games

MDPI

Article

Improving Strategic Decisions in Sequential Games by Exploiting Positional Similarity

Sabrina Evans [1,2] and Paolo Turrini [3,*]

[1] Department of Mathematics, Yale University, New Haven, CT 06511, USA; sabrina.evans997@hotmail.co.uk
[2] Bloop AI, London WC1H 9SE, UK
[3] Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK
* Correspondence: p.turrini@warwick.ac.uk; Tel.: +44-(0)24-7652-3193

Abstract: We study the strategic similarity of game positions in two-player extensive games of perfect information by looking at the structure of their local game trees, with the aim of improving the performance of game-playing agents in detecting forcing continuations. We present a range of measures over the induced game trees and compare them against benchmark problems in chess, observing a promising level of accuracy in matching up trap states. Our results can be applied to chess-like interactions where forcing moves play a role, such as those arising in bargaining and negotiation.

Keywords: limited foresight; traps; similarity; strategy

Citation: Evans, S.; Turrini, P. Improving Strategic Decisions in Sequential Games by Exploiting Positional Similarity. *Games* **2023**, *14*, 36. https://doi.org/10.3390/g14030036

Academic Editors: Abhinay Muthoo and Ulrich Berger

Received: 7 January 2022
Revised: 10 February 2022
Accepted: 7 April 2022
Published: 28 April 2023

1. Introduction

In complex sequential interactions, such as those arising in bargaining and negotiation, boundedly rational participants are often not in a position to fully calculate the consequences of their own decisions and need to make a judgment call on which move to make next. These interactions are often compared to chess, for the importance of forward thinking, opponent modelling and prediction, surprise moves and deceptive concessions (for an interesting take on the connections between chess and negotiation, see [1]). However, the right way to analyse them is far from obvious. Game theory offers perhaps the most natural toolbox to do so, but it is important to strike a good balance between the simplicity of the solution and its applicability.

In his book *Modelling Bounded Rationality* [2], Ariel Rubinstein says:

> At the beginning of the twentieth century, Zermelo proved a proposition which can be interpreted: "chess is a trivial game". [2] (p. 130).

This sentence ironically revisits the seminal result of Zermelo on the determinacy of two-player zero-sum games (for a modern and neat technical exposition see [3]; for a comprehensive account of its historical relevance, including the original proof of the result, see [4]) which rarely scales up to games that are played in practice, such as chess. In a way, Rubinstein notes, there is a *game theory* and a *game practice*, and the often idealised toy models from the former abstract away from many important features in the latter, which we should not forget about. Some games have a solution *in theory*, but this solution can be very hard to find in practical play.

Rubinstein continues:

> But [in games like chess] this calculation requires going through a huge number of steps, something no human being can accomplish. Modeling games with limited foresight remains a great challenge [and the frameworks studied thus far] fall short of capturing the spirit of limited-foresight reasoning. [2] (p. 131).

Bounded Decision-Makers. The modern advances in Artificial Intelligence, particularly the work of the Google DeepMind team in games of perfect information and beyond (see, e.g., [5,6]), have shown how limited human knowledge is when it comes to decision making in complex extensive games of perfect information. However, chess, as well as many other chess-like games, is not *solved*, in the sense that we do not yet know an optimal strategy from the start of the game. Even if, thanks to Zermelo, we know that either White has a winning strategy, or Black has a winning strategy, or both players at least have a drawing strategy, we still do not know which one of these disjuncts is true. In fact, even if/when found, the solution would be hard to remember or even store anywhere. Discussing Zermelo's result [3], Maschler, Solan and Zamir claim that, should chess be solved, it will "cease to be of interest" [3] (p. 3). However, this is far from obvious when boundedly rational players are involved. Think of issues such as time pressure, when grandmasters and even supercomputers manage to lose theoretically won positions.

We cannot stress enough how bounded rationality is not just a "human" feature. Any decision-making agent operating in large extensive games, including supercomputers, needs to resort to heuristic assessments to make decisions. Artificial Intelligence can then provide us with stylised and testable models of boundedly rational decision-making, which we can use to support human decision-making as well—a point of view which we take in this paper.

An Issue with the Current AI Methods. When analysing complex extensive games of perfect information, where intermediate positions need to be assessed heuristically, Artificial Intelligence has come up with approximate methods, the most successful of which is Monte Carlo Tree Search (MCTS), which evaluates non-terminal positions by repeated simulation—see [7] for a full description of MCTS methods.

MCTS constitutes an evolution from traditional $\alpha - \beta$ methods, which are decision-making mechanisms that assign a utility to "features" of a game position and select a continuation based on max–min solution concepts (for a detailed description of these methods, see [8]). MCTS-based agents dominate many games of perfect information, such as Go, Checkers, Reversi, and Connect Four [7]; witness the impressive achievements of the DeepMind team against human players [5,6]. Nonetheless, MCTS-based agents still trail behind their $\alpha - \beta$ counterparts when playing game positions requiring accurate play. Recent evidence of this fact was provided in game 6 of the 2018 Chess Championship Match between World Champion Magnus Carlsen (White) and challenger Fabiano Caruana (Black), where DeepMind's AlphaZero missed a mate for Black following a sequence of 30 moves, found by the Sesse supercomputer running this line on the non-MTCS-based Stockfish [9].

A key issue for MCTS's performance in games such as chess is the presence of *trap states*, where an initial move may look strong and then be followed by a forcing sequence of moves by the opponent, leading to a loss or significant disadvantage [10]. In other words, the ubiquitous presence of reachable subgames which admit (practically) winning strategies from the opponent. Despite the breakthroughs of MCTS-based engines, the challenge still remains to equip MCTS with the capability of handling such forcing lines.

The approach we take in this paper is to formulate a generalised notion of similarity between game states to improve the performance of game-playing agents by smart search: if a trap was found after position x and we now analyse position y, which is very similar to x, then chances are we still have a trap after y. As similar strategies are likely to contain similar move sequences but not necessarily similar board positions, our measures are based on possible moves from each position, rather than the appearance of the board itself.

Our Contribution. We study the similarities of game positions in two-player, deterministic games of perfect information, looking at the structure of their local game trees, working with the set of possible moves from each position. We introduce novel similarity measures based on the intersection of move sets, and a structural similarity measure that only considers the arrangement of the local game tree and not the specific moves entailed. We analyse the formal relation of these measures and test them against benchmark

problems in chess, with a number of surprising and promising findings. Notably, our structural similarity measure was able to match trap states to their child-trap states with 85% accuracy without using domain-specific knowledge. On top of this, we introduce a move-matching algorithm, which accurately pairs moves with similar strategic value from different positions. Our results are of immediate relevance to MCTS adaptations to detect and avoid trap states in gameplay.

Other Related Work. Graph comparison is one of the most fundamental problems of theoretical computer science, with graph isomorphism computation having been an open problem for quite some time [11]. With tree structures, possibly the most commonly used metric is the *edit distance* [12]: based on the number of edits (node insertions, deletions, and substitutions) necessary to transform one tree into another, this metric works well for trees of a similar size with many shared nodes and edges. However, it tends to be less suitable when comparing multiple trees of different sizes, as large trees sharing some proportion of their nodes appear further from each other than two completely distinct smaller trees. An alternative measure is the *alignment distance* [13], an adaptation of the edit distance based on the notion of sliding one tree into another and counting the number of edits needed to transform both trees into the combined one. The alignment distance requires lower complexity to compute than the edit distance, but it is technically not a metric and suffers from similar problems comparing trees of different sizes.

In game playing, the presence of forcing continuations is identified as a key problem faced by AI engines, with more acute implications for chess-like games [10]. Surprisingly, though, the theory of similarity metrics to aid strategic decisions in game playing is not well developed.

Similarity measures have instead been used in other areas of AI, as in the case of Siamese neural networks for one-shot learning [14]. In this case, two symmetric convolutional neural networks were trained on same–different pairs and then shown a test instance, as well as one example from each possible classification. The output of the twin networks was then compared using a similarity measure. Here, a cross-entropy objective function was used to determine similarity, but this required the networks to be symmetric and weight tied. New similarity measures based on the structural similarity of networks could remove these requirements, but have not yet been investigated.

Paper Structure. The section "Positional Similarities" introduces our formal setup to compare game trees through a number of similarity measures. The section "Detecting Structural Similarities" uses these as the basis of a dynamic algorithm to detect structural similarities among subtrees. In the "Performance" Section, we compare these against known chess positions. We conclude by discussing potential applications and research directions.

2. Positional Similarities

Let G be a two-player finite extensive form game of perfect information, where players, e.g., Black and White, alternate moves, with White starting the game. Formally, G consists of a set of histories $(x_0, x_1, \ldots, x_K)$ such that x_0 is the starting board position and each x_{k+1} (with $k \leq K$) can be reached from x_k with a single legal move by White whenever k is even, and by Black otherwise (as in, e.g., [3]).

We are interested in comparing trees that result from players exploring game continuations from a certain board position on. In MCTS, for example, these are the game trees generated by the expansion step (see, e.g., [15]). Let T_1, T_2 denote tree roots and T_{1i}, T_{2i} child nodes (board positions) of T_1 and T_2. Then let M_i denote the set of all possible moves from position T_i and M_i^d the set of all possible moves contained in all possible move sequences of length d from position T_i.

We now present three natural measures, of increasing complexity, to establish how similar such trees are: the similarity of continuations, the similarity of sequences, and the tree-edit similarity. All these measures are model free, in the sense that they can be used in all situations that can be described as two-player finite extensive games of perfect

information. We analyse their formal interrelation in this section and use them as the basis of our dynamic algorithm in the subsequent one.

2.1. Similarity of Continuations

Our first measure, which we call *similarity of continuations*, is calculated from the sets M_1^d, M_2^d of 1-ply atomic moves from starting positions T_1, T_2 to their children of depth d. The similarity is the size of the intersection of these two sets divided by the size of their union.

$$P_{cont}(T_1, T_2) = \frac{|M_1^d \cap M_2^d|}{|M_1^d \cup M_2^d|}$$

As M_1^d, M_2^d can be found from a simple expansion of the game trees, computing such a measure takes time $\mathcal{O}(|M_1^d| + |M_2^d|) \sim \mathcal{O}(b^d)$, where b is the breadth of the game tree.

At depth 1, the similarity of continuations simply calculates the proportion of children that two nodes share. When extended to a deeper search, the measure becomes less fine-grained, since a move that occurs at different depths in the trees will still count as shared, and multiple occurrences of the same move are only counted once.

As an example, consider the trees T_1, T_2 in Figure 1, which have depth-2 continuation sets $M_1^2 = \{a, b, c, d, e, f\}$, $M_2^2 = \{a, b, c, d, e, g, h\}$.

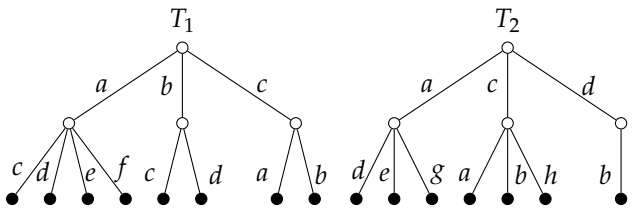

Figure 1. Tree comparison. Starting from the same position, metrics can be used to establish how close the resulting positions will be.

Here,

$$P_{cont}(T_1, T_2) = \frac{|M_1^2 \cap M_2^2|}{|M_1^2 \cup M_2^2|} = \frac{5}{8} = 0.625.$$

2.2. Similarity of Sequences

Our second similarity measure, which we call *similarity of sequences*, uses longer sequences of moves rather than single plies. To ease computation, we require each possible move sequence of length d from tree root T_1 to first be rewritten according to a predetermined move, ordered as a *simplified sequence* S. Formally, two sequences are simplified into one if—and only if—they are the same modulo move permutation. These simplified sequences are then stored in a structure T_1', which we call the *simplified tree* of T_1. As different move permutations can create the same simplified sequence, we also store the *multiplicity* k of each S in T_1', where k corresponds to the number of ways S can be reached from the root note. Then, the similarity of sequences calculates the ratio of the intersection to the union of the simplified trees.

Let k_{1i} be the multiplicity of simplified sequence S_i in T_1', k_{2j} the multiplicity of S_j in T_2', and $n = max(|T_1'|, |T_2'|)$, and the number of nodes in the larger of T_1', T_2'. Then, the similarity of sequences of T_1 and T_2 is given by

$$P_{seq}(T_1, T_2) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} (k_{1i} + k_{2j}) \mathbb{1}_{S_i \cap S_j}}{\sum_{i=1}^{n} k_{1i} + \sum_{j=1}^{n} k_{2j}}.$$

Calculating the similarity of sequences at depth 2 on the example trees in Figure 1 can be achieved as follows. For an alphabetical ordering, the simplified trees can be written as

$$T_1' = \{ac^2, ad^1, ae^1, af^1, bc^2, bd^1\}$$

$$T_2' = \{ac^1, ad^1, ae^1, ag^1, bc^1, bd^1, ch^1\}$$

where the superscript corresponds to the multiplicity of each sequence. Then

$$P_{seq}(T_1, T_2) = \frac{12}{15} = \frac{4}{5} = 0.800.$$

The tree simplification can be performed in one depth-first pass of each tree, taking time $\mathcal{O}(b^d)$. Calculating the proportion of shared sequences takes $\mathcal{O}(|T_1'| + |T_2'|)$, which is equal to $\mathcal{O}(b^d)$ in the worst case, the same logarithmic complexity as the similarity of continuations. It should be noted that the tree-reduction step means that the complexity coefficient is larger for the similarity-of-sequences calculation. This is a trade-off for accuracy at depth $d > 1$, as less information is lost when calculating from sequences rather than continuations.

Relation to Kernels. The similarity of sequences is closely related to the Tanimoto similarity measure or kernel [16,17] based on the intersection over the union of the inner products of two sets. The Tanimoto kernel was successfully used to calculate the similarity of molecule fingerprints in Bioinformatics from the feature map of a molecule by counting the number of paths through the map shared by different molecules [16]. The methods used in this area can be carried over to extensive form games of perfect information, as a board position can be viewed as a fingerprint representing the game that has gone before it. The game tree and feature map can both be traversed and have their matching paths counted. Using a suffix tree data structure [18,19], we can compute the Tanimoto kernel in time $\mathcal{O}(d(n_1m_1 + n_2m_2))$, for depth d, n_i nodes and m_i edges in trees T_1, T_2. The similarity of sequences is also comparable to the random walks kernel [20], a measure of similarity between two graphs found by counting the number of random paths they share. The main difference here is that the similarity of sequences has limited depth and is a normalised metric.

2.3. Tree-Edit Similarity

It may be the case that T_1 is very similar to T_2 but differs by some very shallow moves. If this is the case, the similarity of sequences measure would not detect this similarity. We therefore propose a modified version of the tree-edit distance [21], which traditionally counts the cost-wise minimal number of operations needed to turn one tree into another. The *tree edit similarity*, used to compare subtrees, is normalised, and acts as a metric on the tree-edit space. The normalised tree-edit distance [21] gives values in the range [0, 1], and as such would be suitable as a similarity measure when subtracted from one. The normalised distance is given as

$$\frac{2e(T_1, T_2)}{\alpha(|T_1| + |T_2|) + e(T_1, T_2)},$$

where $e(T_1, T_2)$ is the tree-edit distance between T_1 and T_2, and α is the weight of edit operations. Since there is no need to weigh edit operations differently, we may take α to be one for all operations. Then, as shown by Li and Zhang [21], the formula is valid as a metric. Since calculating the distance between two trees is equivalent to calculating their similarity and subtracting it from one, we define the tree-edit similarity as

$$P_{tree}(T_1, T_2) = 1 - \frac{2e(T_1, T_2)}{|T_1| + |T_2| + e(T_1, T_2)}.$$

Calculating the tree-edit similarity on the example trees in Figure 1 is as follows:

$$P_{tree}(T_1, T_2) = 1 - \frac{2 \times 6}{11 + 10 + 6} = \frac{15}{27} = 0.556.$$

This measure is the most fine-grained of the three detailed so far. Since calculating tree-edit distance on unordered trees is known to be NP-hard [22], we must again order

the nodes in a preprocessing step with complexity $\mathcal{O}(b^d)$, as above. Once we have ordered trees, the time complexity reduces to $\mathcal{O}(b^{2d}d^2)$ when $d < b$, and $\mathcal{O}(b^{2d+2})$ when $d \geq b$ [12]. As such, the improvements made by the tree-edit similarity over the two previous measures must be weighed against the added complexity.

2.4. Comparing Terminal States

It may sometimes be necessary to find the similarity of two terminal states. In terms of the game tree for a zero-sum game, two terminal nodes should have a value of one if they give the same reward for the agent (win–win, draw–draw, lose–lose), and zero if the reward is different. Since two terminal nodes have no children, their fractional similarity measure is undefined, so we must handle this case separately.

The normalised difference between the rewards of the two terminal nodes can be found by subtracting the reward R_1 of one node from the reward R_2 of the other, then dividing the result by the size of the range of possible reward values S_0, S_1. This gives a value between 0 and 1, where 1 represents rewards at opposite ends of the range, and 0 represents equal rewards. Subtracting from 1 then gives a similarity measure, formalised as

$$P(T_1, T_2) = 1 - \frac{|R_2 - R_1|}{|S_1 - S_0|}.$$

This can be used in endgame cases to prevent zero errors when calculating other similarity measures.

2.5. Relationship between Measures

At depth 1, the similarity of sequences and the similarity of continuations are equivalent, as each child move only appears once per tree. At depth 2, the similarity of sequences has greater variation, as can be seen from the following chess-inspired instance.

Example 1 (Chess trees)**.** *Let T_1, T_2 be nodes of a chess game tree where branching factor b is constant, and T_1, T_2 differ only in the placement of two pieces. Then at depth two*

$$P_{cont}(T_1, T_2) \sim \frac{b-2}{b}, \quad P_{seq}(T_1, T_2) \sim \frac{b^2-2}{b^2}.$$

Now consider positions T_3, T_4, which also differ only in the placement of two pieces, except that in T_3 the opponent has chosen a forcing move leading to checkmate at depth 2, while in T_4 the opponent has chosen otherwise. Then T_4 extends past depth two, but T_3 is truncated and only contains depth 1 moves, all of which are shared with T_4. Then at depth two

$$P_{cont}(T_3, T_4) \sim \frac{b-2}{b}, \quad P_{seq}(T_3, T_4) \sim \frac{b-1}{b^2}.$$

So we can see that

$$P_{cont}(T_1, T_2) < P_{seq}(T_1, T_2),$$
$$P_{cont}(T_3, T_4) > P_{seq}(T_3, T_4)$$

and thus, the similarity of sequences has greater variation than the similarity of continuations. The tree-edit similarity is yet more variable than the similarity of sequences, as can be seen from further calculations on the same examples.

$$P_{tree}(T_1, T_2) \sim \frac{2b^2-1}{2b^2+1} > P_{seq}(T_1, T_2)$$

$$P_{tree}(T_3, T_4) \sim \frac{1}{2b+1} < P_{seq}(T_3, T_4)$$

Modulo is the tradeoff between simplicity and complexity. The above similarity measures can be used to analyse any game trees with consistent move labelling. This would be especially useful for games with less dynamic trees; that is, those without capturing or blocking moves that change the game tree structurally between plies. For games such as Go, with the potential to use one piece to exert power over a whole area, these measures provide useful tools for analysis, which could be further explored by accounting for symmetries and abstractions of the board.

3. Detecting Structural Similarities

We may find ourselves comparing positions that do not share many continuations, e.g., those that are far away from one another in a game tree. What we can then do is to extend the previous approach to recursively check for subtree similarity.

Structural Similarity Measure

Our final similarity measure, which we call the *structural similarity measure*, compares the graphical structure of two game trees without comparing their atomic moves directly. The measure is based on calculating the similarity of each starting position T_1 to each of its child nodes T_{1i} using any of the three previously defined measures, before comparing this list of similarities to the list of similarities of another starting position T_2 to its children T_{2i}. The measure uses an assignment algorithm (see Algorithm 1) to pair each child node of T_1 to a child node of T_2 to minimise the sum of the paired nodes' similarities to their respective parents. If one subtree has more children than the other, each unpaired child adds one to this sum. The sum is then divided by the larger number of children and subtracted from one to provide the structural similarity of the two subtrees, where a value of 1 is identical and zero is completely distinct. Let c_1, c_2 be the number of child nodes of T_1, T_2 respectively. Then, for a selected similarity measure P, the structural similarity measure can be expressed as

$$S(T_1, T_2) = 1 - \frac{|c_1 - c_2| + \min_{(i,j)\text{pairs}}(\sum |P(T_{1i}) - P(T_{2j})|)}{\max(c_1, c_2)}.$$

Algorithm 1 Structural Similarity Detection Protocol

function StrucSim(T_1, T_2)
  $max \leftarrow$ max number of children of T_1, T_2
  $min \leftarrow$ min number of children of T_1, T_2
  for child T_{1i} of T_1 **do**
    $sim_1[i] \leftarrow$ Measure(T_{1i}, T_1)
  for child T_{2j} of T_2 **do**
    $sim_2[j] \leftarrow$ Measure(T_{2j}, T_2)
  pad smaller of sim_1, sim_2 with 1s
  for i, j from 1 to max **do**
    $distances[i, j] \leftarrow |sim_1[i] - sim_2[j]|$
  $matches \leftarrow$ Match($distances$)
  for k from 0 to max **do**
    $total \leftarrow total + matches[k]$
return $(\frac{distances}{max})$

The following calculates the structural similarity measure based on the similarity of continuations at depth 1 on the trees in Figure 1. The similarity of each branch to its root is

$$P(T_1, T_{1.1}) = \frac{1}{6}, \quad P(T_1, T_{1.2}) = \frac{1}{4}, \quad P(T_1, T_{1.3}) = \frac{2}{3}$$

$$P(T_2, T_{2.1}) = \frac{1}{5}, \quad P(T_2, T_{2.2}) = \frac{1}{5}, \quad P(T_2, T_{2.3}) = 0.$$

There are two minimum distance matchings:

$$\{(T_{1.1}, T_{2.3}), (T_{1.2}, T_{2.1}), (T_{1.3}, T_{2.2})\},$$

$$\{(T_{1.1}, T_{2.3}), (T_{1.2}, T_{2.2}), (T_{1.3}, T_{2.1})\}$$

and their total distance is 0.683. So

$$S(T_1, T_2) = 1 - \frac{0.683}{3} = 0.772.$$

While the structural similarity measure may calculate more accurate similarities between positions, this comes at a cost, as each calculation requires similarity computations of every child node to its parent. When the similarity of sequences or continuations at depth 1 is used as the base measure, on average it takes time $\mathcal{O}(b^2)$ to calculate the similarity of all children to their parent. Assigning children in pairs using the Hungarian algorithm takes $\mathcal{O}(b^3)$ operations, so the structural similarity algorithm runs in time $\mathcal{O}(b^3)$. To improve the complexity, the measure could be approximated by randomly sampling child nodes and calculating their structural similarity, which warrants further investigation.

Strategic Similarity of Tic Tac Toe Positions. To investigate convergence of the structural similarity measure to an intuitive similarity of board positions based on their strategic advantage, we manually calculated the structural similarity measure on a small section of the Tic Tac Toe game tree, using the similarity of continuations measure as a basis. Figure 2 shows the game tree for a small segment of a Tic Tac Toe game, and Table 1 contains the results of the similarity analysis, where the branches of the game tree are labelled according to their original board position O and the square in which the next move is made. The measure correctly identifies the rotational symmetry between branches $OB1$ and $OB3$, and gives a value of zero for all comparisons of $OC2$ with distinct subtrees, as $OC2$ is terminal and so shares no structural similarity with any of the other depth 1 nodes. This is very promising, as it shows that the measure behaves well on a solved game, so we can be more confident in trusting it in a heuristic setting.

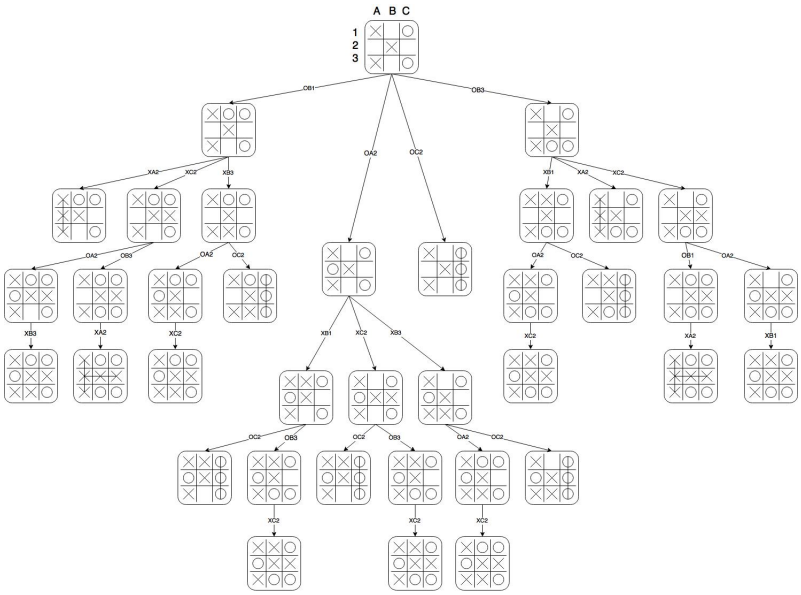

Figure 2. Section of the Tic Tac Toe game tree with moves labelled.

Table 1. Structural similarity of Tic Tac Toe branches in Figure 2.

	OB1	**OA2**	**OC2**	**OB3**
OB1	1	5/6	0	1
OA2	5/6	1	0	5/6
OC2	0	0	1	0
OB3	1	5/6	0	1

Move Matching. As the structural similarity measure pairs moves that are comparably similar to their parent states, this method can be used to pair moves from different board positions that may have similar strategic value. For example, if one position is known to have a killer move in two plies, leading to a win for the opponent, and this position has a high similarity to a new position, the depth 2 matches can be inspected and the move that is most frequently matched to the killer move in the known position can be identified, and this move is likely to be a killer move from the new position. We will evaluate the effectiveness of this approach in the forthcoming section.

Generalisability. The structural similarity measure is generalisable to the analysis of any two local trees with self-consistent move labellings, as the measure can be calculated independently of such labels. This means, e.g., that the structure of a local Go tree can be compared to that of a local chess tree or, alternatively, we can show how a game tree changes through the game.

Calculating how dynamic a game is, in terms of the variability of the connection density of the graph, can be very useful in indicating which gameplay heuristics to use. For example, to use the All-Moves-As-First (AMAF) heuristic, which initially updates sibling nodes with the same estimated value for each move played, an agent first assumes that a move from one node is likely to affect the game in a similar way to the same move played from a sibling node. This may be likely to work on less dynamic games, but could be less reliable for highly dynamic games, where the effect of a move on the state of the game is less consistent. Conversely, pruning may be most helpful for highly dynamic games, as these games offer a stark contrast between reward values for different branches, which is not necessarily the case for less dynamic games.

These hypotheses are supported by studies of successful AMAF use in less dynamic games such as Go [23], Phantom Go [24], Havannah [25], and Morpion Solitaire [26], successful pruning in the dynamic game of Amazons [27] and less successful pruning in Havannah [25].

4. Performance

We tested how effective the first three similarity measures were at detecting nearby trap states in chess, using the similarity of continuations at depth $d = 1$, similarity of sequences at $d = 2$ and tree-edit similarity at $d = 2$. We chose a sample of four distinct trap states which each lead to checkmate within 2 to 4 plies, as shown in Figure 3. We used a sample of all 1000–1500 board positions that were two plies away from each trap state, and recorded whether the trap was maintained or not for each new position. The measures were calculated on each of these board positions, as was a cross-correlation measure that was used as a control, calculated by finding the number of squares where piece placement differed and dividing this number by 64. The similarity of sequences was adapted for chess by including captures in the simplified sequences. This adaptation can be generalised to any game with irreversible moves by recording the irreversible moves from each sequence as well as its standard moves.

Clearly, an effective measure should evaluate trap states as highly similar to the original position with high frequency, so we fixed a threshold value ρ and calculated the proportion of trap and non-trap states with similarity higher than ρ for each measure. For each trap state and each of our similarity measures, when ρ was set to the average value of the similarities, around 70% of all children that were also trap states had above average similarity to the original position, and consistently over 50% of non-trap children had

below average similarity. This was not the case for the cross-correlation, where up to 87% of trap states had below average similarity, and 72% of non-trap states had above average similarity. These results can be seen in Table 2.

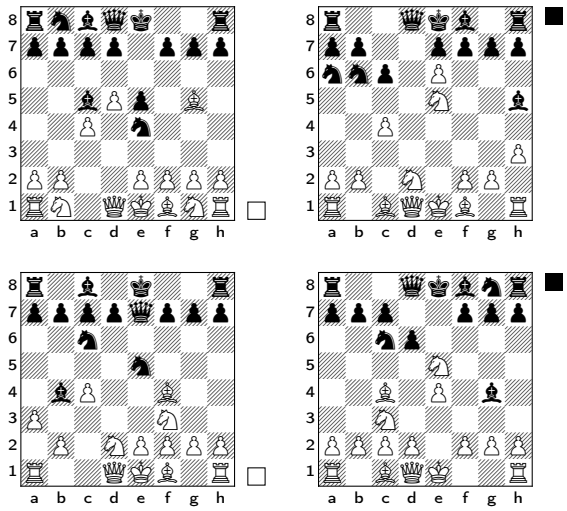

Figure 3. Benchmark positions: (**Top Left**) Trap from Budapest Gambit: 1.d4 ♘f6 2. c4 e5 3.d5 ♗c5 4.♗g5 ♘e4, to be followed by 5.♗xd8 ♗xf2#. (**Top Right**) Trap after mistake in Caro–Kann Defence, Breyer variation: 1. e4 c6 2. d3 d5 3. ♘d2 dxe4 4. dxe4 ♘f6 5. ♘gf3 ♗g4 6. e5 ♘d5 7. h3 ♗h5 8. c4 ♘b6 9. e6 ♘a6 10. ♘e5, to be followed by ... ♗xd1 11. exf7#. (**Bottom Left**) Kieninger Trap: 1.d4 ♘f6 2.c4 e5 3.dxe5 ♘g4 4.♗f4 ♘c6 5.♘f3 ♗b4+ 6. ♘bd2 ♕e7 7.a3 ♘gxe5, to be followed by 8.axb4 ♘d3#. (**Bottom Right**) Légal Trap: 1.e4 e5 2. ♘f3 ♘c6 3. ♗c4 d6 4. ♘c3 ♗g4 5. ♘xe5, to be followed by ... ♗xd1 6. ♗xf7+ ♔e7 7. ♘d5#.

Table 2. Trap states with proportions of false negatives (trap states with lower-than-average similarity) and false positives (non-trap states with higher than average similarity).

Position	Similarity Measure	False Negatives	False Positives
Budapest	Continuations	0.252	0.493
Caro–Kan	Continuations	0.324	0.437
Kieninger	Continuations	0.185	0.393
Légal	Continuations	0.332	0.461
Budapest	Sequences	0.234	0.480
Caro–Kann	Sequences	0.335	0.397
Kieninger	Sequences	0.276	0.424
Légal	Sequences	0.330	0.457
Budapest	Correlation	0.195	0.723
Caro–Kann	Correlation	0.029	0.630
Kieninger	Correlation	0.865	0.380
Légal	Correlation	0.207	0.713

In general, there was no significant difference between the proportion of false positives (non-traps with above average similarity) and false negatives (traps with below average similarity) given by the similarity of sequences, similarity of continuations and tree-edit similarity. However, the added time complexity of the similarity of sequences and tree-edit similarity at depth 2 was significant. Thus, perhaps surprisingly, the similarity of continuations is effectively better as a heuristic similarity measure for evaluating similarities of closely related board positions than the similarity of sequences.

Finally, for complexity considerations, we tested the structural similarity measure on five smaller samples of 40 randomly selected child positions from the first two trap positions. Using this measure, an average of 85% of child trap states had above average

structural similarity to the original position. The high complexity of this measure makes it time intensive to compute, but results clearly show it is rather effective at picking out potential trap states from a select sample of positions.

Move Matching. The move-matching algorithm was also tested on various chess positions to detect moves with similar strategic impact. Frequent matchings were assumed to be a more reliable indicator of moves with a similar effect on gameplay, so only the top five most frequently matched pairings were assessed.

We tested the matching algorithm on three different samples, each with six pairs of board positions, all shown in Table 3. Firstly, we used the algorithm on all traps from the trap-detection sample. For all but one of the pairings (Légal and Budapest traps), all of the five most frequent matches for each pair comprised two decisive or two non-decisive moves. In all but one pairing (Caro–Kann and Kieninger traps), the two most frequently paired moves were both checkmate moves. The second sample we used was based on the Légal and Budapest Gambit traps. We compared each trap with a sample of three child positions. This sample comprised one position containing the original trap but a difference in the placement of two pawns; one position where the bishop that had threatened the queen had been captured; and one position that was selected as the best continuation by the Stockfish chess engine. In all but one pairing, all of the top five matches comprised two decisive or two non-decisive moves. All of the most frequently paired moves were both decisive. The third sample was a selection of positions from the 2016 World Championship match between Magnus Carlsen and Sergey Karjakin, which appeared after 10, 20, 30, and 40 plies. An average of four of the top five matches for each pairing comprised two decisive or two non-decisive moves. Three of the most frequently paired moves were both check moves, and one of them comprised two equivalently unimpactful moves of the king. This sample provided less reliable pairings than the previous two samples, possibly because its positions had a more varied strategic impact than those of the other samples.

These results show that the move-matching algorithm is fairly well suited to finding similarly decisive moves from different board positions, and thus is useful for detecting possible trap states and sacrificial moves from the game-tree structure without evaluating board positions.

Table 3. Pairs of board positions, the number of equally decisive matches in their 5 most frequent move matches, and their top match.

Board Position Pairings	Equally Decisive Top 5 Pairs	Top Match
Légal, Budapest	4	♗f7#, ♗f2#
Légal, Kieninger	5	♗f7#, ♘f3#
Légal, Caro–Kann	5	♗f7#, ef7#
Budapest, Kieninger	5	♗f2#, ♘d3#
Budapes, Caro–Kann	5	♗f2#, ef7#
Caro–Kan, Kieninger	5	ef7#, ♘f3+
Budapest, Budapest + 5. b3 a6	5	♗f2#, ♗f2#
Budapest, Budapest + 5. f3 ♘xe5	3	♗f2#, ♘f3+
Budapest, Budapest + 5. ♗d2 ♕h4	5	♗f2#, ♗d2+
Légal, Légal + ... h6 6. a3	5	♗f7#, ♗f7#
Légal, Légal + ... h6 6. ♘xg4	5	♗f7#, ♗f7+
Légal, Légal + ... ♘xe5 6. ♗e2	5	♗f7#, ♗b5+
Carlsen–Karjakin Move 10, Move 20	5	♗e6, ♕e8
Carlsen–Karjakin Move 10, Move 30	5	♕c3+, ♖g5+
Carlsen–Karjakin Move 10, Move 40	5	♕c3+, ♖g5+
Carlsen–Karjakin Move 20, Move 30	4	♘d4, ♖g5+
Carlsen–Karjakin Move 20, Move 40	3	♔f8, ♔h8
Carlsen–Karjakin Move 30, Move 40	3	♖g5+, ♕g6+

5. Applications

5.1. Amaf/Rave Adaptation

Past papers [28] have shown that MCTS displays a marked improvement when using adaptations such as All-Moves-As-First (AMAF), Rapid Action Value Estimation (RAVE), and Permutation-AMAF. Such adaptations update multiple areas of the game tree at once, where one move is available from many positions (as in AMAF) or where one board position is a permutation of another, on the assumption that the equivalent move from each of these positions will have the same strategic impact on gameplay. We envisage the effective use of a similarity measure when choosing which equivalent positions to update, as this may lead to more effective trap detection than that of MCTS or its AMAF adaptations. We suggest adding a similarity measure to two MCTS adaptations: the killer heuristic, where decisive moves are evaluated first, and killer RAVE, which only applies RAVE to decisive moves [29]. MCTS may more quickly detect a trap ahead when combined with these similarity-based adaptations.

5.2. Wider Game Strategy and Graph Applications

Many modern AI programs use deep learning to recognise tactical patterns from shapes of features in the field of play. It seems natural to use this learning strategy to group atomic moves by their tactical value, to then create an abstracted game-tree with a lower branching factor than the original tree. The structural similarity measure can then be used to detect tactical moves representing equivalent strategies, giving the agent options once it has chosen its desired strategy.

In cases where an agent is trained to predict the moves a human player would make, as was the case for AlphaGo [5], the modified AMAF/RAVE adaptation above can be used to prime the neural network and update predictions for multiple positions at once. This may lead to opportunities for faster reinforcement learning or more efficient learning from smaller data sets.

6. Conclusions

We presented four similarity measures for game positions in two-player, deterministic games of perfect information, based on their game trees with no domain-specific knowledge. We tested the measures on chess and suggested their use in heuristics for MCTS-based agents, noting their application to a range of graphical problems. We showed that, using our first two similarity measures, an average of around 70% of chess positions occurring two plies after a trap state that were also traps had above-average similarity to the original position. This figure rose to 85% using the structural similarity measure. We also showed that our move-matching algorithm consistently paired moves with similar strategic value from different starting positions. We believe this can aid MCTS agents in finding equally decisive moves within different areas of the game tree, as well as in detecting new trap states.

Author Contributions: Conceptualization, S.E. and P.T.; methodology, S.E. and P.T.; software, S.E.; formal analysis, S.E.; writing—original draft preparation, S.E.; writing—review and editing, P.T.; visualization, S.E. and P.T.; supervision, P.T.; project administration, P.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Karpov, A.; Phelizon, J.F.; Kouatly, B. *Chess and the Art of Negotiation: Ancient Rules for Modern Combat*; Praeger: Westport, CT, USA, 2006.
2. Rubinstein, A. *Modeling Bounded Rationality*; MIT Press Books; The MIT Press: Cambridge, MA, USA, 1997.

3. Maschler, M.; Solan, E.; Zamir, S. *Game Theory*; Cambridge University Press: Cambridge, UK, 2016.
4. Schwalbe, U.; Walker, P. Zermelo and the Early History of Game Theory. *Games Econ. Behav.* **2001**, *34*, 123–137.
5. Silver, D.; Huang, A.; Maddison, C.J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. Mastering the game of Go with deep neural networks and tree search. *Nature* **2016**, *529*, 484. [PubMed]
6. Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* **2018**, *362*, 1140–1144. [CrossRef] [PubMed]
7. Browne, C.B.; Powley, E.; Whitehouse, D.; Lucas, S.M.; Cowling, P.I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; Colton, S. A survey of monte carlo tree search methods. *IEEE Trans. Comput. Intell. AI Games* **2012**, *4*, 1–43. [CrossRef]
8. Russell, S.; Norvig, P. *Artificial Intelligence: A Modern Approach*, 3rd ed.; Pearson (Intl): London, UK, 2016.
9. Sadler, M.; Regan, N. *Game Changer: AlphaZero's Groundbreaking Chess Strategies and the Promise of AI*; New in Chess: Alkmaar, The Netherlands, 2019.
10. Ramanujan, R.; Sabharwal, A.; Selman, B. On Adversarial Search Spaces and Sampling-Based Planning. In Proceedings of the Twentieth International Conference on Automated Planning and Scheduling, Toronto, ON, Canada, 12–16 May 2010; Volume 10, pp. 242–245.
11. Babai, L. Graph Isomorphism in Quasipolynomial Time. In Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing, Cambridge, MA, USA, 19–21 June 2016.
12. Zhang, K.; Shasha, D. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.* **1989**, *18*, 1245–1262. [CrossRef]
13. Jiang, T.; Wang, L.; Zhang, K. Alignment of trees—An alternative to tree edit. *Theor. Comput. Sci.* **1995**, *143*, 137–148. [CrossRef]
14. Koch, G.; Zemel, R.; Salakhutdinov, R. Siamese neural networks for one-shot image recognition. In Proceedings of the ICML Deep Learning Workshop, Lille, France, 6–11 July 2015; Volume 2.
15. Sutton, R.S.; Barto, A.G. *Reinforcement Learning—An Introduction*, 2nd ed.; MIT Press: Cambridge, MA, USA, 2018.
16. Swamidass, S.J.; Chen, J.; Bruand, J.; Phung, P.; Ralaivola, L.; Baldi, P. Kernels for small molecules and the prediction of mutagenicity, toxicity and anti-cancer activity. *Bioinformatics* **2005**, *21*, i359–i368. [CrossRef] [PubMed]
17. Bajusz, D.; Rácz, A.; Héberger, K. Why is Tanimoto index an appropriate choice for fingerprint-based similarity calculations? *J. Cheminform.* **2015**, *7*, 20. [CrossRef] [PubMed]
18. Ukkonen, E. On-line construction of suffix trees. *Algorithmica* **1995**, *14*, 249–260. [CrossRef]
19. Weiner, P. Linear pattern matching algorithms. In Proceedings of the 14th Annual Symposium on Switching and Automata Theory, Iowa City, IA, USA, 15–17 October 1973; pp. 1–11.
20. Vishwanathan, S.V.N.; Schraudolph, N.N.; Kondor, R.; Borgwardt, K.M. Graph kernels. *J. Mach. Learn. Res.* **2010**, *11*, 1201–1242.
21. Li, Y.; Zhang, C. A metric normalization of tree edit distance. *Front. Comput. Sci. China* **2011**, *5*, 119–125. [CrossRef]
22. Touzet, H. Tree edit distance with gaps. *Inf. Process. Lett.* **2003**, *85*, 123–129. [CrossRef]
23. Gelly, S.; Silver, D. Monte-Carlo tree search and rapid action value estimation in computer Go. *Artif. Intell.* **2011**, *175*, 1856–1875. [CrossRef]
24. Cazenave, T. A phantom-go program. In *Advances in Computer Games*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 120–125.
25. Teytaud, F.; Teytaud, O. Creating an upper-confidence-tree program for Havannah. In *Advances in Computer Games*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 65–74.
26. Akiyama, H.; Komiya, K.; Kotani, Y. Nested Monte-Carlo search with AMAF heuristic. In Proceedings of the 2010 International Conference on Technologies and Applications of Artificial Intelligence, Hsinchu City, Taiwan, 18–20 November 2010.
27. Lorentz, R.J. *Amazons Discover Monte-Carlo*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 13–24.
28. Helmbold, D.P.; Parker-Wood, A. All-Moves-As-First Heuristics in Monte-Carlo Go. In Proceedings of the IC-AI, Las Vegas, NV, USA, 13–16 July 2009; pp. 605–610.
29. Lorentz, R.J. Improving monte–carlo tree search in havannah. In *International Conference on Computers and Games*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 105–115.