コンピュータ将棋における Magic Bitboard の提案と実装

山本一成[†] 竹内聖悟[†]
金子知適[†] 田中哲朗[††]

本稿では，将棋において Magic Bitboard を適用する手法を提案する．Bitboard はチェスなどの二人ゲームのゲーム木探索で盤面を表現するのに適した手法であり，チェスや将棋の強いプログラムで広く使われている．Magic Bitboard は従来の Bitboard の手法に比べ，よりシンプルなデータ構造を管理するだけで利きを算出することが可能となる近年開発された手法である．Magic Bitboard はチェスの盤面が一つの 64 ビット整数で表現できることに依存した手法であり，盤面のマス目の数が 81 マスの将棋で，Magic Bitboard を使う方法は知られていなかった．しかし我々は初めて，複数の整数で表現された盤面において Magic Bitboard を使って利きを算出する手法を発明し，Bonanza を使った実験で我々の手法の効果を示した．

Proposal and Implementation of Magic Bitboards in Shogi

Issei Yamamoto,[†] Shogo Takeuchi,[†] Tomoyuki Kaneko [†]
and Tetsuro Tanaka[††]

In this paper, we present a technique to apply magic bitboards into Shogi. A bitboard is a bitset representation of a position suitable for efficient game-tree searches in two-player games such as chess. They have widely been used in popular strong programs in chess and Shogi. Magic bitboards are recent improvements that can efficiently calculate attacks with simpler data than those needed to be maintained in previous bitboard techniques. While magic bitboards in chess depend on the fact that a chess board can be represented in one 64-bit integer, a board of Shogi has 81 squares. Thus, we first developed a technique to calculate attacks of a board represented in multiple integers. Then, we show the effectiveness of our techniques by experiments using the state-of-the-art Shogi program, Bonanza.

1. はじめに

コンピュータチェスやコンピュータ将棋において高速に局面を探索できることは，強さの向上を図る上で非常に重要である．高速な探索の実現には局面更新や利きの算出の高速化が重要である．それらの要求に答える手法のひとつが，Bitboard というテクニックであり，コンピュータチェスやコンピュータ将棋で広く使われている．Bitboard とは盤面の 1 マスを 1 ビットとして駒や利きのあるなしをビットが立っているかどうかで表現したものである．Bitboard は論理演算，四則演算を利用することで比較的高速に利きの算出や局面の更新ができる．近年コンピュータチェスの分野で，従来手法である Rotated Bitboard よりも簡潔に利きの算出ができる Magic Bitboard という手法が注目されている．しかしチェスにおける Magic Bitboard は盤面が 8×8 の 64 マスであることを利用した手法であり，9×9 の 81 マスの盤面を持つ将棋では Magic Bitboard を適用する方法が知られていなかった．本稿では，将棋に Magic Bitboard を適用する新たな手法を提案し，将棋プログラム Bonanza にその手法を実装して従来手法の Rotated Bitboard との比較実験を行い，有効性を示した．

2. Bitboard

チェスの場合，盤面が 8×8 の 64 マスあるので，64 ビット整数を Bitboard として使用する．将棋の場合には，盤面には 9×9 の 81 マスが存在するため，81 ビット必要となるので複数の整数を使用するのが一般的である．たとえば Bonanza[☆] では図 1 のように，32 ビット整数を 3 つ並べることで局面を表現した．

Bitboard を使って利きを算出する方法を考える．チェスの Knight や将棋の金といった駒はその駒の位置のみで利き

† 東京大学大学院総合文化研究科
Department of General Systems Studies, Graduate School of Arts and Sciences, The University of Tokyo
{issei,takeuchi,kaneko}@graco.c.u-tokyo.ac.jp
†† 東京大学情報基盤センター
Information Technology Center, The University of Tokyo
ktanaka@ecc.u-tokyo.ac.jp

☆ http://www.geocities.jp/bonanza_shogi/

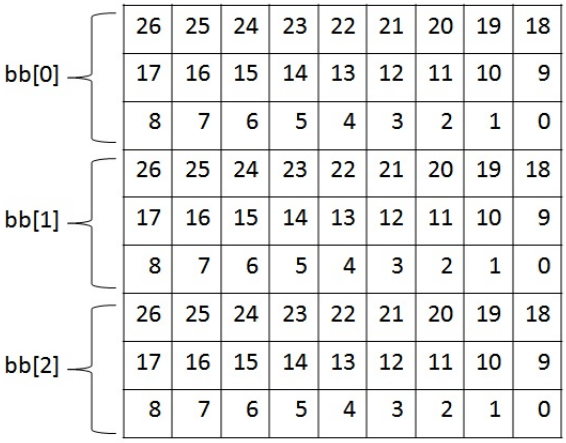

図 1 Bonanza における Bitboard の配置

が算出できる．しかし一定方向に空白が続く範囲で何マスでも移動できるチェスの Rook や将棋の飛車などの駒（以下遠隔駒と呼ぶ）の利きは，簡単に算出できない．なぜなら，他の駒の配置によって利きが変化するからである．遠隔駒の利きを高速で調べることは，利きの算出において重要なテーマである．

Bitboard を使って遠隔駒の一種、飛車の横の利きを求める手順の概要を図 3 に掲載し，手順を説明する．事前に，横一列の駒の配置によって，飛車の横の利きがどのような利きになるかを算出した配列を準備しておく．そして，駒の利きが及ぶ可能性のある一列に駒があるかないかの状態を整数として解釈して，その整数をキーとして配列を参照して利きを求める．なお，今注目している駒が利きを持つ可能性があるすべてのマス目に対して、各マスの駒の有無を整数で表現することを以降は符号化と呼ぶ．

符号化の方法を，八段目に飛車がいて，かつその横利きを調べる場合を例にとって説明する．図 2 のように駒の有無を表す Bitboard が作られているとする．なお，この Bitboard は以下 Occupied Bitboard と呼ぶ．

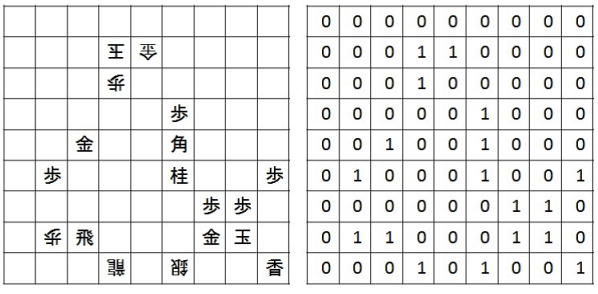

図 2 Occupied Bitboard の例

この例では飛車の横利きに関係がある Occupied Bitboard の 8 八から 2 八の地点のビットに関心がある．そこで，Occupied Bitboard を 10 ビット右シフト後に，下位 7 ビットをマスクすることで，8 八から 2 八までの駒の有無を 2^7 通りに符号化することができる．なお，1 八や 9 八の地点のビットに関心がない理由は，そのマスの駒の有無は，飛車の横利きには影響を与えないからである．

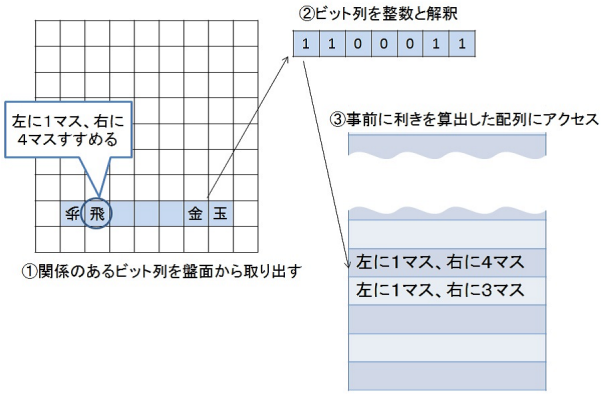

図 3 Bitboard での利きの算出手順

3. 関 連 研 究

3.1 Bitboard と配列の実装の比較

コンピュータチェスやコンピュータ将棋において，Bitboard に代わる盤面の表現方法としては配列を用いた表現方法も挙げられる．この方法はチェスにおいては Fruit☆，将棋においては YSS☆☆，GPS 将棋☆☆☆などが挙げられる．またこの方法の中にも，利き情報を持っているタイプ，持っていないタイプという 2 種類の方法が存在する．利き情報を持つ表現方法は利きの算出は高速であるが，盤面更新が低速で，実装も難しくなる傾向がある．利き情報を持っていない表現方法は盤面更新が高速である．しかし利きの算出は，マス目が空白であるかどうかを一つずつ調べる必要があるので低速である．配列の実装と比較して，Bitboard を使った表現方法は盤面更新は高速でかつ，利きの算出も比較的高速で実行できることがメリットに挙げられる．それぞれの特徴を表 1 にまとめる．

表 1 配列（利きあり，なし），Bitboard の比較

	利きあり	利きなし	Bitboard
盤面更新	低速	高速	高速
利きの算出	高速	低速	中速

☆ http://www.fruitchess.com/
☆☆ http://www32.ocn.ne.jp/~yss/index_j.html
☆☆☆ http://gps.tanaka.ecc.u-tokyo.ac.jp/gpsshogi/

3.2 Rotated Bitboard

2 章において Bitboard を用いて遠隔駒の横の利きを算出する方法を示した．しかし，縦や斜めの利きに関してはこのように単純なシフトとマスクでは求められない．Rotated Bitboard[1),2)] と呼ばれる手法では，補助的なデータを用いて縦や斜めの利きも単純なシフトとマスク処理で算出する．具体的には 90°回転，±45°回転させた Occupied Bitboard を局面と一緒に管理する．図 4 のように Occupied Bitboard を 90°回転させれば，縦の利きは横の利きと同様の方法で求めることが可能となる．斜めの利きに関しても同様である．しかし回転させた複数の Occupied Bitboard を持つと，局面更新の処理が増加し，実装も複雑になるという問題がある．

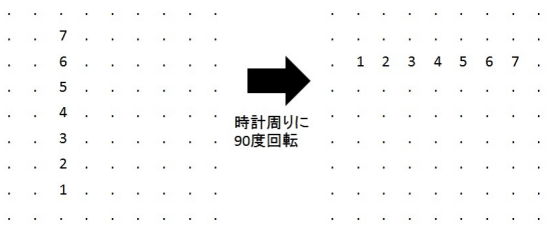

図 4 90°回転させた Occupied Bitboard

3.3 Magic Bitboard

Magic Bitboard は 90°回転，±45°回転させた Occupied Bitboard を使わないで遠隔駒の利きを算出する方法である． Magic Bitboard で利きを求める手順を説明する．大枠は図 3 と同じだが，関係するマスの状態の符号化の手法が Rotated Bitboard とは異なり，マスク，乗算，右シフトを用いる．

具体例として，図 5 のように右下に Rook がいる局面での符号化の方法を説明をする．まず Rook にとって関係があるマス目，12 ビット以外がすべて 0 になるように Occupied Bitboard をマスクする．次に右下の Rook の対応する適切なマジックナンバーをかけ，最後に (64 - 12) ビット右シフトを行う．この符号化した数と関係のあるマス目の状態が一対一の写像になっているので，利きの状態が衝突することなく配列に収められることが可能となる．

この Magic Bitboard[☆1] は StockFish[☆2]，Glaurung[☆3] といった近年現れた強豪チェスソフトウェアで採用されている手法である．Magic Bitboard は利きの算出が一枚の Occupied Bitboard のみで実現可能なので，局面更新の処理が Rotated BitBoard より簡潔で高速である．また，複数の筋の利きが一度の計算で求まるという利点もある．一方 Rotated Bitboard の場合は一方向ずつしか算出できない．それぞれの特徴を表 2 にまとめた．

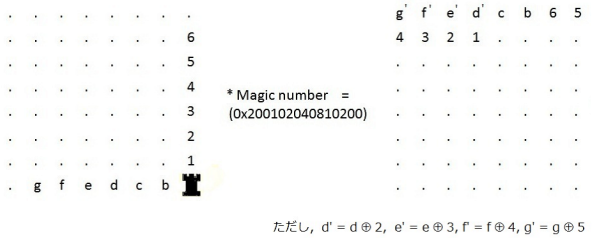

図 5 チェスの場合の Magic Bitboard

表 2 Rotated BitBoard と Magic Bitboard の比較

	Rotated	Magic
利きの算出	一方向ずつ	全方向
Occupied Bitboard	4 枚	1 枚

4. Magic Bitboard の将棋における実装

4.1 将棋の場合の利きの算出

本節ではコンピュータ将棋において，Magic Bitboard を利用する方法を提案する．将棋における Bitboard は複数の整数の組み合わせで表現されているので，チェスのように一つの整数にマジックナンバーで乗算して，シフトするという方法での符号化は不可能である．

そこで新たな方法を提案する（Algorithm 1）．事前準備として図 1 の盤面の表現における bb[0] と bb[1] をひとつの 64 ビット整数型とみなし，以下，これを q と定義する[☆4]．ビットのレイアウトは図 6 のようになる．

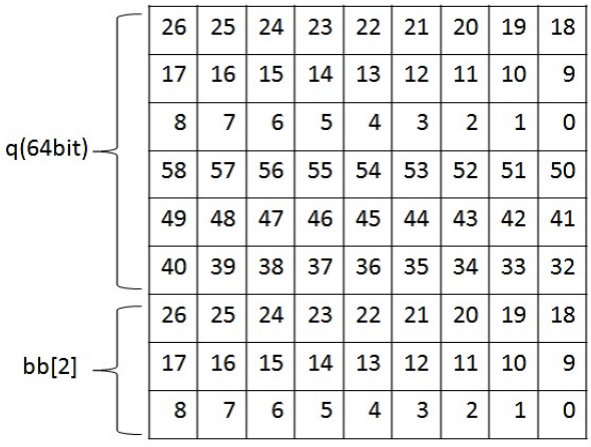

図 6 q を定義した場合の Bonanza における Bitboard の配置

☆1 http://chessprogramming.wikispaces.com/Magic+Bitboards
☆2 http://www.stockfishchess.com/
☆3 http://www.glaurungchess.com/

☆4 C 言語の union の機能を使って実装した

まず利きの状態に関係があるマス目以外が，すべて 0 になるように Occupied Bitboard の q と bb[2] にマスクをする．次に適切なマジックナンバー二つをそれぞれ q と bb[2] にかけ，かけた結果の二つの数を XOR して，最後に右シフトを行う．具体例として，5 五にいる角の利きの算出を図 7 に示す．

Algorithm 1 提案手法による符号化

(1) Bitboard の bb[0] と bb[1] を q，加えて b[2] をそれぞれ 64 ビット整数だと解釈
(2) 利きに関係しないマス目が 0 になるように Occupied Bitboard にマスク処理をする
(3) それぞれの数に事前に準備された適切な 64 ビットのマジックナンバーをかける
(4) 掛け算を行った 2 つの数の XOR を求める
(5) 利きに関係のあるマス目のビットの数だけ残るように右シフトを行う

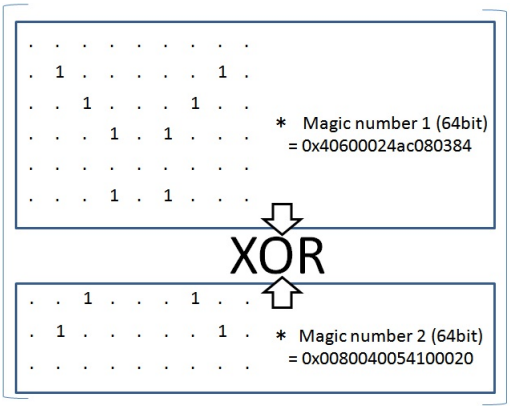

図 7 提案手法の Magic Bitboard

4.2 マジックナンバー表の作成

提案手法が機能するには，符号化した数と関係のあるマス目の状態が一対一の写像となる，適切なマジックナンバーが必要である．そこで，Algorithm 2 に掲載した手順を使い，実際にすべてのマス目に関する飛車，角の適切なマジックナンバーを求めることに成功した．チェスの事例では，Algorithm 2 で使用する適切なマジックナンバーの候補は一様分布で生成される乱数 3 つのビット論理積を用いる☆．将棋においては，何回論理積をとるのが適切なマジックナンバーの候補となるかは未知であった．そこで論理積をとる回数を変えたいくつかの乱数で，1 九飛，5 五飛，5 五角の利きはそれぞれ何回試行すれば求まるのかの予備実験を行った．表 3 は結果を表している．論理積をとった回数が 1 回および 6 回のものは 1,000,000 回試行しても適切なマジックナンバーを算出することが出来なかった．またチェスの事例と同様に 3 つのビット論理積を用いたものが，もっとも良い結果であった．そこで 3 つのビット論理積を用いたものをマジックナンバーの候補として以降は使用した．なおすべてのマスに関する飛車，角の適切なマジックナンバーの算出にかかった計算時間は Core i7 950 上の windows マシンで数分であった．

Algorithm 2 マジックナンバーの求め方

(1) 駒の種類ごとに，その駒が存在しうる盤上の全てのマスについて以下の処理を行う
(2) 対象の駒と位置での利きについて，関係するマス目の駒の有無のすべてのパターンを列挙する
(3) マジックナンバーの候補として 64 ビット整数を二つ作成する
(乱数は一様分布で生成される整数 3 つのビット論理積を用いた)
(4) （2）で列挙した全てのパターンに対して，作成した候補で配列を作成した場合に衝突がないかどうかを確認する．衝突がなければ，適切なマジックナンバーである．衝突があれば（3）に戻る

表 3 マジックナンバーの生成にかかった試行回数

	1 九飛	5 五飛	5 五角
論理積 1 回	>1,000,000	>1,000,000	>1,000,000
論理積 2 回	189,704	116,971	60,249
論理積 3 回	11,288	15,961	7,898
論理積 4 回	16,164	41,504	356,239
論理積 5 回	867,446	552,592	872,409
論理積 6 回	>1,000,000	>1,000,000	>1,000,000

5. Bonanza を利用した実験

将棋においても Magic Bitboard が有効であることを示すために，Rotated BitBoard を採用している Bonanza☆☆を使用して比較実験を行った．なお Bonanza では局面を表すデータ構造として，表 4 にある Bitboard を持っている．

本実験では変更なしの Bonanza（以下 Original と呼ぶ）と，Magic Bitboard に変更した実装で性能の比較を行った．Magic Bitboard は十字または斜めの十字の利きを算出するが，Bonanza では，一方向の利きのみしか用いない場合も多い．基本的には十字の利きを算出した後に必要な方向のみを取り出すことで対応したが，使用回数の多かった縦方向に関しては次の三つの手法を試した．Simple は十

☆ 整数 3 つのビット論理積を用いる方法は Tord Romstad 氏の提案

☆☆ 使用したバージョンは Bonanza Feliz 0.0 である

表 4 Bonanza の局面のデータ構造として持っている Bitboard

すべての種類ごと (24 種類) の駒の有無を示す Bitboard
先手の駒の有無を示す Bitboard (Occupiued Bitboard)
後手の駒の有無を示す Bitboard (Occupiued Bitboard)
駒の有無を示す 90°回転した Bitboard (Occupiued Bitboard)
駒の有無を示す時計周り 45°回転した Bitboard (Occupiued Bitboard)
駒の有無を示す反時計回り 45°回転した Bitboard (Occupiued Bitboard)
先手の玉，馬，龍の有無を示す Bitboard
後手の玉，馬，龍の有無を示す Bitboard
先手の金，もしくは金と同じ動きをする駒の有無を示す Bitboard
後手の金，もしくは金と同じ動きをする駒の有無を示す Bitboard
先手のすべての歩の利きを示す Bitboard
後手のすべての歩の利きを示す Bitboard

字の利きを Magic Bitboard で算出したあと，縦の利きを出すマスク処理をする．FileM は直接縦の利きを算出する Magic Bitboard をもつ，FileR は縦の利きだけは従来手法の Rotated Bitboard で算出する．

表 5 は各実装の状況を一覧にしたものである．Occupied Bitboard は Occupied Bitboard の枚数を表す．XorFile，SetClearFile，XorDiag1 と 2，SetClearDiag1 と 2 はマクロの状態を表して, 有効なら「○」に無効なら「×」になっている．AttackRook，AttackBishop，AttackFile，AttackRank，AttackDiag1 と 2，BishopAttack0 と 1 と 2 はそれぞれどのように利きを算出したかを表しいる．「R」は Rotated Bitboard を使用して算出している,「M」は Magic Bitboard を使用して算出している．「MM」は Magic Bitboard で算出した後に，マスク処理を行って算出していることを表している．

表 5 各実装の状況

	Original	Simple	FileM	FileR
Occupied Bitboard	4	1	1	2
XorFile	○	×	×	○
SetClearFile	○	×	×	○
XorDaig1,2	○	×	×	×
SetClearDiag1,2	○	×	×	×
AttackRook	R	M	M	M
AttackBishop	R	M	M	M
AttackFile	R	MM	M	R
AttackRank	R	R	R	R
AttackDiag1,2	R	MM	MM	MM
BishiopAttack0,1,2	R	M	M	M

性能を比較するため様々な局面を探索させ，その速度を測った．結果を表 6，7 にまとめた．NPS は 1 秒間に探索したノード数，Time は問題集を解答するのにかかった秒数，速度比は Original との NPS 比を取っている．実験環境は Core i7 950 上の Windows 7 で，MSC コンパイラ☆を用いた．使用した局面は平手初期局面と問題集として「らくらく次の一手」[3] 合計 216 問を使用した．探索の深さは平手初期局面では深さ 15，らくらく次の一手では深さ 9 とした．測定は合計三回行い，平均値を求めた．

表 6 初期局面での比較（深さ 15 で探索）

初期局面	Original	Simple	FileM	FileR
NPS	92,769	86,146	87,543	90,240
Time	148.00	159.33	157.00	152.00
速度比	100.00%	92.89%	94.27%	97.37%

表 7 「らくらく次の一手」での比較（深さ 9 で探索）

らくらく	Original	Simple	FileM	FileR
NPS	128,159	112,130	118,957	123,762
Time	362.22	414.00	390.24	375.09
速度比	100.00%	87.49%	92.81%	96.56%

探索速度は Original がもっとも高速で，続いて FileR，FileM，Simple の順となった．また，以下で述べることから，縦の利きを求める処理の重要性が読み取れる．FileR は 90°回転した Rotated BitBoard を余分に持つ点で Simple より盤面更新の処理が増えるが，総合的な速度は Simple より速く Original に近い．そして，縦方向に特化した処理を行う FileM も Simple より高速である．このような縦の利きの重要性の背景とそれを踏まえた高速化のアイデアは 6 章で述べる．

Magic Bitboard 版は Original の速度にはわずかに達しなかったが，その理由の一つとしては Bonanza が Rotated Bitboard の機能に合わせて作られている点がある．具体例を挙げ考察する．図 8 は Bonanza のソースコードの一部である．b_gen_checks() 関数の中ではマクロの AttackDiag1() と AttackDiag2() が呼ばれている．なお，AttackDiag1，2 は一方向だけの斜めの利きを算出するマクロである．Magic Bitboard を使用した現在の実装では Magic Bitboard ですべての斜めの利きを算出した後，わざわざ一方向だけになるようにマスク処理を行っている．この方法はすべての斜めの利きを 2 度出している点で，明らかに重複が存在する．もともと Magic Bitboard の実装が前提であれば，ここでは AttackDiag1()，AttackDiag2() とマクロを呼び出したりせず，一度だけ AttackBishop() を呼ぶ実装となるだろう．マクロの置換だけではなく，Magic Bitboard の性質を理解した上で，実装を行えばさらなる高速化が見込めると予想される．

☆ 32 ビットの実行ファイルとしてコンパイルをした
コンパイルオプションは Visual Studio 2010 のデフォルトオプションに /DNDEBUG を加えたもの

```
unsigned int *
b_gen_checks( tree_t * restrict __ptree__, unsigned
int * restrict pmove )
{
  /*中略*/
  bb_diag1_chk = AttackDiag1( sq_wk );
  bb_diag2_chk = AttackDiag2( sq_wk );
```

図 8 Bonanza の genchk.c 内の b_gen_checks() 関数

Magic Bitboard の真の性能を測るためには，同時に複数の筋の利きを算出する利点を活かしたプログラムを作成するか，Bonanza に大きな改良を施す必要があり，現状では難しい．しかし，現時点でも Original と FileR との性能がすでにかなり近いことから，Magic Bitboard は将棋プログラムの基礎的技術として大いに役に立つ可能性があると著者らは考えている．

6. 将棋における Magic Bitboard の高速化のアイデア

将棋で縦の利きを求める処理が多い理由は，前のみに進む香車という遠隔駒が 4 枚存在するためである．そして縦の利きのみを求める計算は Magic Bitboard より Rotated Bitboard の方が速い．なぜなら一方向の利きを求めるだけならば，Rotated Bitboard のほうが少ない演算で算出できるからである．そのため FileR では Rotated Bitboard を一部併用して Bitboard を 2 枚持つことで速度を向上させたが，そのアイデアを進めて，Bitboard のレイアウトを図 1 のような横方向から図 9 のような縦方向に変更することを提案する．

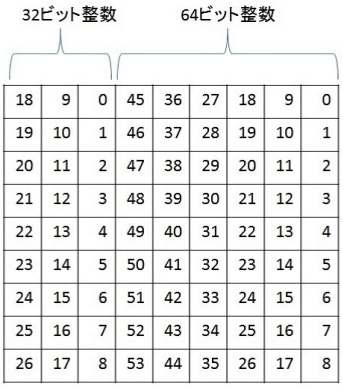

図 9 縦方向を基本とする Bitboard

これにより，縦方向の利きを Rotated Bitboard と同様に求めつつ，他の利きは Magic Bitboard の手法で求めることが，Occupied Bitboard 一枚で実現可能となる．なお，縦方向の変更した Bitboard を用いてもすべてのマス目に関して飛車と角の利きを算出できるマジックナンバーが存在することは実験による確認が終わっている．

また Bitboard を縦方向にすることで遠隔駒でない駒および香車の利きの算出が 6，7 筋以外ではひとつの整数型にのみ注目すればよくなるといった効果もある．図 10 を例にとって考える．図 10 のように 5 五の地点に利く遠隔駒でない駒，もしくは香車は必ず 4，5，6 筋に存在する．そのため図 9 の右側の 64 ビット整数のみを注目すれば，遠隔駒でない駒および香車の利きの算出は可能となる．なおこれは，桂馬や香車がいるので，横向きの Bitboard では実現できない．

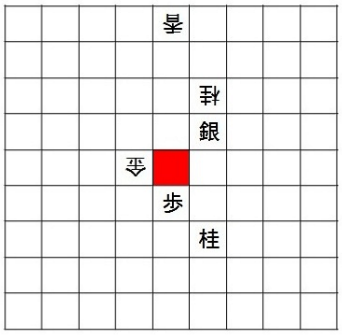

図 10 5 五の地点に利く，駒の一例

加えて Bonanza ではすべての歩の利きを状態を持っている Bitboard を持っていたが，これも歩の Bitboard をそれぞれの整数を 1 ビットだけシフトすれば簡単に算出できるので，盤面の更新時に負担をかけずに高速に歩の利きの算出が行える．図 11 のように先手の歩を例にとって考える．先手の歩は利きはちょうど 1 ビット右シフトした先にあることが図 11 より分かる．この方法は一段目に先手の歩は決していないという将棋特有の条件を利用している．これも横向きの Bitboard では実現できない．

一方 Bitboard を縦方向にすることで，Bonanza の手生成で使われる自陣や敵陣を Bitboard 内の整数の位置で判断するというテクニックが使えなくなるデメリットもある．しかし著者はこのデメリットを考慮しても，Bitboard を縦方向にすることのメリットは大きいと考えており，現在 Bitboard を縦方向にした実装を作成中である．

7. おわりに

Magic Bitboard は，近年コンピュータチェスで有望視されている新しい Bitboard のテクニックである．しかし，こ

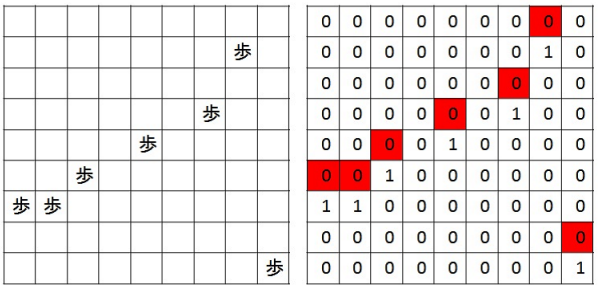

図 11 先手の歩の利きの算出

れまで 81 マスの盤面を持つ将棋において Magic Bitboard を活用する方法は知られていなかった．本稿では Magic Bitboard を将棋に適用する方法を提案し，Bonanza を題材にして，提案手法が実用に足ることを示した．

Rotated Bitboard を前提に書かれた Bonanza を Magic Bitboard に変更をした場合でも，その性能は実用的な速度で動作するものであった．Bonanza では Magic Bitboard の利点は十分活かすことが難しいこと．Bonanza の Bitboard のレイアウトに工夫の余地があること．そして Magic Bitboard を使用した場合，より簡潔なデータ構造で利きを算出できること．これらの理由から Magic Bitboard は将棋プログラムにとって Rotated Bitboard と同等な，あるいはより適した基礎技術となると考えられる．

なお本稿で作成されたマジックナンバーは http://www.graco.c.u-tokyo.ac.jp/~issei/magicnumber.txt に公開されているので，活用されたい．

参 考 文 献

1) Hyatt, R. M.(1999).Rotated Bitmaps, a New Twist on an Old Idea.*ICCA Journal, Vol. 22, No. 4, pp. 213–222.*
2) R. Grimbergen.(2007). Using Bitboards for Move Generation in Shogi.*ICGA Journal Vol. 30, No. 1, pp. 25-34.*
3) 日本将棋連盟編集 (2002).ラクラク次の一手一基本手筋集.日本将棋連盟書籍.