Completeness and Diversity in Depth-First Proof-Number Search with Applications to Retrosynthesis

Christopher Franz\textsuperscript{1*}, Georg Mogk\textsuperscript{2}, Thomas Mrziglod\textsuperscript{2} and Kevin Schewior\textsuperscript{3}
\textsuperscript{1}Frankfurt, Germany
\textsuperscript{2}Bayer AG, Leverkusen, Germany
\textsuperscript{3}University of Southern Denmark, Odense, Denmark
christopher@kotaico.de, \{georg.mogk,thomas.mrizglo\}@bayer.com, kevs@sdu.dk

Abstract

We revisit Depth-First Proof-Number Search (DFPN), a well-known algorithm for solving two-player games. First, we consider the completeness property of the algorithm and its variants, i.e., whether they always find a winning strategy when there exists one. While it is known that the standard version is not complete, we show that the combination with the simple Threshold Controlling Algorithm is complete, solving an open problem from the area. Second, we modify DFPN to compute a diverse set of solutions rather than just a single one. Finally, we apply this new variant in Chemistry to the synthesis planning of new target molecules (Retrosynthesis). In this domain a diverse set of many solutions is desirable. We apply additional modifications from the literature to the algorithm and show that it outperforms Monte-Carlo Tree-Search, another well-known algorithm for the same problem, according to a natural diversity measure.

1 Introduction

A challenging problem in AI is solving two-player games: Formally, one is given a directed graph, which in our case is finite. The game takes place by moving a token, initially on a given start node, between the nodes of the graph. At all times, depending on which node the token is on, either Player 0 or Player 1 may move the token along one of the outgoing edges to another node. Upon any such move, depending on the rules of the game, either of the players may win. The goal is to efficiently compute a winning strategy for either of the players or to decide that no such strategy exists.

The applications of this problem reach far beyond actual games such as Chess and Go [Pearl, 1984]. In particular, in this paper we consider applications to chemical synthesis planning [Heifets and Jurisica, 2012; Segler et al., 2018; Kishimoto et al., 2019]. Indeed, the problem of finding a plan (a route) to synthesize a given target molecule from commercially available molecules can be modeled such that a winning strategy for Player 0 corresponds to a synthesis route for the target molecule.

\textsuperscript{*}Contact Author

Without loss of generality, consider the problem of finding a winning strategy for Player 0 or deciding that no such strategy exists. A well-known algorithm for doing so when the input graph is a tree is the Proof-Number Search (PNS) [Al-lis et al., 1994]. This algorithm explores the tree starting from the start node along edges. In that context, one speaks of a node \( v \) being proved (disproved) when the algorithm has explored a set of nodes that certify that there is (not) a winning strategy for Player 0. To eventually prove or disprove the start node, each node of the tree is equipped with both a proof number and a disproof number. They are lower bounds on the number of nodes that have to be proved or disproved, respectively, to prove or disprove the corresponding node. These two numbers steer the order in which PNS explores nodes. A popular [Kishimoto and Müller, 2004a; Schaeffer et al., 2005] variant of this algorithm is the Depth-First Proof-Number Search (DFPN) [Nagai, 2002], which is more efficient and complete with respect to both time and memory.

In practice, however, the input graph is often not a tree: For instance, in Chess, positions may be repeated. Also in Chemistry, a single molecule may be part of two different routes for synthesizing the same molecule, but as part of different reactions. On the emerging general graphs, several issues arise. Whether a player has won, may not just depend on the current position of the game, but also on the history of the game. This leads to the Graph History Interaction Problem [Campbell, 1985], which can be handled [Kishimoto and Müller, 2004b; Kishimoto, 2005] but, if unhandled, causes DFPN to provide incorrect outputs. Further, DFPN/PNS over- or underestimate the actual proof and disproof numbers due to double counting, an issue that can be mitigated, e.g., [Kishimoto, 2010], but not efficiently solved [Gao, 2021].

Arguably, the most severe issue about DFPN on general graphs is that it is not complete any more [Kishimoto and Müller, 2008], i.e., it is possible that there exists a winning strategy, but DFPN runs into an infinite loop rather than outputting the strategy. As proved in the same paper however, DFPN is complete on directed acyclic graphs. Refinements of DFPN that seem to be complete in experiments are dfpn(r) [Kishimoto, 2005] and DFPN with the Threshold Controlling Algorithm (TCA) [Kishimoto, 2010], but no proofs of completeness for such variants are known.

A second and more practical concern with the state-of-the-art DFPN is that it outputs an arbitrary winning strategy. In-
indeed, in practice, winning strategies may differ with respect to their quality, and a preferred output would be a quality-maximizing winning strategy. In our specific application and arguably other applications, the quality of a synthesis route is, however, not well-defined and a matter of debate. Indeed, it is not clear how to choose the trade-off between quality measures such as safety, yield or difficulty of the route. Therefore, the ideal output seems to be a “diverse” set of routes, from which the user can pick.

In this paper, we address both concerns. First, we provide a proof of the completeness of DFPN with TCA, a variant for which completeness has been posed as an open problem [Kishimoto, 2010]. Second, we provide a general variant of DFPN which outputs a set of multiple solutions. We then experimentally evaluate the diversity of the output. We compare these results with the commonly used Monte-Carlo Tree Search (MCTS) [Browne et al., 2012].

1.1 Our Contribution

The main theoretical contribution of this paper is a proof of completeness of DFPN with TCA in the general case. In other words, we show that DFPN with TCA always outputs a winning strategy when there exists one. Our proof can be viewed as an extension of the argument by Kishimoto and Müller [2008] for directed acyclic graphs. The main technical ingredient is a lemma that relates the distance of nodes from the start node, the crucial quantity used by TCA, to the length of a certain longest path from the start node to that node, the crucial quantity used in the aforementioned proof. With that lemma at hand, only relatively minor modifications to the aforementioned proofs are needed.

We also contribute to the understanding of (regular) DFPN on general graphs by providing a simplified counterexample to the completeness. In contrast to the original example by Kishimoto and Müller [2008] that consists of 17 nodes, our example only consists of 10 nodes.

To address the need for a diverse set of solutions in applications such as retrosynthesis, we propose a novel modification of PNS and its variants that generates a set of solutions. Since PNS is an (up to tie breaking) deterministic algorithm, we cannot hope for a (large) set of solutions by running it multiple times. Instead, when the algorithm finds a solution, we first declare the last node of the solution as winning for Player 1 when visited through the same path. This strictly keeps the algorithm from finding the same solution again. To find a diverse set of solutions, we also modify proof and dis-proof numbers of the nodes that are part of the found solution by adding penalties. Then we iterate until time runs out.

In the experimental part of the paper, we consider the algorithm DFPN-E [Kishimoto et al., 2019] and adapt it to output a set of solutions in the described way. Note that this algorithm is a variant of DFPN that does not address the completeness issue; we still choose it here because it has been successfully applied to retrosynthesis. We call the algorithm that emerges DFPN* and apply it to retrosynthesis as well. We show that the diversity of routes found with DFPN* is greater than it is with MCTS. For a selection of 60 molecules with different levels of difficulty, we use both MCTS and DFPN* to identify potential synthesis routes for each molecule. We compare the results with respect to number of synthesized molecules, complexity of synthesis, diversity, and number of identified synthesis routes.

Note that the term diversity is not sharply defined in this context, however, among chemists it means that different chemical ideas are observed between the individual synthetic routes. We use the total number of different intermediate molecules observed across the set of routes as a proxy for that measure.

1.2 Further Related Work

Besides PNS and variants, other classic algorithms for solving two-player games include the αβ search [Knuth and Moore, 1975], in which scores are used to estimate the advantage of the players in certain situations, and the more recent Monte-Carlo Tree Search (MCTS) [Browne et al., 2012], which introduces randomization. While the most important literature on PNS has been mentioned, we still refer to the survey [Kishimoto et al., 2012].

The term retrosynthesis was first established by E. J. Corey [1967]. Subsequently, many different approaches to computer-assisted synthesis were developed; an overview of the most important ones can be found in [Szymkuć et al., 2016]. Starting from the target molecule, the general approach is to recursively split up molecules by applying chemical reaction rules “inversely”, i.e., by replacing the product with the reactants, until only buyable molecules are left. A winning strategy in the aforementioned game can be interpreted as doing exactly that.

Indeed, search algorithms have been very successful for retrosynthesis. For instance, A*-Search [Chen et al., 2020; Badowski et al., 2019] and Monte-Carlo Tree Search [Segler et al., 2018; MLPDS, 2020] are used in various tools. In 2012, regular PNS was used for retrosynthesis [Heifets and Jurisica, 2012], whereas DFPN was first applied as part of the DFPN-E algorithm [Kishimoto et al., 2019] and later within the CompRet tool [Shibukawa et al., 2020].

Although the topic of diversity is arguably quite important for various applications, there are relatively few sources in the literature. For instance, finding a set of synthesis plans has been looked at [Fagerberg et al., 2018] but not with explicit attention to diversity. Some first approaches to create diversity specific to chemistry can be found in [Schwaller et al., 2020]. Diversity has also been considered recently from a more theoretical point of view [Baste et al., 2020].

2 Preliminaries

Basic Definitions. The input graph is a directed graph $G = (V, E)$ where $v_0 \in V$ is the start node. Further, $V$ is partitioned into $V_0$ and $V_1$, the sets of nodes of Player 0 and Player 1, also referred to as AND and OR nodes, respectively. The sets $W_0$ and $W_1$, the sets of winning nodes of Player 0 and Player 1, respectively, are disjoint subsets of $V$. Without loss of generality, we assume that

- all neighbors of nodes in $V_0$ are in $V_1$ and vice versa.
- for all $v \in V$ it holds that $\delta_{G}^{\text{out}}(v) = \emptyset$ if and only if $v \in W_0 \cup W_1$. 


A play $P$ is a sequence of $v_0, v_1, \ldots, v_k$ such that for all $i \in \{0, \ldots, k-1\}$, we have $v_{i+1} \in N_{G}^{\text{out}}(v_i)$. If $P$ is a path in $G$, i.e., it contains each node at most once, we say $P$ is winning for Player $i$ if $v_k \in \mathcal{W}_i$. If $v_0, \ldots, v_{k-1}$ is a path and $v_k \in \{v_0, \ldots, v_{k-1}\}$ (the last move closes a cycle), we assume the play is winning for either of the players, e.g., Player 0 wins or the Player $i$ with $v_{k-1} \notin \mathcal{W}_i$ wins [Kishimoto and Müller, 2004b].

A strategy for Player $i \in \{0, 1\}$ is a function $\sigma : \mathcal{W}_i \rightarrow V$ such that $\sigma(v) \in N_{G}^{\text{out}}(v)$ for all $v \in \mathcal{W}_i$. A pair of strategies for both players naturally induces a play (that stops the first time it is winning for one of the players). For $i \in \{0, 1\}$, we say that a strategy $\sigma_i$ for Player 1 is a winning strategy if $\sigma_i$ and each strategy $\sigma_{1-i}$ for Player $1 - i$ induces a play that is winning for Player $i$.

**Proof-Number Search.** Proof-Number Search (PNS) [Al-lis et al., 1994] explores $G$ starting from $v_0$, i.e., initially $v_0$ is explored, and all other nodes are not explored. No edges are explored at that time. PNS iteratively expands nodes $v$ that were previously not expanded. This way, PNS learns if $v \in \mathcal{W}_i$ for $i \in \{0, 1\}$, and all edges in $\mathcal{D}_i$ are as well as $v$'s children $N_{G}^{\text{out}}(v)$ are explored. Note that, since $G$ is not necessarily a tree, such endpoints have possibly been explored prior to the expansion of $v$.

A node $v$ is called proved if a winning strategy for Player 0 can be inferred from the explored part of $G$; if a winning strategy for Player 1 can be inferred, it is called disproved. A node that is neither proved nor disproved is said to be unproved.

The (dis)proof number of an explored node $v$ is defined to be the minimum number of explored unproved nodes that have to be (dis)proved for $v$ to be (dis)proved. Clearly, the (dis)proof number of a (dis)proved node is 0; by convention the proof number of a disproved node and the disproof number of a proved node is defined to be $\infty$.

For any explored node $v$, PNS maintains estimates $\text{pn}(v)$ and $\text{dn}(v)$ for the proof and disproof numbers, respectively, of $v$. For any unexplored node $v \in V$ (in particular, $v_0$ at initialization), PNS sets $\text{pn}(v) = 1$ and $\text{dn}(v) = 1$ in accordance with the definition of the actual proof and disproof numbers.

In each iteration, PNS selects a most promising node to expand. To do so, it first selects $v_0$. If the currently selected node $v$ has already been expanded, PNS distinguishes two cases. If $v \in V_0$, PNS next selects a node $c$ in $N_{G}^{\text{out}}(v)$ minimizing $\text{pn}(c)$. Otherwise, i.e., if $v \in V_1$, it selects a node $c \in N_{G}^{\text{out}}(v)$ minimizing $\text{dn}(c)$. Once PNS has selected an unexpanded node, it expands this node.

After expanding a node, PNS first sets $\text{pn}(v) = 1$ and $\text{dn}(v) = 1$ for the newly explored nodes $v$ (as described above). Then it updates $\text{pn}(v)$ and $\text{dn}(v)$ for all the nodes $v$ selected between the expansion that just happened and the expansion before that, in reverse order of selecting. In particular, in case of the just expanded node, PNS may have learned that $v \in \mathcal{W}_i$ for $i \in \{0, 1\}$, or a cycle has just been closed. In accordance with the definition of the actual proof and disproof numbers, upon learning Player 0 wins, it sets $\text{pn}(v) = 0$ and $\text{dn}(v) = \infty$; upon learning that Player 1 wins, it sets $\text{pn}(v) = \infty$ and $\text{dn}(v) = 0$. In all other cases, for $v \in V_0$ it sets

$$\text{pn}(v) := \min_{c \in N_{G}^{\text{out}}(v)} \text{pn}(c); \quad \text{dn}(v) := \sum_{c \in N_{G}^{\text{out}}(v)} \text{dn}(c) \tag{1}$$

and for $v \in V_1$ it sets

$$\text{pn}(v) := \sum_{c \in N_{G}^{\text{out}}(v)} \text{pn}(c); \quad \text{dn}(v) := \min_{c \in N_{G}^{\text{out}}(v)} \text{dn}(c) \tag{2}$$

If $G$ is an out-tree, it is easy to see that $\text{pn}(v)$ and $\text{dn}(v)$ reflect the actual (dis)proof numbers. In a general graph, however, as can be seen, e.g., in the graph shown in Figure 1, this is not necessarily the case.

**Depth-First Proof-Number Search.** Depth-First Proof-Number Search (DFPN) [Nagai, 2002] does not start the search for the most promising node from $v_0$ in each iteration. Instead, when node $v$ is currently selected, it tries to use thresholds $\text{th}_{\text{pn}}(v)$ and $\text{th}_{\text{dn}}(v)$ to decide whether the path taken from $v_0$ to $v$ is part of a path that PNS would take. In particular, if

$$\text{pn}(v) < \text{th}_{\text{pn}}(v); \quad \text{dn}(v) < \text{th}_{\text{dn}}(v), \tag{3}$$

then it continues the search like PNS would, and otherwise it backtracks one step. The values of $\text{pn}$ and $\text{dn}$ are only recomputed at the node that DFPN is currently considering.

The thresholds are determined as follows. Initially, $\text{th}_{\text{pn}}(v_0) := \infty$ and $\text{th}_{\text{dn}}(v_0) := \infty$. Suppose from node $v$, node $c_1$ is selected next, i.e., if $v \in V_0$ ($v \in V_1$), $c_1$ has the smallest value of $\text{pn}$ ($\text{dn}$) among $N_{G}^{\text{out}}(v)$. If $|N_{G}^{\text{out}}(v)| = 1$, the thresholds for $c_1$ are taken over from $v$. Otherwise, let $c_2$ be a node that has the smallest value of $\text{pn}$ ($\text{dn}$) among $N_{G}^{\text{out}}(v) \setminus \{c_1\}$. If $v \in V_0$,

$$\text{th}_{\text{pn}}(c_1) := \min\{\text{th}_{\text{pn}}(v), \text{pn}(c_2) + 1\}, \quad \text{th}_{\text{dn}}(c_1) := \text{th}_{\text{dn}}(v) - \text{dn}(v) + \text{dn}(c_1), \tag{4}$$

and if $v \in V_1$,

$$\text{th}_{\text{pn}}(c_1) := \text{th}_{\text{pn}}(v) - \text{pn}(v) + \text{pn}(c_1), \quad \text{th}_{\text{dn}}(c_1) := \min\{\text{th}_{\text{dn}}(v), \text{dn}(c_2) + 1\}. \tag{5}$$

**Graph History Interaction Problem.** As pointed out in [Palay, 1983], DFPN and PNS as stated above are not necessarily correct on graphs that contain cycles: A Proof or a disproof of some node $v$ that is found when $v$ is visited through some path $P$ cannot certainly be reused when $v$ is visited through some path $P' \neq P$. This is called the Graph History Interaction Problem. As a general solution [Kishimoto and Müller, 2004b; Kishimoto, 2005], one can, upon (dis)proving a node when visiting it through $P$, save the (dis)proof only with respect to $P$. When visiting $v$ again through $P'$, it can be verified if the previously found proof is still valid. For further details, we refer the reader to the aforementioned works.

**Threshold Controlling Algorithm.** To break infinite loops, the Threshold Controlling Algorithm (TCA) [Kishimoto, 2010] maintains a value $\text{md}(v)$ for every node $v \in V$. It represents the (minimum) distance between $v_0$ and $v$ in the
explore part of $G$. Whenever a node $v$ is visited that has an unproved old child $c$, i.e., $md(c) < md(v)$, TCA adjusts $td_{pn}(v)$ and $td_{dn}(v)$ such that DFPN does not backtrack. In particular, it sets $td_{pn}(v)$ to $\max\{td_{pn}(v), pn(v) + 1\}$ and $td_{dn}(v)$ to $\max\{td_{dn}(v), dn(v) + 1\}$, so that (3) is satisfied. The fact that the thresholds have been increased is passed to subsequent recursive calls of the algorithm, prompting these calls to also increase the corresponding thresholds, until a node is expanded or a cycle is closed (i.e., progress is made).

3 Completeness of DFPN with TCA

We start this section with (re-)justifying variants of DFPN by proving that DFPN with TCA is complete. We provide the remaining proofs in the full version.

3.1 Simpler Counter Example without TCA

We consider the graph shown in Figure 1. In that graph, DFPN visits $v_6$ and $v_7$ alternatingly from $v_1$ via $v_2$, $v_4$, $v_6$ and $v_3$, $v_5$, $v_7$, respectively, in an infinite loop. In particular, it never expands $v_8$. When $W_o = \{v_9\}$ and $W_i = \emptyset$, DFPN therefore never discovers the only winning strategy for Player 0 from $v_0$, which includes $v_9$.

3.2 Proof of Completeness with TCA

The purpose of this section is to prove the following theorem.

Theorem 1. DFPN with TCA is complete.

In the proof, we assume towards a contradiction that DFPN with TCA runs into an infinite loop. First, we remove all the parts of $G$ that are irrelevant during the infinite loop, obtaining a new graph $L = (V', E')$. More precisely, let $P^* := v_0, \ldots, v^*$ be the largest path that is a prefix of $P$ throughout the infinite loop. Then

- $V'$ contains all vertices in $P^*$, all vertices visited during the loop, and all their children;
- $E'$ contains all edges in $P^*$ and outgoing edges from vertices visited during the loop.

For instance, for the loop considered in Subsection 3.1, $L$ would consist of all nodes but $v_9$ and all edges but the one leading to $v_9$.

For a node, $v \in V'$, we now define its level $\ell(v)$ to be the length of the longest $v_0$-$v$ path in $L$. Note that in the proof for DFPN on directed acyclic graphs [Kishimoto and Müller, 2008], levels are also used but defined on $G$ rather than $L$. Let $\ell_{\text{max}}$ be the maximal level of any node in $L$. The fundamental property of $L$ is the following.

Consider a call during the infinite loop at some node $n_c$ that eventually returns because the threshold condition is not met anymore, i.e., $pn(n_c) \geq th_{pn}(n_c)$ or $dn(n_c) \geq th_{dn}(n_c)$. Denote by $M$ the set of nodes (including $n_c$) at which the threshold condition had been violated in the meantime.

Lemma 1. Consider some $m \in M$. There is no node $o \in V \setminus \{m\}$ with $\ell(o) \geq \ell(m)$ such that there exists a path from $o$ to $m$ in $L$.

Note that the statement is trivially true in acyclic graphs, in which an $v_0$-$o$ path can always be extended by an $o$-$m$ path to a longer path, but not in cyclic graphs. We provide a proof.

Proof of Lemma 1. Suppose such a node $o$ exists, and denote by $P'$ the $o$-$m$ path in $L$. First note that $o$ must be visited during the infinite loop. The reason is that otherwise

- either $o$ is part of $P^*$, in which case the only $v_0$-$o$ path in $L$ (a prefix of $P^*$) could be extended by an $o$-$m$ path in $L$, contradicting $\ell(o) \geq \ell(m)$;
- or $o$ is a child of a node visited during the loop with $N_L^\text{out}(o) = \emptyset$, contradicting the fact that $P'$ exists.

Next, note that for any node $p$ in $P^*$ (in particular $m$) it must hold that $md(o) < md(p)$ during the loop. This is because otherwise there must exist a node $p'$ on $P^*$ with successor $p''$ on $P'$ such that $md(p') \geq md(p'')$. Upon visiting $p'$, TCA would then increase the thresholds. This already contradicts the assumption that DFPN with TCA is in an infinite loop because it would eventually make progress by either expanding an unexpanded node or finding a new cycle.

Now consider the $v_0$-$o$ path $P''$ of length $\ell(o)$ in $L$. Since $\ell(o) \geq \ell(m)$, $P''$ cannot be extended to a $v_0$-$m$ path by $P'$. The reason for that must be that $P''$ intersects $P'$ at some node $p_0$. By the same argument as above, applied to the $p_0$-$o$ subpath of $P''$, it holds that $md(p_0) < md(o)$.

Hence, $md(o) < md(p_0) < md(o)$; a contradiction.

The remaining part of the proof is quite similar to the proof of completeness of directed acyclic graphs [Kishimoto and Müller, 2008]. Indeed, we call a node $v \in V'$ consistent if it fulfills (1) and (2) (as equations rather than assignments) and inconsistent otherwise. We also define the inconsistency tuple to be $(N_{\ell_{\text{max}}}, N_{\ell_{\text{max}}-1}, \ldots, N_0)$ where, for $i \in \{\ell_{\text{max}}, \ldots, 0\}$, $N_i$ is the number of inconsistent nodes at level $i$ of $L$. We call $L$ $i$-consistent if $N_i = N_{\ell_{\text{max}}-1} = \ldots = N_1 = 0$.

Using Lemma 1, the proofs of the following two lemmata are now quite similar to the completeness proof on acyclic graphs [Kishimoto and Müller, 2008].

Lemma 2. Let $n \in V'$, suppose $L$ is $\ell(n)$-consistent, and $pn(n) < th_{pn}(n)$ as well as $dn(n) < th_{dn}(n)$ holds. If the algorithm now searches $n$, it will expand a node or find a new cycle before the call returns.

For the next lemma, recall that a tuple $(a_1, a_2, \ldots, a_k)$ is lexicographically smaller than a tuple $(b_1, b_2, \ldots, b_k)$, denoted $(a_1, a_2, \ldots, a_k) <_{\text{lex}} (b_1, b_2, \ldots, b_k)$, if for some $i$, it holds that $a_1 = b_1, a_2 = b_2, \ldots, a_i = b_i$ and $a_{i+1} < b_{i+1}$.

Lemma 3. Suppose not all nodes in $L$ are consistent. Denote by $T$ and $U$ the inconsistency tuples right before $n_c$ is searched and right after the call has returned, respectively. Then $U <_{\text{lex}} T$.

We provide the remaining proofs in the full version.
4 Adaptation of DFPN to Multiple Solutions

Since the algorithm is deterministic, running it multiple times would only lead to the same solution each time. To find multiple solutions, we modify PNS and its variants, we change the type of diversity we obtain for the set of found solutions. However, since we might end up at these exact nodes by following a different path, we do not want to completely neglect such nodes. For that we keep track of the path we used to get to such a node and store it. If we encounter the same node again, we check if the path that was used to reach it was stored. If so, the node will stay disproved, otherwise we can use it again.

To further control the diversity of the solutions, the algorithm penalizes every node used by a found winning strategy. This happens by adding a penalty $p_{AND}$ to the values of $pn$ for every AND-node $v \in V_1$ in a path to a node selected by the above diversity controlling strategy: $pn(v) := pn(v) + p_{AND}$. Adding a penalty $p_{OR}$ to the values of $pn$ for all $v \in V_0$ represents an additional approach. Afterwards the values of $pn$ and $dn$ of all previous nodes get updated according to (1) and (2). The penalties are adjustable. Higher penalties lead to more diverse routes but longer computation times.

5 Application to Retrosynthesis

Retrosynthetic planning is, as described in the introduction, the task of finding one or multiple suitable routes for the synthesis of a given target molecule. This challenge can also be modeled as a two player game in the following way: Here, one identifies the molecules with the nodes from which Player 0 may move, and the reactions with the nodes from which Player 1 may move. If a molecule is the product of a reaction or a reaction requires a certain molecule, a directed edge is added between the corresponding nodes. The node corresponding to the target molecule is the start node. Player 0 wins if a node corresponding to a buyable molecule is reached, and Player 1 wins if a dead end (not corresponding to a buyable molecule) is reached or a node is visited for the second time.

Now a winning strategy for Player 0 represents a synthesis plan for the target molecule. If there exists a winning strategy for Player 1, the target molecule cannot be synthesized.

5.1 Heuristic Edge Cost in DFPN-E

Earlier we mentioned the DFPN-E algorithm by Kishimoto et al. [2019], which uses a heuristic function that evaluates the cost of using a certain edge from a molecule $n$ to its child (i.e., reaction) $c$:

$$h(n, c) = \min(M_{pn}, [-\log(P(n, a) + \epsilon) + 1]),$$

where the constant $M_{pn}$ is used to limit the size of $h$, and the constant $\epsilon$ prevents a calculation of $\log(0)$.

For every template (i.e., a class of reactions) $a$ in a given database, $P(n, a)$ represents the probability of this template $a$ being used to synthesize the molecule $n$. To estimate $P(n, a)$, Kishimoto et al. use an Artificial Neural Network. Applying template $a$ to molecule $n$ results in the reaction $c$, the child of $n$. While calculating $pn(n)$, the value of $h$ is added to $pn(c)$ for all $c \in N_{out}(n)$. Additionally the computation of $th_{pn}(c_1)$ in (4) is modified to

$$th_{pn}(c_1) := \min(th_{pn}(n), pn(c_2) + \sigma) - h(n, c_1),$$

where $\sigma$ is the so called threshold controlling parameter. It is used to limit the number of visits to the same node in a trade off to a potentially less precise search.

5.2 Multiple Solutions in DFPN*

To obtain our algorithm DFPN* for Retrosynthesis, we apply a diversity controlling strategy from the framework from Section 4 to DFPN-E. More specifically, the single node we choose to disprove as part of our diversity controlling strategy is a deepest reaction in the found route, i.e., a reaction reached through a longest path from the target molecule. By doing so, we force the algorithm to find shorter routes without destroying too many possible routes. The reason we choose a reaction rather than a molecule to disprove is that disproving a molecule leads to a disproof of a reaction anyway. For the same reason, we set $p_{OR} = 0$.

6 Experiments

In order to verify whether the MCTS and the DFPN* provide different quality levels in diversity, we performed computer experiments. We analyze a sample data set consisting of 60 molecules\(^1\). These are collected from different sources: 40 molecules picked up from a public benchmark data set on synthesizability [Ertl and Schuffenhauer, 2009] and 20 drug-like molecules were provided by Bayer chemists. These molecules have been classified by synthesis chemists from easy to challenging in synthesis. For these molecules we have determined synthesis routes by applying both DFPN* and MCTS.

6.1 Setup

We use an Artificial Neural Network (ANN) to estimate the success probability $P(n, a)$ that the molecule $n$ can be synthesized in one step with the reaction template $a$. To train the

\(^1\)available at https://doi.org/10.5281/zenodo.6511731
ANN, we had a total of 31 million reactions at our disposal. These reactions were collected from several public and non-public data sources: 60% from publications and 40% from Bayer’s internal laboratory journals.

Based on the reaction database we have generated, after extensive data cleaning and preprocessing, 8,616,239 pairs \((n, a)\) of molecules \(n\) and reaction templates \(a\). In total, we have identified 270,605 reaction templates. The input to the neural network is the molecule \(n\) (converted to a Morgan fingerprint). The output is a vector of length 270,605, which indicates for each template the probability that the template can be successfully applied to \(n\).

Theoretically, more than 270,000 edges would be attached to each node. However, this is a priori reduced by similar heuristic rules described in [Segler et al., 2018] to a maximum of 35 edges.

Recall DFPN-E is used as a foundation for our DFPN* algorithm. Constants such as \(M_{\text{fn}}\) and \(\epsilon\) are set as in [Kishimoto et al., 2019] \((M_{\text{fn}} = 20\) and \(\epsilon = 10^{-30}\)). The only constant we changed is the threshold controlling parameter \(\sigma\) (they call it \(\delta\)). Kishimoto et al. use \(\sigma = 2\), whereas we set \(\sigma = 3\), because it resulted in better overall solutions in our experiments. The penalties \(p_{\text{AND}}\) and \(p_{\text{OR}}\) from Section 4 were set to \(p_{\text{AND}} = 10\) and \(p_{\text{OR}} = 0\). In addition to the aforementioned solution to the Graph History Interaction Problem and the Threshold Controlling Algorithm, the DFPN* also implements a maximum search depth, limiting the amount of reactions that need to be executed consecutively for each synthesis. We set it to 7 and the maximum branching factor (i.e., the maximum amount of reactions for a molecule) to 50.

For MCTS we used the current version of [MLPDS, 2020] and, to match the parameters of DFPN*, the exploration constant was set to 2.0, the maximum depth to 7, and the maximum branching to 50. Chemicals were labeled buyable according to the chemical suppliers for Bayer Research.

6.2 Results

MCTS and DFPN* were each run on a single core. The computing time varied between 120 s, 300 s, 600 s, 900 s, and 1200 s. The maximum number of routes was set to 500 for both algorithms. After finishing the computation, the 500 routes with maximum success probability were picked from the set of found routes.

We consider the following performance indicators: The number of molecules solved, the number of routes found per target molecule, the mean number of reactions per route, and the number of intermediate molecules for all routes of a target molecule (see Table 1). Here, the number of solved molecules is the most important quality criterion for the search algorithms. Both algorithms deliver similar results with a slight advantage for the DFPN*.

It turns out that the number of routes that MCTS finds is much larger than that number for DFPN*. However, the sheer number of routes is arguably not important without considering their diversity.

As a measure of diversity, we use the total number of different intermediates used across all routes. If the same molecules always appear in different routes for a target molecule, this indicates that only little is changed in the routes; perhaps only the order of the reactions is switched. Although MCTS provides significantly more routes, the number of different molecules is actually much larger for the DFPN*; the routes of the DFPN* are more diverse.

The number of reactions per route is a measure of the synthesis effort. The average route length for DFPN* is shorter than for MCTS. This means that the higher diversity of the DFPN* is not caused by longer routes. On average, even one reaction step less is required than with MCTS.

As a measure of diversity, we use the total number of different intermediates used across all routes. If the same molecules always appear in different routes for a target molecule, this indicates that only little is changed in the routes; perhaps only the order of the reactions is switched.

Although MCTS provides significantly more routes, the number of different molecules is actually much larger for the DFPN*; the routes of the DFPN* are more diverse.

The number of reactions per route is a measure of the synthesis effort. The average route length for DFPN* is shorter than for MCTS. This means that the higher diversity of the DFPN* is not caused by longer routes. On average, even one reaction step less is required than with MCTS.

7 Conclusion

In this paper, we provided a proof of completeness of DFPN with TCA, adapted DFPN to finding multiple solutions, and conducted first experiments. Directions for future work include further experiments (in particular, comparisons with the other approaches discussed in Section 1.2) and considering parallelized versions of PNS.

Acknowledgements

We thank Yannic Alber for his help with the experiments.

<table>
<thead>
<tr>
<th>computation time</th>
<th>MCTS</th>
<th>DFPN*</th>
</tr>
</thead>
<tbody>
<tr>
<td>solved molecules</td>
<td>38</td>
<td>41</td>
</tr>
<tr>
<td>median routes per target molecule</td>
<td>396</td>
<td>72</td>
</tr>
<tr>
<td>mean different molecules per target(^2)</td>
<td>101</td>
<td>347</td>
</tr>
<tr>
<td>mean number of reactions per route(^2)</td>
<td>6.0</td>
<td>4.9</td>
</tr>
</tbody>
</table>

\(^2\) solved target molecules only
References


