




IndiCon: Selecting SAT Encodings for Individual Pseudo-Boolean and Linear Integer Constraints

Felix Ulrich-Oltean
Dept. of Computer Science
University of York, UK
felix.ulrich-oltean@york.ac.uk 

Peter Nightingale
Dept. of Computer Science
University of York, UK
peter.nightingale@york.ac.uk 

James Alfred Walker
Dept. of Computer Science
University of York, UK
james.walker@york.ac.uk 

Abstract—Encoding to SAT and applying a state-of-the-art SAT solver can be a highly effective way of solving constraint problems. For many types of constraints there exist several alternative SAT encodings; and the choice of encoding can significantly affect SAT solver performance for any given problem. Previous work has shown that machine learning (ML) can be used to select SAT encodings for some constraint types, making a choice for each relevant constraint type in a problem instance. The state-of-the-art approach achieves good performance by first building a small portfolio of configurations, then selecting a configuration for a given problem instance using an ML model. The approach necessitates generating training data for every combination of encodings for the constraint types, thus it scales exponentially as more constraint types are added. In this work, we select potentially different encodings for each individual constraint in a problem instance. We are able to match the state-of-the-art performance while avoiding any limitation on the number of constraint types considered. To achieve this we are proposing new individual constraint features, we present a novel method for generating training data, and we have developed a new machine learning pipeline involving both unsupervised and supervised learning.

Index Terms—Constraint programming, SAT encodings, machine learning, global constraints, pseudo-Boolean constraints, linear constraints

I. INTRODUCTION

A popular and effective way of solving constraint satisfaction and optimisation problems (CSPs and COPs) is by reformulating them as instances of the Boolean satisfiability problem (SAT). This process, known as *encoding*, can take into account the particular structure of different constraint types in the constraint modelling language. For many constraint types, a variety of SAT encodings exist, i.e. schemes for representing a constraint in a CSP as a set of Boolean variables and a propositional formula over those variables.

The task of automatically selecting suitable encodings for constraints into SAT has been addressed previously [1], [2], [3]. Recent work shows that machine learning (ML) models can be trained to select a good encoding for pseudo-Boolean (PB) and linear integer (LI) constraints [3]. The ML-based selections lead to significant performance improvements over the single best encoding (based on the training set) even when predicting encodings for problem classes not present in the training set. The custom setup is also shown to greatly outperform the sophisticated algorithm selection tool AUTO-FOLIO [4]. In that work the choice was made once for each

relevant constraint type. In this work we also use specialised features, but we make the selection for each individual PB or LI constraint. We use the same constraint modelling pipeline as the authors of LEASE-PI [3], namely the SAVILEROW constraint modelling assistant [5] and the Kissat solver [6]. SAVILEROW works with models written in Essence Prime and can produce output for a variety of back-end solvers, including CP, SAT and SMT; Kissat has been consistently competitive in SAT solving competitions during recent years [7].

The ML setup discussed in [3] was shown to learn well to select good SAT encodings for PB and LI constraints, especially when using features of the specific constraint types. However, choosing one encoding for all constraints of a given type in a problem instance potentially leads to two issues. Firstly, a problem instance might contain many constraints of the same type but with quite different features. A single encoding selection may not be the best for all the constraints of that type. Secondly, the features of individual constraints are combined in [3] to produce a feature vector per instance, potentially losing valuable information by aggregation.

Two questions naturally arise. Is it practical to train an ML system to predict SAT encodings for each individual constraint of a given type? And if so, how does the performance compare to making one choice per constraint type in a CSP instance? In this work we address these questions, describing and evaluating an ML-based approach to learning to predict encodings at the individual constraint level. We refer to this system as INDICON. For ease of comparison, we refer to the approach set out in [3] as LEASE-PI (for “Learning to Select Encodings Per Instance”).

It is worth also noting that in LEASE-PI, the choice of PB and LI encoding was made together; for training, a reduced-size portfolio was used with 6 combinations of PB and LI encodings (out of a potential 81 combinations). Other constraint types exist for which a variety of SAT encodings are available (such as *at-most-one* and *table*). Extending the LEASE-PI approach to further constraint types becomes impractical. In INDICON, the training and prediction is made separately and therefore allows easy extension to any further constraint types.

In summary, our contributions are as follows:

- We address the problem of selecting SAT encodings for individual PB and LI constraints in instances of CSPs from *unseen* problem classes.

- We present and discuss how to obtain useful training data for individual constraints.
- We adapt and extend the *lipb* features from [3] for PB and LI constraints, in order to describe individual constraints.
- We evaluate empirically a number of alternative setups for our approach.

The focus is not primarily on performance compared to earlier work, but on scientific and methodological contributions:

- It is natural to ask whether setting constraint encodings individually is more effective than setting the encoding for all constraints of one type. We are investigating this question by building the INDICON system.
- INDICON scales better than the earlier LEASE-PI approach, as discussed below.
- INDICON produces simpler ML models and that is beneficial for explainability of decisions.

This work is from a PhD [8], which contains further details.

II. PRELIMINARIES

A *constraint satisfaction problem* (CSP) is defined as a set of variables X , a function that maps each variable to its domain, $D : X \rightarrow 2^Z$ where each domain is a finite set, and a set of constraints C . A *constraint* $c \in C$ is a relation over a subset of the variables X . The *scope* of a constraint c , named $\text{scope}(c)$, is the set of variables that c constrains. A *constraint optimisation problem* (COP) also minimises or maximises the value of one variable. A *solution* is an assignment to all variables that satisfies all constraints $c \in C$. Boolean Satisfiability (SAT) is a subset of CSP with only Boolean variables and only constraints (*clauses*) of the form $(l_1 \vee \dots \vee l_k)$ where each l_i is a literal x_j or $\neg x_j$. A *SAT encoding* of a CSP variable x is a set of SAT variables and clauses with exactly one solution for each value in $D(x)$. A SAT encoding of a constraint c is a set of clauses and additional Boolean variables A , where the clauses contain only literals of A and of the encodings of variables in $\text{scope}(c)$. An encoding of c has *at least* one solution corresponding to each solution of c . Pseudo-Boolean (PB) and Linear Integer (LI) constraints are in the form $\sum_{i=1}^n q_i x_i \diamond k$, where $\diamond \in \{<, \leq, =, \neq, \geq, >\}$, $q_1 \dots q_n$ are integer coefficients, k is an integer constant and x_i are Boolean or integer decision variables for PB and LI constraints respectively. An *at-most-one* (AMO) constraint over a set of Boolean decision variables requires that zero or one of them are set to true.

III. METHOD

Figure 1 summarises the steps involved in INDICON:

- 1) We start with a corpus of problems (A), solve instances with each single encoding choice per constraint type and record the timings (B). These allow us to identify a good default encoding choice for the instance.
- 2) We extract features of each individual PB or LI constraint in the problem instances (C).
- 3) We use a clustering algorithm to group all the relevant constraints across all instances into clusters with similar features (D).

Preparation of ML Dataset

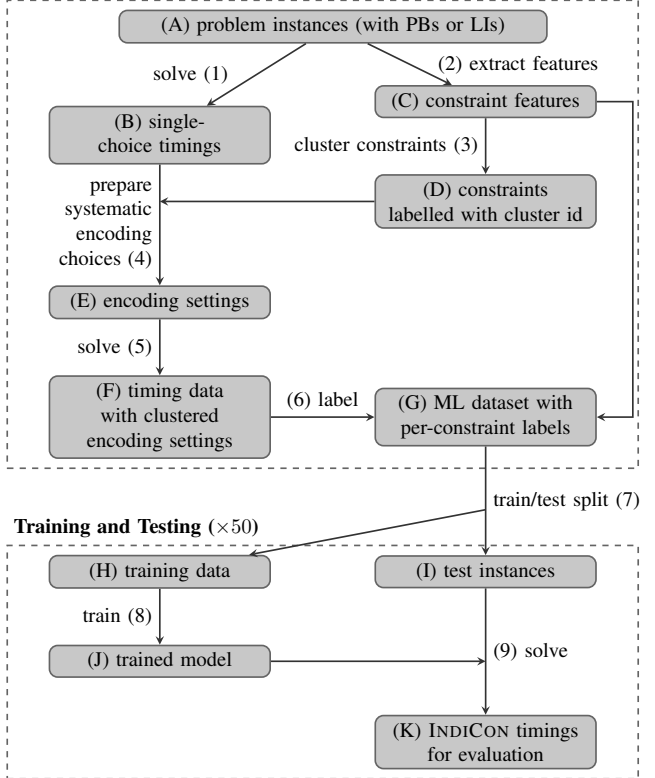


Fig. 1: The steps involved in INDICON.

- 4) We prepare a number of encoding settings (E) for each instance so that we can systematically try different encodings for constraints by cluster.
- 5) Each problem is solved by SAVILEROW [5] using a fast SAT solver with the per-cluster encoding settings (E) and we record the resulting runtimes (F).
- 6) The timing results allow us to generate a training set (G) with the best encoding label for each constraint.

We are now in a position to configure and train an ML model on the features and labels obtained above. For the sake of robustness, the entire process of splitting the corpus, training and testing is repeated 50 times for each setup.

- 7) The problems (A) are split into training and testing instances (H,I), keeping instances from the same problem class together, i.e. only in either the training or test set. This means we are making predictions for problem classes which have not been seen in the training phase.
- 8) An ML model is trained (J) to predict per-constraint encodings.
- 9) To evaluate performance, we solve the instances in our test set (I), consulting the ML model (J) to decide which encoding to use for each individual constraint, and recording the time taken (K) in order to analyse the performance.

In the rest of this section we briefly present the details of the steps introduced above. Further details, results and discussion can be found in [8].

A. Problem Corpus

The problems used are largely the same as in LEASE-PI [3], [9] (with the addition of more nurse scheduling problems based on instances in NSPLIB [10]), providing a variety of problem categories. The corpus consists of 50 constraint models with up to 50 instances each. There are 551 instances featuring PBs and 347 with LIs. Constraint models include problem classes such as nurse scheduling, car sequencing, knapsack, n -queens, balanced incomplete block design (BIBD), quasigroups, equidistant frequency permutation arrays (EFPA), multi-mode resource-constrained project scheduling (MRCPS), and optimum portfolio design (OPD).

B. Menu of PB and LI Encodings

We use the same 9 encodings for PB and LI as in [3]: the 8 PB(AMO) encodings described and analysed in [11] (GGPW, GGT, GGTd, GLPW, GMT0, GSWC, MDD, RGGT) as well as the non-AMO-aware *Tree* encoding [3] – all 9 encodings are in SAVILEROW. A PB(AMO) is a PB constraint where the decision variables are partitioned into subsets, each subject to an AMO constraint. The AMO partition is automatically detected [12] for existing PB constraints. LI constraints (if not encoded with *Tree*) are reformulated into PB(AMO)s where each integer variable becomes a set of Boolean variables with an AMO constraint. An equality (either PB or LI) can be either encoded with *Tree* or broken down into two inequalities and encoded with a PB(AMO) encoding.

Integer decision variables are encoded with the order encoding for *Tree* and with the direct encoding for the PB(AMO) encodings. Where both encodings are generated (due to a mix of PB/LI encodings or the presence of other constraints), SAT clauses are generated to channel the two representations. This has the potential to create larger encodings, but *Tree* performs well overall and so we include it in our menu nevertheless.

C. Feature Extraction

We adapt the feature extraction to consider the same aspects of PB/LI constraints as in the *lipb* featureset used in LEASE-PI [3]. In that work, the features of individual constraints were aggregated across all the constraints in a problem instance, but here we record each feature of the individual constraint. In addition, we extract the following: **is_equality** records whether the constraint is an equality (rather than \leq); **amog_maxw_med** is the median maximum weight across AMO groups and gives more information about the distribution of maximum weights in the AMO groups when coupled with the existing mean measure `amog_maxw_mn`; **amog_maxw_mn2k** is the ratio of the mean of maximum coefficients in the AMO groups to the upper limit k and could be an indication of how difficult the constraint will be to satisfy, with a higher value meaning a tighter constraint; **amog_maxw_sum** (the sum of the maximum coefficients) tells us the size that the “left-hand side” of the comparison could potentially reach. The complete list of resulting features is given in Table I.

TABLE I: Features used by INDICON, with their identifier and brief description. The *New* column shows new features added just for INDICON, in addition to LEASE-PI’s *lipb* features [3].

New Feature	Description
n	Number of terms
wsum	Sum of coefficients
q0	Minimum coefficient
q2	Median coefficient
q4	Maximum coefficient
iqr	IQR of coefficients
skew	Coefficients’ quartile skew
sepw	Number of distinct coefficient values
sepwr	Ratio of distinct coefficient values to number of coefficients
✓ is_equality	Is it an equality constraint?
k	Right-hand side k of the constraint
amogs	Number of At-Most-One groups (AMOGs)
amog_size_mn	Mean size of AMOGs
amog_size_mn_r2n	Mean AMOG size \div number of terms
✓ amog_maxw_med	Median size of the maximum coefficient across AMOGs
amog_maxw_mn	Mean size of the maximum coefficient across AMOGs
✓ amog_maxw_mn2k	The ratio of <code>amog_maxw_mn</code> : k
✓ amog_maxw_sum	Sum of the maximum coefficients in each AMOG
amogs_maxw_skew	Skew of the maximum coefficient in AMOGs
✓ amog_maxw_sum_k_prod	<code>amog_maxw_sum</code> $\times k$

D. Obtaining Training Data

The requirement for our training data is a feature vector per constraint (for the constraint type in question) as well as a target label to learn. We initially solve each instance in the corpus using each of the encodings available to establish a baseline default encoding for each problem instance. To obtain the target label, we adopt two approaches.

Inheriting from Host Instance: In this approach we use the baseline encoding of the instance (as described above) as the target label for all the constraints in the instance. This is an easy way to generate a training set, but it relies heavily on the assumption that constraints of one type will have similar characteristics within one problem instance, which seems in opposition to the motivation behind INDICON. Nevertheless, it turns out to be a useful method overall.

Clustering Across the Corpus: We cluster the individual constraints into groups across all the instances in the corpus. We use agglomerative clustering (as implemented in [13]) because it allows us to choose the number of clusters according to the data, rather than having to specify it arbitrarily as in some alternative unsupervised learning algorithms. Each feature vector of a constraint begins by being its own cluster. As we increase the allowed distance between points in a cluster, clusters merge. This is illustrated in Figure 2. In the first dendrogram (for PBs), we see for example that as the

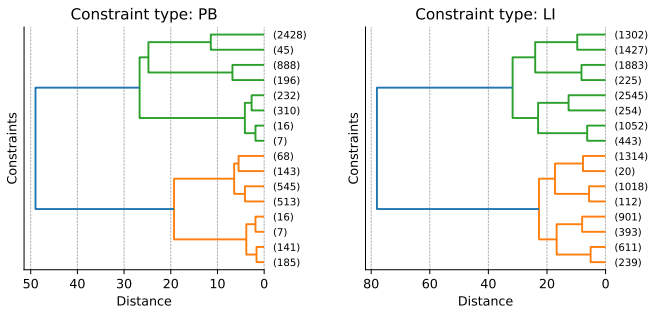


Fig. 2: Dendrograms showing agglomerative clustering by constraint features. The x-axis shows the Euclidean distance between clusters. On the y-axis labels indicate the number of data points in a branch.

inter-cluster distance passes 12, 6 clusters become 5. The data remains split into 5 clusters until the distance is 19. Then, the clusters keep joining quite quickly until we reach a distance of 27 and we’re left with 2 clusters. For our purpose we observe that 2 clusters and 5 clusters cover the largest ranges of distances. Of these we choose 5 clusters as this allows us to try more configurations for our encodings while still being practical to implement. Using similar reasoning for the LI constraints (illustrated in the second dendrogram), we select 6 clusters. In preliminary experiments we also worked with 3 clusters, but 6 clusters gave better results.

Systematically Timing Combinations of Encodings: Once each individual constraint is associated to a cluster, we need to try different combinations of encodings by cluster in order to obtain runtimes and thereby label the constraints in each cluster with a target encoding. Recall that we have 9 candidate encodings. Consider an instance with 5 clusters of constraints – an exhaustive approach would try every possible combination; however, that would amount to 9^5 combinations which would take an enormous amount of compute time to run. A compromise is to choose the single best choice as the baseline encoding for all constraints and then change the encodings of one cluster of constraints in turn. This way the example above would require $(1 + 8 \times 5) = 41$ combinations. Solving the instance 41 times remains practical to implement and the number of runs needed scales linearly with the number of encoding choices for a fixed number of clusters.

E. Training and Testing

We split our corpus into training and test sets randomly, ensuring that instances of a problem class are either in the training or test set, never in both. This means we are attempting the challenging task of making predictions for unseen problem classes. The train/test split is approximately 80% : 20% (approximate because of the different numbers of instances available for each problem class). We carry out 50 splits with different random seeds.

Similar to [3] we train classifiers to select between pairs of encodings; the final prediction is the result of voting from the

TABLE II: INDICON performance for the best 3 setups for PB and LI constraints, ordered from best to worst performing. Each setup is tested over 50 train/test splits. Performance is measured using PAR10 and shown as a multiple of the Virtual Best* (single-choice) time. Also shown are: the Single Best time; the number of clusters from which the training data was obtained; the ML classifier used – decision trees (DT), random forests (RF), or gradient boosted trees (GB).

INDICON for PB				INDICON for LI			
Setup		Runtime		Setup		Runtime	
Clusters	Classifier	PAR10	VB*	Clusters	Classifier	PAR10	VB*
1	RF	5.57					
1	DT	5.69		<i>Single Best</i>	RF	4.53	
5	GB	8.10		6	RF	6.44	
	<i>Single Best</i>	11.58		1	RF	6.70	
				6	GB	11.12	

trained classifier models. We separately train random forests, gradient boosted trees and simple decision tree models. In early experiments we also tried k-nearest neighbours and simple neural networks, as well as an ensemble of all classifiers mentioned above, but the performance was worse than with the classifiers we present here.

F. Experimental Setup

Experiments were carried out on a cluster with Intel Xeon 6138 20-core 2.0 GHz processors; the memory limit per job was 6GB. SAVILEROW was run with AMO detection switched on, a SAT clause limit of 10 million, and a 1 hour timeout. Kissat (*sc2021-sweep*) was used, with its own 1-hour timeout. Each run is repeated with 5 different seeds; the median runtime is calculated and a 10-fold penalty is applied for any total runtime over 1 hour to give PAR10 results.

IV. RESULTS AND DISCUSSION

The corpus contains problem classes with PBs, LIs, or both. We apply INDICON separately to these two constraint types, using the problems containing the relevant constraint type.

A. Selecting Encodings for One Constraint Type

To evaluate the performance of INDICON, we record the PAR10 running time for the 50 test sets as a multiple of the virtual best time achievable by using a single encoding. It is prohibitive to calculate a true virtual best by running every single combination of encodings for every constraint in any sizeable instance. Our reference is called *VB** to emphasise that this is a single-choice virtual best. We also show the single best (SB) result, which is the result of always choosing the one encoding which performed best on the training set.

The results are shown in Table II. For this corpus INDICON seems to work better for PBs than for LIs, with the best setup achieving 5.57 times the *VB** time, compared to 11.58 times for the single best (SB) time. In the LI setting, INDICON does not even match the SB performance of 4.53 on this corpus, coming in at 6.44 times *VB**. This may be explained to an extent by the fact that in [3] the authors found that the choice of PB encoding could make a much bigger difference

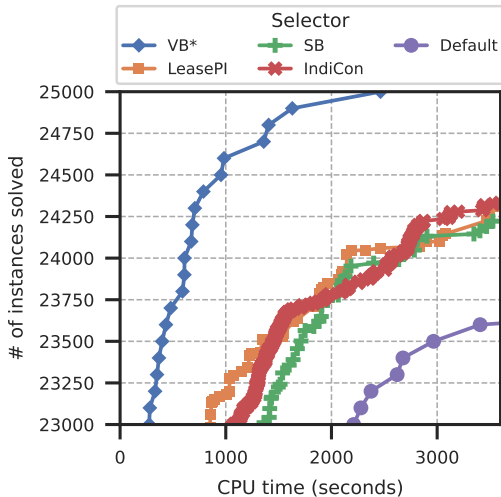


Fig. 3: Number of instances solved by CPU time up to 1 hour for the single-choice virtual best (VB*), single best (SB), default encoding (Def), best LEASE-PI setup and INDICON.

to solving performance, whereas for LIs it tended to be the case that there was one encoding (GGPW) which usually outperformed the others.

In terms of the classifiers used, it is interesting that a simple decision tree classifier employed in a pairwise voting setup is performing almost as well as random forests for selecting the PB encoding – this opens up the way to more explainability for the encoding choice made.

The best setups for PB come from the non-clustered training data, i.e. where the target label for training was simply the encoding which worked best for the host instance of every constraint. In the LI setting, the setup using 6 clusters slightly outperforms the setup based on the simpler labelling source.

B. Comparison with Per-Instance Selection in LEASE-PI

We carry out a second experiment in order to compare the performance of INDICON with LEASE-PI. We consider the 250 instances which contain both PB and LI constraints and appear in at least one of the LEASE-PI and one of the INDICON test sets. For each instance we randomly sample with replacement 100 results from LEASE-PI and 100 results of running INDICON to select both PB and LI encodings. Each solving run is done 5 times and the median time is recorded to account for randomness in SAT solving.

The results are shown in Figures 3 and 4. Figure 3 shows how many instances were solved as we increase the CPU time. INDICON is competitive with LEASE-PI and does better for some of the harder instances which take around 3000 seconds to solve. This slight edge is confirmed by the mean PAR10 solving time across the 25000 “contests”: for LEASE-PI the mean is 1161 seconds with 689 timeouts, and for INDICON the mean is 1145 seconds with 668 timeouts. In Figure 4 we see a fairly consistent performance between the two selectors, without any extreme differences as there are no crosses in

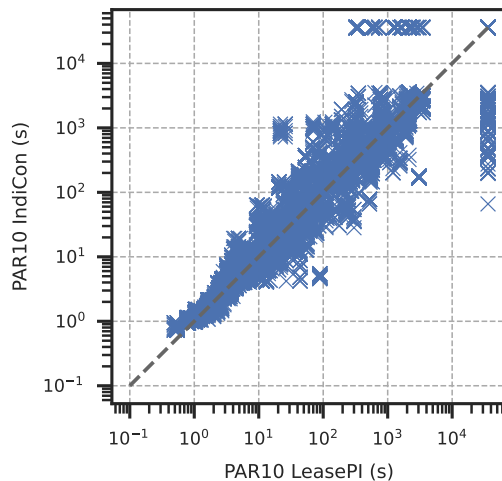


Fig. 4: PAR10 times for the best LEASE-PI versus INDICON where INDICON has set both PB and LI encodings. The number of timeouts is 689 for LEASE-PI and 668 for INDICON.

the top left or bottom right corners. The curve at the bottom left indicates that LEASE-PI is doing better on the easiest instances, whose runtime is under 1 second. The overhead of retrieving a separate encoding choice for each constraint from the ML classifiers becomes less significant as the overall solving times increase for harder problems.

The Lease-PI paper [3] has a section (Analysis of the configuration space) which shows that the best choice for LI was dominated by one good choice, whereas for PB constraints the best choice of encoding was much more varied. This is not fully explained, but it could be that in the corpus we used the sum (LI) constraints were encoding larger integer values, whereas the coefficients in the PBs were smaller. The mean value for median coefficient in a PB is 1.03, compared to 142 for LI constraints. In INDICON we once again find that selecting encodings for PB constraints affects the performance more than for LI constraints, as discussed in Section IV-A.

A final observation is that in LEASE-PI the ML setup is able to take into account both types of constraints across the entire instance, so to some extent it could learn combinations of constraint choices based on how decision variables are shared between the different constraints. Here, we make the choice in isolation for each constraint type but are still able to match the performance of LEASE-PI. We did run trials which included whole-instance features in the INDICON training data, but performance actually suffered.

C. Analysis of Encodings Choices

1) *PB constraints*: We present in Table III a summary of the variety of distinct encoding choices made for all the PB constraints in our corpus when using the most successful setup, i.e. pairwise random forest classifiers using a single cluster for target labels. Leaving aside the models which just have one PB constraint, such as *knapsack*, we observe that for many

TABLE III: Distinct PB encoding choices made per instance by INDICON for problems with at least 10 instances. We show the model name, # of instances, the mean number of PBs per instance, the min., max., and mean number of choices, as well as the standard deviation.

Problem Class	# inst	# PBs	# distinct encodings used			
			min	max	mean	std
bibd	19	174.42	1	4	2.15	0.54
bibd-implicit	22	212.86	1	3	2.15	0.39
blackHole	11	101.09	1	2	1.14	0.35
bpmp	14	7.00	1	1	1.00	0.00
carSequencing	49	975.20	1	3	1.43	0.73
efpa	20	78.30	2	3	2.08	0.28
handball7	20	516.00	1	2	1.17	0.37
killerSudoku2	50	1149.10	1	2	1.17	0.38
knapsack	18	1.00	1	1	1.00	0.00
knights	44	85.23	1	2	1.02	0.14
langford	39	73.10	1	2	1.10	0.30
mrcpsp-pb	20	100.45	1	3	1.39	0.53
n_queens	20	1326.50	1	2	1.25	0.44
n_queens2	16	256.50	1	2	1.33	0.48
nurse-sched	50	99.12	1	2	1.89	0.32
opd	33	11.18	1	2	1.73	0.45
sonet2	24	10.00	1	2	1.47	0.50

TABLE IV: PB encodings selected by INDICON in 50 test sets. We show the number of constraints for which each encoding has been selected, as a total and as a proportion; we also show how often each encoding was the single best encoding (SB) for an instance.

Encoding	individual constraints		instance single best	
	freq.	%	freq.	%
Tree	1,134,061	94.8	255	46.3
MDD	60,335	5.0	76	13.8
GGPW	968	0.1	79	14.3
GGTd	733	0.1	49	8.9
GGT	0	0	17	3.1
GMTO	0	0	20	3.6
GLPW	0	0	8	1.5
RGGT	0	0	21	3.8
GSWC	0	0	26	4.7

problems several encoding choices are made within instances. These include BIBD, EFPA, nurse scheduling and OPD.

A further insight is given in Table IV which shows how often each encoding was chosen by INDICON across the test sets for PB constraints, once again using the best-performing setup. The table also shows the distribution of *single best* encoding for instances as a whole (recall that this is used at the beginning of building the training data to obtain good base encodings). We see that the selections made by INDICON rely mostly on the *Tree* and *MDD* encodings, using them almost exclusively. The distribution of encodings in the single best column suggests that INDICON is not making full use of the suite of encodings available; however, the results shown earlier in Table II indicate that the ability to select the encodings at individual constraint level gives a significant performance advantage over the single best choice.

As an aside, we carried out the same analysis of encoding

TABLE V: Distinct LI encoding choices as in Table III.

Problem Class	# inst	# LIs	# distinct encodings used			
			min	max	mean	std
briansBrain	16	1.00	1	1	1.00	0.00
handball7	20	633.15	2	3	2.67	0.48
immigration	23	1.00	1	1	1.00	0.00
killerSudoku2	50	64.96	2	5	2.49	0.63
knapsack	24	1.00	1	1	1.00	0.00
knights	44	168.45	1	2	1.15	0.36
life	16	219.94	2	4	2.52	0.70
molnars	17	2.00	1	2	1.45	0.50
mrcpsp-pb	19	29.21	1	6	2.42	0.87
opd	33	80.58	1	3	2.00	0.47
sonet2	24	1.00	1	1	1.00	0.00

TABLE VI: LI encodings selected by INDICON as in Table IV.

Encoding	individual constraints		instance single best	
	freq.	%	freq.	%
GGPW	150,933	72.3	78	22.5
RGGT	42,486	20.3	27	7.8
GMTO	6,623	3.2	10	2.9
GSWC	3,735	1.8	16	4.6
GLPW	2,021	1.0	3	0.9
GGTd	2,001	1.0	22	6.3
GGT	871	0.4	18	5.2
Tree	178	0.1	136	39.2
MDD	0	0	37	10.7

selections using the second-best performing setup which uses decision trees rather than random forests. We observed a bigger spread of selections, and almost identical performance. It appears that the random forest classifiers are yielding a narrower set of predictions which are more robust than the ones obtained from just using decision trees.

2) *LI constraints*: We turn our attention to the selection of encodings for LI constraints. We focus on the best performing setup, which for LI is pairwise random forest classifiers trained on timings which were obtained from 6 constraint clusters. The summary statistics of the number of distinct LI encoding selections is shown in Table V. Broadly there are fewer LIs per instance (compared to problems which had PBs, Table III), but there are more problem classes where at least two choices are used and indeed cases where 6 different encodings are used.

The distribution of encodings used is broader for LIs than for PBs; in Table VI there is still a dominant choice (this time GGPW), but RGGT is also selected for 20% of constraints and all encodings except MDD are picked at some point. When we compare against the single best encodings shown on the right of the table, we may deduce that INDICON is missing out on some very good choices; however this conclusion is not necessarily sound because the single best choice cannot apply different encodings to individual constraints.

D. Explaining Decisions using Decision Trees

We noted earlier that decision trees in a pairwise arrangement perform almost as well as random forests for selecting PB encodings. To illustrate how this selection is made, we show in Figure 5 one example decision tree produced in the

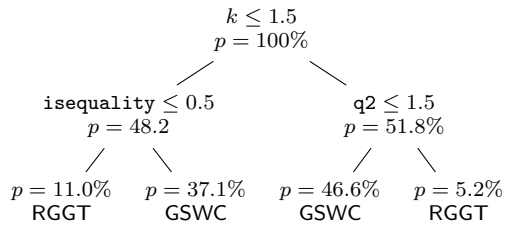


Fig. 5: Decision tree from one pairwise training run for PB encodings. The p values are the proportion of training samples; on the left side of operators are features, e.g. k is the upper limit of the PB constraint, $q2$ is the median coefficient, and $isequality$ is 1 for an equality constraint and 0 otherwise.

training phase. This particular tree is selecting between RGGT and GSWC. The first branch is on the upper limit k of the PB constraint; when the upper bound is small (the left branch), then the choice comes down to whether we are dealing with an equality; however, when the upper limit is at least 2 (the right branch), then the choice depends on the median coefficient $q2$.

V. RELATED WORK

LEASE-PI selects SAT encodings per constraint type using ML; in [3] the authors compare LEASE-PI's performance with AUTOFOLIO [4] which is a sophisticated (albeit general) algorithm selection tool. LEASE-PI significantly outperforms AUTOFOLIO on the specific task in question. In this paper we show that INDICON performs slightly better even than LEASE-PI. In earlier work, Soh et al. [14] select between the order, log and a hybrid encoding on the basis of a single criterion related to the domain size of the variables involved. PBLIB [15] provides a *PB2CNF* class which selects the encoding based purely on the size of the constraint. ML is used to select SAT encodings for integer variables in MeSAT [2]. Proteus [1] also makes this kind of choice based on CSP instance features, having first chosen whether to use a SAT or CP solver; it also predicts which SAT solver to use. An automated approach to extracting SAT problem features has been proposed [16], using autoencoders to represent the SAT formula in a low-dimensional space – this approach could be used to extract features of PB and LI constraints in our work.

VI. CONCLUSION

We have presented INDICON, an ML system for selecting SAT encodings of individual constraints in a CP model. To our knowledge, INDICON is unique in choosing an encoding for each constraint separately. We have shown that the performance of INDICON for selecting both PB and LI constraint encodings is marginally better than the existing state of the art. The key benefits of INDICON compared to the prior work are *scaling* and *simplicity* (leading to explainability). It treats each constraint type as a separate ML problem, and as a consequence it scales linearly in the number of constraint types (unlike LEASE-PI [3], the best version of which scales exponentially in the number of constraint types). INDICON

typically learns simpler models than LEASE-PI, which benefits explainability of the system. Even very simple ML models such as decision trees can provide competitive results in this context as demonstrated in our experimental evaluation, in which selections are made for unseen problem classes. We also share insights about the distribution of encoding selections made by INDICON for PB and LI constraints.

ACKNOWLEDGMENTS

We thank the UK EPSRC for grants EP/W001977/1 and EP/R513386/1. We used Viking, a compute cluster provided by the University of York. We are grateful to the University of York, IT Services and the Research IT team.

REFERENCES

- [1] B. Hurley, L. Kotthoff, Y. Malitsky, and B. O'Sullivan, "Proteus: A Hierarchical Portfolio of Solvers and Transformations," in *Integration of AI and OR Techniques in Constraint Programming*, ser. Lecture Notes in Computer Science, H. Simonis, Ed. Cham: Springer International Publishing, 2014, pp. 301–317.
- [2] M. Stojadinović and F. Marić, "meSAT: Multiple encodings of CSP to SAT," *Constraints*, vol. 19, no. 4, pp. 380–403, Oct. 2014.
- [3] F. Ulrich-Oltean, P. Nightingale, and J. A. Walker, "Learning to select SAT encodings for pseudo-Boolean and linear integer constraints," *Constraints*, Nov. 2023.
- [4] M. Lindauer, H. H. Hoos, F. Hutter, and T. Schaub, "AutoFolio: An Automatically Configured Algorithm Selector," *Journal of Artificial Intelligence Research*, vol. 53, pp. 745–778, Aug. 2015.
- [5] P. Nightingale, Ö. Akgün, I. P. Gent, C. Jefferson, I. Miguel, and P. Spracklen, "Automatically improving constraint models in Savile Row," *Artificial Intelligence*, vol. 251, pp. 35–61, Oct. 2017.
- [6] A. Biere, K. Fazekas, M. Fleury, and M. Heisinger, "CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020," in *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*. University of Helsinki, 2020, pp. 51–53.
- [7] "SAT Competition Results 2023," <https://satcompetition.github.io/2023/>.
- [8] F. Ulrich-Oltean, "Learning SAT Encodings for Constraint Satisfaction Problems," Ph.D. dissertation, University of York, Sep. 2023. [Online]. Available: <https://etheses.whiterose.ac.uk/34581/>
- [9] "Learning to Select SAT Encodings: Data and Code for Experiments," <https://github.com/felixvuo/lease-data>, Jan. 2023.
- [10] M. Vanhoucke and B. Maenhout, "NSPLib – A Nurse Scheduling Problem Library: A tool to evaluate (meta-)heuristic procedures," p. 11.
- [11] M. Bofill, J. Coll, P. Nightingale, J. Suy, F. Ulrich-Oltean, and M. Villaret, "SAT encodings for Pseudo-Boolean constraints together with at-most-one constraints," *Artificial Intelligence*, vol. 302, p. 103604, Jan. 2022.
- [12] C. Ansótegui, M. Bofill, J. Coll, N. Dang, J. L. Esteban, I. Miguel, P. Nightingale, A. Z. Salamon, J. Suy, and M. Villaret, "Automatic detection of at-most-one and exactly-one relations for improved SAT encodings of pseudo-boolean constraints," in *International Conference on Principles and Practice of Constraint Programming*. Springer, 2019, pp. 20–36.
- [13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [14] T. Soh, M. Banbara, and N. Tamura, "A Hybrid Encoding of CSP to SAT Integrating Order and Log Encodings," in *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, Nov. 2015, pp. 421–428.
- [15] T. Philipp and P. Steinke, "PBLib – A Library for Encoding Pseudo-Boolean Constraints into CNF," in *Theory and Applications of Satisfiability Testing – SAT 2015*, M. Heule and S. Weaver, Eds., 2015, pp. 9–16.
- [16] M. Dalla, A. Visentin, and B. O'Sullivan, "Automated SAT Problem Feature Extraction using Convolutional Autoencoders," in *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*. Washington, DC, USA: IEEE, Nov. 2021, pp. 232–239.