JOB-LEVEL ALGORITHMS FOR CONNECT6 OPENING BOOK CONSTRUCTION

Ting han Wei[1]*, I-Chen Wu*[1]*, Chao-Chin Liang*[1]*, Bing-Tsung Chiang*[1]*, Wen-Jie Tseng*[1]*, Shi-Jim Yen*[2]*, and Chang-Shing Lee*[3]

Hsinchu, Hualien, and Tainan / Taiwan

ABSTRACT

This article investigates the use of job-level (JL) algorithms for Connect6 opening book construction. In the past, JL proof-number search (JL-PNS) was successfully used to solve Connect6 positions. Using JL-PNS, many opening plays that lead to losses can be eliminated from consideration during the opening game. However, it is unclear how the information of unsolved positions can be exploited for opening book construction. For this issue, the current article first proposes four heuristic metrics when using JL-PNS to estimate move quality. The article then proposes a JL upper confidence tree (JL-UCT) algorithm and three heuristic metrics that work with JL-UCT. Of the three, the best way to estimate move quality for JL-UCT is the number of nodes in each candidate move's subtree. In order to compare the heuristic metrics among the two algorithms objectively, we proposed two kinds of measurement methods to analyze the suitability of these metrics when choosing best moves for a set of benchmark positions. Experimental results show that the node count heuristic metric for JL-UCT outperforms all other heuristic metrics, including the four for JL-PNS. We then verify the results by constructing three separate opening books using the top three performing heuristic metrics. Competitive play also shows that the node count heuristic metric for JL-UCT is most suitable among the currently proposed heuristic metrics for Connect6 opening book construction.

1. INTRODUCTION

In the field of computer games, an *opening book* refers to the technique of storing a pre-calculated database of, typically, the best moves to play and their corresponding evaluation values at the beginning of the game. Since computing time is often limited during competitions, the use of opening books allows a program to offload the time cost to any point before the competition, where time constraints are not an issue. In addition to saving time during play, opening books may also be constructed to provide a more accurate move evaluation. For these reasons, the construction of opening books is often critical in designing a strong game-playing program (Buro, 1999; Hyatt, 1999; Lincke, 2001).

While manual construction of opening books have shown success in the early days, recent efforts have mostly been focused on the automatic generation of opening books (Buro, 1999; Hyatt, 1999; Lincke, 2001; Audouard, Chaslot, Hoock *et al.*, 2009; Chaslot, Hoock, Pérez *et al.*, 2009; Gaudel, Hoock, Pérez *et al.*, 2011). The automatic generation of opening books is especially important to the game of Connect6, a relatively young game that was introduced in 2005 (Wu and Huang, 2006), since few expert game records are available for opening book generation. In the past, many search algorithms such as alpha-beta search (Knuth and Moore, 1975) and Monte Carlo tree search (MCTS) were applied to explore new opening moves automatically, as done in Awari (Lincke, 2001), Othello (Buro, 1999; Lincke, 2001), Amazons (Karapetyan and Lorentz, 2006; Kloetzer, 2011), and Go (Audouard, Chaslot, Hoock *et al.*, 2009; Baier and Winands, 2011).

Job-level (JL) computing was proposed by Wu et al. (2011a; 2013) to help solve positions by multiple simultaneous execution of game-playing programs as jobs. Based on JL computing, the *JL proof-number*

[1] Dept. of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu / Taiwan. Email: icwu@cs.nctu.edu.tw; tinghan.wei@gmail.com;
[2] Dept. of CS and Inf. Engineering, National Dong-Hwa University, 1, Scc, Da Hsueh Rd., Shou-Feng, Hualien 974 / Taiwan. Email: sjyen@csie.ndhu.edu.tw
[3] Department of Computer Science and Information Engineering, National University of Tainan, Tainan 700 / Taiwan. Email: leecs@mail.nutn.edu.tw

search (JL-PNS) algorithm was proposed, solving various Connect6 positions successfully with significant speedups. Saffidine, Jouandeau, and Cazenave (2012) also used JL-PNS to solve positions of Breakthrough. Chen, Wu, Tseng et al. (2014) proposed a JL alpha-beta search (JL-ABS) algorithm to help construct a Chinese chess opening book. Other opening book generation methods similar to JL methods include the meta MCTS method proposed by Chaslot, Hoock, Pérez et al. (2009), and the job queue by Schaeffer, Burch, Björnsson et al. (2007).

Using JL-PNS (Wu, Lin, Lin *et al.*, 2011a; Wu, Lin, Sun *et al.*, 2013), many opening plays that lead to losses can be eliminated from consideration. However, a major drawback to opening book generation using JL-PNS is that it does not yield a good estimate value of positions if the search is terminated before a solution can be obtained. Since many opening positions tend to be difficult to solve, a large amount of computation spent on expanding opening positions may be wasted, if the results from the computation are not used for other purposes. To make most of these computations useful, we attempt to use these results to help construct an opening book.

To utilize these results, four *heuristic metrics* are proposed to distinguish the best move from all possible candidate moves during JL-PNS analysis. However, none of these heuristic metrics can indicate move quality universally. The main reason is that PNS is designed to prove/solve positions, not to estimate the strengths of positions.

To solve these issues, we also propose a *JL upper confidence tree* (JL-UCT) algorithm. Opening positions are viewed as *multi-armed bandit problems* (Auer, Cesa-Bianchi and Fischer, 2002), where each possible move from a given position is treated as a choice. UCT has been successful in balancing between exploiting good moves and exploring new possibilities like the common MCTS (Browne, Powley, Whitehouse *et al.*, 2012). One of the heuristic metrics for JL-UCT simply chooses the move with the maximum number of nodes, as most MCTS methods do.

In order to compare these metrics objectively, we propose two kinds of measurement methods to analyze the suitability of these metrics for a set of benchmark positions. We also use the three best performing metrics to construct opening books, and conduct competitive play over a collection of 100 openings to determine which metric and algorithm results in the strongest opening book. The results show that the heuristic metric of node count using JL-UCT outperforms all the others in most cases.

The organization of this article is as follows. Section 2 presents related work. This includes a brief introduction to Connect6 and the game program NCTU6, JL-PNS, and other JL algorithms for opening position analysis. Section 3 outlines the JL-PNS heuristic metrics that are devised to pick out the best candidate move during opening position analysis by JL-PNS, while Section 4 describes the JL-UCT algorithm. Section 5 describes the experiments performed and gives a discussion of the experiment results. Lastly, we make concluding remarks in Section 6. An earlier version of this article was published in (Wei, Wu, Liang *et al.*, 2014). This article includes, in addition to the contents of the previous version, the description on opening book construction and the opening book competitive play experiments.

2. PREVIOUS WORK

Below we discuss three types of previous work. They are: Connect6 and NCTU6 (in Subsection 2.1), job-level proof-number search (in Subsection 2.2), and other modified or job-level-like methods for opening book generation (in Subsection 2.3).

2.1 Connect6 and NCTU6

Connect6 is a k-in-a-row game proposed by Wu and Huang (2006b). In this game, the first of two players, Black, starts the game by placing a single black stone on an empty square of a typical 19x19 Go board. Each subsequent move is then played by Black or White alternately using two stones of his color on empty squares, starting with White's response to Black's first move. The first player that is able to get six consecutive stones in a line (horizontally, vertically or diagonally) wins. For simplicity of discussion, we call consecutive stones live if they are not blocked at either end, or dead if they are blocked at exactly one end. A common strategy in Connect6 involves playing live-fours (L4), creating a so-called double threat where the opponent must block both ends of the pattern with two stones or lose immediately. Another example of a double threat involves playing two dead-fours (D4), where each D4 is a single threat. Winning by continuously forcing the opponent to block double threats is a common strategy.

NCTU6 is a Connect6 program that was developed by a team consisting of some authors of this article. It has won all Connect6 tournaments and man-machine championships (Wu and Yen, 2006a; Wu and Lin, 2008; Lin and Wu, 2009; Wu, Lin, Tsai *et al.*, 2011b; Wei, Tseng, Wu *et al.*, 2013). It consists of a solver component, which uses threat space search (Wu and Lin, 2010), and an alpha-beta search component (Wu, Tsai, Lin *et al.*, 2012). Two features of NCTU6 are particularly important in the scope of this article.

First, NCTU6 is able to verify victories involving continuous threats, or when such methods fail to find a solution, give an estimate of a position's strength based on the program's evaluation function. The resulting estimate is categorized into 13 distinct game statuses. A winning position for Black is categorized as "B:W"; "B4" indicates the game is extremely favourable for Black, while "B3", "B2", and "B1" indicate Black's advantage in decreasing order. The five game statuses from White's perspective are "W:W", "W4", "W3", "W2", and "W1", also in decreasing order from White winning to the position being slightly favourable for White. For positions where neither player has an advantage, there are three game statuses. "Stable" indicates that the evaluation values for subsequent moves are unlikely to fluctuate significantly. There are two unstable statuses, "unstable1" and "unstable2", where the evaluation fluctuations are greater for the latter. It is worth noting, however, that evaluation function fluctuations may also exist for one-sided advantage positions that are evaluated as B1/W1 through B4/W4 as well.

The second important feature of NCTU6 is that a set of prohibited moves can be given in addition to the position we wish NCTU6 to evaluate. NCTU6 will then calculate and suggest the best move to play that does not exist in the set of prohibited moves, given the input position. If NCTU6 cannot come up with a suggestion that does not exist in the prohibited set without losing, it will consider the position to be a loss with respect to the prohibited set. This feature is critical in applying NCTU6 to the job-level computing model, which we will describe in the next section.

2.2 Job-Level Proof-Number Search

This section reviews the JL-PNS algorithm. The overall JL computing model is briefly explained (in Subsection 2.2.1), followed by the generic *best-first job-level search* (BF-JL search) in Subsection 2.2.2. Proof-number search (PNS) and the process of applying it to BF-JL search are then summarized in Subsection 2.2.3.

2.2.1 Job-Level Computing Model

The job-level computing model starts by defining two parties: the *client*, whose role is to dynamically create tasks, and the *JL system*, the role of which is to complete these dynamically created tasks. The JL system is comprised of a collection of *workers*, who are responsible for the computation of jobs. When used in a search algorithm, for example, the client maintains a game tree and may choose a position (which corresponds to a node in the game tree) to encapsulate the move generation and evaluation of this position into a job. The system notifies the client when there are idle workers, at which time the client, who plays a passive role, submits jobs that are pending execution. The worker then evaluates this position and returns the result to the client via the JL system.

2.2.2 Best-First Job-Level Search

To apply the JL computing model to a generic best-first search algorithm, we must identify the common components that are shared. In the scope of computer games, a typical search operation consists of a game tree, for which each node represents a position, while the edges of the tree represent a move from one position to another. There are three common phases to search algorithms such as PNS or MCTS. These three phases are selection, execution, and update. The common data structures and behaviors can be consolidated into a JL framework to minimize BF-JL search development efforts. The design and framework abstraction is discussed in further detail in (Wei, Liang, Wu *et al.*, 2015).

2.2.3 Job-Level Proof-Number Search Algorithm

PNS is an algorithm that outperforms many variants of alpha-beta search when solving game trees (Allis, Van der Meulen and Van den Herik, 1994). This is made possible by utilizing the proof-number (PN) and disproof-number (DN) of each explored game tree node. For an arbitrary node n, its PN/DN counts the minimum number of child nodes that must be expanded before n can be solved as a winning/losing position. To be more specific: for an AND node, (1) its PN is the sum of its children's PN, since each child needs to be solved for the AND node to be considered proven, and (2) its DN is the minimum of its children's DN, since from the opponent's point of view, the AND node is a losing position as soon as one of its children can be solved as a loss. For an OR node, (3) the PN is the minimum of its children's PN, and (4) the DN is the sum of its children's DN.

During the selection phase of PNS, the node that contributes the most to solving the root node of the game tree is chosen. At an OR node, the child with the smallest PN is chosen. Conversely, at an AND node, the child with the smallest DN is chosen. This process is repeated until we arrive at a leaf node, which is called the most proving node (MPN). The MPN is then expanded through evaluation, and its corresponding PN/DN is then updated along its path to the root node.

To apply PNS to the BF-JL search for Connect6, we use the PNS algorithm for the selection and update phases, while the execution phase is encapsulated as jobs by the client. The workers in this case execute multiple simultaneous instances of NCTU6, each ready to evaluate jobs. As mentioned earlier in the description of NCTU6, the game tree is gradually expanded by supplying each worker with two pieces of information for every job: (a) the position that needs to be examined, and (b) a set of prohibited moves that cannot be returned. When a result is returned to the client, it adds the new node to the game tree and to the set of prohibited moves in the current level so that existing nodes will not be repeatedly added.

Domain knowledge can be used to accelerate the overall BF-JL search process by initializing the PN/DN of each node based on game statuses given by NCTU6. For example, the PN/DN of a position with B4 are set to 1/18, those for B3 are 2/12, etc. The details of the settings are in (Wu, Lin, Lin *et al.*, 2011a).

A second detail that is critical to the success of JL-PNS is avoiding the selection of the same MPN multiple times before its result is returned. To solve this problem, an extra phase is added to the BF-JL search algorithm called the pre-update phase. The pre-update phase is placed after the selection phase but before the execution phase, such that the selected node can be flagged to avoid being chosen multiple times.

2.3 Other Job-Level or Job-Level-Like Methods for Opening Book Generation

In (Chen, Wu, Tseng *et al.*, 2014), a JL-ABS method was used to help construct a Chinese chess opening book, where the emphasis was on avoiding weak spots when dropping out of an opening book. The idea is that out-of-book positions should still be evaluated as good with respect to the game-playing program.

Chaslot, Hoock, Pérez et al. (2009) proposed the meta Monte Carlo tree search method in which a two-tiered MCTS is performed to automatically generate an opening book for the game of Go. The typical MCTS simulation phase uses a fast routine that follows simple policies, putting very little emphasis on playing strength. The meta MCTS method replaces this simulation policy by a full game-playing program, which also uses MCTS but in the typical fashion. The upper level is the first tier of the overall algorithm that selects nodes that are worthy of expansion, while MoGo was used for the lower level. This method has been applied with success to 7x7 Go position analysis (Chou, Chou, Doghmen *et al.*, 2012).

3. HEURISTIC METRICS FOR JOB-LEVEL PROOF-NUMBER SEARCH

To apply JL-PNS to opening book generation, we must first attempt to devise a method of distinguishing good candidate moves from bad ones. A heuristic metric quantifies move quality. This allows us to choose a move to play for any positions that were searched but not solved by ranking candidate moves according to each heuristic metric. During the construction of the opening book, a heuristic metric is chosen in advance. The highest-ranked moves according to the heuristic metric are then saved. Four heuristic metrics that are closely related to the principles of PNS were used to generate the NCTU6 opening book. They are: Node Count (Subsection 3.1), Proof-Number/Disproof-Number (Subsection 3.2), Minimax Evaluation Value (Subsection 3.3), and the Hybrid Metric (Subsection 3.4).

3.1 Node Count

PNS is designed to favor exploring moves that allow it to solve a position using the least number of explorations, instead of the strongest moves to play. However, we observe that an MPN as well as its ancestors, which all lie on the most proving path (MPP), still tend to be strong moves. Thus, it is likely that the node that is more often included in the MPP tends to be stronger than its fewer included siblings. Nodes that are more often included in the MPP will have a larger number of nodes in their subtrees, so we may say that nodes with a higher node count are more likely to be stronger moves.

However, there is no guarantee that the node with the largest node count is the best move to play, so the node count metric alone should not be used as a definite sign of a good move. However, it is an important metric to consider when other metrics are used together.

3.2 Proof-Number/Disproof-Number Ratio

The PN and DN are critical to PNS in that they allow the algorithm to locate nodes which contribute most to solving a position. As explained in the PNS review earlier, the PN/DN for a specific node n is the minimum number of child nodes that need to be evaluated in order to prove that n is a winning/losing position. Therefore, we may deduce that a lower PN means that n is likely to be favourable, since it is closer to winning than another node which has a high PN. Similarly, a lower DN is likely to be unfavourable, since n is closer to losing.

To use this as a heuristic metric, we must keep in mind that both PN and DN for a node need to be considered. To do this, we consider the ratio between the PN and DN of a node as a valid heuristic metric. In practice, the PN is divided by the DN, and nodes with the lowest ratio are chosen as the best move when constructing the opening book.

Domain knowledge from NCTU6 is used in the form of PN/DN initialization during JL-PNS. With this in mind, the PN/DN ratio is a mostly adequate metric. However, there are two drawbacks with this metric. First, the actual move quality is highly dependent on the node count metric. In many cases, PN/DN values while the number of child nodes is still relatively small do not indicate move quality definitively. In other words, the PN/DN ratio cannot be used as an indication of move quality with confidence if the node count is not sufficiently large for the node. Second, minimax evaluations are not considered when using the PN/DN ratio metric. Situations may also arise where nodes with similar PN/DN ratios but distinctly different evaluation values are treated similarly when they should not be.

3.3 Minimax Evaluation Value

The minimax value of each internal node, which is computed by NCTU6, can be used to give a rough estimate of the strength of each position. This metric, however, is more of an intermediate one that can be used by another metric rather than a practical one on its own. It is often worse than the other metrics since Connect6 can be a highly unstable game. As described in Subsection 2.1, game statuses may vary rapidly, so the minimax evaluation plays a largely variable role in the game's outcome.

3.4 Hybrid Metric

The hybrid method combines the PN/DN ratio with NCTU6's game status estimation to form a heuristic metric. Upon receiving evaluation results, the client stores the game status of the newly expanded node, then updates this information upwards to the root node of the game tree in a minimax fashion. When choosing the best move to play, the candidate move with the best game status is chosen. Since there are only 13 distinct game statuses, it is not uncommon to see several candidate moves share the best game status. The minimum PN/DN ratio is then used as the tie-breaker when determining the best move to play.

While this metric is more accurate than the PN/DN metric, it also depends highly on the node count. The hybrid metric score may still be untrustworthy if the number of nodes in the subtree is insufficient.

4. JOB-LEVEL UPPER CONFIDENCE TREE

In this section, we propose the JL-UCT algorithm in Subsection 4.1. Next, the upper confidence bound (UCB) function that is used to balance exploration and exploitation is provided for discussion in Subsection 4.2. We discuss the pre-update policy that is used for JL-UCT in Subsection 4.3. Lastly, we list the three heuristic metrics that may be used with JL-UCT book generation in Subsection 4.3.

4.1 Algorithm Description

There are two tiers of the JL-UCT search. The upper tier is similar to MCTS, where the selection phase chooses the node that has the highest score according to the UCB function, maintaining a UCT as the algorithm continues the search. This upper tier is managed by the JL client. The lower tier consists of the execution phase of the generic job-level algorithm. An existing game-playing program, in this case NCTU6, is used as a worker that evaluates and suggests the best possible move for the selected node from the upper tier after excluding any prohibited moves. A pre-update phase is added to avoid repeatedly selecting the same node as a job (see Subsection 4.3). Once a node is evaluated by a worker, the best move to play for that position and its corresponding win rate are returned to the client. The win rate is then used to update the UCT from the leaf node to the root node by increasing the visit count of the evaluated node by 1, and recalculating the aggregate win rate of the evaluated node and its ancestors. This process is continued until the root position is solved, or if the pre-defined number of jobs has been completed.

4.2 Upper Confidence Bound Function

For the selection phase of the UCT, we chose the commonly used UCB1 function (Browne, Powley, Whitehouse *et al.*, 2012).

$$WR + C\sqrt{\frac{log(N_p)}{N}}$$

where WR is the win rate for the Black player of the node, C is a pre-defined constant, N_p is the parent visit count, and N is the visit count of the node itself. Similar to JL-PNS, the parameters that are used in the UCB function need to be initialized using domain knowledge from NCTU6. The game statuses returned by NCTU6 are converted into the following win rate initializations.

Status	B:W	B4	B3	B2	B1	Stable	Unstable 1	Unstable 2	W1	W2	W3	W4	W:W
Win Rate (%)	100	90	80	70	60	50	50	50	40	30	20	10	0

Table 1: JL-UCT win rate initialization.

4.3 Pre-update Phase

To ensure that the JL algorithm does not choose the same node multiple times during the selection phase, we used the virtual loss policy (Wu, Lin, Lin *et al.*, 2011a). That is, once a node is selected for expansion, its win rate value is temporarily set to 0% if the move associated with the node is one played by Black, or 100% if it is played by White. By setting the virtual value of the node to a loss, JL-UCT is guaranteed to avoid choosing the same node in subsequent selection phases. Once the job results have been received, the node win rate value is then updated according to the results, removing the virtual loss and allowing the node to be chosen again.

4.4 Heuristic Metrics for JL-UCT

Below, we list the three heuristic metrics for JL-UCT: Node Count (Subsection 4.4.1), Win Rate (Subsection 4.4.2), and Upper Confidence Bound Value (Subsection 4.4.3).

4.4.1 Node Count

Similar to the reasoning given in the description for JL-PNS, the number of nodes that belong to a subtree is an intuitive indicator of the move strength that is associated with the root of the subtree. While JL-UCT prefers to devote resources to nodes that have a higher win rate, this is subtly different from JL-PNS where the node that contributes the most to proving the game is given higher priority.

4.4.2 Win Rate

Different from typical MCTS, JL-UCT does not contain a simulation phase where a simple program plays according to a preset policy until game resolution. Therefore the win rates that are used here depend mostly on the initialization values as described in 4.4.1. From this perspective, this metric suffers from the same drawback as the metrics in JL-PNS, where values may not be trustworthy if the node count is insufficient.

4.4.3 Upper Confidence Bound Value

This metric is the one that the JL-UCT algorithm uses during the selection phase to choose the node that is most worthy of expansion. Since the UCB1 function tries to balance exploration with exploitation, this metric will tend to try different options even when they may be weak choices. Therefore this metric exists more as a conceptual point of observation rather than a practical indicator during opening book generation.

5. EXPERIMENTS AND DISCUSSION

We conduct three types of experiments to determine the best way to construct Connect6 opening books. For the first experiment, which is described in Subsection 5.1, we solve a set of benchmark positions using the two algorithms JL-PNS and JL-UCT. In the second experiment, which is described in Subsection 5.2, we devise two methods to analyze the effectiveness of each heuristic metric used by the two algorithms. For the third experiment, which we describe in Subsection 5.3, the three top performing heuristic metrics were used to construct three separate opening books. Competitive play between the same program using differing opening books was performed on a set of 100 openings, which are different from the benchmark positions used in the first experiment.

5.1 Solving Game Positions

In the first experiment, we compare the performance between JL-PNS and JL-UCT by using them to solve a set of 22 benchmark game positions, where each can be solved as winning for the first player to move. The benchmark was chosen from the 35 Connect6 positions provided in (Wu, Lin, Sun *et al.*, 2013) in the following way. Among these positions, 15 can be solved as winning for the first player to move and 20 as losing. We chose the 15 winning positions and created 7 more winning positions out of the 20 losing ones by using their parent positions as the root node instead. We omit the 13 remaining losing positions because their solution trees are too large and therefore too time-consuming.

The experiment was designed this way because we are interested in comparing how efficiently an algorithm is able to converge on a single winning move. With a win position for the first player, only one winning move needs to be found for the proof. If we were to choose losing positions for the first player to play instead, all child moves will need to be solved as losses for the first player to play. This is advantageous to JL-PNS since there is no urgency to locate the winning moves, as all moves will eventually be proven as losses, and PNS is inherently superior when it comes to solving game positions. We did not choose unsolved positions for the benchmark because it is difficult to devise an objective metric when the winning move is not known.

The experiment was conducted on an 8-core grid consisting of Intel Pentium E2180s. For clarity of discussion, we now define r as the root position of a benchmark, $c_1, c_2, c_3, \dots, c_k$ as child moves of r, where k is the total number of possible candidate moves from r. As described above, for each benchmark, r is solved as a win for the first player. This implies that some child move, denoted by c_w, can be proven as a win. It is worth mentioning that for some benchmarks there exists more than one c_w. Let n^A denote the total number of nodes required to solve r, using algorithm A.

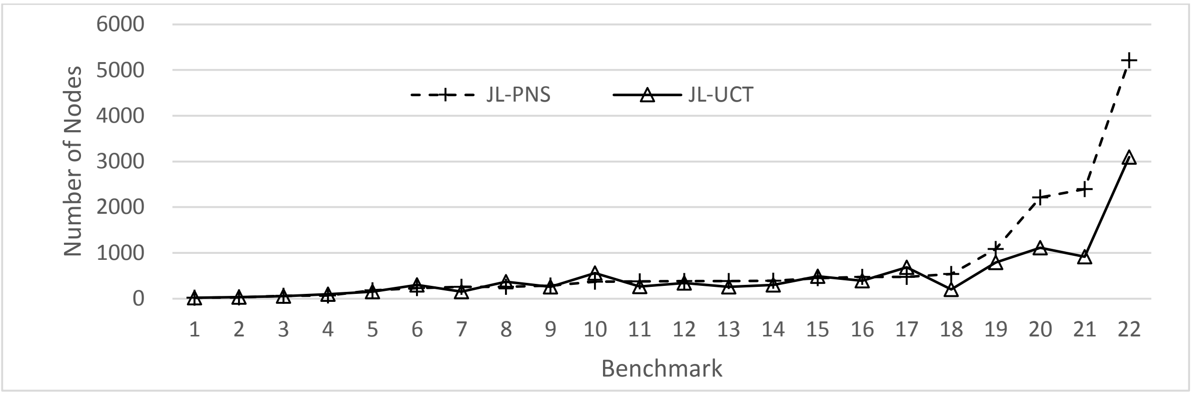

Fig. 1: Number of nodes required to solve each benchmark.

	Average nodes	Median number of nodes	Number of positions in the benchmark where the method requires less nodes	Average ratio of nodes $\left(\frac{\text{JL-PNS}}{JL-UCT}\right)$
JL-PNS	732	377	8	1.286
JL-UCT	491	297	14	

Table 2: Comparison between JL-PNS and JL-UCT.

The experiment results for n^{PNS} and n^{UCT} are as shown in Fig. 1. The benchmarks were numbered according to the increasing order of n^{PNS}. While PNS is intuitively more suitable for solving positions, we can see that JL-UCT does not perform worse than JL-PNS. In some cases, JL-UCT can even find solutions significantly faster than JL-PNS. From the results, JL-UCT tends to guide its search towards c_w much earlier and spends less resources verifying the non-winning child moves than JL-PNS in OR trees (winning positions), thereby saving precious computational resources.

5.2 Measuring the Quality of Various Heuristic Metrics

We now attempt to measure the quality of various heuristic metrics that are used by JL-PNS and JL-UCT. We introduce two ways of measuring heuristic metric quality, which we will refer to as the ε measuring method (Subsection 5.2.1) and the θ measuring method (Subsection 5.2.2). Namely, heuristic metrics are used to pick the best move to play when a solution cannot be found, while ε and θ are used to measure the quality of heuristic metrics.

We define m_i^π as the best move that is chosen by the metric π when the ith job result has been received and its corresponding node has been added in the update phase. That is, m_i^π is the best chosen move by π, after only i jobs have been completed in the JL system. For example, when we are using JL-PNS, m_1^{PNDN} is the best move to play from r according to the PN/DN metric after exactly 1 job has been completed. Using our notation, we can see that $m_N^{PNDN} = c_w$, where $N = n^{PNS}$.

5.2.1 The ε Measuring Method

To evaluate the quality of each heuristic metric, we then recorded $\{m_i^{PNS-N}, m_i^{PNDN}, m_i^{Minimax}, m_i^{Hybrid}\}_{i=1,2,\dots,n^{PNS}}$ and $\{m_i^{UCT-N}, m_i^{Winrate}, m_i^{UCB}\}_{i=1,2,\dots,n^{UCT}}$ for all benchmarks. Since all benchmarks are solvable, the winning child move c_w for each benchmark is also known. For any value of i such that $1 \le i \le n^A$, we say that the metric π will pick the correct move if $m_i^\pi = c_w$. We can then express the last job for which a metric π is unable to pick the correct move as:

$$\varepsilon^\pi = \arg\max_i (m_i^\pi \ne c_w).$$

In other words, the specified heuristic metric converged on the correct move c_w after ε^π jobs. Therefore, the smaller ε^π is, the better the metric is for the following reason. Assume that a position can be solved eventually with n^A nodes, where n^A is unknown (and quite possibly very large). Then, it is likely that the JL search may stop before r is solved. Between two algorithms, the algorithm with a smaller ε^π is more likely to pick the correct move c_w, even if we do not know what c_w or n^A are, since it converges earlier.

For example, in an extreme case, if for a certain benchmark, ε^{UCT-N} has a value of 0 using the JL-UCT node count heuristic metric, we know that at any given time the subtree of c_w is always the biggest among all other candidate move subtrees. Consequently, if we stop the JL-UCT algorithm at any time, we will be able to pick the correct move to play if we decide to use the node count heuristic metric. Of course, we will not know for certain the move to play is the correct one unless the benchmark is completely solved, but we may conjecture that an algorithm with smaller values of ε^{π} can find the correct move more often than an algorithm with a larger value.

The results for JL-PNS, JL-UCT, and the best performing metrics for both algorithms are shown in Fig. 2, Fig. 3 and Fig. 4. Since we are interested in examining the ratio between two heuristic metrics, for example ε^{PNS-N} and ε^{PNDN}, and also since ε^{π} values may vary from less than 10 to well over 1000, we use the base 10 logarithmic scale for all figures in this article. All values of ε^{π} are added by 1 so that we may analyze the experiment data with division and the logarithm function.

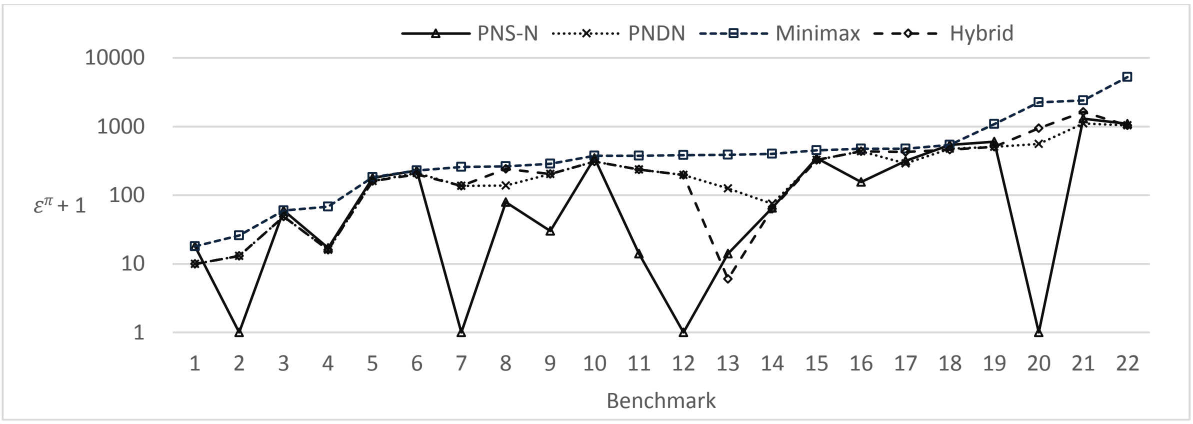

Fig. 2: JL-PNS metric comparison using last incorrect job ε^{π}. All values of ε^{π} are added by 1 so that the data may be plotted on a logarithmic scale.

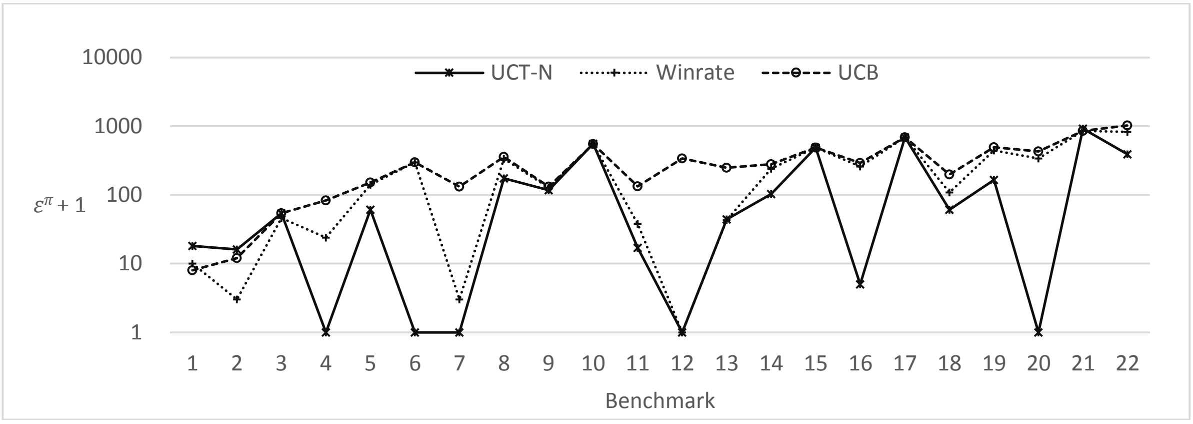

Fig. 3: JL-UCT metric comparison using last incorrect job ε^{π}. All values of ε^{π} are added by 1 so that the data may be plotted on a logarithmic scale.

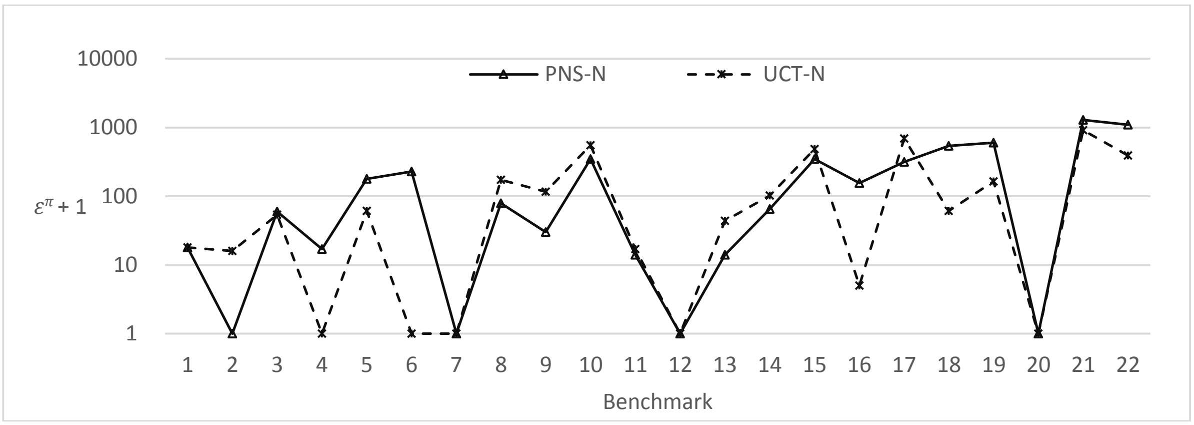

Fig. 4: Best performing metrics PNS-N and UCT-N comparison using last incorrect job ε^{π}. All values of ε^{π} are added by 1 so that the data may be plotted on a logarithmic scale.

We examine the various JL-PNS heuristic metrics in Fig. 2, the JL-UCT heuristic metrics in Fig. 3, and the best performing heuristic metric from the two methods above are compared in Fig. 4. For JL-PNS, the node count metric outperformed the hybrid method in nearly all benchmarks. While the PN/DN ratio is much better than the minimax evaluation metric, it does not perform as well as the hybrid metric.

From the JL-UCT results in Fig. 3, we can see that the upper confidence bound value is the worst metric. This is reasonable, since the upper confidence bound value is designed so that it balances between exploration and exploitation, therefore even poor choices may be indicated as the most suitable move every once in a while. The win rate metric performs reasonably well, but is overall inferior to the node count metric. For many benchmarks, the node count metric is able to stay fixed on the correct move from the very beginning.

The best metric from JL-PNS is compared with JL-UCT in Fig. 4. While JL-UCT performs much better for many benchmarks, it sometimes appears to perform worse than JL-PNS, such as in benchmark 2, 8, 9, 10, and 17. Further investigation indicated that while the node count metric seemed to focus on the moves other than the correct move for most of the game in these benchmarks, these seemingly incorrect moves can in fact be solved as well. This fits our intuition that JL-UCT seems to put more emphasis on locating strong moves, which then often lead to solving the position. This is in strong contrast to JL-PNS, which puts solving positions first and foremost for all occasions.

The total sum of ε^{PNS-N} for all benchmarks is 5394, while the total sum of ε^{UCT-N} is 3855. The base-10 logarithm of the ratio between these two metrics, $\frac{\varepsilon^{PNS-N}+1}{\varepsilon^{UCT-N}+1}$, is summed for all benchmarks, yielding a result of 4.103. For two identical metrics, this logarithmic sum should have a value of 0. Therefore we can see that ε^{UCT-N} is the superior metric.

5.2.2 The θ Measuring Method

We now define a second way of measuring the quality of heuristic metrics. While ε^{π} is concerned with the last job for which a metric π makes an incorrect choice, we are also interested in the total number of times the metric will lead to an incorrect choice. We define the number of times a metric π chooses incorrect moves for a solved position as follows:

$$\theta^{\pi} = |\{m_i^{\pi} | m_i^{\pi} \neq c_w\}|$$

As a different way of measuring heuristic metric performance, θ^{π} is interpreted differently from ε^{π} in the following aspect. Consider an extreme case where $\varepsilon^{\pi} = n^A - 1$ and $\theta^{\pi} = 1$. From ε^{π}, the metric π is poor since we will surely pick c_w after n^A jobs are completed. In contrast, from θ^{π}, the metric is good since we fail to pick c_w only once, at the time when $n^A - 1$ jobs are completed. However, in general, if ε^{π} is small like 1, it implies that θ^{π} is small too and that the heuristic metric is superior; and if θ^{π} is high like $n^A - 1$, it implies that ε^{π} is high too and that the heuristic metric is inferior.

Fig. 5, Fig. 6 and Fig. 7 show the comparison between the algorithms and metrics using θ^{π}. The gap between the hybrid metric and the PNS-N metric is not as large as when comparing by ε^{π}. In fact, the θ^{Hybrid} is slightly smaller than θ^{PNS-N} when their values are close, but the PNS-N metric makes up for this disadvantage for benchmarks 2, 7, 12 and 20, where the ratio between θ^{Hybrid} and θ^{PNS-N} is more than 100-fold. Meanwhile, the gap between θ^{UCT-N} and $\theta^{Winrate}$ is also smaller than when comparing by ε^{π}, but it is clear that UCT-N is still the superior metric. For benchmarks 2, 8, 9, 17, UCT-N performs worse than PNS-N, similar to Fig. 4.

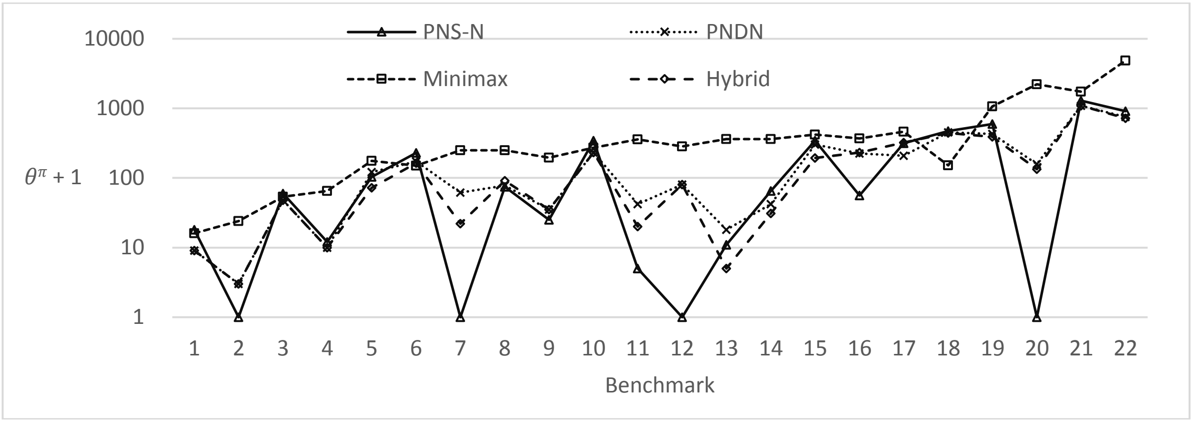

Fig. 5: JL-PNS metric comparison using number of incorrect jobs θ^{π}. All values of θ^{π} are added by 1 so that the data may be plotted on a logarithmic scale.

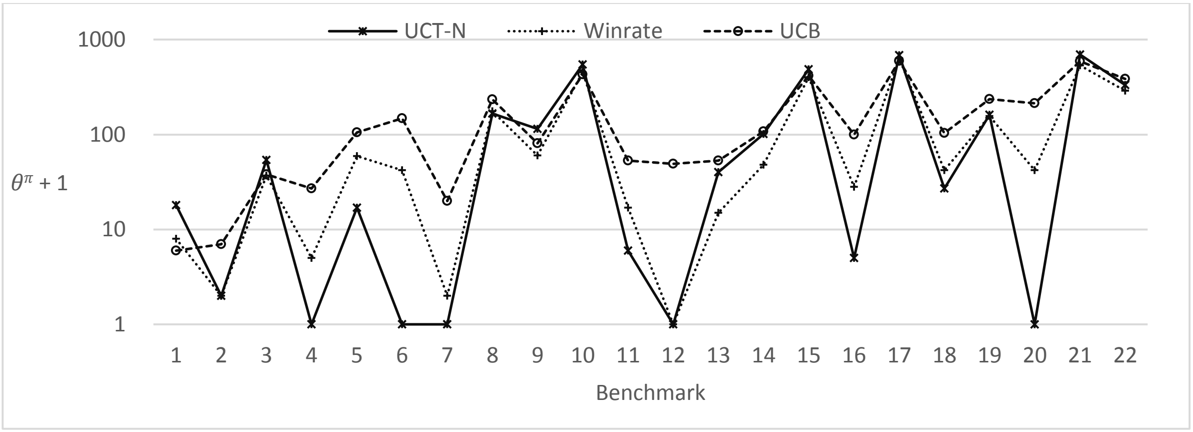

Fig. 6: JL-UCT metric comparison using number of incorrect jobs θ^{π}. All values of θ^{π} are added by 1 so that the data may be plotted on a logarithmic scale.

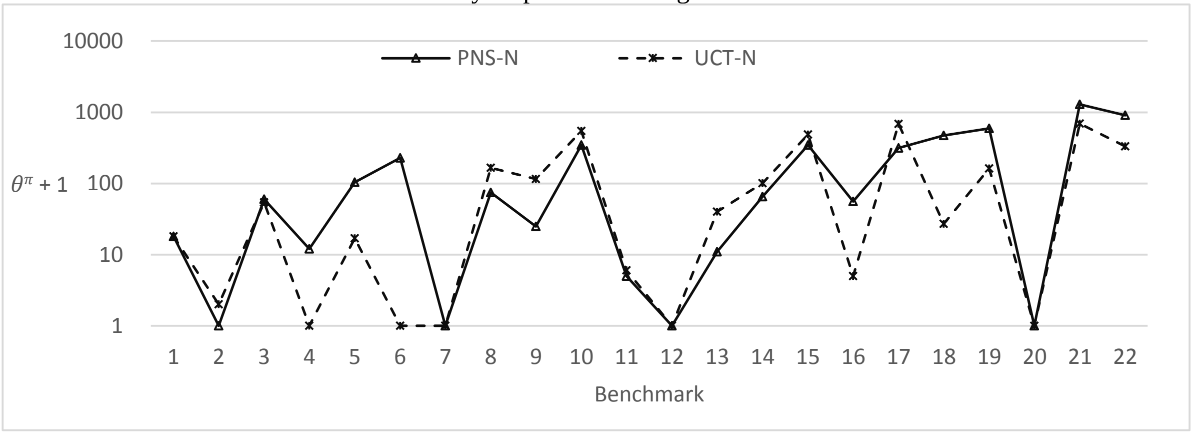

Fig. 7: Best performing metrics PNS-N and UCT-N comparison using number of incorrect jobs θ^{π}. All values of θ^{π} are added by 1 so that the data may be plotted on a logarithmic scale.

Again, we calculate the sum of θ^{PNS-N} and θ^{UCT-N} for all benchmarks, with values of 4938 and 3461, respectively. We then calculate the sum of $\log_{10}\left(\frac{\theta^{PNS-N}+1}{\theta^{UCT-N}+1}\right)$ for all benchmarks for a value of 5.010. We can again see that the JL-UCT algorithm and the UCT-N metric is superior. In addition, since the positions chosen as our benchmarks are all solved, JL-PNS tends to be advantageous in solved positions. For unsolved positions, our conjecture is that JL-UCT would perform even better.

5.3 Evaluating Opening Book Quality Through Competitive Play

We now construct opening books using the three top performing heuristic metrics determined by the previous experiment. Namely, we will use the node count heuristic metric for both JL-PNS and JL-UCT, and the hybrid heuristic metric for JL-PNS. The details of the opening book construction method will be described in Subsection 5.3.1, and the experiment will be described in Subsection 5.3.2.

5.3.1 Opening Book Construction

An opening book can be generated using any BF-JL search algorithm by providing several opening positions, then attempting to solve them using the BF-JL search. Since most opening positions are difficult to solve, we must also specify the maximum number of node expansions, so that the BF-JL search does not run indefinitely. Once the BF-JL search has completed, either due to the opening position being solved or when the node expansion limit has been reached, the search tree is then stored into an external file, where moves and their corresponding evaluation values can be retrieved easily by the game program. In this article, we choose to store the search tree using SQLite, a relatively simple transactional SQL database engine which NCTU6 may read via a locally accessible file.

In order to exploit transpositions and, in turn, reduce book size, a hash key is calculated for each node in the search tree using Zobrist hashing (Zobrist, 1970). The edges (all the moves) in the search tree, are stored in the database. For each entry in the moves table, we store the hash key of the game position prior to making the move, and the hash key of the game position after making the move. In addition, we define two values, the move status and the score, so that the game program may utilize the opening book. There are three move statuses: *win*, *loss*, and *general*, which tell the game program whether the corresponding move is winning, losing, or neither. During play, the game program will prioritize choosing winning moves, then general moves according to score.

The scores of the moves can be calculated in various ways depending on the heuristic metrics described above. For example, if the score is calculated using the move count heuristic metric, the score of a particular move is calculated by dividing the visit count of the move by the total visit count of its siblings. For all general moves, the game program can be designed to either pick the move with the best score, or choose a move according to a probability distribution that corresponds with the score. In practice, the latter option is more often used, since an opponent may take advantage of the game program’s predictability otherwise. The probability distribution can be naïve, where the probability of choosing a move is proportional to its score. Alternatively, there are more sophisticated methods of mapping scores to probabilities, such as softmax or epsilon-greedy. For the purposes of this article, the best move is always chosen to allow for a consistent experiment. In other words, the best move to play is given the maximum score, while all other candidate moves are given a score of 0. The criteria for which the best move is chosen is described in the heuristic metric sections above (Section 3 and Subsection 4.4).

5.3.2 Experiment Setup and Discussion

We use a set of commonly played openings on the website Little Golem (2015) for the next experiment. Of the 176 openings collected in (Wu, Tsai, Lin *et al.*, 2012), 100 were chosen at random for the construction of three opening books using the node count heuristic metric for JL-PNS and JL-UCT, and the hybrid heuristic metric for JL-PNS. The openings used in this experiment are not related to the set of benchmark positions used in the first and second experiments. The JL node limit was set to 3000, and the game program was configured to always choose the best move to play. With 3000 node expansions, each opening required about 8 hours to compute using JL-PNS and JL-UCT. This works out to roughly 33 days with the 100 randomly chosen openings. The significant time cost required to construct the openings is the main reason why only 100 of the 176 openings were used.

The three opening books were then used by the same version of NCTU6 that was used during the opening book generation to play against each other. The competition benchmarks consisted of the 100 openings used, so that it is guaranteed that at least the first move would be played by the book. For each pair of programs, each opening was to be played twice for fairness (the first program plays first, and the second program responds, and vice versa), for a total of 200 games per pair of programs. This competition scheme is also run using the JL framework, as described in more detail in (Wei, Liang, Wu *et al.*, 2015).

	Total Games	**Win**	**Draw**	**Win Rate**
UCT-N vs. PNS-N	200	105	15	0.5625
UCT-N vs. PNS-hybrid	200	107	13	0.5675
PNS-hybrid vs. PNS-N	200	94	15	0.5075

Table 3: Comparison of opening book generation methods.

From Table 3, we can see that UCT-N performs better than both heuristic metrics for JL-PNS, and PNS-hybrid is slightly better than PNS-N when playing directly against each other.

Lastly, we compare the playing strength of NCTU6 with and without using opening books. The PNS-N, PNS-PA, and UCT-N opening books were able to achieve a winning rate of 0.54, 0.5225, and 0.61 against NCTU6 without an opening book, respectively. This confirms that UCT-N is the strongest of the three JL-based methods.

6. CONCLUSION

In this article, we applied the previously proposed JL-PNS algorithm to active Connect6 opening book construction. Since PNS is not designed to provide estimates of position strength, we proposed a set of four heuristic metrics that enable us to indicate the best move to play after a game tree has been generated. These four metrics are the node count, proof/disproof-number ratio, minimax evaluation value, and a hybrid method that uses both proof/disproof-numbers and the game status, which is generated by the minimax evaluation value. Experiments show that for solved positions, the node count heuristic metric performs the best. In contrast, during competitive play, the opening book generated by the hybrid metric performs slightly better than the book generated by the node count metric for JL-PNS.

We then propose the JL-UCT algorithm, which treats opening positions as multi-armed bandit problems. JL-UCT uses the common UCB1 function as its selection criterion in conjunction with the game playing program acting as the expansion mechanism. Experiments show that for solved positions, JL-UCT is capable of providing good interim estimates of position strength, and can locate the best move to play more accurately, even from very early stages of the search. The above conclusions are also supported through competitive play. The opening book generated by JL-UCT with the node count heuristic metric is most effective, compared with the node count and hybrid methods with JL-PNS.

Future research topics include the tuning of JL-UCT win rate value initializations, additional features in the UCB1 function that deals with variance and instability, and the possibility of applying other multi-armed bandit models.

ACKNOWLEDGEMENTS

The authors would like to thank the National Center for High-performance Computing (NCHC) for computer time and facilities, and thank the Ministry of Science and Technology of the Republic of China (Taiwan) for financial support of this research under contract numbers MOST 102-2221-E-009-069-MY2, 102-2221-E-009-080-MY2, 104-2221-E-009-127-MY2, and 104-2221-E-009-074-MY2.

7. REFERENCES

"Little Golem." (2015). From http://www.littlegolem.net.

"Sqlite Home Page." (2015). From http://www.sqlite.org/.

Allis, L. V., Van der Meulen, M. and Van den Herik, H. J. (1994). Proof-Number Search. *Artificial Intelligence*, Vol. 66, No. 1, pp. 91-124.

Audouard, P., Chaslot, G., Hoock, J.-B., Pérez, J., Rimmel, A. and Teytaud, O. (2009). Grid Coevolution for Adaptive Simulations: Application to the Building of Opening Books in the Game of Go. *Applications of Evolutionary Computing*, pp. 323-332, Springer.

Auer, P., Cesa-Bianchi, N. and Fischer, P. (2002). Finite-Time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, Vol. 47, No. 2-3, pp. 235-256.

Baier, H. and Winands, M. H. (2011). Active Opening Book Application for Monte-Carlo Tree Search in 19× 19 Go. *23rd Benelux Conference on Artificial Intelligence (BNAIC 2011)*.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. and Colton, S. (2012). A Survey of Monte Carlo Tree Search Methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, Vol. 4, No. 1, pp. 1-43.

Buro, M. (1999). Toward Opening Book Learning. *ICCA Journal*, Vol. 22, No. 2, pp. 98-102.

Chaslot, G. M.-B., Hoock, J.-B., Pérez, J., Rimmel, A., Teytaud, O. and Winands, M. H. (2009). Meta Monte-Carlo Tree Search for Automatic Opening Book Generation. *Proc. 21st Int. Joint Conf. Artif. Intell., Pasadena, California*.

Chen, J.-C., Wu, I.-C., Tseng, W.-J., Lin, B.-H. and Chang, C.-H. (2014). Job-Level Alpha Beta Search. *Computational Intelligence and AI in Games, IEEE Transactions on*, Vol. 7, No. 1, pp. 28-38.

Chou, C.-W., Chou, P.-C., Doghmen, H., Lee, C.-S., Su, T.-C., Teytaud, F., Teytaud, O., Wang, H.-M., Wang, M.-H. and Wu, L.-W. (2012). Towards a Solution of 7x7 Go with Meta-MCTS. *13th Advances in Computer Games, ACG 2011, Tilburg, the Netherlands*, Vol. 7168, pp. 84-95, Springer, Heidelberg, Germany.

Gaudel, R., Hoock, J.-B., Pérez, J., Sokolovska, N. and Teytaud, O. (2011). A Principled Method for Exploiting Opening Books. *7th Computers and Games Conference, CG 2010, Kanazawa, Japan*, Vol. 6515, pp. 136-144, Springer, Heidelberg, Germany.

Hyatt, R. M. (1999). Book Learning-a Methodology to Tune an Opening Book Automatically. *ICCA Journal*, Vol. 22, No. 1, pp. 3-12.

Karapetyan, A. and Lorentz, R. J. (2006). Generating an Opening Book for Amazons. *4th Computers and Games Conference, CG 2004, Ramat-Gan, Israel*, Vol. 3846, pp. 161-174, Springer, Heidelberg, Germany.

Kloetzer, J. (2011). Monte-Carlo Opening Books for Amazons. *7th Computers and Games Conference, CG 2011*, Vol. 6515, pp. 124-135, Springer, Heidelberg, Germany.

Knuth, D. E. and Moore, R. W. (1975). An Analysis of Alpha-Beta Pruning. *Artificial Intelligence*, Vol. 6, No. 4, pp. 293-326.

Lin, P.-H. and Wu, I. (2009). Nctu6 Wins Man-Machine Connect6 Championship 2009. *ICGA Journal*, Vol. 32, No. 4, pp. 230.

Lincke, T. R. (2001). Strategies for the Automatic Construction of Opening Books. *Second Computers and Games Conference, CG 2000, Hamamatsu, Japan*, Vol. 2063, pp. 74-86, Springer, Heidelberg, Germany.

Saffidine, A., Jouandeau, N. and Cazenave, T. (2012). Solving Breakthrough with Race Patterns and Job-Level Proof Number Search. *13th Advances in Computer Games, ACG 2011, Tilburg, the Netherlands*, Vol. 7168, pp. 196-207, Springer, Heidelberg, Germany.

Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P. and Sutphen, S. (2007). Checkers Is Solved. *Science*, Vol. 317, No. 5844, pp. 1518-1522.

Wei, T.-H., Tseng, W.-J., Wu, I. and Yen, S.-J. (2013). Mobile6 Wins Connect6 Tournament. *ICGA Journal*, Vol. 36, No. 3, pp. 178-179.

Wei, T.-H., Wu, I.-C., Liang, C.-C., Chiang, B.-T., Tseng, W.-J., Yen, S.-J. and Lee, C.-S. (2014). Job-Level Algorithms for Connect6 Opening Position Analysis. *Third Workshop on Computer Games, CGW 2014, ECAI 2014, Prague, Czech Republic*. Computer Games, Vol. 504, pp. 29-44, Springer, Switzerland.

Wei, T., Liang, C.-C., Wu, I.-C. and Chen, L.-P. (2015). Software Development Architecture for Job-Level Algorithms. *ICGA Journal*, Vol. 38, No. 3, pp 149-164.

Wu, I.-C., Lin, H.-H., Lin, P.-H., Sun, D.-J., Chan, Y.-C. and Chen, B.-T. (2011). Job-Level Proof-Number Search for Connect6. *7th Computers and Games Conference, Kanazawa, Japan*, Vol. 6515, pp. 11-22, Springer, Heidelberg, Germany.

Wu, I.-C., Lin, H.-H., Sun, D.-J., Kao, K.-Y., Lin, P.-H., Chan, Y.-C. and Chen, P.-T. (2013). Job-Level Proof Number Search. *Computational Intelligence and AI in Games, IEEE Transactions on*, Vol. 5, No. 1, pp. 44-56.

Wu, I.-C. and Lin, P.-H. (2010). Relevance-Zone-Oriented Proof Search for Connect6. *Computational Intelligence and AI in Games, IEEE Transactions on*, Vol. 2, No. 3, pp. 191-207.

Wu, I.-C. and Lin, P. (2008). Nctu6-Lite Wins Connect6 Tournament. *ICGA Journal*, Vol. 31, No. 4, pp. 240-243.

Wu, I.-C., Tsai, H.-T., Lin, H.-H., Lin, Y.-S., Chang, C.-M. and Lin, P.-H. (2012). Temporal Difference Learning for Connect6. *13th Advances in Computer Games, ACG 2011, Tilburg, the Netherlands*, Vol. 7168, pp. 121-133, Springer, Heidelberg, Germany.

Wu, I.-C. and Yen, S.-J. (2006). Nctu6 Wins Connect6 Tournament. *ICGA Journal*, Vol. 29, No. 3, pp. 157-158.

Wu, I. and Huang, D. (2006). Connect6. *ICGA Journal*, Vol. 28, No. 4, pp. 234-242.

Wu, I., Lin, Y.-S., Tsai, H.-T. and Lin, P.-H. (2011). The Man-Machine Connect6 Championship 2011. *ICGA Journal*, Vol. 34, No. 2, pp. 103.

Zobrist, A. L. (1970). A New Hashing Method with Application for Game Playing. *ICCA journal*, Vol. 13, No. 2, pp. 69-73.