Summary

Classical planning is a problem within artificial intelligence with continuous improvement and is applicable in areas such as scheduling and robotics [1], [2]. There is an ongoing demand for new and refined planning algorithms. Classical planning problems are often described using the Planning Domain Definition Language (PDDL) [3] and the goal is to find a sequence of actions that transition from an initial state to a desired goal state. State-of-the-art planners such as FastForward, Fast Downward, and LAMA [4]–[6] can solve many classical planning problems with high efficiency. However, the search space for solutions grows exponentially when the complexity of a planning problem increases leading to increased computational complexity. This issue remains a challenge and motivates research to come up with an approach to deal with this problem.

One such approach addressing this challenge is partial grounding [7], which seeks to reduce the number of actions for a planner to consider by identifying and eliminating actions unlikely to be useful before the search for a solution begins. The GOFAI system [8], which won the learning track at the International Planning Competition (IPC) 2023 [9], utilised partial grounding with the inductive logic programming (ILP) tool Aleph [10] to learn rules that identify useful and useless actions. While Aleph has proven effective, there is still room for improvement in its application within classical planning.

This paper proposes Beyond Exhaustive Search (BES) which is a successor to the author's previous work Yet Another Hypothesis Generator (YAPHG) [11]. BES is an approach to learning classification models for PDDL domains that classify operators in a given PDDL task as good or bad. BES' model consists of decision tree classifiers which are build using the STreeD [12] framework which can create small and optimal binary decision trees.

One of the main features of BES is that it leverages partial order causal link (POCL) [13] graphs. These graphs are constructed from optimal plans, and provide contextual information for actions in these plans. Using these graphs, BES searches for rules, that can generalise the use of different actions in a planning domain. BES also integrates a relational database management system, to facilitate the evaluation of rules on actions in given planning problems.

The performance of models constructed by BES was compared to models constructed by state-of-the-art tool Aleph [10] and BES' predecessor YAPHG [11]. Empirical results indicate that models built using BES generally either outperform or perform comparably to those created by Aleph and YAPHG. Furthermore, our approach can construct high-performing models in planning domains where both Aleph and YAPHG fail to provide any model. While its effectiveness in a planning environment is yet to be tested, the promising predictive capabilities suggest potential applications in guiding partial grounding and other planning-related tasks.

Beyond Exhaustive Search: Rule Learning with Informed Search

Alexander Droob
Department of Computer Science
Aalborg University
Aalborg, Denmark
adroob19@student.aau.dk

Rune Bohnstedt
Department of Computer Science
Aalborg University
Aalborg, Denmark
rbohns18@student.aau.dk

Frederik Langkilde Jakobsen
Department of Computer Science
Aalborg University
Aalborg, Denmark
flja14@student.aau.dk

Abstract:
Classical planning is a core problem in artificial intelligence. Planners, which are programs that solve classical planning problems, often faces challenges due to the exponential growth of the search space for solutions as the number of actions and states increases. This paper introduces Beyond Exhaustive Search (BES), an approach designed to address this issue by learning classification models that can effectively classify actions in planning problems as either *good* or *bad*. BES utilises an adapted Inductive Logic Programming (ILP) approach and partial order causal link (POCL) graphs to derive additional knowledge from solved planning problems, in order to learn relevant rules. Utilising a relational database management system (RDBMS) for rule evaluation, BES constructs its classification models using the decision tree framework called STreeD, known for creating small and optimal binary decision trees. Empirical results show, that BES not only achieves comparable results to the state-of-the-art ILP tool Aleph and its predecessor YAPHG but also succeeds in scenarios where both Aleph and YAPHG fail to produce models. Although the practical integration of BES into planning systems remains to be fully explored, the promising predictive capabilities suggest potential applications in guiding partial grounding and other planning-related tasks.

I. Introduction

Classical planning is a fundamental problem in artificial intelligence with many applications, such as puzzle solving, robotics, and scheduling [1], [2]. Despite advancement in the field, new and improved planning algorithms are still being developed and showcased in international planning competitions emphasising the demand for advancement in this area of research [9], [14]–[16].

Given an environmental model, classical planning aims to find a series of actions that leads from an initial state to a desired goal state [1]. Often planning problems are described using the Planning Domain Definition Language (PDDL) [**?**]. Programs for solving classical planning problems such as FastForward as proposed by Hoffmann et. al [5], Fast Downward as introduced by Helmer [4], or LAMA by Richard et. al. [6] are called planners. Most state-of-the-art planners are very efficient at solving classical planning problems however, as the number of actions and states increases the search space of plans grows exponentially. This growth can lead to high computational complexity and can make it difficult and in some cases impossible for a planner to find a plan within a reasonable time frame. This challenge has led to much ongoing research in this area with multiple approaches attempting to provide a solution for this problem. One such method is introducing a heuristic to guide the search for a plan in a planner as proposed by Heusner et. al. [17]. Another approach is partial grounding as proposed by Gnad et. al. [7]. This method aims to reduce the number of actions for a planner to consider, by attempting to identify and remove actions that are unlikely to be useful in a given planning problem before beginning the search for a plan.

During the International Planning Competition (IPC) 2023 [9] the learning track was won by a system called GOFAI [8]. GOFAI utilised partial grounding by leveraging the inductive logic programming (ILP) tool Aleph to learn a set of rules that could help identify potentially useful and useless actions. Aleph is a state-of-the-art ILP tool providing stability, efficiency, and customisability [10]. However, while Aleph has proven effective in various domains there remained much room for improvement to its application within classical planning. In previous work, we proposed a hypothesis learning ILP tool: Yet Another Hypothesis Generator (YAPHG) [11] that given a planning domain and training data, can construct a hypothesis and identify useful actions for that domain. YAPHG was able to provide

hypotheses with results comparable to Aleph, but also had many limitations. One such limitation was that YAPHG generated its rule space in a nearly exhaustive manner. This meant that YAPHG would often consider many rules, with little to no relevance to the domain.

In this paper, we propose a successor to YAPHG called Beyond Exhaustive Search (BES). BES leverages an adaptation of Inductive Logic Programming (ILP) and a relational database management system (RDBMS) in order to learn a classification model, that can classify actions in a planning problem as $good$ or bad. BES utilises partial order causal link (POCL) graphs to extract additional knowledge from solved planning problems providing insights into when and why an action is used in a plan. This knowledge is used to guide a search in the rule space, in an attempt to learn more relevant rules. The model constructed by BES is a set of decision tree classifiers, which are learned using the decision tree framework STreeD [12] which is able to create small and optimal binary decision trees. Like YAPHG, BES specifically targets classical planning problems without relying on generic ILP tools that typically depend on logical programming environments like Prolog [10], [18].

To assess the performance of BES, we compare models constructed by BES to models constructed by YAPHG [11], and Aleph [10] as used in GOFAI [8]. Empirical results show that BES can generate models with results comparable to those produced by Aleph [10] and YAPHG. Furthermore in many cases where Aleph and YAPHG fail to produce a model within a given time limit, BES is able to construct consistently well-performing models.

II. Background

In this section, we present foundational definitions, notation, and terminology for classical planning, logic programming, and inductive logic programming (ILP). Many of these definitions are defined as in YAPHG [11].

A. *Classical Planning*

In classical planning, the objective is to transition a system from an initial state to a desired goal state by applying a sequence of actions. For every action the preconditions and effects of an action are known and the application of an action is deterministic [19].

Representing a classical planning task is facilitated through the use of the planning domain definition language (PDDL) which since its introduction in 1998 [14] has emerged as the standard language for expressing planning problems [20]. We use the blocks world domain as an ongoing example. In blocksworld, the goal is to arrange a fixed number of blocks, which are either placed on a table or on top of each other. In PDDL, planning tasks are represented with objects in the world such as blocks in blocksworld, predicates that define the relation between the objects (e.g., whether a block is on the table or stacked on top of another block), actions that modify the world state (e.g., moving a block), an initial state defining the initial world state, and a goal state that defines the desired world state [11], [20]. For simplicity in our ongoing example, we slightly modify the blocksworld domain. We remove the arm-empty predicate and assume that there is always an empty arm prepared to pick up a block in the world.

Definition 1 (Lifted PDDL task) Let Π_L be a lifted PDDL task defined as a tuple (P, A, Σ, I, G) such that P is a set of predicates with a arity ≥ 0, A constitutes a set of action schemas, Σ denotes the non-empty set of objects and I and G is the initial and goal state respectively. I and G consists of predicates from P instantiated with objects in Σ [21]. A set of parameters is denoted as X_i and a single parameter is denoted as x_i where i is an integer that allows for distinction. An action schema $a(X)$ consists of preconditions, add list and delete list, and is represented as a tuple $(pre(a), del(a), add(a))$ where X is a parameter mapping in a. Let $pre(a)$ represent a set of predicates to verify as preconditions, $del(a)$ denote the set of predicates to remove, and $add(a)$ indicate the predicates to include. For all predicates $p(X_1)$ belonging to the union of $pre(a)$, $del(a)$, and $add(a)$, X_1 is a subset of X [21].

An action schema example of the modified blocksworld domain is: $\text{pickup}(?x)$ where
$pre = \{\text{clear}(?x), \text{on-table}(?x)\}$
$del = \{\text{clear}(?x), \text{on-table}(?x)\}$ and
$add = \{\text{holding}(?x)\}$

A lifted PDDL task is a representation that combines both a PDDL domain and a PDDL problem.

Definition 2 (PDDL domain) A PDDL domain $\mathcal{D}$ is defined as a tuple (P, A) such that P is a set of predicates and A is a set of action schemas. Both P and A are consistent across all instances within the domain [22].

In our modified blocksworld domain, the PDDL domain includes four action schemas and three predicates. The action schemas are: $\text{pickup}(?ob)$, $\text{putdown}(?\text{ob})$, $\text{stack}(?ob, ?underob)$, and $\text{unstack}(?ob, ?underob)$ The predicates are: $\text{clear}(?x)$, $\text{on-table}(?x)$, and $\text{on}(?x, ?y)$. Here the question marks ? denotes a parameter.

Definition 3 (PDDL problem instance) a PDDL problem instance Π is a triple (Σ, I, G), where: Σ denotes the set of objects, I is the initial state and G is the goal state. A problem instance delineates the specific problem scenario and varies across different instances [21].

In the modified blocksworld domain a PDDL problem instance could consist of the following: The set of objects $\Sigma = \{b1, b2\}$, an initial state: $I = \{\text{clear}(b2), \text{on-table}(b2), \text{clear}(b1), \text{on-table}(b1)\}$, and a goal state:

$G = \{\text{clear}(b1), \text{on}(b1, b2), \text{on-table}(b2)\}$.

Definition 4 (Grounded PDDL task) A grounded PDDL task Π_G is defined as a tuple (F, O, I, G) where F represents a set of facts, O represents a set of operators, I $\subseteq F$ and $G \subseteq F$ represents the initial state and the goal state respectively [21].

The set of facts F is constructed from the predicates P of the lifted task representation Π_L and the parameters are substituted with objects from the set of objects Σ from Π_L [11], [21]. Similarly, the set of operators O is constructed from the set of action schemas A of the lifted PDDL task representation Π_L. An operator o can be applied in a state $s \subseteq F$ only if the preconditions of o are satisfied in s resulting in a new state $s' = (s \setminus del(o)) \cup add(o)$ [21].

The action schema for $pickup(?x)$ from the example given earlier can be grounded with the object b1 such that we get the operator:
pickup$(b1)$ =
$pre = \{\text{clear}(b1), \text{on-table}(b1)\}$,
$del = \{\text{clear}(b1), \text{on-table}(b1)\}$,
$add = \{\text{holding}(b1)\}$

Definition 5 (PDDL plan) A succession of operators constitutes a PDDL plan Φ. We say that $s \xrightarrow{o} s'$ denotes the transition from s to s' by operator o. A succession of operators is denoted as $\bar{o}$, if the operators in $\bar{o}$ can be applied to s continuously such that $s \xrightarrow{\bar{o}} s''$. A sequence $\bar{o}$ such that $I \xrightarrow{\bar{o}} s''$, where $G \subseteq s''$ is a plan for a task Π.

Applications that aim to solve PDDL problem instances are referred to as planners [7].

B. Logic Programming

Logic programming is a paradigm for programming and knowledge representation that originates from first-order logic. Our definitions are formulated according to the principles of logic programming as outlined by Lloyd in 1984 [23]. We can define first-order logic theories using a first-order language constructed from a first-order alphabet.

Definition 6 (Alphabet) An alphabet is defined as a tuple (V, C, F, P) where V is a set of variables, C is a set of constants, F is a set of function symbols and P is a set of predicate symbols.
Function symbols and predicate symbols all have an arity larger than 0, and for any given alphabet, only C and F may be empty. We can define a first order language given by an alphabet.

Definition 7 (Term) A term in a first order language is either a variable $v \in V$, a constant $c \in C$ or an n-ary function symbol $f \in F$ on the form $f(t_1, ..., t_n)$ where $t_1, ..., t_n$ are terms.

Definition 8 (Formula) A formula is a logical expression that determines a truth value. A formula can be inductively defined as:

- An n-ary predicate symbol $p \in P$ on the form $p(t_1, ..., t_n)$, where $t_1, ..., t_n$ are terms, is a formula. Formulas on this form are also called atoms.
- Any two formulas in-fixed with logic symbol $\wedge, \vee, \rightarrow$ or $\leftarrow$ are also formulas.
- If F is a formula then the negation $\neg F$ is also a formula.
- If F is a formula and x is a variable then then $(\forall x\ F)$ and $(\exists x\ F)$ are also formulas.

Some examples of formulas are: $\forall x\ p(x, y)$, $\neg(\forall x\ p(x, y))$ and $p(x, y) \rightarrow q(x)$ where $p, q \in P$ are predicate symbols and $x, y \in V$ are variables.

When a variable is prefixed by $\forall$ or $\exists$ we say that that variable is *bound* in the scope of the following formula. All unbound variables in a formula are *free*. For example in $\forall x\ F$ where $F = p(x, y)$, x is bound in the scope of F and y is free.

Definition 9 (Literal) A literal is an atom or the negation of an atom. A negated atom is called a negative literal and an atom is called a positive literal.

In first order logic we can define clauses that are propositional formulas:

Definition 10(Clause) A clause is a formula on the form:
$$\forall x_1 ... \forall x_n\ (l_1 \vee ... \vee l_m)$$
where $l_1, ..., l_m$ are literals and $x_1, ..., x_n$ are variables.

For convenience a clause can be written in causal form, also called implicative form:
$$A_1, ..., A_n \leftarrow B_1, ..., B_m$$
Where $A_1, ..., A_n$ and $B_1, ..., B_m$ are literals.
This notaition is equivalent to:
$$\forall x_1 ... x_k\ (A_1 \vee ... \vee A_n \leftarrow B_1 \wedge ... \wedge B_m)$$
which is equivalent to
$$\forall x_1 ... x_k\ (A_1 \vee ... \vee A_n \vee \neg B_1 \vee ... \vee \neg B_m)$$
Which is the standard form of a clause.

Finally we have the concept of horn clauses which are clauses in implicative form with only one positive literal:

Definition 11 (Horn clause) A horn clause is a clause on the form:
$$A \leftarrow B_1, ..., B_n$$
Here A is called the $head$ and contains exactly one positive literal and $B_1, ..., B_n$ is called the body and consists of only literals.

C. *Inductive Logic Programming*

Inductive Logic Programming (ILP) is a type of machine learning that employs symbolic logic, using first-order logic to represent both data and learning tasks. The goal in ILP is to create a set of horn clauses that can logically deduce all positive instances in a learning dataset while excluding any negative instances [24], [25]. This method aids in extracting general patterns from data, allowing for the discovery of complex relationships within intricate datasets.

The concept of an ILP input is defined as follows:

Definition 12 (ILP Input) An ILP input is denoted as a tuple $(E^+, E^-, B, \mathcal{H})$ Where E^+ represents a finite set of positive examples and E^- represents a finite set of negative examples, both sets are expressed as literals. B signifies some background knowledge in the form of logical clauses, and $\mathcal{H}$ denotes the hypothesis space which comprises the powerset of the set, encompassing all conceivable horn clauses.

Definition 13 (ILP solution) The solution of an ILP problem is a hypothesis $H \in \mathcal{H}$ Where H is a set of horn clauses. A hypothesis H is considered a correct solution, when it satisfies the requirements:

$$\text{Completeness: } B \cup H \models E^+$$
$$\text{Consistency: } B \cup H \not\models E^-$$

The symbol $\models$ denotes logical entailment. Completeness ensures that the conjunction of the clauses in the hypothesis entails all possible examples given the background knowledge, and consistency ensures that the hypothesis does not entail the negative examples given the background knowledge.

III. Learning Task

The learning task considered in this paper, is the task of learning a model that can classify operators in a classical planning problem as either $good$ or bad. This classification can then be used to reduce the amount of operators, a planner has to consider when searching for a solution, for example, by guiding a partial grounding [7].

This learning task closely relates to a standard ILP learning task. In a standard ILP approach, the system is provided with training data in the form of positive and negative examples as well as background knowledge. For our learning task the training data consists of what we call problem examples:

Definition 14 (Problem example) A problem example ε is a tuple (Π_G, O^+, O^-, Φ) where $\Pi_G = (F, O, I, G)$ is a grounded PDDL task, O^+ denotes the set of operators for Π_G labeled $good$, O^- denotes the set of operators for Π_G labeled bad and Φ is a valid PDDL plan for Π_G.

These problem examples are constructed from solved PDDL tasks. Here the PDDL plan Φ is an optimal plan provided by a planner, The sets of operators O^+ and O^- for $\Pi_G = (F, O, I, G)$ are labeled training instances, and are defined such that $O^+ \cup O^- = O$.

We can define the learning task targeted by this paper:

Definition 15 (Learning task) A learning task solved by the BES system is a tuple $\Gamma = (D, \mathcal{E})$ where $D = (P, A)$ is a PDDL domain, and $\mathcal{E}$ is a set of problem examples within the domain D.

The solution to such a learning task, would be a model that given a PDDL task Π_G and an operator o can classify o as either $good$ or bad.

IV. Beyond Exhaustive Search

The goal of BES is to provide a solution for the learning task defined in section III. The goal is to construct a model, that can classify operators in a classical planning problem as $good$ or bad.

To achieve a model that can perform this classification, BES takes advantage of an inductive logic programming (ILP) approach. To apply an ILP approach within classical planning, we have to introduce a few modifications to the standard definition of ILP. For convenience we define a form of horn clauses applicable in a classical planning domain as a rule:

Definition 16 (Rule) A rule r for a PDDL domain $D = (P, A)$ is a tuple $(head, body)$ where $head$ is an action schema $a \in A$ instantiated with a set of variables and the $body$ is a tuple $(\mathrm{P}_I, \mathrm{P}_G)$ where P_I is a set of atoms to be fulfilled in the initial state, based on predicates from P that are instantiated with variables and P_G is similarly a set of atoms to be fulfilled in the goal state, based on predicates from P that are instantiated with variables.

An example of a rule for the modified blocksworld domain could be: $unstack(x_1, x_2)$:-$init$:$on(x_1, x_2), goal : on(x_1, x_3)$ which reads: If $on(x_1, x_2)$ is in the initial state, and $on(x_1, x_3)$ is in the goal state for some assignment of x_1, x_2 and x_3 then $unstack(x_1, x_2)$ is useful in reaching the goal state.

Any variable that occurs more than once in a rule is considered a bound variable and all other variables are considered free. In the example above, x_1 and x_2 are bound and x_3 is free.

The variables of an atom in the body of a rule can be replaced with objects from a given PDDL task. The resulting atom is referred to as a grounded atom. Each grounded atom corresponds to a fact in the given PDDL task.

For a classical planning domain, the rule space is the set containing all possible rules for that domain. Since the body of a rule can contain any number of atoms, and thus any number of variables, the rule space is infinite.

To evaluate a rule on an operator within a PDDL task, the variables in the head and corresponding variables in the body, are replaced with values matching the parameters of the operator. If there then is an assignment of the remaining unassigned variables, such that the resulting grounded atoms in the body appear as facts in the initial and/or goal state of the problem, the rule evaluates to true. If no such assignment exists the rule evaluates to false. How this evaluation is done in BES is described in section VIII

We can consider the operator $unstack(b_1, b_2)$ for a PDDL task $\Pi_G = (F, O, I, G)$ such that $unstack(b_1, b_2) \in O$ and b_1 and b_2 are objects. To evaluate the rule from the earlier example on this operator, the variables x_1 and x_2 which appear in the head would be replaced by the values b_1 and b_2 along with any corresponding variables in the body. The resulting rule would be: $unstack(b_1, b_2)$:-$init$:$on(b_1, b_2), goal : on(b_1, x_3)$. If there then is an assignment of x_3 such that $on(b_1, b_2) \in I$ and $on(b_1, x_3) \in G$ then the rule would evaluate to true.

Given a learning task, the goal of BES is to construct a model, that can classify operators in a given PDDL task as $good$ or bad. Rather than using a disjunction of rules like most standard ILP algorithms, the model constructed by BES is a set of decision tree classifiers. This ensures, that the model can depict both disjunctions and conjunctions as well as the negation of rules. Each decision tree can classify operators based on a single action. We define a BES decision tree as:

Definition 17 (BES Decision tree) A BES decision tree τ is binary a decision tree where each node is a rule, and each leaf is a classification $good$ or bad.

Given a grounded PDDL task $\Pi_G = (F, O, I, G)$, a BES decision tree τ can be used to classify an an operator $o \in O$. This is done by evaluating the rule in each node of τ on o. If a rule evaluates to true o is evaluated on the rule in the left child, and if the rule is evaluated to false, o is evaluated on the rule in the right child. Once a leaf node is reached, τ returns a classification for o corresponding to the value in the leaf.

As an example of this, we can consider the following grounded PDDL task in our modified blocksworld domain: $\Pi_G = (F, O, I, G)$ where
$I = \{on(b1, b2), \text{on-table(b2)}, clear(b1)\}$ and
$G = \{\text{on-table(b1)}, \text{on-table(b2)}, clear(b1), clear(b2)\}$. A simple example of what a small BES decision tree τ could look like is shown on Figure 1.

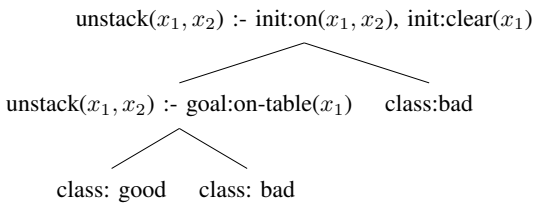

Fig. 1: Example of what a simple BES decision tree could look like

If τ were given the operator $unstack(b1, b2) \in O$ then the rule in the root node would evaluate to true, since $on(b1, b2) \in I$ and $clear(b1) \in I$. o would then be evaluated on the left child of the root which would again evaluate to true since $\text{on-table}(b1) \in G$, and the classification given by τ would be $good$. The operator $unstack(b2, b1)$ on the other hand, would be classified as bad by τ, since the root node would evaluate to false.

Given this definition of a BES decision tree, we can now define the final model constructed by BES:

Definition 18 (BES Model) A BES Model for a PDDL domain $D = (P, A)$ is a mapping $\mathcal{M} : A \to \mathcal{T}$ where A is the set of actions in D and $\mathcal{T}$ is the space of possible BES models in D.

BES learns a BES decision tree for each action in a domain, and the resulting model M then maps each action to the corresponding model. When classifying an operator o on M the action a that o is based on is used to identify the correct decision tree τ. We denote this as $M[a] = \tau$. Once τ is found, it is used to classify o. For simplicity, we denote the classification of an operator o as $\mathcal{M}(o)$.
When searching for relevant rules, BES utilises partial order causal link (POCL) graphs to extract additional knowledge from PDDL plans. POCL grahs and how these are exploited by BES is explained in detail in section V and section VI.

The BES system is described on algorithm 1 and can be separated into 3 main parts: Partial order causal link (POCL) graph construction, POCL rule generation, and model construction. As seen in lines 3-5 on algorithm 1 BES first converts the PDDL plan for each problem example into a POCL graph. The process of converting the provided PDDL plans to POCL graphs is described in depth in section V.

These constructed POCL graphs are then used to create a set of potentially useful rules for each action in the domain, as shown in line 7. The rule generation is described in section VI.

Finally, as shown in line 8 BES constructs the decision tree classifier for the action, using the created input. The creation of the input and the construction of the final model, is described in more detail in section VII.

Algorithm 1: BES algorithm

Input : A BES learning task $\Gamma = (D, \mathcal{E})$, where $D = (P, A)$
Output: A BES model $\mathcal{M}$

$graphs \leftarrow \varnothing$; //Set of POCL graphs
$\mathcal{M} \leftarrow A \rightarrow \mathcal{T}$ //Maps an action to a BES decision tree. Initially all actions maps to nothing
for $\varepsilon \in \mathcal{E}$ **do**
  $graphs \leftarrow graphs \cup generate\_pocl\_graph(\varepsilon)$;
end
for $a \in A$ **do**
  $rules \leftarrow generate\_rules(a, graphs)$;
  $M[a] \leftarrow construct\_optimal\_tree(a, rules, \mathcal{E})$;
end
return $\mathcal{M}$;

V. POCL Graph Generation

As mentioned in section IV BES utilises partial order causal link (POCL) graphs to extract additional knowledge from a given PDDL plan.

POCL planning is a planning method that follows the principle of least commitment [26] by only maintaining a partial order in the operators in a solution. We define POCL planning based on the definitions by Bercher [13]. A POCL solution to a PDDL task is represented as a POCL Graph.

Definition 19 (POCL graph) A POCL graph for a grounded PDDL task Π_G is a tuple $P_{POCL} = (PS, CL)$. PS is a finite set of POCL plan steps. Each plan step $ps \in PS$ is a tuple $ps = (i, o)$ where o is an operator and i is a unique identifier for the plan step which enables distinction between plan steps with identical operators. We denote $(pre(ps),\ del(ps),\ add(ps))$ such that $pre(ps) = pre(o), del(ps) = del(o)$ and $add(ps) = add(o)$. $CL \in PS \times F \times PS$ is a set of causal links where $F \in \Pi_G$ is the set of facts for the PDDL Task. Each causal link $cl \in CL$ is a tuple $cl = (ps, f, ps')$ where ps and ps' are plan steps in PS and f is a fact connecting them such that $f \in add(ps)$ and $f \in pre(ps')$. ps is called the producer of f and ps' is called the consumer

To incorporate the initial and goal state in a PDDL task $\Pi_G = (F, O, I, G)$, every POCL graph for a PDDL task introduces plan steps $init$ and $goal$ such that $add(init) = I$ and $pre(goal) = G$. The $init$ plan step precedes all other steps in the plan, and the $goal$ plan step succeeds all other steps. $pre(init) = del(init) = add(goal) = del(goal) = \emptyset$.

The notion of causal links have in many instances been used for explainability [27]. In POCL graphs, the causal links provide a context for each plan step chosen for a solution. BES aims to utilise these causal links, in an attempt to discover the connection between facts in the initial and goal state of a given PDDL task, and the choice of operators in the solution.

For every problem example $\varepsilon = (\Pi_G, O^+, O^-, \Phi)$ in the provided learning task BES generates a corresponding POCL graph based on Φ. The conversion of a PDDL plan to a POCL graph is described by algorithm 2.

Algorithm 2: POCL algorithm

Input : A problem example $\varepsilon = (\Pi_G, O^+, O^-, \Phi)$ where $\Pi_G = (F, O, I, G)$
Output: A POCL graph $P_{POCL} = (PS, CL)$

1 $init \leftarrow create\_plan\_step\_init(I)$; // Plan step for initial state I
2 $goal \leftarrow create\_plan\_step\_goal(G)$; // Plan step for goal state G
3 $PS \leftarrow \{init, goal\}$; // Set of plan steps for P_{POCL}
4 $CL \leftarrow \varnothing$; // Set of causal links for P_{POCL}
5 $current\_facts \leftarrow \varnothing$; // Set of currently present facts in the POCL graph P_{POCL} with corresponding producing plan step
6 **for** $fact \in I$ **do**
7 $current\_facts \leftarrow current\_facts \cup \{(init, fact)\}$;
8 **end**
9 **for** $o \in \Phi$ **do**
10 $ps \leftarrow (unique\_id(), o)$;
11 $PS \leftarrow PS \cup ps$;
12 **for** $(ps', fact) \in current\_facts$ **do**
13 **if** $fact \in pre(ps)$ **then**
14 $CL \leftarrow CL \cup \{(ps', fact, ps)\}$;
15 **end**
16 **if** $fact \in del(ps)$ **then**
17 $current\_facts \leftarrow current\_facts \setminus \{(ps', fact)\}$;
18 **end**
19 **end**
20 **for** $fact \in add(ps)$ **do**
21 $current\_facts \leftarrow current\_facts \cup (ps, fact)$;
22 **end**
23 **end**
24 **for** $(ps, fact) \in current\_facts$ **do**
25 **if** $fact \in pre(goal)$ **then**
26 $CL \leftarrow CL \cup \{(ps, fact, goal)\}$;
27 **end**
28 **end**
29 **return** (PS, CL);

When constructing a POCL graph for a plan in a problem example, BES first creates plan steps for the initial and goal state of the problem example, and adds these to the POCL graph. BES then adds all facts present in the initial state to the set of currently present facts as seen in lines 6-8. This set, keeps a reference to each fact currently present in the graph, paired with the producing plan step.

Next BES iterates over each operator in the provided valid plan for the problem example. For each operator o BES assigns a unique identifier and creates a plan step ps which is added to the graph. For every fact $f \in pre(ps)$, a tuple

(ps', f) is found in the set of current facts[1] where ps' is the producer of f. A causal link (ps', f, ps) is added to the graph and if $f \in del(ps)$ then f is removed from the set of current facts.

Additionally, for every fact $f' \in add(ps)$, (ps, f') is added to the set of current facts. This iteration over each operator in the plan is described on lines 9-23.

Once all operators in the provided plan have been processed each fact in the goal state is found in the set of current facts and a causal link between their producing plan step and the goal plan step is added using the fact as seen on lines 24-28. Finally, the resulting POCL graph is returned on line 29.

Since the plans provided in the given problem examples are required to be valid, the POCL graphs constructed by BES are also guaranteed to be valid. A valid plan is guaranteed to be applicable in sequence, given the PDDL task, meaning that each fact must be present when applying an action ensuring that each causal link can be created. Additionally, a valid plan guarantees that the goal state is fulfilled after applying each action in the plan, meaning that a causal link can be created for every fact in the goal state.

As an example we can consider the problem example $\varepsilon = (\Pi_G, O^+, O^-, \Phi)$ for the blocksworld domain where $\Pi_G = (F, O, I, G)$ is a grounded PDDL task, $I = \{\text{on\_table(b1)}, \text{on(b2, b1)}, \text{clear(b2)}, \text{on\_table(b3)}, \text{on(b4, b3)}, \text{clear(b4)}\}$ is the initial state for the PDDL task $G = \{\text{on\_table(b2)}, \text{clear(b1)}, \text{clear(b3)}, \text{on(b3, b4)}\}$ is the goal state for the PDDL task and $\Phi = \text{unstack(b2, b1)} \rightarrow \text{putdown(b2)} \rightarrow \text{unstack(b4, b3)} \rightarrow \text{putdown(b4)} \rightarrow \text{pickup(b3)} \rightarrow \text{stack(b3, b4)}$ is the sequence of operators making up an optimal PDDL plan for the PDDL task. Following the graph construction described above, the resulting graph is depicted on Figure 2.

VI. POCL Rule Generation

To generate rules, BES uses the POCL graphs described in section V. BES aims to utilise the causal links in the POCL graphs, to discover which facts in the initial and goal state might be relevant for the inclusion of a given action in the plan.

For convenience we define the notion: $f \rightsquigarrow ps$ where f is a fact in some causal link and ps is a plan step. This denotes that f is connected to ps by some sequence of causal links. Similarly, $ps \rightsquigarrow f$ defines that ps is connected to f through some sequence of causal links.

Given a POCL graph $P_{POCL} = (PS, CL)$ and an action a, BES will then for each plan step $ps \in PS$, that is constructed from an operator based on a, generate rules by following three steps:

The first step is to find all facts from the initial and goal state, that are connected to the plan step in the graph. BES first finds the set of connected facts from the initial state: $F_{init} = \{f | f \rightsquigarrow ps, (init, f, ps') \in CL\}$ where $init \in PS$ is the plan step for the initial state. Similarly, a set of facts for the goal state is created: $F_{goal} = \{f | ps \rightsquigarrow f, (ps', f, goal) \in CL\}$ where $goal$ is the plan step in P_{POCL} for the goal state.

In the example on Figure 2 the sets F_{init} and F_{goal} achieved from using the plan step $putdown(b2)$ would be:
$F_{init} = on(b2, b1), clear(b2)$
$F_{goal} = on\_table(b2)$

The second step is to generate a set of rules based on the operator in ps and the facts from F_{init} and F_{goal}: $\mathcal{R} = \{(o, (P_I, P_G)) | P_I \in \mathcal{P}(F_{init}), P_G \in \mathcal{P}(F_{goal}, P_I \cup P_G \neq \varnothing, (i, o) = ps)\}$ where $\mathcal{P}(S)$ denotes the powerset of S. For every possible combination of facts in the initial or goal state, that might be relevant for the plan step ps, $\mathcal{R}$ will contain a rule with a body containing the combination.

The third step is to construct the final set of rules $\mathcal{R}'$ by replacing the parameters in the head and body of each rule with variables $x_1, x_2, ..., x_n$, where n is the total number of unique objects in the parameters of the rule.

Continuing from the example, the resulting set of rules $\mathcal{R}$ would be:
$\{putdown(b2)$:-$init : on(b2, b1)$,
$putdown(b2)$:-$init : clear(b2)$,
$putdown(b2)$:-$goal : on\_table(b2)$,
$putdown(b2)$:-$init : on(b2, b1), init : clear(b2)$,
$putdown(b2)$:-$init : on(b2, b1), goal : on\_table(b2)$,
$putdown(b2)$:-$init : clear(b2), goal : on\_table(b2)$,
$putdown(b2)$:-$init : on(b2, b1), init : clear(b2)$,
$goal : on\_table(b2)\}$

After replacing objects with variables the resulting set of rules $\mathcal{R}'$ would be:
$\{putdown(x_1)$:-$init : on(x_1, x_2)$,
$putdown(x_1)$:-$init : clear(x_1)$,
$putdown(x_1)$:-$goal : on\_table(x_1)$,
$putdown(x_1)$:-$init : on(x_1, x_2), init : clear(x_1)$,
$putdown(x_1)$:-$init : on(x_1, x_2), goal : on\_table(x_1)$,
$putdown(x_1)$:-$init : clear(x_1), goal : on\_table(x_1)$,
$putdown(x_1)$:-$init : on(x_1, x_2), init : clear(x_1)$,
$goal : on\_table(x_1)\}$

These three steps are repeated across all plan steps of the given action, in all graphs created by BES. By default, the resulting rules returned by BES' rule generation is the union of all rule sets $\mathcal{R}'$ generated across all plan steps for the given

[1] (ps', f) is guaranteed to be present in the set of current facts, since we know that the provided plan is valid and thus the preconditions for every action must either be present in the initial state or be produced by a preceding action

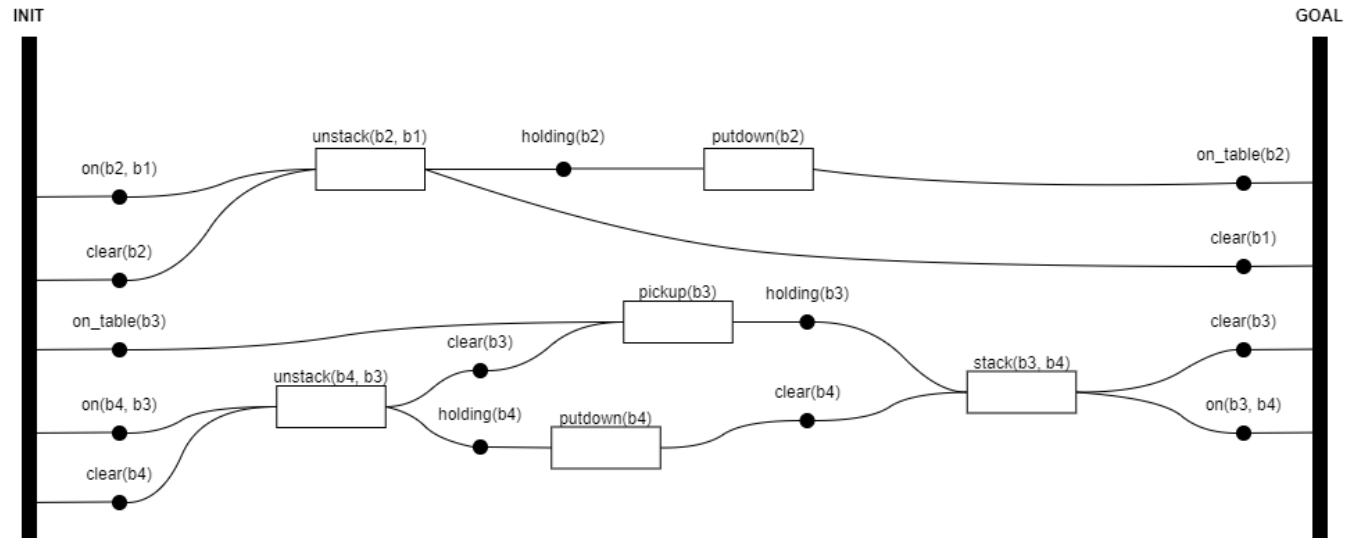

Fig. 2: The POCL graph illustrates the result of converting the plan $\text{unstack}(\text{b2}, \text{b1}) \rightarrow \text{putdown}(\text{b2}) \rightarrow \text{unstack}(\text{b4}, \text{b3}) \rightarrow \text{putdown}(\text{b4}) \rightarrow \text{pickup}(\text{b3}) \rightarrow \text{stack}(\text{b3}, \text{b4})$ from the problem example given in section V to a POCL graph.

action in all constructed POCL graphs. BES also introduces an optional hyper-parameter $\omega \in \mathbb{N}$. When ω is set, the rule generation will only return the ω most frequently occurring rules across all rule sets $\mathcal{R}'$.

VII. Model Construction

As mentioned in section IV the model constructed by BES is a set of decision trees. To construct these trees, BES utilises a decision tree framework named STreeD. STreeD is able to optimise for non-linear metrics such as F_1-score and generates optimal decision trees, providing high predictive capability in small trees [12].

Given a set of binary features $\mathcal{F}$ and a set of labels $\mathcal{K}$ used to describe training instances, the input data D for the STreeD framework consists of training instances (x, k) where $x \in \{0,1\}^{|\mathcal{F}|}$ is a feature vector, which denotes the satisfaction of each feature in the instance. $k \in \mathcal{K}$ is a label describing the value of the instance. Using D, $\mathcal{F}$, $\mathcal{K}$ and integers MAX_d and MAX_f, STreeD constructs an optimal decision tree $\tau_{STreeD} = (\mathcal{B}, \mathcal{L}, b, l)$ with a max depth MAX_d and a maximum number of feature nodes MAX_f where $\mathcal{B}$ is the set of branching internal nodes, $\mathcal{L}$ is the set of leaf nodes $b : \mathcal{B} \rightarrow \mathcal{F}$ is an assignment of features to the branching nodes, $l : \mathcal{L} \rightarrow \mathcal{K}$ is an assignment of leaf nodes to labels.

To use STreeD, BES uses the set of constructed rules as described in section VI as the feature set $\mathcal{F}$. The set of labels $\mathcal{K} = \{1, 0\}$ where 1 represents that an operator is *good*, and 0 represents that an operator is *bad*. Each training instance $(x, k) \in D$ describes an operator in a problem example $\varepsilon = (\Pi_G, O^+, O^-, \Phi)$, where $x \in \{1,0\}^{|F|}$ describes the evaluation of each rule $r \in F$ on o and $k \in \mathcal{K}$ is a label describing if an operator is good (1) or bad (0).

Given an action a, a set of rules $\mathcal{F}$ for a and a set of problem examples $\mathcal{E}$, BES can construct an input for STreeD. For each problem example $\varepsilon \in \mathcal{E}$, where $\varepsilon = (\Pi_G, O^+, O^-, \Phi)$ BES creates sets O_a^+ and O_a^- such that O_a^+ contains all operators $o \in O^+$ that are based on a and O_a^- contains all operators $o \in O^-$ that are based on a. BES then constructs $D_\varepsilon^+ = \{(x, 1) | x = \{eval(r, o, \varepsilon) | r \in \mathcal{F}\}, o \in O_a^+\}$ and $D_\varepsilon^- = \{(x, 0) | x = \{eval(r, o, \varepsilon) | r \in \mathcal{F}\}, o \in O_a^-\}$
Where $eval(r, o, \varepsilon)$ is the evaluation of a rule r on an operator o in the problem example ε. Using these BES constructs the set $D_\varepsilon = D_\varepsilon^+ \cup D_\varepsilon^-$
Once this has been done for every problem example $\varepsilon \in \mathcal{E}$, BES constructs the input for STreeD:

$$D = \bigcup_{\varepsilon \in \mathcal{E}} D_\varepsilon$$

Using this input, BES uses STreeD to construct a BES decision tree τ_a for the action a.
This process is then performed for all actions $a \in A$ for the domain $\mathcal{D} = (P, A)$ in the learning task to construct a BES model $\mathcal{M}$ such that $\mathcal{M}[a] = \tau_a$.

The STreeD framework supports optimisation for both accuracy and F_1-score. Additionally, modifications have been made for BES such that STreeD can optimise for a custom F_β score.

VIII. Rule Evaluation

The evaluation of rules in BES leverages a relational database management system (RDBMS). To facilitate this BES represents its learning task in a relational database. This section describes this representation and the rule evaluation using relational algebra as described by Codd [28] as well as extended relational algebra [29]. The symbol $\bowtie$ denotes a natural join operation, $\bowtie_\phi$ denotes a join operation with join

conditions ϕ and σ denotes a selection.

To represent a learning task $\Gamma = (\mathcal{D}, \mathcal{E})$ where $\mathcal{D} = (P, A)$. For each predicate $p \in P$ BES constructs relations $T_I(p)$ and $T_G(p)$. Both $T_I(p)$ and $T_G(p)$ have attributes $\alpha_1, \alpha_2, .., \alpha_n$ where n is the arity of p and α_i represents the i'th parameter of p. Additionally, both relations have the attribute $\#$ which is an identifier used to distinguish between problem examples. Using these relations the initial and goal states in the PDDL tasks of each problem example can be represented. For each problem example $\varepsilon \in \mathcal{E}$ where $\varepsilon = (\Pi_G, O^+, O^-, \Phi)$ and $\Pi_G = (F, O, I, G)$ each fact $f \in I$ can be represented by a tuple in $T_I(p)$ where p is the predicate that f is based on. The tuple representing f is on the form $(f(1), f(2), ..., f(n), id(\varepsilon))$ where $f(i)$ denotes the i'th parameter of f and $id(\varepsilon)$ denotes a unique identifier for ε. Similarly, each fact $f \in G$ can be represented by a tuple in $T_G(p)$ where p is the predicate that f is based on. The tuple representing f is on the form $(f(1), f(2), ..., f(n), id(\varepsilon))$.

As an example we can consider a learning task for our modified blocksworld domain containing two problem examples: $\varepsilon_0 = (\Pi_{G0}, O_0^+, O_0^-, \Phi_0)$ and $\varepsilon_1 = (\Pi_{G1}, O_1^+, O_1^-, \Phi_1)$ where $\Pi_{G0} = (F_0, O_0, I_0, G_0)$, $\Pi_{G1} = (F_1, O_1, I_1, G_1)$ $id(\varepsilon_0) = 0$ and $id(\varepsilon_1) = 1$. If we had: $I_0 = \{on(b1, b2), on\_table(b2)\}$ and $I_1 = \{on(b3, b5), on(b1, b2), on\_table(b2), on\_table(b5)\}$ then the relations shown on Figure 3 are created.

$T_I(on)$

α_1	α_2	$\#$
$b1$	$b2$	0
$b1$	$b2$	1
$b3$	$b5$	1

$T_I(on\_table)$

α_1	$\#$
$b2$	0
$b2$	1
$b5$	1

Fig. 3: Relations $T_I(on)$ and $T_I(on\_table)$ for the given example

For the goal states we could have
$G_0 = \{on(b2, b1), on\_table(b2)\}$ and
$G_1 = \{on(b5, b4), on(b2, b1), on\_table(b4), on\_table(b1)\}$.
This would result in the relations shown on Figure 4.

$T_G(on)$

α_1	α_2	$\#$
$b2$	$b1$	0
$b5$	$b4$	1
$b2$	$b1$	1

$T_G(on\_table)$

α_1	$\#$
$b2$	0
$b4$	1
$b1$	1

Fig. 4: Relations $T_G(on)$ and $T_G(on\_table)$ for the given example

Using this representation, BES can evaluate a rule $r = (head, body)$ where $body = (P_I, P_G)$ on an operator o in a given problem example ε.

For convenience we define a mapping pos such that for an atom p in the body of a rule, $pos(p, x_i)$ gives the placement of variable x_i in p, for example in the atom $p = on(x_1, x_2)$, $pos(p, x_1) = 1$ since x_1 is in the first position. The mapping pos is also applicable to the action in the head of a rule.

To evaluate r on o in ε (denoted $eval(r, o, \varepsilon)$)
BES constructs and executes the query:
$T(r, o, \varepsilon) = \sigma_{\varphi_1}(T_{s_1}(p_1)) \bowtie_{C_1}, ..., \bowtie_{C_n - 1} \sigma_{\varphi_n}(T_{s_n}(p_n))$

Where $p_1, p_2, ..., p_n$ are the atoms in the body of r, $s_i = G$ if $p_i \in P_G$ and $s_i = I$ if $p_i \in P_I$. C_i is a set of join conditions between p_{i+1} and every atom $p_1, p_2, ..., p_i$. For every variable x present in p_{i+1} and for each atom $p \in \{p_1, p_2, ..., p_i\}$ if x is present in p then a condition is constructed:
$c_{p,p_{i+1}} = (T(p).\alpha_{pos(p,x)} = T(p_{i+1}).\alpha_{pos(p_{i+1},x)})$
C_i is then the union of all such constructed conditions.

φ_i is a selection criteria for each relation $T_{s_i}(p_i)$ such that for each variable x that is present in the head of r if x is present in p_i then φ_i contains the condition $\alpha_{pos(p_i,x)} = o(pos(head, x))$, where $o(j)$ denotes the j'th parameter of o. Additionally φ contains a condition: $\# = id(\varepsilon)$ which ensures that only tuples from the relevant problem example is considered.

If $|T(r, o, \epsilon)| > 0$ then the rule r evaluates to true.

IX. Experiments

To evaluate how well BES performs, we investigate BES' ability to learn a model given a learning task for a given domain, and the model's ability to correctly classify operators in PDDL tasks within that domain. To achieve this we benchmark the results of BES and compare them to the state-of-the-art ILP tool Aleph [10] as used by the winners of the international planning competition (IPC) learning track 2023 GOFAI [8]. Additionally, we benchmark against the YAPHG tool [11], which is the predecessor to BES.

The implementation of BES used to generate the experimental results for this paper, was implemented in Python. BES relies on SQLite for rule evaluation [30] as described in section VIII and STreeD [12] for model construction as described in section VII. The source code for the BES system can be found here: https://github.com/p10-AI-planning/BeyondExhaustiveSearch

A. Experimental Setup

All experiments reported in this paper were conducted on a machine with an Intel i5-13400 4.6 GHZ Processor with 32 GB DDR4 RAM running at 3000 MHz.

Each experiment presented in this section, consists of training a model on a set of training data, and then evaluating the performance of the model on a set of testing data.

Experiments were conducted on data from the IPC learning track 2023. For each domain used, around 40 problem examples were used as training data, and around 15 problem examples were used for testing data. Additionally, experiments were conducted on the autoscale-dataset [31]. For experiments conducted on this dataset, the training data consists of 140 problem examples for the training data and 60 problem examples for the testing data.

To assess the performance of each tool, domains of varying complexity were used. BES, YAPGH, and Aleph were evaluated on the blocksworld, rover and satellite domains from the IPC learning track 2024 dataset. Additionally, BES and Aleph were evaluated on the blocksworld, floortile, gripper, rover, satellite and zenotravel domains from the autoscale-dataset. A small description of each domain can be seen below [31]:

- **Blocksworld:** A robot arm must pick up and place blocks on a table. The blocksworld domain consists of 5 predicates with 0 to 2 parameters, and 4 actions with 1 to 2 parameters.
- **Floortile:** A grid-like floor with different types of tiles a robot must colour the tiles with two colours. The floortile domain consists of 10 predicates with 1 to 2 parameters and 6 actions with 3 to 4 parameters.
- **Gripper:** In gripper, a robot with two grippers must transport balls between rooms. The gripper domain consists of 7 predicates with 1 to 2 parameters and 3 actions with 2 to 3 parameters.
- **Rover:** The goal of rover is to perform several gathering tasks collecting soil and rock samples, taking pictures of objectives, and communicating the result to a lander. The rover domain consists of 23 predicates with 1 to 3 parameters and 9 actions with 2 to 6 parameters.
- **Satellite:** Multiple satellites equipped with different instruments must observe and acquire images based on their instruments. The satellite domain consists of 8 predicates with 1 to 2 parameters, and 5 actions with 2 to 4 parameters.
- **Zenotravel:** In zenotravel, people are transported around in planes. The zenotravel domain consists of 4 predicates with 2 parameters each, and 5 actions, with 3 to 6 parameters.

The Aleph configuration used in our experiments is the configuration employed by GOFAI which follows the default settings outlined in the Aleph manual [10][2]. Aleph was set with a 30-minute limit to search the rule space for each action schema.

In our experiments BES is run with STreeD optimised for F_1 score as well as optimised for a F_β score with $\beta = 2$ meaning that recall is 2 times as important as precision. This will encourage BES to prioritise classifying $good$ operators as $good$. The BES system is configured to have a tree depth of 3 and a max of 7 nodes, since this configuration provided the best results in testing. The rule generation step of BES was given a 60-second time limit.

Every experiment using the YAPHG system employs the default configuration as described in the YAPHG paper [11] with a rule generation time of 30 minutes.

B. Performance metrics

To access the performance of each system and configuration, we evaluate them based on the following metrics as defined by Manning et. al. [32].

- $Precision = \frac{TP}{TP+FP}$ indicates how many operators that are assessed to be positive are actually positive.
- $Recall = \frac{TP}{TP+FN}$ specifies how many actual positive values have been predicted positive.
- F_1-Score $= \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$ is the harmonic mean of recall and precision.

C. Results

For the blocksworld, rover and satellite domain, experiments were run using the IPC learning track 2023 dataset to train and evaluate BES, Aleph and YAPHG. Here BES was only run with optimisation for F_1-score. The results of these experiments are reported in terms of precision, recall, and F_1-score for all models as well as time for BES and YAPHG. For Aleph the runtime is unknown, but the time limit guarantees that it never exceeds 30 minutes. The Training and Testing columns report the number of good and bad operators as Pos/Neg respectively for both the training and testing dataset shown on Table I.

For the blocksworld domain we see that Aleph produced the best results when comparing precision with a score of one across all actions. This suggests that the Aleph configuration used in GOFAI favours precision. Aleph did however fall behind both BES and YAPGH when comparing against recall and F_1-score. Overall the performance for recall and F_1-score favored BES but YAPGH marginally outperforms BES in regards to F_1-score for stack and Recall and F_1-Score for unstack.

When looking at more complex domains such as rover and satellite, Aleph was completely unable to construct a model for the satellite domain within the 30 minute time limit, and the model constructed for the rover domain was unable to classify operators for most actions. Similarly, the models created by YAPHG are unable to classify operators for most actions in both domains. BES is able to construct well-performing models for both domains, and was able to classify operators for every actions except switch_off for which there was no training or testing data.

[2]Note, that the results of Aleph presented in this section, are based on the configuration used by GOFAI [8]. Other configurations of Aleph, might produce different results.

TABLE I: IPC Data

Action	Training Pos / Neg	Testing Pos / Neg	BES F1-Score Time (s)	BES F1-Score Precision	BES F1-Score Recall	BES F1-Score F1-Score	Aleph Precision	Aleph Recall	Aleph F1-Score	YAPHG Time (s)	YAPHG Precision	YAPHG Recall	YAPHG F1-Score
BLOCKSWORLD													
stack	510 / 2969	263 / 1072	144.023	0.812	**0.411**	0.545	**1**	0.392	0.563	1800	0.990	0.395	**0.565**
unstack	512 / 2967	262 / 1073	81.383	0.765	0.385	0.513	**1**	0.389	0.560	1800	0.990	**0.393**	**0.563**
pickup	195 / 170	88 / 37	67.649	0.863	**0.932**	**0.896**	**1**	0.727	0.842	1800	0.910	0.693	0.787
putdown	197 / 168	87 / 38	72.398	0.85	**0.977**	**0.909**	**1**	0.724	0.840	1800	0.905	0.655	0.760
ROVER													
sample_rock	167 / 229	34 / 63	70.208	0.64	**0.941**	**0.762**	**1**	0.235	0.381	1800	**1**	0.206	0.341
sample_soil	148 / 249	29 / 76	69.75	**0.622**	**0.966**	**0.757**	N/A	N/A	N/A	1800	N/A	N/A	N/A
calibrate	330 / 335	76 / 88	112.668	0.541	**0.868**	**0.667**	**0.923**	0.316	0.471	1800	0.774	0.316	0.449
communicate_image_data	889 / 5139	219 / 1118	72.038	**0.426**	**0.982**	**0.594**	N/A	N/A	N/A	1800	N/A	N/A	N/A
take_image	1321 / 7629	288 / 1955	92.476	**0.479**	**0.972**	**0.642**	N/A	N/A	N/A	1800	N/A	N/A	N/A
communicate_rock_data	308 / 1113	68 / 271	62.303	0.496	**0.838**	**0.623**	**1**	0.074	0.137	1800	N/A	N/A	N/A
communicate_soil_data	282 / 1230	52 / 356	62.602	0.35	**0.692**	**0.465**	**0.857**	0.115	0.203	1800	N/A	N/A	N/A
drop	84 / 64	16 / 23	73.649	**0.611**	**0.688**	**0.647**	N/A	N/A	N/A	1800	N/A	N/A	N/A
navigate	559 / 2301	121 / 591	70.833	**0.333**	**0.636**	**0.437**	N/A	N/A	N/A	1800	N/A	N/A	N/A
SATELLITE													
switch_on	88 / 165	52 / 138	1.641	**0.274**	**1**	**0.43**	N/A	N/A	N/A	1800	N/A	N/A	N/A
switch_off	0 / 0	0 / 0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1800	N/A	N/A	N/A
take_image	278 / 1205	237 / 1911	1.591	**0.655**	**0.641**	**0.648**	N/A	N/A	N/A	1800	N/A	N/A	N/A
calibrate	88 / 165	52 / 138	2.543	0.408	**0.942**	**0.57**	N/A	N/A	N/A	1800	**0.667**	0.087	0.154
turn_to	472 / 4017	463 / 4968	**3.701**	**0.325**	**0.378**	**0.349**	N/A	N/A	N/A	1800	N/A	N/A	N/A

TABLE II: Auto-scale data F_1-Score & F_2-Score

Action	Training Pos / Neg	Testing Pos / Neg	BES F_2-Score Time (s)	BES F_2-Score Precision	BES F_2-Score Recall	BES F_2-Score F1-Score	BES F_1-Score Time (s)	BES F_1-Score Precision	BES F_1-Score Recall	BES F_1-Score F1-Score	Aleph Precision	Aleph Recall	Aleph F1-Score
BLOCKSWORLD													
stack	2479 / 13737	1088 / 6059	150.342	0.352	**0.805**	0.49	121.381	0.768	0.471	**0.583**	**1.0**	0.291	0.451
unstack	2251 / 13965	973 / 6174	156.288	0.371	**0.819**	0.511	123.717	0.889	0.512	**0.65**	**1**	0.23	0.374
pickup	744 / 720	323 / 316	48.243	0.73	**0.997**	0.843	40.77	0.788	0.957	0.864	**0.996**	0.842	**0.912**
putdown	710 / 754	320 / 319	48.633	0.726	**0.984**	0.836	38.201	0.804	0.934	**0.864**	**1**	0.434	0.605
FLOORTILE													
paint-up	4338 / 6230	1850 / 2598	87.203	0.853	0.994	0.918	81.115	0.887	**0.966**	**0.925**	**1**	0.06	0.11
change-color	615 / 745	252 / 308	61.506	0.90	**1**	**0.947**	60.067	**0.9**	**1**	**0.947**	**1**	0.85	0.92
up	2544 / 2740	1177 / 1047	69.215	0.711	**0.976**	0.822	65.198	**0.741**	0.946	**0.831**	N/A	N/A	N/A
down	3395 / 1889	1474 / 750	86.138	0.827	**0.99**	0.901	76.629	0.885	0.954	**0.918**	N/A	N/A	N/A
right	1265 / 617	1264 / 618	70.011	0.834	**0.989**	0.905	65.766	0.865	0.967	**0.913**	N/A	N/A	N/A
left	2935 / 1569	1265 / 617	71.169	0.831	**0.99**	0.903	66.747	**0.867**	0.957	**0.91**	N/A	N/A	N/A
GRIPPER													
drop	5852 / 5852	2506 / 2506	64.608	**1**	**1**	**1**	61.9	**1**	**1**	**1**	1	1	1
pick	5852 / 5852	2506 / 2506	64.318	**1**	**1**	**1**	61.791	**1**	**1**	**1**	**1**	**1**	**1**
move	280 / 280	120 / 120	61.505	**1**	**1**	**1**	60.079	**1**	**1**	**1**	**1**	**1**	**1**
ROVER													
sample_rock	243 / 408	122 / 204	62.44	**0.72**	**0.967**	0.825	60.777	**0.72**	**0.959**	**0.827**	N/A	N/A	N/A
sample_soil	236 / 460	117 / 160	62.356	**0.721**	**0.949**	**0.819**	60.749	**0.721**	**0.949**	**0.819**	N/A	N/A	N/A
calibrate	456 / 866	230 / 353	63.803	0.516	**0.9**	0.656	61.778	**0.756**	0.783	**0.769**	N/A	N/A	N/A
communicate_image_data	583 / 6193	241 / 2415	66.014	0.358	**0.963**	0.522	62.952	**0.454**	0.797	**0.578**	N/A	N/A	N/A
take_image	701 / 5624	328 / 2200	71.367	0.486	**1**	0.654	66.886	**0.63**	0.845	**0.721**	N/A	N/A	N/A
drop	86 / 274	49 / 101	62.791	**0.549**	**0.796**	**0.65**	60.88	0.535	0.776	0.633	N/A	N/A	N/A
communicate_rock_data	448 / 2030	222/1085	62.907	0.374	**0.928**	0.533	61.02	**0.672**	0.829	**0.742**	N/A	N/A	N/A
communicate_soil_data	421 / 2305	176 / 901	62.587	0.441	**0.909**	0.594	60.859	**0.535**	0.835	**0.652**	N/A	N/A	N/A
navigate	666 / 3200	294 / 1334	63.759	0.296	**0.922**	0.448	61.617	**0.5**	0.639	**0.561**	N/A	N/A	N/A
SATELLITE													
switch_on	349 / 694	157 / 267	11.562	**0.37**	**1**	**0.54**	9.678	**0.37**	**1**	**0.54**	N/A	N/A	N/A
switch_off	54 / 989	16 / 408	1.591	**0.038**	**1**	**0.073**	0.194	**0.038**	**1**	**0.073**	N/A	N/A	N/A
take_image	1845 / 26924	862 / 10649	13.177	**0.483**	**1**	**0.651**	7.49	**0.483**	**1**	**0.651**	N/A	N/A	N/A
calibrate	494 / 1088	227 / 415	34.552	**0.354**	**1**	**0.522**	29.741	**0.354**	**1**	**0.522**	N/A	N/A	N/A
turn_to	11019 / 92562	4795 / 37373	1132.521	0.313	**0.747**	0.442	761.372	**0.325**	0.73	**0.449**	N/A	N/A	N/A
ZENOTRAVEL													
refuel	4530 / 11556	1531 / 4409	23.082	0.361	**0.953**	0.524	17.118	**0.429**	0.837	**0.568**	N/A	N/A	N/A
debark	1625 / 15176	571 / 5986	10.211	**0.467**	**1**	**0.636**	5.789	**0.467**	**1**	**0.636**	N/A	N/A	N/A
board	1623 / 15178	571 / 5986	8.635	**0.467**	**1**	**0.636**	4.722	**0.467**	**1**	**0.636**	N/A	N/A	N/A
fly	11542 / 76028	3532 / 27668	326.612	0.184	**0.86**	0.303	223.783	0.234	0.601	**0.337**	N/A	N/A	N/A

In cases where Aleph is able to deliver a result, it again outperforms BES in precision, however BES heavily outperforms Aleph in both recall and F_1-score.

Experiments were also conducted using the autoscale-dataset [31] in order to assess and compare the performance of both BES and Aleph given more training data. Here BES was trained using both the optimisation for F_1-score as well as F_2-score. The F_2-score optimisation will favour recall and was included since this is favourable if the model were to be used for a partial grounding, since a high recall, decreases the chance of classifying a $good$ operator as bad. The results of these experiments are shown on Table II.

For the blocksworld domain we see, that like with the experiments conducted on the IPC data, Aleph outperforms both configurations of BES in terms of precision, however it is generally outperformed on both recall and F_1-score by both BES configurations.

For the gripper domain, we see that both BES configurations as well as Aleph were able to construct models, that could perfectly classify all operators in the testing data. This indicates, that the usefulness of operators in the gripper domain is easily generalisable.

Like with the IPC data, we see that Aleph struggles to construct viable models within the given time limit for more complex domains. In Floortile we see that Aleph was only able to provide a classification model for two of six actions. For both rover, satellite and zenotravel Aleph was unable to provide a model within 30 minutes.

When comparing the results of the two BES configurations, we see that BES with an optimisation for F_2-score achieves a higher recall but a slightly lower F_1-score than BES with an optimisation for F_1-score. This matches expectations, since F_2-score favours recall over precision.

Overall we see, that BES generally outperforms Aleph in terms of recall and F_1-score but with fairly comparable results but Aleph outperforms BES in terms of precision however we also see that Aleph struggles to generate models within 30 minutes for most domains included in these experiments.

These results show, that BES is able to provide consistently well-performing models for every domain, however, these results might not necessarily translate to good performance for a planner using these models to guide a partial grounding. A more accurate depiction of the performance of BES would be to run a planner on a number of difficult PDDL tasks using a partial grounding guided by the models constructed by BES. This would allow us to compare the effectiveness of a planner using a partial grounding guided by BES compared to one guided by other tools like for example Aleph.

X. Conclusion

In this paper, we proposed an approach to learning classification models for PDDL domains, that can classify operators in a given PDDL task as either $good$ or bad. Our approach applies an adaptation of a standard ILP approach and leverages partial order causal link (POCL) graphs to extract additional knowledge from optimal PDDL plans in order to generate relevant rules. Additionally a relational database management system (RDBMS) is utilised to perform evaluation of rules. The classification models are constructed using STreeD, a framework that can create small and optimal binary decision trees.

Empirical results show, that models constructed using our approach generally outperform but have comparable results to models created by the state-of-the-art tool Aleph and our approach's predecessor YAPHG. Additionally, our approach is shown to be able to construct high-performing models for planning domains where both Aleph and YAPHG are unable to provide a model. The effectiveness of our approach in a planning environment remains to be tested, and the benefit of implementing our approach in a planner is therefore uncertain, however, the promising predictive capabilities suggest potential applications in guiding partial grounding and other planning-related tasks.

XI. Acknowledgements

We would like to thank our supervisor Alvaro Torralba for his valuable feedback and guidance for this thesis. Furthermore, we would like to thank Emir Demirovi and Koos van der Linden for providing insight into STreeD. We also express our appreciation to the authors of the GOFAI system for which the system contributed to the achievements in this paper.

REFERENCES

[1] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[2] M. Speranza and C. Vercellis, "Hierarchical models for multi-project planning and scheduling," *European Journal of Operational Research*, vol. 64, no. 2, pp. 312–325, 1993.

[3] M. Ghallab, C. Knoblock, D. Wilkins, A. Barrett, D. Christianson, M. Friedman, C. Kwok, K. Golden, S. Penberthy, D. Smith, Y. Sun, and D. Weld, "Pddl - the planning domain definition language," 08 1998.

[4] M. Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research*, vol. 26, p. 191246, Jul. 2006. [Online]. Available: http://dx.doi.org/10.1613/jair.1705

[5] J. Hoffmann, "Ff: The fast-forward planning system," *AI Magazine*, vol. 22, pp. 57–62, 09 2001.

[6] S. Richter and M. Westphal, "The lama planner: Guiding cost-based anytime planning with landmarks," *Journal of Artificial Intelligence Research*, vol. 39, p. 127177, Sep. 2010. [Online]. Available: http://dx.doi.org/10.1613/jair.2972

[7] D. Gnad, A. Torralba, M. Domnquez, C. Areces, and F. Bustos, "Learning how to ground a plan partial grounding in classical planning," 2019.

[8] D. Gnad, A. Torralba, M. Dominguez, C. Areces, and F. Bustos, "Learning how to ground a plan -partial grounding in classical planning," 01 2019.

[9] C. Linares Lpez, S. Jimnez Celorrio, and ngel Garca Olaya, "The deterministic part of the seventh international planning competition," *Artificial Intelligence*, vol. 223, pp. 82–119, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370215000144

[10] A. Srinivasan, "Aleph manual." [Online]. Available: https://www.cs.ox.ac.uk/activities/programinduction/Aleph/aleph.html

[11] A. D. Frederik Langkilde Jakobsen, Rune Bohnstedt, "Yaphg: Yet another pddl hypothesis generator. aalborg universitet." 2024.

[12] E. Demirovi and P. J. Stuckey, "Optimal decision trees for nonlinear metrics," 2021.

[13] P. Bercher, "A closer look at causal links: Complexity results for delete-relaxation in partial order causal link (pocl) planning," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 31, p. 3645, May 2021.

[14] D. M. McDermott, "The 1998 ai planning systems competition," *AI Magazine*, vol. 21, no. 2, p. 35, Jun. 2000. [Online]. Available: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/1506

[15] R. Černoch and F. Železný, "Speeding up planning through minimal generalizations of partially ordered plans," in *Inductive Logic Programming*, P. Frasconi and F. A. Lisi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 269–276.

[16] A. Botea, M. Enzenberger, M. Mueller, and J. Schaeffer, "Macro-ff: Improving ai planning with automatically learned macro-operators," *Journal of Artificial Intelligence Research*, vol. 24, p. 581621, Oct. 2005. [Online]. Available: http://dx.doi.org/10.1613/jair.1696

[17] F. P. Manuel Heusner, Martin Wehrle and M. Helmert, "Under-approximation refinement for classical planning," 2024. [Online]. Available: https://aaai.org/papers/00365-13678-under-approximation-refinement-for-classical-planning/

[18] A. Cropper and R. Morel, "Learning programs by learning from failures," 05 2020.

[19] H. Geffner and B. Bonet, *Classical Planning: Full Information and Deterministic Actions*. Cham: Springer International Publishing, 2013, pp. 15–36. [Online]. Available: https://doi.org/10.1007/978-3-031-01564-9_2

[20] M. Helmert, "Concise finite-domain representations for pddl planning tasks," *Artificial Intelligence*, vol. 173, no. 5, pp. 503–535, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370208001926

[21] D. Gnad, . Torralba, M. Domnguez, C. Areces, and F. Bustos, "Learning how to ground a plan partial grounding in classical planning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 7602–7609, Jul. 2019. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/4753

[22] T. Silver, S. Dan, K. Srinivas, J. B. Tenenbaum, L. P. Kaelbling, and M. Katz, "Generalized planning in pddl domains with pretrained large language models," 2023.

[23] J. W. Lloyd, *Foundations of Logic Programming*. Springer, 1984.

[24] L. D. Raedt, *Inductive Logic Programming*. Boston, MA: Springer US, 2010, pp. 529–537. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_396

[25] S.-H. Nienhuys-Cheng and R. d. Wolf, *Foundations of Inductive Logic Programming*. Springer, 1997.

[26] D. S. Weld, "An introduction to least commitment planning," *AI Magazine*, vol. 15, 1994.

[27] B. Seegebarth, F. Mller, B. Schattenberg, and S. Biundo, "Making hybrid plans more clear to human users - a formal approach for generating sound explanations," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 22, no. 1, pp. 225–233, May 2012. [Online]. Available: https://ojs.aaai.org/index.php/ICAPS/article/view/13503

[28] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM*, vol. 13, no. 6, p. 377387, jun 1970. [Online]. Available: https://doi.org/10.1145/362384.362685

[29] S. Ceri and G. Gottlob, "Translating sql into relational algebra: Optimization, semantics, and equivalence of sql queries," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 4, pp. 324–345, 1985.

[30] K. P. Gaffney, M. Prammer, L. Brasfield, D. R. Hipp, D. Kennedy, and J. M. Patel, "Sqlite: Past, present, and future," *Proc. VLDB Endow.*, vol. 15, no. 12, p. 35353547, aug 2022. [Online]. Available: https://doi.org/10.14778/3554821.3554842

[31] A. Torralba, "autoscale-learning." [Online]. Available: https://github.com/alvaro-torralba/autoscale-learning

[32] C. D. Manning, P. Raghavan, and H. Schtze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.