lower bound | upper bound	$\{-2, [-1, 1], 2\}^2$
	Type: binary | integer | continuous | implicit integer	one-hot encoding
$\mathbf{C}$	Absolute objective parallelism (cosine similarity)	[0,1]
	Normalised RHS per constraint	$[-1, 1]$
	Type: linear | logicor | knapsack | setppc | varbound	one-hot encoding
$\mathbf{E}$	Normalised coefficients per constraint	$[-1, 1]$

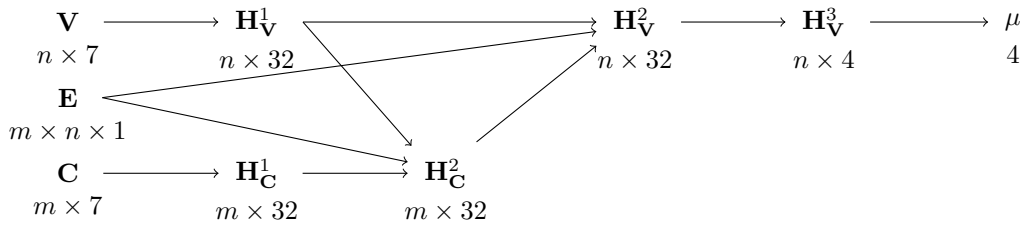

Figure 2 The architecture of policy network $\pi_\theta(a_0|s_0)$. $\mathbf{H}$ represent hidden layers of the network.

We note that equation (10) varies from the standard definition as seen in [28], and those presented in similar research [15, 29], as our action space is continuous. Additionally, as the set $\mathcal{G}$ is infinite and we do not know the density function $p(s)$, we use sample average approximation, creating a uniform distribution around our input data set.

5.3 Policy Architecture

Our policy network, $\pi_\theta(\cdot\,|s_0 \in \mathcal{G})$, is parameterised as a graph convolutional neural network, and follows the general design as in [15], where θ fully describes the complete set of weights and biases in the GCNN. The changes in design are that we use 32 dimensional convolutions instead of 64 due to our lower dimensional input, and output a 4 dimensional vector as we are interested in cut selector parameters. This technique of using the constraint-variable graph as an embedding for graph neural networks has gained recent popularity, see [10] for an overview of applications in combinatorial optimisation.

Our policy network takes as input the constraint-variable bipartite graph representation $s_0 = \{\mathbf{V}, \mathbf{C}, \mathbf{E}\}$. Two staggered half-convolutions are then applied, with messages being passed from the embedding $\mathbf{V}$ to $\mathbf{C}$ and then back. The result is a bipartite graph with the same topology but new feature matrices. Our policy is then obtained by normalising feature values over all variable nodes and averaging the result into a vector $\mu \in \mathbb{R}^4$. This vector $\mu \in \mathbb{R}^4$ represents the mean of a multivariate normal distribution, $\mathcal{N}_4(\mu, \gamma I)$, where $\gamma \in \mathbb{R}$. We note that having the GCNN only output the mean was a design choice to simplify the learning process, and that our design can be extended to also output γ or additional distribution information. Any sample from the distribution $\mathcal{N}_4(\mu, \gamma I)$ can be considered an action $a_0 \in \mathbb{R}^4$, which represents $\{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}$ with the non-negativity constraints relaxed. Figure 2 provides an overview of this architecture. For a walk-through of the GCNN, see Appendix C.

5.4 Training Method

To train our GCNN we use policy gradient methods, specifically the REINFORCE algorithm with baseline and gaussian exploration, see [28] for an overview. An outline of the algorithm is given in Algorithm 2.

Algorithm 2: Batch REINFORCE

Input : Policy network π_θ, MILP instances *batch*, $n_{\text{samples}} \in \mathbb{N}$, $N_r \in \mathbb{N}$

$\mathcal{L} \leftarrow 0$
for s_0 *in batch* **do**
  $\mu \leftarrow \pi_\theta(\cdot\,|s_0)$ `// Note that` $\pi_\theta(\cdot\,|s_0)$ `is technically` $\mathcal{N}_4(\mu, \gamma I)$
  for i *in* $\{1, \ldots, n_{samples}\}$ **do**
    $a_0 \leftarrow$ sample $\mathcal{N}_4(\mu, \gamma I)$
    $s_1 \leftarrow$ Apply N_r rounds of separation and cut selection to s_0
    $r \leftarrow$ Relative dual bound improvement of s_1 to some baseline
    $\mathcal{L} \leftarrow \mathcal{L} + (-r \times \log(\pi_\theta(a_0|s_0)))$ `// Use log probability for numeric stability`
  end
end
$\theta \leftarrow \theta + \nabla_\theta \mathcal{L}$ `// We use the Adam update rule in practice [20]`

Algorithm 2 is used to update the weights and biases, θ, of our GCNN, $\pi_\theta(\cdot\,|s_0 \in \mathcal{G})$. It does this for a batch of instances by minimising $\mathcal{L}$, referred to as the loss function, see [17]. We used default parameter settings in the

Adam update rule, aside from a learning rate with value 5×10^{-4}. Our training approach is performed offline, and only the final GCNN is used for evaluation.

6 Experiments

We use MIPLIB 2017[1] [16] as our first data set, which we simply refer to it as MIPLIB, and a set of neural network verification instances[2] [24] as our second data set, which we refer to as NN-Verification. For all subsections we run experiments on instances that have gone through SCIP's default presolve, see [2] for an overview on presolve techniques. Each individual run on a presolved instance consists of a single round of presolve (to remove fixed variables), then solving the root node, using 50 separation rounds with a limit of 10 cuts per round. Propagation, heuristics, and restarts are disabled for the runs, with a slightly modified version of SCIP's cut selector in Algorithm 1 being used, where $\boldsymbol{\lambda}$ is defined by the user for each run. A pre-loaded MIP start is also provided, which is the best solution found within 600s when solved with default settings. In the case of less than 10 cuts being selected due to parallelism filtering, the highest filtered scoring cuts are added until the 10 cut per round limit is reached, or no more cuts exist. We believe these conditions best represent a sandbox environment that allows cut selection to be the largest influence on solver performance. Additionally, all results are obtained by averaging results over the SCIP random seeds $\{1, 2, 3\}$. All code for reproducing experiments can be found at `https://github.com/Opt-Mucca/Adaptive-Cutsel-MILP`.

Table 2 Percentage of instances removed from MIPLIB and NN-Verification data sets.

Criteria	% MIPLIB	% NN-Verification
Tags: *feasibility*, *numerics*, *infeasible*, *no solution*	4.5%, 17.5%, 2.8%, 0.9%	-
Unbounded objective, MIPLIB solution unavailable	0.9%, 2.6%	-
Presolve longer than 300s under default conditions	3.6%	0%
No feasible solution found in 600s under default conditions	10.9%	0.2%
Solved to optimality at root	13.7%	0%
Root solve longer than 20s	22.5%	7.1%
Too few cuts applied (< 250)	7.5%	26.4%
Primal-dual difference < 0.5	0.9%	40.5%
LP errors	0.5%	0.1%

The modification of SCIP's default cut selector for our experiment is done to standardise the range of the individual cut measures, simplifying the learning process of those measure's coefficients. The measures `isp` and `obp` for any cut are in the range $[0, 1]$, while the measures `eff` and `dcd`, following the assumption that $\mathbf{x}^{LP}$ is separated, are in the range $[0, \infty)$. We therefore substitute `eff` and `dcd` in the default SCIP cut scoring rule by the following normalised measures `eff'` and `dcd'`:

$$\texttt{eff}'(\boldsymbol{\alpha}, \mathbf{x}^{LP}, \mathcal{S}') := \left(\frac{\texttt{log}(\texttt{eff}(\boldsymbol{\alpha}, \mathbf{x}^{LP}) + 1)}{\texttt{log}(\texttt{max}_{\boldsymbol{\alpha}' \in \mathcal{S}'}\{\texttt{eff}(\boldsymbol{\alpha}', \mathbf{x}^{LP})\} + 1)} \right)^2 \tag{11}$$

$$\texttt{dcd}'(\boldsymbol{\alpha}, \mathbf{x}^{LP}, \widehat{\mathbf{x}}, \mathcal{S}') := \left(\frac{\texttt{log}(\texttt{dcd}(\boldsymbol{\alpha}, \mathbf{x}^{LP}, \widehat{\mathbf{x}}) + 1)}{\texttt{log}(\texttt{max}_{\boldsymbol{\alpha}' \in \mathcal{S}'}\{\texttt{dcd}(\boldsymbol{\alpha}', \mathbf{x}^{LP}, \widehat{\mathbf{x}})\} + 1)} \right)^2 \tag{12}$$

For all experiments SCIP 8.0.1 [9] is used, with PySCIPOpt [22] as the API, and Gurobi 9.5.1 [18] as the LP solver. PyTorch 1.7.0 [25] and PyTorch-Geometric 2.0.1 [13] are used to model the GCNN. All experiments for MIPLIB are run on a cluster equipped with Intel Xeon E5-2670 v2 CPUs with 2.50GHz and 128GB main memory, and for NN-Verification on a cluster equipped with Intel Xeon E5-2690 v4 CPUs with 2.60GHz and 128GB main memory.

For instance selection we discard instances from both instance sets that satisfy any of the criteria in Table 2. To minimise bias, instances were discarded if any criteria were triggered in an individual run on any seed under default condition or those tested in Experiment 6.1. We believe that these conditions focus on instances where a good selection strategy of cuts can improve the dual bound in a reasonable amount of time. We note that

[1] MIPLIB 2017 – The Mixed Integer Programming Library `https://miplib.zib.de/`.
[2] `https://github.com/deepmind/deepmind-research/tree/master/neural_mip_solving`

improving the dual bound is a proxy for overall solver performance, and does not necessarily result in improved solution time. We additionally note that only 1000 randomly selected instances from the NN-Verification data set were used as opposed to the entire data set. All instance sets following instance filtering are split into training-test subsets subject to a 80-20 split.

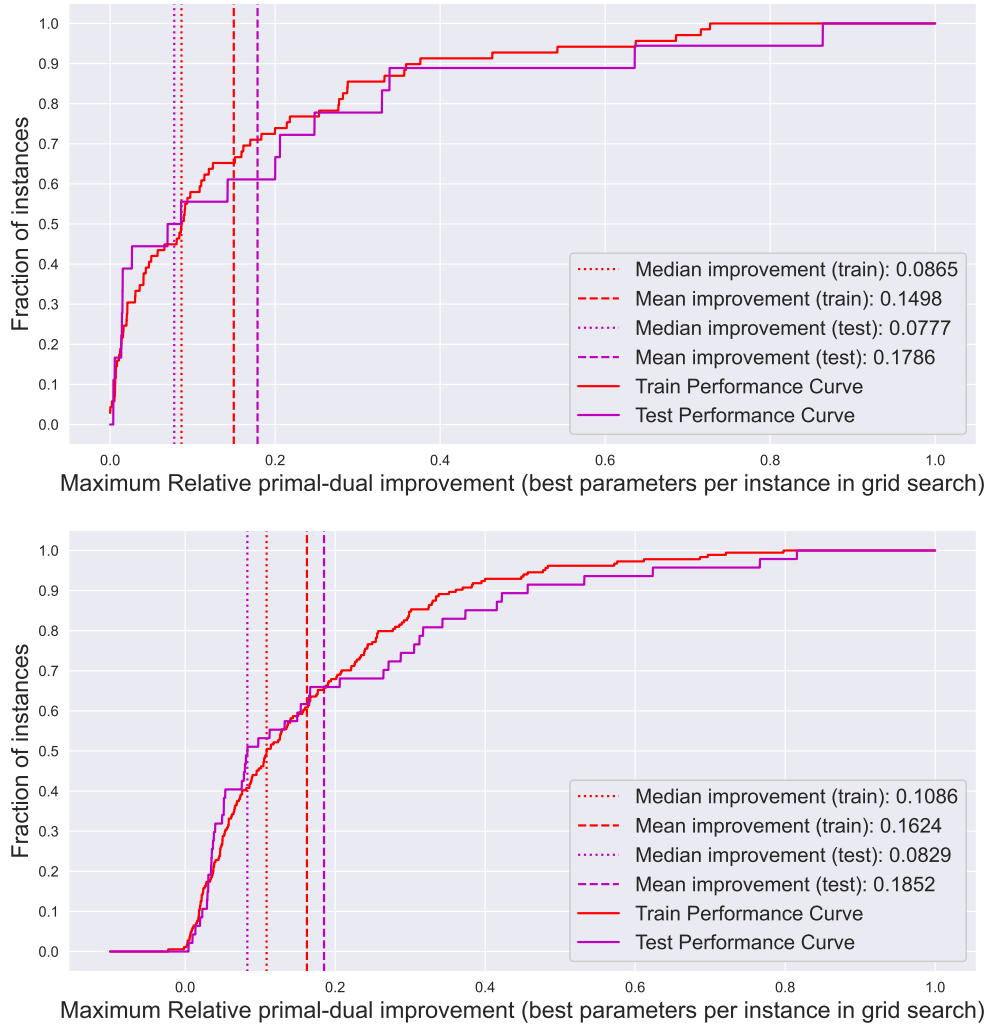

Figure 3 Relative improvement of best choice parameters compared to default parameters in Experiment 6.1. (top) MIPLIB. (bottom) NN-Verification.

6.1 Lower Bounding Potential Improvement

To begin our experiments, we first perform a grid search to give a lower bound on the potential improvement that adaptive cut selection can provide. We generate all parameter scenarios satisfying the following condition:

$$\sum_{i=1}^{4} \lambda_i = 1, \quad \text{where } \lambda_i = \frac{\beta_i}{10}, \quad \beta_i \in \mathbb{N}, \quad \forall\, i \in \{1,2,3,4\}$$

Recall that λ_i for all $i \in \{1,2,3,4\}$ are respectively multipliers of the cut scoring measures normalised directed cutoff distance (`dcd'`), normalised efficacy (`eff'`), integer support (`isp`), and objective parallelism (`obp`).

We solve the root node for all instances and parameter choices, and store the cut selector parameters that result in the smallest primal-dual difference, as well as their relative primal-dual difference improvement compared to that when using default cut selector parameter values. We remove all instances where the worst case parameter choice compared to the best case parameter choice differ by a relative primal-dual difference performance of less than 0.1%. Additionally, we remove instances where a quarter or more of the parameter choices result in the identical best performance. These removals are made due to the sparse learning opportunities provided by the

instances, as the best case performance is minimally different from the worst, or the best case performance is too common. This results in an additional 2.5% and 0.2% of instances being removed for MIPLIB, leaving 87 (8.2%) instances remaining. For NN-Verification no additional instances are removed under these conditions, leaving 231 instances (23.1%) instances remaining. We note that all criteria for instance removal in Table 2 were performed using the grid search results as well as those under default conditions to ensure no bias throughout instance selection.

We conclude from the results presented in Figure 3 that there exists notable amounts of improvement potential per instance from better cut selection rules. Specifically, we observe that the median relative primal-dual difference improvement compared to standard conditions is at least 7.7% over the training and test sets of both MIPLIB and NN-Verification. We consider this difference very large considering at most 500 cuts (50 rounds of 10 cuts) are added, with this value being only a lower bound on potential improvement as the results come from a grid search of the parameter space. Instance specific results for MIPLIB are available in Appendix D.

We draw attention to the aggregated best performing parameter results from the grid search in Table 3. We see in both data sets that a distance based metric has the largest mean value, being λ_1 (multiplier of `dcd'`) for MIPLIB and λ_2 (multiplier of `eff'`) for NN-Verification. We also see that λ_3 and λ_4 take on much larger mean values than those in the default SCIP scoring rule, where they have value 0.1, suggesting that the measures `isp` and `obp` are not only useful in distance dominated scoring rules. These are aggregated results, however, and we note that they best summarise how every measure can be useful for some instances, further motivating the potential of instance-dependent based cut selection. We stress that this motivation is also true for the homogeneous NN-Verification, where all parameters are still useful.

Table 3 Statistics of best choice parameters per instance (train + test) in Experiment 6.1.

	MIPLIB			NN-Verification		
Parameter	Mean	Median	Std Deviation	Mean	Median	Std Deviation
λ_1 (`dcd`)	0.312	0.200	0.252	0.223	0.200	0.204
λ_2 (`eff`)	0.177	0.100	0.181	0.315	0.300	0.216
λ_3 (`isp`)	0.232	0.200	0.226	0.220	0.200	0.214
λ_4 (`obp`)	0.279	0.200	0.248	0.241	0.200	0.218

6.2 Random Seed Initialisation

Let θ_i be the initialised weights and biases using random seed i, where $i \in \mathbb{N}$. To minimise the bias of our initialised policy with respect to $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \lambda_3, \lambda_4)$, the random seed that satisfies (13) is used throughout our experiments.

$$\underset{i\in\{0,\dots,999\}}{\operatorname{argmin}} \sum_{s_0} \left\| \mathbb{E}[\pi_{\theta_i}(\,\cdot\,|s_0)] - \left[\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right] \right\|_1 \tag{13}$$

We believe this random seed minimises bias as the GCNN initially outputs approximately equal values over the data set, allowing the GCNN to best decide the importance of each parameter. This was motivated from the observation that some random initialisations resulted in a cut measure always having an output value of 0 starting from the untrained GCNN. Different random seeds were used for the MIPLIB and NN-Verification experiments, and the random seeds were found using combined training and test sets.

The performance of the randomly initialised GCNN can be seen in Figure 4 and Table 4, with instance specific results available in Appendix D. We observe a larger than expected mean improvement over the test set for MIPLIB, however from the size of the test set and the much lower median improvement, conclude that it's the result of outliers. Surprisingly, the median and mean relative improvement for training and test sets for MIPLIB are positive, while they are negative for NN-Verification. We believe that the positive, albeit small, performance improvement of our random initialisation over default SCIP on MIPLIB is from our slight modification of the cut scoring rule with `eff'` and `dcd'`. For NN-verification, we believe the negative performance comes from the decrease in λ_2 (the multiplier of `eff'`), which is weighted highly in default SCIP, and is important for the instance set according to results in Experiment 6.1.

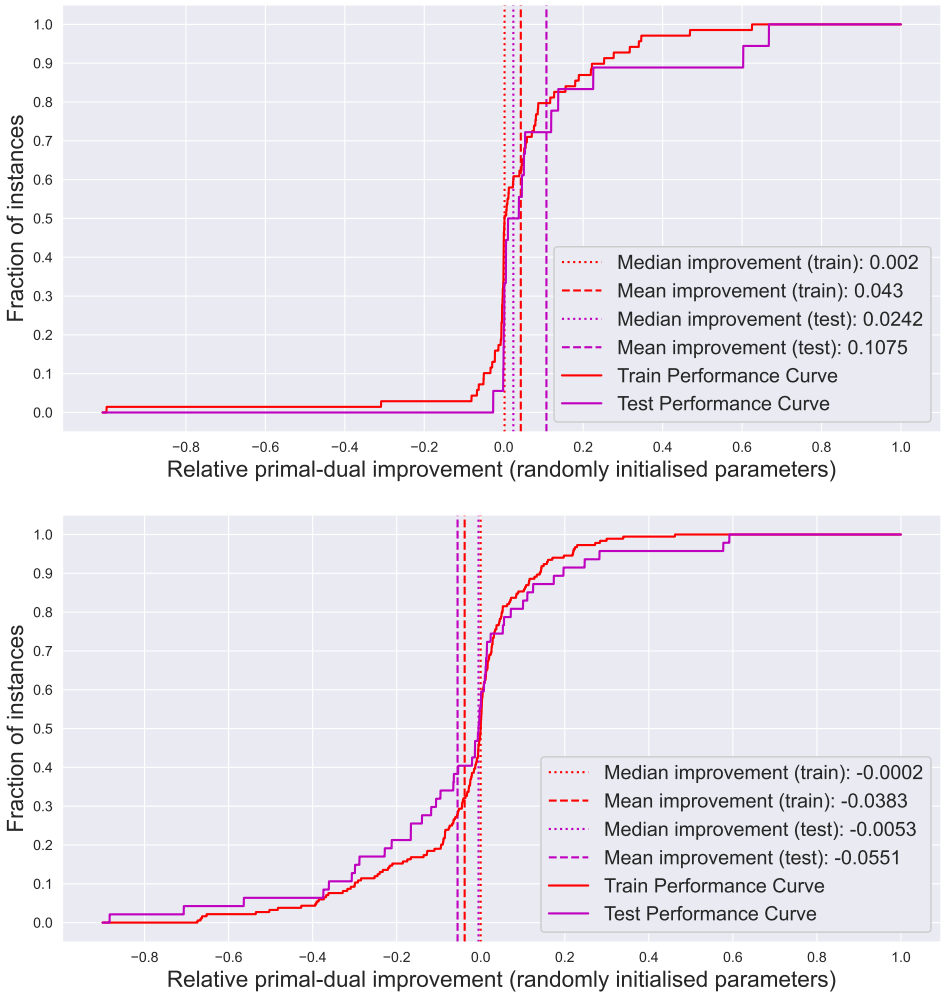

Figure 4 Relative primal-dual difference improvement from random initialised parameters compared to default parameters in Experiment 6.2. (top) MIPLIB. (bottom) NN-Verification.

Table 4 Statistics of random initialised parameters per instance (train + test) in Experiment 6.2.

	MIPLIB			NN-Verification		
Parameter	Mean	Median	Std Deviation	Mean	Median	Std Deviation
λ_1 (`dcd`)	0.253	0.253	0.006	0.253	0.254	0.005
λ_2 (`eff`)	0.294	0.296	0.012	0.215	0.214	0.006
λ_3 (`isp`)	0.306	0.300	0.018	0.279	0.279	0.002
λ_4 (`obp`)	0.147	0.149	0.011	0.253	0.253	0.007

6.3 Standard Learning Method

Before we attempt to determine the capability of our RL framework, policy architecture, and training method, we first design an experiment using SMAC (Sequential Model Algorithm Configuration)[3], see [21]. SMAC is a standard package in the field of algorithm configuration, and is largely based on Bayesian optimisation. Unlike our approach, which returns instance-dependent cut selector parameters, SMAC will return a single set of parameter values that works over the entire instance set. It can therefore be thought of as a more intelligent approach than traditional grid searches, which have been used to define SCIP default parameter values. We therefore aim to outperform SMAC given the adaptive advantage of our algorithm.

We use SMAC4BB, which is targeted at low dimensional and continuous black box functions, and provide SCIP 8.0.1's default values for $\boldsymbol{\lambda}$. We run 250 epochs of SMAC (the same as we will in Experiment 6.4), however we note that our approach requires additional solver calls due to taking more than one sample of cut selector

[3] `https://github.com/automl/SMAC3`

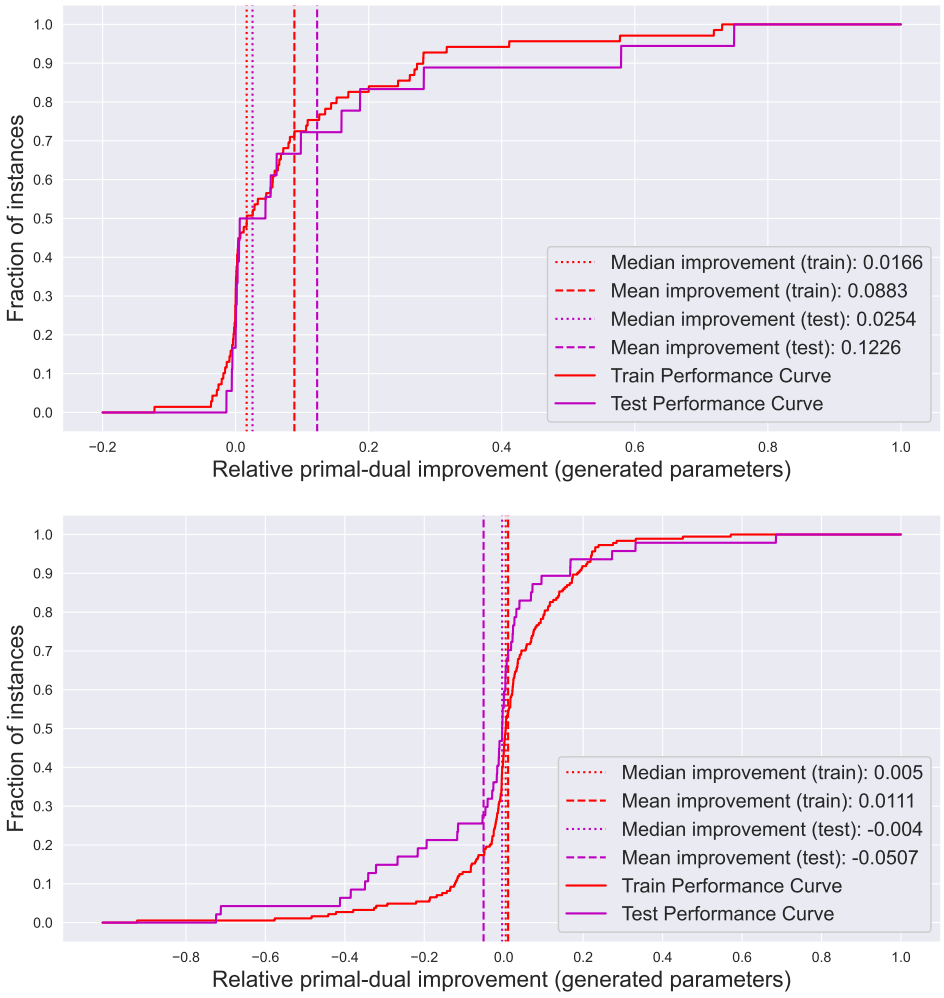

Figure 5 Relative primal-dual difference improvement compared to default SCIP from generated parameters in Experiment 6.3. (top) MIPLIB. (bottom) NN-Verification.

parameters from the generated distributions during training. The function that SMAC attempts to minimise is the average primal-dual difference over all instance-seed pairs relative to that produced by SCIP with default cut selector parameter values.

Table 5 Statistics of generated constant parameters in Experiment 6.3

	MIPLIB	NN-Verification
Parameter	Constant	Constant
λ_1 (`dcd`)	0.600	0.065
λ_2 (`eff`)	0.124	0.633
λ_3 (`isp`)	0.175	0.301
λ_4 (`obp`)	0.100	0.000

We observe an increase in performance over MIPLIB after using SMAC compared to that of the random initialisation as seen in Figure 5. The median improvement over default SCIP for the training set increases to 1.6% from 0.2%, and to 2.5% from 2.4% for the test set, with the mean improvement over both sets increasing by at least 2%. For NN-Verification, we only observe a median increase to 0.5% from −0.02% for the training set and −0.4% from −0.5% for the test set, with the mean performance of each set increasing by less than 2%. From the best found constant parameter choices generated by SMAC as displayed in Table 5, we conclude that an efficacy dominated cut scoring rule, such as default SCIP, is likely the best choice for NN-Verification if restricted to a non-adaptive rule.

6.4 Learning Adaptive Parameters

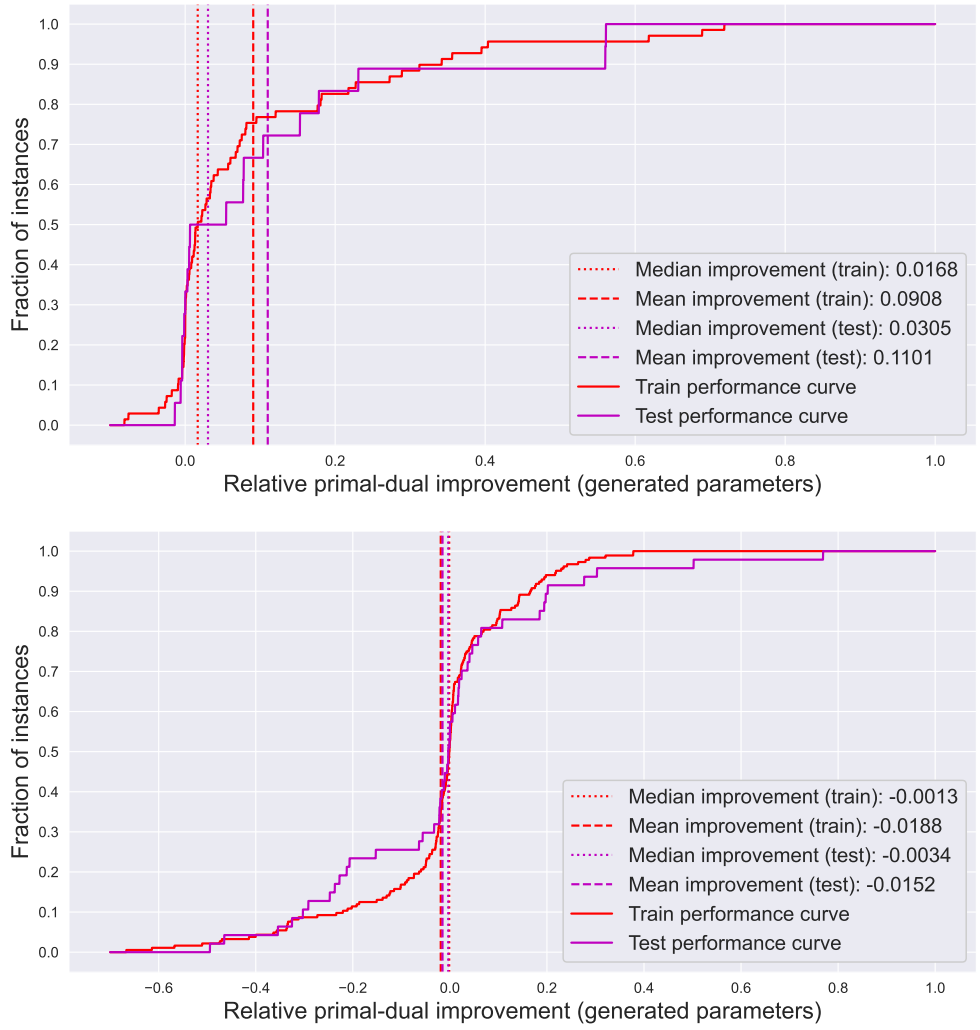

Figure 6 Relative primal-dual difference improvement from generated parameters compared to default SCIP parameters in Experiment 6.4. (top) MIPLIB. (bottom) NN-Verification.

We now show the performance of our RL framework, policy architecture, and training method compared to default SCIP parameter choices over MIPLIB and NN-Verification. To do so, we run 5000 iterations of Algorithm 2 (250 epochs), with n_{samples} set to 20, and allocate 10% of instances from the training set per batch. γ of the multivariate normal distribution, $\mathcal{N}_4(\mu, \gamma I)$, is defined by the following, where n_{epochs} is the total amount of iterations of Algorithm 2 and i_{epoch} is the current epoch:

$$\gamma(i_{\text{epoch}}, n_{\text{epochs}}) := 0.01 - \frac{0.009 * i_{\text{epoch}}}{n_{\text{epochs}}}, \quad i_{\text{epoch}}, n_{\text{epochs}} \in \mathbb{N}, \quad i_{\text{epoch}} \leq n_{\text{epochs}} \tag{14}$$

We note that γ represents one of many opportunities, such as the GCNN structural design and training algorithm, where a substantial amount of additional effort could be invested to (over)tune the learning experiment. We also note that a forward pass of the trained network takes on average less than 0.1s over both data sets, see Table 8 in Appendix D, and that updating the GCNN is negligible w.r.t. time compared to solving the MILPs.

The randomly initialised GCNN over the training set of MIPLIB has a median relative primal-dual difference improvement of 0.2% over default as seen in Figure 4, compared to the 1.7% of our MIPLIB trained GCNN as seen in Figure 6. This improvement is minimally better than the 1.6% improvement over default from SMAC in Figure 5, with our approach slightly improving over SMAC for the training set of MIPLIB, and performing comparably over the test set, having better median improvement and worse mean. These results suggest that our approach works, in that it is comparable with other standard approaches, and can provide improvement over

default parameter choices, but that it is unable to capture the full extent of performance improvement that is shown to exist in Experiment 6.1. Interestingly, we note that over MIPLIB, Experiments 6.1–6.4 all on average set λ_1 (the multiplier for `dcd`) to be the largest coefficient, as seen in Tables 3, 5, and 6. This is in contrast to the default parameter values used in SCIP 8.0, where the multiplier is set to 0. We believe this difference is strengthened by our computational setup, where we provide SCIP a good initial starting solution. This starting solution is often optimal and better than what initial heuristics would produce. Additionally, as we only add cuts to the root node, the distance to the cut in the direction of the primal solution will reliably point inside the feasible region. This is not the case when the search space has been partitioned like in branch and bound. We also note that over all experiments and learning techniques, for the heterogeneous data set MIPLIB, every cut measure is useful for some instances. For specific instance results, see Appendix D.

For NN-Verification, we see from the data presented in Figure 6 that our framework, similar to SMAC, failed to perform on the homogeneous data set and capture the performance improvement that was shown to exist in Experiment 6.1. We performed comparably to the random initialisation from which training began, and converged to an efficacy dominated scoring rule featuring integer support, see Table 6, in a near identical manner to the constant scoring rule learned by SMAC. Interestingly, we see that the standard deviation for all measures is near 0, meaning that our framework converged to a constant output. We believe that this is due to a local minima existing for an efficacy dominated model, with default parameters being quite good, and our restriction to static based features, i.e. those available before the first LP solve, being insufficiently diverse for the more homogeneous NN-Verification.

Table 6 Statistics of generated parameters per instance (train + test) in Experiment 6.4

	MIPLIB			NN-Verification		
Parameter	Mean	Median	Std Deviation	Mean	Median	Std Deviation
λ_1 (`dcd`)	0.414	0.385	0.094	0.000	0.000	0.000
λ_2 (`eff`)	0.171	0.157	0.058	0.592	0.591	0.002
λ_3 (`isp`)	0.232	0.244	0.052	0.408	0.408	0.002
λ_4 (`obp`)	0.183	0.211	0.072	0.000	0.000	0.000

6.5 Generalisation to Branch and Bound

Until this point we have focused on root node restricted experiments and used the primal-dual difference as a surrogate for solver performance. We now deploy the best instance-dependent parameter values from the grid search in Experiment 6.1 and from our approach in Experiment 6.4 to the full solving process. We keep the same sandbox environment that we have used until this point, however we no longer limit ourselves to the root node, and we set a time limit of 7200s.

From the results presented in Table 7, specifically Table 7 (a), we see that instance-dependent cut selector parameters that induce good root node performance do generalise to the larger solving process. This follows from the best grid search instance-dependent parameter values from Experiment 6.1 clearly outperforming the default parameter choice. We thus believe that in general, the primal-dual difference (or gap) after applying cuts is an adequate surrogate of overall solver performance for a given set of parameters. We note, however, that there exists many solution paths where this statement is not true, and many instances where dual bound progression at the root is a poor surrogate. The improvement generalisation was not as clear for our framework as observable in Table 7 (b), with our framework outperforming both data sets in terms of time, and losing in terms of nodes. Interestingly, over MIPLIB instance-seed pairs that time out, our framework has a better dual bound than default 66.11% of the time.

7 Conclusion

We presented a parametric family of MILPs together with infinitely many family-wide valid cuts. We showed for a specific cut selection rule, that any finite grid search of the parameter space will always miss all parameter values, which select integer optimal inducing cuts in an infinite amount of our instances. We then presented a reinforcement learning framework for learning cut selection parameters, and phrased cut selection in MILP as a Markov decision process. By representing MILP instances as a bipartite graph, we used policy gradient methods to train a graph convolutional neural network.

Table 7 Results of generated cut selection parameters compared to SCIP default parameters. *Time* is a comparison of the solution time of a run, *Nodes* the number of nodes, and *Dual bound* the dual bound when the time limit is hit. For *Time*, instance-seed pairs are considered when at most one of the two runs (default parameters and generated parameters) hit the time limit. For *Nodes*, instance-seed pairs that always solved to optimality are considered, and for *Dual bound* instance-seed pairs that always hit the time limit are considered. The columns *Wins* and *Ties* are the percentage of instance-seed pairs for which the generated parameters outperformed or respectively tied with the default parameters under the given metric.

(a) Generalisation to branch and bound of instance-dependent parameters from Experiment 6.1

	MIPLIB			NN-Verification		
Metric	Instance-Seeds	% Wins	% Ties	Instances	% Win	% Ties
Time	81	60.49	0.00	675	56.74	0.00
Nodes	73	58.90	10.96	628	56.21	1.59
Dual bound	180	80.05	1.66	18	72.22	0.00

(b) Generalisation to branch and bound of instance-dependent parameters from Experiment 6.4.

	MIPLIB			NN-Verification		
Metric	Instance-Seeds	% Wins	% Ties	Instances	% Win	% Ties
Time	81	53.09	0.00	673	50.37	0.00
Nodes	74	41.89	12.16	632	47.78	1.74
Dual bound	180	66.11	2.22	20	50.00	0.00

The framework generates good performing, albeit sub-optimal, parameter values for a modified variant of SCIP's default cut scoring rule over MIPLIB 2017, with the performance being comparable to standard learning techniques, and clearly better than the random initialisation. Our framework, however, was subject to mode collapse over the NN-Verification data set, and failed to generate a diverse and well performing set of instance-dependent cut selector parameter values.

Results from our grid search experiments showed that there is a large amount of potential improvements to be made in adaptive cut selection, with a median relative primal-dual difference improvement of 7.77% over MIPLIB and 8.29% over NN-Verification with only 50 rounds of 10 cuts. The generalisation of these best performing instance-dependent parameter values to branch and bound then revealed a correlation between the primal-dual difference after cut rounds and overall solver performance in terms of both solution time and number of nodes.

We suggest three key areas of further research for those wanting to build on this research. Firstly, there is a dire need for more instance sets that are sufficiently diverse, non-trivial, yet not overly difficult. Secondly, throughout this paper we restricted ourselves to individual cut measures already featured in SCIP's default rule. Further research could explore rules containing non-linear combinations of additional measures. Third and finally, we suggest that a focus on the larger selection algorithm could lead to further improved performance. For all experiments the separator algorithm's parameters were set to constant values, and we ignored other cut selector related parameters, and restricted ourselves to parallelism based filtering method. We end by noting that a major contribution of this work, the new cut selector plugin for SCIP, enables the last two key areas of further research via easy inclusion of custom cut selection algorithms in a modern MILP solver.

A Proof of Theorem 1 from Section 3

For the following theorem, we will simulate a pure cutting plane approach to solving MILPs using scoring rule (4). We will use custom MILPs, cutting planes, and select exactly one cut per round. Each call to the selection subroutine is called an *iteration* or *round*. The theorem is intended to show how a fixed cut selection rule can consistently choose "bad" cuts.

Theorem 1. *Given a finite discretisation of λ, an infinite family of MILP instances together with an infinite amount of family-wide valid cuts can be constructed. Using a pure cutting plane approach and applying a single cut per selection round, the infinite family of instances do not solve to optimality for any value in the discretisation, but do solve to optimality for an infinite amount of alternative λ values.*

The parametric MILP we use to represent our infinite family of instances is defined as follows, where $a \in \mathbb{R}_{\geq 0}$ and $d \in [0, 1]$:

$$\mathsf{P}(a,d) := \begin{cases} \min & x_1 - (10+d)x_2 - ax_3 \\ & -\frac{1}{2}x_2 + 3x_3 \leq 0 \\ & -x_3 \leq 0 \\ \text{s.t.} & -\frac{1}{2}x_1 + \frac{1}{2}x_2 - \frac{7}{2}x_3 \leq 0 \\ & \frac{1}{2}x_1 + \frac{3}{2}x_3 \leq \frac{1}{2} \\ & x_1 \in \mathbb{Z}, \quad x_2 \in \mathbb{R}, \quad x_3 \in \{0,1\} \end{cases}$$

The polytope of our MILPs LP relaxation is the convex hull of the following points:

$$\mathcal{X} := \left\{(0,0,0), (1,0,0), (1,1,0), \left(\frac{-1}{2}, 3, \frac{1}{2}\right)\right\} \tag{15}$$

The convex hull of $\mathcal{X}$ is a 3-simplex, or alternatively a tetrahedron, see Figure 7 for a visualisation. For such a feasible region, we can exhaustively write out all integer feasible solutions:

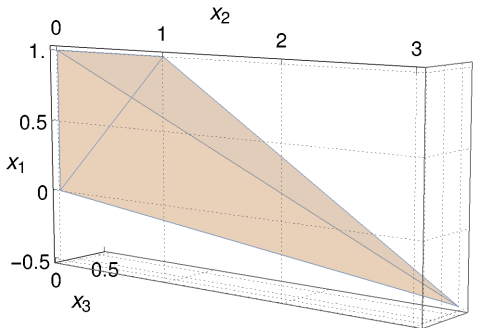

Figure 7 The feasible region of the LP relaxation of $\mathsf{P}(a,d)$.

Lemma 2. *The integer feasible set of* $\mathsf{P}(a,d)$*,* $(a,d) \in \mathbb{R}_{\geq 0} \times [0,1]$ *is:*

$$\{(0,0,0)\} \cup \{(1,x_2,0) : \forall\, x_2 \in [0,1]\}$$

As we are dealing with linear constraints and objectives, we know that $(1,x_2,0)$, where $0 < x_2 < 1$ cannot be optimal without both $(1,1,0)$ and $(1,0,0)$ being also optimal. We therefore simplify the integer feasible set, $\mathcal{I}_{\mathcal{X}}$, to:

$$\mathcal{I}_{\mathcal{X}} := \{(0,0,0), (1,0,0), (1,1,0)\} \tag{16}$$

At each iteration of adding cuts we will always present exactly three candidate cuts. We name these cuts as follows:

- The "good cut", denoted $\mathcal{GC}$: Applying this cut immediately results in the next LP solution being integer optimal.
- The "integer support cut", denoted $\mathcal{ISC}^n$: Applying this cut will result in a new LP solution barely better than the previous iteration. The cut has very high integer support as the name suggests, and would be selected if λ from (4) is set to a high value. The superscript n refers to the iteration number.
- The "objective parallelism cut", denoted $\mathcal{OPC}^n$: Applying this cut will also result in a new LP solution barely better than the previous iteration. The cut has very high objective parallelism, and would be selected if λ from (4) is set to a low value. The superscript n refers to the iteration number.

The cuts are defined as follows, where $\mathcal{GC}$ has an additional property of it being selected in the case of a scoring tie:

$$\mathcal{GC} : -10x_1 + 10x_2 + x_3 \leq 0 \tag{17}$$

$$\mathcal{ISC}^n : -x_1 + x_3 \leq 1 - \epsilon_n \tag{18}$$

$$\mathcal{OPC}^n : -x_1 + 10x_2 \leq \frac{61}{2} - \epsilon_n \tag{19}$$

We use ϵ_n here to denote a small shift of the cut, with a greater ϵ_n resulting in a deeper cut. We define ϵ_n as follows:

$$0 < \epsilon_i < \epsilon_{i+1} \quad \forall\, i \in \mathbb{N} \qquad \lim_{n\to\infty} \epsilon_n = 0.1 \tag{20}$$

A better overview of the proof of Theorem 1 can now be imagined. At each cut selection round we present three cuts, where for high values of λ, $\mathcal{ISC}^n$ is selected, and for low values $\mathcal{OPC}^n$ is selected. As the scoring rule (4) is linear w.r.t. λ, our aim is to controllably sandwich the intermediate values of λ that will select $\mathcal{GC}$. Specifically, for any given Λ, we aim to construct an infinite amount of parameter values for a and d s.t. the intermediate values of λ all belong to $\widetilde{\Lambda}$.

Lemma 3. *The vertex set of the LP relaxation of* $\mathsf{P}(a,d), (a,d) \in \mathbb{R}_{\geq 0} \times [0,1]$*, after having individually applied cuts* (17)*,* (18)*, or* (19) *are, respectively:*

$$\mathcal{GC}_{\mathcal{X}} := \mathcal{I}_{\mathcal{X}} \cup \left\{ \left(\frac{61}{91}, \frac{60}{91}, \frac{10}{91} \right) \right\} \tag{21}$$

$$\mathcal{ISC}^n_{\mathcal{X}} := \mathcal{I}_{\mathcal{X}} \cup \left\{ \left(\frac{-1}{2} + \frac{3\epsilon_n}{4}, 3 - \frac{3\epsilon_n}{2}, \frac{1}{2} - \frac{\epsilon_n}{4} \right), \left(\frac{-1}{2} + \frac{3\epsilon_n}{4}, 3 - \epsilon_n, \frac{1}{2} - \frac{\epsilon_n}{4} \right), \left(\frac{-1}{2} + \frac{\epsilon_n}{2}, 3 - 3\epsilon_n, \frac{1}{2} - \frac{\epsilon_n}{2} \right) \right\} \tag{22}$$

$$\mathcal{OPC}^n_{\mathcal{X}} := \mathcal{I}_{\mathcal{X}} \cup \left\{ \left(\frac{-1}{2} + \frac{\epsilon_n}{21}, 3 - \frac{2\epsilon_n}{21}, \frac{1}{2} - \frac{\epsilon_n}{63} \right), \left(\frac{-1}{2} + \frac{3\epsilon_n}{43}, 3 - \frac{4\epsilon_n}{43}, \frac{1}{2} - \frac{\epsilon_n}{43} \right), \left(\frac{-1}{2} + \frac{\epsilon_n}{61}, 3 - \frac{6\epsilon_n}{61}, \frac{1}{2} - \frac{\epsilon_n}{61} \right) \right\} \tag{23}$$

Proof. Apply $\mathcal{GC}$, $\mathcal{ISC}^n$, and $\mathcal{OPC}^n$ to $\mathsf{P}(a,d)$ individually and then compute the vertices of the convex hull of the LP relaxation. ◀

We note that because both integer support and objective parallelism do not depend on the current LP solution, iteratively applying deeper cuts of the same kind would leave the cut's scores unchanged. Thus, provided they do not separate any integer points and continue to cut off the LP solution, deeper cuts of the same kind can be recursively applied. This is why both $\mathcal{ISC}^n$ and $\mathcal{OPC}^n$ have a superscript.

After applying a cut of one kind, e.g. $\mathcal{ISC}^n$, we cannot always simply increment n in the other cut, e.g. $\mathcal{OPC}^{n+1}$. This is because $\mathcal{OPC}^{n+1}$ does not guarantee separation of the now new LP solution in problem $\mathsf{P}(a,d) \cap \mathcal{ISC}^n$ for all sequences of $\{\epsilon_n, \epsilon_{n+1}\}$. Instead, we create a variant of $\mathcal{OPC}^{n+1}$, namely $\widehat{\mathcal{OPC}^{n+1}}$, which will always entirely remove the facet of the LP created by adding $\mathcal{ISC}^n$ independent of how the series of ϵ_n values increase. Once again, we note that as only the RHS values are changing, the score for all cuts within the same type remain unchanged, and thus no two cuts from different types can be applied. We have proven our results using Mathematica [31], and a complete notebook containing step-by-step instructions can be found at `https://github.com/Opt-Mucca/Adaptive-Cutsel-MILP`. Below we will outline the necessary cumulative lemmas to prove Theorem 1, and summarise the calculations we have taken to achieve each step.

Lemma 4. *Having applied the cut* $\mathcal{ISC}^n$ *to* $\mathsf{P}(a,d), (a,d) \in \mathbb{R}_{\geq 0} \times [0,1]$*, a new facet is created. Applying either* $\mathcal{GC}$ *or a deeper variant of* $\mathcal{OPC}^{n+1}$ *cuts off that facet. The deeper variant, denoted* $\widehat{\mathcal{OPC}^{n+1}}$*, which differs from* $\mathcal{OPC}^{n+1}$ *only by the RHS value is defined as:*

$$\widehat{\mathcal{OPC}^{n+1}} : -x_1 + 10x_2 \leq \frac{61}{2} - 31\epsilon_n \tag{24}$$

Proof. One can first verify that the vertex set of the facet is $\mathcal{ISC}^n_{\mathcal{X}} \setminus \mathcal{I}_{\mathcal{X}}$. One can then find the smallest ϵ' s.t the following cut is valid for all $\mathbf{x} \in \mathcal{ISC}^n_{\mathcal{X}} \setminus \mathcal{I}_{\mathcal{X}}$:

$$-x_1 + 10x_2 \leq \frac{61}{2} - \epsilon'$$

The statement is valid for all $\epsilon' > \frac{61\epsilon_n}{2}$, and we arbitrarily select $\epsilon' = 31\epsilon_n$. One can also check that $\mathcal{GC}$ dominates $\mathcal{ISC}^n$ by seeing that it separates all vertices of $\mathcal{ISC}^n_{\mathcal{X}} \setminus \mathcal{I}_{\mathcal{X}}$ for all $n \in \mathbb{N}$. Finally, we need to ensure that no integer solution is cut off. We can verify this by checking that every $\mathbf{x} \in \mathcal{I}_{\mathcal{X}}$ satisfies (24). This statement holds whenever $\epsilon_n < 0.1$. Therefore $\epsilon' = 31\epsilon_n$ is valid, and we arrive at the cut $\widehat{\mathcal{OPC}^{n+1}}$. ◀

Lemma 5. *Having applied the cut* $\mathcal{OPC}^n$ *to* $\mathsf{P}(a,d), (a,d) \in \mathbb{R}_{\geq 0} \times [0,1]$*, a new facet is created. Applying* $\mathcal{ISC}^{n+1}$ *or* $\mathcal{GC}$ *cuts off that facet.*

Proof. This follows the same structure as the proof of Lemma 4. We get that $\epsilon' > \frac{4\epsilon_n}{43}$, and that $\epsilon' = \epsilon_{n+1}$ is valid w.r.t. the integer constraints. ◀

Using our definition of integer support and objective parallelism in (5)–(6), we derive the scores for each cut from the simple cut selection scoring rule (4). We let $\mathbf{c}_{\mathsf{P}(a,d)}$ denote the vector of coefficients from the objective of $\mathsf{P}(a,d)$, $(a,d) \in \mathbb{R}_{\geq 0} \times [0,1]$. The integer support and objective parallelism values of each cut are as follows:

$$\texttt{isp}(\mathcal{GC}) = \frac{2}{3} \tag{25}$$

$$\texttt{isp}(\mathcal{ISC}^n) = 1 \qquad \forall\, n \in \mathbb{N} \tag{26}$$

$$\texttt{isp}(\mathcal{OPC}^n) = \frac{1}{2} \qquad \forall\, n \in \mathbb{N} \tag{27}$$

$$\texttt{obp}(\mathcal{GC}, \mathbf{c}_{\mathsf{P}(a,d)}) = \frac{110 + a + 10d}{\sqrt{201}\sqrt{1 + a^2 + (10+d)^2}} \tag{28}$$

$$\texttt{obp}(\mathcal{ISC}^n, \mathbf{c}_{\mathsf{P}(a,d)}) = \frac{1 + a}{\sqrt{2}\sqrt{1 + a^2 + (10+d)^2}} \qquad \forall\, n \in \mathbb{N} \tag{29}$$

$$\texttt{obp}(\mathcal{OPC}^n, \mathbf{c}_{\mathsf{P}(a,d)}) = \frac{101 + 10d}{\sqrt{101}\sqrt{1 + a^2 + (10+d)^2}} \qquad \forall\, n \in \mathbb{N} \tag{30}$$

Using our simplified cut scoring rule as defined in (4), we derive the necessary conditions defining the λ values, which assign $\mathcal{GC}$ a score at least as large as the other cuts.

Lemma 6. *$\mathcal{GC}$ is selected and added to $\mathsf{P}(a,d)$, $(a,d) \in \mathbb{R}_{\geq 0} \times [0,1]$, using scoring rule (4) if and only if a λ is used that satisfies the following conditions:*

$$\lambda * \texttt{isp}(\mathcal{GC}) + (1-\lambda) * \texttt{obp}(\mathcal{GC}, \mathbf{c}_{\mathsf{P}(a,d)}) \geq \lambda * \texttt{isp}(\mathcal{ISC}^n) + (1-\lambda) * \texttt{obp}(\mathcal{ISC}^n, \mathbf{c}_{\mathsf{P}(a,d)}) \tag{31}$$

$$\lambda * \texttt{isp}(\mathcal{GC}) + (1-\lambda) * \texttt{obp}(\mathcal{GC}, \mathbf{c}_{\mathsf{P}(a,d)}) \geq \lambda * \texttt{isp}(\mathcal{OPC}^n) + (1-\lambda) * \texttt{obp}(\mathcal{OPC}^n, \mathbf{c}_{\mathsf{P}(a,d)}) \tag{32}$$

Proof. We know that the integer support and objective parallelism do not depend on ϵ_n as seen in equations (25)–(30). Our cut selector rule also selects exactly one cut per iteration, namely the largest scoring cut. Therefore, whenever λ satisfies constraints (31) and (32), $\mathcal{GC}$ will be selected over both $\mathcal{OPC}^n$ and $\mathcal{ISC}^n$, and applied to $\mathsf{P}(a,d)$. If λ does not satisfy constraints (31) and (32), then $\mathcal{GC}$ is not the largest scoring cut and will not be applied to $\mathsf{P}(a,d)$. ◀

The inequalities (31)–(32) define the region, $\mathcal{R}_{\mathcal{GC}}$, which exactly contains all tuples $(a,d,\lambda) \in \mathbb{R}_{\geq 0} \times [0,1]^2$ that result in $\mathcal{GC}$ being the best scoring cut. The region, $\mathcal{R}_{\mathcal{GC}}$ is visualised in Figure 8. We define the function $\mathtt{r}_{\mathcal{GC}}(a,d)$ for all $(a,d) \in \mathbb{R}_{\geq 0} \times [0,1]$, which maps any pairing of (a,d) to the set of λ values contained in $\mathcal{R}_{\mathcal{GC}}$ for the corresponding fixed (a,d) values.

$$\mathtt{r}_{\mathcal{GC}} : \mathbb{R}_{\geq 0} \times [0,1] \to \mathcal{P}([0,1]) \tag{33}$$

Here $\mathcal{P}$ refers to the power set. We are interested in $\mathcal{R}_{\mathcal{GC}}$ as we believe that we can find a continuous function that contains all $(a,d,\lambda) \in \mathbb{R}_{\geq 0} \times [0,1]^2$ pairings, which score all cuts equally. Using this function, we can find for a fixed $d \in [0,1]$, the values of $a \in \mathbb{R}_{\geq 0}$ that would result in a $\lambda \in [0,1]$ value that scores all cuts equally. By perturbing our value of $a \in \mathbb{R}_{\geq 0}$, we aim to generate an infinite amount of $\lambda \in [0,1]$ values that score $\mathcal{GC}$ the largest. Then, by choosing different values of $d \in [0,1]$ originally, we aim to find such an infinite set of λ values that can adaptively lie between any given finite discretisation of $[0,1]$.

Lemma 7. *There exists a closed form symbolic solution for the maximum value of a in terms of d over $\mathcal{R}_{\mathcal{GC}}$. We denote this maximum value of a as a function over d, namely $\mathtt{a}_{\max}(d)$, $d \in [0,1]$.*

Proof. We verify this by solving the following optimisation problem, where $\mathtt{a}_{\max}(d)$ is printed in Appendix B:

$$\underset{a,\lambda}{\operatorname{argmax}}\{a \mid (a,d,\lambda) \in \mathcal{R}_{\mathcal{GC}},\ \ a \geq 0,\ \ 0 \leq d \leq 1,\ \ 0 \leq \lambda \leq 1\}$$ ◀

Lemma 8. *Closed form symbolic solutions for the upper and lower bounds of λ can be found. These bounds are continuous functions defined over $0 \leq d \leq 1$ and $0 \leq a \leq \mathtt{a}_{\max}(d)$, and we refer to them as $\lambda_{ub}(a,d)$ and $\lambda_{lb}(a,d)$ respectively.*

Proof. We know that the region respects inequalities (31) and (32) and that $\mathtt{a}_{\max}(d)$ is an upper bound on a for all $d \in [0,1]$. Using this information, we can rearrange the inequalities to get $\lambda_{ub}(a,d)$ and $\lambda_{lb}(a,d)$. The

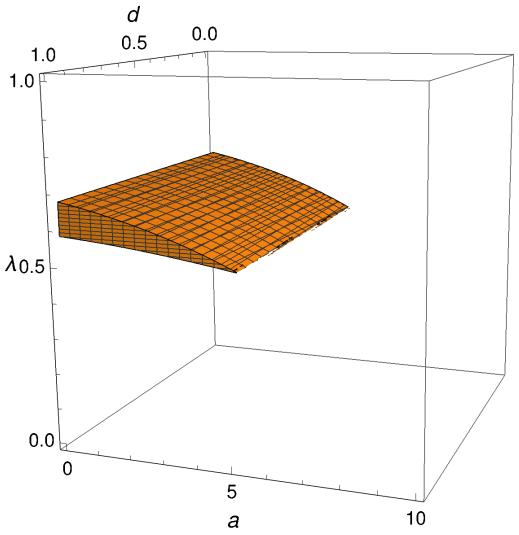

Figure 8 The region, $\mathcal{R}_{\mathcal{GC}}$, where $\mathcal{GC}$ is scored at least as large as both $\mathcal{OPC}^n$ and $\mathcal{ISC}^n$ under cut selection rule (4).

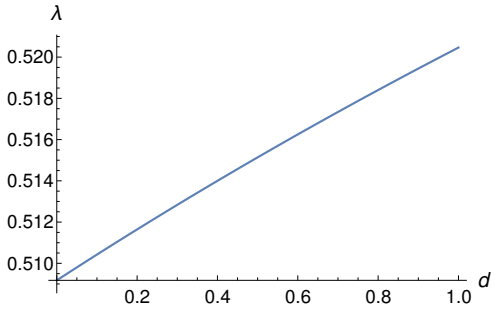

Figure 9 Plot of λ values that scores all cuts equally for all $d \in [0,1]$. That is, $\lambda_{ub}(\mathtt{a}_{\max}(d), d)$, $d \in [0,1]$ (identically: $\lambda_{lb}(\mathtt{a}_{\max}(d), d)$)

result for both $\lambda_{ub}(a,d)$ and $\lambda_{lb}(a,d)$ is a ratio of polynomials in terms of the parameters a and d. As the zeros of the denominators lie outside of the domains $0 \leq d \leq 1$ and $0 \leq a \leq \mathtt{a}_{\max}(d)$, we can conclude that both $\lambda_{ub}(a,d)$ and $\lambda_{lb}(a,d)$ are continuous and defined over our entire domain. These bounds define the interval, $[\lambda_{lb}(a,d), \lambda_{ub}(a,d)]$, of λ values for any fixed a and d, which result in $\mathcal{GC}$ being the largest scoring cut. Taken together with the bounds $0 \leq d \leq 1$ and $0 \leq a \leq \mathtt{a}_{\max}(d)$ they make up $\mathcal{R}_{\mathcal{GC}}$. ◀

Lemma 9. *The lower and upper bounds for λ meet at $a = \mathtt{a}_{\max}(d)$, $d \in [0,1]$. That is, $\lambda_{ub}(\mathtt{a}_{\max}(d), d) = \lambda_{lb}(\mathtt{a}_{\max}(d), d)$ for all $0 \leq d \leq 1$. This means that for all $d \in [0,1]$, $\lambda = \lambda_{ub}(\mathtt{a}_{\max}(d), d)$ (identically $\lambda = \lambda_{lb}(\mathtt{a}_{\max}(d), d)$) would score all cuts equally.*

Proof. This can be checked by substituting $\mathtt{a}_{\max}(d)$ into the equations of $\lambda_{ub}(\mathtt{a}_{\max}(d), d)$ and $\lambda_{lb}(\mathtt{a}_{\max}(d), d)$, equating both sides and rearranging. The result is that $\lambda_{ub}(\mathtt{a}_{\max}(d), d) = \lambda_{lb}(\mathtt{a}_{\max}(d), d)$. ◀

Lemma 10. *$\lambda_{ub}(\mathtt{a}_{\max}(d), d)$ (identically: $\lambda_{lb}(\mathtt{a}_{\max}(d), d)$), where $0 \leq d \leq 1$, is a continuous function and has different valued end points.*

Proof. We know from Lemma 8 that $\lambda_{ub}(a,d)$ is continuous, and can conclude that $\lambda_{ub}(\mathtt{a}_{\max}(d), d)$ is continuous. The different valued endpoints can be derived by evaluating $\lambda_{ub}(\mathtt{a}_{\max}(0), 0)$ and $\lambda_{ub}(\mathtt{a}_{\max}(1), 1)$, which have the relation $\lambda_{ub}(\mathtt{a}_{\max}(1), 1) > \lambda_{ub}(\mathtt{a}_{\max}(0), 0)$. ◀

Figure 9 visualises the function $\lambda_{ub}(\mathtt{a}_{\max}(d), d)$ (identically $\lambda_{lb}(\mathtt{a}_{\max}(d), d)$) for $0 \leq d \leq 1$. For any $d \in [0,1]$, these functions alongside slight changes to $\mathtt{a}_{\max}(d)$, will be used to generate intervals of λ values, which score $\mathcal{GC}$ the largest and lie between a finite discretisation of $[0,1]$.

Lemma 11. *$\lambda_{ub}(a,d)$ - $\lambda_{lb}(a,d) > 0$ for all $0 \leq d \leq 1$ and $0 \leq a < \mathtt{a}_{\max}(d)$. That is, $a = \mathtt{a}_{\max}(d)$ is the only time at which $\lambda_{ub}(a,d) = \lambda_{lb}(a,d)$ for $0 \leq a \leq \mathtt{a}_{\max}(d)$.*

Proof. We can verify this by setting the constraints (31)–(32) to hard equalities, and then solving over the domain $0 \leq d \leq 1$ and $0 \leq a \leq \mathtt{a}_{\max}(d)$. Solving such a system gives the unique solution $a = \mathtt{a}_{\max}(d)$ for $0 \leq d \leq 1$. As $\lambda_{ub}(0,d) > \lambda_{lb}(0,d)$, for all $d \in [0,1]$, and both $\lambda_{ub}(a,d)$ and $\lambda_{lb}(a,d)$ are continuous functions from Lemma 8, we can conclude that $\lambda_{ub}(a,d)$ - $\lambda_{lb}(a,d) > 0$ for all $0 \leq d \leq 1$ and $0 \leq a < \mathtt{a}_{\max}(d)$. ◀

Lemma 12. *An interval* $[\lambda_{lb}(a',d), \lambda_{ub}(a',d)] \subseteq \mathtt{r}_{\mathcal{GC}}(a',d)$ *can be constructed, where* $\lambda_{ub}(a',d) - \lambda_{lb}(a',d) > 0$ *for all* $0 \leq d \leq 1$ *and* $0 \leq a' < \mathtt{a}_{\max}(d)$.

Proof. We define following function, $\widehat{\mathtt{a}_{\max}}(d,\widehat{\epsilon})$, representing $\mathtt{a}_{\max}(d)$ with a shift of $\widehat{\epsilon}$:

$$\widehat{\mathtt{a}_{\max}}(d,\widehat{\epsilon}) := \mathtt{a}_{\max}(d) - \widehat{\epsilon}, \text{ where } \quad 0 \leq d \leq 1, \quad 0 < \widehat{\epsilon} \leq \mathtt{a}_{\max}(d) \tag{34}$$

We know from Lemma 11 that $a = \mathtt{a}_{\max}(d)$ is the only time at which $\lambda_{lb}(a,d) = \lambda_{ub}(a,d)$ for any $d \in [0,1]$. We also know that $\lambda_{ub}(a,d)$ and $\lambda_{lb}(a,d)$ are defined over all $0 \leq d \leq 1$ and $0 \leq a \leq \mathtt{a}_{\max}(d)$. Therefore the following holds for any $d \in [0,1]$ and $\widehat{\epsilon} \in (0, \mathtt{a}_{\max}(d)]$:

$$\lambda_{ub}(\widehat{\mathtt{a}_{\max}}(d,\widehat{\epsilon}),d) - \lambda_{lb}(\widehat{\mathtt{a}_{\max}}(d,\widehat{\epsilon}),d) > 0$$

Additionally, by the definition of $\mathcal{R}_{\mathcal{GC}}$ from the inequalities (31)–(32), we know that the following interval is connected:

$$\mathbf{I}(\widehat{\mathtt{a}_{\max}}(d,\widehat{\epsilon}),d) := [\lambda_{lb}(\widehat{\mathtt{a}_{\max}}(d,\widehat{\epsilon}),d), \lambda_{ub}(\widehat{\mathtt{a}_{\max}}(d,\widehat{\epsilon}),d)], \text{ where } \quad 0 \leq d \leq 1, \quad 0 < \widehat{\epsilon} \leq \mathtt{a}_{\max}(d)$$

We therefore can construct a connected non-empty interval $\mathbf{I}(a',d) \subseteq \mathtt{r}_{\mathcal{GC}}(a',d)$ for all $d \in [0,1]$, where $a' = \widehat{\mathtt{a}_{\max}}(d,\widehat{\epsilon})$ and $0 \leq \widehat{\epsilon} < \mathtt{a}_{\max}(d)$. ◀

While we have shown the necessary methods to construct an interval of λ values, $\mathbf{I}(a,d)$, that result in $\mathcal{GC}$ being selected, we have yet to guarantee that at all stages of the solving process, the desired LP optimal solution is taken for all $0 \leq d \leq 1$ and $0 \leq a \leq \mathtt{a}_{\max}(d)$. Specifically, we need to show that the originally optimal point is always $(\frac{-1}{2}, 3, \frac{1}{2})$, that after applying $\mathcal{GC}$ the integer solution $(1,1,0)$ is optimal, and that after applying $\mathcal{ISC}^n$ (or $\mathcal{OPC}^n$) a fractional solution from $\mathcal{ISC}^n_{\mathcal{X}}$ (or $\mathcal{OPC}^n_{\mathcal{X}}$) for all $n \in \mathbb{N}$, is optimal.

Lemma 13. *The fractional solution* $(\frac{-1}{2}, 3, \frac{1}{2})$ *is LP optimal for* $\mathsf{P}(a,d)$ *for all* $0 \leq d \leq 1$ *and* $0 \leq a \leq \mathtt{a}_{\max}(d)$.

Proof. This can be done by substituting all points from $\mathcal{X} \setminus (\frac{-1}{2}, 3, \frac{1}{2})$ into the objective, and then showing that the objective is strictly less when evaluated at $(\frac{-1}{2}, 3, \frac{1}{2})$. This shows that for all $0 \leq d \leq 1$ and $0 \leq a \leq \mathtt{a}_{\max}(d)$:

$$\mathsf{P}(a,d)\big|_{\mathbf{x}=(\frac{-1}{2},3,\frac{1}{2})} < \mathsf{P}(a,d)\big|_{\mathbf{x}=\mathbf{x}'} \quad \forall\, \mathbf{x}' \in \mathcal{X} \setminus \left\{ \left(\frac{-1}{2}, 3, \frac{1}{2} \right) \right\}$$ ◀

Lemma 14. *The integer solution* $(1,1,0)$ *is LP optimal after applying* $\mathcal{GC}$ *to* $\mathsf{P}(a,d)$ *for all* $0 \leq d \leq 1$ *and* $0 \leq a \leq \mathtt{a}_{\max}(d)$.

Proof. This can be done in an identical fashion to Lemma 13. That is, we show that for all $0 \leq d \leq 1$ and $0 \leq a \leq \mathtt{a}_{\max}(d)$:

$$\mathsf{P}(a,d)\big|_{\mathbf{x}=(1,1,0)} < \mathsf{P}(a,d)\big|_{\mathbf{x}=\mathbf{x}'} \quad \forall\, \mathbf{x}' \in \mathcal{GC}_{\mathcal{X}} \setminus \{(1,1,0)\}$$ ◀

Lemma 15. *Having applied the cut* $\mathcal{ISC}^n$ *to* $\mathsf{P}(a,d)$*, a point from* $\mathcal{ISC}^n_{\mathcal{X}} \setminus \mathcal{I}_{\mathcal{X}}$ *is LP optimal for all* $0 \leq d \leq 1$ *and* $0 \leq a \leq \mathtt{a}_{\max}(d)$.

Proof. This can be done by showing that for any choice of $a \in [0, \mathtt{a}_{\max}(d)]$ and $d \in [0,1]$, there is at least one point from $\mathcal{ISC}^n_{\mathcal{X}} \setminus \mathcal{I}_{\mathcal{X}}$ at which the objective is strictly less than at all integer points $\mathcal{I}_{\mathcal{X}}$. Specifically, for all $0 \leq d \leq 1$ and $0 \leq a \leq \mathtt{a}_{\max}(d)$:

$$\exists\, \mathbf{x}' \in \mathcal{ISC}^n_{\mathcal{X}} \setminus \mathcal{I}_{\mathcal{X}} \quad s.t \quad \mathsf{P}(a,d)\big|_{\mathbf{x}=\mathbf{x}'} < \mathsf{P}(a,d)\big|_{\mathbf{x}=\mathbf{x}''} \quad \forall\, \mathbf{x}'' \in \mathcal{I}_{\mathcal{X}}$$ ◀

Lemma 16. *Having applied the cut* $\mathcal{OPC}^n$ *to* $\mathsf{P}(a,d)$*, a point from* $\mathcal{OPC}^n_{\mathcal{X}} \setminus \mathcal{I}_{\mathcal{X}}$ *is LP optimal for all* $0 \leq d \leq 1$ *and* $0 \leq a \leq \mathtt{a}_{\max}(d)$.

Proof. This proof follows the same logic as that of Lemma 15. ◀

We can now prove Theorem 1 using the Lemmas 2–16 that we have built up throughout this paper.

Theorem 1. *Given a finite discretisation of λ, an infinite family of MILP instances together with an infinite amount of family-wide valid cuts can be constructed. Using a pure cutting plane approach and applying a single cut per selection round, the infinite family of instances do not solve to optimality for any value in the discretisation, but do solve to optimality for an infinite amount of alternative λ values.*

Proof. From Lemmas 2–5, we know the exact vertex set of our feasible region at each stage of the solving process, as well as the exact set of cuts at each round. Furthermore, as at each round only the RHS value for each proposed cut changes, the scoring of the cuts at each new round remains constant, and we can therefore completely describe the three scenarios of how cuts would be added. Let $\mathcal{S}$ be the set containing all cuts added during the solution process to an instance $\mathsf{P}(a,d)$, where $0 \leq d \leq 1$ and $0 \leq a \leq \mathtt{a}_{\max}(d)$:

$$\mathcal{S} := \begin{cases} \{\mathcal{ISC}^n :\forall\, n \in \mathbb{N}\}, & \text{if } \lambda > \lambda_{ub}(a,d) \\ \{\mathcal{GC}\}, & \text{if } \lambda_{lb}(a,d) \leq \lambda \leq \lambda_{ub}(a,d) \\ \{\mathcal{OPC}^n :\forall\, n \in \mathbb{N}\}, & \text{if } \lambda < \lambda_{lb}(a,d) \end{cases} \tag{35}$$

From Lemma 6 we know the sufficient conditions for a λ value that results in $\mathcal{GC}$ being scored at least as well as the other cuts. Lemmas 7–11 show how these sufficient conditions can be used to construct the region $\mathcal{R}_{\mathcal{GC}}$. Moreover, they show that $\mathcal{R}_{\mathcal{GC}}$ is bounded, and that $a = \mathtt{a}_{\max}(d)$, for all $d \in [0,1]$, is the only time at which the following occurs:

$$\lambda_{lb}(a,d) = \lambda_{ub}(a,d) \quad \forall\, d \in [0,1]$$

We therefore conclude that $\mathcal{R}_{\mathcal{GC}}$ is connected. We know from Lemma 10 that both $\lambda_{ub}(\mathtt{a}_{\max}(d),d)$ and $\lambda_{lb}(\mathtt{a}_{\max}(d),d)$ are continuous, where $d \in [0,1]$, and that $\lambda_{ub}(\mathtt{a}_{\max}(1),1) > \lambda_{ub}(\mathtt{a}_{\max}(0),0)$. From the intermediate value theorem, we then know the following:

$$\forall\, \lambda \in [\lambda_{ub}(\mathtt{a}_{\max}(0),0),\, \lambda_{ub}(\mathtt{a}_{\max}(1),1)], \quad \exists\, d' \; s.t \; \lambda = \lambda_{ub}(\mathtt{a}_{\max}(d'),d') \tag{36}$$

From Lemma 12 we have shown an explicit way to construct an interval $\mathbf{I}(a,d) \subseteq \mathtt{r}_{\mathcal{GC}}(a,d)$ for all $(a,d) \in [0,\mathtt{a}_{\max}(d)) \times [0,1]$. We can therefore construct the following intervals:

$$\mathbf{I}(\widehat{\mathtt{a}_{\max}}(d',\widehat{\epsilon}),d') = [\lambda_{lb}(\widehat{\mathtt{a}_{\max}}(d',\widehat{\epsilon}),d'),\lambda_{ub}(\widehat{\mathtt{a}_{\max}}(d',\widehat{\epsilon}),d')], \text{ where } \quad 0 \leq d' \leq 1, \quad 0 < \widehat{\epsilon} \leq \mathtt{a}_{\max}(d) \tag{37}$$

These intervals can be arbitrarily small as $\widehat{\epsilon}$ can be arbitrarily small. Moreover, as d' values that satisfy (36) can be used, and $\lambda_{lb}(a,d)$, $\lambda_{ub}(a,d)$, and $\widehat{\mathtt{a}_{\max}}(d,\widehat{\epsilon})$ are polynomials, we can generate infinitely many disjoint intervals. We can therefore conclude that for any finite discretisation of λ, Λ, an interval can be created that contains no values from $\{\lambda_1,\ldots,\lambda_{|\Lambda|}\}$, but contains all values of λ for which $\mathsf{P}(a,d)$ solves to optimality.

$$\mathbf{I}(\widehat{\mathtt{a}_{\max}}(d',\widehat{\epsilon}),d') \subset \{[0,\lambda_1) \cup (\lambda_1,\lambda_2) \cup \cdots \cup (\lambda_{n-1},\lambda_n) \cup (\lambda_{|\Lambda|},1]\}, \text{ where } \quad 0 < \widehat{\epsilon} \leq \mathtt{a}_{\max}(d') \tag{38}$$

Finally, Lemmas 13–16 ensure that each stage of the solving process, all cuts are valid for any fractional feasible LP optimal solution for all $\mathsf{P}(a,d)$, where $0 \leq d \leq 1$ and $0 \leq a \leq \mathtt{a}_{\max}(d)$. Moreover, the Lemmas guarantee that only after applying $\mathcal{GC}$ is an integer optimal solution found.

We therefore have shown how fixing a global value of λ to a constant for use in the MILP solving process while disregarding all instance information can result in infinitely worse performance for infinitely many instances. ◀

Corollary 17. *There exists an infinite family of MILP instances together with an infinite amount of family-wide valid cuts, which do not solve to integer optimality for any λ when using a pure cutting plane approach and applying a single cut per selection round.*

Proof. To show this we take the following function, where $0 \leq d \leq 1$:

$$\widetilde{a_{\max}}(d,\widetilde{\epsilon}) := \mathtt{a}_{\max}(d) + \widetilde{\epsilon}, \quad 0 < \widetilde{\epsilon} \leq 0.1$$

Any such value of a retrieved from this function will lie outside of $\mathcal{R}_{\mathcal{GC}}$ for all $0 \leq d \leq 1$. There thus would exist no λ value that results in finite termination, as $\mathcal{GC}$ is never scored at least as high as the other cuts.

Similar to the proof of Theorem 1, we need to ensure that the LP optimal point at all times during the solving process is appropriate, and that the same integer optimal point stays integer optimal for all $0 \leq d \leq 1$ and $\mathtt{a}_{\max}(d) < a \leq \widetilde{a_{\max}}(d,\widetilde{\epsilon})$. We therefore redo the proofs of Lemmas 13–16 but change the range of values of a. ◀

B Functions of Appendix A

$$\mathsf{a}_{\max}(d) := \frac{-2680\sqrt{101}d + 2020\sqrt{201}d - 6767\sqrt{2} - 27068\sqrt{101} + 22220\sqrt{201}}{6767\sqrt{2} - 202\sqrt{201}}$$

$$\lambda_{lb}(a,d)_1 := \sqrt{20301}\sqrt{(a^2 + d(d+20) + 101)}$$

$$\lambda_{lb}(a,d)_2 := 101a^2 + a(-20(\sqrt{20301} - 101)d - 202(\sqrt{20301} - 110))$$

$$\lambda_{lb}(a,d)_3 := 20d(-10(\sqrt{20301} - 151)d - 211\sqrt{20301} + 31411) - 22220\sqrt{20301} + 3272501$$

$$\lambda_{lb}(a,d)_4 := -606a^2 + 12a(10(\sqrt{20301} - 101)d + 101(\sqrt{20301} - 110))$$

$$\lambda_{lb}(a,d)_5 := 120d(10(\sqrt{20301} - 151)d + 211\sqrt{20301} - 31411) + 606(220\sqrt{20301} - 32401)$$

$$\lambda_{ub}(a,d)_6 := 5555a^2 + 24a(10(\sqrt{20301} - 101)d + 101(\sqrt{20301} - 110))$$

$$\lambda_{lb}(a,d)_7 := d((2400\sqrt{20301} - 355633)d + 50640\sqrt{20301} - 7403300) + 505(528\sqrt{20301} - 76409)$$

$$\lambda_{lb}(a,d) := \frac{2(\lambda_{lb}(a,d)_1\sqrt{\lambda_{lb}(a,d)_2 + \lambda_{lb}(a,d)_3} + \lambda_{lb}(a,d)_4 + \lambda_{lb}(a,d)_5)}{\lambda_{lb}(a,d)_6 + \lambda_{lb}(a,d)_7}$$

$$\lambda_{ub}(a,d)_1 := -(a^2 + d(d+20) + 101)$$

$$\lambda_{ub}(a,d)_2 := (2\sqrt{402} - 203)a^2 + a(20(\sqrt{402} - 2)d + 222\sqrt{402} - 842) + 20d(-10d + \sqrt{402} - 220) + 220\sqrt{402} - 24401$$

$$\lambda_{ub}(a,d)_3 := (6\sqrt{402} - 609)a^2 + 6a(10(\sqrt{402} - 2)d + 111\sqrt{402} - 421) + 60d(-10d + \sqrt{402} - 220) + 660\sqrt{402} - 73203$$

$$\lambda_{ub}(a,d)_4 := (6\sqrt{402} - 475)a^2 + 6a(10(\sqrt{402} - 2)d + 111\sqrt{402} - 421) + 2d(-233d + 30\sqrt{402} - 5260) + 660\sqrt{402} - 59669$$

$$\lambda_{ub}(a,d) := \frac{\sqrt{402}\sqrt{\lambda_{ub}(a,d)_1\lambda_{ub}(a,d)_2} + \lambda_{ub}(a,d)_3}{\lambda_{ub}(a,d)_4}$$

C A Guide to a Forward Pass of the GCNN

This section should be used to provide an intuitive understanding of our policy network, which is parameterised as a GCNN. For a more complete introduction to graph neural networks that also provides helpful visualisations, we refer readers to [26]. Throughout this section we will also refer to multi-layer perceptrons, which from now we simply refer to as (feed-forward) neural networks. A neural network is a function, which passes its input through a series of alternating linear transformations and non-linear activation functions. Note that while our design makes use of standard neural networks, other architecture types can be used. We refer readers to [17] for a thorough overview.

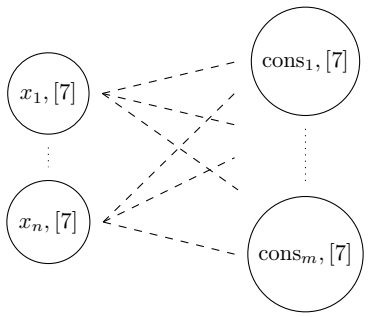

Figure 10 A visualisation the initial state s_0. At each node the size of the corresponding feature vector is given, e.g. [7]. Note that the edges additionally have features, but do not appear for ease of visualisation.

Recall that we present our MILP instance via a constraint-variable bipartite graph, and that a variable and a constraint share an edge when the variable appears in the constraint with a non-zero coefficient. See Figure 10

for an initial representation of $s_0 \in \mathcal{G}$. Recall also that the goal of the GCNN is to parameterise our policy, $\pi_\theta(\cdot|s_0 \in \mathcal{G})$, outputting the mean, $\mu \in \mathbb{R}^4$, of a distribution, $\mathcal{N}_4(\mu, \gamma I)$, over the cut selector parameter space.

We will now begin the forward pass of the GCNN. Consider a node of the bipartite graph that represents the variable x_i of the MILP. This node has an attached set of features, see Table 1 for a complete list, which form a vector. This feature vector gets transformed by a neural network. In our design this initially transforms our 7-dimensional feature vector to a 32-dimensional vector. This operation gets applied to all feature vectors representing variables, using the same neural network. The new bipartite graph is denoted $\mathbf{H}^1_{\mathbf{V}}$. We also do this procedure for all feature vectors corresponding to a constraint of the MILP, albeit with a different neural network, which results in $\mathbf{H}^1_{\mathbf{C}}$. The result is visualised in Figure 11. We note that there is no order to the two transformations.

Figure 11 (Left) $\mathbf{H}^1_{\mathbf{V}}$. (Right) $\mathbf{H}^1_{\mathbf{C}}$

The key to graph neural networks, for example the GCNN, is using the same neural network to transform multiple feature vectors, e.g. all constraint feature vectors. This allows the GCNN to take arbitrarily sized bipartite graphs as input, and therefore work on any MILP instance.

Until this point, no information has been shared between any two variables or constraints. In our design information is gathered per node from its neighbours by summing the transformed feature vectors of all incident edges and adjacent nodes. Note that due to the bipartite nature of our graph information either flows from variables to constraints, or vice-versa. This gathering of information for either all variable nodes or all constraint nodes is called a *half-convolution*, or alternatively message passing. To perform this half-convolution we require the transformed feature vectors to all have the same dimension, with 32 being our choice. Note that the feature vectors on the edges are also transformed during the half-convolution. For a feature vector representing a variable, the half-convolution is defined as:

$$\mathbf{v}''_i := \mathtt{nn}'\left(\sum_{j \in N(x_i)} (\mathtt{nn}(\mathbf{v}'_i + \mathbf{e}'_{(i,j)} + \mathbf{c}'_j)), \mathbf{v}'_i\right)$$

Here, $\mathbf{v}''_i$, $\mathbf{v}'_i$, $\mathbf{c}'_j$, and $\mathbf{e}'_{(i,j)}$ are transformed feature vectors of variable x_i, constraint j, and edge (i,j). The functions $\mathtt{nn}$ and $\mathtt{nn}'$ are neural networks, with $\mathtt{nn}'$ taking a concatenated input, and $N(x_i)$ is the neighbourhood of the variable node of x_i. For a feature vector representing a constraint, the half-convolution is defined in a mirrored manner. See Figure 12 for $\mathbf{H}^2_{\mathbf{C}}$ and $\mathbf{H}^2_{\mathbf{V}}$.

Figure 12 (Left) $\mathbf{H}^2_{\mathbf{C}}$. (Right) $\mathbf{H}^2_{\mathbf{V}}$

As our design will ultimately extract μ from the transformed variable feature vectors, we first perform the half-convolution over all constraints, and then over all variables. This guarantees that at the representation $\mathbf{H}^2_{\mathbf{V}}$, all variables that feature together in a constraint have shared and received information. In a similar manner to

the beginning of our forward pass, we now reduce all resulting variable feature vectors using a neural network to dimension 4, i.e. the amount of cut selector parameters. The result $\mathbf{H}_{\mathbf{V}}^{3}$ is shown in Figure 13. Finally, we average over all reduced variable feature vectors, resulting in the mean $\mu \in \mathbb{R}^4$ of our policy. This is a forward pass of the GCNN.

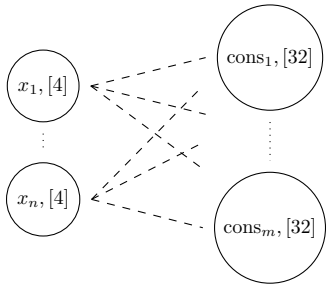

Figure 13 $\mathbf{H}_{\mathbf{V}}^{3}$.

We note that the decision to average the reduced variable feature vectors was inspired by the original design for branching, see [15]. For instance, it would be possible to change the order of the half-convolutions and extract μ from the transformed constraint feature vectors. This is just one way to change the specific design, with other examples being the type of activation functions, layer structure, and the dimension of each embedding. Our design also makes heavy use of layer normalisation, see [5], which followed from observations in [15] on improved generalisation capabilities. For our complete design we refer readers to `https://github.com/Opt-Mucca/Adaptive-Cutsel-MILP`.

D Per Instance Results and Statistics of Section 6

Table 8 Inference time statistics of trained GCNN from Experiment 6.4 per instance-seed pair.

	MIPLIB			NN-Verification		
	Min	Median	Max	Min	Median	Max
Time (s)	0.001	0.037	0.944	0.032	0.087	1.378

Table 9 Instance statistics for Experiments 6.1, 6.2, 6.3, 6.4, and 6.5.

	MIPLIB			NN-Verification		
Variable Information	Range	Mean	Median	Range	Mean	Median
Num. vars	[154, 23618]	3251.33	1909	[987, 2101]	1647.92	1653
Num. bin. vars	[0, 16360],	1498.03	396	[107, 312]	183.84	183
Num. int. vars	[0, 5081]	171.70	0	[0, 0]	0	0
Num. impl. int. vars	[0, 1]	0.01	0	[0, 0]	0	0
Num. cont. vars	[0, 23520]	1581.59	393	[869, 1856]	1464.08	1473
Constraint Information	Range	Mean	Median	Range	Mean	Median
Num. cons	[32, 17366]	2261.62	865	[923, 1933]	1444.43	1452
Num. linear cons	[0, 13263]	1181.14	306	[813, 1629]	1263.82	1276
Num. logicor cons	[0, 17323]	225.29	0	[0, 0]	0	0
Num. knapsack cons	[0, 288]	7.48	0	[0, 0]	0	0
Num. setppc cons	[0, 11538]	284.04	0	[0, 0]	0	0
Num. varbound cons	[0, 5000]	563.67	17	[106, 304]	180.61	180

Table 10 Per instance results of Experiment 6.1 (Grid search). Primal-dual refers to the relative primal-dual difference improvement. $\{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}$ are the multipliers for `dcd`, `eff`, `isp`, and `obp`. #BP refers to the number of best parameter combinations. In the case of #BP > 1, a single best choice $\{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}$ is provided.

Instance	primal-dual	λ_1	λ_2	λ_3	λ_4	#BP
22433	0.22	0.0	0.1	0.7	0.2	1
23588	0.02	0.2	0.5	0.1	0.2	2
50v-10	0.04	0.1	0.0	0.4	0.5	1
a1c1s1	0.25	0.6	0.1	0.3	0.0	1
a2c1s1	0.36	0.6	0.1	0.3	0.0	1
app3	0.28	0.6	0.1	0.2	0.1	1
b1c1s1	0.38	0.2	0.2	0.3	0.3	1
b2c1s1	0.33	0.1	0.2	0.3	0.4	1
beasleyC1	0.17	0.2	0.1	0.0	0.7	1
beasleyC2	0.04	0.0	0.5	0.0	0.5	1
berlin	0.09	0.2	0.6	0.0	0.2	1
berlin_5_8_0	0.2	0.0	0.4	0.3	0.3	1
bg512142	0.02	0.0	0.8	0.0	0.2	1
bienst1	0.29	0.2	0.0	0.6	0.2	1
bienst2	0.21	0.2	0.0	0.7	0.1	1
bppc8-09	0.03	0.7	0.3	0.0	0.0	1
brasil	0.05	0.5	0.0	0.2	0.3	1
dg012142	0.0	0.1	0.0	0.0	0.9	1
dws008-01	0.03	0.2	0.7	0.1	0.0	1
eil33-2	0.01	0.1	0.5	0.4	0.0	1
exp-1-500-5-5	0.69	0.3	0.0	0.3	0.4	1
fhnw-schedule-paira100	0.01	0.0	0.2	0.7	0.1	1
g200x740	0.64	0.4	0.1	0.1	0.4	1
glass4	0.01	0.5	0.2	0.1	0.2	1
gmu-35-40	0.01	0.4	0.0	0.0	0.6	1
graphdraw-domain	0.01	0.2	0.0	0.8	0.0	1
graphdraw-gemcutter	0.01	0.6	0.1	0.3	0.0	1
h50x2450	0.54	0.0	0.0	0.0	1.0	1
hgms-det	0.29	0.5	0.3	0.0	0.2	1
ic97_potential	0.01	0.8	0.0	0.1	0.1	1
ic97_tension	0.09	0.2	0.3	0.5	0.0	1
icir97_tension	0.14	0.2	0.1	0.0	0.7	1
k16x240b	0.12	0.5	0.3	0.0	0.2	1
lotsize	0.15	0.3	0.1	0.5	0.1	1
mik-250-20-75-2	0.11	0.3	0.4	0.0	0.3	1
mik-250-20-75-3	0.09	0.3	0.1	0.0	0.6	1
mik-250-20-75-5	0.16	0.2	0.1	0.7	0.0	1
milo-v12-6-r2-40-1	0.33	0.9	0.0	0.1	0.0	1
milo-v13-4-3d-4-0	0.01	0.7	0.2	0.0	0.1	1
misc07	0.06	0.8	0.0	0.1	0.1	1
mkc	0.28	0.4	0.3	0.2	0.1	1
n3700	0.08	0.1	0.0	0.2	0.7	1
n3707	0.09	0.2	0.0	0.8	0.0	1
n370b	0.08	0.3	0.1	0.6	0.0	1
n5-3	0.2	0.1	0.4	0.4	0.1	1
n7-3	0.09	0.6	0.0	0.4	0.0	1
n9-3	0.16	0.1	0.2	0.3	0.4	1
neos-1423785	0.04	0.2	0.0	0.0	0.8	1
neos-1445738	0.0	0.2	0.4	0.2	0.2	1
neos-1456979	0.21	0.5	0.0	0.0	0.5	1
neos-3046601-motu	0.01	0.2	0.4	0.1	0.3	1
neos-3046615-murg	0.01	0.6	0.2	0.1	0.1	1
neos-3381206-awhea	0.02	0.1	0.0	0.6	0.3	12
neos-3426085-ticino	0.0	0.2	0.1	0.0	0.7	4
neos-3530905-gaula	0.01	0.0	0.1	0.0	0.9	14
neos-3627168-kasai	0.12	0.2	0.0	0.0	0.8	1
neos-4333464-siret	0.0	0.0	0.4	0.3	0.3	1
neos-4387871-tavua	0.02	0.2	0.6	0.0	0.2	1
neos-4650160-yukon	0.02	0.1	0.1	0.0	0.8	1
neos-4736745-arroux	0.07	0.3	0.1	0.1	0.5	1
neos-4954672-berkel	0.25	0.9	0.0	0.0	0.1	1
neos-5076235-embley	0.18	0.6	0.0	0.0	0.4	4
neos-5107597-kakapo	0.01	0.0	0.4	0.0	0.6	1
neos-5260764-orauea	0.02	0.5	0.1	0.4	0.0	1
neos-5261882-treska	0.09	0.4	0.1	0.4	0.1	1
neos-631517	0.07	0.7	0.1	0.1	0.1	1
neos-691058	0.64	0.3	0.0	0.2	0.5	1
neos-860300	0.1	0.1	0.4	0.4	0.1	1
neos16	0.0	0.0	0.1	0.1	0.8	3
newdano	0.05	0.1	0.1	0.5	0.3	1
nexp-150-20-1-5	0.28	0.1	0.0	0.1	0.8	3
nexp-150-20-8-5	0.86	0.3	0.0	0.3	0.4	1
p200x1188c	0.01	0.0	0.1	0.8	0.1	1
p500x2988	0.11	0.6	0.2	0.1	0.1	1
pg	0.46	0.4	0.1	0.2	0.3	1
pg5_34	0.34	1.0	0.0	0.0	0.0	1
probportfolio	0.01	0.2	0.2	0.1	0.5	1
prod2	0.02	0.3	0.1	0.1	0.5	1
r50x360	0.09	0.2	0.2	0.6	0.0	1
ran13x13	0.11	0.0	0.4	0.0	0.6	1
supportcase20	0.36	0.7	0.2	0.0	0.1	1
supportcase26	0.03	0.7	0.1	0.2	0.0	1
swath	0.01	0.0	0.4	0.3	0.3	1
tanglegram6	0.0	0.1	0.1	0.2	0.6	1
timtab1CUTS	0.04	0.0	0.2	0.6	0.2	1
tr12-30	0.73	0.6	0.1	0.2	0.1	1
usAbbrv-8-25_70	0.72	0.2	0.3	0.3	0.2	1

References

1 Tobias Achterberg. *Constraint integer programming*. PhD thesis, TU Berlin, 2007.

2 Tobias Achterberg, Robert E. Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. *INFORMS J. Comput.*, 32(2):473–506, 2020.

3 Tobias Achterberg and Roland Wunderling. Mixed integer programming: Analyzing 12 years of progress. In *Facets of combinatorial optimization*, pages 449–481. Springer, 2013.

4 Giuseppe Andreello, Alberto Caprara, and Matteo Fischetti. Embedding {0, 1/2}-Cuts in a Branch-and-Cut Framework: A Computational Study. *INFORMS J. Comput.*, 19(2):229–238, 2007.

5 Jimmy L. Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. `https://arxiv.org/abs/1607.06450`, 2016.

6 Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In *International Conference on Machine Learning*, pages 344–353. PMLR, 2018.

7 Maria-Florina Balcan, Siddharth Prasad, Tuomas Sandholm, and Ellen Vitercik. Sample complexity of tree search configuration: Cutting planes and beyond. *Adv. Neural Inf. Process. Syst.*, 34, 2021.

8 Radu Baltean-Lugojan, Pierre Bonami, Ruth Misener, and Andrea Tramontani. Scoring positive semidefinite cutting planes for quadratic optimization via trained neural networks. `https://optimization-online.org/2018/11/6943/`, 2019.

9 Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, Leona Gottwald, Christoph Graczyk, Katrin Halbig, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Thorsten Koch, Marco Lübbecke, Stephen Maher, Frederic Matter, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Daniel Rehfeldt, Steffan Schlein, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Boro Sofranac, Mark Turner, Stefan Vigerske, Fabian Wegscheider, Philipp Wellner, Dieter Weninger, and Jakob Witzig. Enabling research through the SCIP optimization suite 8.0. *ACM Trans. Math. Softw.*, 49(2):1–21, 2023.

10 Quentin Cappart, Didier Chételat, Elias Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. `https://arxiv.org/abs/2102.09544`, 2021.

Table 11 Per instance results of Experiment 6.2 (Random seed). Primal-dual refers to the relative primal-dual difference improvement. $\{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}$ are the multipliers for `dcd`, `eff`, `isp`, and `obp`.

Instance	primal-dual	λ_1	λ_2	λ_3	λ_4
22433	0.12	0.24	0.27	0.37	0.11
23588	0.01	0.25	0.28	0.36	0.12
50v-10	0.0	0.26	0.3	0.3	0.15
a1c1s1	0.18	0.26	0.3	0.29	0.15
a2c1s1	0.25	0.26	0.3	0.29	0.15
app3	0.09	0.26	0.3	0.29	0.15
b1c1s1	0.28	0.26	0.3	0.29	0.15
b2c1s1	0.22	0.26	0.3	0.29	0.15
beasleyC1	0.02	0.25	0.3	0.29	0.15
beasleyC2	−0.03	0.25	0.3	0.29	0.15
berlin	−0.31	0.25	0.3	0.29	0.15
berlin_5_8_0	0.0	0.25	0.28	0.33	0.13
bg512142	0.01	0.26	0.3	0.3	0.15
bienst1	−0.05	0.26	0.3	0.29	0.15
bienst2	0.01	0.26	0.3	0.29	0.15
bppc8-09	−0.03	0.25	0.29	0.32	0.13
brasil	−1.0	0.25	0.3	0.29	0.15
dg012142	0.0	0.26	0.3	0.29	0.15
dws008-01	−0.01	0.26	0.3	0.3	0.14
eil33-2	0.0	0.26	0.3	0.3	0.15
exp-1-500-5-5	0.35	0.25	0.31	0.29	0.15
fhnw-schedule-paira100	0.01	0.27	0.27	0.3	0.16
g200x740	0.6	0.25	0.3	0.3	0.15
glass4	0.0	0.25	0.28	0.32	0.14
gmu-35-40	0.0	0.25	0.29	0.32	0.14
graphdraw-domain	0.0	0.24	0.29	0.36	0.12
graphdraw-gemcutter	0.0	0.24	0.29	0.35	0.12
h50x2450	0.32	0.25	0.3	0.3	0.15
hgms-det	0.09	0.25	0.29	0.31	0.14
ic97_potential	0.0	0.26	0.29	0.3	0.15
ic97_tension	0.08	0.26	0.3	0.3	0.15
icir97_tension	0.05	0.25	0.28	0.31	0.16
k16x240b	0.04	0.25	0.3	0.3	0.15
lotsize	0.06	0.26	0.3	0.3	0.15
mik-250-20-75-2	−0.08	0.25	0.29	0.3	0.16
mik-250-20-75-3	−0.01	0.25	0.29	0.3	0.16
mik-250-20-75-5	0.05	0.25	0.29	0.3	0.16
milo-v12-6-r2-40-1	0.23	0.26	0.3	0.3	0.15
milo-v13-4-3d-4-0	0.01	0.25	0.31	0.3	0.14
misc07	0.0	0.26	0.28	0.33	0.13
mkc	0.04	0.26	0.29	0.3	0.15
n3700	0.07	0.26	0.29	0.3	0.15
n3707	0.08	0.26	0.29	0.3	0.15
n370b	0.08	0.26	0.29	0.3	0.15
n5-3	0.0	0.25	0.32	0.28	0.15
n7-3	0.05	0.25	0.32	0.28	0.15
n9-3	0.13	0.25	0.32	0.28	0.15
neos-1423785	0.0	0.26	0.3	0.29	0.15
neos-1445738	0.0	0.25	0.32	0.28	0.15
neos-1456979	0.04	0.25	0.29	0.32	0.14
neos-3046601-motu	0.0	0.26	0.28	0.31	0.15
neos-3046615-murg	0.0	0.25	0.28	0.31	0.15
neos-3381206-awhea	0.0	0.24	0.28	0.31	0.17
neos-3426085-ticino	−0.01	0.24	0.28	0.32	0.16
neos-3530905-gaula	−0.01	0.24	0.28	0.31	0.17
neos-3627168-kasai	0.05	0.25	0.31	0.3	0.15
neos-4333464-siret	−0.03	0.24	0.3	0.32	0.13
neos-4387871-tavua	−0.02	0.25	0.3	0.32	0.14
neos-4650160-yukon	0.01	0.25	0.3	0.3	0.15
neos-4736745-arroux	−0.06	0.24	0.28	0.32	0.16
neos-4954672-berkel	0.14	0.26	0.3	0.29	0.15
neos-5076235-embley	−0.07	0.25	0.31	0.29	0.15
neos-5107597-kakapo	0.0	0.27	0.27	0.31	0.16
neos-5260764-orauea	0.01	0.26	0.29	0.31	0.14
neos-5261882-treska	0.06	0.25	0.29	0.32	0.14
neos-631517	0.05	0.25	0.28	0.33	0.14
neos-691058	0.34	0.25	0.29	0.31	0.14
neos-860300	−0.05	0.26	0.28	0.33	0.13
neos16	0.0	0.25	0.27	0.33	0.15
newdano	0.02	0.26	0.3	0.3	0.15
nexp-150-20-1-5	0.22	0.25	0.3	0.3	0.15
nexp-150-20-8-5	0.67	0.25	0.29	0.32	0.14
p200x1188c	0.0	0.25	0.3	0.29	0.15
p500x2988	−0.02	0.25	0.3	0.29	0.15
pg	0.16	0.26	0.3	0.3	0.14
pg5_34	0.12	0.25	0.32	0.28	0.15
probportfolio	0.0	0.27	0.27	0.29	0.16
prod2	0.0	0.26	0.3	0.3	0.14
r50x360	0.05	0.25	0.3	0.3	0.15
ran13x13	−0.01	0.26	0.29	0.3	0.15
supportcase20	0.19	0.26	0.3	0.29	0.15
supportcase26	0.01	0.26	0.27	0.3	0.16
swath	0.0	0.26	0.29	0.31	0.14
tanglegram6	0.0	0.25	0.28	0.35	0.11
timtab1CUTS	0.0	0.25	0.3	0.3	0.15
tr12-30	0.62	0.26	0.3	0.3	0.15
usAbbrv-8-25_70	0.47	0.25	0.28	0.33	0.14

11 Santanu S. Dey and Marco Molinaro. Theoretical challenges towards cutting-plane selection. *Math. Program.*, 170(1):237–266, 2018.

12 Jian-Ya Ding, Chao Zhang, Lei Shen, Shengyin Li, Bing Wang, Yinghui Xu, and Le Song. Accelerating primal solution findings for mixed integer programs based on solution prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1452–1459, 2020.

13 Matthias Fey and Jan E. Lenssen. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

14 Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald, Katrin Halbig, Gregor Hendel, Christopher Hojny, Thorsten Koch, Pierre Le Bodic, Stephen Maher, Frederic Matter, Matthias Miltenberger, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Christine Tawfik, Stefan Vigerske, Fabian Wegscheider, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 7.0. ZIB-Report 20-10, Zuse Institute Berlin, 2020.

15 Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. `https://arxiv.org/abs/1906.01629`, 2019.

16 Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, et al. MIPLIB 2017: data-driven compilation of the 6th mixed-integer programming library. *Math. Program. Comput.*, 13(3):443–490, 2021.

17 Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

18 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021. `https://www.gurobi.com`.

19 Zeren Huang, Kerong Wang, Furui Liu, Hui-ling Zhen, Weinan Zhang, Mingxuan Yuan, Jianye Hao, Yong Yu, and Jun Wang. Learning to Select Cuts for Efficient Mixed-Integer Programming. `https://arxiv.org/abs/2105.13645`, 2021.

20 Diederik P. Kingma and Jimmy L. Ba. Adam: A method for stochastic optimization. `https://arxiv.org/abs/1412.6980`, 2014.

21 Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. Smac3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *J. Mach. Learn. Res.*, 23(54):1–9, 2022.

22 Stephen Maher, Matthias Miltenberger, Joao Pedro Pedroso, Daniel Rehfeldt, Robert Schwarz, and Felipe Serrano.

Table 12 Per instance results of Experiment 6.3 (Standard Learning Method - SMAC). Primal-dual refers to the relative primal-dual difference improvement. $\{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}$ are the multipliers for `dcd`, `eff`, `isp`, and `obp`.

l@ S@ S@ S@ S@ S Instance	primal-dual	λ_1	λ_2	λ_3	λ_4
22433	0.03	0.6	0.12	0.18	0.1
23588	0.0	0.6	0.12	0.18	0.1
50v-10	−0.01	0.6	0.12	0.18	0.1
a1c1s1	0.2	0.6	0.12	0.18	0.1
a2c1s1	0.28	0.6	0.12	0.18	0.1
app3	0.15	0.6	0.12	0.18	0.1
b1c1s1	0.28	0.6	0.12	0.18	0.1
b2c1s1	0.27	0.6	0.12	0.18	0.1
beasleyC1	0.06	0.6	0.12	0.18	0.1
beasleyC2	−0.04	0.6	0.12	0.18	0.1
berlin	0.03	0.6	0.12	0.18	0.1
berlin_5_8_0	0.0	0.6	0.12	0.18	0.1
bg512142	0.0	0.6	0.12	0.18	0.1
bienst1	0.11	0.6	0.12	0.18	0.1
bienst2	0.03	0.6	0.12	0.18	0.1
bppc8-09	−0.01	0.6	0.12	0.18	0.1
brasil	0.02	0.6	0.12	0.18	0.1
dg012142	0.0	0.6	0.12	0.18	0.1
dws008-01	−0.02	0.6	0.12	0.18	0.1
eil33-2	0.0	0.6	0.12	0.18	0.1
exp-1-500-5-5	0.58	0.6	0.12	0.18	0.1
fhnw-schedule-paira100	0.0	0.6	0.12	0.18	0.1
g200x740	0.58	0.6	0.12	0.18	0.1
glass4	−0.01	0.6	0.12	0.18	0.1
gmu-35-40	0.0	0.6	0.12	0.18	0.1
graphdraw-domain	0.0	0.6	0.12	0.18	0.1
graphdraw-gemcutter	0.0	0.6	0.12	0.18	0.1
h50x2450	0.32	0.6	0.12	0.18	0.1
hgms-det	0.06	0.6	0.12	0.18	0.1
ic97_potential	0.0	0.6	0.12	0.18	0.1
ic97_tension	0.08	0.6	0.12	0.18	0.1
icir97_tension	0.04	0.6	0.12	0.18	0.1
k16x240b	0.08	0.6	0.12	0.18	0.1
lotsize	0.09	0.6	0.12	0.18	0.1
mik-250-20-75-2	0.05	0.6	0.12	0.18	0.1
mik-250-20-75-3	−0.03	0.6	0.12	0.18	0.1
mik-250-20-75-5	0.13	0.6	0.12	0.18	0.1
milo-v12-6-r2-40-1	0.28	0.6	0.12	0.18	0.1
milo-v13-4-3d-4-0	0.01	0.6	0.12	0.18	0.1
misc07	0.0	0.6	0.12	0.18	0.1
mkc	0.24	0.6	0.12	0.18	0.1
n3700	0.07	0.6	0.12	0.18	0.1
n3707	0.07	0.6	0.12	0.18	0.1
n370b	0.07	0.6	0.12	0.18	0.1
n5-3	0.14	0.6	0.12	0.18	0.1
n7-3	0.05	0.6	0.12	0.18	0.1
n9-3	0.11	0.6	0.12	0.18	0.1
neos-1423785	−0.01	0.6	0.12	0.18	0.1
neos-1445738	0.0	0.6	0.12	0.18	0.1
neos-1456979	0.1	0.6	0.12	0.18	0.1
neos-3046601-motu	0.0	0.6	0.12	0.18	0.1
neos-3046615-murg	0.0	0.6	0.12	0.18	0.1
neos-3381206-awhea	0.0	0.6	0.12	0.18	0.1
neos-3426085-ticino	−0.01	0.6	0.12	0.18	0.1
neos-3530905-gaula	−0.01	0.6	0.12	0.18	0.1
neos-3627168-kasai	0.06	0.6	0.12	0.18	0.1
neos-4333464-siret	−0.12	0.6	0.12	0.18	0.1
neos-4387871-tavua	0.0	0.6	0.12	0.18	0.1
neos-4650160-yukon	0.0	0.6	0.12	0.18	0.1
neos-4736745-arroux	−0.04	0.6	0.12	0.18	0.1
neos-4954672-berkel	0.19	0.6	0.12	0.18	0.1
neos-5076235-embley	0.13	0.6	0.12	0.18	0.1
neos-5107597-kakapo	0.0	0.6	0.12	0.18	0.1
neos-5260764-orauea	0.01	0.6	0.12	0.18	0.1
neos-5261882-treska	0.05	0.6	0.12	0.18	0.1
neos-631517	0.06	0.6	0.12	0.18	0.1
neos-691058	0.41	0.6	0.12	0.18	0.1
neos-860300	−0.02	0.6	0.12	0.18	0.1
neos16	0.0	0.6	0.12	0.18	0.1
newdano	−0.03	0.6	0.12	0.18	0.1
nexp-150-20-1-5	0.27	0.6	0.12	0.18	0.1
nexp-150-20-8-5	0.75	0.6	0.12	0.18	0.1
p200x1188c	0.0	0.6	0.12	0.18	0.1
p500x2988	0.0	0.6	0.12	0.18	0.1
pg	0.17	0.6	0.12	0.18	0.1
pg5_34	0.16	0.6	0.12	0.18	0.1
probportfolio	0.01	0.6	0.12	0.18	0.1
prod2	−0.02	0.6	0.12	0.18	0.1
r50x360	0.06	0.6	0.12	0.18	0.1
ran13x13	0.01	0.6	0.12	0.18	0.1
supportcase20	0.26	0.6	0.12	0.18	0.1
supportcase26	0.02	0.6	0.12	0.18	0.1
swath	0.0	0.6	0.12	0.18	0.1
tanglegram6	0.0	0.6	0.12	0.18	0.1
timtab1CUTS	0.0	0.6	0.12	0.18	0.1
tr12-30	0.73	0.6	0.12	0.18	0.1
usAbbrv-8-25_70	0.72	0.6	0.12	0.18	0.1

PySCIPOpt: Mathematical programming in python with the SCIP optimization suite. In *International Congress on Mathematical Software*, pages 301–307. Springer, 2016.

23 Hugues Marchand, Alexander Martin, Robert Weismantel, and Laurence Wolsey. Cutting planes in integer and mixed integer programming. *Discrete Appl. Math.*, 123(1-3):397–446, 2002.

24 Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, et al. Solving mixed integer programs using neural networks. `https://arxiv.org/abs/2012.13349`, 2020.

25 Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. 2019.

26 Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B. Wiltschko. A gentle introduction to graph neural networks. *Distill*, 6(9), 2021.

27 Zachary Steever, Chase Murray, Junsong Yuan, Mark Karwan, and Marco Lübbecke. An Image-based Approach to Detecting Structural Similarity Among Mixed Integer Programs. *INFORMS J. Comput.*, 34(4):1849–1870, 2022.

28 Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT Press, 2018.

29 Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In *International Conference on Machine Learning*, pages 9367–9376. PMLR, 2020.

30 Franz Wesselmann and Uwe Stuhl. Implementing cutting plane management and selection techniques. Technical report, Technical report, University of Paderborn, 2012.

31 Wolfram Research, Inc. Mathematica, Version 12.2, 2020. `https://www.wolfram.com/mathematica`.

Table 13 Per instance results of Experiment 6.4 (Learning Adaptive Parameters). Primal-dual refers to the relative primal-dual difference improvement. $\{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}$ are the multipliers for `dcd`, `eff`, `isp`, and `obp`.

Instance	primal-dual	λ_1	λ_2	λ_3	λ_4
22433	0.12	0.3	0.21	0.27	0.21
23588	0.01	0.31	0.21	0.27	0.22
50v-10	0.01	0.39	0.16	0.23	0.22
a1c1s1	0.22	0.37	0.12	0.27	0.24
a2c1s1	0.34	0.37	0.12	0.27	0.24
app3	0.18	0.37	0.13	0.26	0.24
b1c1s1	0.31	0.37	0.12	0.27	0.24
b2c1s1	0.29	0.37	0.12	0.27	0.24
beasleyC1	0.03	0.6	0.08	0.24	0.07
beasleyC2	−0.08	0.59	0.09	0.25	0.07
berlin	0.03	0.58	0.1	0.24	0.08
berlin_5_8_0	0.0	0.32	0.16	0.3	0.22
bg512142	0.0	0.36	0.13	0.27	0.23
bienst1	−0.03	0.36	0.13	0.26	0.25
bienst2	0.04	0.37	0.13	0.26	0.24
bppc8-09	−0.01	0.39	0.19	0.21	0.21
brasil	0.02	0.58	0.1	0.24	0.08
dg012142	0.0	0.36	0.13	0.27	0.23
dws008-01	−0.01	0.38	0.17	0.23	0.22
eil33-2	0.0	0.61	0.39	0.0	0.0
exp-1-500-5-5	0.62	0.39	0.12	0.27	0.21
fhnw-schedule-paira100	0.01	0.35	0.16	0.26	0.24
g200x740	0.56	0.55	0.11	0.24	0.1
glass4	−0.01	0.37	0.2	0.23	0.2
gmu-35-40	0.01	0.39	0.22	0.19	0.19
graphdraw-domain	0.0	0.32	0.19	0.28	0.21
graphdraw-gemcutter	0.01	0.32	0.19	0.28	0.21
h50x2450	0.4	0.43	0.15	0.23	0.19
hgms-det	0.18	0.36	0.18	0.24	0.23
ic97_potential	0.0	0.37	0.14	0.26	0.23
ic97_tension	0.08	0.34	0.16	0.26	0.24
icir97_tension	0.08	0.3	0.19	0.26	0.25
k16x240b	0.03	0.52	0.11	0.25	0.11
lotsize	0.09	0.36	0.15	0.25	0.23
mik-250-20-75-2	−0.02	0.33	0.22	0.2	0.25
mik-250-20-75-3	0.02	0.33	0.22	0.2	0.25
mik-250-20-75-5	0.01	0.33	0.22	0.2	0.25
milo-v12-6-r2-40-1	0.23	0.37	0.14	0.26	0.23
milo-v13-4-3d-4-0	0.0	0.39	0.12	0.27	0.22
misc07	0.02	0.57	0.32	0.1	0.01
mkc	0.18	0.39	0.21	0.19	0.2
n3700	0.07	0.41	0.16	0.23	0.21
n3707	0.08	0.41	0.16	0.23	0.2
n370b	0.08	0.41	0.16	0.23	0.2
n5-3	0.07	0.43	0.13	0.24	0.19
n7-3	0.08	0.44	0.13	0.25	0.19
n9-3	0.07	0.44	0.13	0.24	0.19
neos-1423785	−0.01	0.37	0.12	0.27	0.24
neos-1445738	0.0	0.46	0.13	0.23	0.18
neos-1456979	0.1	0.36	0.2	0.23	0.22
neos-3046601-motu	0.0	0.44	0.24	0.16	0.16
neos-3046615-murg	0.0	0.44	0.24	0.16	0.16
neos-3381206-awhea	0.0	0.28	0.23	0.23	0.25
neos-3426085-ticino	0.0	0.28	0.23	0.25	0.25
neos-3530905-gaula	0.01	0.28	0.24	0.24	0.25
neos-3627168-kasai	0.06	0.43	0.12	0.26	0.18
neos-4333464-siret	−0.08	0.47	0.18	0.24	0.11
neos-4387871-tavua	0.0	0.43	0.17	0.24	0.15
neos-4650160-yukon	0.0	0.39	0.15	0.25	0.21
neos-4736745-arroux	−0.02	0.27	0.22	0.26	0.25
neos-4954672-berkel	0.18	0.36	0.14	0.25	0.25
neos-5076235-embley	0.01	0.39	0.13	0.25	0.23
neos-5107597-kakapo	0.0	0.36	0.15	0.26	0.23
neos-5260764-orauea	0.01	0.61	0.28	0.08	0.03
neos-5261882-treska	0.06	0.38	0.2	0.21	0.2
neos-631517	0.05	0.35	0.2	0.24	0.21
neos-691058	0.4	0.37	0.16	0.25	0.22
neos-860300	−0.04	0.59	0.31	0.1	0.0
neos16	0.0	0.3	0.21	0.27	0.22
newdano	0.0	0.37	0.13	0.26	0.24
nexp-150-20-1-5	0.27	0.59	0.09	0.26	0.05
nexp-150-20-8-5	0.56	0.4	0.2	0.21	0.19
p200x1188c	0.0	0.57	0.11	0.24	0.08
p500x2988	0.03	0.56	0.11	0.24	0.09
pg	0.23	0.35	0.14	0.25	0.25
pg5_34	0.15	0.42	0.12	0.26	0.2
probportfolio	0.01	0.43	0.24	0.15	0.18
prod2	0.0	0.64	0.28	0.08	0.0
r50x360	0.03	0.54	0.11	0.24	0.11
ran13x13	0.04	0.45	0.14	0.23	0.18
supportcase20	0.36	0.53	0.11	0.24	0.12
supportcase26	0.01	0.43	0.16	0.22	0.19
swath	0.0	0.63	0.29	0.08	0.0
tanglegram6	0.0	0.51	0.3	0.19	0.0
timtab1CUTS	0.0	0.36	0.15	0.27	0.23
tr12-30	0.69	0.36	0.15	0.26	0.24
usAbbrv-8-25_70	0.72	0.31	0.17	0.3	0.22