Bitwise-Parallel Reduction for Connection Tests

Cameron Browne, *Member, IEEE,* Stephen Tavener

Abstract—This paper introduces *bitwise-parallel reduction* (BPR), an efficient method for performing connection tests in hexagonal connection games such as Hex and Y. BPR is based on a known property of Y that games can be reduced to a single value indicating the fully-connected player (if any) through a sequence of reduction operations. We adapt this process for bitwise-parallel implementation and demonstrate its benefit over a range of board sizes. BPR is by far the fastest known method if connection tests only need to be performed once per game, for example to evaluate board fills following Monte Carlo playouts.

Index Terms—Connection game, Combinatorial game, Hex, Y, Y reduction, Bitwise parallelism.

1 Introduction

Connection games are board games in which players vie to complete a specific type of connection with their pieces [1]. The connection theme imbues such games with an inherent depth that allows complex play to emerge from simple rule sets, and means that games tend to scale well to different board sizes. Many connection games are played on hexagonal grids for topological reasons that will be discussed shortly.

Monte Carlo tree search (MCTS) methods such as UCT, which have recently had such spectacular success in computer Go [2], have also been shown to work well with connection games [3], [4]. As the strength of such Monte Carlo players depends on their simulation rate [5], it is important to perform the menial operations of the game, such as legal move generation and win testing, as efficiently as possible.

Some connection games share the property that every game is guaranteed to produce exactly one winner *even if the board is filled randomly*. This makes them especially conducive to Monte Carlo analysis, as random playouts can be made quickly without the need for win testing until the board is full. Optimised win tests have the potential to allow faster simulation rates and better Monte Carlo results.

This paper describes a new method for win testing in hexagonally-based connection games that is more efficient than previously known methods.

2 Connection Games

This section describes two well known connection games, Hex and Y, and some common properties that allow efficient win tests in these and other games.

• *C. Browne and S. Tavener are with the Department of Computing, Imperial College, London, 180 Queens Gate, SW17, UK.*
E-mail: camb,sct110@doc.ic.ac.uk

Manuscript received November 27, 2011; revised January 1, 2012.

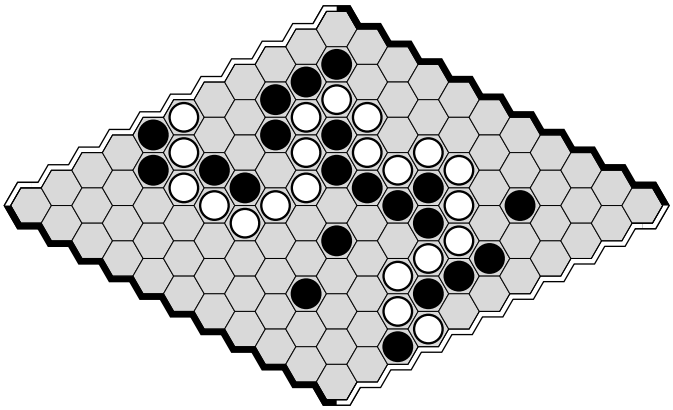

Fig. 1. A game of Hex won by white.

2.1 Hex

Hex is the quintessential connection game. Players take turns colouring cells of a hexagonally tiled rhombus, and win by connecting their sides of the board with a path of their colour [6]. Figure 1 shows a game won by white.

Hex was invented independently by mathematicians Piet Hein and John Nash in the 1940s. 11×11 was the original board size specified by Hein (although he later preferred 12×12) and is the official size used today in competitions such as the ICGA Computer Olympiad [4].

2.2 Y

Y is played in a hexagonally tiled triangle. Players take turns colouring cells and win by connecting all three sides of the board with a path of their colour (Figure 2). There are various "official" sizes, with size 11 being a common choice.

Y was invented by Claude Shannon in the 1950s as a variant of Hex without coloured edges. It was later mapped to a non-regular board by Schensted and Titus to bring the corners more into play [7].

It can be argued that Y is a fundamental form of Hex, as Hex can be phrased as a sub-game of Y. Figure 3 (left) shows a 5×5 game of Hex embedded in a size 9 game of Y; whichever player wins the Y game must also win

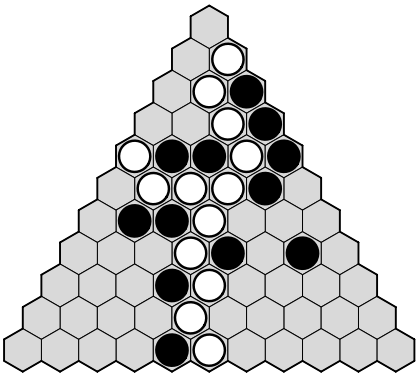

Fig. 2. A game of Y won by white.

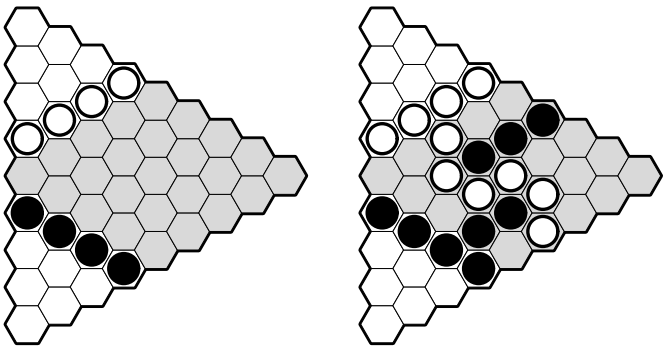

Fig. 3. Hex can be phrased as a sub-game of Y.

the embedded Hex game at the same time. The row of four white pieces and four black pieces constitute a *board template*, which guarantees that the player who connects either of these sets to the facing edge must necessarily connect all three sides (as per Y) while also completing an oriented cross-board path of a specific colour (as per Hex). Figure 3 (right) shows a sub-game won by white.

Such templates can be used to phrase a wide range of hexagonal connection tasks as a game of Y. For example, Figure 4 shows a board template that detects black connections between three non-adjacent board sides, as per the games Unlur and Cross. As a further example, Figure 5 shows a board template that detects white paths between a corner and the far side of the board.

2.3 Complementary Goals

An interesting feature of both games, Hex and Y, is that *exactly* one player must win every game. It is easy to see that the winner of both games shown in Figures 1 and 2 will remain unchanged regardless of how the remaining cells are coloured. What is not so obvious, however, is that each game will produce exactly one winner even if all board cells are randomly coloured from the outset.

This is largely due to the fact that the hexagonal grid is *trivalent*,[1] so deadlock problems that plague connection games on non-trivalent tilings, such as the square grid,

1. Three cells meet around each intersection.

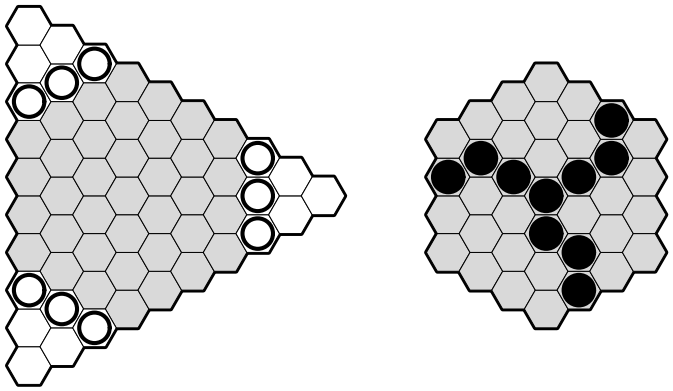

Fig. 4. A board template that detects black connections between three non-adjacent board sides.

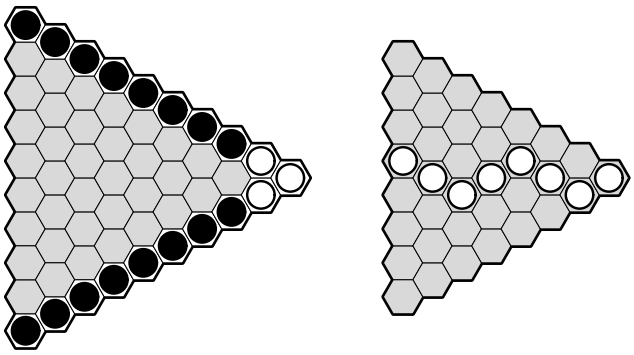

Fig. 5. A board template that detects white connections between a corner and the far side.

do not occur [8]. Hexagonal tiling is therefore used for many connection games.

This feature is a boon for Monte Carlo implementations, as a result is guaranteed for each game even if the board is filled randomly, and that win testing can be delayed to a single test at the end of the game rather than with every move. Even if a player wins early in the game, their winning path will still exist at the end.

This idea is exploited in UCT-based connection game players by Raiko and Peltonen [3] and the world champion Hex programme MoHex by Arneson et al. [4]. It is similar to the "fill the board" heuristic used to good effect in computer Go programs[9], and raises the possibility of very efficient playouts for massively parallel architectures and GPU implementation.

2.4 Connection Testing

One of the simplest methods for win testing in connection games is *path following*. This involves visiting each cell along one of the target board sides, and recursively visiting each unvisited cell of the specified player's colour – and its same-coloured neighbours – to see if that path visits the other target board side(s). Another approach is *bitwise-parallel breadth-first search*, attributed to Lukasz Lew, in which a bitset front indicating coloured cells connected to one side is iteratively grown, until the

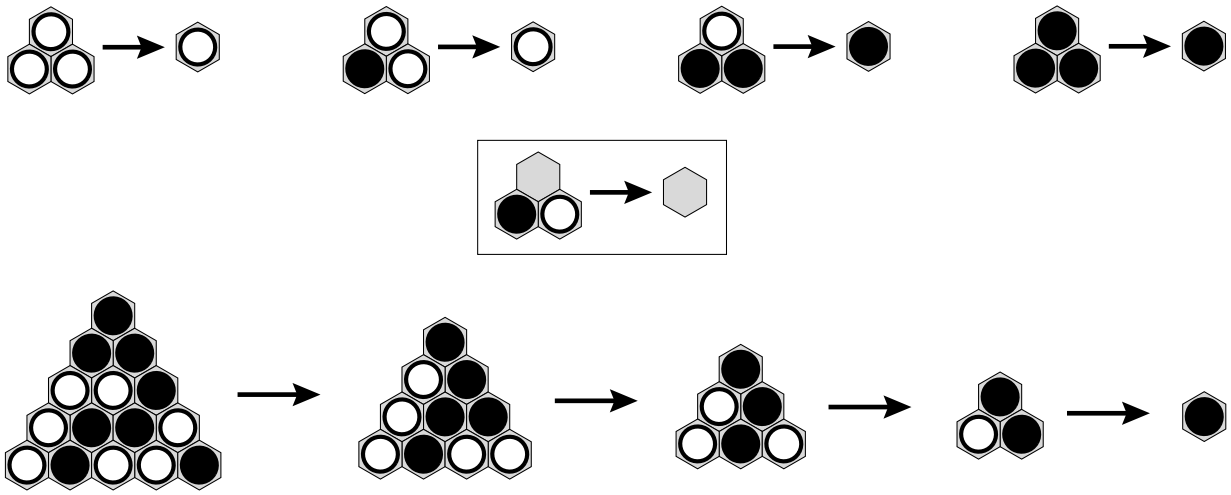

Fig. 6. The rules of Y reduction (top), the tiebreaker rule (middle) and a game of Y reduced to a black win (bottom).

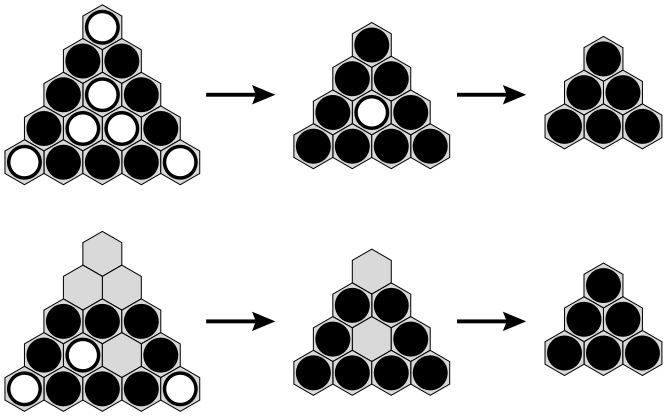

Fig. 7. Loop detection by Y reduction.

target is reached or no more growth occurs. These are reasonably efficient for delayed win tests performed after the fact.

If win testing is to be performed with every move, however, *union find* (UF) methods usually provide a better alternative. These incrementally build records of connected sets as each move is made, then each win test is a simple lookup to check whether the goal regions belong to the same connected set. UF methods amortise the cost of win tests across the lifespan of the game, but must still be updated on a per-move basis even if win tests are delayed. Sedgewick presents several UF algorithms, ranging in efficiency, as a programming exercise [10, pp.11-20].

3 Y Reduction

A property of Y known as *Y reduction* provides an alternative method for win testing [11]. Consider the sets of mutually adjacent cell triplets shown on the bottom row of Figure 6, each fully coloured. The connectivity of each triplet may be summarised by a single cell of the majority colour, as deadlocks cannot occur on such trivalent groupings.

Applying this rule to the cell triplets of a fully coloured board (bottom row), this means that each board of size d can be reduced to a board of size $d-1$ that maintains the connectivity of the larger game. Hence for a game of size d, $d-1$ reduction passes will produce a single cell whose colour indicates the winner.

The Y reduction property was observed by Schensted and Titus in the 1970s and later rediscovered in 2002 by game designer Steven Meyers [12]. Y reduction also holds for non-full boards if tied $\{empty, white, black\}$ triplets reduce to $\{empty\}$, as per the tiebreaker rule shown in Figure 6. Non-full boards will then reduce to a single value that indicates the winning colour if there is one, else $empty$ if the game has not yet been won.

The fact that Hex can be embedded in Y (Figure 3) means that Y reduction can also be used for win testing in Hex, provided that the Hex game is framed in the appropriate board template. Y reduction can potentially be used for win testing in many other hexagonal connection games (Figures 4 and 5) and even non-hexagonal ones, provided that the tiling is consistently trivalent.

Loops can also be detected as a by-product of the reduction process (Figure 7), by observing that:

1) All groups reduce to a single cell at some point.
2) Each interior cell c contributes to three reduction values $\{c'_0, c'_1, c'_2\}$ at the next level.
3) If $\{c'_0, c'_1, c'_2\}$ are the same (non-empty) colour, and this differs from c, then a loop has been reduced.

4 Bitwise-Parallel Reduction (BPR)

The encoding of game boards as sets of bits (known as *bitboards*) has been a common practice in Chess programming for decades [13].[2] This is done primarily to reduce memory requirements so that large databases of

2. See for example http://www.cis.uab.edu/info/faculty/hyatt/pubs.html for further details.

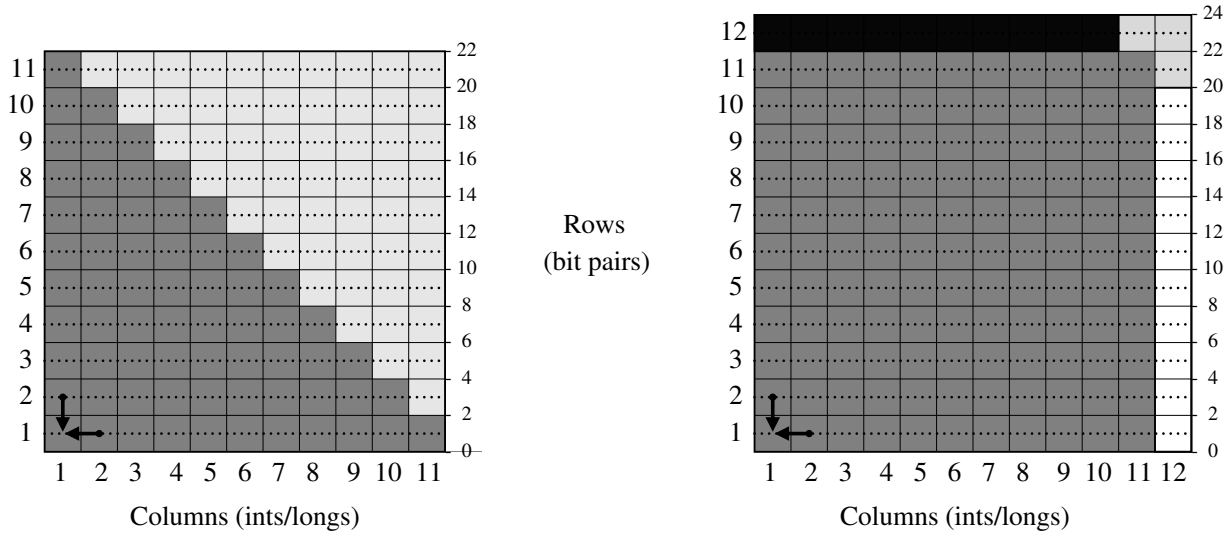

Fig. 8. Bitwise encodings for games of Y (left) and Hex (right). Each cell is a pair of bits.

positions may be stored and analysed. However, another reason to encode games at the bit level is to exploit the parallelism implied in the fact that operations applied to a datum are applied to *all* bits within it simultaneously.

For example, Ledvina et al. [14] demonstrate the benefit of bitwise-parallel operations for improving efficiency in communication signals, by extending known algorithms from operating on a single channel to operating on 32 channels simultaneously. Similar principles are now applied to the Y reduction process.

4.1 Bitwise Encoding

The possible states for each board cell ($empty$, $white$, $black$) are encoded in bit pairs as follows:

$$\begin{aligned} 00 &= empty \\ 01 &= white \\ 10 &= black \\ 11 &= unused \end{aligned}$$

The board is described by a one-dimensional array of integers (labelled $bits$), with each integer encoding one entire column as a set of bit pairs. The size of the array D_y will be the size of the enclosing Y board, which for Y will be the board size dim and for Hex will be $2{*}dim-1$, as the Hex game is embedded in a larger Y board.

Figure 8 shows such bitwise encodings for a size-11 game of Y (left) and an 11×11 game of Hex (right), as columns of bit pairs. Dark grey cells represent board cells which will take values from {00,01,10} according to the game being encoded, and light grey cells represent unused bits.[3] The black and white cells surrounding the Hex game encoding are the board template required for cross-board connection testing, which add one more row and column to be processed than for Y.

3. Unused bits do not affect computation time as bitwise operations are applied to all bits within the integer simultaneously.

4.2 Bitwise Reduction

Three bit pairs a, b and c can be reduced to a single bit pair according to the rules shown in Figure 6 as follows:

$$\begin{aligned} f(a,b,c) &= a\&b \mid a\&c \mid b\&c \\ &= a\&(b \mid c) \mid b\&c \end{aligned}$$

Table 1 show that this operation gives the correct result for all possible triplets {a,b,c} that can occur during the reduction process. The order of a, b and c is not important and the bit pair 11 will never occur. Note that this operation correctly implements the tiebreaker rule:

$$\begin{array}{cccc} 0 & 0 & 1 & \rightarrow \quad 0 \\ 0 & 1 & 0 & \rightarrow \quad 0 \end{array}$$

4.3 BPR Algorithm

These elements lead to an elegant algorithm for *bitwise-parallel reduction* (BPR), as shown in Listing 1. C is the maximum number of columns that need to be reduced each pass, which will be d for Y and $d+1$ for Hex due to the extra column required for the board template.

Listing 1 Bitwise-parallel reduction.

```
int BPR()
{
    for (int pass = 0; pass < D_y − 1; pass++)
        for (int col = 0; col < min(C, D_y-pass)-1; col++)
        {
            final int a = bits[col];
            final int b = (a >> 2);
            final int c = bits[col + 1];
            bits[col] = (a & (b | c)) | (b & c);
        }
    return bits[0] & 0x3;
}
```

TABLE 1
Reduction truth table for bit pair triplets $\{a,b,c\}$.

a	b	c	$a\&(b \mid c) \mid b\&c$
0	0	0	0
0	0	0	0
0	0	0	0
0	0	1	0
0	0	1	0
0	0	0	0
0	0	1	0
0	1	0	0
0	0	0	0
0	1	1	1
1	0	0	0
0	1	1	1
0	0	0	0
1	1	1	1
0	1	1	1
0	0	0	0
0	1	1	1
1	0	0	0
1	1	1	1
0	0	0	0

For each reduction pass, the $(a\&(b \mid c) \mid a\&c)$ operation is applied to a decreasing number of columns of the $bits$ array. The number of passes must always be exactly one less than the size of the enclosing Y board D_y. Bitwise-parallelisation therefore reduces the $O(n^3)$ reduction process [11] to an $O(n^2)$ process for boards that can pack each column into a single integer, i.e. board sizes up to 31x31 Hex and size 32 Y.

For each reduction operation, the a component is set to the current column, the b component is set to current column shifted down by 2 bits, and the c component is set to the next column. The reduction operation is therefore applied to all cells of each column simultaneously in a bitwise-parallel way. The result of the overall reduction process is the value remaining in the least significant bit pair of column 0, which will indicate the player who has connected all board sides, else 0 if none.

Bits above the board template row (row 12 in Figure 8) can be clipped without affecting the result. 32-bit ints can therefore be used for Hex games up to size 15×15 and Y games up to size 16. 64-bit longs can be used for Hex games up to size 31×31 and Y games up to size 32. The conversion from 32-bit to 64-bit is as simple as changing the base data type from int to long in the code.

4.4 Bit Packing

BPR can be optimised by using 64-bit longs and packing each with multiple board columns. This allows multiple board columns to be reduced in a single operation, further exploiting the benefits of bitwise-parallelism.

Even more savings can be made for per-game win tests, if each cell is described by a single bit indicating whether it is coloured white or black at the end of a playout. This allows an efficient encoding, e.g. an 11×11 Hex board packs into three 64-bit longs as follows:

1) bits per column = 11 + 1 (for template) = 12.
2) columns per long = 64 bits / 12 per column = 5.
3) longs per board = 12 columns / 5 per long + 1 = 3.

If per-move win tests are required, then the board can be split into two boards – one for *white* and one for *black* – with each bit indicating the absence or presence of a piece of that colour. Reduction can then be applied to the board of the appropriate colour each turn, halving the number of bits that must be processed.

BigInteger and BitSet classes were also tested to encode entire boards as single data items. All reductions could then be applied simultaneously each pass, removing the inner loop of the BPR algorithm entirely. However, the computational overhead for maintaining these more complex data types proved to outweigh any benefits obtained from increased parallelism.

4.5 Non-Destructive BPR

The BPR algorithm shown in Listing 1 is destructive as it acts directly upon the elements of the $bits$ array, and can only be used once per game. However, it is a simple matter to make the algorithm non-destructive, by saving the results of the first reduction pass to a secondary array and performing subsequent passes on it instead.

Another approach to non-destructive BPR is *iterative BPR*, based on an observation by van Rijswijck [11], in which the reduction values at all levels persist in a reduction pyramid, and only those few values affected by each move are updated each turn. This preserves reduction values between turns and is more conducive to per-move win testing, as each move necessitates relatively few updates and the win test is then a single access of the reduction pyramid's apex.

5 Experiments

This section describes the timing tests used to compare the algorithms in practice. Timing tests were on games of Hex and Y at typical board sizes.

5.1 Code

A programme for playing Hex and Y at arbitrary board sizes was implemented in Java. The programme supports legal moves, random playouts and win detection, but does not yet provide an AI opponent. The code is available at: http://www.doc.ic.ac.uk/~camb/bpr.zip

The timing experiments were run on a standard dual core Macbook Pro with *i*5 processor, running Java 1.6.0_26 under JVM 20.1 with Eclipse Helios.

TABLE 2
Milliseconds per 10,000 playouts over 1,000 sets, with win tests applied per-game.

One Connection Test Per Game								
		Board Size						
Game	Method	7	9	**11**	13	15	17	19
Hex	UF	34.47 ±0.221	64.30 ±0.135	**94.34 ±0.163**	121.7 ±0.208	160.6 ±0.189	205.5 ±0.283	256.2 ±0.222
	WQHUF	35.90 ±0.183	62.60 ±0.150	**91.21 ±0.175**	119.8 ±0.252	153.6 ±0.205	196.0 ±0.234	245.3 ±0.443
	Path	9.83 ±0.072	16.83 ±0.050	**23.51 ±0.054**	31.15 ±0.058	40.98 ±0.090	51.77 ±0.093	64.14 ±0.123
	BPR	1.44 ±0.013	2.61 ±0.019	**3.16 ±0.010**	4.30 ±0.016	5.13 ±0.021	7.04 ±0.040	8.60 ±0.029
Y	UF	30.85 ±0.103	52.30 ±0.140	**69.23 ±0.102**	94.57 ±0.166	123.5 ±0.237	159.0 ±0.213	197.4 ±0.345
	WQHUF	30.76 ±0.082	51.37 ±0.128	**68.34 ±0.147**	93.80 ±0.155	123.2 ±0.192	157.1 ±0.206	194.1 ±0.326
	Path	7.05 ±0.067	12.54 ±0.044	**16.91 ±0.050**	23.22 ±0.061	29.42 ±0.062	38.38 ±0.073	48.31 ±0.104
	BPR	0.70 ±0.007	1.19 ±0.013	**1.42 ±0.009**	1.78 ±0.012	2.32 ±0.014	2.94 ±0.013	3.31 ±0.013

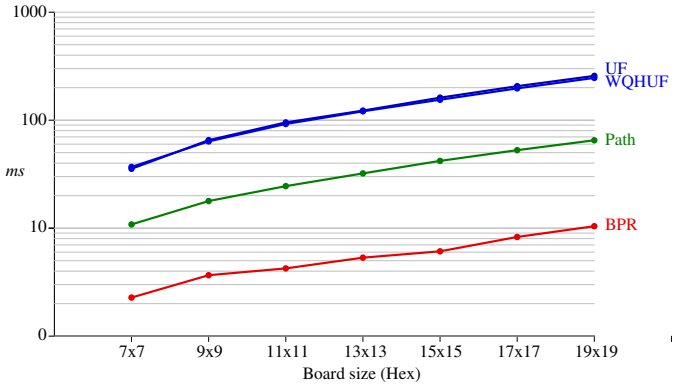

Fig. 9. Milliseconds per 10,000 trials, per-game (Hex).

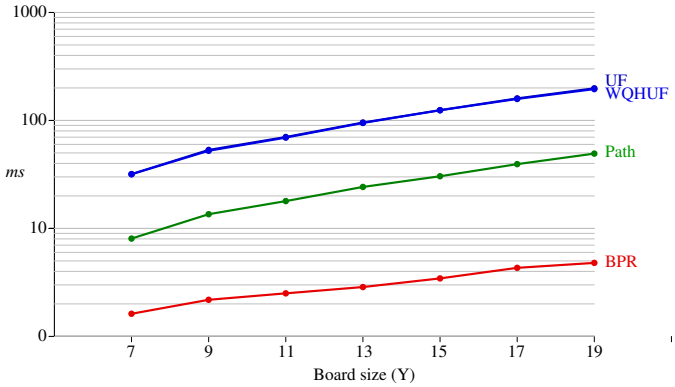

Fig. 10. Milliseconds per 10,000 trials, per-game (Y).

5.2 Connection Tests

The following win test algorithms were implemented:

- ***UF***: Standard union find search as described by Sedgewick [10].
- ***WQHUF*** (Weighted Quick Halving Union Find): Quick union find weighted to connect smaller trees to larger ones, with path compression [10].
- ***Path***: Simple path following from a target region.
- ***BPR***: Bitwise-parallel reduction, non-destructive, using packed 64-bit longs.
- ***BPR+***: Iterative bitwise-parallel reduction, using 64-bit longs.

5.3 Method

A number of timing trials were conducted for random games played out on odd-sized boards ranging from 7 to 19 in size. This gives a practical range, as boards smaller than size 7 or larger than size 19 are rare.

For each game at each board size, 1,000 sets of 10,000 trials were run for each type of win test.[4] Each trial consisted of a random playout with win testing followed by a random playout without win testing, with the second playout halted at the same number of moves as the first. For per-game trials, win tests were only performed once the board was full. For per-move trials, win tests were performed after every move.

Each pair of playouts was timed using System.nanoTime() to give playout times t_w and t_{wo} respectively. Each trial's overall time was taken as $t_t = t_w - t_{wo}$, indicating the average time spent win testing for a game of that length, using the current win test method. Note that t_t includes the time spent both on the win test itself and any preparatory operations.

6 Results

The results are divided into per-game and per-move win tests. The per-game results are the real findings of this study, while the per-move results are more of a cautionary tale of when BPR should *not* be used.

6.1 Per-Game Win Tests

Table 2 shows the timings for per-game win tests for odd-sized Hex and Y boards ranging from sizes 7 to 15. The times shown are the mean times for each run of 10,000 trials, over 1,000 runs, in milliseconds for easy comparison, along with their 95% confidence interval. Size 11 is highlighted as 11×11 is the standard size for Hex, so is the case of most interest.

It can be seen that BPR methods are around 25-30 times faster than union find methods for Hex and 50 times faster for Y when win tests are performed once per game, with path-following lying midway between these extremes. These results are shown graphically in Figures 9 (Hex) and 10 (Y). Note that the Y axis in the graphs (milliseconds) is a logarithmic scale.

4. That is, timings were based on a total of 10,000,000 trials for each game at each board size for each win test algorithm.

TABLE 3
Milliseconds per 10,000 playouts over 1,000 sets, with win tests applied per-move.

One Connection Test Per Move								
		Board Size						
Game	Method	7	9	**11**	13	15	17	19
Hex	UF	38.39 ±0.090	66.67 ±0.146	**92.35 ±0.145**	128.8 ±0.233	170.4 ±0.237	218.4 ±0.239	279.8 ±0.614
	WQHUF	38.59 ±0.071	66.61 ±0.139	**92.93 ±0.153**	128.0 ±0.158	170.8 ±0.180	219.9 ±0.261	278.7 ±0.548
	Path	159.6 ±0.369	419.4 ±0.664	**757.8 ±1.425**	1,379.0 ±1.299	2,334.0 ±3.044	3,778.0 ±4.919	5,924.0 ±13.32
	BPR+	65.93 ±0.146	131.4 ±0.231	**226.8 ±0.394**	357.5 ±0.783	508.7 ±0.905	708.0 ±1.476	943.1 ±1.729
Y	UF	21.57 ±0.080	31.63 ±0.105	**40.12 ±0.107**	53.45 ±0.183	67.15 ±0.172	85.48 ±0.206	100.1 ±0.212
	WQHUF	21.90 ±0.088	32.14 ±0.107	**40.75 ±0.110**	53.67 ±0.184	68.00 ±0.163	86.88 ±0.192	103.5 ±0.173
	Path	82.36 ±0.270	179.9 ±0.372	**312.8 ±0.445**	543.0 ±0.670	890.5 ±1.161	1,430.0 ±1.996	2,067.0 ±3.794
	BPR+	22.64 ±0.062	47.34 ±0.085	**83.34 ±0.167**	126.2 ±0.246	182.2 ±0.260	251.9 ±0.232	342.5 ±0.632

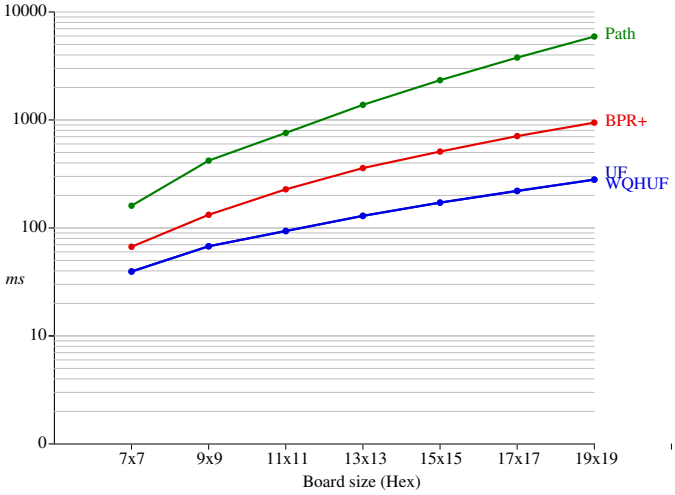

Fig. 11. Milliseconds per 10,000 trials, per-move (Hex).

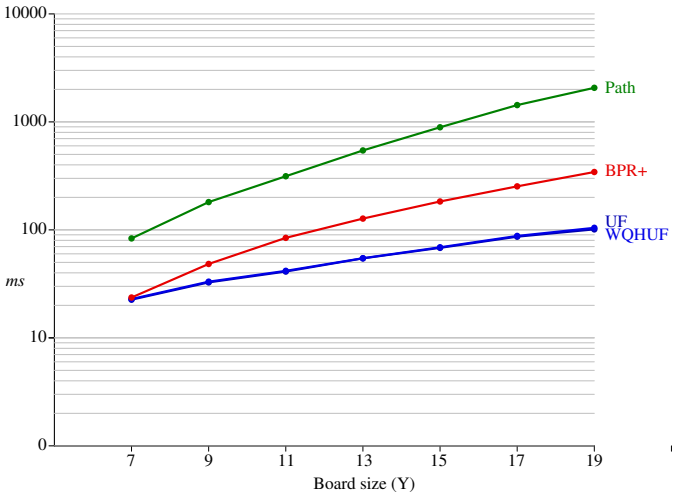

Fig. 12. Milliseconds per 10,000 trials, per-move (Y).

6.2 Per-Move Win Tests

Table 3 shows timings for the same set of games, but with win testing applied after every move. The mean times for each run of 10,000 trials, over 1,000 runs, are shown in milliseconds for easy comparison, along with their 95% confidence interval.

This time, however, it can be seen that BPR methods are around 2-3 times slower than union find methods when win tests are performed after each move, with path-following performing worse than either of these methods. These results are shown graphically in Figures 11 (Hex) and 12 (Y).

6.3 Discussion

For per-game win testing, BPR clearly outperformed the other connection test methods. BPR consistently achieved 25-30 times as many win tests for Hex as the best union find (UF) method in the same amount of time, and around 50 times as many win tests for Y. This suggests that playout rates for Monte Carlo approaches to Hex and Y might be greatly increased using BPR for win testing on a per-game basis. Path following also outperformed UF methods for per-game win tests.

The non-destructive BPR algorithm unexpectedly proved slightly faster than the basic BPR algorithm (Listing 1). This may be due to caching of the copied items, the Java compiler optimising the partially unravelled inner loop,[5] or the fact that the non-destructive version reads from one memory location but writes to another on the first (longest) reduction pass, rather than reading and writing to the same location over all passes.

For per-move win testing, the UF methods offer the best performance of those tested, with iterative BPR (BPR+) being around 2-3 times slower on average. Path following offers much worse performance than the other methods for per-move win testing. Iterative BPR+ was noticeably faster than non-destructive BPR for per-move win testing.

Regarding the UF methods, WQHUF generally outperformed standard UF by a slight margin in the per-game win tests, as expected. However, WQHUF showed almost no improvement over standard UF in the per-move win tests, and was in fact marginally slower for all per-move win tests for the game of Y (Table 3, lower half). This may be due to the nature of the games being modelled, as the board sizes and number of elements involved are small compared to other graph connectivity problems, and most time is spent merging single or very

5. The first (longest) reduction pass is performed separately.

small groups of pieces. The overhead of the optimisations that make up WQHUF appear to outweigh the benefits for these problems of such low complexity.

The UF methods show similar times for per-move and per-game win testing, especially for Hex. The actual UF win test itself is trivially fast (a quick lookup of the two target regions), but these consistent timings reflect the incremental connection updates that must be performed with each move for this lookup to have any meaning, for both the per-move and per-game cases.

7 Conclusion

We demonstrate how a class of hexagonally-based connection games can be phrased as the game of Y using appropriate board templates, and how connection (win) testing can be achieved for such games using a known method called Y reduction. We introduce the bitwise-parallel reduction (BPR) algorithm for performing such win tests efficiently using a bitwise-parallel approach, and compare it with existing methods for win testing.

The BPR algorithm is simple to implement, and allows per-game win testing to be performed many times faster than known methods for games such as Hex and Y. It is ideally suited to Monte Carlo tree search approaches that only need to perform win tests once per game following each board fill, and has the potential to increase playout rates, especially for parallel implementations in which many board fills must be evaluated quickly.

Future work might involve the extension of similar bitwise-parallel approaches to dead cell analysis (for move pruning), and for incorporating domain knowledge (such as bridge intrusion/repair) during playouts. We would also like to investigate the possible application of BPR methods to games on non-trivalent grids. One of the anonymous reviewers suggests that BPR might be applied to an 11×11 Hex board packed into two rather than three 64-bit longs, for efficiency, if the effect of the reduction template is instead hard-coded into the first reduction pass. This is another idea worth exploring.

Acknowledgements

The authors would like to thank the editor and anonymous reviewers for their suggestions, and Professor Ryan Hayward and Greg Schmidt for discussions on the topic.

References

[1] C. Browne, *Connection Games: Variations on a Theme*. Natick, Massachusetts: AK Peters, 2005.
[2] A. Rimmel, O. Teytaud, C.-S. Lee, S.-J. Yen, M.-H. Wang, and S.-R. Tsai, "Current Frontiers in Computer Go," *IEEE Trans. Comp. Intell. AI Games*, vol. 2, no. 4, pp. 229–238, 2010.
[3] T. Raiko and J. Peltonen, "Application of UCT Search to the Connection Games of Hex, Y, *Star, and Renkula!," in *Proc. Finn. Artif. Intell. Conf.*, (Espoo, Finland), 2008.
[4] B. Arneson, R. B. Hayward, and P. Henderson, "Monte Carlo Tree Search in Hex," *IEEE Trans. Comp. Intell. AI Games*, vol. 2, no. 4, pp. 251–258, 2010.
[5] P. Henderson, B. Arneson, and R. B. Hayward, "Hex, braids, the crossing rule, and XH-search," in *Proc. Adv. Comput. Games, LNCS 6048*, (Pamplona, Spain), pp. 88–98, Springer, 2010.
[6] C. Browne, *Hex Strategy: Making the Right Connections*. Natick, Massachusetts: AK Peters, 2000.
[7] C. Schensted and C. Titus, *Mudcrack & Poly-Y*. Peaks Island, Maine: NEO Press, 1975.
[8] P. Hein, "Polygon," *Politiken*, December 26 1942.
[9] G. M. J.-B. Chaslot, C. Fiter, J.-B. Hoock, A. Rimmel, and O. Teytaud, "Adding Expert Knowledge and Exploration in Monte-Carlo Tree Search," in *Proc. Adv. Comput. Games, LNCS 6048*, vol. 6048, (Pamplona, Spain), pp. 1–13, 2010.
[10] R. Sedgewick and K. Wayne, *Algorithms*. Boston, Massachusetts: Addison Wesley, fourth ed., 2011.
[11] J. van Rijswijck, "Search and evaluation in Hex," tech. rep., Univ. Alberta, Edmonton, 2002.
[12] S. Meyers, "Personal correspondence," 2002.
[13] G. M. Adelson-Velskiy, V. L. Arlazarov, A. R. Bitman, A. A. Zhivotovsky, and A. V. Uskov, "Programming a Computer to Play Chess," *Russian Math. Surveys*, vol. 25, pp. 221–262, 1970.
[14] B. M. Ledvina, M. L. Psiaki, S. P. Powell, and P. M. Kintner, "Bit-Wise Parallel Algorithms for Efficient Software Correlation Applied to a GPS Software Receiver," *IEEE Trans. Wireless Commun.*, vol. 3, pp. 1469–1473, 2004.

Cameron Browne received the Ph.D. degree in Computer Science from QUT, Brisbane, Australia, in 2008, winning the Dean's Award for Outstanding Thesis.

He has worked for several technology companies and was Canon Research Australia's Inventor of the Year for 1998. He has designed dozens of board games, many published, and is the author of the books *Hex Strategy*, *Connection Games* and *Evolutionary Game Design*.

Dr. Browne is a Research Fellow in the Computational Creativity Group at Imperial College, London. He is currently working on the three-year EPSRC project *UCT for Games and Beyond*, investigating Monte Carlo tree search (MCTS) methods for procedural content generation in creative domains such as game design.

Stephen Tavener received a B.Sc. from Queen Mary and Westfield College, London, in 1989.

He is currently pursuing a Ph.D. in the Computational Creativity Group, Imperial College, London, on benevolence in artificial agents. He has worked for the BBC and Zillions of Games, reviewed for major abstract games magazines, and run the London-based retail shop Gamesale. His interests include board game design, fractal geometry and program optimisation.