A Method to Construct Knowledge Table-base in K-in-a-row Games

Chang-ming XU, Z.M. MA, Xin-he XU
Northeastern University
110004, Shenyang, Liaoning Province, P.R.China
Fax:+86-24-83681823, Phone:+86-24-83681582
changmingxu@gmail.com, mazongmin@ise.neu.edu.cn, xuxinhe@ise.neu.edu.cn

ABSTRACT
In any *k*-in-a-row game, the player should always analyze each consecutive sequence in 4 directions, which consists of either the void intersections or the intersections occupied by the same color stones. Although the difficult problem in *k*-in-a-row game is that any void intersection on board can be placed a stone on, just like Go, however, the hints in *k*-in-a-row are far more than those in Go. We find it is a good method to decrease the complexity of the *k*-in-a-row games by using Connection to represent the states of the game position. Then, a precise classification criterion for Connection, as well as a precise classification for the intersections is given. As the high-level knowledge of games seems hard to be acquired, the program designers always resort to human masters. However, we can construct a good knowledge table-base based on above idea without masters.

Categories and Subject Descriptors
I.2.4 [**Knowledge Representation Formalisms and Methods**]: *Representations (procedural and rule-based)*

General Terms
Performance, Experimentation.

Keywords
Computer games, *k*-in-a-row, Knowledge Table-base.

1. INTRODUCTION
Connect(*m*, *n*, *k*, *p*, *q*) denotes a family games of *k*-in-a-row. There are two players, the black and the white, in a connect(*m*, *n*, *k*, *p*, *q*) game. The first player, always the black side, places *q* stones for the first move. Then, two players alternately place *p* stones on *m*×*n* board in each turn. The player who gets *k* consecutive stones first win. A very interesting *k*-in-a-row game is connect(*m*, *n*, 6, 2, 1), Connect6, is first introduced in [1][2]. In this paper, we will try to show how to construct a knowledge table-base for *k*-in-a-row games. The prominent characters distinguishing Connect6 from other traditional *k*-in-a-row games are its potential fairness and the simplicity of rule, which is discussed in [3]. Connect6 is more complex than all the solved games, and its board size is unrestrained. The states space complexity is 10^{172}, and the game tree complexity is 10^{140} for connect(19, 19, 6, 2, 1). The discussion about the complexity of computer games is introduced in [4] [5].

Although the method in this paper is applicable to most *k*-in-a-row games, examples are only suitable to Connect6. The rest of this paper is organized as follows. In section 2 we introduce Connection which is the most important concept in our work. In section 3 a series of religious definitions of Connection types are given. In section 4 we evaluate all the types of Connection, and discuss how to construct a knowledge table-base for each shape of all the Connections. In section 5 each type of the intersections is well defined and a more complex knowledge table-base is presented. At last, we give conclusions in section 6.

SAC'09, March 8-12, 2009, Honolulu, Hawaii, U.S.A.

2. CONNECTION IN K-IN-A-ROW
To analyze a game position efficiently, we proposed a novel method originally that regards a straight line on board as several segments, each of which is called *Connection*. The concept, Connection, is a foundation to our work.

2.1 Definition of Connection

168	167	166	165	164	163	162	161	160	159	158	157	156		
155	154	153	152	151	150	149	148	147	146	145	144	143		
142	141	140	139	138	137	136	135	134	133	132	131	130		
129	128	127	126	125	124	123	122	121	120	119	118	117		
116	115	114	113	112	111	110	109	108	107	106	105	104		
103	102	101	100	099	098	097	096	095	094	093	092	091		
090	089	088	087	086	085	084	083	082	081	080	079	078		
077	076	075	074	073	072	071	070	069	068	067	066	065		
064	063	062	061	060	059	058	057	056	055	054	053	052		
051	050	049	048	047	046	045	044	043	042	041	040	039		
038	037	036	035	034	033	032	031	030	029	028	027	026		
025	024	023	022	021	020	019	018	017	016	015	014	013		
012	011	010	009	008	007	006	005	004	003	002	001	000		

Figure 1. Encoding intersections on 13×13 board

To represent a game position state, the most popular method is to use an array. Each element of the array has a value to represent the state (void, black stone, or white stone) of the corresponding intersection on board. We call above method as IB (Intersection Based) method, shown in Figure 1. Although it is an intelligible

way to describe the state of game position in a program, the implementation always results in inefficient.

In the games of *k*-in-a-row, the direct relations only exist among the stones in a straight line. For example, in Figure 2, there are 3 segments marked with ①, ②, and ③ respectively, each of which consists of either intersections occupied by the same color stones or void intersections.

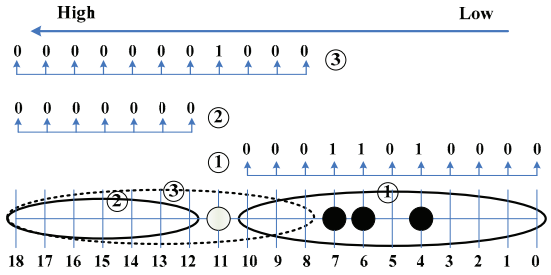

Figure 2. Connections and its binary representation

We use Connection to describe the segments shown in Figure 2 and to record the state of *k*-in-a-row game position. Here, we named our new method as CB (Connection Based) method. Contrary to IB, the merits of CB are: retaining the natural relation among stones and getting rid of the redundant of the description for the game position state.

Definition 1 (Connection). On *n*×*n* *k*-in-a-row board, a Connection is a maximal consecutive sequence, which consists of either void intersections or intersections occupied by stones in the same color in a straight line. Generally, *c* is used to represent a Connection in this paper.

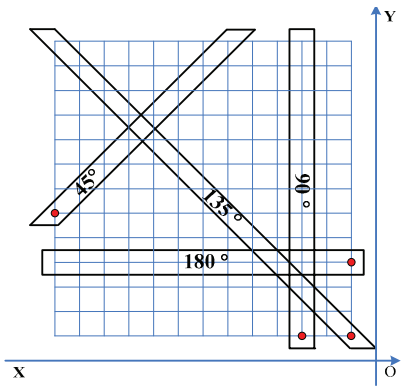

Figure 3. Directions of Connections on 13×13 board

Definition 2 (Properties of Connection). On *n*×*n* *k*-in-a-row board, the properties of Connection are:

(1) The head of Connection. Designate the intersection with the smallest serial number as the head of Connection. In Figure 3 the intersection marked by a solid dot indicates that the intersection closest to it will be selected as the head of Connection in a certain line. (2) The direction of Connection. Designate 4 directions for Connections on the board: 45°, 90°, 135° and 180°. As an exception, the direction of Connection in an isolated line is always from right to left, e.g. Connections in Figure 2. (3) The length of Connection. The number of intersections in *c* is defined as the length of *c*, written as $|c|$. If $|c|<k$, *c* is useless according to the game rules. So, we assume that a sequence *c* is a Connection implies $|c|\geq k$. (4) The color of Connection. For a Connection with black (white) stones, the color is black (white). For a void Connection (without any stone in it), the color can be black, or white. Depending on the concrete game position context, the color of a void Connection can be different. (5) The shape of Connection. Denote the shape of *c* as $\|c\|$. $\|c_1\|=\|c_2\|$, iff: 1) $|c_1|=|c_2|$; 2) $\forall i \in \{0, 1, \ldots, |c|-1\}$, If i^{th} intersection of c_1 is void, so does i^{th} intersection of c_2; vice versa.

2.2 Examples for Connection

In Figure 2, there exist and only exist 3 Connections, which are Connection① 0~10, Connection② 12~18, and Connection③ 8~18. The head of Connection① is 0^{th} intersection, the length is 11, the color is black; the head of Connection② is 12^{th} intersection, the length is 7, the color is black; the head of Connection③ is 8^{th} intersection, the length is 7, the color is white.

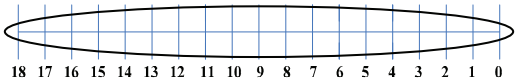

Figure 4. Void Connection

The intersections sequence from 0 to 7 is not a Connection, as well as the sequence from 4 to 10, because they do not obey the constraint that Connection must be the maximal sequence that consists of either void intersections or intersections occupied by the same color stones. We should indicate that there are two void Connections actually in Figure 4. One is black and the other is white, and they overlap each other. On *n*×n board, only the initial void line contains two void (with different colors) Connections. The color of void Connection ② in Figure 2 will only be black, can not be white. Because we can easily confirm that ③ is a white Connection, if ② is a white Connection too, according to definition 1, we will draw that both ② and ③ are the maximal white consequences. Conflict obviously! Here, if we regard ② as a black Connection, it accords with definition 1 well. The other sequence, such as the one from 0 to 3, and the one from 8 to 10 are not the Connection, because the length of them is less than 6.

However, the meanings of Connection in this paper have become more plentiful. Firstly, Connection implies a place on the board that a 6-in-a-row may occur. Secondly, it clearly outlines all the spheres of influence belonging to the black and the white in a line. Although the spheres of influence may overlap, they keep independent and integrated to each other as whole units.

2.3 Representation of Connection Shape

The most complex and most important property of Connection is its shape. To establish the relation between the shapes of Connections with a natural number, Lemma 1 is given.

Lemma 1. $n \in N$, N is the natural number set. $6 \leq n$. $C_n = \{\|c\| \mid |c| = n\}$, $M_n = \{m\} = \{0, 1, 2, \ldots, 2^{n-1}\}$, Then

$\varphi: \|c\| \rightarrow m = \varphi(\|c\|)$

is a bijection from C_n to M_n.

In the rest of this paper, $\varphi(\|c\|)$ is the function, and the variable of it is Connection *c*, the result of it is a digit corresponding with the shape of *c*, just like shown in Figure 2. That is, one bit of the digit is corresponding to one intersection of Connection, and the lowest bit of the digit is corresponding to the head of Connection. The

idea Mapping a Connection shape to a natural number is inspired by the bitboard in computer games, see [6][7].

Definition 3 (Binary tuple representation for the shape of Connection). In connect(n, n, k, p, q), the shape of c can be described as ($|c|$, $\varphi(\|c\|)$).

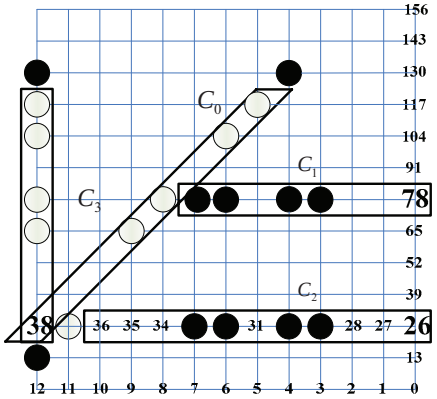

Figure 5. Connections on board for connect(13, 13, 6, 2, 1)

Lemma 2. In connect(n, n, k, p, q), $n \in N$, $6 \le n$. $C = \{\|c\| \mid |c| \le n\}$, and $M = \{0, 1, 2, \ldots, 2^{n+1}-2^{k-1}\}$. Then

$\Phi: \|c\| \rightarrow 2^{|c|} - 2^k + \varphi(\|c\|) = \Phi(\|c\|)$

is a bijection from C to M. where, $\varphi(\|c\|)$ is the binary digit of $\|c\|$ by the mapping way shown in Figure 2.

Definition 4 (Unary tuple representation for the shape of Connection). In connect(n, n, k, p, q), the shape of a connection c can be represented as $\Phi(\|c\|)$.

In Figure 5, $\|c_1\| \neq \|c_2\|$, $\varphi(\|c_1\|) = 216$, $\varphi(\|c_2\|) = 216$, $|c_1| = 8$ and $|c_2| = 11$. Based on Definition 3, $\|c_1\|$ can be described as (8, 216), and $\|c_2\|$ can be described as (11, 216). While $\Phi(\|c_1\|) = 2^{|c1|} - 2^k + \varphi(\|c_1\|) = 2^8 - 2^6 + 216 = 408$ and $\Phi(\|c_2\|) = 2^{|c2|} - 2^k + \varphi(\|c_2\|) = 2^{11} - 2^6 + 216 = 2200$. So, $\|c_1\|$ can be represented by 408, and $\|c_2\|$ can be represented by 2200 according to Definition 4. In some cases, the form of definition 3 is preferred, because we can easily know the real shape of Connection c by the binary value of $\varphi(\|c\|)$. While in other cases, the form of definition 4 is preferred. For example, if we want to know quickly whether a stone has been placed on i^{th} intersection of c, we have to resort to the form of definition 3. While to allocate each shape an equal size memory, to make each of them have a unique address, and to keep all the addresses consecutive, definition 4 is needed.

3. FORMAL DESCRIPTION FOR TYPES OF CONNECTION

It is discussed the importance of the shape of a sequence of Connect6 very well in [1][2]. Here, we will re-address such a subject, and propose a precise classification of Connection. The concept of Connection reflects the natural relations among intersections. Each Connection will have a specific type which is determined by their shape. The set of all the Connections can be divided into several disjoint sets according their shapes.

Definition 5 (Several operations and predicates). Given a Connection c, all the intersections mentioned below in this definition, such as x, y, z, s, and t belongs to c, the operations and predicates are given as follows:

(1) The PLACE(c, x, y) operation. It will place zero, one or two stones on the intersections x and/or y. The details of the operation are as follows. 1) If x is void, it can be taken by a stone with the same color as c; otherwise, the operation on x is ignored. 2) The operation on y is the same with x. PLACE(c, x, x) can be simplified as PLACE(c, x). (2) The C6(c) predicate. C6(c) is true means that a 6-in-a-row has been formed in c. If false, we denote it as $\neg$C6(c). (3) The G(c, x, y, …, z, …) predicate. G(c, x, y, …, z, …) is true means that a 6-in-a-row will be formed in c after the following operations: 1) if x is void, it can be taken by a stone with the same color as c; Otherwise, placing a stone on x will be ignored. 2) The operation on all the other intersections y, …, z, …, etc. is the same with x. (4) The H(c, x, y, s, t) predicate. H(c, x, y, s, t) is true means that there will form a new Connection c' after the following operations: 1) if x is void, place an opposite stone on it; Otherwise, placing a stone on x will be ignored. 2) The operation on y is the same with x. 3) Then there will form at most 3 sequences, and one Connection c' among which will make G(c', s, t) true.

The universal set of all the Connections can be divided into several disjoint subsets according to their functions of attacking or defending. One subset denotes one type of Connections. Connections belong to the same subset have the same Connection type.

Definition 6 (Types of Connection). Any Connection must belong to one of following subsets: win (C6), definitely win (DW), live-5 (L5), dead-5 (D5), live-4 (L4), dead-4 (D4), live-3 (L3), sleep-3 (S3), dead-3 (D3), live-2 (L2), dead-2 (D2), live-1 (L1), dead-1 (D1), void (V). The type Connection c is denoted as $\Gamma(c)$.

Table 1. Connection Types

NO.	Type	Iff
1	C6	C6(c)
2	DW	$(\forall x)(\forall y)(\exists s)(\exists t)$H($c$, x, y, s, t)
3	L5	$(\exists x)$G(c, x) $\wedge$ $(\forall x)(\exists s)(\exists t)$H($c$, x, x, s, t) $\wedge$ $(\exists x)(\exists y)(\forall s)(\forall t)\neg$H($c$, x, y, s, t)
4	D5	$(\exists x)$G(c, x) $\wedge$ $(\exists x)(\forall s)(\forall t)\neg$H($c$, x, x, s, t)
5	L4	$(\forall x)\neg$G(c, x) $\wedge$ $(\exists x)(\exists y)(\forall s)(\forall t)\neg$H($c$, x, y, s, t) $\wedge$ $(\exists x)(\exists y)$G(c, x, y) $\wedge(\forall x)(\exists s)(\exists t)$H($c$, x, x, s, t)
6	D4	$(\forall x)\neg$G(c, x) $\wedge$ $(\exists x)(\exists y)$G(c, x, y) $\wedge$ $(\exists x)(\forall s)(\forall t)\neg$H($c$, x, x, s, t)
7	L3	$(\forall x)(\forall y)\neg$G($c$, x, y) $\wedge$ $(\exists x)$L4(PLACE(c, x))
8	S3	$(\forall x)(\forall y)\neg$G($c$, x, y) $\wedge$ $(\forall x)\neg$L4(PLACE(c, x)) $\wedge$ $(\exists x)$D4(PLACE(c, x))$\wedge(\exists x)(\exists y)$L5(PLACE($c$, x, y))
9	D3	$(\forall x)(\forall y)\neg$G($c$, x, y) $\wedge$ $(\exists x)$D4(PLACE(c, x)) $\wedge$ $(\forall x)(\forall y)\neg$L5(PLACE($c$, x, y))
10	L2	$(\forall x)(\forall y)(\forall z)\neg$G($c$, x, y, z) $\wedge$ $(\exists x)(\exists y)$L4(PLACE(c, x, y))
11	D2	$(\forall x)(\forall y)(\forall z)\neg$G($c$, x, y, z) $\wedge$ $(\exists x)(\exists y)$D4(PLACE(c, x, y))
12	L1	$(\forall x)(\forall y)(\forall u)(\forall v)\neg$G($c$, x, y, u, v) $\wedge$ $(\exists x)$L2(PLACE(c, x))
13	D1	$(\forall x)(\forall y)(\forall u)(\forall v)\neg$G($c$, x, y, u, v) $\wedge$ $(\forall x)\neg$L2(PLACE(c, x)) $\wedge$ $(\exists x)$D2(PLACE(c, x))
14	V	c is a void connection

14 Connection types and their necessary and sufficient conditions are listed in Table 1. The definitions of some types of Connections are dependent on the others in the table. Concretely, ″DW″, ″L5″, ″D5″, ″L4″, and ″D4″ are dependent on ″C6″; while ″L3″, ″S3″, ″D3″, ″L2″, ″D2″, ″L1″, and ″D1″ are dependent on ″DW″, ″L5″, ″D5″,″L4″, ″C6″; only ″V″ are directly given, due to simple features. We will give the meanings of above Connection types, and take ″DW″, ″L4″as examples. For the attacker, "DW(c) is true" implies the Connection c possesses such a character: even the defender try his best to defend, that is, putting his own two stones on any two intersections, the defender couldn't escape from the state of lose. "L4(c) is true" implies that: 1) if the defender gives up the chance to defend at all, the attacker will win; 2) if the defender defends with one stone only, the attacker win; 3) if the defender defends with two stones, a strategy to prevent the attacker to win exists.

We find the types ″D2″, ″L1″, ″D1″, and ″V″ are very popular, but work little. 4 types mentioned above are merged into one type ″O″. Thus, the universal set of Connection types is T_{conn} = {″C6″, ″DW″, ″L5″, ″D5″, ″L4″, ″D4″, ″L3″, ″S3″, ″D3″, ″L2″, ″O″}.

4. EVALUATION OF CONNECTION

The types of Connection are used to reflect their relative importance. So, we must know the total ordering relation on T_{conn}. Then the concept of promotion between Connections is given, which is used to describe if a Connection promote to a better one, and how a Connection promote to a better one.

4.1 Total Ordering Relation

Different types of Connections have different threats to the opposite. To discriminate the importance of Connection by its shape, a total ordering relation on T_{conn} is needed. Let $\leqq$ represents the total ordering relation on the set T_{conn}. $a \leqq b$ means Connection type a is not better than b. According to our experiments, the relation is presented as follows: ″O″$\leqq$″L2″$\leqq$″D3″$\leqq$″S3″$\leqq$L3″$\leqq$″D4″$\leqq$″L4″$\leqq$″D5″$\leqq$″L5″$\leqq$″ DW″$\leqq$″C6″.

4.2 Knowledge Table-base of the Shapes

If all the knowledge about the shapes of Connections not only can be saved in advance, but also can be retrieved easily, the efficiency of the program will be enhanced greatly. In connect(n, n, k, p, q), the number of all the shapes is 2^{n+1}-2^k. Assume the information of each shape occupies one unit memory, the space complexity of the knowledge table-base is shown in Table 2.

Table 2. Size of knowledge-base

Games	19×19 Connect6	15×15 Connect6	13×13 Connect6	15×15 Go-Moku
Space Complexity	999.94K	63.93K	15.94K	63.97K

Table 2 shows that the size of the memory for the knowledge table-base is acceptable. Given a shape of any Connection, its entrance of the table-base can be obtained easily based on Definition 4. The most simple knowledge table-base is only to save the type of each connection. Given a connection c, the entrance of c in table-base is $\Phi(\|c\|)$, then $\Gamma(c)$ is saved there. Merits to construct a knowledge table-base are not only converting the online computation to offline, but also to help reusing the knowledge of the shape.

4.3 Promotions among Types

Definition 7 (Promotions). $\Gamma(c) \neq$″C6″, $x \in \{0, 1, \ldots, |c|\text{-}1\}$, c' is formed by PLACE(c, x). $\leqq$ is the total ordering relation on T_{conn}. We say the void intersection x let Connection c be promoted to Connection c', if $\Gamma(c) \leqq \Gamma(c')$. The promotion can be denoted as $\Gamma(c)\uparrow\Gamma(c')$.

To introduce the concept, promotions among the Connection types, aims at answering two questions: 1) Whether a stone is placed into a Connection make it better? 2) Given a certain type Connection, what target types it can promote to? According to the total ordering relation $\leqq$ on T_{conn}, The accepted promotion is listed in Table 3, and The original type is written in lowercase, and the target type is written in capital. "√" indicates the promotion exists, while "×" not.

Table 3. Promotions among Connection types

	C6	**DW**	**L5**	**L4**	**D5**	**D4**	**L3**	**S3**	**D3**	**L2**	**O**
dw	√	×	×	×	×	×	×	×	×	×	×
l5	√	√	×	×	×	×	×	×	×	×	×
l4	×	√	√	×	×	×	×	×	×	×	×
d5	√	√	√	×	×	×	×	×	×	×	×
d4	×	√	√	√	√	×	×	×	×	×	×
l3	×	×	×	√	×	√	×	×	×	×	×
s3	×	×	×	×	×	√	√	×	×	×	×
d3	×	×	×	×	×	√	√	√	×	×	×
l2	×	×	×	×	×	×	√	√	√	×	×
o	×	×	×	×	×	×	×	√	√	√	×

Table 3 contains some redundant information, and they are: (1) The unnecessary promotions. Although the promotion D5↑L5 is acceptable, we should avoid this happening for ever. It is obvious that the player can change the Connection shape type from ″D5″ to ″C6″ through putting a same color stone at any time and win at once. So, D5↑L5 is irrational and should be eliminated from the table, due to miss the opportunity to win. (2) Never consider ″DW″ as a target type. From Table 3, we know original types can promote to ″DW″ directly are ″L5″, ″D5″, ″L4″ and ″D4″. Promotions, such as D5↑DW and L5↑DW, should be forbidden, due to missing the opportunity to win.

We get Table 4 by eliminating the useless promotions from Table 3. In Table 4: 1) using ″Δ″ represents a forbidden promotion; 2) dropping ″DW″ from the set of target types of promotion.

Table 4. Reduced Promotions among Connection types

	C6	**L5**	**L4**	**D5**	**D4**	**L3**	**S3**	**D3**	**L2**	**O**
dw	√	×	×	×	×	×	×	×	×	×
l5	√	×	×	×	×	×	×	×	×	×
l4	×	√	×	×	×	×	×	×	×	×
d5	√	Δ	×	×	×	×	×	×	×	×
d4	×	√	√	√	×	×	×	×	×	×
l3	×	×	√	×	√	×	×	×	×	×

s3	×	×	×	×	√	√	×	×	×	×
d3	×	×	×	×	√	√	√	×	×	×
l2	×	×	×	×	×	√	√	√	×	×
o	×	×	×	×	×	×	√	√	√	×

5. TYPE OF INTERSECTION

Based on the promotion among Connections, the types of intersections are given. To accelerate the program's execution, an enhanced knowledge table-base is given.

5.1 Types of Intersections

Any Connection has a determinate type, so does any void intersection. Just because putting a stone on an intersection may change the type of the Connection, the importance of an intersection has a close relevancy with the type of Connection. In this case, the value of the intersection can be evaluated by a newly formed Connection through putting a stone on it.

Let T_{is} = {″C6″, ″DW″, ″L5″, ″D5″, ″L4″, ″D4″, ″L3″, ″S3″, ″D3″, ″L2″, ″O″, ″U″}. Where, the new introduced type ″U″ is associated with an intersection which should never be put a stone on, such as the void intersections in a sequence whose length is shorter than 6. To evaluate an intersection, we also need a total ordering relation on T_{is}. For example, ″U″≦″O″≦″L2″≦″D3″≦″S3″≦″L3″≦″D4″≦″L4″≦″D5″≦″L5″≦″DW″≦″C6″. Given a Connection c, its type is $\zeta(c) \neq$″C6″. c' = PLACE(c, x), the type of intersection x of c, written as $\tau(c, x)$. The algorithm to figure out the type of each intersection in a Connection is as follows.

(1) If x is nonvoid, $\tau(c, x)$ = ″U″. End.
(2) If $\zeta(c) \in$ {″DW″, ″L5″, ″D5″, ″L4″, ″D4″}, then
  1) If c promotes to c', then $\tau(c, x) = \Gamma(c')$. End.
  2) If c promotes to c', then $\tau(c, x)$ = ″U″. End.
(3) If $\zeta(c) \in$ { ″L3″, ″S3″, ″D3″, ″L2″, ″O″}, then
  1) If c promotes to c', then $\tau(c, x) = \Gamma(c')$. End.
  2) If c promotes to c', then $\tau(c, x)$ = ″O″. End.

To a player, $\tau(c, x)$ = ″U″ means x is refuted by c, and should never to be taken by any stone; $\tau(c, x)$ = ″O″ means whether x is taken by a stone or not seems unprevailing; Otherwise, placing a stone on x should be encouraged, and to what extent the encourage should be decided by the type of x and the relation ≦.

5.2 Knowledge Table-base of Intersections

Knowledge table-base of shapes in section 4.2 accelerates the program, because the offline calculation saved the online calculation. But the type of void intersection is needed frequently, especially in the procedure of move generation and move selection. To figure out the type of each intersection in an n-length Connection, we will generate no more than n new Connection and need to know the type of them, and to query form the table-base. So, the complexity to calculate entrances of the type of each intersection within single Connection in the knowledge table-base is O(n). Furthermore, the jumped entrance of table-base may cause the visiting to such a table-base inefficient, because the method isn't friendly to the cache. The way to improve it is to save the type of each intersection into table-base. In connect(n, n, k, p, q), there needs n+1 units to store the knowledge of a Connection, the first unit of which is stored the information of Connection type, (i-1)th intersection type is stored in the following i^{th} unit.

6. CONCLUSION AND FUTURE WORK

We propose a novel method to construct the knowledge table-base in k-in-a-row games. The concept of Connection, Connection type, intersection type, and the composite type of intersection all simplified the design of the program and brought many enhancement of it. The most important way to improve the efficiency of the program is the construction of the knowledge, in which the Connection type and the intersection type is stored. In the future, we will give more attention to it, and try to get some meaningful results.

7. REFERENCES

[1] Wu I-C, Huang D-Y, A New Family of k-in-a-row Games, in *Proceedings of The 11th Advances in Computer Games Conference*, 2005, 88-100.

[2] Wu I-C, Huang D-Y, Chang H-C, Connect6, *ICGA Journal*, 28, 4 (Dec. 2005), 234-242.

[3] Hsieh Ming-Yu, Tsai Shi-Chun, On the fairness and complexity of generalized k-in-a-row games, *Theoretical Computer Science*, 385, 1-3 (Oct. 2007), 88-100.

[4] H. J. van den Herik, J. W. H. M. Uiterwijk, and J. van Rijswijck. Games solved: Now and in the future. *Artificial Intelligence*, 134, 1-2 (Jan. 2002), 277-311.

[5] Allis LV, *Searching for solutions in games and artificial intelligence*. Ph.D. thesis, Maastricht, University of Limburg, 1994.

[6] Heinz EA, How Dark Thought Plays Chess. *ICCA Journal*, 20, 3 (Sept. 1997), 167-176.

[7] Pablo S-S, Ramón G, Fernando M, Diego R-L, Agustín J. Efficient Search Using BitBoard Models. In Proceedings of International Conference on Tools with Artificial Intelligence, 2006,132-138.