JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 28, 99-113 (2012)

Solving Nine Layer Triangular Nim*

YI-CHANG SHAN, I-CHEN WU, HUNG-HSUAN LIN AND KUO-YUAN KAO+
Department of Computer Science
National Chiao Tung University
Hsinchu, 300 Taiwan
+*Department of Information Management*
National Penghu University of Science and Technology
Penghu, 880 Taiwan

Triangular Nim, one variant of the game Nim, is a common two-player game in Taiwan and China. In the past, Hsu [9] strongly solved seven layer Triangular Nim while some of the authors recently strongly solved eight layer Triangular Nim. The latter required 8 gigabytes in memory and 8,472 seconds. Using a retrograde method, this paper strongly solves nine layer Triangular Nim. In our first version, the program requires four terabytes in memory and takes about 129.21 days aggregately. In our second version, improved by removing some rotated and mirrored positions, the program reduces the memory by a factor of 5.72 and the computation time by a factor of 4.62. Our experiment result also shows that the loss rate is only 5.0%. This is also used to help improve the performance.

Keywords: Nim, Triangular Nim, retrograde, misère games, impartial games

1. INTRODUCTION

Triangular Nim [1, 2], which is a common two-player game in Taiwan and China, is one variant of the game *Nim.* Nim is a two-player mathematical game of strategy in which players take turns removing pieces from distinct heaps, each containing a set of pieces. Two players alternatively remove one or more pieces from the same heaps. The game won by the player who removes the last piece is called a normal play game. In contrast, the game won by the other is called a *misère play game*, or simply a *misère game* [3, 4].

Many interesting and useful theories were developed for Nim in [3-5]. Obviously, Nim games are never drawn. Nim games are also called *impartial* games since from all positions[1] the moves available to move by either player are exactly the same. Sprague [6] and Grundy [7] also gave some useful theoretical analysis, such as *Sprague-Grundy theorem*.

Since Nim games are never drawn, all positions are either winning or losing to the player to move. The positions that the player is to move wins are commonly called *N-positions*, while the others are called *P-positions* [3]. The positions that have no subsequent moves are called *terminal positions*. Apparently, all terminal positions are *P*-positions in the normal play game, while they are *N*-positions in the misère play game. In addition, it is clear to see the following two rules [8]:

1. For each *N*-position, there is at least one move to a *P*-position.
2. For each *P*-position, every move is to an *N*-position.

Received February 26, 2011; revised August 19, 2011; accepted August 29, 2011.
Communicated by Yuh-Jye Lee.
* This work was supported by the National Science Council of Taiwan, under Contracts No. NSC 97-2221-E-009-126-MY3 and NSC 99-2221-E-009-102-MY3.
[1] In this paper, *positions* for Triangular Nim include the information of the remaining pieces, but exclude the information of the player turn.

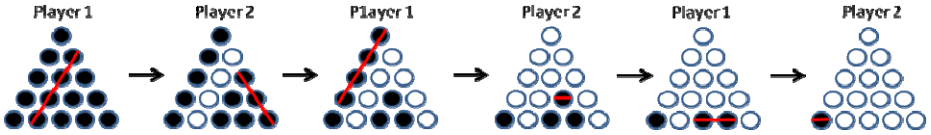


Fig. 1. A scenario of triangular Nim with side size five.

In the game of Triangular Nim, all pieces are placed in an equilateral triangle as shown in Fig. 1. Two players alternatively remove one or more consecutive pieces from one row or diagonal in the triangle of pieces. Triangular Nim shares some properties of Nim. For example, the games are never drawn; and the games are also impartial games. Similarly, the game won by the player who removes the last piece is called a *normal Triangular Nim*, while the game won by the other is called a *misère Triangular Nim*. Illustrated in Fig. 1, player 2 who removes the last piece loses the game in the misère. The version commonly played in Taiwan and China is the misère Triangular Nim.

A Triangular Nim is called a *k layer Triangular Nim*, if the equilateral triangle is of side size *k*. For example, Fig. 1 shows a 5 layer Triangular Nim which has 15 pieces in total. The position with all the 15 pieces is *the initial position* of a 5 layer Triangular Nim.

In the past, Hsu [9] solved 7 layer Triangular Nim by a brute force approach, called a *backward induction approach* in [8]. This approach tends to verify exhaustively all game positions to win or lose, from those with fewer pieces to more pieces. For the position without any pieces, set it to *P*-position or *N*-position depending on normal or misère, initially. Then, evaluate positions from those with fewer pieces to those with more pieces. For each position *P* to be evaluated, set it to *P*-position or *N*-position from all legal moves, according to the above two rules. Since all the moves will make the number of pieces fewer, the new position, say *P'*, must have been evaluated earlier than *P*. Thus, we can quickly derive the result of *P* from all *P'*. The method checking all legal moves for each position is called *forward checking method* in this paper.

Recently, both research teams [10, 11] solved 8 layer Triangular Nim by different methods independently. The researchers in [10] used the forward checking method by Hsu, while the researchers in [11], some of the authors of this paper, used a faster method to improve the performance. This method is a kind of *retrograde methods*.

In the past, the retrograde methods were successfully used by researchers in [12-15]. In retrograde methods, whenever the game-theoretical value (or just value) of a position, the win/loss status of a position, is derived to be a loss, we update the values of all its parents to be wins. Note that this paper follows the definitions of the terms in [16], such as strongly solved and game-theoretical values of positions.

The retrograde methods were shown to be very efficient, since the loss ratio (the number of losing positions over the number of all positions) is usually low. Thus, the update operations are not done often. In this paper, we will also show that the loss ratio is very low in Triangular Nim.

This paper follows, in principle, the retrograde method used in [11] to strongly solve 9 layer Triangular Nim and obtains the results that the first player wins in both normal and misère. However, strongly solving 9 layer Triangular Nim requires huge amount of memory. Our first version with the retrograde method requires 4 terabytes of data in memory. Since it is impossible to store all the data into RAM in most computer systems, the hard disks are used. Hence, it also increases much more computation time to solve.

For this problem, this paper proposes the block representation to decrease I/O frequencies and allow parallel processing to speed up the performance. Thus, it takes about 32.31 days on a machine with four cores and 129.21 days on one core aggregately.

Furthermore, we also improve the performance by making use of the characteristics of rotation and mirroring. The second version with the improvement efficiently reduces the memory by a factor of 5.72 and the computation time by a factor of 4.62.

This paper is organized as follows. Section 2 describes two approaches to solve Triangular Nim. Section 3 describes our algorithm and improvement. Section 4 does experiments for comparing different the index mappings. Section 5 makes concluding remarks.

2. RELATED WORKS

This section reviews the research for Triangular Nim in the past. Section 2.1 introduces the backward induction approach and described the forward checking method used in [9, 10]. Section 2.2 described the retrograde method used in [11] and this paper.

2.1 Backward Induction Approach

As described above, the backward induction approach is to evaluate the values of all game positions, from those with fewer pieces to more pieces. Since either player wins in Triangular Nim, the value of a given position is either a win or a loss, and therefore can be represented by one bit of data. Without loss of generality, let 1 indicate an *N*-position where the player is to move win; and let 0 indicate a *P*-position, otherwise.

For k layer Triangular Nim, since it has $k(k + 1)/2$ pieces, the total number of positions is $2^{k(k+1)/2}$. Thus, the space complexity is $O(2^{k(k+1)/2})$. From above, we can use $2^{k(k+1)/2}$ bits to represent the values of all the positions. For example, 5 layer Triangular Nim requires 2^{15} bits to represent all the values, while 9 layer Triangular Nim requires 2^{45} bits, equivalent to 2^{42} bytes, about 4 terabytes.

In order to identify the value of a given position in all the $2^{k(k+1)/2}$ bits, Hsu [9] simply used a $k(k + 1)/2$-bit binary number as a *position identifier*. In the position identifier, each bit indicates whether a corresponding piece exists. The corresponding pieces to bits are designated by a *piece mapping* in advance. For example, in Fig. 2, a piece mapping is shown, and a position of 5 layer Triangular Nim is identified as 011010111001011, equivalent to 13,771 in decimal, by a piece mapping. The initial position of 5 layer Triangular Nim is 111111111111111, all 1s. The initial positions for other layer Triangular Nim are all 1s, too.

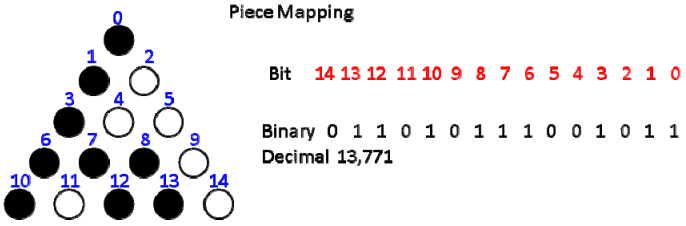


Fig. 2. A piece mapping of 5 layer Triangular Nim.

For a piece mapping, a legal move, removing some consecutive pieces, can also be represented by a $k(k + 1)/2$-bit binary number corresponding to these removed pieces. For example, for the piece mapping in Fig. 2, 000000000001011 is a legal move that removes the top three pieces at 0, 1 and 3. Since a legal move must remove consecutive pieces in a row or diagonal, the move at 1, 3 and 7 and the one at 1 and 8 are illegal. In [9], Hsu derived that the total number of moves is $k^2(k + 1)/2$ in k layer Triangular Nim. Hence, the number of legal moves is 75 for 5 layer Triangular Nim, while the number is 405 for 9 layer Triangular Nim. Thus, the time complexity for k layer Triangular Nim is roughly $O(k^3 2^{k(k+1)/2})$.

For a given position P, assume that a move M is legal in P. The bits with 1 in M must be 1 in P, too. We can use the following simple bit-wise operation to detect whether M is a legal move in P: The bit-wise AND operation on P and M is still M. In addition, we can also easily derive the position after the move M, by using bit-wise operations to remove all 1s of M from P. Namely, the new position P' is $P \oplus M$, where $\oplus$ is a bit-wise exclusive OR operation. For simplicity of discussion, P' is called a child of P, and P is the parent of P'.

For Triangular Nim, the backward induction approach is to derive $V(P)$ from $P = 0$ to $2^{k(k+1)/2} - 1$. $V(P)$ denotes the value of a given position P. An important property is: The values $V(P')$ for all children P' of P must have been evaluated, before the value $V(P)$ is evaluated.

In [9, 10], the researchers simply used a forward checking method to evaluate $V(P)$. Initially, the position 0 is a P-position in the normal play, that is, $V(0) = 0$, while it is an N-position in the misère play, that is, $V(0) = 1$. When scanning to the position P, evaluate $V(P)$ from the values $V(P')$ for all of its children P' by the two rules in section 1.

2.2 Retrograde Method

The research in [11] used a more efficient method, a kind of retrograde method, which is also used in this paper. Initially, the value $V(0)$ is initialized as the forward checking method. In addition, all the other $V(P)$ are initialized to 0. Then, the retrograde method also follows the backward induction approach to visit position P from position 0 to $2^{k(k+1)/2} - 1$ and perform the following update operation when visiting P.

- If $V(P) = 0$, then set $V(P_{par}) = 1$ for all of its parents P_{par}.

By induction, we can show that the above operation satisfies the following property: When position P is visited, the value $V(P)$ indicates the correct value. Assume that the properties are satisfied for all of its children. Then, if some child P' is a P-position, $V(P)$ is set to 1 when visiting P' based on the above operation. This is correct since the position P is indeed an N-position based on the first rule in section 1. If all children are N-positions, $V(P)$ remains 0. This is also correct since P is indeed a P-position based on the second rule. Thus, it is concluded that the above property is satisfied.

The retrograde method is very efficient for the following reason. The above update operation is done only when the position P is a P-position (a loss to the next player to move). Most importantly, the loss rate, the number of P-positions to that of all positions, is normally low. The loss rate is only 6.0% for 8 layer Triangular Nim according to [11],

while the loss rate is only 5.0% for 9 layer Triangular Nim according to the experiments in section 4. That is, the computation times can be reduced by a factor of 20 or so.

3. OUR APPROACH

This section presents our approach to strongly solve 9 layer Triangular Nim. In our approach, we use the retrograde method and also propose some new methods for efficient data structures and removing redundancies, respectively described in sections 3.1 and 3.2.

3.1 Data Structures and Algorithms

Since 9 layer Triangular Nim requires a huge amount of memory (4 terabytes for the design described in section 2.1), the memory space must be cut into disjoint blocks to make it feasible to process. To facilitate identifying the blocks, we let the most significant bits of position identifiers be the block identifier. For simplicity of discussion, the block representation in 8 layer Triangular Nim is illustrated in the rest of this section.

As described in section 2.1, 8 layer Triangular Nim requires 2^{36} bits, equivalent to 2^{33} bytes or 8 gigabytes. Assume to cut the memory space into 16 blocks, each with 512 megabytes. Then, the most four significant bits in the position identifier are used to indicate the *block identifier* (*Block ID* or *BID*), 0 to 15. The 16 blocks are denoted by B_0, …, B_{15}. From the piece mapping described in section 2.1, four pieces are designated as the bits of block identifier. The four pieces are called *Block-ID pieces* or *BID pieces*. For example, two such block representations using different BID pieces, marked with double cycles, are shown in Figs. 3 (a) and (b). Other block representations using six BID pieces are shown in the rest of Fig. 3.

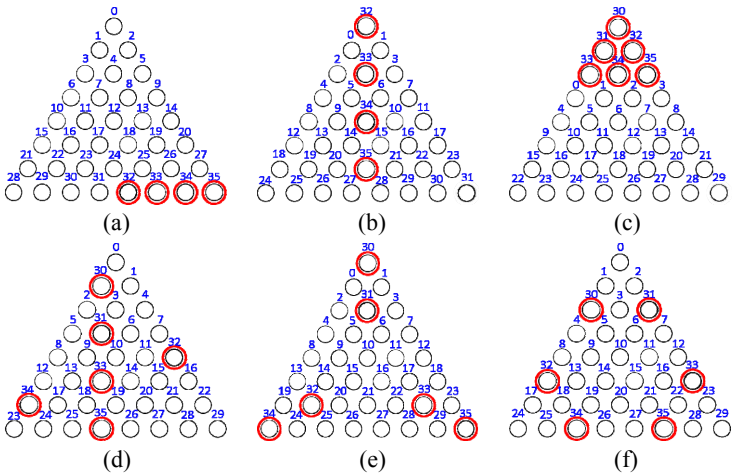


Fig. 3. Six block representations for 8 layer Triangular Nim.

Following the backward induction approach (in section 2.1), we visit positions from B_0 to B_{15}. Now, the next question is how to evaluate each block. Let us still follow the

approach as the retrograde method. We need to update the values in B_{i+1}, …, B_{15}, when visiting B_i. However, since the system memory (like RAM) may not be able to include all blocks, we use the following operations when visiting the block B_i.

Step 1: Load the block B_i (into the system memory).
Step 2: Evaluate the values of B_i as follows.
  2.1 Evaluate all positions P inside B_i by following the retrograde method. Namely, for each P with $V(P) = 0$, set $V(P_{par}) = 1$ for all of its parents P_{par} inside B_i.
  2.2 Restore the block B_i into the peripheral memory like hard disks.
Step 3: For all $j > i$, repeatedly perform the following.
  3.1 Load the block B_j.
  3.2 For each position P in B_i with $V(P) = 0$, set $V(P_{par}) = 1$ for all of its parents P_{par} in B_j.
  3.3 Restore the block B_j.

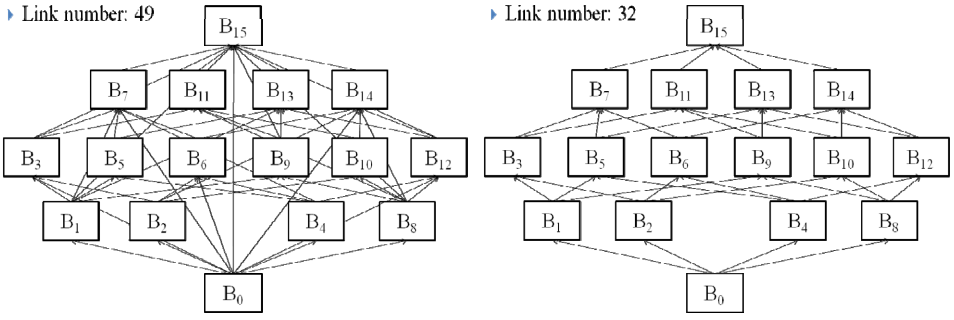
The above operations can be separated into different jobs. The operations 1 and 2 together, called *intra-block update* on B_i, is to update B_i itself. The operations 1 and 3 together for each j, called *inter-block update* from B_i to B_j, is to update B_j from B_i. For inter-block updates from B_i to B_j, the block B_j is called a parent of B_i, or B_i is the child of B_j.

The above operations can be further improved by saving the restore operations in the inter-block updates (at step 3.3), if we use a forward-checking method in the block level, instead of a retrograde method in the block level. Namely, all inter-block updates from B_i to B_j are done when visiting block B_j, not when visiting block B_i. The improved algorithm is modified as follows.

Step 1: Initialize the block B_j.
Step 2: For all $i = 0$ to $j - 1$, repeatedly perform the following inter-block updates.
  2.1 Load the block B_i.
  2.2 For each position P in B_i with $V(P) = 0$, set $V(P_{par}) = 1$ for all of its parents P_{par} in B_j.
Step 3: Process the intra-block update on B_j as follows.
  3.1 Evaluate all positions P inside B_j by following the retrograde method. Namely, for each P with $V(P) = 0$, set $V(P_{par}) = 1$ for all of its parents P_{par} inside B_j.
  3.2 Restore the block B_j into the peripheral memory like hard disks.

The above operation for 8 layer Triangular Nim requires 16 intra-block updates and 120 (= 15 × 16/2) inter-block updates. In fact, the number of inter-block updates can be far below the above number. For example, for the block representation in Fig. 3 (a), no operations are required for the inter-block update from B_0 to B_5, since it is impossible to have a move remove pieces 32 and 34 without removing 33.

The number of required inter-block updates is usually small if the BID pieces are distributed sparsely, and large if the BID pieces are grouped. After our analysis, the number of required inter-block updates is 49 for the one in Fig. 3 (a), and 32 for the one in Fig. 3 (b). Fig. 4 shows the inter-block updates for the two block representations, respectively. Usually, the smaller number of required inter-block updates is, the better performance is. This is also shown in our experiment in section 4.



(a) (b)

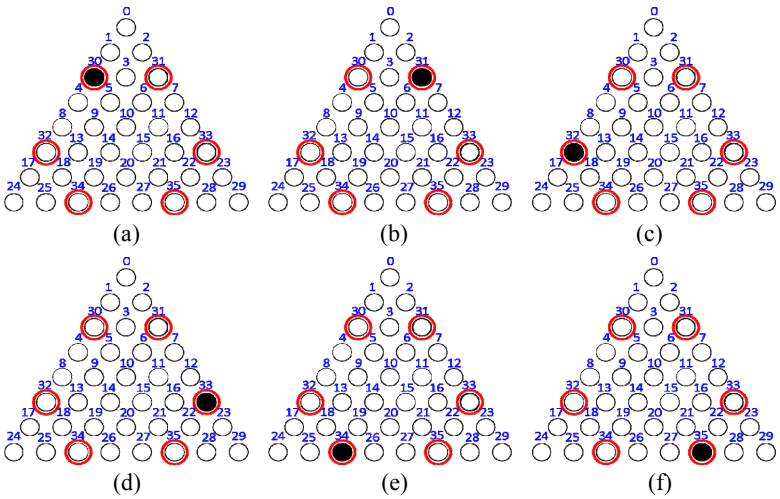
Fig. 4. Inter-block updates of the block representations in Figs. 3 (a) and (b), respectively.

3.2 Removing Redundancy

Due to symmetry and rotation, we do not have to evaluate all the positions. However, it is hard to save space for all symmetric and rotated positions directly, since it is hard to identify saved bits in our data structure shown in section 2.1. Fortunately, it becomes easier to save space on blocks.

Let us illustrate the block representation with six BID pieces shown in Fig. 3 (f) and the blocks, B_1, B_2, B_4, B_8, B_{16} and B_{32}, shown in Fig. 5. Due to symmetry and rotation, all the position values in one block, say B_2, can be found from those in another, say B_1, by finding the corresponding mirrored or rotated position values, and *vice versa*. Here, B_1 is said to be an isomorphic block of B_2, and *vice versa*.

All of the six blocks shown in Fig. 5 are actually isomorphic. Thus, it is sufficient to evaluate the values of one block, say B_1, only. And, we can get the values of other blocks from the values of B_1. These blocks form *an isomorphic group of blocks*.



(a) (b) (c)

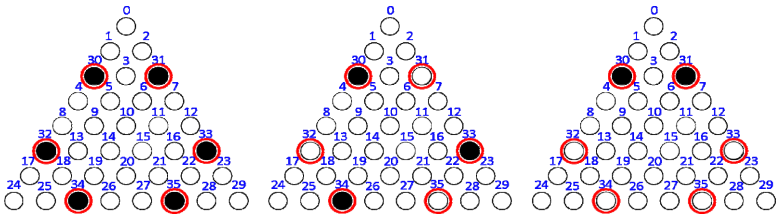
(d) (e) (f)

Fig. 5. Six rotated and mirrored blocks.

In an isomorphic group of blocks, we can simply choose to evaluate the block with smallest BID for simplicity. For example, in the group shown in Fig. 5, we choose B_1 to evaluate. For the improved algorithm in the previous section, we add the following rules.

1. When visiting Block B_i, skip it if there exists another isomorphic block with smaller BID.
2. When making an inter-block update from B_i to B_j, access $B_{i'}$ instead of B_i, if $B_{i'}$ is the block with smallest BID among all the isomorphic blocks of B_i.

In the second rule, when accessing block B_i, the block $B_{i'}$ (with the smallest BID) must be available, since we evaluate blocks from the lower BID to higher. In the rule, when accessing positions in B_i, access the corresponding positions in $B_{i'}$ according to the directions of rotation and mirroring.



(a) (b) (c)

Fig. 6. An isomorphic group with one, two and three distinct blocks only.

An isomorphic group may contain one, two, three, six distinct blocks, as illustrated in Figs. 6 (a)-(c), and Fig. 5 (a), respectively. Since an isomorphic group contains at most six distinct blocks, we can reduce the space by a factor of at most six. For the block representation (in both Figs. 5 and 6), the number of isomorphic groups with one distinct block is 2, that with two is 1, that with three is 6, and that with six is 7. Thus, for this block representation, we need to evaluate 16 blocks, among all the 64 blocks. Thus, we can save the space by a factor of four in this case.

The space saving rate grows higher and approximates to 6, if the block representation is well designed with large number of BID pieces. For example, for the block representation for 9 layer Triangular Nim shown in Fig. 10 (b), the number of isomorphic groups with one distinct block is 16, that with two is 8, that with three is 496, and that with six is 5,208. Thus, for this block representation, we need to evaluate 5,728 blocks only, among all the 32,768 blocks. The space is reduced by a factor of 5.72, approximate to 6. More specifically, we can reduce the space from 4 terabytes to 716 gigabytes.

4. EXPERIMENTAL RESULTS

In this section, all experiments were done on a 4 cores personal computer equipped with Intel® Core™ 2 Quad CPU Q6600, 2.4GHz CPU, 8 gigabytes RAM and 6 terabytes hard disks with RAID, which had been used in the desk-top system [17]. We used a simple parallel method to exploiting the parallelism of the four cores as follows. Whenever a

core is idle, we choose a block B_i which has not been evaluated yet and whose children have been evaluated. However, since parallelism is not the goal of this paper, we used the total times aggregated from the four cores.

Since it takes a huge amount of computation time for solving 9 layer Triangular Nim, we first investigate 8 layer Triangular Nim in section 4.1. Then, based on the experiences for 8 layer Triangular Nim, we solve 9 layer Triangular Nim in section 4.2.

4.1 Eight Layer Triangular Nim

This section investigates how to solve 8 layer misère Triangular Nim. We used the six block representations in Fig. 3 as our testing cases. We used (a), (b), (c), (d), (e) and (f) to indicate the six representations respectively.

Table 1 shows the experimental results, using the method without removing redundancy as described in section 3.1. Without removing redundancy, the total memory space requires 8 gigabytes for all cases. Table 1 clearly shows that the block representations with the less numbers of inter-block updates are more efficient, since the overhead for inter-block updates becomes smaller. Due to the overhead for inter-block updates, the block representations with larger block sizes tend to run faster. On the other hand, the block size cannot exceed the physical or virtual memory size of the operating systems. Thus, this is a tradeoff for the block size.

Table 2 shows the experimental results with removing redundancy. This table does not include the cases (a), (b) and (d), since the results are the same as Table 1. Both cases (a) and (d) are not symmetric and rotatable. Although the case (b) is symmetric, all the isomorphic groups contain one block only, and therefore the results are the same as Table 1.

In Table 2, both cases (e) and (f) perform apparently more efficiently than the case (c), since the case (c) does not remove the redundancy caused by rotation. The case (f) performs more efficiently than the case (e), since both the number of blocks and the number of inter-block updates are smaller.

Table 1. Experimental results for the block representations without removing redundancy in Fig. 3.

	Inter-block Number	Block Number	Block Size	Total Space	Aggregated Time
(a)	49	16	512MB	8GB	8,992 sec
(b)	32	16	512MB	8GB	8,472 sec
(c)	360	64	128MB	8GB	13,453 sec
(d)	208	64	128MB	8GB	11,042 sec
(e)	288	64	128MB	8GB	12,339 sec
(f)	336	64	128MB	8GB	13,128 sec

Table 2. Experimental results for three block representations with removing redundancy in Fig. 3.

	Inter-block Number	Block Number	Block Size	Total Space	Aggregated Time
(c)	228	40	128MB	5GB	8,423 sec
(e)	96	20	128MB	2.5GB	4,161 sec
(f)	87	16	128MB	2GB	3,705 sec

From the above experiments, we observe that the most important factor for high performance is to remove redundancy, and then to reduce the number of blocks or the number of inter-block updates. From the observation, we solve 9 layer Triangular Nim.

From the above experiments, we strongly solved all positions of 8 layer Triangular Nim with both normal and misère play. The results show that both initial positions for both normal and misère play are *N*-positions. The first player wins both by making the moves shown in Fig. 7 for the normal play and those in Fig. 8 for the misère play.

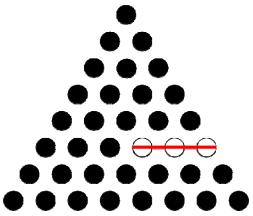


Fig. 7. The moves to win in 8 layer normal Triangular Nim.

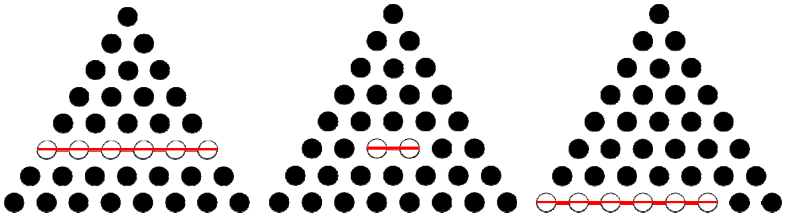
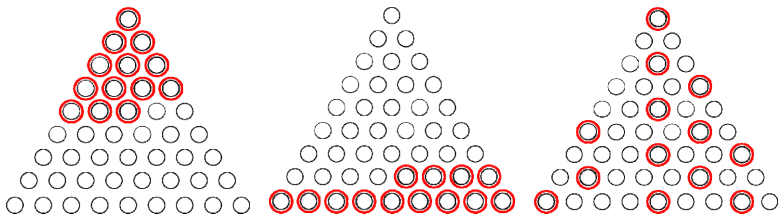


Fig. 8. The moves to win in 8 layer misère Triangular Nim.

4.2 Nine Layer Triangular Nim

In this section, we first investigated the block representations with 13 BID pieces, three of them shown in Fig. 9. Although each block requires a large block size, 512 megabytes, these BID pieces cannot form symmetry or rotation so that we cannot remove redundant blocks as described in section 3.2. Among the three in Fig. 9, the one in Fig. 9 (c) has the least number of inter-block updates. Unfortunately, since it is still not symmetric and rotatable, it took 32.31 days on four cores (129.21 days on one core) to finish the whole computation in Table 3.



(a) (b) (c)

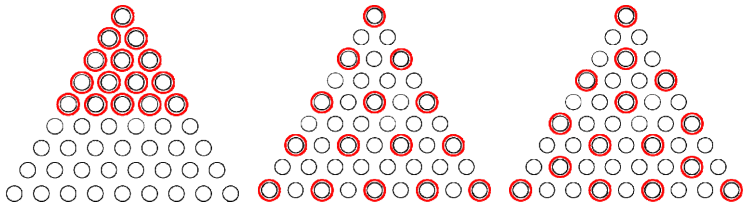
Fig. 9. Three block representations with 13 BID pieces.

Table 3. Experimental results for the block representations in Fig. 9.

	Inter-block Number	Blocks Number	Block Size	Total Space	Parallel Computation Time	Aggregated Time
(a)	121,600	8,192	512MB	4TB	N/A	N/A
(b)	107,024	8,192	512MB	4TB	N/A	N/A
(c)	88,064	8,192	512MB	4TB	32.31 days	129.21 days

Table 4. Experimental results for the block representations in Fig. 10.

	Inter-block Number	Blocks Number	Block Size	Total Space	Parallel Computation Time	Aggregated Time
(a)	298,320	16,640	128MB	2,080GB	N/A	N/A
(b)	102,950	5,728	128MB	716GB	8.32 days	33.23 days
(c)	78,852	5,728	128MB	716GB	6.99 days	27.93 days



(a) (b) (c)

Fig. 10. Three block representations with 15 BID pieces.

This section also investigates the block representations with 15 BID pieces, three of them shown in Fig. 10. Apparently, the one in Fig. 10 (a) is symmetric, while the other two are symmetric and rotatable. Due to symmetry and rotation, the number of blocks can be reduced by a factor of near 2 for the first and near 6 for the other two. The results showed that the computation times for the second and third are reduced to 8.32 days and 6.99 days as shown in Table 4, respectively. The third is slightly better than the second, since the number of inter-block updates is smaller as shown in Table 4. In brief, from the one in Fig. 9 (c) to that in Fig. 10 (c), we reduce the memory by a factor of 5.72 and the computation time by a factor of 4.62.

From the above experiments, we strongly solved all positions of 9 layer Triangular Nim in both normal and misère play. The results also show that both initial positions in both normal and misère play are *N*-positions. The first player wins both by making the moves shown in Fig. 11 for the normal play and those in Fig. 12 for the misère play.

Table 5 lists all the values of the initial positions of k layer Triangular Nim in both normal and misère, where k is 1 to 9. This table shows that the first player tends to win or that the initial positions tend to be *N*-positions. This is because the ratio of the number of *P*-positions to that of *N*-positions tends to be low, that is, most positions are *N*-positions.

We also analyzed the ratio in both normal and misère in Fig. 13 for all k layer Triangular Nim, where k is 1 to 9. From the Fig. 13, the ratios get smaller for large k. For 9 layer Triangular Nim, the ratio is 5.0% only. This explains why most of initial positions are *N*-positions. In addition, since most positions are *N*-positions, this shows that the retrograde method is clearly superior.

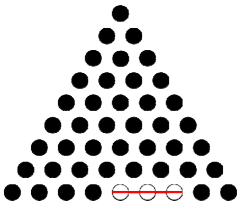


Fig. 11. The moves to win in 9 layer normal Triangular Nim.

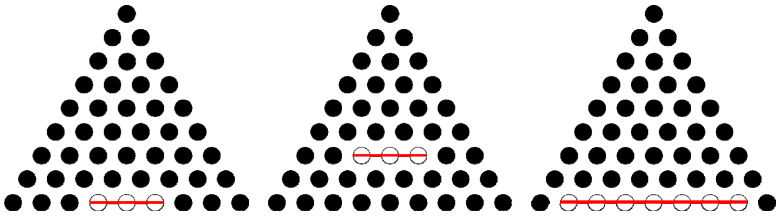


Fig. 12. The moves to win in 9 layer misère Triangular Nim.

Table 5. The result of *k* layer Triangular Nim.

k	*k* Layer Normal Triangular Nim	*k* Layer Misère Triangular Nim
1	Win	Loss
2	Loss	Win
3	Win	Loss
4	Win	Win
5	Win	Loss
6	Win	Win
7	Win	Win
8	Win	Win
9	Win	Win

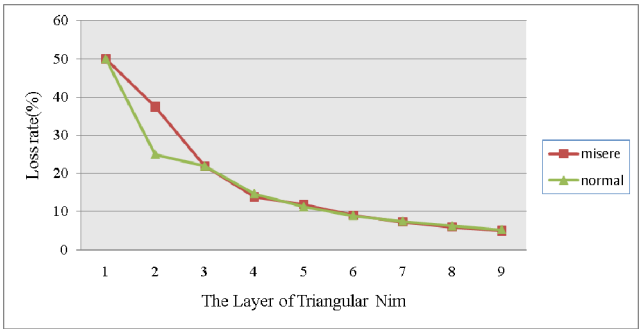


Fig. 13. The ratio of the number of *P*-positions to that of *N*-positions.

5. CONCLUSION

This paper has the following contributions.

- Using the retrograde methods, this paper strongly solves 9 layer Triangular Nim.
- This paper also proposes some methods to improve the performance, such as designing data structures in blocks, using the retrograde method, removing redundancy and selecting the block representation with the less number of inter-block updates. Especially, by removing redundancy, we reduce the memory by a factor of 5.72 and the computation time by a factor of 4.62.
- Our experiment result also shows that the ratio of the number of *P*-positions to that of *N*-positions is low, 5.0% for 9 layer Triangular Nim. Due to the low ratio, the retrograde method does perform well when compared with the traditional forward checking.

We believe the approach used in this paper can be applied to strongly solving other games, such as Chilled Domineering [18], drawn *k*-in-a-row games [19].

ACKNOWLEDGEMENT

The authors would like to thank anonymous reviewers for their valuable comments, and thank the National Science Council of Taiwan, for financial support of this research.

REFERENCES

1. A. Tucker, *Applied Combinatiorics*, 3rd ed., Wiley, New York, 1994, pp. 396-403.
2. B. Brian, *A Mathematical Pandora's Box*, Cambridge University Press, New York 1993.
3. E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning Ways for your Mathematical Plays*, 1st ed., Academic Press, New York, 1982.
4. J. H. Conway, *On Numbers and Games*, Academic Press, New York, 1976.
5. C. L. Bouton, "Nim, A game with a complete mathematical theory," *Annals of Mathematics*, Vol. 3, 1902, pp. 35-39.
6. R. P. Sprague, "Über mathematische Kampfspiele," *Tohoku Mathematical Journal*, Vol. 41, 1936, pp. 438-444.
7. P. M. Grundy, "Mathematics and games," *Eureka* 2, 1939, pp. 6-8, Reprint *Eureka* 27, 1964, pp. 9-11.
8. T. S. Ferguson, *Game Theory*, Lecture Notes, UCLA, 2008.
9. S. C. Hsu, "Solving the problem of 7 layer of triangular Nim," in *Proceedings of National Computer Symposium*, 1985, pp. 798-802.
10. S. Q. Bai and S. S. Lin, "On the study of 8 layer Triangular Nim," in *Proceedings of National Computer Symposium*, 2009, pp. 60-71.
11. H. H. Lin, I. C. Wu, and Y. C. Shan, "Solving eight layer Triangular Nim," in *Proceedings of National Computer Symposium*, 2009, pp. 200-204.
12. T. S. Hsu and P. Y. Liu, "Verification of endgame databases," *International Computer Game Association Journal*, Vol. 25, 2002, pp. 132-144.
13. K. Thompson, "Retrograde analysis of certain endgames," *Journal of the International Computer Chess Association*, Vol. 9, 1986, pp. 131-139.
14. K. Thompson, "6-Piece endgames," *Journal of the International Computer Chess Association*, Vol. 19, 1996, pp. 215-226.

15. P. H. Wu, P. Y. Liu, and T. S. Hsu, "An external-memory retrograde analysis algorithm," in *Proceedings of the 4th International Conference on Computers and Games*, LNCS 3846, 2004, pp. 145-160.
16. H. J. van den Herik, J. W. H. M. Uiterwijk, and J. V. Rijswijck, "Games solved: Now and in the future," *Artificial Intelligence*, Vol. 134, 2002, pp. 277-311.
17. I. C. Wu, C. P. Chen, P. H. Lin, G. Z. Huang, L. P. Chen, D. J. Sun, Y. C. Chan, and H. Y. Tsou, "A volunteer-computing-based grid environment for connect6 applications," in *Proceedings of the 12th IEEE International Conference on Computational Science and Engineering*, 2009, pp. 110-117.
18. K. Y. Kao, I. C. Wu, Y. C. Shan, and H. H. Lin, "Chilled domineering," in *Proceedings of International Conference on Technologies and Applications of Artificial Intelligence*, 2010, pp. 427-432.
19. S. H. Chiang, I. C. Wu, and P. H. Lin, "Drawn *k*-in-a-row games," *Theoretical Computer Science*, Vol. 412, 2011, pp. 4558-4569.

Yi-Chang Shan (單益章) received the B.S. and M.S. degrees in Computer Science from National Taiwan Normal University. He is currently working toward the Ph.D. degree in Institute of Computer Science, National Chiao Tung University, Hsinchu, Taiwan. His research interests include computer game, combinatorial game theory, volunteer computing and cloud computing.

I-Chen Wu (吳毅成) received his B.S. in Electronic Engineering from National Taiwan University (NTU), M.S. in Computer Science from NTU, and Ph.D. in Computer Science from Carnegie-Mellon University, in 1982, 1984 and 1993, respectively. He is currently a Professor of the Department of Computer Science and the director of the Institute of Multimedia Engineering, at National Chiao Tung University. He introduced the new game, Connect6, a kind of six-in-a-row game in 2005. Since then, Connect6 has become a tournament item in Computer Olympiad. He also led a team developing many computer game programs, which won 14 gold and at least 20 other pedals in computer game tournaments. Among them, a Connect6 program, named NCTU6, also won the latest three man-machine Connect6 championships. His research interests include artificial intelligence, Internet gaming, volunteer computing and cloud computing. He is a member of IEEE Computational Intelligence Society (CIS), and currently serves in the Games Technical Committee of IEEE CIS. More information about him can be found in the following URL: http://java.csie.nctu.edu.tw/~icwu/.

Hung-Hsuan Lin (林宏軒) received the B.S. degrees in Computer Science from National Chiao Tung University. He is currently working toward the Ph.D. degree in Institute of Computer Science, National Chiao Tung University, Hsinchu, Taiwan. His research interests include artificial intelligence, computer game, volunteer computing and cloud computing.

Kuo-Yuan Kao (高國元) received his Ph.D. degree from University of North Carolina at Charlotte. His program won the US Computer Go Championship in 1994. Since 1995, he has engaged in the research of combinatorial game theory and published several papers in this field. Kao is also a 6-dan amateur Go player; he currently serves as the director of Chinese Go Association, Taiwan.