An Empirical Comparison of Cost Functions in Inductive Logic Programming

CÉLINE HOCQUETTE, University of Southampton, United Kingdom
ANDREW CROPPER, ELLIS Institute, Finland and University of Helsinki, Finland

Recent inductive logic programming (ILP) approaches learn optimal hypotheses. An optimal hypothesis minimises a given cost function on the training data. There are many cost functions, such as minimising training error, minimising textual complexity, or minimising the description length of hypotheses. However, selecting an appropriate cost function remains a key question. To address this gap, we extend a constraint-based ILP system to learn optimal hypotheses for seven standard cost functions. We then empirically compare the generalisation error of optimal hypotheses induced under these standard cost functions. Our results on over 20 domains and 1,000 tasks, including game playing, program synthesis, and image reasoning, show that, while no cost function consistently outperforms the others, minimising training error or description length has the best overall performance. Notably, our results indicate that minimising the size of hypotheses does not always reduce generalisation error.

JAIR Track: Constraint Programming and Machine Learning

JAIR Associate Editor: Christian Bessiere

JAIR Reference Format:
Céline Hocquette and Andrew Cropper. 2026. An Empirical Comparison of Cost Functions in Inductive Logic Programming. *Journal of Artificial Intelligence Research* 86, Article 24 (July 2026), 27 pages. DOI: 10.1613/jair.1.21950

1 Introduction

Inductive logic programming (ILP) (Cropper and Dumancic 2022; S. Muggleton 1991) is a form of machine learning. The goal of ILP is to use training examples to find a hypothesis that generalises to unseen examples. The key characteristic of ILP is that it uses logic programs (sets of logical rules) to represent hypotheses.

Like other machine learning approaches, ILP uses a cost function to guide the search for a hypothesis. A cost function assigns a score to each hypothesis, establishing an ordering over the hypothesis space (the set of all possible hypotheses). Choosing a suitable cost function is critical to learning performance. For instance, minimising training error alone often leads to *overfitting*, where the learned hypothesis fits the training examples, including noise, too closely, resulting in poor generalisation (Dietterich 1995). Many cost functions mitigate overfitting by balancing training error and hypothesis complexity, following the minimal description length (MDL) principle (Rissanen 1978). However, these cost functions often struggle with limited training data (Kearns et al. 1995).

Previous studies have empirically compared the performance of different cost functions in machine learning (Fürnkranz and Flach 2005; Mingers 1989b). However, these studies evaluate cost functions used as heuristics within non-optimal learning algorithms. In other words, they assess the performance of a cost function only in the context of a particular algorithm rather than in isolation. For instance, Mingers (1989b) evaluates heuristics

Authors' Contact Information: Céline Hocquette, ORCID: 0000-0001-6732-1587, c.m.e.j.hocquette@soton.ac.uk, University of Southampton, United Kingdom; Andrew Cropper, ORCID: 0000-0002-4543-7199, andrew.cropper@helsinki.fi, ELLIS Institute, Finland and University of Helsinki, Finland.

DOI: 10.1613/jair.1.21950

for attribute selection in decision-tree learning, where the heuristic determines the sequence of splits in a greedy manner, resulting in non-optimal trees. This tight coupling with a specific learning algorithm makes it difficult to isolate the intrinsic effectiveness of the cost functions, as their impact is entangled with the algorithm's procedural biases.

The same issue has long affected ILP because classical ILP approaches do not learn optimal hypotheses (Ahlgren and Yuen 2013; Blockeel and Raedt 1998; S. H. Muggleton 1995; Quinlan 1990; Srinivasan 2001; Zeng et al. 2014). Instead, they use cost functions heuristically to guide the construction of a hypothesis, rather than directly optimising a global cost function applied to entire hypotheses. For instance, Aleph (Srinivasan 2001) follows a set-covering strategy and uses a cost function to iteratively select individual rules that entail some positive examples. As a result, evaluating cost functions independently of a specific learning algorithm has been difficult in ILP.

A major advancement in recent ILP systems is their ability to learn *globally optimal* hypotheses (Cropper and Dumancic 2022). An optimal hypothesis minimises a given cost function, evaluated on the training data, within the hypothesis space. For instance, many ILP systems learn a hypothesis that entails all positive training examples, no negative ones, and has minimal size (Bembenek et al. 2023; Corapi et al. 2011; Cropper and Morel 2021; Evans and Grefenstette 2018; Evans, Hernández-Orallo, et al. 2021; Kaminski et al. 2019; S. H. Muggleton, Lin, et al. 2015). Other approaches support noisy data by searching for hypotheses that balance data fit with hypothesis complexity (Hocquette, Niskanen, et al. 2024; Law, Russo, and Broda 2018). In theory, given the same bias, training data, and sufficient learning time, these optimal systems should learn the same hypothesis (or one with the same cost, since multiple hypotheses may have the same cost).

This recent progress allows us to directly compare the impact of cost functions on generalisation performance for the first time. In this work, we present the first empirical study comparing the generalisation performance of globally optimal hypotheses learned using different cost functions. Moreover, unlike previous studies focusing on classification (Mingers 1989b) or rule learning (Fürnkranz and Flach 2005) tasks, our study spans a broad range of domains, including image reasoning, program synthesis, and game playing.

To conduct our study, we use the constraint-based ILP system Popper (Cropper and Hocquette 2023; Cropper and Morel 2021; Hocquette, Niskanen, et al. 2024). We use this system because it is guaranteed to learn globally optimal hypotheses and can learn recursive hypotheses. Popper frames the ILP problem as a constraint satisfaction problem (CSP), where each solution to the CSP represents a hypothesis. The goal of Popper is to accumulate constraints to restrict the hypothesis space and thus constrain the search. Following Cropper and Hocquette (2023), we first search for small hypotheses that generalise a subset of the examples, then combine them to form a larger hypothesis. We formulate the search for combinations as a maximum satisfiability (MaxSAT) problem (Bacchus et al. 2021). We extend Popper to support seven widely-used cost functions, including MDL, training error minimisation, and hypothesis size minimisation. We train Popper on many tasks from many domains and evaluate the generalisation performance of hypotheses optimised using each cost function.

Our empirical results show that no single cost function consistently outperforms the others in all domains. However, minimising training error and description length are the best-performing cost functions overall. Moreover, our results indicate the best-performing cost functions greatly vary depending on the domains, suggesting that the suitability of a cost function highly depends on the specific characteristics of the domain. Notably, the MDL cost function struggles with few positive examples, as it struggles to identify a hypothesis that compresses the data. In addition, our results indicate that minimising hypothesis size can either improve or degrade generalisation performance, challenging the assumption that smaller hypotheses inherently generalise better.

Novelty and Contributions. This study is the first to compare the performance of *globally optimal* hypotheses for different cost functions, thus directly comparing cost function performance. The main contribution is a large-scale

empirical evaluation of seven cost functions on over 20 domains and 1000 tasks. Overall, we make the following contributions:

(1) We extend the constraint-based ILP system Popper to support seven standard cost functions. We formulate the search for optimal hypotheses as a MaxSAT problem.
(2) We empirically compare the generalisation performance of hypotheses provably optimal for seven standard cost functions, using over 1000 tasks from diverse domains such as image reasoning, program synthesis, and game playing. Our study is the first empirical study directly comparing the generalisation performance of globally optimal hypotheses for multiple cost functions.

Our results provide insights into the conditions under which each cost function performs best, providing guidance for researchers when selecting cost functions for training ILP systems.

2 Related Work

2.1 Cost/loss Functions

Machine learning algorithms use a cost or loss function to find a hypothesis[1] that minimises it. The loss function quantifies the difference between the hypothesis's predictions and the target values. For instance, the mean squared error is often used for numerical predictions and cross-entropy for classification tasks (Q. Wang et al. 2022). By contrast, we focus on learning relational hypotheses, where predictions are logical assertions rather than numerical values.

2.2 Regularisation

Overfitting occurs when the hypothesis fits the noise in the training data, resulting in poor generalisation to unseen data (Mitchell 1997). Regularisation techniques aim to improve generalisation by sacrificing a small reduction in training accuracy to mitigate overfitting. Standard techniques, such as L_0 and L_1 regularisation, add a penalty term to the cost function to limit the complexity of the hypothesis (Tian and Y. Zhang 2022). By contrast, we empirically compare cost functions that include both the error of the hypothesis and regularisation penalties. Furthermore, we go beyond identifying overfitting and analyse conditions under which each tested cost function performs best. Suboptimal algorithms implicitly embody regularisation terms (Dietterich 1995). For instance, C. Zhang et al. (2021) show that unregularised overparametrised deep learning models can still generalise well, suggesting implicit regularisation during training. However, the amount of regularisation remains unclear. By contrast, we explicitly learn optimal hypotheses. Kearns et al. (1995) provide empirical and theoretical evidence that penalisation techniques which only consider empirical loss and hypothesis structure, such as MDL, cannot perform well for all sample sizes. In this work, we learn logic programs rather than numerical or Boolean functions.

2.3 Occam's Razor

Occam's razor (Blumer et al. 1987) is a widely used principle which recommends preferring simpler hypotheses over more complex ones. P. M. Domingos (1999) identifies two main interpretations for this principle. The first states that, if two hypotheses have the same generalisation error, the simpler one should be preferred because simplicity is inherently desirable. The second states that if two hypotheses have the same training error, the simpler one should be preferred because it is likely to generalise better. While the first interpretation is widely accepted, the second is generally incorrect. Esmeir and Markovitch (2007) provide empirical evidence supporting the validity of this second interpretation in the context of decision tree learning. However, Zahálka and Železný (2011) refutes the claim that hypothesis complexity causes overfitting, showing that overfitting is related to the

[1] In machine learning, a *hypothesis* is sometimes referred to as a *model*. Following Mitchell (1997), we use the terminology *hypothesis*.

number of hypotheses tested. Furthermore, Schaffer (1993) empirically shows that overfitting avoidance strategies, such as minimising hypothesis complexity, are a form of bias and can improve generalisation performance when this bias is appropriate for the application context. In contrast to these studies focused on classification tasks, we empirically evaluate whether learning textually minimal hypotheses improves generalisation performance in the context of rule learning. Additionally, while previous studies evaluate suboptimal hypotheses, our study evaluates optimal hypotheses.

2.4 Evaluation Metrics

Evaluation metrics, such as accuracy, mean squared error, and the area under the ROC curve, assess the performance of a learned hypothesis. Ferri et al. (2009) empirically show that the correlation between different evaluation metrics decreases in small datasets and multiclass problems, especially with highly imbalanced class distributions. Provost et al. (1998) argue that predictive accuracy is often an inadequate evaluation metric, primarily because misclassification costs and class distributions are usually unknown. While evaluation metrics are applied to test data, we focus on cost functions used during training. Moreover, while both studies focus on classification tasks, we learn logic programs.

2.5 Decision Trees

Decision trees are typically induced in a two-stage process: tree-building and pruning. During tree-building, a heuristic, such as entropy, information gain, Gini impurity, or MDL, is used to determine the best data split at each node (Quinlan and Rivest 1989). While decision tree heuristics evaluate tree nodes, our cost functions evaluate entire hypotheses. Empirical results show that, while the predictive accuracy of decision trees is not sensitive to the choice of splitting heuristic (Mingers 1989b), pruning strategies can substantially impact their performance (Mingers 1989a). Moreover, Murphy and Pazzani (1993) empirically show that slightly larger trees among trees consistent with training data often outperform the shortest ones in terms of predictive accuracy. This result suggests that minimising tree size does not always provide the best generalisation performance. In this work, we learn logic programs, including recursive programs, instead of decision trees. Moreover, we learn optimal hypotheses which allows us to compare cost functions directly. Recent work shows that optimal decision trees often perform similarly to sub-optimal ones (Bessiere, Hebrard, et al. 2009; Narodytska et al. 2018). Linden et al. (2024) compare complexity tuning methods for optimal decision trees and show that the tested methods perform similarly. However, these approaches predict binary classification labels while we learn relational rules. Moreover, they learn trees with minimal size and do not evaluate the performance of alternative cost functions.

2.6 Data Mining

Tan et al. (2002) compare metrics for association rule mining and feature selection. They show no measure is better than others in all application domains because they have different properties. Moreover, these measures are often highly correlated. While these metrics capture dependencies between variables in a dataset, we focus on cost functions that evaluate learned hypotheses.

2.7 Rule Mining

Geng and Hamilton (2006) survey interestingness metrics for ranking patterns in data mining based on nine criteria defining interestingness, including conciseness, reliability, and novelty. Bayardo Jr. and Agrawal (1999) show that many well-known interestingness measures (including gain, Gini, entropy gain, and Laplace) are monotone functions of support and confidence. Therefore, optimal rules for these measures are located on the support-confidence border.

2.8 Rule Learning

Separate-and-conquer algorithms use heuristics to evaluate the quality of partial rules. Fürnkranz and Flach (2005) survey and Janssen and Fürnkranz (2010) empirically evaluate common rule learning heuristics for greedy inductive rule learning algorithms. By contrast, we evaluate cost functions, which assign a cost to an entire logic program rather than to individual rules. Additionally, while separate-and-conquer algorithms are not optimal learners, we learn optimal hypotheses and directly compare the performance of different cost functions.

2.9 Constraint Acquisition

In constraint acquisition (Bessiere, Koriche, et al. 2017), hypotheses are represented as constraints, and the goal is to learn a constraint network that explains observed examples. Unlike ILP, which learns logical rules that entail examples, constraint acquisition learns declarative constraints over finite domains. Constraint-based ILP is closely related to CA, as both use constraints to define the hypothesis space and employ solver-based, cost-guided search to identify consistent models.

2.10 Program Synthesis

Several program synthesis systems use heuristics to guide a bottom-up search algorithm (Ameen and Lelis 2023; Barke et al. 2020; Odena et al. 2021). These heuristics evaluate partial hypotheses to inform the search process. By contrast, we compare cost functions, which assign a cost to an entire hypothesis. Cropper and Dumančic (2021) use user-provided cost functions as heuristics to guide the search. However, they do not learn optimal hypotheses. Additionally, program synthesis systems use a domain-specific ranking function to resolve ambiguities in user intent and prioritise certain hypotheses over others (Polozov and Gulwani 2015). A common approach is to choose the shortest hypothesis consistent with the examples (Gulwani 2011). Alternatively, to support noisy examples, Handa and Rinard (2020) propose an objective function that balances training error and hypothesis complexity, and a lexicographic ordering to first minimise training error, then complexity. Bester (Peleg and Polikarpova 2020) ranks hypotheses based on a linear combination of factors, including the number of training examples satisfied, the distance of unsatisfied examples from their intended output, hypothesis size, and the number of inputs used. Singh and Gulwani (2015) use machine learning to learn ranking functions as weighted combinations of both hypothesis and data features. In this work, we empirically compare several ranking functions.

2.11 ILP

Early ILP systems do not learn optimal hypotheses and use heuristics to guide the search. These heuristics evaluate partial hypotheses, often partial rules. For instance, Foil (Quinlan 1990) uses an information-based heuristic to guide rule construction. Progol (S. H. Muggleton 1995) searches for a hypothesis that minimises compression. Aleph (Srinivasan 2001) supports 13 cost functions, including training accuracy and compression. In contrast to these approaches, our work focuses on learning optimal hypotheses and, therefore, directly evaluates the performance of cost functions. Recent ILP systems learn optimal hypotheses (Corapi et al. 2011; Cropper and Morel 2021; Evans and Grefenstette 2018; Law, Russo, and Broda 2014; S. H. Muggleton, Lin, et al. 2015). These systems often learn textually minimal hypotheses (Corapi et al. 2011; Cropper and Morel 2021; Law, Russo, and Broda 2014; S. H. Muggleton, Lin, et al. 2015), following Occam's razor, which suggests that simpler hypotheses are preferable. In this work, we empirically evaluate whether minimising the size of hypotheses improves generalisation performance. Several approaches learn an MDL hypothesis and trade off hypothesis complexity with the fit of the data to support noisy examples (Evans, Hernández-Orallo, et al. 2021; Hocquette, Niskanen, et al. 2024). In this work, we evaluate how well the MDL cost function performs compared to other cost functions. Several systems allow user-provided cost functions (Law, Russo, Bertino, et al. 2020; Srinivasan

2001). However, it is unclear how to choose these cost functions. Our study investigates which cost functions are appropriate in which context.

3 Problem Setting

We describe our problem setting.

3.1 Terminology

We assume familiarity with logic programming (Lloyd 2012) but restate some key terminology. A *variable* is a string of characters starting with an uppercase letter. A *function symbol* is a string of characters starting with a lowercase letter. A *predicate symbol* is also a string of characters starting with a lowercase letter. The *arity* of a function or predicate symbol is the number of arguments it takes. A *constant symbol* is a function symbol with arity zero. A *term* is a variable, a constant symbol, or a function symbol of arity n applied to an n-tuple of terms. An *atom* is a tuple $p(t_1, \ldots, t_n)$, where p is a predicate of arity n and $t_1, \ldots, t_n$ are terms. A *literal* is either an atom or the negation of an atom. A *clause* is a set of literals. A clausal theory is a set of clauses. A *constraint* is a clause without a positive literal. A *definite* clause is a clause with exactly one positive literal. We use the term *rule* interchangeably with *definite clause*. A *definite program* is a set of definite clauses under the least Herbrand model semantics. We refer to a definite program as a *logic program*. We use the term *program* interchangeably with *hypothesis*, i.e. a hypothesis is a program.

3.2 ILP Problem

We formulate the ILP problem in the learning from entailment setting (Raedt 2008). We define an ILP input:

Definition 1 (ILP input). An ILP input is a tuple $(E, B, \mathcal{H})$ where $E = (E^+, E^-)$ is a pair of sets of ground atoms denoting positive (E^+) and negative (E^-) examples, B is a definite program denoting background knowledge, and $\mathcal{H}$ is a hypothesis space, i.e. a set of possible hypotheses.

We restrict hypotheses and background knowledge to definite programs. Positive and negative examples are ground atoms, not necessarily input-output pairs. We define a cost function:

Definition 2 (Cost function). Given an ILP input $(E, B, \mathcal{H})$, a cost function $cost_{E,B} : \mathcal{H} \rightarrow S$ maps hypotheses to a totally ordered set S.

We define an *optimal* ILP hypothesis:

Definition 3 (Optimal hypothesis). Given an ILP input $(E, B, \mathcal{H})$ and a cost function $cost_{E,B}$, the ILP problem is to find an *optimal* hypothesis $h^* = \arg\min_{h \in \mathcal{H}} \ cost_{E,B}(h)$.

In Section 4, we describe an ILP system that learns optimal hypotheses for various cost functions. In Section 5, we empirically compare the generalisation performance of optimal hypotheses based on these different cost functions.

4 Algorithm

Our algorithm builds on the learning from failures (LFF) approach to ILP (Cropper and Morel 2021). A LFF learner uses hypothesis constraints to restrict the hypothesis space. Let $\mathcal{L}$ be a meta-language that defines hypotheses. A *hypothesis constraint* is a constraint (a headless rule) expressed in $\mathcal{L}$. Let C be a set of hypothesis constraints written in a language $\mathcal{L}$. A hypothesis h is consistent with a set of constraints C if, when written in $\mathcal{L}$, h does not violate any constraint in C.

We build on the LFF algorithm Popper (Cropper and Hocquette 2023; Cropper and Morel 2021; Hocquette, Niskanen, et al. 2024). We build on Popper because it supports learning recursive hypotheses, predicate invention, and relations with any arity. Moreover, Popper is guaranteed to find an optimal hypothesis.

In the following section, we use the following notation. We assume an ILP input $(E, B, \mathcal{H})$, where $E = (E^+, E^-)$. Given a hypothesis h, a false positive is a negative example entailed by $h \cup B$. A false negative is a positive example not entailed by $h \cup B$. We denote the number of false positives and false negatives as $fp_E(h)$ and $fn_E(h)$ respectively. We consider a function $size : \mathcal{H} \rightarrow \mathbb{N}$, which evaluates the size of a hypothesis $h \in \mathcal{H}$ as the number of literals in it.

4.1 Popper

We first describe Popper (Algorithm 1) to delineate our contributions. Popper takes as input background knowledge (bk), positive (E^+) and negative (E^-) training examples, and a maximum hypothesis size (max_size) (line 1). Popper searches for a hypothesis which entails all the positive examples, no negative ones, and has minimal size. To do so, Popper builds a constraint satisfaction problem (CSP) program C, where each model of C corresponds to a hypothesis (a definite program). It uses a generate, test, combine, and constrain loop to find an optimal hypothesis. In the *generate stage*, Popper searches for a model of C for increasing hypothesis sizes (line 6). If no model is found, Popper returns the best hypothesis found thus far (lines 7-8). If a model exists, Popper converts it to a hypothesis h. In the *test stage*, Popper uses Prolog to test h on the training examples (line 9). If h entails at least one positive example ($tp > 0$) and no negative examples ($fp = 0$), Popper saves h as a *promising hypothesis* (lines 10-11). In the *combine stage*, Popper searches for a combination (a union) of promising hypotheses that entails all the positive examples and has minimal size (line 12). Popper formulates the search for an optimal combination as a MaxSAT problem. If a combination exists, Popper saves it as the best hypothesis so far and updates the maximum hypothesis size (lines 13-15). Popper does not save a hypothesis as promising if it is recursive or has predicate invention (line 10). The reason is that a combination of recursive hypotheses or hypotheses with invented predicates can entail more examples than the union of the examples entailed by each individual hypothesis. However, Popper can learn hypotheses with recursion or predicate invention as they can be output by the generate stage (line 6) and evaluated (line 9). If a hypothesis with recursion or predicate invention entails all the positive examples and no negative ones, it is saved as the best hypothesis so far (lines 16-17). In the *constrain stage*, Popper uses h to build constraints. It adds these constraints to C to prune models and thus prune the hypothesis space (line 18). For instance, if h does not entail any positive example, Popper adds a constraint to eliminate its specialisations as they are guaranteed not to entail any positive example. Popper repeats this loop until it exhausts the models of C. Popper then returns the best hypothesis found, which is guaranteed to have minimal size.

We describe our extension of Popper to support lexico-linear cost functions. We first introduce lexico-linear cost functions.

4.2 Lexico-linear Cost Functions

A lexicographic ordering defines a hierarchy of objectives, where higher-priority objectives are optimised first, and lower-priority objectives are only considered when the higher ones yield identical results. We write $Lex(c_0, \ldots, c_k)$ for the tuple $(c_0, \ldots, c_k)$ ordered lexicographically, i.e. tuples are compared by their first component, then, only if those are equal, by the second, and so on. We define a lexico-linear cost function:

Definition 4. A lexico-linear cost function is a cost function of the form:

$$cost_{E,B}(h) = Lex(c_0(h), c_1(h), c_2(h), ...)$$

Algorithm 1 Popper

```
1 def popper(bk, E+, E-, max_size):
2   cons = {}
3   promising = {}
4   best_solution = {}
5   while True:
6     h = generate_increasing_size(cons, max_size)
7     if h == UNSAT:
8       return best_solution
9     tp, fp = test(E+, E-, bk, h)
10    if tp > 0 and fp == 0 and not_rec(h) and not_pi(h):
11      promising += h
12      combine_outcome = combine(promising, max_size)
13      if combine_outcome != NO_SOLUTION:
14        best_solution = combine_outcome
15        max_size = size(best_solution)-1
16    elif tp == |E+| and fp == 0 and (rec(h) or pi(h)):
17        best_solution = h
18    cons += constrain(h, tp, fp)
19  return best_solution
```

where each c_i is a linear function of the form:

$$c_i(h) = Afp_E(h) + Bfn_E(h) + Csize(h) \text{ with } A, B, C \in \{0, 1\}.$$

4.3 Popper with Lexico-linear Cost Functions

Popper (Algorithm 1) searches for a hypothesis that entails all the positive examples, none of the negative ones, and is minimal in size. We extend Popper to learn hypotheses minimal with respect to a lexico-linear cost function (Algorithm 2). Our extension involves three main changes: (i) we generate hypotheses in any order (instead of by increasing size) for cost functions where we do not minimise the size (Algorithm 1, line 6), (ii) we change the constraints in the constraint stage (Algorithm 1, line 18), and (iii) we change the objective function in the combine stage (Algorithm 1, line 12). We describe each of these changes in turn.

4.3.1 Generate. In the generate stage (line 6), Popper searches for hypotheses by increasing size. It enforces a hard constraint on hypothesis size, and progressively increments the size during the search. We generate hypotheses in any order for cost functions that do not minimise size by simply disabling this size constraint.

4.3.2 Constrain. In the constrain stage (line 18), Popper uses hypothesis constraints to prune the hypothesis space. We base our constraints on those described by Hocquette, Niskanen, et al. (2024). We also design constraints to prune obviously redundant hypotheses for any cost function. For instance, we prune rules that do not entail any positive examples or which contain redundant literals.

4.3.3 Combine. In the combine stage (line 12), Popper searches for a combination (a union) of promising hypotheses that entails all the positive examples and is minimal in size. Instead, we search for a combination that minimises a lexico-linear cost function. We formulate the search for an optimal combination of hypotheses as a

Algorithm 2 Popper with cost functions

```
def popper_cost(bk, E+, E-, max_size, cost):
  cons = {}
  promising = {}
  best_solution, best_cost = {}, ∞
  while True:
    h = generate(cons, max_size, cost)
    if h == UNSAT:
      return best_solution
    tp, fp = test(E+, E-, bk, h)
    if tp > 0 and not_rec(h) and not_pi(h):
      promising += h
      combine_outcome = combine(promising, max_size, cost)
      if combine_outcome != NO_SOLUTION:
        best_solution = combine_outcome
        max_size = size(best_solution)-1
    elif cost(h) < best_cost and (rec(h) or pi(h)):
        best_solution = h
    cons += constrain(h, tp, fp, cost)
  return best_solution
```

MaxSAT problem (Bacchus et al. 2021). In MaxSAT, given a set of hard clauses and a set of soft clauses with an associated weight, the task is to find a truth assignment which satisfies each hard clause and minimises the sum of the weights of falsified soft clauses.

Following Hocquette, Niskanen, et al. (2024), our MaxSAT encoding is as follows. For each promising hypothesis h, we use a variable p_h to indicate whether h is in the combination. For each example $e \in E^+ \cup E^-$, we use a variable c_e to indicate whether the combination entails e. For each positive example $e \in E^+$, we include the hard clause $c_e \rightarrow \bigvee_{B \cup h \models e} p_h$ to ensure that, if the combination entails e, then at least one of the hypotheses in the combination entails e. For each negative example $e \in E^-$, we include the hard clause $\neg c_e \rightarrow \bigwedge_{B \cup h \models e} \neg p_h$ to ensure that, if the combination does not entail e, then none of the hypotheses in the combination entails e. We encode the cost function as soft clauses. For each promising hypothesis h, we include the soft clause $(\neg p_h)$ with weight $C \cdot size(h)$ to minimise the size. For each positive example $e \in E^+$, we include the soft clause (c_e) with weight A to minimise the number of false negatives. For each negative example $e \in E^-$, we include the soft clause $(\neg c_e)$ with weight B to minimise the number of false positives. We use a MaxSAT solver on this encoding. The MaxSAT solver finds an optimal hypothesis corresponding to a combination of promising hypotheses that minimises the cost function.

To minimise a lexicographic ordering, we decompose the lexicographic minimisation problem into a sequence of MaxSAT instances, each focusing on one objective. For each objective c_i, we encode a MaxSAT instance as described above. We add soft constraints to minimise c_i, and hard constraints to fix the optimal value of all previously minimised objectives c_j, with $j < i$. We do not fix the specific assignments of previous objectives to ensure that every solution achieving the minimal value of the earlier objectives is considered when optimising subsequent ones, thus guaranteeing a globally minimal lexicographic cost.

4.3.4 Cost Functions. We define 7 standard lexico-linear cost functions.

Error. *Error* minimises the total number of incorrect predictions made by a hypothesis on the training data:

$$Error(h) = fp_E(h) + fn_E(h)$$

In other words, *error* maximises training accuracy. *Error* is the default cost function of Aleph (Srinivasan 2001).

ErrorSize. *ErrorSize* minimises the lexicographic ordering of training error and hypothesis size:

$$ErrorSize(h) = Lex(fp_E(h) + fn_E(h), size(h))$$

This approach is motivated by Occam's razor and is adopted by many systems (Corapi et al. 2011; Cropper and Morel 2021; S. H. Muggleton, Lin, et al. 2015).

FnFp. *FnFp* minimises a lexicographic ordering of false negatives and false positives:

$$FnFp(h) = Lex(fn_E(h), fp_E(h))$$

This cost function is motivated by applications where missing a true positive is more costly than incorrectly identifying a false positive, such as disease detection or fraud detection (Provost et al. 1998).

FnFpSize. *FnFpSize* minimises a lexicographic ordering of false negatives, false positives, and hypothesis size:

$$FnFpSize(h) = Lex(fn_E(h), fp_E(h), size(h))$$

FpFn. *FpFn* minimises a lexicographic ordering of false positives and false negatives:

$$FpFn(h) = Lex(fp_E(h), fn_E(h))$$

This cost function is motivated by applications where false positives are more costly than false negatives, such as spam email filtering.

FpFnSize. *FpFnSize* minimises a lexicographic ordering of false positives, false negatives, and hypothesis size:

$$FpFnSize(h) = Lex(fp_E(h), fn_E(h), size(h))$$

MDL. The MDL principle (Rissanen 1978) trades off model complexity (hypothesis size) and data fit (training accuracy). The MDL cost function is used by many ILP systems (Hocquette, Niskanen, et al. 2024; Huang and A. R. Pearce 2007; Jain et al. 2021; S. H. Muggleton 1995; Quinlan 1990; Srinivasan 2001). To define it, we use the terminology of Conklin & Witten 1994. According to the MDL principle, the most probable hypothesis h for the data E is the one that minimises the complexity $L(h|E)$ of the hypothesis given the data. The MDL principle can be expressed as finding a hypothesis that minimises $L(h) + L(E|h)$, where $L(h)$ is the syntactic complexity of the hypothesis h and $L(E|h)$ is the complexity of the data when encoded using h. Following Hocquette et al. 2024, we evaluate $L(E|h)$ as the cost of encoding the exceptions to the hypothesis, i.e. the number of false positives and false negatives. We evaluate the complexity of the hypothesis $L(h)$ as its size. We define the *MDL* cost function as:

$$MDL(h) = fp_E(h) + fn_E(h) + size(h)$$

However, our encoding represents just one possible interpretation, and other encodings are possible.

In the next section, we empirically compare these seven lexico-linear cost functions using Popper.

5 Empirical Study

Our main contribution is an empirical evaluation of the generalisation performance of different cost functions. Therefore, our first research question is:

Q1. What is the impact of different cost functions on generalisation performance?

To answer **Q1**, we compare the generalisation performance of hypotheses optimised using various cost functions on a wide range of tasks and domains.

To understand circumstances where cost functions perform best, our evaluation aims to answer the question:

Q2. What is the impact of different cost functions on generalisation performance when training data is limited versus abundant?

To answer **Q2**, we compare the generalisation performance of hypotheses optimised using various cost functions on datasets with a varying number of positive training examples.

Many applications require learning from noisy data. To evaluate how well standard cost functions handle noisy data, our evaluation aims to answer the question:

Q3. What is the impact of different cost functions on generalisation performance when training data is noisy or not?

To answer **Q3**, we compare the generalisation performance of hypotheses optimised using various cost functions on noisy and non-noisy tasks.

Many ILP approaches learn textually minimal hypotheses (Corapi et al. 2011; Cropper and Morel 2021; Evans, Hernández-Orallo, et al. 2021; S. H. Muggleton, Lin, et al. 2015). To understand whether learning textually minimal hypotheses can improve generalisation performance, our evaluation aims to answer the question:

Q4. What is the impact of minimising the size of hypotheses on generalisation performance?

To answer **Q4**, we compare the generalisation performance of hypotheses with and without minimising the size across different cost functions and many tasks.

5.1 Evaluation Metrics

We describe our metrics to evaluate the generalisation performance of learned hypotheses. Given a hypothesis h and a set of examples T, a true positive is a positive example from T entailed by $h \cup B$. A true negative is a negative example from T not entailed by $h \cup B$. We denote the number of true positives and true negatives as $tp_T(h)$ and $tn_T(h)$, respectively. We assume a set T of unseen test data in the following.

Predictive accuracy. Predictive accuracy is the proportion of correct predictions on unseen test data:

$$accuracy(h) = \frac{tp_T(h) + tn_T(h)}{tp_T(h) + tn_T(h) + fp_T(h) + fn_T(h)}$$

Predictive accuracy is one of the most commonly used evaluation metrics. However, it can be misleading, particularly with imbalanced data, as it tends to favour the majority class (Provost et al. 1998).

Balanced predictive accuracy. Balanced predictive accuracy addresses the issue of imbalanced data by evaluating the average performance across both positive and negative classes:

$$balancedaccuracy(h) = \frac{1}{2}\left(\frac{tp_T(h)}{tp_T(h) + fn_T(h)} + \frac{tn_T(h)}{tn_T(h) + fp_T(h)}\right)$$

Balanced accuracy is equivalent to standard accuracy when the hypothesis performs equally well on both classes or when the data is balanced.

Precision. Precision is the proportion of positive predictions that are true positives:

$$precision(h) = \frac{tp_T(h)}{tp_T(h) + fp_T(h)}$$

Recall. Recall is the proportion of actual positive examples that are correctly identified:

$$recall(h) = \frac{tp_T(h)}{tp_T(h) + fn_T(h)}$$

5.2 Domains

We use 25 domains commonly used to evaluate ILP systems. Table 1 shows the characteristics of our domains. For datasets with predefined training/test splits, we use the original partition. Otherwise, we randomly generate a new split for each run. We briefly describe each domain.

1D-ARC. The *1D-ARC* dataset (Xu et al. 2023) is a one-dimensional adaptation of the Abstraction and Reasoning Corpus (ARC) (Chollet 2019), designed to evaluate visual abstract reasoning capabilities. We use a relational representation (Hocquette and Cropper 2024).

Alzheimer (alzh). These real-world tasks (King et al. 1995) involve learning rules describing four properties desirable in drug design for Alzheimer's disease.

ARC. The ARC dataset (Chollet 2019) evaluates the ability to perform abstract reasoning and problem-solving from a small number of examples. Each task requires transforming two-dimensional input images into corresponding output images. The tasks vary widely, including pattern recognition, geometric transformations, colour manipulation, and counting. We use the *training* subset of the original dataset[2]. We use a relational representation (Hocquette and Cropper 2024).

Carcinogenesis (carcino). The goal is to predict carcinogenic activity in rodents (Srinivasan, King, S. H. Muggleton, and M. J. E. Sternberg 1997).

Concept ARC (CA). Concept ARC (Moskvichev et al. 2023) is a set of manually designed tasks inspired by the abstraction and reasoning corpus (Chollet 2019) and organised into 16 concepts such as copying, counting, object extraction, and boundary movement.

Game policies (GP). The goal is to learn game strategies (Silver et al. 2020) for grid-world games.

Graph. We use frequently used graph problems (Evans and Grefenstette 2018; Glanois et al. 2022), including detecting graph cyclicity, determining node connectedness, and identifying improper colouring. Three of the six tasks involve learning recursive hypotheses.

IGGP. The goal of *inductive general game playing* (IGGP) (Cropper, Evans, et al. 2020) is to induce rules to explain game traces from the general game playing competition (Genesereth and Björnsson 2013).

IMDB. This widely used real-world dataset (Mihalkova et al. 2007), created from the International Movie Database (IMDB.com), contains relations between movies, actors, and directors.

KRK. The task is to learn chess patterns in the king-rook-king (*KRK*) endgame, where white has a king and a rook, and black has only a king (Hocquette and S. H. Muggleton 2020).

List functions. The list functions dataset (Rule et al. 2024; Rule 2020) evaluates human and machine concept learning ability. Each task involves identifying a function that maps input lists to output lists of natural numbers. The tasks vary in complexity, ranging from basic operations like duplication and removal to more advanced

[2]The ARC challenge uses top-3 accuracy (checking if any of three predictions are correct). However, we follow related work (R. Wang et al. 2024; Xu et al. 2023) and use top-1 accuracy.

functions involving conditional logic, arithmetic, and pattern-based reasoning. We use a relational representation (Hocquette and Cropper 2024).

NELL. Nell (Carlson et al. 2010) is a relational dataset generated by the Never Ending Language Learner (NELL). We use the sports domain, which contains information about players and teams.

PTC. The predictive toxicology challenge (PTC) (Helma et al. 2001) consists of over three hundred organic molecules labelled based on their carcinogenicity in rodents. The goal is to predict the carcinogenicity of new compounds. This challenge is the successor of the predictive toxicology evaluation (PTE) challenge.

PTE. The goal of the predictive toxicology evaluation challenge (Srinivasan, King, S. H. Muggleton, and M. J. Sternberg 1997) is to predict the carcinogenicity from molecular structure in rodents.

Satellite. The task is to learn diagnostic rules for battery faults in the power subsystem of a satellite, which consists of 40 components and 29 sensors (D. A. Pearce 1988).

Strings. The goal is to learn hypotheses to transform strings (Lin et al. 2014). This real-world dataset gathers user-provided examples from online forums and is inspired by a dataset of user-provided examples in Microsoft Excel (Gulwani 2011).

Synthesis. We use a program synthesis dataset (Cropper and Morel 2021) containing tasks that require learning from non-factual data and recursive hypotheses, both of which are challenging problems (S. H. Muggleton, Raedt, et al. 2012).

Trains. The goal is to find a hypothesis that distinguishes eastbound and westbound trains (Larson and Michalski 1977).

UMLS. The unified medical language system (UMLS) (McCray 2003) is a biomedical ontology.

UW-CSE. This real-world dataset is a knowledge base describing relations between academics, publications, and courses in the Department of Computer Science and Engineering at the University of Washington (Richardson and P. Domingos 2006).

Visual Genome (VG). Visual genome (Hudson and Manning 2019; Krishna et al. 2017) is a knowledge base of image descriptions. We filter out predicates with fewer than 12 occurrences and learn rules to describe 8 object classes.

WebKB. The world wide knowledge base dataset (Craven et al. 1998) consists of 4159 web pages from four academic domains. Each webpage is represented by its text content and hyperlinks to other pages.

WN18RR. WN18RR (Bordes et al. 2014) is a widely used real-world knowledge base from WordNet.

Yeast. The goal is to learn rules that determine whether a yeast gene encodes a protein involved in metabolism (Davis et al. 2005). The data comes from the MIPS (Munich Information Center for Protein Sequence) comprehensive yeast genome database.

Zendo. Zendo is an inductive reasoning game where one player creates a secret rule for structures, and the other players try to discover it by building and studying labelled structures. The first player to correctly identify the rule wins. Zendo is a challenging game that has attracted interest in cognitive science (Bramley et al. 2018).

5.3 Method

We train Popper with the seven cost functions described in Section 4.3.4 and evaluate the generalisation performance of the learned hypotheses. We do not use a timeout, therefore, Popper learns an optimal hypothesis (a hypothesis minimising the cost function within the search space). To reiterate, learning provably optimal hypotheses enables a direct comparison of the generalisation ability of hypotheses optimised for different cost functions.

Each task has a fixed syntactic bias, which defines the hypothesis space by constraining the form of admissible hypotheses. Each cost function is optimised with respect to this bias, and the generalisation ability of an optimal hypothesis depends on the bias. However, since we use the same bias for all cost functions, the experiment is fair.

Table 1. Description of our domains. We report the number of training examples. We average statistics over all tasks in each domain. The column *relations* indicates how many distinct predicate symbols are available in the background knowledge, while the column *facts* counts the total number of ground atoms instantiated from those predicates. Enabling recursion allows the system to learn both recursive and non-recursive hypotheses. 'Some' means that recursion is enabled only for certain tasks.

Domain	# tasks	# pos exs	# neg exs	# relations	# facts	recursion	noise	real-world
1DARC	18	38	910	27	2 209	No	No	No
alzheimer	4	354	354	32	628	No	Yes	Yes
ARC	400	157	4 846	18	1 502	No	No	No
carcinogenesis	1	130	109	31	24 745	No	Yes	Yes
concept arc	16	59	2 040	19	535	No	No	No
game policies	5	30	4 236	8	824 292	No	No	No
graph	6	20	20	5	11 103	Some	No	No
IGGP	340	2 745	375 995	64	11 191	No	No	No
IMDB	3	2 217	64 757	6	1 330	No	No	Yes
KRK	3	10	10	9	4 218	No	No	No
list functions	250	57	7 520	51	7 426	No	No	No
NELL	1	210	420	6	7 824	No	Yes	Yes
PTC	1	152	191	29	49 494	No	Yes	Yes
PTE	1	162	136	36	39 197	No	Yes	Yes
satellite	1	7	481	32	17 933	No	Yes	No
synthesis	10	100	100	108	∞	Yes	No	No
strings	328	5	0	16	∞	Yes	No	Yes
trains	4	351	424	32	8 561	No	No	No
UMLS	2	514	514	46	5 725	No	Yes	Yes
UW-CSE	1	64	24 585	12	1 365	No	Yes	Yes
visual genome	8	50	50	321	65 762	No	Yes	Yes
WebKB	1	154	594	5	1 913	No	Yes	Yes
WN18RR	2	1 031	1 031	11	85 805	No	Yes	Yes
yeast	1	974	4 092	12	99 607	No	Yes	Yes
zendo	4	100	100	16	1 000	No	No	No

Understanding how the performance of different cost functions varies with the bias is an important direction for future work.

We aggregate results within each domain to avoid the over-representation of domains with many tasks. We do not aggregate results from different domains, as domain-specific characteristics significantly impact the results.

We repeat each learning task three times and calculate the mean and standard error. We use an Intel compute node with dual 2.0 GHz Intel Xeon Gold 6138 processors, 40 CPU cores, and 192 GB of DDR4 memory. For each domain, we exclude tasks where Popper does not learn any hypothesis for all cost functions.

Reproducibility. The evaluation data and the code to reproduce the results are included as an appendix and will be made publicly available if the manuscript is accepted for publication.

5.4 Results

5.4.1 Q1: What Is the Impact of Different Cost Functions on Generalisation Performance? Figure 1 shows how often each cost function ranks 1st, 2nd, or 3rd across our domains. It shows that *ErrorSize*, *MDL*, and *Error* are the best-performing cost functions overall. Each ranks in the top three in the majority of domains (at least 15/25). Among these, *ErrorSize* is the best-performing, ranking first in the majority of domains (15/25) and top three in 20/25 domains. *MDL* is the second-best-performing one, ranking first in 10/25 domains and top three in 15/25

domains. *Error* performs well but is not as dominant. It achieves first rank in 7/25 domains and top three in 16/25 domains[3]. By contrast, cost functions that separately minimise false positives and false negatives (*FpFn*, *FpFnSize*, *FnFp*, and *FnFpSize*) generally perform worse, ranking in top three in fewer than 12 domains each. This result indicates that *ErrorSize* and *MDL* are the most reliable default choices when there is little prior knowledge about the domain. Notably, the better performance of these cost functions suggests that minimising overall error is generally more effective than balancing false positives and false negatives separately.

However, no single cost function consistently outperforms the others across all domains, suggesting that the suitability of a cost function highly depends on the specific characteristics of the domain. This result highlights the importance of selecting a cost function that aligns with these domain-specific characteristics. For instance, for the program synthesis task *evens*, *Error* and *ErrorSize* achieve perfect predictive accuracy (100%), while *MDL* achieves only 85%. Conversely, *MDL* outperforms the other cost functions with 71% accuracy on *alzheimer*. *Error* and *ErrorSize* achieve 83% on *nell* while *FpFn* and *FpFnSize* achieve 50%. However, *FpFn* and *FpFnSize* outperform the other cost functions on the *list* domain, where there are many negative examples.

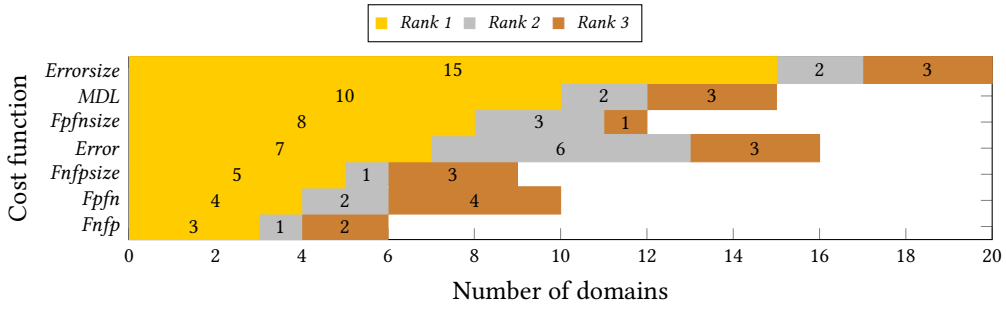

Fig. 1. Ranking of cost functions based on average predictive accuracy across domains. Ties occur when multiple cost functions achieve the same average performance within a domain.

Figure 2 shows the performance of the different cost functions. It shows *Error* and *ErrorSize* are the best-performing cost functions overall, achieving average predictive accuracies of 88% and 87% respectively, compared to 83% for the third-best option, *MDL*. A Wilcoxon signed-rank test confirms that the difference between either *Error*, *ErrorSize*, or *MDL* and the four other cost functions is statistically significant ($p < 0.01$). By contrast, cost functions that separately minimise false positives and false negatives (*FpFnSize*, *FpFn*, *FnFpSize*, and *FnFp*) have greater variability in performance and broader spread in accuracy, suggesting they are more sensitive to dataset characteristics. This result suggests that *Error* and *ErrorSize* are the most reliable default choices when limited information about the dataset is available. Moreover, the difference between cost functions highlights the importance of selecting an appropriate cost function based on domain characteristics.

To account for unbalanced testing sets, Figure 2 shows the balanced predictive accuracies. It shows that *ErrorSize* and *Error* are again the best-performing cost functions, while *MDL* achieves the lowest balanced accuracy among all cost functions. Balanced accuracy shows more variation than accuracy, indicating that the cost functions tested lead to models that perform well overall but may be biased toward majority classes.

Figure 2 shows the precision. It shows that *MDL*, *FpFnSize*, and *FpFn* achieve the highest precision on average. A Wilcoxon signed-rank test confirms that the difference in precision between *MDL* and the other cost functions, *FpFn* and the other cost functions, and between *FpFnSize* and the other cost functions is statistically significant ($p < 0.01$). They achieve consistently high precision, with many domains clustered near 100% and relatively narrow

[3]Ties occur when multiple cost functions achieve the same average performance within a domain.

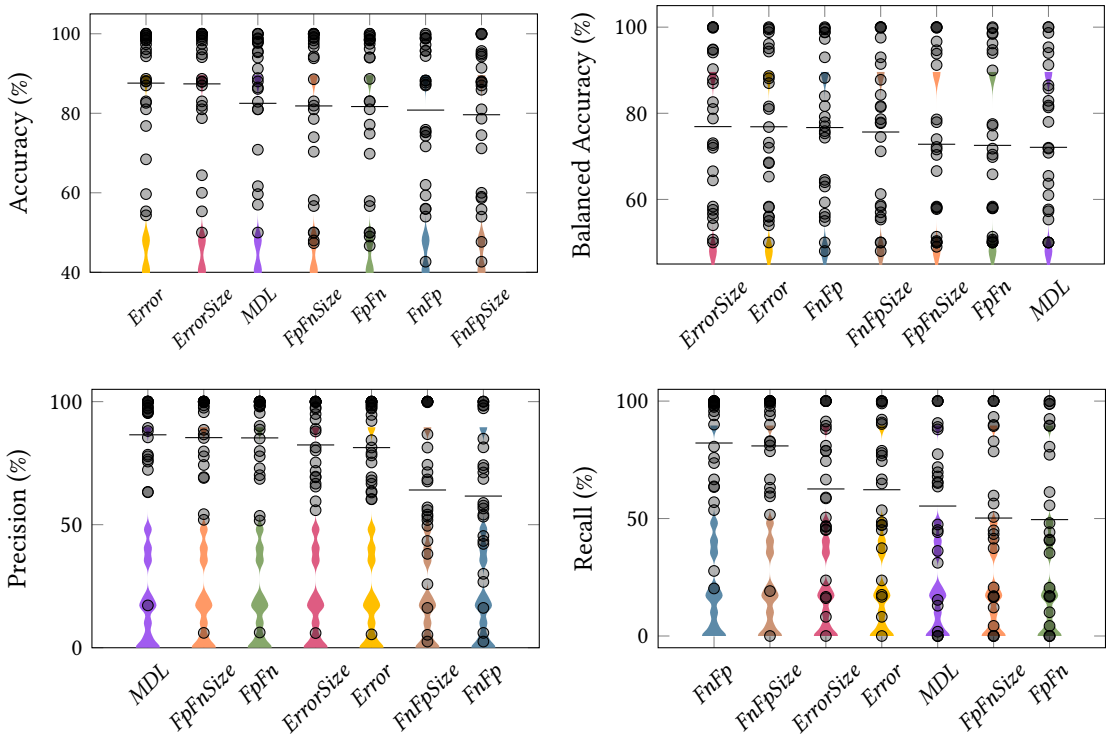

Fig. 2. Evaluation over all domains. Each dot represents the average performance of all tasks with one domain. The horizontal bar represents the overall mean across all domains. Within each panel, cost functions are ordered from bottom to top by overall mean.

distributions, suggesting reliable performance. This result is not surprising as *FpFn* and *FpFnSize* minimise the number of false positives in priority. These cost functions are cautious. In particular, since the empty hypothesis has no false positives, a hypothesis is considered only if it also has no false positives. Therefore, in some cases, such as for *UW-CSE*, the optimal hypothesis learned is the empty hypothesis, which has default predictive accuracy. *FnFp* and *FnFpSize* have low precision. *MDL* also is cautious and only learns a rule if it compresses the positive examples. This prevents it from finding overly general hypotheses, which could lead to false positives.

Figure 2 shows the recall. It shows that *FnFp* and *FnFpSize* achieve the highest recall on average. A Wilcoxon signed-rank test confirms that the difference in recall between *FnFp* and the other cost functions, and between *FnFpSize* and the other cost functions are statistically significant ($p < 0.01$). These cost functions minimise in priority the number of false negatives. Therefore, these cost functions tend to find over-general hypotheses with high recall. By contrast, *FpFn* and *FpFnSize* have low recall. Minimising false positives can be at the expense of incorrectly labelling positive examples as negative, leading to a low recall compared to the other cost functions. *Error*, *ErrorSize*, and *MDL* have lower recall than precision. Since most datasets we use have more training negative examples than positive examples, the ILP system may learn an overly specific hypothesis.

Figure 3 shows the correlation of the predictive accuracies of the cost functions tested. It shows that the cost functions are positively correlated, with moderate to strong correlation coefficients. Notably, *Error* has a strong correlation with *ErrorSize*, *FpFn* with *FpFnSize*, and *FnFp* with *FnFpSize*, all having correlation coefficients greater

than 0.81. This result suggests they tend to perform similarly on a given dataset. We discuss this result further in Section 5.4.4.

MDL has the lowest correlations with other cost functions, with coefficients below 0.65. This result indicates that *MDL* better suits different applications than the other cost functions evaluated. Given this difference in performance, it may be worthwhile to try *MDL* in addition to another cost function, as they could yield different results.

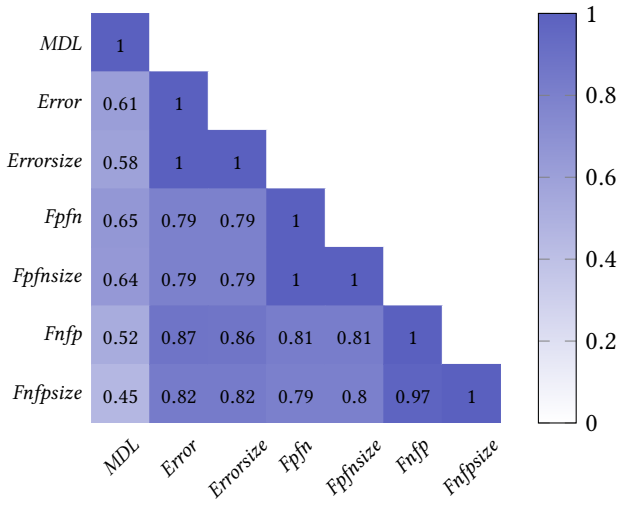

Fig. 3. Correlation matrix of the predictive accuracies across cost functions

Overall, these results suggest that the answer to **Q1** is that *ErrorSize* and *MDL* are the best-performing cost functions overall. This result suggests that minimising overall error is generally more effective than balancing false positives and false negatives separately. Moreover, *ErrorSize* and *MDL* show only a weak positive correlation, indicating that they perform well across different datasets.

5.4.2 Q2: What Is the Impact of Different Cost Functions on Generalisation Performance When Training Data Is Limited versus Abundant? Figure 4 shows the predictive accuracies versus the number of positive training examples. When the training data contains fewer than 100 positive examples, the cost functions *FnFp* and *FnFpSize* achieve the highest predictive accuracies. A Wilcoxon signed-rank test confirms that the difference in accuracy between *FnFp* or *FnFpSize* and the other cost functions is statistically significant ($p < 0.01$) with fewer than 100 examples. These functions prioritise minimising false negatives, which helps ensure that as many positive examples as possible are covered. Therefore, they generalise better with limited data. Moreover, *ErrorSize* and *FnFpSize* perform better than *Error* and *FnFp*, respectively with fewer than 10 positive training examples, which suggests that minimising the size improves performance with limited data. By contrast, *MDL* struggles in low-data settings (fewer than 10 positive examples), as it cannot identify a hypothesis that compresses the data. A Wilcoxon signed-rank test confirms that the difference in accuracy between *MDL* and the other cost functions is statistically significant ($p < 0.01$) with fewer than 10 examples. For instance, *MDL* does not learn any hypothesis for the *satellite* task, which contains 9 positive training examples and struggles with *strings*, which contains 5 positive training examples per task. This result highlights a fundamental limitation of compression-based approaches in data-sparse scenarios: without enough training examples, there are insufficient regularities for

effective compression. This observation corroborates prior theoretical and empirical results on MDL (Kearns et al. 1995).

By contrast, when the training data contains more than 100 positive examples, *MDL* outperforms the other cost functions. With more positive examples, *MDL* can better compress the training data, leading to better performance. Meanwhile, the performance of *FnFp* and *FnFpSize* declines as the number of positive examples increases. Once many positive examples are covered, the benefit of further minimising false negatives is less impactful, and minimising false positives and false negatives jointly yields better performance.

Overall, these results suggest that the answer to **Q2** is that the performance of a cost function depends on the amount of available training data. *FnFp* and *FnFpSize* are more effective than the other cost functions when the training data is limited, while *MDL* outperforms the others with many positive examples.

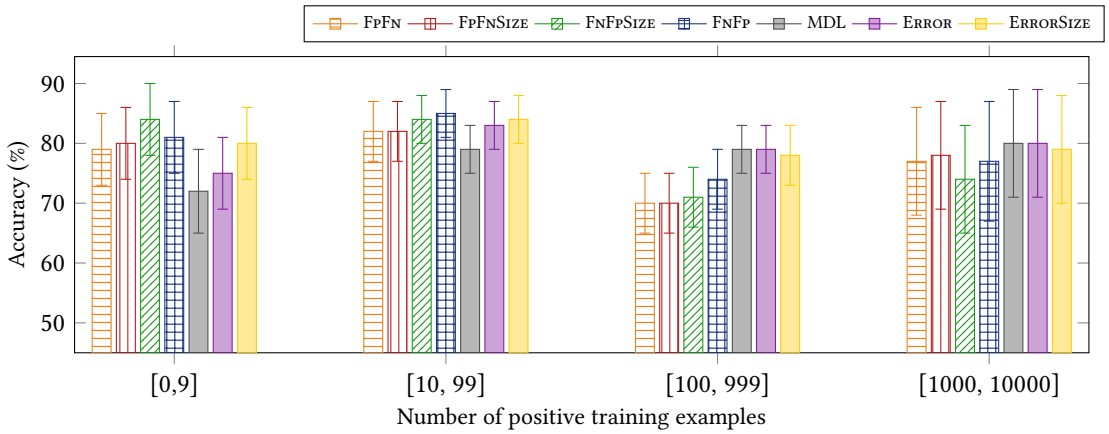

Fig. 4. Predictive accuracies versus the number of positive training examples.

5.4.3 Q3: What Is the Impact of Different Cost Functions on Generalisation Performance When Training Data Is Noisy or Not? Figure 5 shows the performance with noisy data. It shows that *Error*, *ErrorSize*, and *MDL* outperform the other cost functions with noisy data overall. The difference in performance between these cost functions and those that separately minimise false positives and false negatives (*FpFn*, *FpFnSize*, *FnFp*, and *FnFpSize*) is greater than in Figure 2, which suggests that minimising overall error, rather than false positives and false negatives separately, is more important for noisy data. A Wilcoxon signed-rank test confirms that the difference between either *Error*, *ErrorSize*, or *MDL* and the 4 other cost functions is statistically significant ($p < 0.01$).

The cost function *MDL* outperforms *Error* and *ErrorSize* on *WN18RR*, *alzheimer*, *carcinogenesis*, *PTC*, and *PTE*, which corroborates the results of Hocquette et al. (Hocquette, Niskanen, et al. 2024). These datasets are characterised by a balanced and moderately large number of training examples. However, *Error* and *ErrorSize* outperform *MDL* on *NELL*, *UW-CSE*, *satellite*, and *visualgenome*. For instance, for *UW-CSE*, the hypothesis learned with *ErrorSize* entails 5 positive training examples and 2 negative training examples and has size 5. Since this hypothesis does not compress the positive training examples, *MDL* does not learn it. This result highlights a limitation of *MDL*, which performs poorly when the number of positive training examples is small, and compression is not feasible.

The cost functions *FpFn* and *FpFnSize* have the lowest balanced accuracy on noisy data. These cost functions prioritise minimising the number of false positives. Therefore, hypotheses where the number of false positives is only near-minimal are pruned from the hypothesis space. Since the empty hypothesis has no false positive, *FpFn*

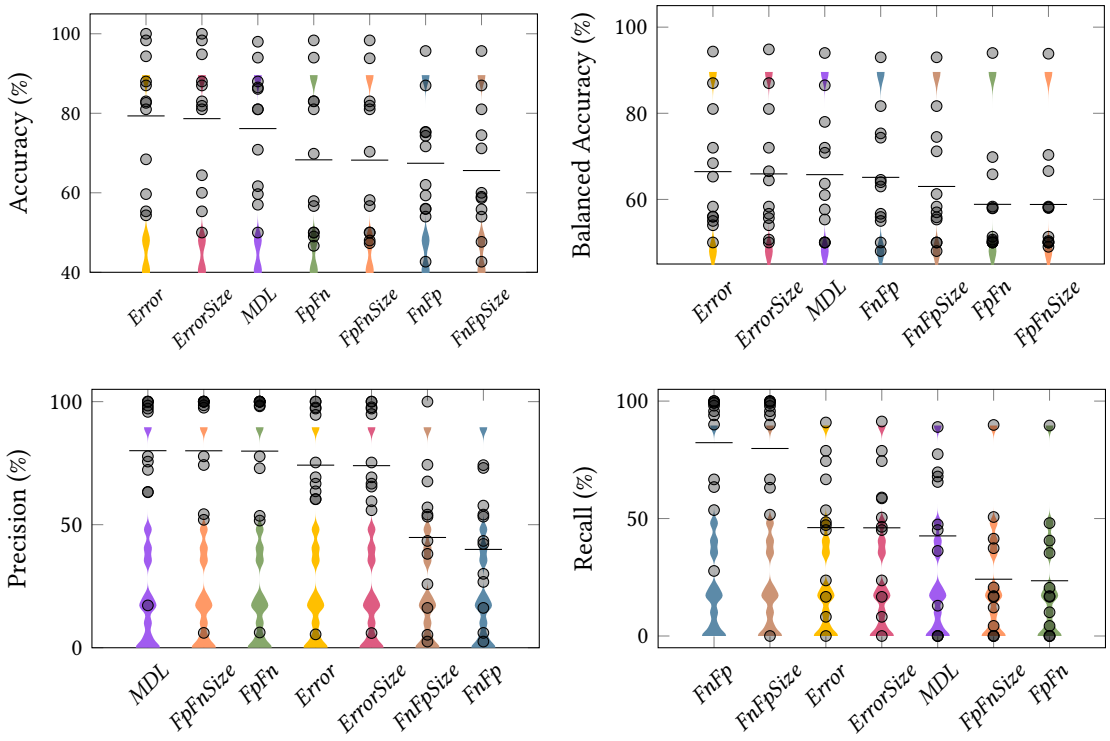

Fig. 5. Evaluation over noisy domains. Each dot represents the average performance of all tasks with one domain. The horizontal bar represents the overall mean across all noisy domains.

and *FpFnSize* only consider hypotheses without any false positive. As a result, they often learn overly specific hypotheses that do not generalise. Sometimes, as observed for *UWCSE* and *NELL*, these cost functions default to the empty hypothesis, which has no generalisation ability. By contrast, the cost functions *MDL*, *ErrorSize*, and *Error* trade off false positives and false negatives and, therefore, tolerate some false positives if it leads to a bigger reduction in false negatives.

Similarly, *FnFp* and *FnFpSize* have the lowest accuracy on noisy data. These cost functions prioritise minimising the number of false negatives. Therefore, they try to find a hypothesis that generalises as many positive examples as possible, including noisy ones, at the risk of overgeneralisation.

Overall, these results suggest that the answer to **Q3** is that *Error*, *ErrorSize*, and *MDL* tend to perform better on noisy data.

5.4.4 Q4: What Is the Impact of Minimising the Size of Hypotheses on Generalisation Performance? Figure 3 shows the correlation between the cost functions evaluated. It shows that *Error* is strongly correlated with *ErrorSize* (with a correlation coefficient $\rho = 1.00$), *FpFn* with *FpFnSize* ($\rho = 1.00$), and *FnFp* with *FnFpSize* ($\rho = 0.97$). These high correlation coefficients suggest that adding a size-based penalty does not substantially change the results for domains where cost functions perform well. This result is expected because the lexicographic ordering prioritises minimising the primary cost components before considering the size for the cost functions evaluated.

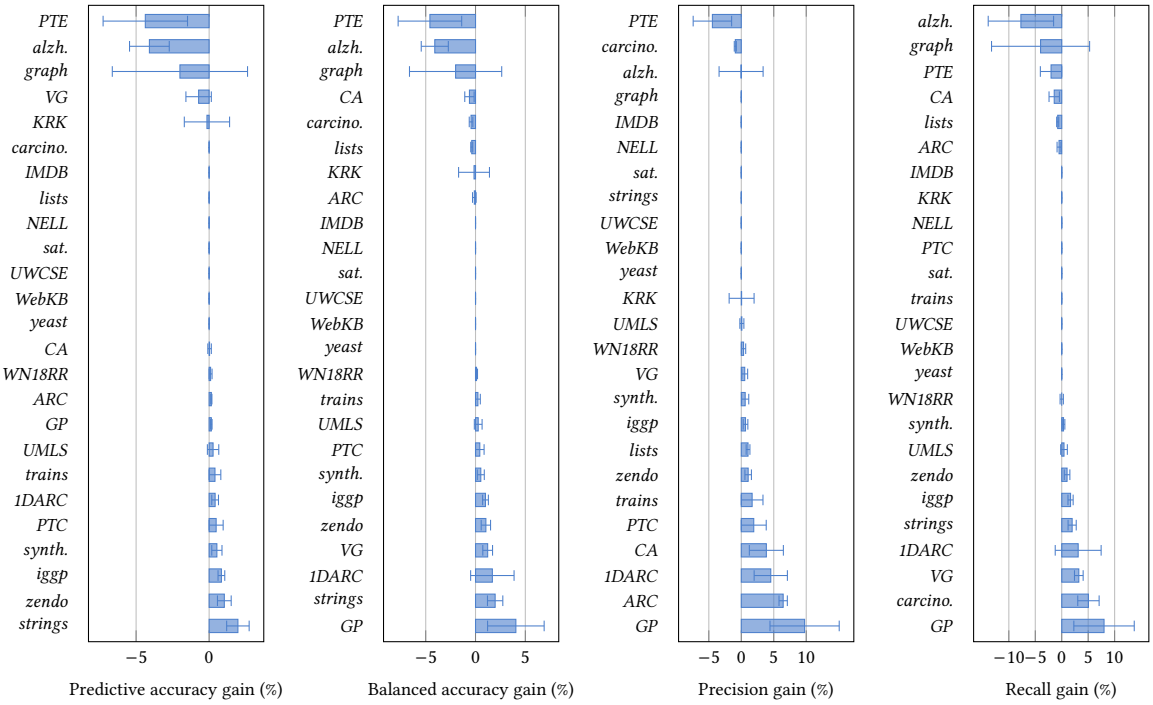

Fig. 6. Gain of minimising the size for the cost function *Error*.

Figures 6, 7, and 8 show the loss/gain of minimising the size for the cost functions *Error*, *FnFp*, and *FpFn*, respectively. They show that minimising the size can either improve or degrade predictive accuracy, depending on the domain. For instance, with the cost function *Error*, minimising the size degrades the predictive accuracy by 4% for *alzheimer*, but improves it by 1% for *zendo*. Similarly, with the cost function *FnFp*, minimising the size degrades the predictive accuracy by 8% for *NELL*, but improves it by 3% for *strings*.

A possible explanation for this variation is the nature of the datasets. Benchmarks are either synthetic (artificially constructed) or derived from real-world data. Synthetic datasets, such as *zendo*, *1DARC*, *synthesis*, or *trains*, are often designed to contain simple underlying relationships, making smaller hypotheses more suitable. In these cases, a bias towards simplicity can improve performance. Conversely, real-world datasets, such as *NELL*, *WebKB*, or *carcinogenesis*, tend to involve more complex relationships and noise. In these cases, a preference for smaller models does not impact performance and can be detrimental. A Wilcoxon signed-rank test confirms that the difference between either *Error* and *ErrorSize*, *FnFp* and *FnFpSize*, *FpFn* and *FpFnSize* is statistically significant on synthetic data but not on real-world data ($p<0.01$). This result suggests that the impact of minimising the size is determined by the extent to which this simplicity bias fits with the dataset's characteristics rather than any inherent advantage of size minimisation itself. As noted by Schaffer (1993), selecting a cost function introduces a bias, and its effect on performance depends on how well this bias fits the domain, rather than on any intrinsic advantage of the cost function. When simple models are more appropriate to model the data, any method that introduces a bias toward simplicity is likely to improve performance. This observation aligns with the classical bias-variance trade-off. Penalising large hypotheses introduces a bias towards simpler models, which can be beneficial when the underlying relationships are simple. However, this bias may lead to underfitting, reducing

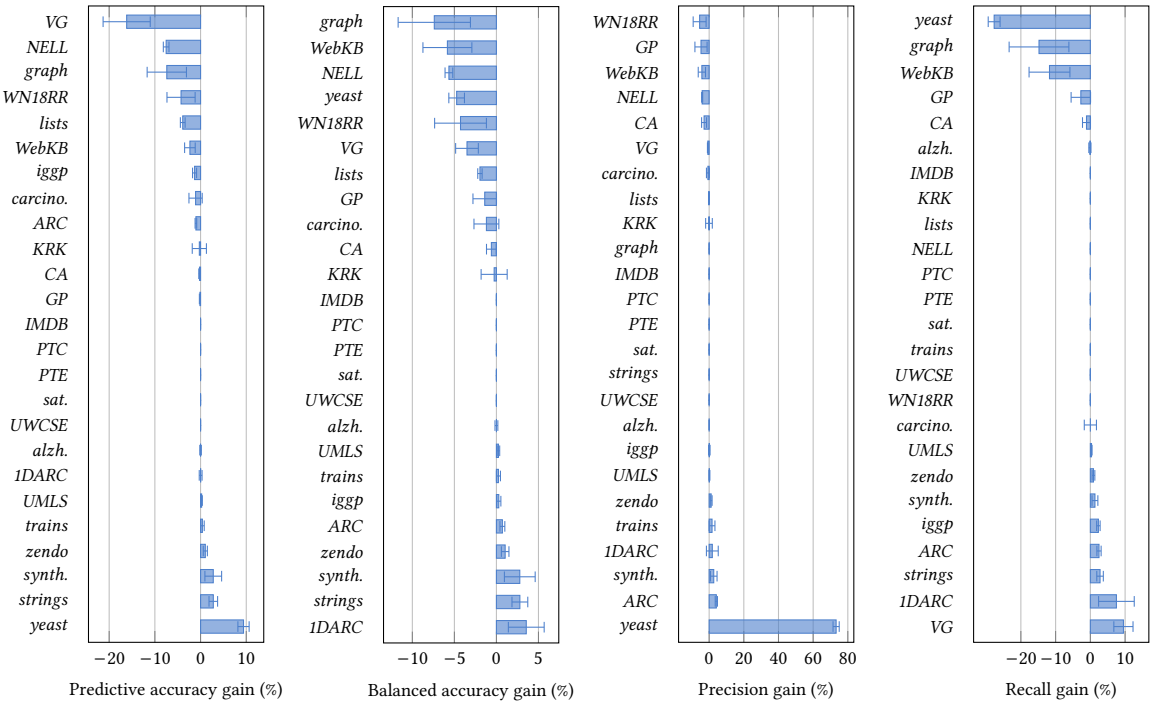

Fig. 7. Gain of minimising the size for the cost function *FnFp*.

predictive accuracy. This trade-off highlights the importance of selecting a cost function that balances bias and variance according to the complexity of the domain.

Minimising the size is also helpful for learning recursive hypotheses. For instance, for the *strings* domain, it improves predictive accuracy by 1% for *FpFn*, 2% for *Error*, and 3% for *FnFp*. Size minimisation encourages more concise recursive definitions, and recursion allows to write compact hypotheses that generalise better.

Minimising the size does not affect performance in many domains. For instance, minimising the size does not impact predictive accuracies for *lists* for *Error* and *FpFn*. Popper finds longer hypotheses with rules that subsume each other. In other words, rules include redundant literals that do not affect their overall coverage. However, finding shorter hypotheses can have other advantages, such as better interpretability.

Overall, these results suggest that minimising the size often plays a secondary role compared to the choice of cost function itself (e.g. *Error* versus *FnFp*). The answer to **Q4** is that minimising the size can either degrade or improve learning performance depending on the complexity of the relationships within the data. It typically improves performance with synthetic data. Moreover, minimising the size helps learn recursive hypotheses.

6 Conclusion

We conducted a large-scale empirical comparison of seven standard cost functions in ILP across more than 20 domains and 1,000 tasks. To conduct our comparison, we extended a constraint-based ILP system to learn optimal hypotheses for each of these cost functions. Our results show the following insights:

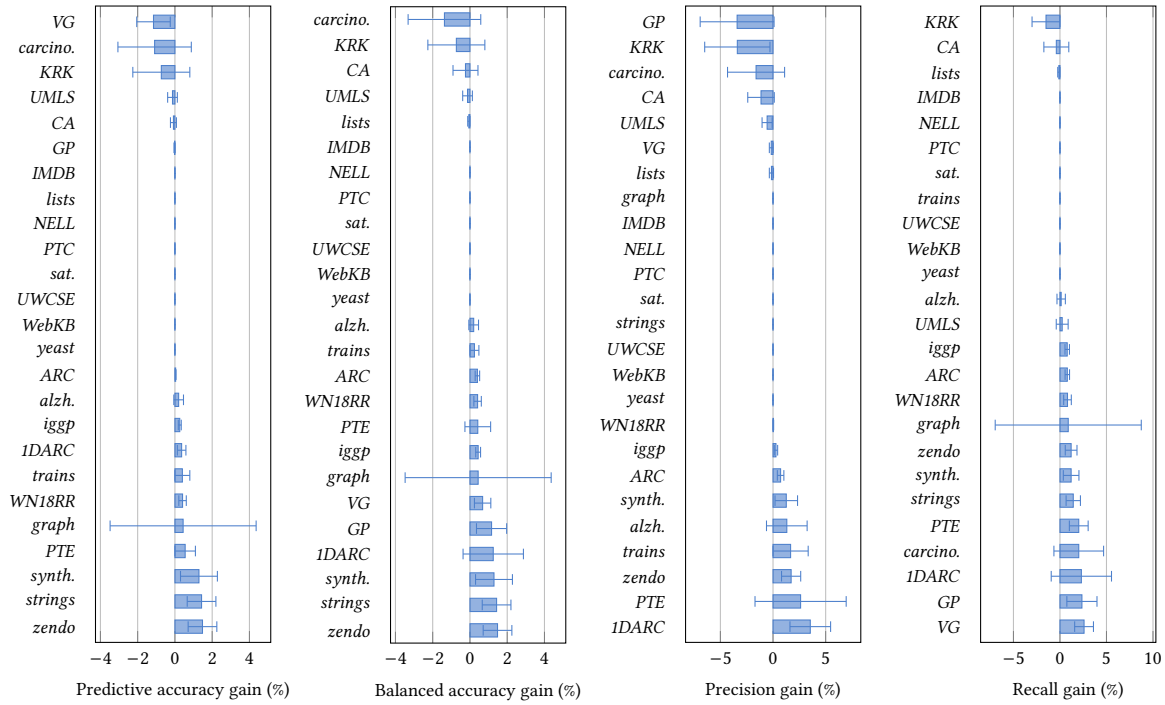

Fig. 8. Gain of minimising the size for the cost function *FpFn*.

(1) *ErrorSize*, *MDL*, and *Error* are the best-performing cost functions overall. This result suggests that minimising overall error is generally more effective than balancing false positives and false negatives separately.
(2) No single cost function consistently outperforms the others across all domains, which indicates that the effectiveness of a cost function is highly dependent on the domain characteristics. In other words, cost functions act as a form of bias, and their effect on performance is determined by the degree to which this bias aligns with the domain's characteristics rather than by any inherent advantage of the cost function.
(3) *MDL*, *FpFn*, and *FpFnSize* achieve the highest precision, while *FnFp* and *FnFpSize* achieve the highest recall.
(4) While all cost functions show moderate to strong positive correlations, *MDL* shows the lowest correlations with the other cost functions. This result indicates that *MDL* may be better suited for domains where the other cost functions are less effective.
(5) *FnFp* and *FnFpSize* perform better when the positive training data is limited, whereas *MDL* outperforms others with a large amount of training data.
(6) *Error* performs best on noisy data, followed by *ErrorSize* and *MDL*.
(7) Minimising the size of hypotheses often plays a secondary role compared to the choice of cost function itself (e.g. *Error* versus *FnFp*). While minimising hypothesis size can improve generalisation performance in some contexts, particularly when learning recursive hypotheses or with synthetic data, it does not improve performance on real-world data and can sometimes degrade it.

We hope that these insights will guide users in selecting cost functions that best align with the specific characteristics of a given dataset.

Practical Recommendations. Based on our results, we recommend selecting *MDL* or *FpFn* when minimising false positives is prioritised over false negatives, and *FnFp* when minimising false negatives is prioritised over false positives. We recommend trying both *Error* and *FnFp* if the number of positive training examples is limited, and both *Error* and *MDL* when it is large. Additionally, minimising the size of a hypothesis is particularly important when learning recursive hypotheses or working with synthetic data. Figure 9 summarises these recommendations.

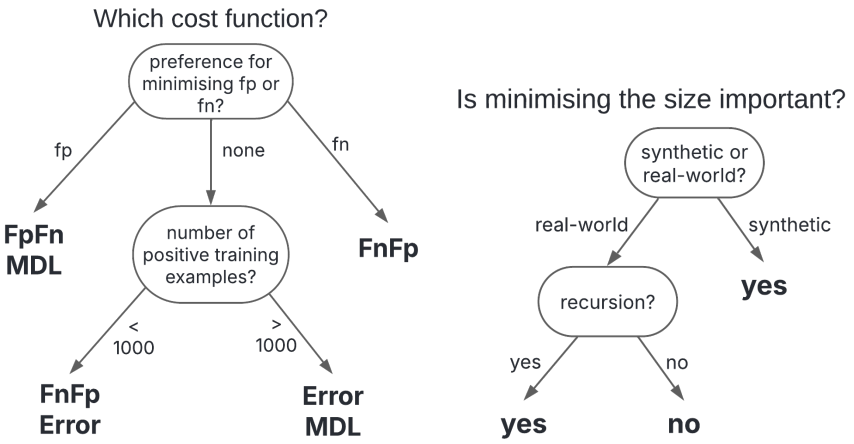

Fig. 9. Summary of our results.

Limitations and Future Work

Theoretical analysis. While our study spans many diverse domains, our results remain inherently empirical. A promising direction for future work is to develop a theoretical framework, grounded in computational learning theory, to validate this empirical analysis.

Weighted cost functions. Future work should investigate whether minimising a weighted sum of the number of false positives and false negatives improves performance on unbalanced datasets.

More datasets. While our large-scale evaluation reveals consistent trends across many domains, it cannot guarantee stability under all conditions.

More cost functions. We evaluated seven commonly used cost functions. However, it is possible that untested cost functions provide better performance. Future work should evaluate other cost functions, including novelty (Lavrač et al. 1999). We hope our results highlight the strengths and weaknesses of standard cost functions in relational program synthesis, and stimulate further research in the development of better-performing cost functions.

Adaptative cost functions. A natural next step is to develop methods that adaptively select or combine cost functions based on task characteristics. One promising direction is to design a meta-cost function that adaptively combines several cost functions through meta-learning approaches. Another is to explore multi-objective optimisation that jointly optimise competing criteria (eg: training accuracy, size, and compression) rather than enforcing a fixed lexicographic order. Together, these approaches could enable ILP systems to learn context-aware optimisation strategies that flexibly adapt to the characteristics of each domain.

Acknowledgments

Both authors were supported by the EPSRC fellowship (*EP/V040340/1*).

References

J. Ahlgren and S. Y. Yuen. 2013. "Efficient program synthesis using constraint satisfaction in inductive logic programming." *J. Machine Learning Res.*, 14, 1, 3649–3682. http://dl.acm.org/citation.cfm?id=2627674.

S. Ameen and L. H. Lelis. 2023. "Program synthesis with best-first bottom-up search." *Journal of Artificial Intelligence Research*, 77, 1275–1310.

F. Bacchus, M. Järvisalo, and R. Martins. 2021. "Maximum Satisfiabiliy." In: *Handbook of Satisfiability - Second Edition*. Frontiers in Artificial Intelligence and Applications. Vol. 336. Ed. by A. Biere, M. Heule, H. van Maaren, and T. Walsh. IOS Press, 929–991. doi:10.3233/FAIA201008.

S. Barke, H. Peleg, and N. Polikarpova. 2020. "Just-in-time learning for bottom-up enumerative synthesis." *Proceedings of the ACM on Programming Languages*, 4, OOPSLA, 1–29.

R. J. Bayardo Jr. and R. Agrawal. 1999. "Mining the Most Interesting Rules." In: *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 15-18, 1999*. Ed. by U. M. Fayyad, S. Chaudhuri, and D. Madigan. ACM, 145–154. doi:10.1145/312129.312219.

A. Bembenek, M. Greenberg, and S. Chong. 2023. "From SMT to ASP: Solver-Based Approaches to Solving Datalog Synthesis-as-Rule-Selection Problems." *Proc. ACM Program. Lang.*, 7, POPL, 185–217. doi:10.1145/3571200.

C. Bessiere, E. Hebrard, and B. O'Sullivan. 2009. "Minimising decision tree size as combinatorial optimisation." In: *International Conference on Principles and Practice of Constraint Programming*. Springer, 173–187.

C. Bessiere, F. Koriche, N. Lazaar, and B. O'Sullivan. 2017. "Constraint acquisition." *Artificial Intelligence*, 244, 315–342.

H. Blockeel and L. D. Raedt. 1998. "Top-Down Induction of First-Order Logical Decision Trees." *Artif. Intell.*, 101, 1-2, 285–297. doi:10.1016/S0004-3702(98)00034-4.

A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. 1987. "Occam's Razor." *Inf. Process. Lett.*, 24, 6, 377–380. doi:10.1016/0020-0190(87)90114-1.

A. Bordes, X. Glorot, J. Weston, and Y. Bengio. 2014. "A semantic matching energy function for learning with multi-relational data - Application to word-sense disambiguation." *Mach. Learn.*, 94, 2, 233–259. doi:10.1007/S10994-013-5363-6.

N. Bramley, A. Rothe, J. Tenenbaum, F. Xu, and T. Gureckis. 2018. "Grounding compositional hypothesis generation in specific instances." In: *Proceedings of the 40th annual conference of the cognitive science society*.

A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. H. Jr., and T. M. Mitchell. 2010. "Toward an Architecture for Never-Ending Language Learning." In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. Ed. by M. Fox and D. Poole. AAAI Press, 1306–1313. doi:10.1609/AAAI.V24I1.7519.

F. Chollet. 2019. "On the Measure of Intelligence." *CoRR*, abs/1911.01547. http://arxiv.org/abs/1911.01547 arXiv: 1911.01547.

D. Conklin and I. H. Witten. 1994. "Complexity–Based Induction." *Mach. Learn.*, 16, 203–225.

D. Corapi, A. Russo, and E. Lupu. 2011. "Inductive Logic Programming in Answer Set Programming." In: *Inductive Logic Programming - 21st International Conference, ILP 2011* (Lecture Notes in Computer Science). Vol. 7207. Springer, 91–97. doi:10.1007/978-3-642-31951-8_12.

M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. 1998. "Learning to extract symbolic knowledge from the World Wide Web." *AAAI/IAAI*, 3, 3.6, 2.

A. Cropper and S. Dumancic. 2022. "Inductive Logic Programming At 30: A New Introduction." *J. Artif. Intell. Res.*, 74, 765–850. doi:10.1613/jair.1.13507.

A. Cropper and S. Dumančic. 2021. "Learning large logic programs by going beyond entailment." In: *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 2073–2079.

A. Cropper, R. Evans, and M. Law. 2020. "Inductive general game playing." *Mach. Learn.*, 109, 1393–1434.

A. Cropper and C. Hocquette. 2023. "Learning Logic Programs by Combining Programs." In: *ECAI 2023 - 26th European Conference on Artificial Intelligence*. Vol. 372. IOS Press, 501–508. doi:10.3233/FAIA230309.

A. Cropper and R. Morel. 2021. "Learning programs by learning from failures." *Mach. Learn.*, 110, 4, 801–856. doi:10.1007/s10994-020-05934-z.

J. Davis, E. Burnside, I. de Castro Dutra, D. Page, and V. S. Costa. 2005. "An Integrated Approach to Learning Bayesian Networks of Rules." In: *Machine Learning: ECML 2005*. Ed. by J. Gama, R. Camacho, P. B. Brazdil, A. M. Jorge, and L. Torgo. Springer Berlin Heidelberg, Berlin, Heidelberg, 84–95. ISBN: 978-3-540-31692-3.

T. G. Dietterich. 1995. "Overfitting and Undercomputing in Machine Learning." *ACM Comput. Surv.*, 27, 3, 326–327. doi:10.1145/212094.212114.

P. M. Domingos. 1999. "The Role of Occam's Razor in Knowledge Discovery." *Data Min. Knowl. Discov.*, 3, 4, 409–425. doi:10.1023/A:1009868929893.

S. Esmeir and S. Markovitch. 2007. "Occam's Razor Just Got Sharper." In: *IJCAI*. Citeseer, 768–773.

R. Evans and E. Grefenstette. 2018. "Learning Explanatory Rules from Noisy Data." *J. Artif. Intell. Res.*, 61, 1–64. doi:10.1613/jair.5714.

R. Evans, J. Hernández-Orallo, J. Welbl, P. Kohli, and M. Sergot. 2021. "Making sense of sensory input." *Artificial Intelligence*, 293, 103438.

C. Ferri, J. Hernández-Orallo, and R. Modroiu. 2009. "An experimental comparison of performance measures for classification." *Pattern Recognit. Lett.*, 30, 1, 27–38. doi:10.1016/J.PATREC.2008.08.010.

J. Fürnkranz and P. A. Flach. 2005. "Roc 'n'rule learning—towards a better understanding of covering algorithms." *Mach. Learn.*, 58, 39–77.

M. R. Genesereth and Y. Björnsson. 2013. "The International General Game Playing Competition." *AI Mag.*, 34, 2, 107–111. doi:10.1609/AIMAG.V34I2.2475.

L. Geng and H. J. Hamilton. 2006. "Interestingness measures for data mining: A survey." *ACM Comput. Surv.*, 38, 3, 9. doi:10.1145/1132960.1132963.

C. Glanois, Z. Jiang, X. Feng, P. Weng, M. Zimmer, D. Li, W. Liu, and J. Hao. 2022. "Neuro-Symbolic Hierarchical Rule Induction." In: *International Conference on Machine Learning, ICML 2022*. Vol. 162. PMLR, 7583–7615.

S. Gulwani. 2011. "Automating string processing in spreadsheets using input-output examples." In: *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*. Ed. by T. Ball and M. Sagiv. ACM, 317–330. doi:10.1145/1926385.1926423.

S. Handa and M. C. Rinard. 2020. "Inductive program synthesis over noisy data." In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 87–98.

C. Helma, R. D. King, S. Kramer, and A. Srinivasan. 2001. "The predictive toxicology challenge 2000–2001." *Bioinformatics*, 17, 1, 107–108.

C. Hocquette and A. Cropper. 2024. "Relational decomposition for program synthesis." In: *arXiv, 2408.12212*. https://arxiv.org/abs/2408.12212.

C. Hocquette and S. H. Muggleton. 2020. "Complete Bottom-Up Predicate Invention in Meta-Interpretive Learning." In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*. ijcai.org, 2312–2318. doi:10.24963/ijcai.2020/320.

C. Hocquette, A. Niskanen, M. Järvisalo, and A. Cropper. 2024. "Learning MDL Logic Programs from Noisy Data." In: *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*. Ed. by M. J. Wooldridge, J. G. Dy, and S. Natarajan. AAAI Press, 10553–10561. doi:10.1609/AAAI.V38I9.28925.

J. Huang and A. R. Pearce. 2007. "Collaborative Inductive Logic Programming for Path Planning." In: *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*. Ed. by M. M. Veloso, 1327–1332. http://ijcai.org/Proceedings/07/Papers/214.pdf.

D. A. Hudson and C. D. Manning. 2019. "Gqa: A new dataset for real-world visual reasoning and compositional question answering." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 6700–6709.

A. Jain, C. Gautrais, A. Kimmig, and L. D. Raedt. 2021. "Learning CNF Theories Using MDL and Predicate Invention." In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021*. ijcai.org, 2599–2605. doi:10.24963/ijcai.2021/358.

F. Janssen and J. Fürnkranz. 2010. "On the quest for optimal rule learning heuristics." *Mach. Learn.*, 78, 3, 343–379. doi:10.1007/s10994-009-5162-2.

T. Kaminski, T. Eiter, and K. Inoue. 2019. "Meta-Interpretive Learning Using HEX-Programs." In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. Ed. by S. Kraus. ijcai.org, 6186–6190. doi:10.24963/ijcai.2019/860.

M. Kearns, Y. Mansour, A. Y. Ng, and D. Ron. 1995. "An experimental and theoretical comparison of model selection methods." In: *Proceedings of the Eighth Annual Conference on Computational Learning Theory*, 21–30.

R. D. King, M. J. Sternberg, and A. Srinivasan. 1995. "Relating chemical activity to structure: an examination of ILP successes." *New Gener. Comput.*, 13, 411–433.

R. Krishna et al.. 2017. "Visual genome: Connecting language and vision using crowdsourced dense image annotations." *International journal of computer vision*, 123, 32–73.

J. Larson and R. S. Michalski. 1977. "Inductive inference of VL decision rules." *SIGART Newsletter*, 63, 38–44. doi:10.1145/1045343.1045369.

N. Lavrač, P. Flach, and B. Zupan. 1999. "Rule evaluation measures: A unifying view." In: *International Conference on Inductive Logic Programming*. Springer, 174–185.

M. Law, A. Russo, E. Bertino, K. Broda, and J. Lobo. 2020. "FastLAS: scalable inductive logic programming incorporating domain-specific optimisation criteria." In: *AAAI*.

M. Law, A. Russo, and K. Broda. 2014. "Inductive Learning of Answer Set Programs." In: *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings* (Lecture Notes in Computer Science). Ed. by E. Fermé and J. Leite. Vol. 8761. Springer, 311–325. doi:10.1007/978-3-319-11558-0_22.

M. Law, A. Russo, and K. Broda. 2018. "Inductive Learning of Answer Set Programs from Noisy Examples." *CoRR*, abs/1808.08441. http://arxiv.org/abs/1808.08441 arXiv: 1808.08441.

D. Lin, E. Dechter, K. Ellis, J. B. Tenenbaum, and S. H. Muggleton. 2014. "Bias reformulation for one-shot function induction." In: *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)* (Frontiers in Artificial Intelligence and Applications). Vol. 263. IOS Press, 525–530. doi:10.3233/978-1-61499-419-0-525.

J. G. M. van der Linden, D. Vos, M. M. de Weerdt, S. Verwer, and E. Demirovic. 2024. "Optimal or Greedy Decision Trees? Revisiting their Objectives, Tuning, and Performance." *CoRR*, abs/2409.12788. arXiv: 2409.12788. doi:10.48550/ARXIV.2409.12788.

J. W. Lloyd. 2012. *Foundations of logic programming*. Springer Science & Business Media.
A. T. McCray. 2003. "An upper-level ontology for the biomedical domain." *Comparative and Functional genomics*, 4, 1, 80–84.
L. Mihalkova, T. Huynh, and R. J. Mooney. 2007. "Mapping and revising markov logic networks for transfer learning." In: *Aaai*. Vol. 7, 608–614.
J. Mingers. 1989a. "An empirical comparison of pruning methods for decision tree induction." *Mach. Learn.*, 4, 227–243.
J. Mingers. 1989b. "An empirical comparison of selection measures for decision-tree induction." *Mach. Learn.*, 3, 319–342.
T. M. Mitchell. 1997. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill.
A. Moskvichev, V. V. Odouard, and M. Mitchell. 2023. "The ConceptARC Benchmark: Evaluating Understanding and Generalization in the ARC Domain." *CoRR*, abs/2305.07141. arXiv: 2305.07141. doi:10.48550/ARXIV.2305.07141.
S. Muggleton. 1991. "Inductive Logic Programming." *New Gener. Comput.*, 8, 4, 295–318. doi:10.1007/BF03037089.
S. H. Muggleton. 1995. "Inverse Entailment and Progol." *New Gener. Comput.*, 13, 3&4, 245–286. doi:10.1007/BF03037227.
S. H. Muggleton, D. Lin, and A. Tamaddoni-Nezhad. 2015. "Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited." *Mach. Learn.*, 100, 1, 49–73. doi:10.1007/s10994-014-5471-y.
S. H. Muggleton, L. D. Raedt, D. Poole, I. Bratko, P. A. Flach, K. Inoue, and A. Srinivasan. 2012. "ILP turns 20 - Biography and future challenges." *Mach. Learn.*, 86, 1, 3–23. doi:10.1007/s10994-011-5259-2.
P. M. Murphy and M. J. Pazzani. 1993. "Exploring the decision forest: An empirical investigation of Occam's razor in decision tree induction." *Journal of Artificial Intelligence Research*, 1, 257–275.
N. Narodytska, A. Ignatiev, F. Pereira, and J. Marques-Silva. 2018. "Learning optimal decision trees with SAT." In: *International Joint Conference on Artificial Intelligence 2018*. Association for the Advancement of Artificial Intelligence (AAAI), 1362–1368.
A. Odena, K. Shi, D. Bieber, R. Singh, C. Sutton, and H. Dai. 2021. "BUSTLE: Bottom-Up Program Synthesis Through Learning-Guided Exploration." In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=yHeg4PbFHh.
D. A. Pearce. 1988. "The induction of fault diagnosis systems from qualitative models." In: *Proceedings of the Seventh AAAI National Conference on Artificial Intelligence*, 353–357.
H. Peleg and N. Polikarpova. 2020. "Perfect is the Enemy of Good: Best-Effort Program Synthesis (Artifact)." *Dagstuhl Artifacts Ser.*, 6, 2, 16:1–16:2. doi:10.4230/DARTS.6.2.16.
O. Polozov and S. Gulwani. 2015. "FlashMeta: a framework for inductive program synthesis." In: *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2015, part of SPLASH 2015, Pittsburgh, PA, USA, October 25-30, 2015*, 107–126. doi:10.1145/2814270.2814310.
F. J. Provost, T. Fawcett, and R. Kohavi. 1998. "The Case against Accuracy Estimation for Comparing Induction Algorithms." In: *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998), Madison, Wisconsin, USA, July 24-27, 1998*. Ed. by J. W. Shavlik. Morgan Kaufmann, 445–453.
J. R. Quinlan. 1990. "Learning Logical Definitions from Relations." *Mach. Learn.*, 5, 239–266. doi:10.1007/BF00117105.
J. R. Quinlan and R. L. Rivest. 1989. "Inferring Decision Trees Using the Minimum Description Length Principle." *Inf. Comput.*, 80, 3, 227–248. doi:10.1016/0890-5401(89)90010-2.
L. D. Raedt. 2008. *Logical and relational learning*. Cognitive Technologies. Springer. ISBN: 978-3-540-20040-6. doi:10.1007/978-3-540-68856-3.
M. Richardson and P. Domingos. 2006. "Markov logic networks." *Mach. Learn.*, 62, 107–136.
J. Rissanen. 1978. "Modeling by shortest data description." *Autom.*, 14, 5, 465–471.
J. S. Rule, S. T. Piantadosi, A. Cropper, K. Ellis, M. Nye, and J. B. Tenenbaum. 2024. "Symbolic metaprogram search improves learning efficiency and explains rule learning in humans." *Nature Communications*, 15, 1, 6847. ISBN: 2041-1723. doi:10.1038/s41467-024-50966-x.
J. S. Rule. 2020. "The child as hacker: building more human-like models of learning." Ph.D. Dissertation. Massachusetts Institute of Technology.
C. Schaffer. 1993. "Overfitting avoidance as bias." *Mach. Learn.*, 10, 153–178.
T. Silver, K. R. Allen, A. K. Lew, L. P. Kaelbling, and J. Tenenbaum. 2020. "Few-Shot Bayesian Imitation Learning with Logical Program Policies." In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 10251–10258. doi:10.1609/AAAI.V34I06.6587.
R. Singh and S. Gulwani. 2015. "Predicting a correct program in programming by example." In: *Computer Aided Verification: 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I 27*. Springer, 398–414.
A. Srinivasan. 2001. *The ALEPH manual*. (2001). http://www.cs.ox.ac.uk/activities/machinelearning/Aleph/aleph.
A. Srinivasan, R. D. King, S. H. Muggleton, and M. J. Sternberg. 1997. "The predictive toxicology evaluation challenge." In: *IJCAI (1)*. Citeseer, 4–9.
A. Srinivasan, R. D. King, S. H. Muggleton, and M. J. E. Sternberg. 1997. "Carcinogenesis Predictions Using ILP." In: *Inductive Logic Programming, 7th International Workshop, ILP-97, Prague, Czech Republic, September 17-20, 1997, Proceedings* (Lecture Notes in Computer Science). Ed. by N. Lavrac and S. Dzeroski. Vol. 1297. Springer, 273–287. doi:10.1007/3540635149_56.

P. Tan, V. Kumar, and J. Srivastava. 2002. "Selecting the right interestingness measure for association patterns." In: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*. ACM, 32–41. doi:10.1145/775047.775053.

Y. Tian and Y. Zhang. 2022. "A comprehensive survey on regularization strategies in machine learning." *Information Fusion*, 80, 146–166. doi:https://doi.org/10.1016/j.inffus.2021.11.005.

Q. Wang, Y. Ma, K. Zhao, and Y. Tian. 2022. "A Comprehensive Survey of Loss Functions in Machine Learning." *Annals of Data Science*, 9, 2, 187–212. ISBN: 2198-5812. doi:10.1007/s40745-020-00253-5.

R. Wang, E. Zelikman, G. Poesia, Y. Pu, N. Haber, and N. Goodman. 2024. "Hypothesis Search: Inductive Reasoning with Language Models." In: *ICLR 2024*. https://openreview.net/forum?id=G7UtIGQmjm.

Y. Xu, W. Li, P. Vaezipoor, S. Sanner, and E. B. Khalil. 2023. "LLMs and the Abstraction and Reasoning Corpus: Successes, Failures, and the Importance of Object-based Representations." *CoRR*, abs/2305.18354. arXiv: 2305.18354. doi:10.48550/ARXIV.2305.18354.

J. Zahálka and F. Železný. 2011. "An experimental test of Occam's razor in classification." *Mach. Learn.*, 82, 3, 475–481. doi:10.1007/s10994-010-5227-2.

Q. Zeng, J. M. Patel, and D. Page. 2014. "QuickFOIL: Scalable Inductive Logic Programming." *Proc. VLDB Endow.*, 8, 3, 197–208.

C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. 2021. "Understanding deep learning (still) requires rethinking generalization." *Communications of the ACM*, 64, 3, 107–115.

Received 15 February 2026; accepted 18 March 2026