Integrating Deep Learning Techniques into Hierarchical Task Planning for Effect and Heuristic Predictions in 2D Domains

Michael Staud

Ulm University, Institute of Artifical Intelligence, D-89069 Ulm, Germany
michael.staud@uni-ulm.de

Abstract

In this paper, we present a novel approach that combines *Hierarchical Task Planning* (HTN) with deep learning techniques to address the challenges of scalability and efficiency in large-scale planning problems. Building upon the Hierarchical World State Planning (HWSP) algorithm, our method utilizes a multi-layered world state representation, which allows for planning at abstract levels without the need to consider lower-level details. We propose a deep learning method for predicting the effects of abstract tasks, which opens the door to enhancements in both planning performance and plan quality. Additionally, we employ the same approach to create a domain-dependent planning heuristic. Our contributions demonstrate the potential of integrating HTN planning with deep learning techniques, paving the way for future research in various application domains such as robotics, logistics, and urban planning. The proposed approach employs standard deep learning techniques, ensuring adaptability as the state of the art advances.

1 Introduction

Automatic planning is a fundamental problem in *artificial intelligence* with numerous practical applications, such as robotics and logistics. One approach to tackling complex problems is total-order *Hierarchical Task Planning* (HTN), which involves devising an abstract plan and gradually decomposing it into a concrete plan (Ghallab, Nau, and Traverso 2004, 238). However, existing planning methods often struggle with scalability and efficiency, particularly for large, complex problems. In this paper, we propose a novel approach that combines HTN planning with deep learning techniques to overcome these limitations.

Recent progress in *deep reinforcement learning* (Arulkumaran et al. 2017) has demonstrated the efficiency of this approach, with the hierarchical policies improving reinforcement agent performance and expediting the learning process (Vezhnevets et al. 2017; Huang 2020). By combining classical planning with reinforcement learning (Rivlin, Hazan, and Karpas 2020), it is possible to efficiently solve significantly larger planning instances. While not directly comparable, since these results involve policy generation, they offer insight into the potential benefits of deep learning for hierarchical planning problems. Additionally, reinforcement learning has been utilized in HTN planning to improve plan quality (Hogg, Kuter, and Munoz-Avila 2010). The Refinement Acting Engine (RAE), which also uses a hierarchical task structure like HTN planning but is an online algorithm, can also be successfully combined with a learning algorithm (Patra et al. 2020).

And in classical planning, deep learning has been used to learn planning heuristics. For instance, ASNets (Toyer et al. 2020) employ a clever approach to create domain-dependent heuristics by encoding action schemas in a neural network, allowing weight sharing across different problems. These can also be combined with *Monte-Carlo Tree Search* (Shen et al. 2019). However, this method requires problem grounding, which is inefficient for large problems like urban simulation (Staud 2022). Alternatively, domain-independent heuristics can be generated by providing a fixed set of features to a neural network (Gomoluch et al. 2017; Trunda and Barták 2020), or through the use of hypergraph networks (Shen, Trevizan, and Thiébaux 2020). In contrast, we aim to develop a domain-dependent heuristic designed to exploit the unique features and structure of a domain.

Symbolic Networks (Garg, Bajpai et al. 2020) achieve *state-of-the-art* performance on some planning benchmarks but are less relevant to our approach due to their focus on relational MDPs rather than HTN planning.

In this paper, we build upon the *Hierarchical World State Planning* (HWSP) algorithm (Staud 2022). This algorithm enhances total-ordered HTN planning by using a multi-layered world state representation, where the higher layers offer a more abstract perspective on the world state than the lower layers. By planning at a higher layer, there is no need to consider the details of the lower layers. The domain designer defines the effects of so-called separable abstract tasks, which can be applied directly without decomposition on the higher layers.

Our proposed deep learning method for predicting the effects of separable abstract tasks within the HWSP algorithm not only has the potential to improve planning performance but also to enhance plan quality, enabling more effective solutions for large-scale planning problems. We employ standard deep learning techniques (Géron 2022), allowing for easy adaptation as the state of the art changes. The use of specialized DNNs, such as ASNets, would anchor us to a specific architecture, limiting this flexibility. Similarly, our decision to convert the state into a 2D representation, in-

stead of directly feeding the state into the DNNs, stems from our goal of retaining compatibility with planning algorithms that use state representations beyond a set of atoms. This approach also facilitates the use of diverse DNN architectures, further broadening the applicability of our work. Furthermore, we utilize the same approach to estimate the number of steps to the goal, thereby creating a domain-dependent heuristic.

Our contributions are:

- A deep learning method for predicting the effects of abstract tasks within the *Hierarchical World State Planning* algorithm (Staud 2022).
- A deep learning method capable of heuristic learning. The heuristic can then be used by the HTN planning algorithm (Höller et al. 2019).

In the subsequent sections, we will first provide an overview of the Hierarchical World State Planning algorithm. Next, we will explain the process of inputting the world state into a neural network. We will then discuss the prediction mechanism and elaborate on the neural network architectures employed in our study. Lastly, we will present and analyze the results obtained from our experiments.

2 Hierarchical World State Planning

This section describes the Hierarchical World State Planning (HWSP) (Staud 2022) algorithm, which extends the hierarchical task network (HTN) planning paradigm. It introduces an innovative type of abstract tasks, referred to as *separable abstract tasks*, devised to enhance planning performance. This enhancement is achieved by partitioning the planning process into smaller, manageable sub-processes functioning across multiple layers of abstraction.

The set of all constants, variables, and atoms are designated as C, V, and A, respectively. A literal is an atom or its negation, while an atom is a predicate applied to a tuple of terms. Here, a term could be a constant or a variable. Every predicate p belongs to the set P.

2.1 Hierarchical Task Planning Domain

A hierarchical task planning domain is denoted as $D = (T_a, T_p, M)$, comprising three finite sets. Here, T_a is the set of abstract tasks, T_p represents primitive tasks, and M consists of methods. Both primitive and abstract tasks are tuples in the form $t(\overline{\tau}) = \langle prec_t(\overline{\tau}), \mathit{eff}_t(\overline{\tau})\rangle$. Each task is characterized by a precondition $prec_t(\overline{\tau})$, an effect $\mathit{eff}_t(\overline{\tau})$, and a set of parameters $\overline{\tau}$. The effects can modify atoms and fluents in the world state, while the precondition can examine if a particular atom is present in the world state or if a numerical equation is fulfilled.

A method is a tuple $m = \langle t_a(\overline{\tau}_m), P_m\rangle$, where $t_a(\overline{\tau}_m)$ is the abstract task it can decompose, and P_m is a set of plan steps, with $\overline{\tau}_m$ as the parameters of the method. An abstract task can be decomposed via methods into other tasks, where the plan steps P_m of method m replace the original abstract task in the plan.

A plan is a total ordered sequence of tasks. A problem is defined as a tuple $P = \langle init_P, goal_P, PS_P\rangle$, consisting of the initial plan PS_P, the initial state $init_P$, and the goal state $goal_P$. A solution is a plan where each task is an action (primitive task), satisfying its precondition at each step, thereby transforming the initial state into the goal state. A world state $w \in W$ is a set of atoms and a finite number of numerical fluents.

2.2 Multi-Layered World State

The HWSP algorithm divides the world state into n layers. The layer n encapsulates the actual world state containing factual information, whereas layer $l < n$ is more abstract than layer $l + 1$. Each predicate p and task t is associated with a specific layer $\hat{l}(p) = l$, and can only exist on that layer. The layer function is formulated as $\hat{l} : T_a \cup T_p \cup P \to \mathbb{N}$. Predicates at layer $l < n$ are derived predicates (Staud 2022; Edelkamp and Hoffmann 2004), while those at layer n are non-derived predicates. Derived predicates can only rely on predicates of layer $l+1$, facilitating information flow between layers. Consequently, the planning domain is then expressed as a tuple $D = (T_a, T_p, M, \hat{l})$.

2.3 Task Representation

Each task t is confined to having predicates in its effects that operate only on the same layer $\hat{l}(t) = l$. However, its preconditions can contain predicates p from layer $\hat{l}(p) < l$. Standard abstract tasks can only decompose into tasks on the same layer and lack effects. In contrast, separable abstract tasks must have effects, which must be defined by the domain designers. These effects should approximate the indirect influence the separable task have on the derived predicates at its layer.

Distinct from traditional abstract tasks that are instantaneously decomposed into a method, separable abstract tasks function as markers that signal the requirement for decomposition in a separate planning process (see section 2.6) that operates on layer $l+1$. These tasks are subsequently decomposed by a method akin to a standard abstract task in the new planning process.

It is important to note that unlike derived predicates in PDDL (Edelkamp and Hoffmann 2004), derived predicates can appear in effects within this planning algorithm. This is possible because they are treated as non-derived predicates in the child planning process.

2.4 Fluents and Functions

In contrast to the planning algorithm outlined by Staud (2022), our approach incorporates the support for numerical fluents as described by (Fox and Long 2003). These fluents are managed identically to predicates, and we make use of derived functions to handle these numerical fluents similarly to the way derived predicates are treated (Edelkamp and Hoffmann 2004).

The derived functions utilize numerical formulas rather than logical ones. These formulas support basic arithmetic operations such as addition, subtraction, multiplication, and division. Further, these derived functions provide the capability to aggregate the values of a set of numerical fluents using the sum operation.

The set over which the aggregation takes place is determined by a PDDL goal description (Kovacs 2011), in a manner akin to the forall operator used in PDDL. This implies that a derived function can apply a mathematical operation on a subset of numerical fluents defined by a PDDL goal description .

Notably, the derived functions can be modified via effects and can be involved in the preconditions of tasks in a child planning process. We also extend the capability to include operations that count the number of elements that fulfill a goal description. This allows for calculations such as averaging, which requires the count of the relevant fluents.

This makes derived functions a flexible and powerful tool in managing numerical fluents, expanding the range of possible planning tasks and scenarios that can be modeled.

2.5 Main Planning Algorithm

The main planning algorithm maintains a stack of planning processes, where only the top-most process is active. The algorithm comprises a main plan, which contains the final resultant plan, and the current world state, which is modified when a new primitive task is appended to the main plan (see figure 1). Each planning process is distinct, possessing its own unique plan and set of goals.

The initial planning process, which always operates on layer 1, uses the most abstract layer of the world state, the original problem goal, and the initial plan derived from the problem. The goal is restricted to predicates of layer 1. However, as these are derived predicates and depend recursively on predicates of layer n, any goal can be transformed into a form supported by the planner through the introduction of derived predicates.

The system is designed to support full backtracking, allowing it to continue across planning process boundaries (Russell et al. 2010). This feature ensures the completeness of the planning algorithm (Staud 2022).

2.6 Child Planning Algorithm

In each iteration, the main planning algorithm triggers the *child planning algorithm* within the top-most *planning process* on the stack. This process functions on a distinct layer l. The child planning algorithm subsequently returns a new task, which can either be a primitive task or a separable abstract task. If it is a primitive task, it is incorporated into the main plan, and the current world state is updated by applying the task's resultant effects. If the task's preconditions are unfulfilled or if no task is returned, the system reverts via backtracking. If the returned task is a separable abstract task, a novel planning process is set up to realize its effects, operating on layer $l + 1$. A planning process functions in the following manner:

Planning Process Initialization: The initiating plan of a process is comprised of tasks present in the method that was used to decompose the separable abstract task that created it. The goal of the planning process aligns with the effects of the task (Staud 2022).

Method Selection and Decomposition: The child planning process uses the Monte Carlo Tree Search (MCTS) algorithm (Kocsis and Szepesvári 2006), complemented by forward decomposition, to generate plans (with the H0 heuristic (Ghallab, Nau, and Traverso 2004)). Instead of producing a complete plan as might be expected, the child planning process returns a single task that has a high likelihood of guiding the main planning process towards the goal. The child planning process treats tasks differently: separable abstract tasks are treated as primitive tasks, with no need for further decomposition, whereas derived predicates are handled as non-derived predicates. However, standard abstract tasks are still decomposed through methods, as their decomposition is strictly restricted to tasks within the same layer.
This unique approach enables the child planning algorithm to formulate plans on the abstract layer as though dealing with a non-hierarchical planning problem. A significant reduction in the world state is achieved by exclusively utilizing the horizon (see section 2.7, thereby enhancing performance. It is crucial for the declared effects of the separable abstract tasks, as defined by the domain designer, to closely align with their actual effects when fully executed in the main planner. To increase precision, we employ neural networks in the method proposed in this paper. This approach leads to a decrease in the required amount of backtracking and an overall enhancement of plan quality

Planning Process Termination: Upon accomplishing its goal in the world state of the main planning process, a child planning process is discarded from the stack.

2.7 Horizon and Planning in the Planning Process

The horizon, composed of a set of atoms and fluents, plays a pivotal role in the child planning algorithm. It encapsulates only the portion of the world state pertinent to the child planning process. All atoms and fluents present within the task decomposition graph of the separable abstract graph, as outlined by (Bercher, Keen, and Biundo 2014), are included in this set. Importantly, the horizon is not a static construct, it is dynamically reconstructed from the world state of the main planning algorithm every time a new task is added the main plan (Staud 2022).

2.8 Completion of Planning

Planning processes are tasked with identifying subsequent actions for integration into the main plan. Upon the successful fulfillment of the original planning problem's goal, the main planning process is finished. Then, the main plan contains the result. If no plan could be found the algorithm returns an empty plan.

2.9 Illustrative Example

The concepts discussed in this paper can be best understood through an example scenario: a robot navigation challenge in a grid environment populated with obstacles. The robot's task is to navigate from its starting position to a designated goal location while circumventing these obstacles.

Layered Representation The world state can be represented in multiple layers:

Algorithm 1: The Main Planning Algorithm of the Hierarchical World State Planning (HWSP) (Staud 2022) algorithm.

```
1  Function MainPlanningProcess:
2   while not stack.isEmpty():
3    cp = stack.top
4    new_task = cp.getNextTask()
5    if (cp.finished())
6     stack.remove(cp)
7    if new_task is Primitive_Task:
8     add_to_plan(new_task)
9     update_world_state(new_task)
10   elif new_task is
         Separable_Abstract_Task:
11    newp = new PlanningProcess(new_task)
12    stack.push(newp)
13   else:
14    backtrack()
15   if goal_achieved():
16     return plan
17   return None
```

- Layer n: This represents the actual grid, complete with the robot's current location, the goal location, and the obstacles.
- Layer $l < n$: This is a simplified rendition of the grid, with regions consolidated into higher-layer zones (e.g., rooms within a building).

Planning Process The primary planning algorithm begins with the most abstract layer, where the objective is to maneuver the robot from one high-layer zone to another. As the algorithm advances, it may encounter separable abstract tasks, such as transitioning from one room to another. In such an instance, a fresh planning process will be established to operate on the subsequent layer ($l+1$), offering a more detailed view of the environment.

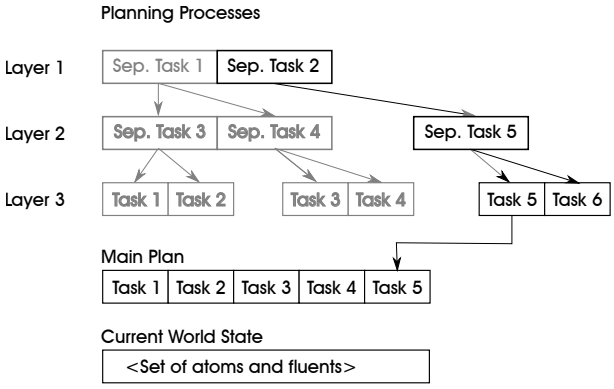

Figure 1: Adapted from the paper by Staud (2022), this figure provides an example of what the state of the main planning algorithm might look like in a domain with 3 layers. Completed processes are marked in grey. The current state of the stack would be: Sep. Task 1, Sep. Task 5.

3 Representation of the World State

Deep Learning usually takes a fixed-size representation as input (Géron 2022). To generate it out of the world state and enable predictions, a 2D representation is chosen due to its suitability for planning domains that are fundamentally two-dimensional, such as ware transport and robot steering. For domains with different dimensionalities, 1D or 3D representations can be used, which makes the algorithm, with modifications, also usable in this case. The usage of a 2D input image to guide planning is well explored in reinforcement learning (Mnih et al. 2013) or algorithm selection for planning (Sigurdson et al. 2019; Kaduri, Boyarski, and Stern 2020) and has been shown to be effective. Note that we will only project one layer of the multi-layer world state as the planning processes also have only access to one layer of the world state.

A 2D domain is a domain for which a projection $f_p : W \to I$ of its world state w into a 2D space I exists that has the following properties:

- $i \in I$ is a RGB pixels image. This means it is a matrix with dimensions $M \times N \times 3$.
- f_p is consistent: let $w_1 \in W$ and $w_2 \in W$ be two world states. When they are equivalent in regarding planning, then it holds $f_p(w_1) = f_p(w_2)$. When w_1 and w_2 are not equivalent it holds $f_p(w_1) \neq f_p(w_2)$.
- f_p is reversible: for all 2D images $i \in I$ that were generated by f_p we can construct a world state $w \in W$ such that $f(w) = i$. And this world state is equivalent to the original world state in terms of planning.
- The encoding of f_p should be in a way that allow a deep learning algorithm to recognize and learn patterns. So the spacial organization of the image is meaningful and informative.

The last property isn't fully formalized due to ongoing research about what specific neural networks can detect. Important is that the spatial locality is preserved and that the complexity in the output image is minimized while still preserving the information.

We propose a series of annotations to create the required domain-specific projection function. These annotations, added to a PDDL domain or problem file (Fox and Long 2003), can be used to automatically render a world state to an image. The annotation system allows the user to define position and shape information for every atom, object, predicate or function. Annotations are inspired by Java annotations (Gosling et al. 2005), and multiple annotations are possible.

- `;@position(vec2(x,y))`: Defines the position of an object, with x and y as floats in image coordinates.
- `;@position(arg)`: Sets the position of an atom or fluent equal to the object of the given argument. It is used as a shift in combination with `;@propagate`.
- `;@appearance(shape, size, zlayer, {<paint attributes>})`: Defines the appearance of an object, with the shape as either a box, ellipse, PNG, or a line; the size as a `vec2`; and the `zlayer` as the draw order. The paint attributes contain pen and brush style, with colors specified directly or derived from a fluent. Each object also has an internal instance

ID, which is automatically encoded via hashing into the final color of the object to differentiate between multiple instances in the final image.

- `;@propagate(source, target)`: Can only be applied to an atom. Propagates the position of the object in the source argument to the object in the target argument.

The line shape can only be applied to an atom or fluent and takes the position of two objects in the arguments to determine the orientation of the line. The objects on the same z-layer are not rendered on top of each other even if they have the same position. Instead, they will be arranged deterministically by a layouting algorithm to ensure that every atom, object, or fluent is visible. We support other features like, for example, a slot mechanism to give the user more control but we will only present the basic features in this paper.

The annotations are placed in the comments so that if a PDDL planner does not support them, they can be ignored. The resulting projection function can be used not only for rendering the world state but also for rendering the goal state.

3.1 Algorithm

The algorithm accepts as input a world state, consisting of a set of atoms and a finite number of numerical fluents. It comprises two integral functions:

The *draw* function initiates a set for object positions, defined directly via annotations in the problem file, and computes the appearance and rectangle information for each object. This function then proceeds to call the *propagate* function iteratively until no further propagation transpires. Post initial propagation, the function processes any residual objects that have not been assigned positions, bestowing default positions and appearance information. Another round of propagation is undertaken to update the positions of these newly handled objects, continuing until no additional propagation occurs. The *draw* function concludes by sorting objects according to their z-layer, thereby ensuring objects with lower z-layer values are rendered first. The object's attributes such as shape, size, z-layer, position, and paint are taken into consideration during this drawing phase. The function adeptly handles a variety of shapes, including boxes, ellipses and lines, adjusting the pen and brush styles and colors as required.

The *propagate* function accepts a set of determined object positions, the world state, and a map describing child-parent relationships as defined by the annotation. It traverses the atoms contained within the world state and applies propagation rules specified by the annotations to compute new object positions. If an object has a valid position, it propagates this position to its child objects. If the space of a child object is already occupied by another object on the same layer, thus violating the properties of the projection function, the child object will be displaced until a free space in the same z-layer is located. A straightforward rule is employed for this, shifting any object along the x-axis to the right until a free space is found in the same z-layer or until the right image border is reached.

Thus, objects are arranged in a deterministic manner. Note, it is quite straightforward for a PDDL domain designer to design an annotation that breaches the conditions of the projection function, such as insufficient space in the image, child-parent relationships not being visible in the image due to excessive object displacement, or predefined object positions overlapping. Under such circumstances, the system responds by returning an error message to the user.

3.2 Example

The proposed annotation system's utility is demonstrated via an example from the airport domain (Anders 2015). This domain encompasses three types of objects: airports, planes, and cargos, with the objective being to transport cargo from one airport to another utilizing planes.

Here is a simplified annotated PDDL domain for this problem:

```
(define (domain airport)
  (:types
    gobject - object
    ;@appearance(box, vec2(4,4), 0, {
        color = white})
    airport - gobject
    ;@appearance(box, vec2(2,2), 1, {
        color = magenta})
    cargotype - gobject
    ;@appearance(ellipse, vec2(3,3), 1,
        {color = red})
    planetype - gobject
  )
  (:predicates
    ;@propagate(?obj2, ?obj1) ;@position
        (vec2(2,2))
    (in ?obj1 - gobject ?obj2 - gobject)
    ;@propagate(?airport, ?obj1) ;
        @position(vec2(-2,2))
    (at ?obj1 - gobject ?airport -
        airport)
    (cargo ?obj1 - gobject)
    (plane ?obj1 - gobject)
    (airport ?obj1 - gobject)
  )
  ... ; Rest of the domain
)
```

In the provided PDDL file, different object types' appearances are defined using `;@appearance` annotations. For instance, an airport is denoted by a 4x4 pixel white box as specified by the `;@appearance(box, vec2(4,4), 0, color = white)` annotation.

The `;@position` and `;@propagate` annotations determine how object positions are established. For instance, the `;@propagate(?obj2, ?obj1)` annotation associated with the "in" predicate suggests that the position of `?obj1` mirrors that of `?obj2`, and the `;@position(vec2(2,2))` annotation stipulates that `?obj1` is displaced by `vec2(2,2)` from the position of `?obj2`. In the PDDL problem file, the specific positions of the airports are established.

This annotated PDDL domain can now be leveraged by the planning algorithm to generate a two-dimensional projection of the world state, which can serve as an input to a

deep learning algorithm.

4 Prediction for Planning

In this section, we discuss how we employ neural networks to predict various important aspects of planning, including the effects of separable abstract tasks and the number of actions required to reach a goal.

4.1 Effects of Abstract Tasks

We aim to predict two key factors for separable abstract tasks: simple effects and fluent effects.

Simple Effects Simple Effects (McDermott et al. 1998) add or remove atoms. Our goal is to predict whether these effects will be fulfilled when a separable abstract task is executed. We use a neural network that returns a probability corresponding to the likelihood that the effect will be applied. To train the network we are using the binary cross-entropy loss functions (Good 1952).

Fluent Effects Fluent effects (Fox and Long 2003) change fluents, which are numerical values. In this case, we do not predict a probability; instead, we directly predict the delta value (residual learning) of the fluent using a neural network. To train the network we are using mean absolute error (MAE) (Bishop 2006) as loss function. The network performs *deep regression* which is competitive to problem specific estimators (Lathuilière et al. 2019).

Input To predict these effects, we first encode both the current world state and the world state after the separable abstract task was applied with the default effects defined by the domain designer (Staud 2022). This process results in a rough approximation of the new world state. However, this approximation is sufficient for the neural network to identify which separable abstract task was applied. Note that when the prediction of the effects is not correct only the performance of the algorithm will decrease, it will not affect completeness (Staud 2022).

Both world states are then converted into two separate 2D representations, which are subsequently provided as input along with their delta to the neural network. In addition, we also provide the current values of the fluents to assist the neural network in predicting the delta changes of the fluents.

4.2 Heuristic Learning

Deep learning can be used to create planning heuristics, which estimate the number of planning steps required to reach a goal (Shen et al. 2019; Gomoluch et al. 2017; Shen, Trevizan, and Thiébaux 2020). We train a neural network using a 2D representation of the current world state and the goal state. The training data is generated from example domains and includes the states, and the actual number of steps required to reach the goal. We determine the number of steps using a planning system, which supports PDDL 2.1 (Fox and Long 2003) and can return the minimal amount of actions to reach a goal. As loss function we use MSE (Bishop 2006). The network performs *deep regression* which is competitive to problem specific estimator (Lathuilière et al. 2019).

It is not easy to generate the (full) goal state out of the initial state and the goals in the problem. In our tests, we determined the set of predicates which were stationary and simply combined them with the goal from the problem. This is mostly sufficient. If not the domain designer has to provide additional annotations so that the (full) goal state is generated correctly.

It's important to note that the resulting heuristic is likely to be non-admissible (Ghallab, Nau, and Traverso 2004), as we're approximating the number of steps. This heuristic can then be used in an HTN planner (Höller et al. 2019).

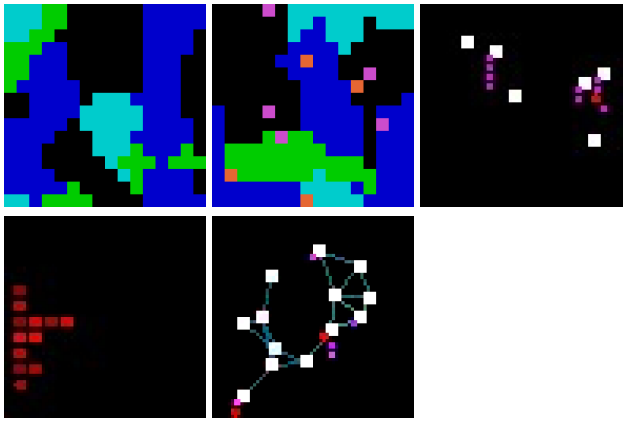

Figure 2: The images displayed, arranged from left to right and top to bottom, represent the following domains: City1, City2, Airport, Blocksworld, and Transport. These graphical outputs were generated by applying the projection functions, as defined by the annotation process detailed in section 3. They provide a visual representation of each domain's state.

4.3 Architecture of the Neural Network

We employ three distinct types of neural networks to evaluate our approach:

- **Conv**: This network consists of three convolution layers (LeCun et al. 1998) without max pooling, followed by a fully connected dense layer. Batch normalization (Ioffe and Szegedy 2015) is applied after each layer:
  - Conv2D(16, kernelsize=1), ReLU
  - Conv2D(32, kernelsize=3), ReLU
  - Conv2D(64, kernelsize=3), ReLU
  - Flatten()
  - Dense(n, activation=<see text>)

  The network has 115940 weights. It uses the convolution layers to first determine complex features in the input and then uses a highly connected dense layer for regression.
- **Conv Max**: Like *Conv*, but this network includes max pooling $(2, 2)$ after the convolutional layers (not including the first one). The network has 26340 weights, and this kind of architecture is usually used for image classification tasks (LeCun et al. 1998). Additionally, it can also be used for the analysis of non-visual 2D data, like a spectrogram of a word (Warden and Situnayake 2019).
- **ResNet**: This network is a truncated version of *ResNet-34*, which is used for image classification and showed

Domain Variable	Conv Fluents	Integer	Atoms	Conv Max Fluents	Integer	Atoms	ResNet Fluents	Integer	Atoms
City1 All	0.3523	0.3732	0.1295	0.3885	0.3756	0.2830	0.0075	0.0087	0.088
City1 Industrial	0.2547	0.2674	0.0299	0.2765	0.2832	0.2181	0.0177	0.0204	0.0109
City1 Residential	0.2531	0.2575	0.0324	0.2805	0.2932	0.2243	0.0199	0.0256	0.0021
City1 Commercial	0.2488	0.2763	0.0415	0.2751	0.2812	0.2256	0.0178	0.0254	0.0166
City2 All	0.1271	0.1343	0.2456	0.2812	0.3012	0.3042	0.0090	0.0120	0.0186
City2 Industrial	0.3083	0.3123	0.1087	0.2726	0.2823	0.2304	0.01328	0.0170	0.0162
City2 Residential	0.3158	0.3323	0.1021	0.2675	0.2742	0.2202	0.0204	0.0232	0.01256
City2 Commercial	0.3212	0.3202	0.1081	0.2695	0.2743	0.2248	0.0191	0.0205	0.0202
City2 Mine	0.2834	0.3004	0.0907	0.2434	0.2404	0.2193	0.0235	0.0223	0.0158

Table 1: Prediction of separable abstract task effects using three neural networks (Conv, Conv Max, and ResNet) across different city domains. The table shows the mean absolute error (MAE) (Bishop 2006) on normalized data and the binary cross-entropy loss for atoms. The networks were trained on 200 epochs, although fewer epochs may suffice for less stringent accuracy requirements. The dataset size was 16384, with 20% of the data used for validation (Bishop 2006). The size of each 2D representation was 16×16.

Domain	Conv	Conv Max	ResNet	ResNet +H0
Airport	1.4943	0.8883	0.8272	0.2899
Blocksworld	4.345	1.5293	1.5322	1.995
Transport	3.423	1.5075	1.6820	0.5117

Table 2: Heuristic learning results for various domains using Conv, Conv Max, and ResNet neural networks. Displayed is the unnormalized loss when predicting the number of stes to the goal. The input data size is larger (64×64), requiring two additional residual or convolutional units. ResNet+H0 represents a combination of the traditional H0 planning heuristic with a neural network, where the neural network outputs a delta value that is added to the H0 heuristic. It shows the best results.

Probability	City1	City2
0.01	6.2480	6.4520
0.1000	9.1330	12.8180
0.2000	15.2060	34.3300
0.3000	29.6240	100.1370
0.4000	59.2270	402.3630
0.5000	134.0850	2066.6430

Table 3: The table demonstrates the influence of prediction quality on the average number of planning processes generated by the primary planning algorithm. The "Probability" column indicates the frequency at which an atomic effect is inaccurately predicted. Such mispredictions necessitate backtracking in the overall system, which consequently increases the number of planning processes. A higher volume of planning processes typically leads to a decrease in overall system performance. Hence, the improvement of prediction quality can significantly enhance the planning algorithm's efficiency by reducing the need for backtracking and subsequently minimizing the number of planning process processes required.

very high performance in this task (He et al. 2016):
- Conv2D(64, kernelsize=7, stride=2), BatchNormalization, Relu, MaxPooling2D($(3, 3)$, stride=2)
- ResidualUnit(64, stride=1) $\times 3$
- ResidualUnit(128, stride=2)
- ResidualUnit(128, stride=1) $\times 3$
- GlobalAvgPool2D(), Flatten()
- Dense(n, activation=<see text>)

The network has 1367524 weights. The ResidualUnit is composed of two convolution layers with ReLU activation and batch normalization, in addition to a skip connection (He et al. 2016).

In each neural network, the dense layer employs linear activation (Géron 2022) for predicting fluents and sigmoid activation (Rumelhart, Hinton, and Williams 1986) for atoms. To enable efficient training, we implement data normalization (Bishop 2006).

5 Results

In this section, we first present results from training the neural network. The domains were manually annotated domains in order to create a 2D representation of the world state.

- **Prediction of Abstract Task Effects** (see Table 1): We implemented a city domain like to the one used in the paper (Staud 2022) to test the prediction of abstract task effects which we will call *City1*. We also created a more sophisticated urban domain called *City2*, which will additionally simulate effects like an electric power grid, criminality, and fires. In these domains, we consider several separable abstract tasks (industrial, residential, commercial, mine). The training data set is created via planning and simulation of the effects of the abstract tasks.
  We used one neural network to predict the effects of all abstract task. Additionally we also measured what happens when we use a seperate neural network for each abstract task.
- **Heuristic Learning** (see Table 2): We test the heuristic learning approach on the airport domain (Anders 2015), as well as the blocksworld and transport domains, which

were presented as challenges in the International Planning Competition and are available on GitHub (Seipp, Torralba, and Hoffmann 2022).

The data presented in the tables 1 and 2 indicates that the impact of an abstract task and the number of steps to the goal can be approximated with a high degree of accuracy using neural networks. As anticipated, the ResNet variant outperformed the other two networks. Intriguingly, the general ResNet variant, which approximates all abstract tasks simultaneously, yielded better results than training separate neural networks for each abstract task. Conversely, the Convolution variants exhibited improved performance when individual networks were trained. The performance of the Conv Max network equals or surpasses that of the standard Conv network when predicting fluents; however, this superiority is not observed in the prediction of atoms. This suggests that the Max Pooling operation may inadvertently discard crucial information necessary for accurate atom prediction. On the other hand, Conv might perform poorly on large representations due to the dense network.

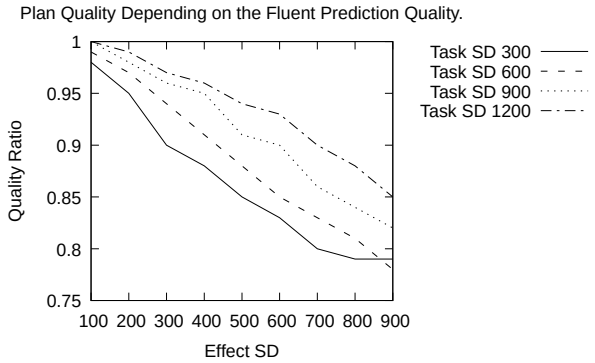

Figure 3: The graph illustrates how the quality of numerical fluent effect predictions impacts plan quality within a simplified 'City1' domain. This domain has steady income-building tasks (which make buildings) without secondary effects like crime or fire. The main plan quality metric is the five-year city treasury income. Ideally, the planner would select the highest income-generating building at each step, but prediction errors skew this process. The 'quality ratio' (y-axis) is the actual gain to maximum possible gain. We assumed a *Gaussian prediction* error distribution with a mean of 0, and the standard deviation denoted by 'Effect SD' (x-axis). The curves show the influence of different numerical fluent effect distribution in the tasks. Notably, diverse original numerical effects lessen the error's impact on plan quality, while similar fluent effects heighten the influence of prediction errors, decreasing plan quality.

It is also noteworthy that the measurements can be divided into two categories: those that predict the data with high accuracy and those that have a relatively low accuracy. This observation suggests that when the accuracy is low, the neural network might not have successfully fully generalized the underlying structure of the problem. In our experiments, we always employed separate networks to predict atoms and fluents because we saw a performance drop when using a single network for both tasks.

We conducted an evaluation to measure the influence of our method on planning performance and the quality of the plan produced. Table 3 provides a comparative analysis of the number of planning processes generated, depending on the probability of an incorrect prediction of an atom's effect. An erroneous prediction triggers backtracking. For this specific analysis, the planner employs the *City1* domain (Staud 2022). The *City2* domain will create more planning processes as its tasks have more effects and backtracking is necessary when a single one fails.

Furthermore, we evaluated the impact of inaccurately predicted numerical fluents on the quality of the plan. The objective within the *City1* domain is to optimize the quantity of money generated by the city. Graph 3 depicts the ratio of actual money generated to the optimal possible generation of money.

Upon reviewing the results, it appears that the most effective course of action for automatic planning is to utilize a ResNet, as this network consistently delivers the best outcomes. The primary drawback, however, is the substantial amount of data that must be processed each time the network is evaluated. As such, it would be prudent to primarily adopt this approach in complex domains like *City1* or *City2* where a traditional general heuristic would take too much time to evaluate.

6 Conclusions

In this paper, we have presented a practical and simple method for leveraging deep learning to improve the performance of automatic planning. Our approach requires no complex data structures and can be easily integrated into existing planners. Notably, the ResNet+H0 combination demonstrates a significant improvement over other tested networks. This finding hints at the potential of combining traditional planning heuristics with neural networks to achieve superior performance.

Looking forward, we plan to extend our approach to automatically generate abstraction hierarchies for hierarchical world state planners. This will allow us to transform non-hierarchical domains into hierarchical ones and enjoy the performance benefits which come with the hierarchical world state (Staud 2022). Overall, we believe that our research provides a promising direction for integrating deep learning with automated planning systems to achieve better performance and scalability.

References

Anders, A. 2015. Planning Exercises. Https://github.com/-arii/planning_exercises, Accessed: 20.3.2023.

Arulkumaran, K.; Deisenroth, M. P.; Brundage, M.; and Bharath, A. A. 2017. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Processing Magazine*, 34(6): 26–38.

Bercher, P.; Keen, S.; and Biundo, S. 2014. Hybrid Planning Heuristics Based on Task Decomposition Graphs. In *SoCS 2014*, 35–43. AAAI Press.

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning*. Berlin, Heidelberg: Springer-Verlag.

Edelkamp, S.; and Hoffmann, J. 2004. PDDL 2.2: The Language for the Classical Part of IPC-4. In *Int. Planning Competition*.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*.

Garg, S.; Bajpai, A.; et al. 2020. Symbolic Network: Generalized Neural Policies for Relational MDPs. In *International Conference on Machine Learning*, 3397–3407.

Géron, A. 2022. *Hands-On Machine Learning With Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, Inc.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Elsevier.

Gomoluch, P.; Alrajeh, D.; Russo, A.; and Bucchiarone, A. 2017. Towards Learning Domain-Independent Planning Heuristics. *CoRR*.

Good, I. J. 1952. Rational Decisions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 14(1): 107–114.

Gosling, J.; Joy, B.; Steele, G.; and Bracha, G. 2005. Java (TM) Language Specification (The 3rd Edition).

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

Hogg, C.; Kuter, U.; and Munoz-Avila, H. 2010. Learning Methods to Generate Good Plans: Integrating HTN Learning and Reinforcement Learning. In *Proceedings of the AAAI Conference on AI*, volume 24, 1530–1535.

Höller, D.; Bercher, P.; Behnke, G.; and Biundo, S. 2019. On Guiding Search in HTN Planning with Classical Planning Heuristics. In *IJCAI*, 6171–6175.

Huang, Y. 2020. *Hierarchical Reinforcement Learning*, 317–333. Singapore: Springer Singapore.

Ioffe, S.; and Szegedy, C. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*, 448–456. Proceedings of Machine Learning Research.

Kaduri, O.; Boyarski, E.; and Stern, R. 2020. Algorithm Selection for Optimal Multi-Agent Pathfinding. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 161–165.

Kocsis, L.; and Szepesvári, C. 2006. Bandit Based Monte-Carlo Planning. In *ECML*, 282–293. Springer.

Kovacs, D. 2011. BNF Definition of PDDL3.1: Completely Corrected, Without Comments. *Unpublished Manuscript from the International Planning Competition Website*.

Lathuilière, S.; Mesejo, P.; Alameda-Pineda, X.; and Horaud, R. 2019. A Comprehensive Analysis of Deep Regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(9): 2065–2081.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11): 2278–2324.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL - The Planning Domain Definition Language. *AIPS-98*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing Atari with Deep Reinforcement Learning.

Patra, S.; Mason, J.; Kumar, A.; Ghallab, M.; Traverso, P.; and Nau, D. 2020. Integrating Acting, Planning, and Learning in Hierarchical Operational Models. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 478–487.

Rivlin, O.; Hazan, T.; and Karpas, E. 2020. Generalized Planning With Deep Reinforcement Learning. *CoRR*.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1986. Learning Representations by Back-Propagating Errors. *Nature*, 323(6088): 533–536.

Russell, S. J.; Norvig, P.; Canny, J. F.; Malik, J. M.; and Edwards, D. D. 2010. *Artificial Intelligence: A Modern Approach*. Prentice Hall Upper Saddle River, 3 edition.

Seipp, J.; Torralba, Á.; and Hoffmann, J. 2022. PDDL Generators. . Https://github.com/AI-Planning/pddl-generators, Accessed: 20.3.2023.

Shen, W.; Trevizan, F.; and Thiébaux, S. 2020. Learning Domain-Independent Planning Heuristics with Hypergraph Networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 574–584.

Shen, W.; Trevizan, F.; Toyer, S.; Thiébaux, S.; and Xie, L. 2019. Guiding Search with Generalized Policies for Probabilistic Planning. In *Proceedings of the International Symposium on Combinatorial Search*, volume 10, 97–105.

Sigurdson, D.; Bulitko, V.; Koenig, S.; Hernandez, C.; and Yeoh, W. 2019. Automatic Algorithm Selection in Multi-Agent Pathfinding. *CoRR*.

Staud, M. 2022. Urban Modeling via Hierarchical Task Network Planning. *HPlan 2022*, 73.

Toyer, S.; Thiébaux, S.; Trevizan, F.; and Xie, L. 2020. ASNets: Deep Learning for Generalised Planning. *Journal of Artificial Intelligence Research*, 68: 1–68.

Trunda, O.; and Barták, R. 2020. Deep Learning of Heuristics for Domain-independent Planning. In *ICAART*, 79–88.

Vezhnevets, A. S.; Osindero, S.; Schaul, T.; Heess, N.; Jaderberg, M.; Silver, D.; and Kavukcuoglu, K. 2017. Feudal Networks for Hierarchical Reinforcement Learning. In *International Conference on Machine Learning*, 3540–3549. Proceedings of Machine Learning Research.

Warden, P.; and Situnayake, D. 2019. *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O'Reilly Media.