# Open and End Game Databases

Tsan-sheng Hsu

徐讚昇

*tshsu@iis.sinica.edu.tw*

http://www.iis.sinica.edu.tw/~tshsu

# Abstract

- The open book.
- The endgame database.
  - Construction of an endgame database: retrograde analysis
  - Consistence check of endgame knowledge

# The opening

- **During the open game, it is frequently the case**
  - **branching factor is huge;**
  - **it is difficult to write a good evaluation function;**
  - **the number of possible distinct positions up to a limited length is small as compared to the number of possible positions encountered during middle game search.**
- **Difficult to search in the open and need some extra procedures to help.**
  - **Obtain domain knowledge.**
    - ***Expert generated meta rules.***
    - ***Expert annotated game logs.***
- **Enumerate and then pre-compute the first few plys to save time.**
  - **Trade space with time.**

# Meta rules

- **Build or construct meta-knowledge for the opening.**
  - **Expert systems or databases built from human knowledge.**
  - **Examples using CDC.**
    - ▷ ***Example 1: when the first player reveals a king, then try to flip its adjacent piece for a possible pawn or to flip for a cannon attack.***
    - ▷ ***Example 2: Enumerate all possible combinations, including locations and pieces revealed, of the first and the second plys and then find the strategies with the best expected outcome.***
  - **Machine learning or deep learning programs to mine domain knowledge from games logs.**

# The open book (1/2)

- **Acquire game logs from**
  - **books;**
  - **games between masters;**
  - **games between computers;**
    - ▷ ***Use off-line computation to find out the value of a position for a given depth that cannot be computed online during a game due to resource constraints.***
  - $\cdots$

# The open book (2/2)

- Assume you have collected $r$ games.
  - For each position in the $r$ games, compute the following 3 values:
    - $win$: *the number of games reaching this position and then wins.*
    - $loss$: *the number of games reaching this position and then loss.*
    - $draw$: *the number of games reaching this position and then draw.*
- When $r$ is large and the games are trustful, then use the 3 values to compute an estimated level of goodness for this position.
  - $win + 0.5 * draw$
  - $win$
  - ...

# Example: Chinese chess open book (1/3)

- A total of 28,591 (Red win)+21,072 (Red lose)+55,930 (draw) games.

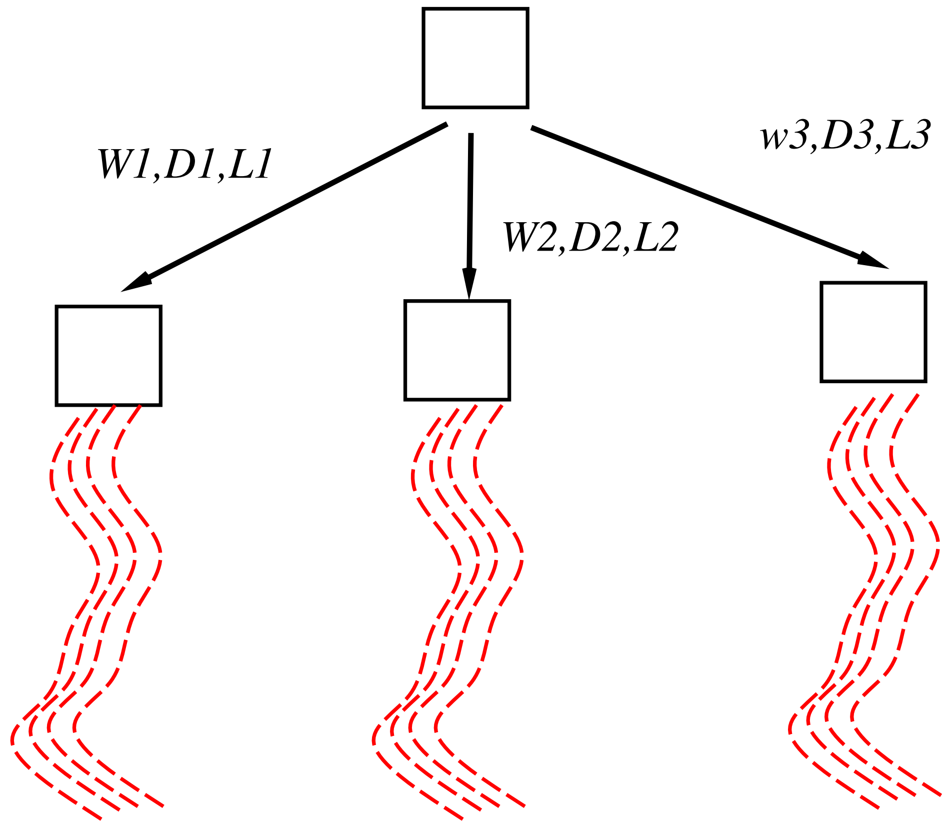
# Example: Chinese chess open book (2/3)

- **Can be sorted using different criteria.**
  - **Win-lose**
  - **winning rates**
  - **...**

# Example: Chinese chess open book (3/3)

- A tree-like structure.

# Illustration



# Comments (1/2)

- **Pure statistically.**
  - **Try to have some varieties. Do not always use the best one to avoid falling into a trap set up by opponents that have been watching your playing records.. Let the second one have some chance to be used.**
  - **Use ideas from UCB.**
    - *First build the open game tree using existing databases.*
    - *Then add computer self-playing logs or Monte-Carlo like simulations using UCB formulations.*
    - *Best first tree growing like MCTS*
- **Need to figure out a way to handle loops.**
- **Can build a static open book.**
  - **It is difficult to acquire large amount of "trustful" game logs.**
  - **Can build the open book off-line by using your program to search a time longer than the tournament time**

# Comments (2/2)

- **Drawbacks**
  - You program may not be able to **take over** when the open book is over.
  - If your opening is fixed, namely only uses the best in your book, your opponent can use that to design a strategy to your disadvantage.
  - If you do not use the best move, then you may use a very bad one.
  - Some sort of Monte-Carol simulation strategy can be used.
- **Research opportunities**
  - Automatically analysis of game logs written by human experts [Chen et. al 2006]
  - Using high-level meta-knowledge to guide searching:
    - ▷ *Chinese dark chess (CDC): adjacent attack of the opponent's Cannon [Chen and Hsu 2013]*
  - Semi-auto cleaning of massive amount of data collected from online and other resources.
    - ▷ *errors*
    - ▷ *broken connections*
    - ▷ *logs from creditable/non-creditable sources*

# Endgame

- **Entering the endgame, it is frequently the case**
  - **the number of remaining pieces is small;**
  - **special strategies or heuristics differ from the one used in other phases of the game exist.**
- **Solving the endgame by**
  - **implementing heuristics;**
  - **systematically enumeration of all possible combinations.**

# Endgame databases

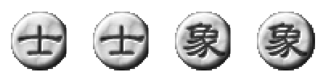
- **Chinese chess endgame database:**
  - **Indexed by a sublist of pieces $S$, including both Kings.**

| K | G | M | R | N | C | P |
|---|---|---|---|---|---|---|
| King | Guard | Minister | Rook | Knight | Cannon | Pawn |
| 帥 將 | 仕 士 | 相 象 | 俥 車 | 傌 馬 | 炮 包 | 兵 卒 |

    - ▷ *KCPGGMMKGGMM ( 炮 兵 仕 仕 相 相 vs. 士 士 象 象 ): the database consisting of RED Cannon and Pawn, and Guards and Ministers from both sides.*
  - **A *position* in a database $S$: A legal arrangement of pieces in $S$ on the board and an indication of who the next player is.**
  - ***Perfect information* of a position:**
    - ▷ *What is the best possible outcome, i.e. win/loss/draw, that the player can achieve starting from this position?*
    - ▷ *What is a strategy to achieve the best possible outcome?*
  - **Given $S$, to be able to give the perfect information of all *legal positions* formed by placing pieces in $S$ on the board.**

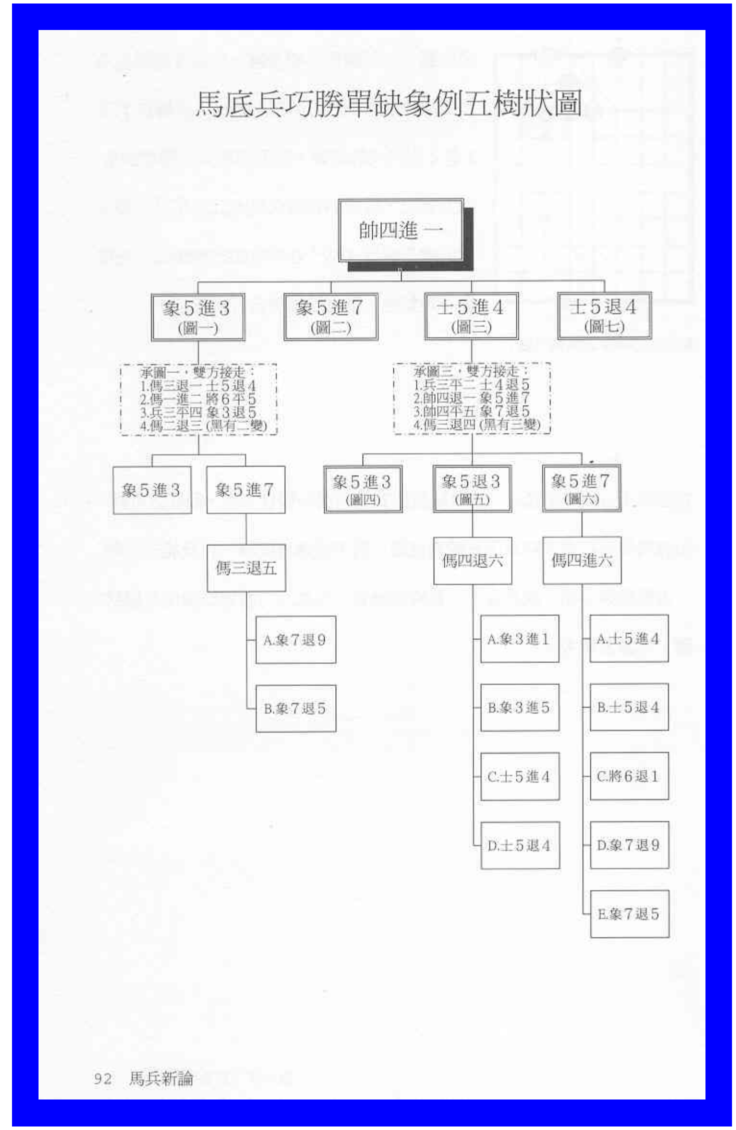
# Usage of endgame databases

- The database may only contain *partial information* of a position:
  - win/loss/draw; DTM, DTC; DTZ.
    - ▷ *DTM: depth to mate, i.e., the largest number of plys that your opponent can stall before you win.*
    - ▷ *DTC: depth to conversion, i.e., the largest number of plys that your opponent can stall before you can capture a piece and stay winning.*
    - ▷ *DTZ: depth to zeroing, i.e., the largest number of plys that your opponent can stall before you can make a progress and stay winning to avoid a draw by rules.*
- Improve the "skill" of Chinese chess computer programs.
  - KNPKGGMM ( 

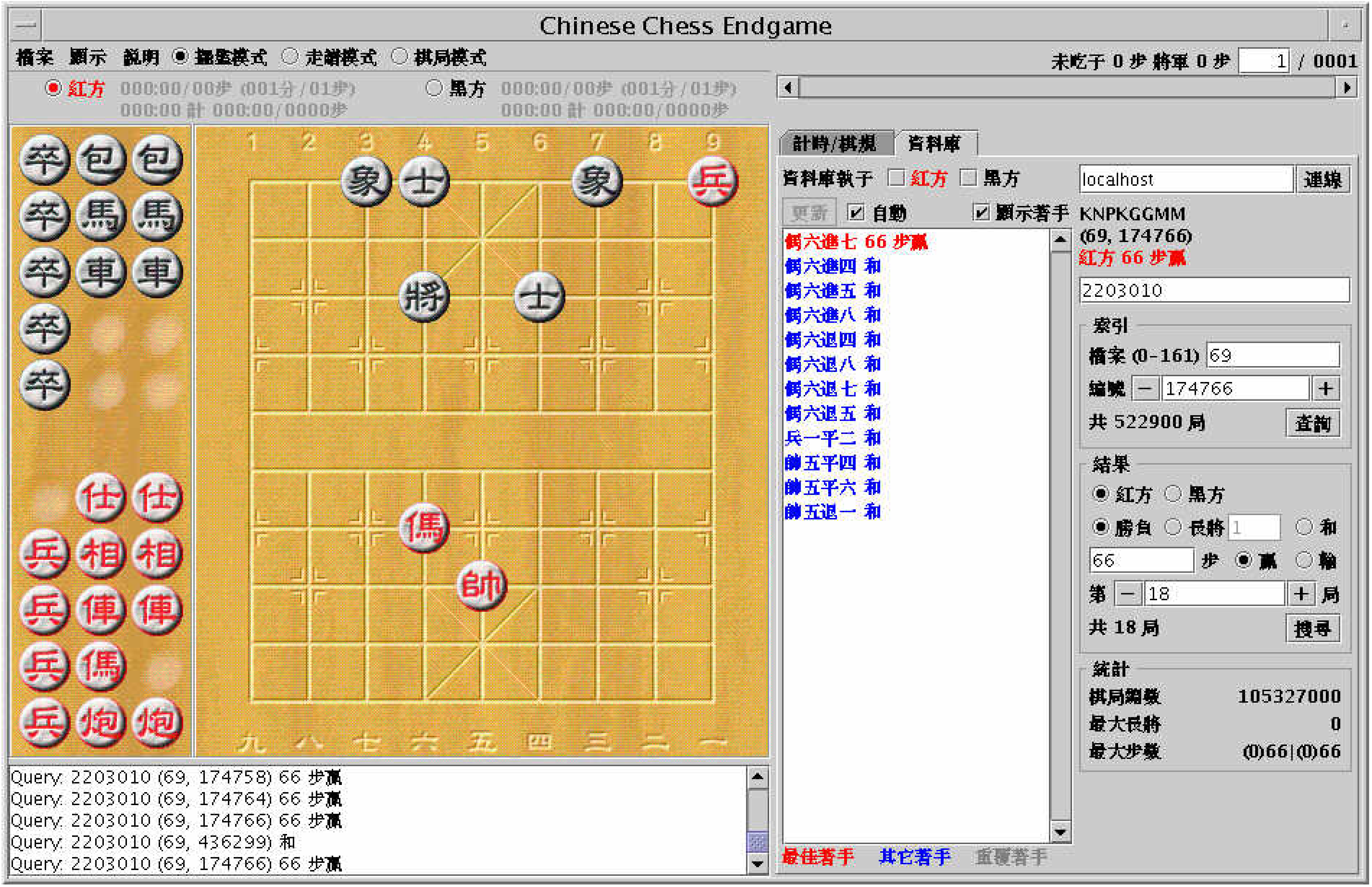
vs. 

)
- Educational:
  - Teach people to master endgames.
- Recreational.

# An endgame book

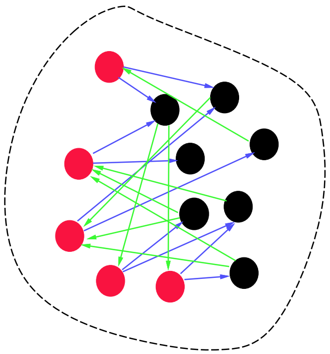
# Books


Chinese Chess Endgame
檔案 顯示 說明 鑑藍模式 走譜模式 棋局模式
未吃子 0 步 將軍 0 步 1 / 0001
紅方 000:00/00步 (001分/01步) 000:00 計 000:00/0000步
黑方 000:00/00步 (001分/01步) 000:00 計 000:00/0000步
計時/棋規 資料庫
資料庫執子 紅方 黑方
localhost 連線
更新 自動 顯示著手
KNPKGGMM
(69, 174766)
紅方 66 步贏
2203010
偶六進七 66 步贏
偶六進四 和
偶六進五 和
偶六進八 和
偶六退四 和
偶六退八 和
偶六退七 和
偶六退五 和
兵一平二 和
帥五平四 和
帥五平六 和
帥五退一 和
索引
檔案 (0-161) 69
編號 174766
共 522900 局 查詢
結果
紅方 黑方
勝負 長將 1 和
66 步 贏 輸
第 18 局
共 18 局 搜尋
統計
棋局總數 105327000
最大長將 0
最大步數 (0)66|(0)66
最佳著手 其它著手 重覆著手
Query: 2203010 (69, 174758) 66 步贏
Query: 2203010 (69, 174764) 66 步贏
Query: 2203010 (69, 174766) 66 步贏
Query: 2203010 (69, 436299) 和
Query: 2203010 (69, 174766) 66 步贏


Chinese Chess Endgame
未吃子 0 步 將軍 0 步 1 / 0001
紅方 000:00/00步 (001分/01步)
黑方 000:00/00步 (001分/01步)
資料庫執子 紅方 黑方
localhost
連線
KNPKGGMM
(69, 203899)
紅方 和
2203010
傌六進四 和
傌六進五 和
傌六進七 和
傌六進八 和
傌六退四 和
傌六退八 和
傌六退七 和
傌六退五 和
兵一平二 和
帥五平四 和
帥五平六 和
帥五退一 和
共 522900 局
共 15527336 局
棋局總數 105327000
Query: 2203010 (69, 174764) 66 步贏
Query: 2203010 (69, 174766) 66 步贏
Query: 2203010 (69, 436299) 和
Query: 2203010 (69, 174766) 66 步贏
Query: 2203010 (69, 203899) 和
最佳著手
其它著手

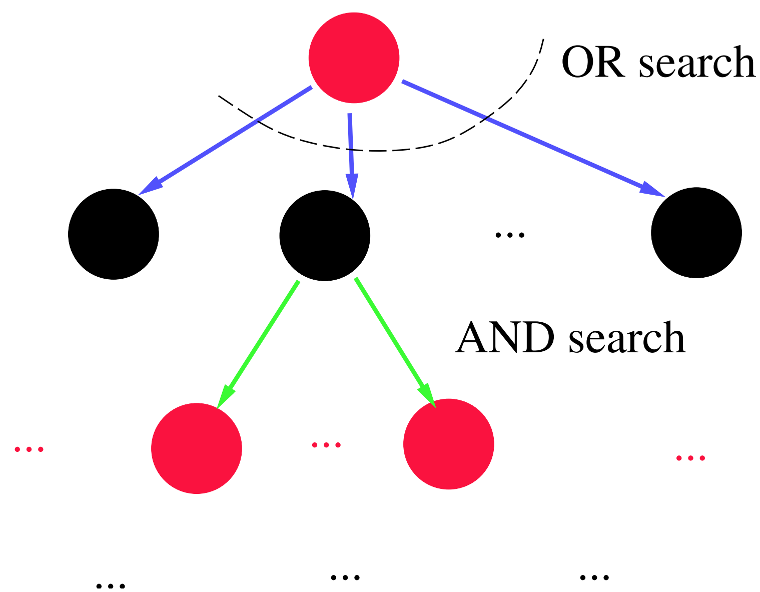
# Definitions

- **State graph for an endgame $H$:**
  - **Vertex: each legal placement of pieces in $H$ and the indication of who the current player (Red/Black) is.**
    - *Each vertex is called a* **position**.
    - *May want to remove symmetry positions.*
  - **Edge: directed, from a position $x$ to a position $y$ if $x$ can reach $y$ in one ply.**
  - **Characteristics:**
    - *Bipartite.*
    - *Huge number of vertices and edges for non-trivial endgames.*
    - *Example: KCPGGMMKGGMM has $1.5 * 10^{10}$ positions and about $3.2 * 10^{11}$ edges.*



# Overview of algorithms

- Forward searching: doesn't work for non-trivial endgames.
  - AND-OR game tree search.
  - Need to search to the terminal positions to reach a conclusion.
  - Runs in exponential time not to mention the amount of main memory.
  - Heuristics: A$^*$, transposition table, move ordering, iterative deepening ...

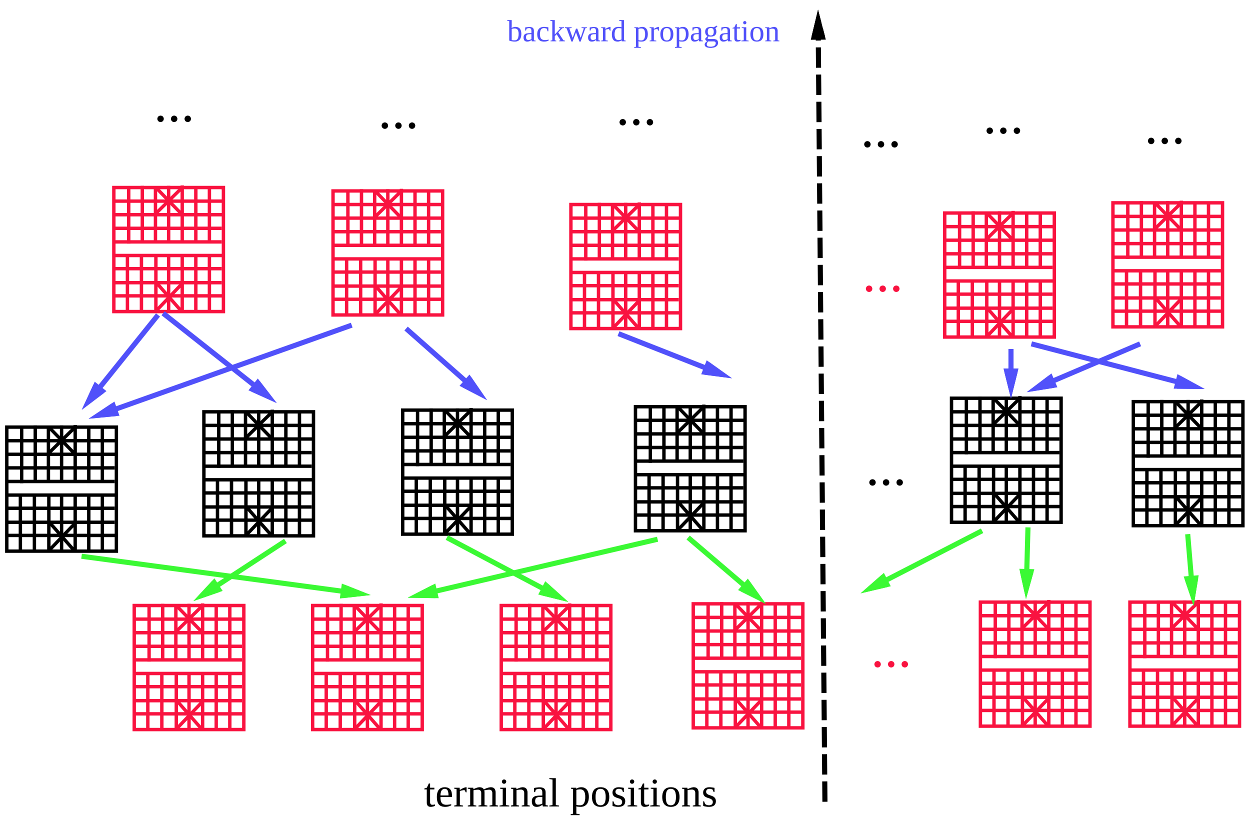


# Retrograde analysis (1/2)

- **First systematic study by Ken Thompson in 1986 for Western chess.**
  - **Retrograde analysis**
- **Algorithm:**
  - **List all positions.**
  - **Find all positions that are initially "stable", i.e., solved.**
  - **Propagate the values of stable positions backward to the positions that can reach the stable positions in one ply.**
    - ▷ *Watch out the and-or rules.*
  - **Repeat this process until no more changes is found.**

# Retrograde analysis (2/2)

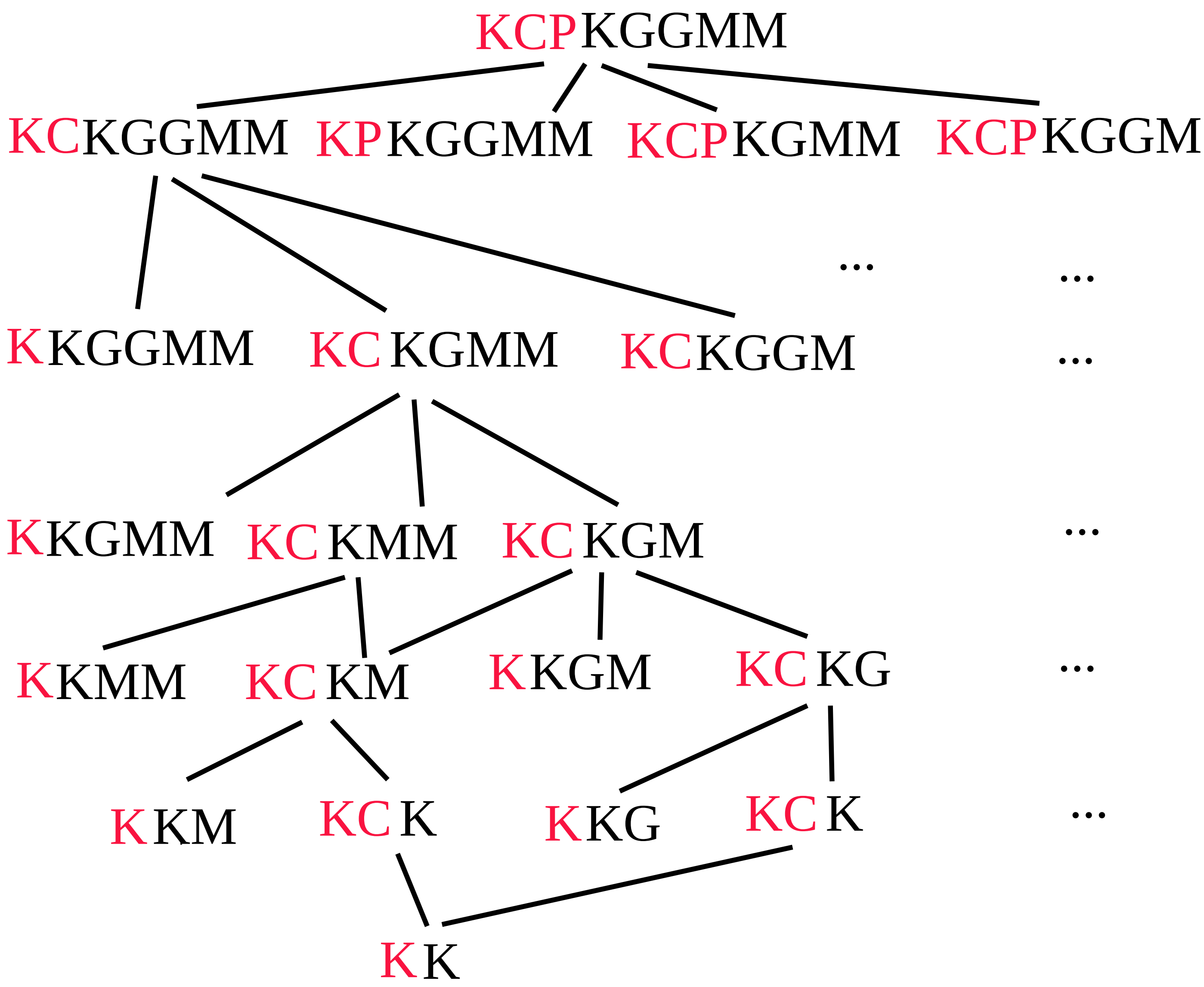
- **Critical issues: time and space trade off.**
  - **Information stored in each vertex can be compressed.**
  - **Store only vertices, generate the edges on demand.**
  - **Try not to propagate the same information.**



# Stable positions

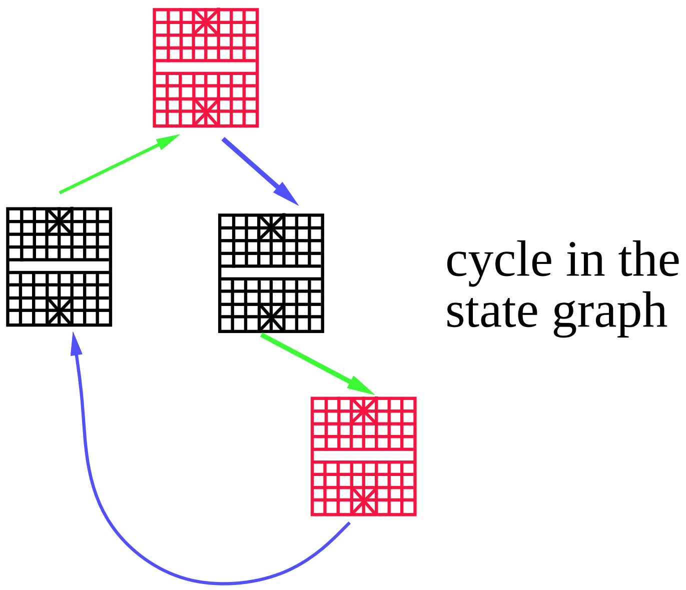
- **Another critical issue: how to find stable positions?**
  - **Checkmate, stalemate, King facing King.**
  - **It maybe the case the best move is to capture an opponent's piece and then win.**
    - ▷ ***so called "depth-to-capture" (DTC);***
    - ▷ ***the traditional metric is "depth-to-mate" (DTM).***
- **Need to access values of positions in other endgames. For example,**
  - **KCPKGGMM needs to access**
    - ▷ ***KCKGGMM***
    - ▷ ***KPKGGMM***
    - ▷ ***KCPKGMM, KCPKGGM***
  - **A lattice structure for endgame accesses.**
  - **Need to access lots of huge databases at the same time.**
- **[Hsu & Liu, 2002] uses a simple graph partitioning scheme to solve this problem with good practical results.**

# An example of the lattice structure



# Cycles in the state graph (1/2)

- **Yet another critical issue: cycles in the state graph.**
  - **Can never be stable.**
  - **In terms of graph theory,**
    - ▷ ***a stable position is a pendant in the current state graph;***
    - ▷ ***a propagated position is removed from the sate graph;***
    - ▷ ***no vertex in a cycle can be a pendant.***



# Cycles in the state graph (2/2)

- For most games, a cyclic sequence of moves means draw.
  - Positions in cycles are stable.
  - Only need to propagate positions in cycles once.
- For Chinese chess, a cyclic sequence of moves can mean win/loss/draw.
  - Special cases: only one side has attacking pieces.
    - ▷ *Threaten the opponent and fall into a repeated sequence is illegal.*
    - ▷ *You can threaten the opponent only if you have attacking pieces.*
    - ▷ *The stronger side does not need to threaten an opponent without attacking pieces.*
    - ▷ *All positions in cycles are draws.*
  - General cases: very complicated.

# Index function

- **Given the set of legal positions $\mathcal{P}$, design a function $f(p) \mapsto I$ where $p \in \mathcal{P}$ is a legal positions and $I$ is a non-negative integer with constraints that**
  - $f(p_1) \neq f(p_2)$ **if** $p_1 \neq p_2$**,**
  - $ratio = \frac{|\mathcal{P}|}{maxI+1} \sim 1$ **where** $maxI = \max_{\forall p \in \mathcal{P}}\{f(p)\}$**.**
- **For performance, we need**
  - **both the encoding function $f$ and the decoding function $f^{-1}$ to be able computed efficiently**
  - **Ne able to store $maxI$ of values in the main memory.**
  - **When $ratio = 1$, the scheme is perfect.**
- **Can also make use of symmetry reduction, namely, mapping symmetrical positions via mirroring, etc, into one.**
- **Compression: after the database is constructed, can use some compression tools to reduce the storage size.**
  - **Can read a particular location in a compressed array without decompression.**
  - **Example: A block based invertible compression functions RRR. cite: RRR: A Succinct Rank/Select Index for Bit Vectors, Alex Bowe, https://www.alexbowe.com/rrr/**

# Commonly used index functions (1/2)

- **bucket based:**
  - **if a piece $h_i$ can be resided in $x_i$ locations, then reserve $\lceil \log_2(x_i) \rceil$ bits for its location.**
  - **For a total of $n$ pieces $h_1, \ldots, h_n$, we use $\sum_{i=1}^{n} \lceil \log_2(x_i) \rceil$ bits for a position.**
- **Comments**
  - **Very easy to encode and decode**
  - **Have some fragmentation, namely the space without any positions can be mapped to, if $x_i$ is not a power of 2 and if two pieces cannot be in one location.**
  - **Not easy to do symmetry reduction when there are more than one piece of the same kind.**

# Commonly used index functions (2/2)

- **radix based:**
  - if every piece $h_I$ can be resided in $x$ locations and there are $n$ pieces, then use an $x$-based numbering system with a total of $x^n$ possible positions.
- **Comments**
  - Easy to encode and decode
  - Have fragmentation, namely the space without any positions can be mapped to, since two pieces may be in one location.
  - If each piece can have different resident locations, then use a **mixed radix** number system such in the example of using hour-minute-second to tell the time.
    - ▷ *Donald Knuth. The Art of Computer Programming, Volume 2: Half Numerical Algorithms, Third Edition. Addison Wesley, 1997. ISBN 0-201-89684-2. 65-66 pages, 208-209 pages, 290 pages.*
    - ▷ *George Cantor. On simple number systems, journal for math and physics 14 (1869), 121-128.*

# Combinatorial code/decode function

- **Combinatorial code/encode function: without any fragmentation if no pieces are of the same kind.**
  - **Assumption: if a position can only be resided by at most one piece, then the locations of the $S$ pieces, assume no two pieces are the same, forms a combination of length $S$.**
    - ▷ *Can use the well-known combinatorial code/encode design to find an $f$ that is 1-1 mapping and $ratio = 1$.*
    - ▷ *Cite: Donald Knuth, The art of computer programming, vol 4A, pp.355–390.*
- **Theorem L: There exists an ordering of visiting all length-$t$ combinations such that the combination $(c_t, \ldots, c_2, c_1)$ with $c_i > c_{i-1}$, $\forall 1 < i \leq t$, is visited after exactly $\sum_{i=1}^{t} \binom{c_i}{i}$ alphabetically smaller such permutations are visited.**

# Example

- **Number of different positions by placing 5 black stones in a 9x9 Go board without doing any symmetry reduction.**
  - **bucket based: need 5*$\lceil \log_2(81) \rceil$ = 35 bits in code.**
    - ▷ *fast and easy to implement*
  - **radix based: need $\lceil \log_2(81^5) \rceil = 32$ bits to code.**
    - ▷ *easy to implement, but is slower than the above*
  - **combinatorial coding: $\binom{81}{5} = 25,621,596$ which needs 25 bits.**
    - ▷ *not too easy to implement, and not too slow in speed compared to the above two*
- **Comments: the above formulations are for the case when all pieces are of the same kind. When pieces are not all the same,**
  - **bucket based and radix based ones can be used without change;**
  - **combinatorial code needs to be extended, and can be done.**

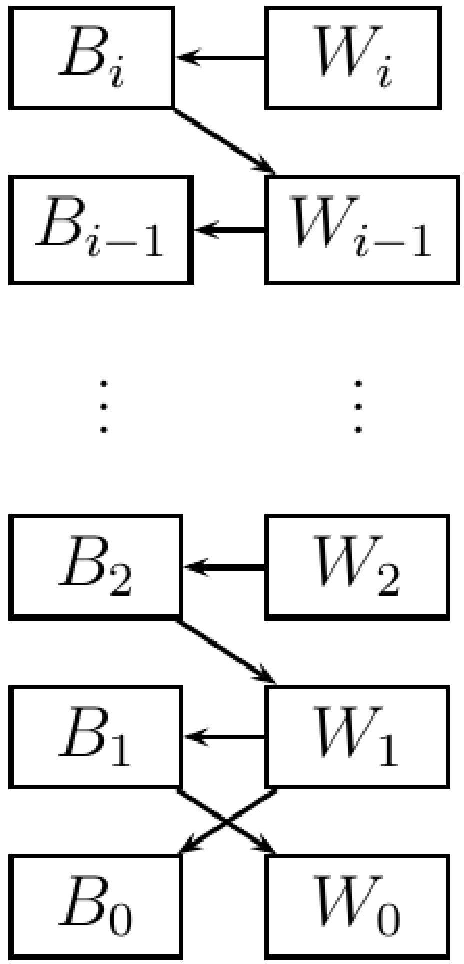
# Overview of retrograde analysis algorithms

- **Forward based algorithms**
- **Backward based algorithm**
- **Advanced techniques**
  - **Layer structure**
  - **Disk based computation**

# Definitions

- **For 2-person game, assume the 2 sides are $B$ and $W$.**
- **Classifications of positions:**
  - **loss-in-$i$, $B_i$: $B$ to move and a sure lose in $i$, or more, plys, if $W$ makes a mistake.**
    - ***$i = 0$ means loss at once***
    - ***For chess, stalemate is illegal, so $B_0$ is the set of positions that $B$ is in-check and remains to be in-check for all moves.***
    - ***For Chinese chess, $B_0$ is stalemate.***
    - ***For $i > 0$, all plys for a position in $B_i$ leads to a position in $W_j$, $j \geq i-1$. Furthermore, there is a ply leads to a position in $W_{i-1}$.***
  - **win-in-$i$, $W_i$: $W$ to move and a sure win in $i$, or less, plys, if $B$ makes a mistake**
    - ***For Chinese chess, $W_1$ is the set of positions that can reach a position in $B_0$ in 1 ply and $W_0$ is the set of positions that can capture the opponent's king in 1 ply.***
    - ***For chess, $W_1$ is the set of positions that can reach a position in $B_0$ in 1 ply and $W_0 = \emptyset$.***
    - ***A position in $W_i$ can reach a position in $B_{i-1}$ in 1 ply when $i > 1$.***

# Structure of positions



# Remarks

- **All positions need to be legal. Hence you cannot define a position resulting from the king being captured.**
- **Use symmetric reduction to find positions of**
  - **white-to-move and lose**
    - ▷ ***To find these positions, flip black and white, and the next player from white to black, find those $B_i$'s.***
  - **black-to-move and win**
    - ▷ ***To find these positions, flip black and white, and the next player from black to white, find those $W_i$'s.***

# Initialization

- **Initialization:**
  - **Depends on rules of the game, $B_0$ and $W_0$ have different initialization methods.**
    - ***For chess, $B_1$ is empty.***
- **For constructing depth-to-mate (DTM) values in a lattice way**
  - **$W_i$ contains the positions that white captures a black piece and then inductively becomes a black-to-move and lose in $i$ plys. If there are several such captures, use the one with the smallest $i$**
  - **$B_i$ contains the positions that black can only capture in the next ply and each capture is inductively win fir white in $i$ plys. Among all these captures, use the one with the largest $i$.**
- **For constructing depth-to-conversion (DTC) values in a lattice way**
  - **initialize $W_0$ to be the positions that white captures a black piece and then inductively becomes black-to-move and win in any plys.**
  - **$B_i$ contains the positions that black can only capture in the next ply and each capture is inductively win fir white in any plys.**

# Summary of algorithms

- **Forward based algorithm (layered)**
- **Backward based algorithms**
  - **Layered**
  - **Propagate only stable nodes once**
  - **Layered and propagate only stable nodes once**
- **Disk based approach**

# Fundamental procedures (1/3)

- **Update the value of $p$'s parent $p'$ using the value of $p$.**
- **UPDATE_B$_b$(position $p$)**
  **// backward update**
  **// $p$ is black-to-move**
  - **if $current(p)$ is lose-in-$i$, then**
    - ▷ ***if $current(p')$ is lose, unknown or win-in-$j$ and $j > i + 1$, then***
      ***$current(p')$ = win-in-$(i + 1)$ // update to a better value***
- **UPDATE_W$_b$(position $p$)**
  **// backward update**
  **// $p$ is white-to-move**
  - **if $current(p)$ is win-in-$i$, then**
    - ▷ ***if $current(p')$ is unknown or loss-in-$j$ and $j < i$, then***
      ***$current(p')$ = lose-in-$i$ // update to a better value***

# Fundamental procedures (2/3)

- **Update the value of $p$ using the values of all its children so far.**
- **UPDATE_W$_f$(position $p$) //$p$ is white-to-move**
  **// forward update for white-to-move and win**
  - **if there exists a child $p_j$ of $p$ such that $current(p_j)$ is loss, then**
    - ▷ ***// a child of $p$ is black-to-move***
    - ▷ ***find a lose child $p^*$ with $current(p^*)$ being the least $k$ in lose-in-$k$***
    - ▷ $current(p)$ = ***win-in-***$(k+1)$
  - **Otherwise, the value of $p$ is un-decided, and remains to be unknown.**
- **UPDATE_B$_f$(position $p$) //$p$ is black-to-move**
  **// forward update for black-to-move and lose**
  - **// there is no losing child**
  - **if all children of $p$ are winning, then**
    - ▷ ***// a child of $p$ is white-to-move***
    - ▷ ***find a win child $p^*$ with $current(p^*)$ being the largest $k$ in win-in-$k$***
    - ▷ $current(p)$ = ***lose-***$(k+1)$
  - **Otherwise, the value of $p$ is un-decided, and remains to be unknown.**

# Fundamental procedures (3/3)

- How to verify a black-to-move position $p$ is sure-to-lose using information of all its children?
- VERIFY$_{loss}$(position $p$)
  // verify $p$ is a losing position
  // $p$ is black-to-move
  - if all children of $p$ are white-to-move and win, then
    - *return TRUE;*
  - else return FALSE;

# Forward based Algorithms

- **A repeatedly forward checking retrograde analysis algorithm.**
- **RFC(endgame $E$)**
  **// build endgame $E$ using repeatedly forward checking RA**
  **// in layers**
  - **initialize $B_0$ and $W_0$**
  - **initialize all other positions to be unknown**
  - **for each unknown black-to-move position $p$ do**
    **if all children of $p$ are in $W_0$ then**
    **put $p$ in $B_1$**
  - $i = 1$
  - **repeat**
    - ***for each unknown white-to-move position $p$ do***
      ***if a child of $p$ is in $B_{i-1}$ then***
      ***put $p$ in $W_i$***
    - ***for each unknown black-to-move position $p$ do***
      ***if all children of $p$ are in some $W_j$, $j \leq i$, then***
      ***put $p$ in $B_{i+1}$***
    - $i++$
  - **until no values of positions is changed in the above for loop**
  - **Mark all unknown positions to be draw**

# Properties

- **For $i > 0$, a position $p$ in $W_i$ has a child in $B_{i-1}$.**
  - **Some children of $p$ may be unknown.**
  - **Some children of $p$ may be in some $B_j$, $j \geq i$.**
  - **No child of $p$ can be in some $B_j$, $j < i - 1$.**
- **For $i > 0$, every child of a position $p$ in $B_i$ is in $W_j$, $j < i$. Furthermore, $p$ has a child in $W_{i-1}$.**

# Layered Backward algorithm (1/2)

- **Di not need to scan the whole database to find updates.**
  - **Use un-move generator to find $W_i$ from $B_{i-1}$.**
    - ▷ ***The parents of positions in $B_{i-1}$.are $W_i$.***
  - **Use un-move generator to find potential candidates of $B_{i+1}$ from $W_i$.**
    - ▷ ***The parents of positions in $W_i$.are potential candidates which is called the set $J_{i+1}$.***
    - ▷ $B_{i+1} \subseteq J_{i+1}$
    - ▷ $J_{i+1}$ ***is much smaller than the whole database***
    - ▷ ***Use $VERIFY_{LOSS}$ to filter $J_{i+1}$ and find $B_{i+1}$.***
- **Cost:**
  - **It is frequently the case that an un-move generator is more difficult to implement than a move generator.**
  - **Need to use a move generator in $VERIFY_{LOSS}$.**

# Layered Backward algorithm (2/2)

- **LBP(endgame $E$)**
  **// build endgame $E$ using backward propagation RA**
  **// in layers**
  - **initialize $B_0$ and $W_0$**
  - **initialize all other positions to be unknown**
  - **$B_1$ = parents of positions in $W_0$ that are unknown;**
  - $i = 1$
  - **repeat**
    - ▷ $W_u$ = *parents of positions in* $B_{i-1}$ *that are unknown;*
    - ▷ $J_{u+1}$ = *parents of positions in* $W_i$ *that are unknown;*
    - ▷ $B_{i+1} = \emptyset$
    - ▷ *for each position* $p$ *in* $J_{i+1}$ *do*
      *if* $VERIFY_{loss}(p)$ *then* $B_{i+1} \cup= \{p\}$
    - ▷ $i++$
  - **until no values of positions is changed in the above for loop**
  - **Mark all unknown positions to be draw**

# Propagate only stable positions

- **Properties:**
  - **Only stable positions are needed to back propagate their scores to their parents.**
  - **A stable position only need to propagate once.**
  - **A terminal position is stable.**
  - **A position whose children are all stable is stable.**
- **Need to record the number of unstable children, when and only when this number becomes 0, then do the propagation.**
  - **Di not need to find potential candidates and the filter some out.**
- **Cost: need to maintain the number of unstable children.**

# Backward propagation with children counting

- **BPC(endgame $E$) // build endgame $E$**
  - **set the number of children in each legal position**
  - **put positions in $B_0$ and $W_0$ to the queue $Q$**
  - **while $Q$ is not empty do**
    - ▷ ***pop a position $p$ from $Q$***
    - ▷ ***if $p$ is a black-to-move position then {***
      ***UPDATE_B$_b$($p$) //$p$ is lose***
      ***put $p$'s parents whose values are changed by $p$ into $Q$ //***
      ***}***
    - ▷ ***if $p$ is a white-to-move position then {***
      ***UPDATE_W$_b$($p$) //$p$ is win***
      ***for each parent $p'$ of $p$ do***
      $nchild(p') - -$
      ***if*** $nchild(p') == 0$ ***then put $p'$ into $Q$***
      ***}***
  - **Mark all unknown positions to be draw**

# Layered Propagation of only stable positions

- **Use both the layered propagation and unknown child counting techniques.**
- **LBPC(endgame $E$) // build endgame $E$**
  - **initialize $B_0$ and $W_0$**
  - **while $B_i \neq \emptyset$ or $W_{i+1} \neq \emptyset$ do**
    - ▷ ***for each position $p$ in $B_i$ do***
      ***UPDATE_B$_b(p)$ //$p$ is lose***
      ***put $p$'s parents whose values are changed by $p$ into $W_{i+1}$***
    - ▷ ***for each position $p$ in $W_{i+1}$ do***
      ***UPDATE_W$_b(p)$ //$p$ is win***
      ***for each parent $p'$ of $p$ do***
      $nchild(p') - -$
      ***if*** $nchild(p') == 0$ ***then put $p'$ into $B_{i+1}$***
    ***}***
  - **Mark all unknown positions to be draw**

# Disk based techniques

- **Problems:**
  - **How to do UPDATE$_f$ and UPDATE$_b$ on the disk efficiently**
- **Main techniques [Hsu and Liu 2002] [Wu et al 2006]:**
  - **Do operations on a file in the disk only sequentially**
    - ▷ ***Randomly access (via $lseek$) 10,000 records in a disk takes a long time and may make the disk to have a shorter life span***
    - ▷ ***Sort and merge the locations of the 10,000 records, and then do a sequential access in ascending order takes not too much time and does not hurt the life span of the disk too much.***
  - **Batched or delayed processing**
    - ▷ ***Accumulate requests of updating and do them at once using the above techniques.***
    - ▷ ***During accumulation, you have a chance to merge all updates of a location into only one request.***

# Example

- **Assumptions:**
  - $DB[0..w]$ is stored on disk
  - In UPDATE$_f(p)$ you want to find whether all children of $p$ are win.
  - calls to UPDATE$_f(p_1)$, ... , UPDATE$_f(p_s)$
  - The children of $p_i$ are stored almost randomly in $DB[]$
- **Naive RFC-based algorithm: very slow and use the disk heavily**
  - for each UPDATE$_f(p_i)$
    - ▷ *use $lseek()$ to retrieve the content of each child of $p_i$*
- **Batched algorithm**
  - for each UPDATE_W$_f(p_i)$
    - ▷ *accumulate record the location $x$ of each child $p_i$ into an array $W$ with $W[j] = (idx = i, loc = x)$*
  - sort and merge $W$ according to the second key
  - for $i = 1$ to $|W|$
    - ▷ *use $lseek()$ to retrieve $BD[W[i].loc]$ and put it in an array $T[j] = (W[i].idx, DB[W[i].loc])$*
  - sort $T$ using the first key
  - read $T$ sequentially to where all information needed by each $p_i$ are in a continuous segment

# Previous results — Retrograde analysis

- **Western chess: general approach.**
  - **Complete 3- to 5-piece, pawn-less 6-piece endgames are built.**
  - **Selected 6-piece endgames, e.g., KQQKQP.**
    - ▷ ***Perfect information for roughly*** $7.75 * 10^9$ ***positions per endgame.***
    - ▷ ***1.5 – 3*** $*10^{12}$ ***bytes for all 3- to 6-piece endgames.***
  - **7-piece endgames were built in 2012. [140TB; http://tb7.chessok.com/]**
- **Awari: machine and game dependent approach.**
  - **Solved in the year 2002.**
  - $2.04 * 10^{11}$ **positions in an endgame.**
    - ▷ ***Using parallel machines.***
    - ▷ ***Win/loss/draw.***
- **Checkers: game dependent approach.**
  - $1.7 * 10^{11}$ **positions in an endgame.**
    - ▷ ***Currently (upto 2020) the largest endgame database of any games using a sequential machine.***
    - ▷ ***Win/loss/draw.***
    - ▷ ***Solved in the year 2007 with a total endgame size of*** $3.9 * 10^{13}$***.***
- **Many other games.**

# Results — Chinese chess

- **Earlier work by Prof. S. C. Hsu ( 許舜欽 ) and his students, and some other researchers in Taiwan.**
  - **KRKGGMM ( 俥 vs. 士 士 象 象 ) [Fang 1997; master thesis]**
    - ▷ *About $4 * 10^6$ positions; Perfect information.*
- **Memory-efficient implementation: general approach.**
  - **KCPGMKGGMM ( 炮 兵 仕 相 vs. 士 士 象 象 ) [Wu & Beal 2001]**
    - ▷ *About $2 * 10^9$ positions; Perfect information.*
  - **KCPGGMMKGGMM ( 炮 兵 仕 仕 相 相 vs. 士 士 象 象 ) [Wu, Liu & Hsu 2006]**
    - ▷ *About $8.8 * 10^9$ positions; $2.6 * 10^{-5}$ seconds per position; Perfect information.*
    - ▷ *The largest single endgame database and the largest collection reported.*
  - **Verification [Hsu & Liu 2002]**
- **Special rules: more likely to be affected by special rules when endgames get larger.**

# Problems and solutions

- **Need to solve the cycle detection and shrinking problem in a graph.**
  - **Modeling using graph theory.**
  - **Using previous knowledge from graph theory.**
- **Need to solve the problem of requiring a huge space o store the database being constructed.**
- **General technique: trading memory usage with time usage.**
  - **Using advanced encoding schemes for each position.**
    - ▷ *Limitation: 1 bit per position.*
  - **Carefully partition the database into disjoint portions so that only the needed parts are loaded into the memory.**
    - ▷ *Using combinatorial properties to do the partition.*
  - **External memory algorithms.**
    - ▷ *Disk-based algorithms.*
  - **Advanced data structures for compressions.**

# Comments

- **Almost all state-of-the-art game programs use some sorts of endgame databases.**
- **Building a large endgame database is one problem, how to use it in searching efficiently is a bigger issue.**
- **Q: Can endgames be replaced with rules similar to the one used by human experts?**
  - **Deep learning?**

# Construction of a huge knowledge base that is consistent

# Motivations

- Computing of the material values is a crucial part of a good evaluating function for Chinese chess.
- Static material values:
  - King: 100
  - Guard/Minister: 2
  - Rook: 10
  - Knight/Cannon: 5
  - Pawn: 1
- Meanings:
  - A knight is about equal to a cannon.
  - A rook is about equal to two knights, two cannons, or a cannon plus a knight.
  - Three defending pieces are better than a knight, but two of them are as good.

# Dynamic piece value

- **Values of pieces are dynamic depending on the combination.**
  - **It is better to have different types of attacking pieces.**
    - *Cannons can "jump" over pieces, rooks can attack in straight-lines, and knights can attack in a very different way.*
    - *Guards are better in protecting the king in facing a rook attack.*
    - *Guards are not good in protecting the king in facing a cannon attack.*
- **Examples:**
  - **Example 1:**
    - *KCPGMMKGGMM is a red-win endgame.*
    - *KNPGMMKGGMM is a draw endgame.*
  - **Example 2:**
    - *KPPKGG and KPPKMM are red-win endgames.*
    - *KPPKGM is a draw endgame.*
  - **Example 3:**
    - *KNPKGM and KNPKGG are red-win endgames.*
    - *KNPKMM is a difficult endgame for red to win.*

# Usage of Endgame Knowledge

- **Computer constructed endgame databases are too large to be loaded into the main memory during searching.**
  - **only useful at the very end of games.**
- **Human experts:**
  - **Studies the degree of "advantageous" by considering only positions of pawns and material combinations.**
  - **Lots of endgame books exist.**
- **What does it mean when we say a material combination $M_1$ of one side is better than $M_2$ of the other side?**
  - **Among all legal positions with $M_1 + M_2$ the side with $M_1$ has a better chance of winning.**
  - **Among all legal and reasonable positions with $M_1 + M_2$ the side with $M_1$ has a better chance of winning.**
    - ▷ ***We only consider quiescent positions.***

# Books

# Format

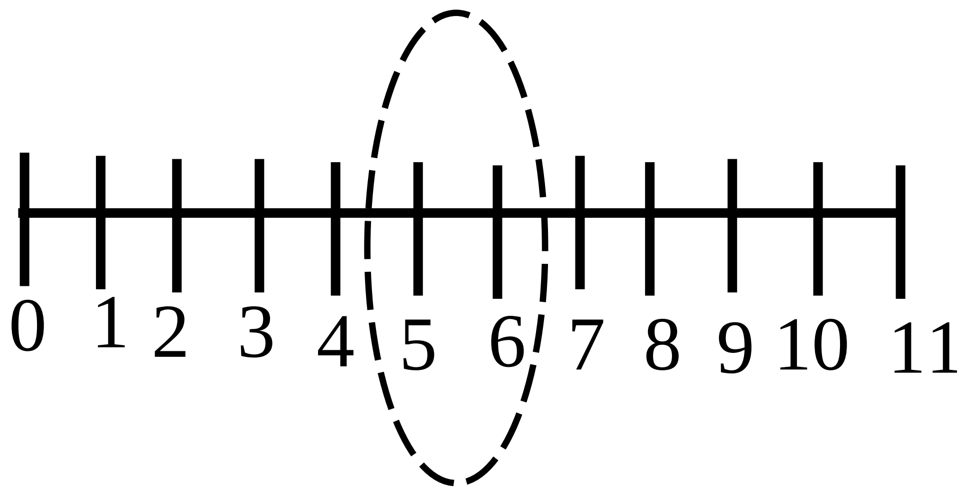
- **Granularity: 12 different levels by considering material combinations only.**
  - 紅必勝(0): The red side is almost sure to win.
  - 紅大優(1): The red side is almost sure to win, but may be draw if the black side takes a very good position.
  - 紅佔優(2): The red side has advantage, but has a chance to lose if the black side is in a very good position.
  - 紅巧勝(3): The red side may win in some good positions, but in most cases it is a draw.
  - 紅難勝(4): The red side has an advantage, but is very difficult to win.
  - 均勢(5): Either side has a chance to win, i.e., tie.
  - 必和(6): No side can win, i.e., draw.
  - 黑難勝(7): The black side has an advantage, but is very difficult to win.
  - 黑巧勝(8): The black side may win in some good positions, but in most cases it is a draw.
  - 黑佔優(9): The black side has advantage, but has a chance to lose if the red side is in a very good position.
  - 黑大優(10): The black side is almost sure to win, but may be draw if the red side takes a very good position.
  - 黑必勝(11): The black side is almost sure to win.

# Motivations

- **There are many existing heuristics about Chinese Chess endgames.**
  - **Books.**
  - **Computer records.**
  - **Annotations from human experts.**
  - $\cdots$
- **Previously, efforts are spent to collect heuristics.**
- **Now, our problem is to compile a consistent set of heuristics.**
  - **Granuality.**
  - **Errors and contradictions.**
    - ▷ *Input error.*
    - ▷ *Cognition error.*
    - ▷ *Approximation and conversion error.*
- **Questions:**
  - **How to compile a consistent set of heuristics?**
  - **How can you choose the "right" one when you have two different selections?**
  - **How can you easily detect a potential conflict?**
    - ▷ *It is difficult to be 100% sure that there is no conflict.*

# Comments

- **Numerical scale.**




- **We do not assume every endgame has a fixed value by simply considering its material combination.**
  - **Many critical endgames have different values according to their positions.**
- **It is an art to integrate the values from material combinations into the evaluating function.**

# Sources (I)

- **Books: about 10,000 combinations**

  - 象棋實用殘局
  - 新殘棋例典1, 2, 3, 4, 5, 6
  - 象棋基本實用殘局詳解
  - 圖說象棋殘局
  - 象棋殘局基礎
  - 巧勢殘局
  - 馬兵專集
  - 馬兵專集增補
  - 炮兵專集
  - · · ·

# Sources (II)

- **Computer constructed endgames: about 2,500.**
- **Endgames input by a human expert: about 17,000.**
  - **Using a web interface to manually input results of endgames with very few total number of attacking pieces.**
- **Using expert systems and rules: about 110,000.**
  - **Differ from collected endgames by one piece after removing some meaningless ones.**
  - **Bo-Nian Chen and Pangfeng Liu and Shun-Chin Hsu and Tsan-sheng Hsu, "Knowledge Inferencing on Chinese Chess Endgames,"** *Proceedings of the 6th International Conference on Computers and Games (CG)*, **Springer-Verlag LNCS# 5131, pages 180–191, 2008.**
- **Total: 140,320 out of the 2,125,764 feasible combinations.**
  - **Bo-Nian Chen, Hung-Jui Chang, Shun-Chin Hsu, Jr-Chang Chen and Tsan-sheng Hsu, "Multi-Level Inference in Chinese Chess Endgame Knowledge Bases," International Computer Game Association (ICGA) Journal, volume 36, number 4, pages 203–214, December 2013.**
  - **Bo-Nian Chen, Hung-Jui Chang, Shun-Chin Hsu, Jr-Chang Chen, and Tsan-sheng Hsu,"Advanced meta-knowledge for Chinese Chess Endgame," International Computer Game Association (ICGA) Journal, volume 37, number 1, pages 17–24, March 2014.**

# Problems

- **Human mistakes.**
  - **Different conclusions from different sources, e.g., books.**
    - ▷ *Different conclusions were made in different eras.*
    - ▷ *Different conclusions were made by different authors.*
    - ▷ *Some books discuss an endgame extensively with detailed positions, but have no general conclusions.*
- **Algorithmic mistakes.**
  - **Our algorithm for computer inferred endgame values has a roughly 90% of correctness.**
- **Granularity.**
  - **Some books only record results using a win-loss-draw format, not in 10 levels as we do.**
  - **Perfect endgame databases obtained by retrograde analysis contain winning rates, not a 12-level value.**
    - ▷ *How to convert rates to levels?*

# How to detect conflicts – Basics

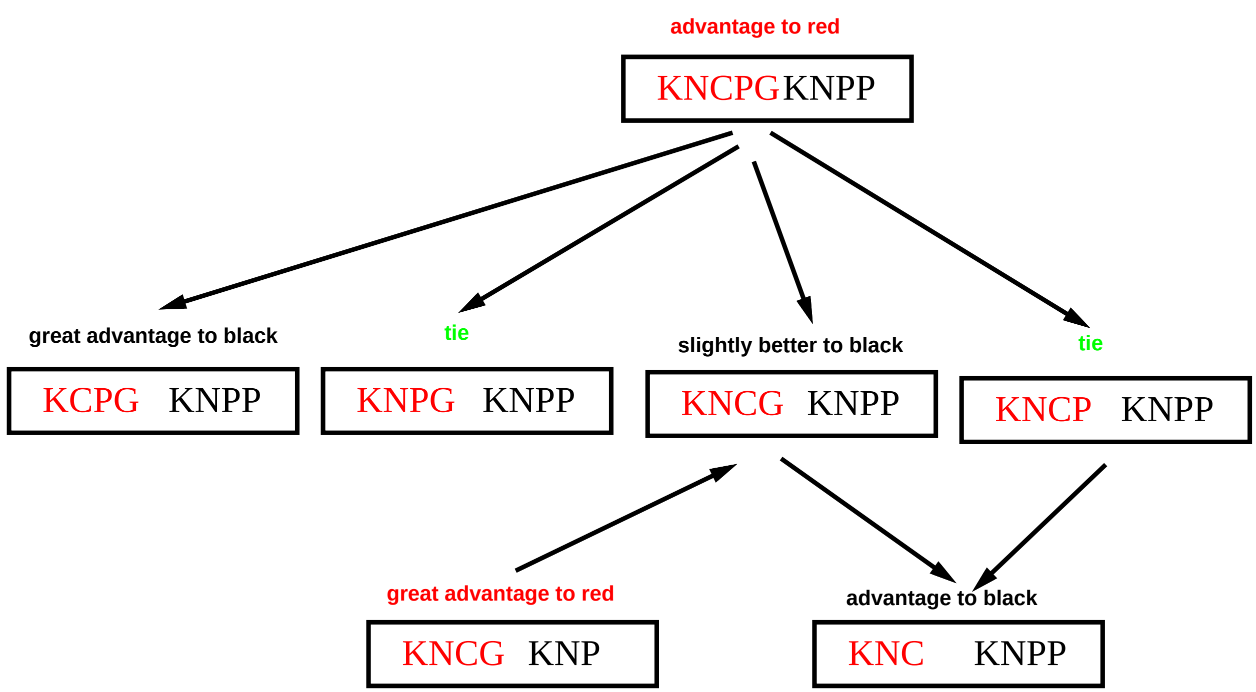
- **Piece additive rule:**
  - **The result of an endgame cannot get worse by**
    - ▷ *gaining extra pieces on your side;*
    - ▷ *losing pieces on your opponent's side.*
  - **The result of an endgame cannot get better**
    - ▷ *by losing pieces on your side;*
    - ▷ *if your opponent gains piece.*
- **Rule of defensive pieces, i.e., Elephant and Guard.**
  - **The result of an endgame cannot normally be greatly changed by gaining/losing an extra defensive piece.**
- **Rule of draw and tie:**
  - **It is a draw if no side can win.**
  - **It is a tie if either side can win.**
  - **An endgame cannot usually be turned from tie into draw by using the piece additive rule.**

# How to detect conflicts – Process

- **Procedure: check rules for endgames that we have already collected.**
  - **Piece additive rule.**
  - **Rule of defensive pieces, i.e., Elephant and Guard.**
  - **Rule of draw and tie.**
- **Using relations between endgames, not just endgames themselves to check for potential conflicts.**
  - **Similar activities applied for human cognitive process.**

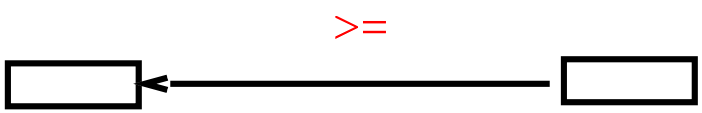
# A graphic view

- **A graph theoretical model.**
  - **vertex: an endgame**
  - **edge: between two vertices $u$ and $v$ if they follow the piece additive rule.**
    - ▷ *the direction from $u$ to $v$ if the value of $u$ must be no worse than that of $v$.*

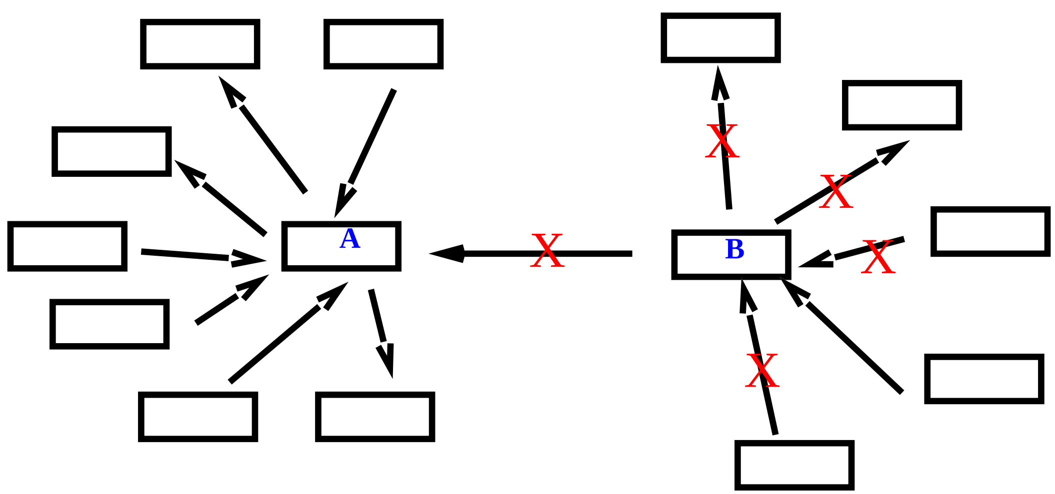


# High level ideas

- Assume the major part of the heuristics are correct.
- A conflict is an edge such that the values between them does not follow the piece additive rule.

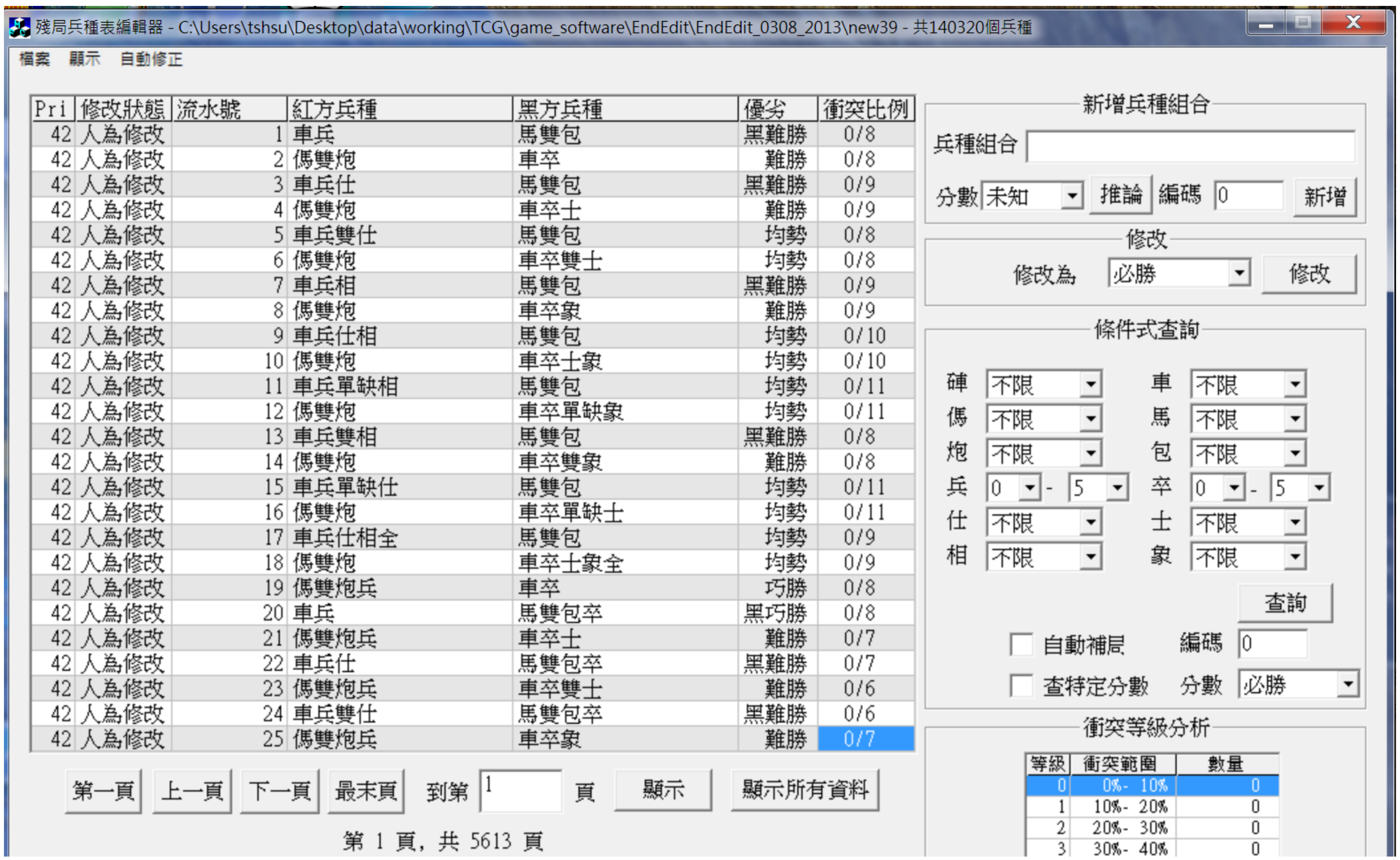



- The vertex who has a large percentage of conflicts is more likely to be incorrect.



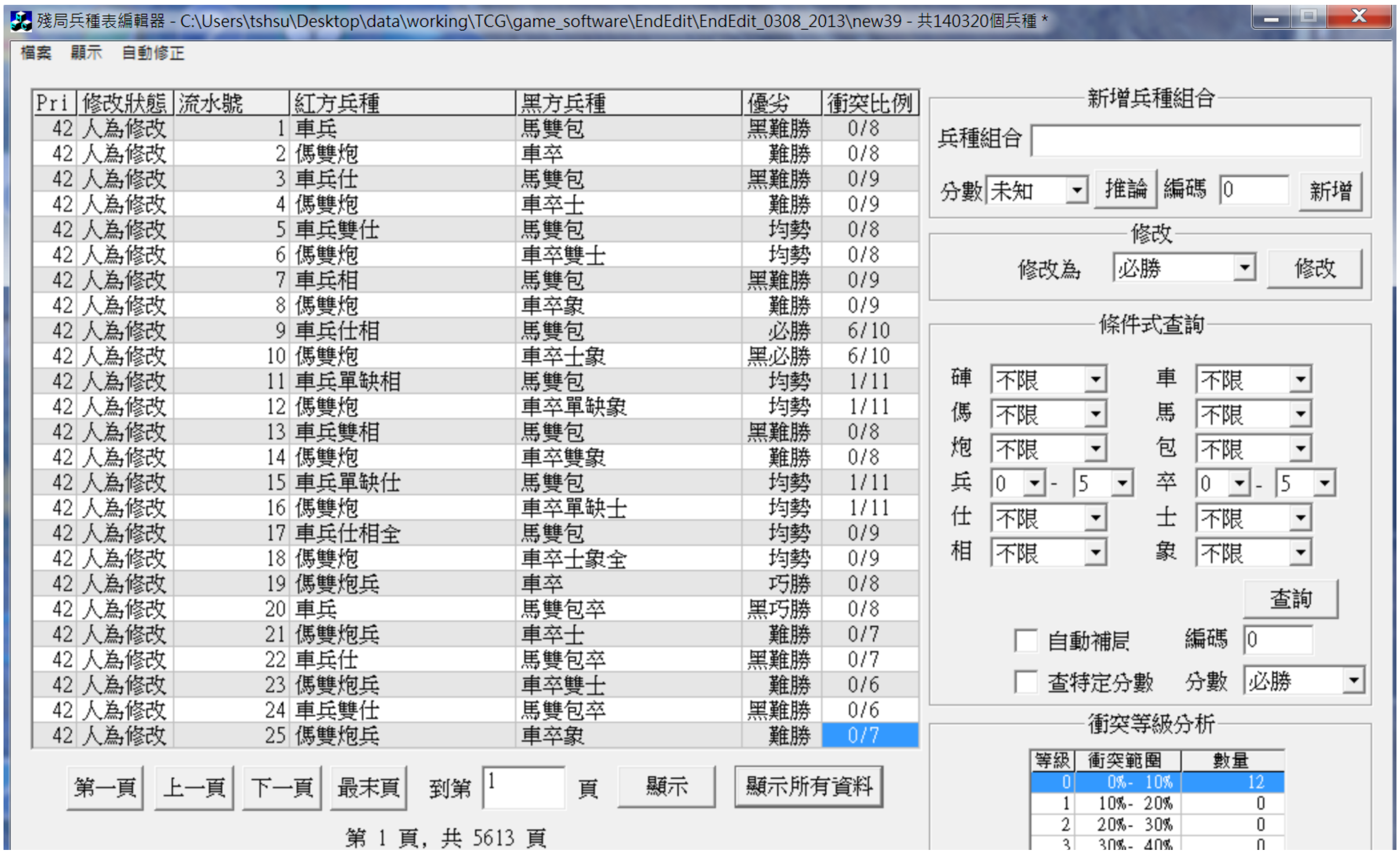
# EndEdit (1/3)

- **Build a software tool to process and find conflicts.**
- **EndEdit.**
  - **140,320 combinations.**



# EndEdit (2/3)

- **Change the value for KRPGMKNCC from tie to sure win.**



# EndEdit (3/3)

- **Conflicts for KRPGMKNCC.**

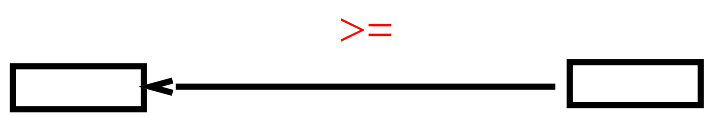
殘局兵種表編輯器 - C:\Users\tshsu\Desktop\data\working\TCG\game_software\EndEdit\EndEdit_0308_2013\new39 - 共1

檔案 顯示 自動修正

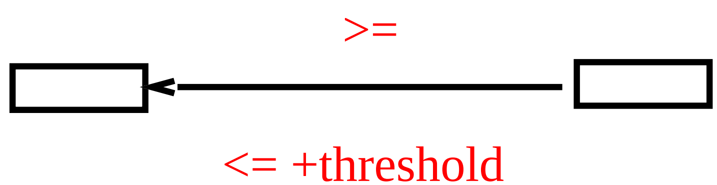
| Pri | 修改狀態 | 流水號 | 紅方兵種 | 黑方兵種 | 優劣 | 衝突比例 |
|---|---|---|---|---|---|---|
| 63 | 未修改 | 110425 | 車炮兵仕相 | 馬雙包 | 大優 | 1/12 |
| 64 | 未修改 | 18181(0) | 炮兵仕相 | 馬雙包 | 黑大優 | 0/11 |
| 42 | 人為修改 | 9(X) | 車兵仕相 | 馬雙包 | 必勝 | 6/10 |
| 64 | 人為修改 | 106475(0) | 車炮仕相 | 馬雙包 | 巧勝 | 0/14 |
| 64 | 未修改 | 109152(0) | 車炮兵仕 | 馬雙包 | 優勢 | 0/12 |
| 64 | 人為修改 | 109998(0) | 車炮兵相 | 馬雙包 | 優勢 | 0/12 |
| 66 | 未修改 | 110267(0) | 車炮兵仕相 | 雙包 | 必勝 | 0/16 |
| 64 | 未修改 | 110373(0) | 車炮兵仕相 | 馬包 | 必勝 | 0/17 |
| 43 | 人為修改 | 110426(0) | 車炮兵仕相 | 馬雙包士 | 優勢 | 0/8 |
| 43 | 人為修改 | 110428(0) | 車炮兵仕相 | 馬雙包卒 | 巧勝 | 0/11 |
| 64 | 未修改 | 110833(0) | 車炮兵單缺相 | 馬雙包 | 大優 | 0/13 |
| 64 | 未修改 | 111689(0) | 車炮兵單缺仕 | 馬雙包 | 大優 | 0/12 |
| 63 | 未修改 | 70126(0) | 傌雙炮 | 車包卒士象 | 黑大優 | 1/12 |

# Enhanced ideas

- **A potential conflict is an edge such that the difference in values between the endpoints is more than a threshold, say 3, and the two connected endgames follow the rule of the defensive pieces.**
- **Original relation.**




- **Enhanced relation.**

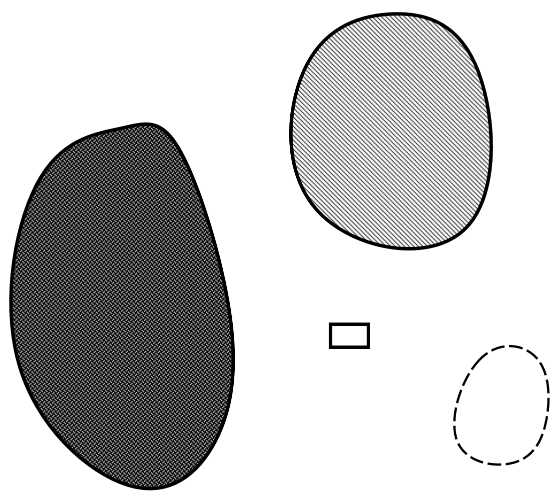


# Algorithm

- **For each endgame, compute the percentage of conflicts, which is the ratio between the number of "corrected" relations and the number of total relations.**
- **Identify the ones with the large percentage of conflicts and either use human or an automatic procedure to re-assign its value.**

# Potential problems for our approach

- A cluster of endgames all with consistent errors.
- A sparse or isolated cluster where inter-relation is few.
- A vertex can have a wide range of possible values due to the fact the values of its neighbors are much higher or lower than it.
- Solution: randomly select endgames in different clusters and verify them by human experts.



# Remarks

- **Out of about 140,320 endgames, there are about 1/12 severe errors (ones whose corrected values differ from the original values by at least 3).**
- **A total of 1/3 endgames are revised.**
- **A period of 1 year is spent to obtain a consistent set of heuristics where it is almost impossible to manually check the consistency of the collection of endgames previously.**

# Ongoing work

- **More testing and analysis are needed.**
- **Using graph theoretical techniques to further process the data.**
  - **Assign a different weight to a different type of relations, and use the weight to find the ones that are most likely to be incorrect.**
  - **More inferencing rules that are not just between direct neighbors.**
    - *Piece exchanges: you cannot get better by exchanging a stronger piece with a weaker piece.*
    - *Depending on the piece involved, assign a confidence factor, e.g., adding a rook and adding a knight have different levels of confidence.*
- **Use expert system to do a better job in self-correcting.**
- **Test how much it can improve the performance of a Chinese chess program.**
- **Further usage:**
  - **Tutoring**
  - **E-learning**
  - **Knowledge abstraction**

# Concluding remarks

- **Open game and endgame databases provide a chance to work off-line before the tournament.**
- **Need to balance between the amount of storage used and the effort to put them into real-time usage.**
  - **If we can load the content of an endgame into the memory and use them while doing search, then it is equivalent to a perfect transposition table.**
  - **Problem: Too large to be fitted into.**
  - **Current status: only use it at the root.**
  - **A very good research opportunity.**
- **Endgame databases provide a gold mine for doing knowledge abstraction.**

# References and further readings


- M. Buro. Toward opening book learning. *International Computer Game Association (ICGA) Journal*, 22(2):98–102, 1999.
- T.-s. Hsu and P.-Y. Liu. Verification of endgame databases. *International Computer Game Association (ICGA) Journal*, 25(3):132–144, 2002.
- P.-s. Wu, P.-Y. Liu, and T.-s Hsu. An external-memory retrograde analysis algorithm. In H. Jaap van den Herik, Y. Björnsson, and N. S. Netanyahu, editors, *Lecture Notes in Computer Science 3846: Proceedings of the 4th International Conference on Computers and Games*, pages 145–160. Springer-Verlag, New York, NY, 2006.
- B.-N. Chen, P.F. Liu, S.C. Hsu, and T.-s. Hsu. Knowledge inferencing on Chinese chess endgames. In H. Jaap van den Herik, X. Xu, Z. Ma, and M. H.M. Winands, editors, *Lecture Notes in Computer Science 5131: Proceedings of the 6th International Conference on Computers and Games*, pages 180–191. Springer-Verlag, New York, NY, 2008.

- B.-N. Chen, P.F. Liu, S.C. Hsu, and T.-s. Hsu. Conflict resolution of Chinese chess endgame knowledge base. In *Lecture Notes in Computer Science 6048: Proceedings of the 12th Advances in Computer Games Conference*, pages 146–157. Springer-Verlag, New York, NY, 2010.
- B.-N. Chen, P.F. Liu, S.C. Hsu, and T.-s. Hsu. Knowledge abstraction in Chinese chess endgame databases. In *Lecture Notes in Computer Science 6515: Proceedings of the 7th International Conference on Computers and Games*, pages 176–187. Springer-Verlag, New York, NY, 2011.
- B.-N. Chen, P.F. Liu, S.C. Hsu, and T.-s. Hsu. Aggregating consists endgame knowledge in Chnese Chess. *Knowledge-Based System*, volume 34, pages 34–42, October 2012.