Finding Reaction Mechanism Pathways with Deep Reinforcement Learning and Heuristic Search

Rojina Panta [1,2], Mohammadamin Tavakoli [3], Christian Geils [1,2], Pierre Baldi [4], Forest Agostinelli [1,2]

[1]AI Institute, University of South Carolina, Columbia, South Carolina, USA
[2] Department of Computer Science and Engineering, University of South Carolina, Columbia, South Carolina, USA
[3]Department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena, California, USA
[4]Department of Computer Science, University of California, Irvine, California, USA

Abstract

Artificial intelligence (AI) has been used to predict the outcomes of chemical reactions. However, most of these reaction predictors are designed to predict the major outcome of overall transformations, skipping the chemical reactions at a mechanistic level. Therefore, we are unable to identify intermediates and byproducts of the reaction. Information on the reaction mechanisms enable practitioners to validate the feasibility of that reaction, identify intermediate molecules, improve reaction efficiency, and anticipate the results of similar reactions under various conditions. Despite recent efforts in developing mechanistic reaction predictors, predicting the sequence of mechanistic reactions given the reactants and products is currently an open area of research in chemistry. To address this issue, we use DeepCubeA with Hindsight Experience Replay to learn a heuristic function that generalizes over start and goal states to guide A* search to predict the sequence of mechanistic reactions of an overall chemical transformation, from reactants to products.

Introduction

A chemical reaction, as the overall transformation of reactant molecules into product molecules, can be broken down into a series of smaller reaction steps called elementary steps. An elementary step reaction involves only a single transition state (Figure 1). Knowledge of the reaction mechanisms in a given chemical reaction allows practitioners to validate the feasibility of that reaction, identify intermediate molecules, improve reaction efficiency, and predict the outcome of similar reactions under different conditions. This knowledge can directly improve reaction efficiency by predicting the reaction impurity (Qiu and Norwood 2007). Other applications range from chemical synthesis and drug discovery to material design and industrial production.

However, the products of a chemical reaction given only its reactants may often be difficult to determine from first principles and expensive to perform in the real world. Therefore, practitioners often rely on chemical reaction prediction (Wei, Duvenaud, and Aspuru-Guzik 2016; Fooshee et al. 2018; Coley et al. 2017; Schwaller et al. 2019). Most chemical reaction prediction methods address a simplified version of the reaction prediction problem which is to find a single major final product given the reacting species, and do not take reaction mechanisms into account. To address this problem, we will pose finding a sequence of reaction mechanisms as a pathfinding problem where the start states are reactants and the goal states are products. We will build on the DeepCubeA (Agostinelli et al. 2019) algorithm to learn a heuristic function represented as a deep neural network (DNN) (Schmidhuber 2015) with deep reinforcement learning and use the learned heuristic function with A* search (Hart, Nilsson, and Raphael 1968) to find paths.

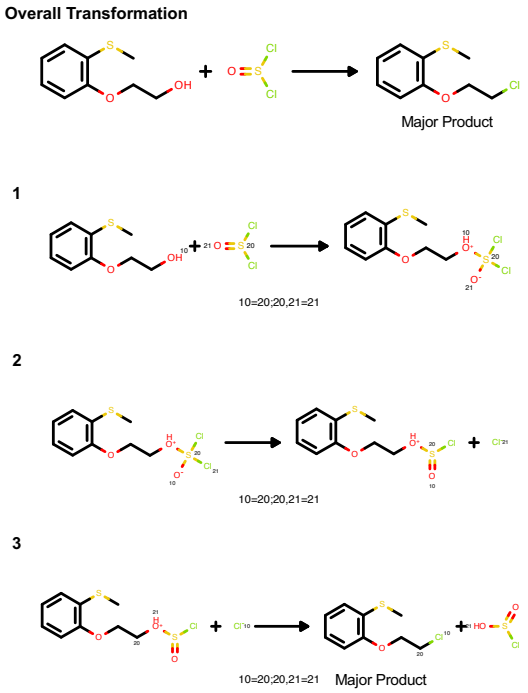

Figure 1: An overall transformation is broken down into a sequence of three elementary steps where all the intermediates are preserved.

In order to perform heuristic search, we first have to define an expansion function that computes all possible reaction mechanism steps for a chemical compound, which is still an ongoing area of research. We build upon OrbChain (Tavakoli et al. 2024a) as a standard model of elementary steps (reaction mechanisms) to accomplish this. However, this also presents a new challenge because the time it takes

to compute the expansion function for a single state can take seconds. We address this by creating an offline dataset of precomputed start states, goals, and expanded start states. Given such a dataset, we can readily build on the DeepCubeA algorithm to learn a heuristic function. To the best of our knowledge, declarative descriptions of reaction mechanisms that can be written in PDDL (McDermott 2000) or STRIPS (Fikes and Nilsson 1971) do not exist. This furthers the need for planning methods that can be used with black-box simulators (Frances et al. 2017).

While the DeepCubeA algorithm was able to learn a heuristic function to solve puzzles, such as the Rubik's cube, it assumed a predefined goal corresponding to the solved state of the puzzle. On the other hand, we would like to train a heuristic function that generalizes over any pair of reactants and products. Therefore, we take inspiration from hindsight experience replay (Andrychowicz et al. 2017) to allow DeepCubeA to learn a heuristic function that generalizes over start and goal states.

Preliminaries

Reaction Mechanism Prediction

Chemical reactions are often represented as overall transformations. This representation consists of a set of starting reactant molecules that, when added together, result in the major product molecule (i.e., the single molecule with the highest yield). The United States Patent and Trademark Office (USPTO) dataset (Lowe 2017), as the primary source of chemical reaction data for training machine learning models, uses this representation. Overall transformations contain no information about intermediates and byproducts that can potentially give rise to other interesting synthetic pathways. Therefore, a predictive model trained on such representation is not able to provide any information and explanation on the intermediate materials and pathways.

On the other hand, the elementary step reactions as building blocks of overall transformations represent a balanced reaction step that contains all product molecules with arrow codes that indicate the interaction of molecular orbitals (MOs). An overall transformation can be broken down into a sequence of these elementary steps, each involving a single transition state. Recently developed datasets of such reaction data (Tavakoli et al. 2023, 2024b) enable the development of predictive models capable of providing reaction pathways with all possible intermediates and byproducts with explainability on the MO interactions (Tavakoli et al. 2024a). However, a major bottleneck of utilizing these elementary step reaction predictors is the expansion of a search tree to identify the most promising reaction pathways. Due to the large branching factor, finding reaction mechanism pathways quickly becomes infeasible with brute-force search. Therefore, developing a more intelligent search method with methods such as DeepCubeA can accelerate the pathway predictions and broaden the search coverage.

Reaction Representations There are several methods to represent a chemical reaction (overall or elementary step) as the input to machine learning models. Schneider et al. (2015) represents a chemical reaction as the weighted sum of the molecular fingerprints of molecules involved in the reaction. Fingerprints, such as circular fingerprints (Rogers and Hahn 2010), are binary or count vectors where each bit indicates the presence or absence of certain functional groups. Fooshee et al. (2018) and Tavakoli et al. (2024a) developed reaction fingerprints as continuous vectors of predefined molecular features and reactive sites within a reaction. Other representations use different modalities of chemical reaction (e.g., text representation). DRFP (Probst, Schwaller, and Reymond 2022) and *rxnfp* (Schwaller et al. 2021) are the most successful representations based on the pre-trained language models on text representations of reactions. In this work, we deploy the simplest method to represent reactions, the circular fingerprints (Rogers and Hahn 2010) of all molecules involved in the reaction. The utilization of other representations is left to future work.

DeepCubeA

DeepCubeA (Agostinelli et al. 2019) is an algorithm to learn domain-specific heuristic functions in a largely domain-independent fashion for use in A* search (Hart, Nilsson, and Raphael 1968). The heuristic function is represented as a parameterized DNN and is trained using approximate value iteration (Puterman and Shin 1978). A batched and weighted version of A* search is then used to find paths.

Approximate Value Iteration Value Iteration (Puterman and Shin 1978) is a dynamic programming (Bellman 1966) algorithm central to reinforcement learning (Sutton and Barto 2018) which is used to find the cost-to-go of every state in the state space. Value iteration updates the value of each state using the Bellman optimality equation as an update rule until convergence. In the context of pathfinding in a graph, which can also be viewed as an undiscounted, deterministic, Markov decision process (Puterman 2014), value iteration update has the following form:

$$V'(s) = \min_{a \in \mathcal{A}} (c^a(s) + V(T(s,a))) \tag{1}$$

Here, V is a table that maps states to their cost-to-go estimates, $\mathcal{A}$ is the set of all possible actions, $c^a(s)$ is the transition cost when taking action a in state s, T is the transition function that returns the state that results from taking action a in state s, and V' is a table representing the updated cost-to-go.

Although convergence is guaranteed in the tabular setting, tabular value iteration is not feasible for domains with large state spaces, such as domains involving molecules. Therefore, we use approximate value iteration, which uses a parameterized function to approximate the value iteration update (Bertsekas and Tsitsiklis 1996). In this case, the approximation architecture is a DNN with parameters, θ, that represents the value function, v_θ. It is trained to minimize the following loss function:

$$L(\theta) = \left(\min_a (c^a(s) + v_{\theta^-}(T(s,a))) - v_\theta(s) \right)^2 \tag{2}$$

Where θ^- is the parameters of target DNN. The parameters of the target DNN are updated periodically to θ during training. This has been shown to make training more stable in the

presence of a non-stationary target (Mnih et al. 2015). If s is a goal state, then the target is set to be zero. Approximate value iteration with a DNN is referred to as deep approximate value iteration (DAVI).

A* Search A* search (Hart, Nilsson, and Raphael 1968) is a widely used pathfinding algorithm in computer science that finds the shortest path between nodes in a graph. It maintains a priority queue of nodes whose priorities are determined by the sum of path cost $g(n)$, and heuristic value $h(n)$. Where $g(n)$ is the cost to get from the start node to the current node, and $h(n)$ is the heuristic value that estimates the cost to get from the current node to the goal node which can be represented as

$$f(n) = g(n) + h(n) \tag{3}$$

Here, each node carries the information of path cost $g(n)$, heuristic estimate $h(n)$, a reference to parent node, action the parent took to generate current node and depth of node. Where parent node reference (parent pointer) refers to the node from which the current node is generated and depth of node indicates number of steps taken from start node to reach the current node. Hence, the start node can be viewed as the start state, and the goal node can be viewed as the goal state. The heuristic function we use to determine $h(n)$ is obtained using DAVI.

We start by initializing the OPEN queue and adding a start node. Then, we select the next node that has minimum $f(n)$ from OPEN. We then expand the selected node by generating its neighboring nodes. We also maintain another queue CLOSED in which we add this currently expanded node if it is not previously present in CLOSED. We calculate $f(n)$, $g(n)$, and $h(n)$ of each neighboring node. The neighboring nodes are added to OPEN if not already in CLOSED. We also update the value of nodes in CLOSED if any of these neighboring child nodes have lesser $g(n)$ for the same nodes it had in CLOSED previously, indicating a shorter path from start node to current node. In addition to updating the nodes in CLOSED, we also add this node with a cheaper path cost to OPEN. The algorithm continues until the OPEN queue becomes empty or the goal node is reached. If a goal node is reached, we follow the path of the parent pointer from this goal node to the starting node to reconstruct the sequence of actions taken to reach a goal.

The above process can be modified to be faster by using a batched version of A* search, in which we calculate the heuristic of multiple nodes in OPEN at once. We expand the best N nodes at each iteration, allowing us to calculate the values of more nodes per second.

Hindsight Experience Replay

Hindsight Experience Replay (HER) (Andrychowicz et al. 2017) leverages universal value function approximators (Schaul et al. 2015) to train a value function that generalizes over start states and goal states. They accomplish this by denoting terminal states along paths that have failed to reach a given goal state as the goal state in "hindsight". They then apply the reinforcement learning update with the given start state and the goal state obtained in hindsight. This helps overcome problems that arise in reinforcement learning when rewards are either sparse or dense, but uniform.

Approach

State Representation

We use a variant of molecular fingerprints called Extended-connectivity Fingerprints (ECFPs) (Rogers and Hahn 2010), which are circular fingerprints derived using the Morgan algorithm (Morgan 1965) to represent states to the heuristic function. Circular fingerprints are bit-vectors in which substructures of a target molecule are encoded as 1. Substructures are generated based on a chosen radius, which determines the maximum number of bonds from each atom in a given molecule for which a substructure is generated. For example, for a radius of 1, that atom and its immediate neighbors are encoded for each atom, which can be shown as a result of Iteration 1 in Figure 2. As a result, fingerprints produced with greater radii encode larger substructures or more 'global' information rather than 'local'. Fingerprints can theoretically be generated for arbitrary lengths, with most implementations ranging from 512 and 16K (Martin 2021). Fingerprints of greater length encode increasingly sparse information. For example, the embedding corresponding to benzaldehyde has only 14 'on' bits at a radius of 2 (Figure 2).

To understand better how the fingerprint generation process works, we outline the process step by step.

Initial Stage In this stage, each atom in a molecule is assigned an integer identifier generated by hashing properties used in daylight atomic invariant rule (Weininger, Weininger, and Weininger 1989), which we show as the initial identifier assignment in Figure 2. This stage is also referred to as iteration 0. If we generate a fingerprint at this stage, the resulting embedding will represent only the presence or absence of individual atoms in the molecule.

Iterative Update Stage In this stage, the atom identifiers generated in the initial stage are updated to use the information from neighboring atoms for a predefined number of iterations. So, in iteration 1, each atom uses information about the number of bonds with its immediate neighbor and information about its neighboring atom identifiers generated in iteration 0 to obtain a new hash value. This process repeats for a fixed number of iterations, which corresponds to the radius selected by the practitioner. For Example, Atom 3 has a single bond with atoms 4 and 6 and a double bond with atom 2. The value to be hashed at iteration 1 will be (1, 3218693969), where 1 is the number of iterations, and 3218693969 is the identifier of atom 3 in iteration 0. Additionally, we use the information of neighboring atoms for hashing where the values to be hashed are (1, 3218693969), (1, 2246703798), (2, 3218693969), where 1, 1, 2 is the number of bonds with atom 4, 6 and 2 and 3218693969, 2246703798, 3218693969 is their identifier number in iteration 0. This results in the final list being sent to the hash function as [1, 3218693969, 1, 3218693969, 1, 2246703798, 2, 3218693969]. Here, iteration can be defined in terms of radius, where radius 1 refers to iteration 1, and radius 2

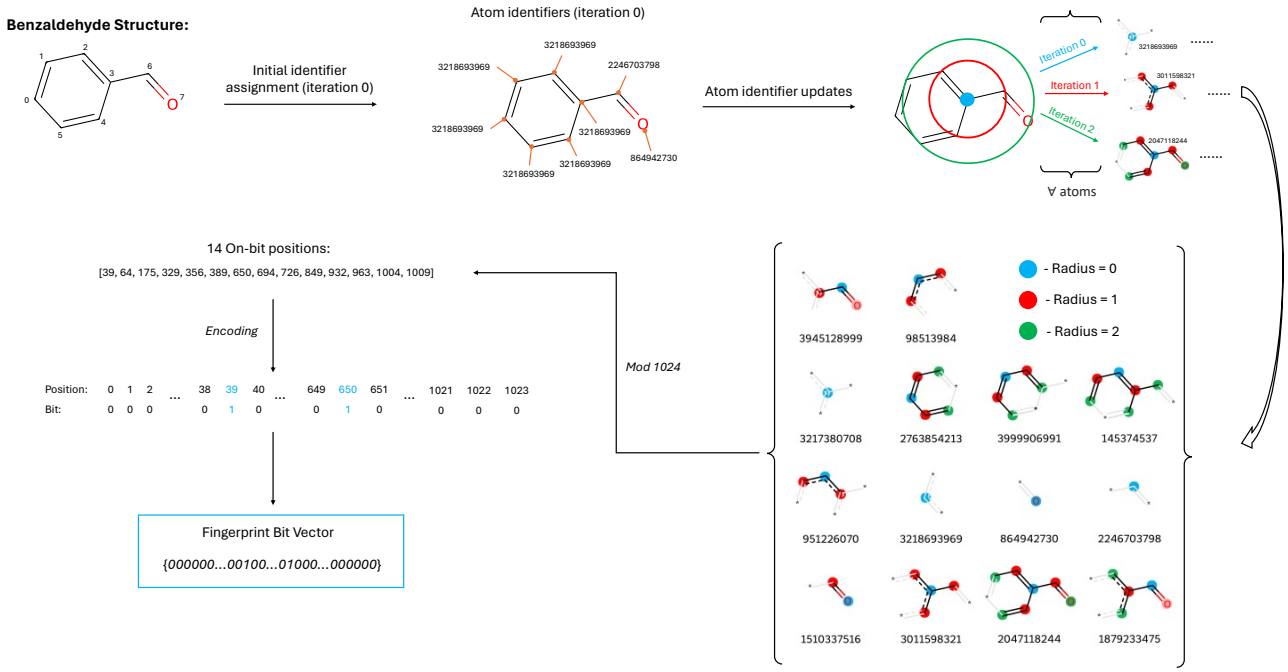

Figure 2: A diagrammatic representation of the ECFP/Circular Fingerprint generation process used to generate a molecular fingerprint embedding.

refers to iteration 2, which we show as atom identifier updates in Figure 2.

Removing Duplicate Identifier With the increase in the number of iterations, it is possible that different atoms may result in the same substructure. For Example, in Figure 2, in initial atom mapping, we can see that different atoms have the same identifier, 3218693969. So, we only keep a copy of it to generate the final fingerprint. One may also choose not to remove these duplicates in the final embedding, resulting in a 'count' fingerprint that encodes not just the presence or absence of a substructure, but also the number of times it appears in the molecule.

Once these three steps are completed, we convert these hashed identifiers to a bit array. In this process, we choose the number of bits in our fingerprint array. In Figure 2, we have chosen 1024 as the number of bits in a fingerprint. Then we initialize an empty array with 0 having length 1024, perform the modulo operation with 1024, and use its remainder to set the value in a bit array where the value of the remainder is equal to a corresponding number of indices in an array, resulting in our fingerprint vector for the molecule.

Because each fingerprint attempts to encode every possible substructure for a given radius in the target molecule, a lower fingerprint length incurs a risk of "collision" also called bit-collision, whereby multiple substructures are encoded identically. Though uncommon due to the sparse nature of ECFPs, lower fingerprint lengths may result in different molecules having identical embedding. For Example, In Figure 2, if we have a fingerprint array of length 2, all the identifiers with even numbers will be indexed to position 0 even though they are structurally different.

Lastly, it should be noted that the hashing mechanism used in the ECFP is irreversible due to bit-collision, which implies that different molecules may be mapped to the same representation. By contrast, graph neural networks (Tavakoli et al. 2022a,b) can be used to learn a representation that can better differentiate molecules.

State Expansion

To perform value iteration, we need to obtain all possible next states of a given state, s. We can obtain these states by using the transition function T, which returns the state that results from taking action a in state s. These next states are required to train DAVI to learn the cost-to-go.

In the context of mechanistic reaction prediction, each state represents a set of molecules. Consequently, we expand one state by considering all possible mechanistic reactions from the molecules of the current state. Therefore, the next states represent intermediates or transitional configurations of molecules that arise during the successive mechanistic steps of the reaction. To expand a state, we used OrbChain, the standard model of mechanistic reactions introduced by Tavakoli et al. (2024a), which builds on OpenEye Scientific. We modify OrbChain to process polar reactions instead of radical reactions. This modification includes changing the half-occupied MOs to fully-occupied and empty MOs. An

example of state expansion is shown in Figure 3.

Dataset

The dataset used for training comes from the United States Patent and Trademark Office. We select 10 reactant and product pairs that do not use any reagent and use them to generate about 2 million reactant and product pairs. We chose these 10 pairs based on the time needed to expand and proceed to the next step. We are using the top 10 reactants that require minimal expansion time, which ranges from 0.81 to 4.25 seconds.

Training

Once we select the initial reactants we want to work with, we use HER to generate more data to train our reinforcement learning algorithm. To achieve this, we start at a given state from USPTO data and take a random walk to generate a start state. We then randomly walk 0 to 6 steps from this start state to generate a goal state. Next, we pass this updated (state, goal) pair to train our heuristic function using DAVI. Since our heuristic function now takes (state, goal) pair as input in contrast to the state as input, mentioned in Equation 2, we will modify the equation as

$$L(\theta) = \left(\min_a(c^a(s) + v_{\theta^-}(T(s,a), s_g)) - v_\theta(s, s_g)\right)^2 \tag{4}$$

Where, s_g is the goal state generated with random walk.

From a total of 2 million datasets we generate, we use 300K datasets per epoch to train our neural network. We also keep the record of number of states solved so far during training using a greedy policy and update our target network θ^- in Equation 4 when the percentage of states solved so far is greater than the percentage of steps solved previously. The greedy policy can be represented using the equation

$$\pi(s) = argmin_a(c^a(s) + v_\theta(T(s,a), s_g)) \tag{5}$$

This policy at each state s selects the action a that minimizes the sum of the transition cost and the approximated value of the next state, as given by the value function v_θ.

Testing

We generate the test dataset by performing random walks from the USPTO data, similar to how we generate a start state and goal state for training. Then, we use the heuristics provided by the trained neural network using DAVI along with the A* search algorithm to find a path from start states to goal states. We also record the number of states solved and their path cost from start states to goal states. Additionally, we represent states during search using canonical SMILES (Weininger 1988; Weininger, Weininger, and Weininger 1989) and convert this representation to molecular fingerprints while training DNN, allowing us to extract molecular pathways from a start state to a goal state if required.

Experiments

Architecture and Hyperparameter Search

To generate the data required for supervised learning used for architecture search, we start with USPTO data and then perform a random walk to generate a start state. Next, we use the breadth-first search algorithm along with our state expansion function to reach our goal state from this start state. The depth of breadth first search we use varies from 0 to 2. Given the expanded states' large branching factor, we can only generate the states up to depth two. To ensure the state we are using is exactly one step away from the goal, we check that the state in depth zero is not appearing again in depth one.

Similarly, to ensure it is exactly two steps away from the goal, we check that none of the states in depth zero or one is appearing again in depth two. We generate 1 million data in total for our supervised learning that contains (start state, goal state, path cost) triplet. Here, path cost is equivalent to a depth of breadth-first search. Our supervised learning algorithm takes the concatenation of the start state and goal state as input and path cost as ground truth. We are concerned with how our training curve changes over each epoch for our architecture search.

To identify an optimal architecture, we conduct experiments with different fingerprint lengths and hidden layer sizes in a supervised learning setting. Our base architecture takes a concatenation of start and goal state fingerprints as input to a fully connected linear model along with four residual blocks (He et al. 2016). The results are shown in Figure 4. The figure shows that the architecture with a fingerprint length of 4096 and a hidden dimension size of 10,000 performs best, so we select this configuration for our reinforcement learning setting.

Generating Offline Dataset

Since the expand function is slow (i.e., expansion of 1 state takes around 0.8 to 4 seconds), it takes a long time to expand 300K states and then train our heuristic function. To overcome this challenge, we save the result of the expand function in a file and retrieve it whenever we need to expand our state during training. It takes about 4-5 days to generate 2 million datasets with parallel processing that utilizes 48 CPUs. We use these offline expanded states for training only. We expand states on the fly for testing.

Finding paths to the goal

We generate 15 start and goal state pairs using a random walk for steps ranging from 0 to 6 steps (105 in total). Next, we pass these state and goal pairs to our A* search with DeepCubeA heuristics, A* search with Tanimoto coefficient for similarity between molecules (Flower 1998; Godden, Xue, and Bajorath 2000) as heuristics, and uniform cost search algorithms. Since the Tanimoto coefficient is used as a similarity measure, it is also called Tanimoto similarity. Here, we subtract the result of Tanimoto similarity from 1 to ensure the heuristic is zero when we reach goal states. To ensure that our search algorithm terminates and has an equal time limit for all states, we set a time limit of 1 hour

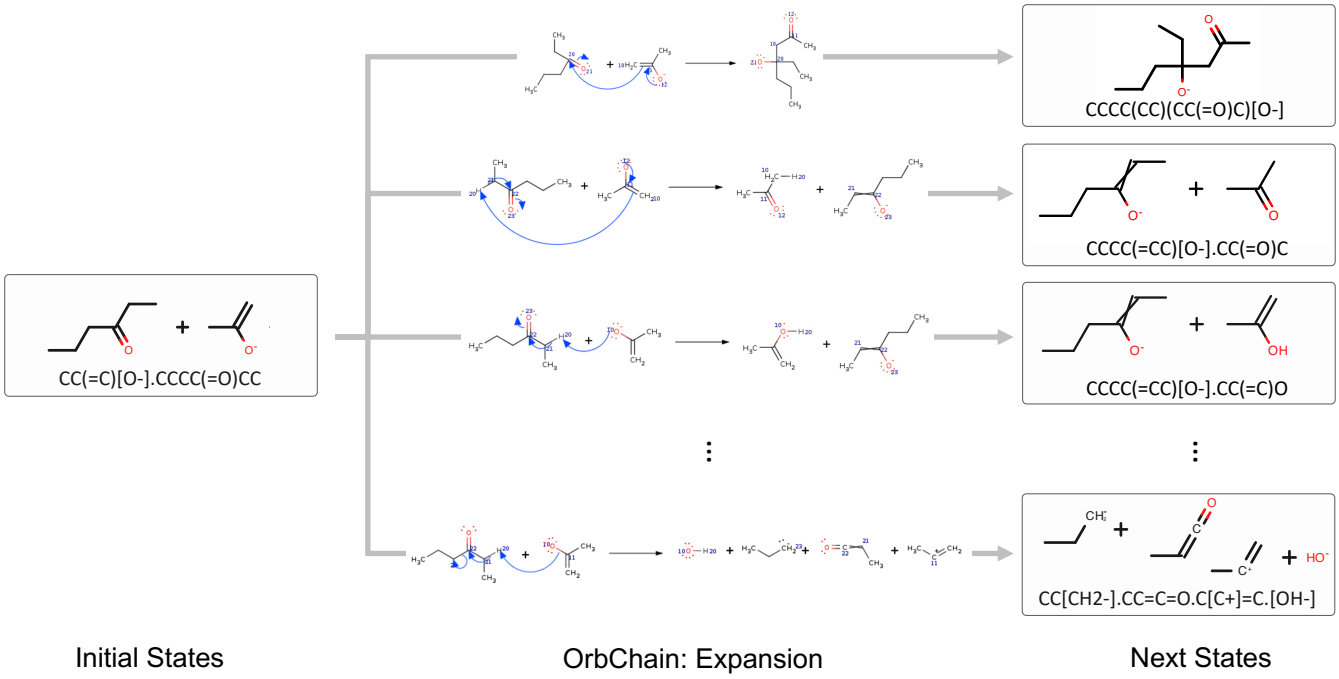

Figure 3: An example of state expansion. We used OrbChain to consider all possible mechanistic reactions from the reacting molecules (initial state). This generates all possible next states. Considering all possible mechanistic reactions results in some implausible next states.

to search each state. Here, we do not make use of batch A* search. Therefore, we use a batch size of 1 and a weight of 1. The search results for A* search with DeepCubeA heuristics, uniform cost search, and A* search with Tanimoto similarity heuristics are shown in Table 1.

Table 1 shows that DeepCubeA, uniform cost search, and Tanimoto similarity can solve all the states for steps 0 and 1. However, there is a significant difference in number of nodes generated and time taken to solve the states for step 1. In the case of steps 2 and 4, DeepCubeA can solve more states than uniform cost search and Tanimoto similarity. For steps 3, 5, and 6, uniform cost search and Tanimoto similarity cannot solve any state within the time limit, whereas DeepCubeA solves 86.67% , 33.33% , and 33.33% of states, respectively.

Discussion

The results in Table 1 show that the learned heuristic significantly outperforms uniform cost search and Tanimoto similarity in terms of both the percentage of instances solved and the time taken. This shows that reinforcement learning can be used to obtain a domain-specific heuristic function without the need for domain-specific heuristic knowledge. More broadly, it may suggest that deep reinforcement learning can be applied to solve other pathfinding problems in chemistry, such as chemical synthesis and search for chemical reactions that could lead to toxic chemicals. Examples of these would be the automation of the screening phase of drug design via drug degradation and mass spectrometry (Wu 2000). However, Table 1 also shows there is significant room for improvement. We discuss this further in the Future Work Section.

The nodes per second metric shown in Table 1 is only on the order of 10^1, which is due to the computationally expensive expansion function. Even when given an accurate heuristic function, such a slow expansion function could make finding paths difficult for goals that are anything more than a few steps away. While batch A* search could be used to parallelize the expansion function, this speed-up is, at most, linear in the number of CPUs used. We discuss methods to handle this in the Future Work Section.

Related Work

Tavakoli et al. (2024a) predict reaction mechanisms pathways by first training a DNN to predict the plausibility of single reaction mechanisms steps. This is then used with a beam search, where the top-K most plausible reaction mechanism steps are kept at each step of the search. However, unlike A* search, beam search is not a complete search algorithm. Furthermore, beam search does not take into account the entire path cost, which will be important for predicting entire pathways that are plausible. In future work, this plausibility estimator could be combined with our approach to compute transition costs. This could allow our method to then learn to find the most plausible pathways.

On a higher level, machine learning and heuristic search have been used to predict chemical reactions. Chen et al. (2020) performs retrosynthesis (Corey 1988, 1991) with a DNN and And-Or search. The DNN is trained to predict the

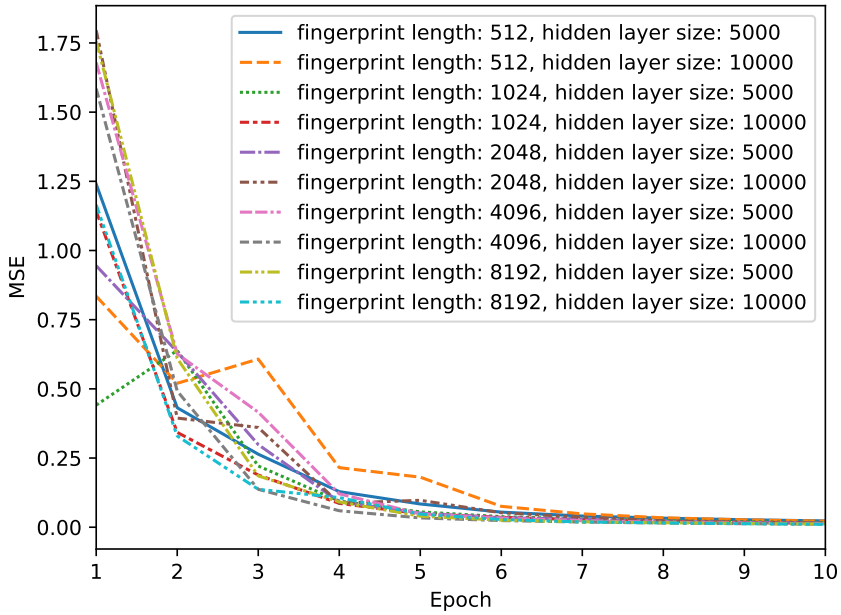

Figure 4: Training loss for fully connected residual neural network hyperparameter search, with variable hidden dimension size and fingerprint lengths. Loss is calculated as mean squared error (MSE), and all models are trained for 10 epochs.

Step/s	Solver	Path Cost	% Solved	Nodes	Secs	Nodes/Sec
Steps=0	DeepCubeA	0.00	**100.00%**	3.09E+2	3.87	79.97
	Uniform Cost Search	0.00	**100.00%**	3.09E+2	4.61	67.13
	Tanimoto Similarity	0.00	**100.00%**	3.09E+2	3.71	83.42
Steps=1	DeepCubeA	1.00	**100.00%**	7.49E+2	9.70	77.26
	Uniform Cost Search	1.00	**100.00%**	4.26E+4	553.33	76.95
	Tanimoto Similarity	1.00	**100.00%**	3.13E+4	429.29	72.97
Steps=2	DeepCubeA	2.07	**100.00%**	1.63E+4	267.16	60.87
	Uniform Cost Search	1.67	20.00%	1.32E+5	1497.77	87.96
	Tanimoto Similarity	1.75	26.67%	1.10E+5	1229.10	89.13
Steps=3	DeepCubeA	2.77	86.67%	4.14E+4	578.88	71.54
	Uniform Cost Search	-	0.00%	-	-	-
	Tanimoto Similarity	-	0.00%	-	-	-
Steps=4	DeepCubeA	3.33	60.00%	6.36E+4	821.64	77.36
	Uniform Cost Search	3.00	6.67%	1.43E+5	1962.28	73.01
	Tanimoto Similarity	3.00	6.67%	2.47E+4	272.15	90.64
Steps=5	DeepCubeA	3.40	33.33%	8.40E+4	968.49	86.69
	Uniform Cost Search	-	0.00%	-	-	-
	Tanimoto Similarity	-	0.00%	-	-	-
Steps=6	DeepCubeA	3.20	33.33%	6.14E+4	933.86	65.73
	Uniform Cost Search	-	0.00%	-	-	-
	Tanimoto Similarity	-	0.00%	-	-	-

Table 1: Comparative analysis of DeepCubeA with Uniform cost search and domain-dependent but not learned Tanimoto similarity across dimensions including solution length, percentage of instances solved, number of nodes generated, time taken to solve the problem (in seconds), and number of nodes generated per second. We report the result of search with start states 0 to 6 steps away from the goal.

promise of retrosynthetic steps using historical data. However, this method does not consider reaction mechanisms. Furthermore, this method could possibly be improved by using reinforcement learning instead of learning from historical data.

Future Work

Representation of Molecules

The result of training depends on the state representation provided to the neural networks. In this work, we use molecular fingerprints, which are static representations predefined based on molecular features that are extracted as substructures generated about each atom. We can improve the state representation by using a learnable graph neural network (Tavakoli et al. 2022a,b) that can learn from the graph structure of molecules, which closely matches how molecules are structurally conceptualized in chemistry (atoms as nodes, bonds as edges).

Handling Slow Expansion Times

The time it takes to do a single expansion significantly impacted our ability to obtain training data as well as to do A* search efficiently. Future work could instead learn an action-value function in the form of a deep Q-network (DQN) (Mnih et al. 2015) that predicts the cost-to-go when taking a given action in a given state, where actions would be determined by molecular orbital interactions. As a result, Q-learning (Watkins and Dayan 1992) could be used to train a DQN and, Q* search (Agostinelli et al. 2021) could be used to find paths, neither of which require that a state be fully expanded.

Generalizing Across Sets of Goal States

The USPTO dataset we use has a single product, also called the major product. However, this major product does not contain any by-products of the chemical reaction. Therefore, product represents, not a single state, but a set of goal states. Therefore, we need to train a heuristic function that generalizes over states and sets of goal states. One potential approach could be to combine DeepCubeA with answer set programming (Brewka, Eiter, and Truszczyński 2011) to specify goals, as was done by Agostinelli, Panta, and Khandelwal (2024).

Predicting Feasibility of Reaction Mechanism

The transition costs we used in our experiments were one for every action. However, in order to find the most feasible real-world reaction mechanism path, we will need to take the feasibility of each reaction mechanism step into account. If transitions are independent, the most feasible path would be the product of the probabilities of each transition. We could then modify our transition cost such that the transition cost is the negative log probability of that transition (Tavakoli et al. 2022a). Therefore, minimizing the path cost will be equivalent to maximizing the product of the probabilities of transition costs.

Conclusion

Knowledge of the reaction mechanism pathway in a given chemical reaction is important for validation, improving the efficiency of a reaction, and predicting the outcome of a reaction in new environments. We build on DeepCubeA and HER to train a domain-specific heuristic function to predict the cost-to-go between two sets of molecules without domain-specific heuristic knowledge. We can then use this heuristic function with A* search to find a reaction mechanism pathway between chemical reactants and products. We also compared the results with uniform cost search and Tanimoto similarity and found that the learned heuristics perform better than uniform cost search and Tanimoto similarity in terms of percentage solved and time. Future work leveraging better representations of molecules and search algorithms that can handle slow expansion times could lead to significant improvements in performance.

References

Agostinelli, F.; McAleer, S.; Shmakov, A.; and Baldi, P. 2019. Solving the Rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1(8): 356–363.

Agostinelli, F.; Panta, R.; and Khandelwal, V. 2024. Specifying Goals to Deep Neural Networks with Answer Set Programming. In *34th International Conference on Automated Planning and Scheduling*.

Agostinelli, F.; Shmakov, A.; McAleer, S.; Fox, R.; and Baldi, P. 2021. A* search without expansions: Learning heuristic functions with deep q-networks. *arXiv preprint arXiv:2102.04518*.

Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Pieter Abbeel, O.; and Zaremba, W. 2017. Hindsight experience replay. *Advances in neural information processing systems*, 30.

Bellman, R. 1966. Dynamic programming. *science*, 153(3731): 34–37.

Bertsekas, D.; and Tsitsiklis, J. N. 1996. *Neuro-dynamic programming*. Athena Scientific.

Brewka, G.; Eiter, T.; and Truszczyński, M. 2011. Answer set programming at a glance. *Communications of the ACM*, 54(12): 92–103.

Chen, B.; Li, C.; Dai, H.; and Song, L. 2020. Retro*: learning retrosynthetic planning with neural guided A* search. In *International conference on machine learning*, 1608–1616. PMLR.

Coley, C. W.; Barzilay, R.; Jaakkola, T. S.; Green, W. H.; and Jensen, K. F. 2017. Prediction of organic reaction outcomes using machine learning. *ACS central science*, 3(5): 434–443.

Corey, E. 1988. Robert Robinson lecture. Retrosynthetic thinking—essentials and examples. *Chemical society reviews*, 17: 111–133.

Corey, E. J. 1991. *The logic of chemical synthesis*.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4): 189–208.

Flower, D. R. 1998. On the properties of bit string-based measures of chemical similarity. *Journal of chemical information and computer sciences*, 38(3): 379–386.

Fooshee, D.; Mood, A.; Gutman, E.; Tavakoli, M.; Urban, G.; Liu, F.; Huynh, N.; Van Vranken, D.; and Baldi, P. 2018. Deep learning for chemical reaction prediction. *Molecular Systems Design & Engineering*, 3(3): 442–452.

Frances, G.; Ramírez Jávega, M.; Lipovetzky, N.; and Geffner, H. 2017. Purely declarative action descriptions are overrated: Classical planning with simulators. In *IJCAI 2017. Twenty-Sixth International Joint Conference on Artificial Intelligence; 2017 Aug 19-25; Melbourne, Australia.[California]: IJCAI; 2017. p. 4294-301*. International Joint Conferences on Artificial Intelligence Organization (IJCAI).

Godden, J. W.; Xue, L.; and Bajorath, J. 2000. Combinatorial preferences affect molecular similarity/diversity calculations using binary fingerprints and Tanimoto coefficients. *Journal of Chemical Information and Computer Sciences*, 40(1): 163–166.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2): 100–107.

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.

Lowe, D. 2017. Chemical reactions from US patents (1976-Sep2016).

Martin, L. J. 2021. State of the Art Iterative Docking with Logistic Regression and Morgan Fingerprints. *ChemRxiv*.

McDermott, D. M. 2000. The 1998 AI planning systems competition. *AI magazine*, 21(2): 35–35.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533.

Morgan, H. L. 1965. The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service. *Journal of chemical documentation*, 5(2): 107–113.

Probst, D.; Schwaller, P.; and Reymond, J.-L. 2022. Reaction classification and yield prediction using the differential reaction fingerprint DRFP. *Digital discovery*, 1(2): 91–97.

Puterman, M. L. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Puterman, M. L.; and Shin, M. C. 1978. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, 24(11): 1127–1137.

Qiu, F.; and Norwood, D. L. 2007. Identification of pharmaceutical impurities. *Journal of liquid chromatography & related technologies*, 30(5-7): 877–935.

Rogers, D.; and Hahn, M. 2010. Extended-connectivity fingerprints. *Journal of chemical information and modeling*, 50(5): 742–754.

Schaul, T.; Horgan, D.; Gregor, K.; and Silver, D. 2015. Universal value function approximators. In *International conference on machine learning*, 1312–1320. PMLR.

Schmidhuber, J. 2015. Deep learning in neural networks: An overview. *Neural networks*, 61: 85–117.

Schneider, N.; Lowe, D. M.; Sayle, R. A.; and Landrum, G. A. 2015. Development of a novel fingerprint for chemical reactions and its application to large-scale reaction classification and similarity. *Journal of chemical information and modeling*, 55(1): 39–53.

Schwaller, P.; Laino, T.; Gaudin, T.; Bolgar, P.; Hunter, C. A.; Bekas, C.; and Lee, A. A. 2019. Molecular transformer: A model for uncertainty-calibrated chemical reaction prediction. *ACS central science*, 5(9): 1572–1583.

Schwaller, P.; Probst, D.; Vaucher, A. C.; Nair, V. H.; Kreutter, D.; Laino, T.; and Reymond, J.-L. 2021. Mapping the space of chemical reactions using attention-based neural networks. *Nature machine intelligence*, 3(2): 144–152.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Tavakoli, M.; Baldi, P.; Carlton, A. M.; Chiu, Y. T.; Shmakov, A.; and Van Vranken, D. 2024a. AI for Interpretable Chemistry: Predicting Radical Mechanistic Pathways via Contrastive Learning. *Advances in Neural Information Processing Systems*, 36.

Tavakoli, M.; Chiu, Y. T. T.; Baldi, P.; Carlton, A. M.; and Van Vranken, D. 2023. Rmechdb: A public database of elementary radical reaction steps. *Journal of chemical information and modeling*, 63(4): 1114–1123.

Tavakoli, M.; Miller, R.; Angel, M.; Pfeiffer, M.; Gutman, E.; Mood, A.; Van Vranken, D.; and Baldi, P. 2024b. PMechDB: A Public Database of Elementary Polar Reaction Steps. *Journal of Chemical Information and Modeling*, 64(6): 1975–1983.

Tavakoli, M.; Mood, A.; Van Vranken, D.; and Baldi, P. 2022a. Quantum mechanics and machine learning synergies: graph attention neural networks to predict chemical reactivity. *Journal of Chemical Information and Modeling*, 62(9): 2121–2132.

Tavakoli, M.; Shmakov, A.; Ceccarelli, F.; and Baldi, P. 2022b. Rxn hypergraph: a hypergraph attention model for chemical reaction representation. *arXiv preprint arXiv:2201.01196*.

Watkins, C. J.; and Dayan, P. 1992. Q-learning. *Machine learning*, 8: 279–292.

Wei, J. N.; Duvenaud, D.; and Aspuru-Guzik, A. 2016. Neural networks for the prediction of organic chemistry reactions. *ACS central science*, 2(10): 725–732.

Weininger, D. 1988. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1): 31–36.

Weininger, D.; Weininger, A.; and Weininger, J. L. 1989. SMILES. 2. Algorithm for generation of unique SMILES notation. *Journal of chemical information and computer sciences*, 29(2): 97–101.

Wu, Y. 2000. The use of liquid chromatography–mass spectrometry for the identification of drug degradation products in pharmaceutical formulations. *Biomedical Chromatography*, 14(6): 384–396.