ICGA JOURNAL

INTERNATIONAL COMPUTER GAMES ASSOCIATION

Vol. 38
No. 1
March 2015
Copyright 2015, ICGA

Published by
Leiden Centre of Data Science (LCDS)
Leiden University
Leiden, The Netherlands

Universiteit Leiden
Leiden Centre of Data Science

ISSN 1389-6911

SOME INFORMATION ABOUT THE INTERNATIONAL COMPUTER GAMES ASSOCIATION (ICGA)

The ICCA (International Computer Chess Association) was founded in 1977 and represents the Computer Chess World vis-à-vis Computer Science Organizations, such as ACM and IFIP, and also vis-à-vis the International Chess Federation (FIDE). In 2002 the name of the Association was changed into International Computer Games Association (ICGA), thus incorporating the International Computer Chess Association (ICCA). In the same way the ICGA also represents the Computer Games world vis-à-vis the various international games federations for games other than chess.

World-wide membership comprises individuals as well as university and industrial members.

The activities of the ICGA are:

(i) to publish a quarterly *ICGA Journal*;
(ii) to hold regular World Computer-Chess Championships, Computer Olympiads, and *Advances in Computer Games* conferences;
(iii) to strengthen ties and promote cooperation among computer-games researchers;
(iv) to introduce computer games to the games world;
(v) to support national computer-games organizations and computer-games tournament organizers.

The ICGA takes pride in listing the roster of the ICCA Past Presidents:

1977 - 1983	Benjamin Mittman
1983 - 1986	Monroe Newborn
1986 - 1992	David Levy
1992 - 1999	Tony Marsland
1999 - 2012	David Levy

THE ICGA JOURNAL

THE EDITORIAL BOARD : Prof.dr. H.J. van den Herik (Editor-in-Chief)
Prof.dr. A. Plaat (Deputy Editor)
Prof.dr. J. Schaeffer (Associate Editor)
Prof.dr. I. Althöfer (Section Editor)
Dr. C. Browne (Section Editor)
Dr. A.S. Fraenkel (Section Editor)
Dr. G.M^{c}C. Haworth (Section Editor)
Prof.dr. H. Iida (Section Editor)
Dr. A. Kishimoto (Section Editor)
Dr. G. Tesauro (Section Editor)
Dr. M. Winands (Section Editor)
Prof.dr. I-C. Wu (Section Editor)
Prof.dr. Xinhe Xu (Section Editor)

EDITOR-IN-CHIEF : Prof.dr. H.J. van den Herik
Leiden University, Faculty of Science / LIACS
Leiden Centre of Data Science (LCDS)
P.O. Box 9512, 2300 RA Leiden, the Netherlands
Email: jaapvandenherik@gmail.com

EDITORIAL MANAGER : Johanna Hellemons. Email: journal@icga.org

The setting of the copy for this issue was performed at the Leiden University. The benevolence of the University Authorities is recognized for sponsoring part of the secretarial staff. Contributors are encouraged to read the **Information for Contributors** on page 54.

TABLE OF CONTENTS

ARIMAA: GAME OVER

Three names, three stories and deep admiration from the Editorial Board. That is the main contents of this issue. The three names are: Omar Syed, David Fotland, and David Wu. The protagonist of our euphoria is undoubtedly David Wu. In 2011, he obtained his B.Sc. degree under the supervision of David Parkes (Harvard University) by delivering the thesis "Move Ranking and Evaluation in the Game of Arimaa". Earlier, in 2008, David's program SHARP had entered the Arimaa Computer Championship for the first time and came second behind David Fotland's program BOMB. In 2011 and 2014 SHARP won the Arimaa Computer Tournament, but not the contest against the best human players of that time.

However, in 2015 SHARP won the Computer Tournament as well as the Arimaa Challenge (see pp. 3-11). A fantastic performance, which is in some sense comparable to DEEP BLUE's victory over Kasparov (1997), and WATSON defeating Ken Jennings and Brad Rutter in JEOPARDY! (2011). In Chess and JEOPARDY! the initiative was in the hands of IBM, and the scientific progress was recognized worldwide.

Now the ICGA community is proud to have its own challenging game owing to the initiative by Omar Syed. In 2003, Omar and his son Aamir Syed published an idea in the *ICGA Journal* on a new game, called Arimaa and offered a prize of US $10,000 for the programmer who developed before the year 2020 a program that defeated the top human Arimaa players. The game looked very difficult for computers (and humans!). Experienced Go programmer David Fotland did the "opening" moves. His program BOMB won the first five Arimaa Computer Tournaments. However, BOMB never had any chance of winning the Arimaa Challenge against the top human players involved.

All in all, your Editor is pleased with the contribution in this issue by Omar Syed in which he reports the Arimaa story: from Inception to Completion. It is breathtaking to read how he arrived at the idea of Arimaa, how he continued the idea by organising a series of tournaments, and then seeing how his idea was realised in a tough competition.

The crown on his idea came, to his own surprise, already in 2015. The achievements of the program developers are well expressed by David Fotland (five-time winner of the computer tournament) and David Wu (three-time winner of the computer tournament and now (2015) also winner of the contest against the top human players).

Although the Arimaa community is not as large as the chess or the JEOPARDY! communities, the success of David Wu's SHARP did not remain unnoticed to the great game community. A search on Google will bring you to a great variety of descriptions of this success.

[1] The Editor gratefully recognizes the cooperation with Kingpin Chess Magazine's Editor Jonathan Manley and author Andy Lewis, in particular he appreciates their permission to reproduce the slightly adapted version of this article.

Of course, “Arimaa: Game Over” is the most evident title for this editorial. However, some remarks are in order. Currently, the term Game Over is used frequently in video games. There are hundreds of designs with the message Game Over. The message was often used for pinball games, and later for Arcade games, meaning “your game is over”. Later on, the expression tended to have a somewhat negative meaning. Nowadays, it is almost an equivalent for “It’s all over”.

However, in our games community we know that it is not so. The last scientific challenge, establishing the game-theoretical value of Arimaa, is still waiting for its solution. Arimaa is now in the class of games in which computers outperform the best human players, just like chess. Yet, the ultimate question is: can we solve the game? As readers of this Journal know we distinguish between weakly solving and strongly solving a game (introduced by Allis, 1994). So, we would like to encourage all researchers to continue the development of advanced techniques to find the ultimate truth of Arimaa.

After completing my Editorial, I was happy with the contents and the title. For curiosity, I checked at Google whether this title was used earlier. Then I arrived at the Kingpin site and found Andy Lewis’ article. David Levy brought me in contact with Jonathan Manley (their Editor) and Andy Lewis. The result is to read on pp. 55-62. Thank you both for the fast cooperation.

Next to the Arimaa euphoria we have still maintained the character of the Journal by publishing two notes, one by Guy Haworth on Chess Endgame News. The second note is on research on the computational complexity of *n* x *n* Chinese Chess by Qiang Gao and Xinhe Xu.

We wish our readers a pleasant reading time and apologize for the late appearance of this issue. We promise to provide you with more interesting news soon.

Reference
Allis, L.V. (1994). *Searching for Solutions in Games and Artificial Intelligence*. Ph.D. Thesis, Rijksuniversiteit Limburg, Maastricht, the Netherlands.

Jaap van den Herik

The credits of the photographs in this issue are to: Fritz Juhnke, Jean Daligault, Mathew Brown, Omar Syed & David Wu, and to Kingpin Chess Magazine.

It is also possible to receive your membership copy in electronic form. Please, arrange the change of your subscription with our Editorial Manager Johanna Hellemons.

ICGA Journal readers who are interested in information on our publications are referred to our website. A complete list of all articles, notes, and literature reviews published in the *ICCA Journal* and the *ICGA Journal* is accessible on the Internet at http://www.icga.org

David Wu

GAME OVER?

THE ARIMAA CHALLENGE: FROM INCEPTION TO COMPLETION

Omar Syed[1]

Murphy TX, USA

ABSTRACT

In 2003, I published a paper in the *ICGA Journal* describing the game Arimaa and offering a challenge prize of $10,000 USD to develop a program capable of defeating the top human player before the year 2020. In 2015, David Wu developed a program called SHARP which won the Arimaa Challenge. This article will discuss my perspective of the Arimaa Challenge over the years from inception to completion.

1 THE INCEPTION

Arimaa was inspired by the DEEP BLUE vs Kasparov match of 1997. It started as a curiosity to see if it might be possible for humans to continue staying ahead of computers if the game still used a standard chess set, but with different rules. My initial attempts at creating such a game simply added multiple moves per turn to Chess in order to increase the branching factor. I soon realized that although such games might be harder for computers, they were probably even more harder for humans due to the large amount of change that can occur from one move to the next. It was not until I was teaching my then four year old son Aamir how to play Chess starting with just the kings and pawns that I noticed using simple one-step moves per turn could also provide a high branching factor while keeping the amount of change between turns manageable for humans. We named the game after my son; Arimaa is Aamir spelled backwards but with a leading A. Even after this breakthrough, it still took quite a bit of experimentation and playtesting to find a set of rules that made the game interesting while not having any obvious flaws. The game, along with the challenge, was announced through the Arimaa web site on November 20, 2002.

A primary motivation for the challenge was my feeling that the spectacular chess match of 1997 was done in a way that was not only unfair to Kasparov, but also did not provide any benefits to the scientific community or even a playable engine for the chess community. It was my hope that Arimaa, along with a well defined annual Arimaa Challenge match, would provide the next feasible target, which if achieved would provide some benefit to the scientific community. The challenge was offered only until 2020, since I wanted the challenge to be won by a software breakthrough and felt that advances in hardware would lead to the challenge being won shortly after 2020, even without any significant advances in software. To ensure that any breakthrough was made available to the scientific community, a condition of receiving the challenge prize was to publish a paper in the *ICGA Journal* describing the winning program. A second requirement of the Arimaa Challenge was that all submitted programs would be permanently saved and be made playable in the online gameroom after the event for next years' programs to train against. It also allowed humans to practice against the current best engines. All games played in the gameroom were archived and made available for research.

2 THE AI-RESEARCH COMMUNITY

Soon after the challenge was announced, several members of the game AI-research community attempted to tackle the challenge. The late Don Dailey (developer of the KOMODO chess engine) was the first to create an Arimaa engine. Within a couple months Don had a surprisingly strong program offering games in the Arimaa gameroom. In designing Arimaa, I had used the Zillions-of-Games general game-playing engine, which allows specifying the rules of the game in a Lisp like language, and then immediately being able to play the game against the Zillions engine. Using only look ahead and no game-specific knowledge, this engine was able to play at a strong level in most games one could dream of. However, it was incredibly lousy at Arimaa due to the high branching factor. This helped build my confidence that Arimaa would be a difficult game for computers if only a brute-force search was used. However, Don's program, called OCCAM, surprised me in how well it played when the search engine was much faster and included some game knowledge. When Don realized that

[1] Email: osyed1@gmail.com

there was very little expert knowledge available for Arimaa, he felt that he could not make much progress on his program and went back to developing his chess engine which now has gone on to become the highest rated in the world.

Don mentioned to me that he had heard about Arimaa through a forum post by David Fotland. When I looked up David Fotland it turned out that he was not only the developer of a world class Go engine, but had also attended Case Western Reserve University, my alma mater. I contacted David to see if he might be interested in developing an engine for Arimaa. He replied back that he could probably win the challenge in the first year, but that the prize was not enough to make it worth the effort he would need to put in. I offered to double the challenge prize for him if he could do it in the first year. After accepting the offer, he explained to me that the Arimaa Challenge could potentially be won in the first year because it typically takes a human player 5 years to reach masters level in deep games like Chess and Go, while computers, in contrast, can be programmed to play better than a beginner within a year. He was right; and this is something that I had not considered before offering the challenge. For all practical purposes, the Arimaa players at the time, including myself, were just beginners. This became quite obvious when David's program called BOMB climbed the playing-strength ladder and passed all of us within a few months and became the highest rated player. Fortunately, Claude Chaunier, the former Lines-of-Action world champion, became interested in Arimaa and discovered some strategies that prevented the game from becoming too tactical and made it more favorable for humans.

3 THE FIRST CHALLENGE MATCH

The first challenge match occurred in early 2004. BOMB easily defeated the competitors in the computer championship to win the rights to play in the challenge match. Although the challenge rules allowed me to pick any human player to defend the challenge, I was perhaps as good a defender as anyone else, since we were all pretty much beginners. So I decided to defend the first challenge match myself. Since BOMB had played many online games and the human players learned some of its weaknesses, I was able to win all of my challenge games; even giving a rabbit handicap in the final game. None the less every game was nerve wrecking and felt like a marathon, with some of the games lasting almost eight hours.

To defend the second challenge match, I asked Frank Heinemann, the winner of the first world championship. He was hesitant to do it since it was a long series and would put him under pressure to perform his best throughout. After much discussion, he finally accepted. Again, BOMB was the dominant program at the time and became the challenger. Frank was able to defend the challenge with an almost perfect record, losing only the fourth game.

The original challenge was structured to be an eight game series against a single human opponent with the requirement that the computer must win the series to win the challenge with a tie being in favor of the human defender. After the experience gained from two years of organizing the challenge match and actually playing one of the matches, I realized that the physiological pressure of defending the challenge was a bit too much for a single human player. Also, playing eight games was quite a burden. Starting with the third challenge match, the format was changed and has stayed the same ever since. In the new format, there are three human defenders each playing a three-game series with the computer, which needs to win the series against each of the human defenders in order to win the challenge. To earn the right to play in the challenge match, the top-two programs from the annual computer championship played against human players for a few weeks, referred to as the screening period. The program with the best record against humans would go on to play in the challenge match. This served a dual purpose; first, it ensured that the computer playing in the challenge match was not just strong against other computers, but also against human players. Second, it provided the challenge match defenders with some information on how the computer would play against humans. The challenge defenders are not allowed to play against the programs during the screening period.

4 THE CHALLENGE RULES

Defining the challenge rules to be fair to both the defenders and the challengers is like trying to balance an asymmetric game. The defenders would like to play the computer as much as possible before the challenge or at least have it play many games against others so they can learn about its weaknesses. The challenge contenders will prefer to have as little exposure of their program as possible so that its games and style cannot be studied by the humans. This was the case in the DEEP BLUE vs Kasparov match, where the latest version of DEEP BLUE had not played any public match games prior to the match against Kasparov. This was clearly not fair to Kasparov, who had been playing in tournaments prior to the match and having his games analyzed by

DEEP BLUE. On the other extreme was the 2002 and 2006 DEEP FRITZ vs Kramnik match, in which Kramnik had complete access to prepare with the exact same program he would play against well in advance of the match. The Arimaa Challenge attempted to find a fair balance between what the defenders want and what the challengers want. The complete archive of all games played in the gameroom are available for download and updated weekly so that challengers can make use of this information. Prior to the challenge, the games from the computer tournament and the screening period provide the challenge defenders with some information about how the current version plays. Previous versions of all programs that participated in the computer tournament are always available for practice.

A more subtle issue with the Arimaa Challenge is that the human defenders are not offered any compensation for their time and effort in defending the challenge match and only do it for the pride and glory of having defended the challenge. Thus, it would not be very difficult for one to exploit this by offering the defenders compensation to not win their series. Having three defenders instead of just one and announcing the challenge defenders only shortly before the start of the challenge helps to increase the security, but if the stakes were high this would not be enough. As is obvious from sports betting if the stakes are high enough, it does not matter how much you pay the players, the outcomes can be rigged. So having a moderately small challenge prize helps to keep the stakes low and avoid such issues.

5 THE PROGRAMS INVOLVED

When BOMB was not able to win the challenge after the first two years, David Fotland began to lose interest in tackling the challenge. He made some experimental changes to BOMB in the third year, which actually made it weaker than the previous version. After the third year he stopped making any enhancements to BOMB. However, the 2005 version of BOMB continued to dominate the computer tournament, winning it the first five years and playing in the challenge match each of those years. Table 1 shows all the programs that attempted to try for the Arimaa Challenge by entering the computer championship tournament as a first step.

There were many more programs that played in the online gameroom but did not enter the computer championship. Some developers with notable prior achievements include:

- Toby Hudson, developer of a former world champion RoShamBo program
- Jeff Bacher, developer of the world champion Octi program
- Martin Piotte and Martin Chabbert, winners of the Netflix million dollar challenge
- Brian Haskin, a contributor to the Google AI challenge and author of a detailed study of the Arimaa branching factor
- Haizhi Zhong, a Phd student under Jonathan Schaeffer of the renowned University of Alberta games research group

In 2008, David Wu first entered his program called SHARP into the annual computer championship tournament. SHARP entered the Arimaa scene with a bang by placing second in the event, losing only to BOMB and earning its way to the screening period. It managed to win 4 out of 16 games against human opponents during the screening period, but BOMB won 9 out of 16 to move on to the challenge match. Nonetheless, it was an impressive performance for a new program.

In 2009, Jeff Bacher swept the events with his program CLUELESS, winning the computer championship and the right to play in the challenge match. CLUELESS had been entering the event each year since 2005, but had not been able to topple BOMB. In 2009 CLUELESS was enhanced to start using multiple processors and was able to surpass BOMB. This marked the turning point for Arimaa programs making use of multiple cores. Before CLUELESS, the programs were all single threaded. After 2009, all programs that won the computer championship were multithreaded. This marked the end of the road for BOMB. It was not only single threaded, but played a deterministic game using the same seed every time. Interestingly, GNOBOT, which made use of book learning by keeping track of all games played in the gameroom, was able to defeat BOMB that year by playing the same sequence of moves that CLUELESS played two rounds earlier.

Program	Developer	Computer Tournament Years	Challenge Years
OCCAM	Don Dailey	2004, 7	
BOMB	David Fotland	2004*, 5*, 6*, 7*, 8*, 9, 10, 11	2004, 5, 6, 7, 8
GNOBOT	Toby Hudson	2004, 5, 6, 7, 9, 10	
CLUELESS	Jeff Bacher	2005, 6, 7, 8, 9*, 10, 11, 12, 13, 14, 15	2009
LOC	Gerhard Trippen	2005, 6, 7, 8	
AAMIRA	Paul Pogonyshev	2006, 7	
ZOMBIE	Evan Dorn	2007, 8, 9	
FAERIE	Evan Dorn	2007	
SHARP	David Wu	2008, 9, 10, 11*, 12, 13, 14*, 15*	2015*
OPFOR	Brian Haskin	2008, 9, 10, 11	
BADGER	Paul Pogonyshev	2009, 10	
RAT	Gerhard Trippen	2009	
MARWIN	Mattias Hultgren	2010*, 11, 12, 13, 14, 15	2010, 11, 13
PRAGMATICTHEORY	Martin Piotte and Martin Chabbert	2010	
BRIAREUS	Ricardo Barreira	2011, 12*	2012
LUCY	Nathan Blaxall	2012	
ZILTOID	Ricardo Barreira	2013*, 14	2014
DRAKE	Tomas Hrebejk	2014	
Z	Nathan Blaxall and Ricardo Barreira	2015	
JUMBO	Peter Mckenzie	2015	
WEISER	Nicolas Barriga	2015	
NWORBA	Mathew Brown	2015	

* = won the event

Table 1: Programs Participating in the Arimaa Challenge.

6 THE HARDWARE USED: UNIFORM PLATFORM

During the computer championship, all programs run on the same hardware. This ensures that the real difference between the competing programs is the algorithms they use and how well they make use of the available hardware. The hardware is chosen to be off-the-shelf commodity hardware that can be purchased within $1000 USD at the time of the event. After the computer championship, the same hardware is used for the challenge match. Table 2 shows the hardware and operating systems that were used over the years.

Year	CPU brand, GHz, cores, total GHz	Memory GB	Disk GB	OS version, bits
2004	Intel P4, 2.4, 1, 2.4	0.5	40	RedHat v9.0, 32
2005	Intel P4, 2.8, 1, 2.8	0.5	80	RedHat ES3, 32
2006	Intel P4, 3.0, 1, 3.0	1	120	Fedora 3, 32
2007	Intel PD, 3.4, 2, 6.8	2	160	Fedora 5, 32
2008	Intel E6420, 2.1, 2, 4.2	2	160	CentOS 5, 32
2009	Intel Q8200, 2.3, 4, 9.3	2	160	CentOS 5.2, 32
2010	Intel Q9550, 2.8, 4, 11.3	2	250	CentOS 5.4, 32
2011	Intel X3360, 2.8, 4, 11.3	4	500	CentOS 5.5, 64
2012	Intel E3-1220, 3.1, 4, 12.4	4	500	CentOS 5.7, 64
2013	AMD 4280, 2.8, 8, 22.4	8	60 SSD	CentOS 6.3, 64
2014	Intel L5639, 2.1, 12, 25.5	32	1000	CentOS 6.5, 64
2015	Intel X5650, 3.0, 12, 36.7	48	500	CentOS 6.5, 64

Table 2: Hardware and OS used in the Arimaa Challenge.

7 MULTI-CORE PROCESSORS

Once the Arimaa programs began using multi-core processors, they were destined to eventually surpass the top human players. But the big question of the Arimaa Challenge was: Could it be done before the year 2020 by finding a software breakthrough?

The high branching factor of Arimaa ensured that a brute force approach would not be feasible, and strong programs would need to incorporate a high level of pruning beyond just minimax search with alpha-beta pruning. When a high level of pruning is needed, the evaluation function would need to be quite complex to select good lines worth investigating and even determining the order in which to investigate them. It is possible that such an evaluation function could become unmanageable for humans to hand code. One approach to managing this would be to use some form of machine learning to develop the evaluation function. My hope was that any machine learning algorithm used to assist the development of the evaluation function could potentially be used for other games and applications beyond just Arimaa.

8 STRATEGIES AND TACTICS PUBLISHED IN BOOKS

A wise man once said: “We create our own demons”. For Arimaa, this came in the way of documented knowledge about the finer points of the game. The first M.Sc. thesis was by Christ-Jan Cox (2006) titled *Analysis and Implementation of the Game Arimaa*. The top human Arimaa players also began documenting and sharing the knowledge they had gained about Arimaa. David Fotland (2006) did so in the Computer and Games Conference in Ramat-Gan, Israel with the title *Building a World-Champion Arimaa Program*. The activities by the top programmers and top players also resulted in an online wikibook, as well as physical books about Arimaa. In 2009, Fritz Juhnke, a former world champion, published *Beginning Arimaa: Chess Reborn Beyond Computer Comprehension*. The book covered some of the early history of Arimaa as well as the current state of Arimaa strategies and tactics. In 2012, Jean Daligault, also a former world champion, published *Arimaa Strategies and Tactics*. The book focused exclusively on how to play Arimaa well, capturing many principles to help one become better at Arimaa faster than had been possible for the early players. This was precisely the kind of game knowledge that Don Dailey was looking for to incorporate into his program. This burst of knowledge reignited the arms race between the humans and computers, which had simmered down after the humans had surpassed BOMB. It was clear that the new generation of programs after 2009 were benefiting not only from using multi-core processors, but also by incorporating the new game knowledge that was becoming available. As David Fotland had pointed out, it is much faster to incorporate game knowledge into a computer than it is for humans to absorb it. For humans, there is the additional problem that even if game knowledge is available, not everyone has the talent to absorb it, even if they put in the time and effort. I, for one, will probably never reach an Arimaa rating of 2500 like that of some of the top players, no matter how much effort I put in. Would Arimaa be able to attract the kind of talent found in games like Chess and Go soon enough to help keep the humans ahead of computers? Part of the reason Juhnke and Daligault wrote the books was to encourage talented players from other games to consider Arimaa. However, I have come to realize now that it might be easier to get a devoted strategy game player to change religions than to change the game they play.

9 SCIENTIFIC PUBLICATIONS AND A BREAKTHROUGH

The years 2009 to 2015 were a vibrant period for Arimaa program development. Not one program seemed to dominate the stage, and the championship crown was passed around almost every year. Various different approaches were tried with varying degree of success. Over thirteen academic papers, thesis and technical reports researching Arimaa were published during this time. The MCTS approach, which provided a breakthrough in Go and Havannah, were applied to Arimaa by Tomas Kozelek in 2009, Sam Miller in 2009, and Thomas Jakl in 2011. The Bonanza Method that had worked well for Japanese Chess was applied to Arimaa by Kanjanapa Thitipong, Komiya Kaneko, and Yoshiyuki Kotani in 2012. Some researchers even tried very unconventional approaches. Using plans, patterns, and move categories to guide a very highly selective search was tried by Gerhard Trippen in 2009. Modeling Arimaa using linguistic geometry was tried by José Roberto Mercado Vega and Zvi Retchkiman Kösberg in 2009. A complete list of research papers on Arimaa can be found at arimaa.com/arimaa/papers.

Although most research did not lead to strong Arimaa programs, there was some very fundamental research going on at Harvard University, which initially did not seem to have much impact, but would eventually culminate to provide the breakthrough needed to win the Arimaa Challenge. It was presented in David Wu's 2011 thesis *Move Ranking and Evaluation in the Game of Arimaa*. Wu was the first to apply machine learning to order Arimaa moves based on the likelihood of an expert playing the move. This approach was first suggested by Rémi Coulom in 2007 and described in his paper "Computing ELO Ratings of Move Patterns in the Game of Go". Although this approach worked well for pruning and ordering the moves to examine further, it took too long to execute and could only be applied at the root node of the search tree. It was not until 2015 that Wu would find a way to apply this technique deeper in the tree using what he calls "tactical move generators". Wu refers to this as a breakthrough, and it remains to be seen if this could be applied to other games.

10 IMPROVEMENT OF RATING

The rate at which the best programs were improving after 2009 seemed to increase slower than expected, despite the continuing improvements in hardware and the best efforts of the developers to incorporate Arimaa knowledge into their programs. The top human players seemed to be about 200 rating points ahead and felt comfortable that they would be able to maintain the lead. Table 3 shows the ratings of the Arimaa Challengers and defenders over the years. The rating is the Arimaa Gameroom rating after the event to incorporate the performance during the event.

11 SHARP'S ENORMOUS INCREASE OF PLAYING STRENGTH

Past performance is no guarantee of future results. The warning often heard on the stock markets was quite applicable to Arimaa in 2015. The top humans players were caught off guard by the sharp increase in SHARP's performance. The nearly 400 point increase over the previous year's challenger could be characterized as nothing less than a breakthrough. Had the increase come more gradually over a number of years, the top players would have had time to adjust and perhaps keep up. I myself was shocked to see the result, and felt that there had to be some breakthrough that allowed SHARP to play this well within such a short period of time.

Congratulations to David Wu on winning the Arimaa Challenge. Your patience, perseverance and hard work have resulted in a major accomplishment. I did not expect it to happen just yet, and I am sure many in the Arimaa community feel the same way, but I am very happy to see that there may actually be a breakthrough that will come out of David Wu's efforts. If this proves to be the case, then the Arimaa Challenge has been quite successful in achieving its goal. Even if there is no breakthrough, I am happy to see that the Arimaa

Challenge has resulted in so many people looking at the problem posed by Arimaa and contributing to scientific research on autonomous game playing.

Year	**Challenger, rating**	**Defenders, rating, win-loss**
2004	BOMB, 1807	Omar Syed, 2102, 8-0
2005	BOMB, 1822	Frank Heinemann, 1969, 7-1
2006	BOMB, 1563	Karl Juhnke, 2258, 3-0 Greg Magne, 2166, 3-0 Paul Mertens, 2088, 2-1
2007	BOMB, 1880, version 2005	Karl Juhnke, 2361, 3-0 Brendan M, 1898, 2-0 Naveed Siddiqui, 1703, 0-1, substituted for Brendan Omar Syed, 1917, 3-0
2008	BOMB, 1911, version 2005	Jean Daligault, 2494, 3-0 Greg Magne, 2342, 3-0 Mark Mistretta, 2095, 1-0 Omar Syed, 1928, 2-0, substituted for Mark
2009	CLUELESS, 1953	Jean Daligault, 2471, 2-0 Karl Juhnke, 2457, 2-1 Jan Macura, 1979, 2-1 Omar Syed, 1939, 1-0, substituted for Jean
2010	MARWIN, 2053	Patrick Dudek, 2142, 0-3 Greg Magne, 2338, 3-0 Daniel Scott, 2145, 2-1
2011	MARWIN, 2066	Gregory Clark, 2236, 3-0 Toby Hudson, 2205, 3-0 Karl Juhnke, 2320, 2-1
2012	BRIAREUS, 2206	Hirohumi Takahashi, 1626, 3-0 Jean Daligault, 2427, 3-0 Eric Momsen, 1994, 0-3
2013	MARWIN, 2096	Mathew Brown, 2548, 3-0 Matthew Craven, 2018, 2-1 Greg Magne, 2335, 3-0
2014	ZILTOID, 2231	Fritz Juhnke, 2497, 3-0 Max Manual, 2228, 2-1 Samuel Schueler, 2230, 2-1
2015	SHARP, 2595	Mathew Brown, 2612, 1-2 Jean Daligault, 2170, 1-2 Lev Ruchka, 2190, 0-3

Table 3: Rating of Arimaa Challenge Participants.

12 CONTROVERSY

Just as there was controversy over the results of the DEEP BLUE vs Kasparov match shortly after it was over, so it was for the Arimaa Challenge match as well. After the challenge was over, it was brought to my attention that Karl Juhnke, who had supported the Arimaa Challenge by offering $1000 to the challenge winner, made a bet with Mathew Brown, the strongest and youngest of the defenders. Actually, it was structured as Karl buying insurance by paying Mathew $2 to receive over $200 if Mathew lost his series. The situation was made more complicated when David Wu offered to compensate Mathew if he lost the series. Although David's intent was to neutralize the situation by taking pressure off of Mathew, it amounted to the challenger paying a defender to lose the series when taken out of context. I was quite disappointed to learn about this, and if I had known about it before the challenge, I would have taken steps to discourage it. Although nothing could be done about it after the fact, I thought it would be a good opportunity to stir some controversy and have the Arimaa Challenge go out with a bang, by announcing that the challenge results had been invalidated for a few days.

13 IMPACT AND DISSEMINATION

Now that computers have caught up to the level of the best human players, it would be interesting to see if they can continue to stay ahead. My guess is that if Arimaa was able to attract enough talented players, the humans may be able to regain the lead temporarily before computers get so far ahead that unaided humans would have no chance against them. It is also very likely that computers will continue to improve much faster than humans and may never relinquish the lead, even temporarily. Either way I would like to see the Arimaa "Man vs Machine" games continue in the future. One format which I refer to as "Crush the Humans" would have the program that won the annual computer championship be made available online for a month, so that any human can play against it. Perhaps multiple copies could be made available if the necessary hardware is available. The human can select the side to play and the time control within interactive game limits. If the computer does not lose any game during the month, the developer wins the prize. Otherwise, the first human to defeat the computer wins the prize. However, in order to receive the prize, the human player must immediately upload a video showing them playing the complete game to a publicly accessible site and send the link to the event organizer. Such a format could also be used for game like Chess and Go, where the top human players do not want to publicly play the computer. This way, if they win the game they can reveal it, but if they lose it would remain anonymous.

RECOMMENDATION

If such challenges were organized by the ICGA, it would go a long way in promoting game AI research as well as sustaining and growing public interest in classical games. It should not be very difficult for an organizer such as the ICGA to find a university or company to sponsor a small prize for such an annual event along with some funds to rent the necessary hardware.

ACKNOWLEDGEMENTS

I would like to thank all the challenge match defenders who have participated in the challenge match over the years. If it was left only to me, the Arimaa Challenge would have fallen years ago. I would like to thank all the program developers who have participated in the challenge match or published the results of your research. I would like to thank all the tournament directors, coordinators, and volunteers who have devoted so much time and efforts over the years to make the Arimaa events possible. Last but not least, I would like to thank Jaap van den Herik for encouraging me to write this article.

REFERENCES

Coulom, R. (2007). Computing Elo Ratings of Move Patterns in the Game of Go. In H. Jaap van den Herik, Jos W. H. M. Uiterwijk, Mark Winands and Maarten Schadd (editors), *Computer Games Workshop*, pp. 113-124. Amsterdam, the Netherlands.

Cox, C-J. (2006). *Analysis and Implementation of the Game Arimaa.* M.Sc. Thesis. MiCC-IKAT 06-05. Maastricht University, Maastricht, the Netherlands.

Daligault, J. (2012). *Arimaa Strategies and Tactics*. CreateSpace Independent Publishing Platform. ISBN 145288417X.

Fotland, D. (2006). Building a World-Champion Arimaa Program. 4th *International Conference on Computers and Games* (eds. H.J. van den Herik, Y. Björnsson, and N.S. Netanyahu), pp. 175-186. Springer Verlag, Heidelberg.

Juhnke, F. (2009). *Beginning Arimaa: Chess Reborn Beyond Computer Comprehension*. Flying Camel Publications. ISBN 0-9824274-0-9.

Kozelek, T. (2009). *Methods of MCTS and the Game Arimaa*, BA Thesis, Charles University, Prague, Czech Republic.

Mercado Vega, J.R. and Retchkiman Kosberg, Z. (2009). Modeling Arimaa Using Linguistic Geometry. *Proceedings IEEE Symposium on Computational Intelligence and Games*, pp. 379-386.

Miller, S. (2009). *Researching and Implementing a Computer Agent to Play Arimaa*, Thesis, University of Southampton, UK.

Jakl, T. (2011). *Arimaa Challenge – Comparison Study of MCTS vs Alpha-Beta Methods*, BA Thesis, Charles University, Prague, Czech Republic.

Syed, O. and Donkers, J. (2004). The Arimaa Match, *ICGA Journal*, Vol. 27, No. 2, p. 109.

Syed, O. and Donkers, J. (2005). The Arimaa Match, *ICGA Journal*, Vol. 28, No. 2, p. 118.

Syed, O. and Syed, A. (2003). Arimaa – A New Game Designed to be Difficult for Computers. *ICGA Journal*, Vol. 26, No. 2, pp. 138-139.

Thitipong, K., Kaneko, K., and Kotani, Y. (2012). Design and Implementation of Bonanza Method for the Evaluation in the Game of Arimaa. *IPSJ SIG Technical Report*, Vol. 2012-GI-27, No. 4.

Trippen, G. (2009). Plans, Patterns and Move Categories Guiding a Highly Selective Search. *12th Advances in Computer Games Conference* (eds. H.J. van den Herik and P. Spronck), pp. 111-123, Springer Verlag, Heidelberg.

Wu, D. (2011). *Move Ranking and Evaluation in the Game of Arimaa*. B.A. Thesis, Harvard College, Cambridge, Mass.

Arimaa website: www.arimaa.com/

COMPUTER ARIMAA: THE BEGINNING

David Fotland[1]

San Jose CA, USA

ABSTRACT

This contribution contains my personal experience with the Arimaa game. It is based on my communication with Omar Syed and my memories of developing the program that won the first five computer competitions. It also lost the human contest five times in a row. In Syed's (2015) contribution the precise results are given. Finally, I will highlight the techniques used in Bomb, which were up to date ten years ago. In summary, the contribution describes how the first Arimaa program was developed.

1 THE VERY BEGINNING

The first time I heard of Arimaa or Omar Sayed was in January 2003, when Omar emailed me out of the blue. Omar's email (1/18/2003) stated in part:

> "On your homepage it mentions that:
> *Our focus is on the game of Go, because it is by far the most difficult strategy game for computers.*
>
> "That is true, but only because no one had ever attempted to design a game that was difficult for computers to play. Over the last several years, actually since GK lost to DEEP BLUE in 1997, I have been working on designing such a game and I've recently finished it. I estimate that this game is about 100x more difficult for computers than Go and about 1000x more difficult than Chess. My estimate is based on a program using a search tree and heuristic function to select the best move. It would not apply to other approaches. Please check it out:
>
> http://arimaa.com/arimaa/
>
> "I would be interested to know what you think since you definitely are an expert in developing programs for games that are difficult for computers to play."

Indeed, Arimaa looked like an interesting game to play, and Omar had succeeded is making a game with a huge branching factor. Since the pieces move slowly, constructing a strategic position would take far more depth than could be searched. It would be quite a challenge for a computer opponent. It was a challenge I was eager to accept, not only because Omar was offering a $10,000 prize for the first Arimaa program to beat a strong human player.

2 MANY FACES OF GO

The timing of Omar's email was fortuitous. In the late 1990's there were several world computer Go tournaments with big prizes, so most of my game engine development time was focused on THE MANY FACES OF GO. It won a major tournament in 1998 (Fotland, 2013) which led to a deal where I licensed the engine to a company in Japan for sale as AI IGO. Every year I had two major tournaments to prepare for, and I delivered an engine update to my Japanese licensee. At the time I was working at Hewlett Packard in a senior engineering job, and had two young children. There was no time for a big new project.

The annual FOST cup with its 1,000,000 yen first price (Fotland 2013) ended in 1999. The Ing Computer Go Championship had a first prize of NT$200,000, and the winner played strong humans for up to NT$40,000,000 (over a million US dollars) (Fotland 2013). The last Ing tournament was in 2000. The 21st Century Cup was formed to provide a new tournament with money prizes, but its last tournament was in 2002, due to lack of sponsors.

[1] Email: fotland@smart-games.com

In July 2002 I won the last 21st Century Cup, then spent a few months finishing and releasing MANY FACES OF GO version 11, so my customers could upgrade to this strong engine. With no major Go tournaments on the horizon I was available for other projects.

3 WORLD WAR CHESS

Although I am most well-known for developing THE MANY FACES OF GO, I have written engines for many other games. In late 1997 I wrote a chess engine for a game compilation CD. I took this three month contract primarily to gain expertise in alpha-beta search, planning to apply it to a highly pruned full board search in MANY FACES OF GO. In 2000-2001 I took another contract to build a client and server for a chess variant called World War Chess (King 2007). This was chess with armor, infantry, and bombers. The rules are different from chess, but it uses the same board. World war chess has some unusual moves. The Bomber moves like a queen, but can fly over friendly pieces. There are three slow pieces (infantry, and two types of tanks) that can only move one square at a time. But when several of the same type are in a connected line, they can move together as a single move. For WWChess I leveraged and improved my existing chess engine. When Omar contacted me I already had code for a modern chess program ready to go that was flexible enough to implement different rule sets.

My chess engine is written in c++. It uses bit-boards for move generation and evaluation. The search is iterative deepening PVS negamax with null move, search extensions, and quiescence search. It has a transposition table and uses the killer and history heuristics for move ordering.

4 A SWEETER OFFER

In my reply to Omar I told him that I thought his prize could only be won quickly, before people learned how to play the game well. Arimaa's branching factor is so high that I did not think alpha-beta search could beat people once the game strategy was well understood.

> (1/18/2003 from Fotland)
> "Part of the reason I think the prize can be won quickly has nothing to with the complexity of the game. Since it is a new game the human players are not as strong as they are in Chess and Go.
>
> "It is the equivalent of writing a chess program to beat a group of people who have played chess for a few years without any contact with strong chess players or any of the chess literature. Chess and Go both have had hundreds of years of concentrated study by many people to develop strong strategies and tactics. None of that is available for your game.
>
> "Your game is more complex than either of these, so I don't think I could write a strong program as quickly. I wish I had time to give it a try.

He replied with a sweeter offer:

> (1/19/2013 from Omar)
> "You are definitely right that humans have a disadvantage here because the game is new and it takes people a long time to develop expertise in a game. Also you do have the advantage that you've developed game engines for several games more complex than chess. But I've made the challenge offer and it will stand; $20,000 if you are able to defeat the best human at the end of this year."

5 MY COOPERATION WITH DON DAILEY

The day after I got Omar's email, I got an email from Don Dailey saying that he had already started to write a program and asking me if I was going to try. I started coding the Arimaa rules the same day. I do not remember if it was Omar's larger prize or competing with Don that made me change my mind.

> (1/20/2003)
> "Hi David,
>
> "I decided to write an Arimaa program. I have some ideas and already have most of the move generation in place. The actual programming is trivial compared to chess!

"Are you going to try?

Don"

Don offered a lot of encouragement, and shared test results so we could verify that our move generators were generating the same set of legal moves. Omar had built an engine using Zillions. Omar, Don, and I compared the number of unique legal moves from a starting position, and initially we each had a different number. Besides bugs, we cleared up a few ambiguities in the rules.

(1/20/03 from Fotland)
"For example, an enemy horse has both my camel and my elephant next to it. I move my elephant away, then the horse into the elephant's square, then the camel into the horse's square. The horse move can be interpreted either as a pull by the elephant or as a push by the Camel.

"If I see it as a camel push, then the camel must move there in the next move, so I assume that if a move can be either a pull or a push, that it is interpreted as a pull.

"But the same move can NOT be both a push and a pull, so if the elephant move was finishing a push elsewhere, then the horse move must be a push by the camel so the camel must move."

Don and I shared some details on our early algorithms with each other. Don wrote:
(2/5/2003 from Don)
"I'm using PVS search and iterative deepening with negamax too. I'm not doing anything with null move yet but the null move assumption is probably very strong in this game. Also, the last ply can be cut if the current "static" board evaluation is below the alpha window, because it's almost certain that the last guy to move will only improve his situation, making the resulting search effort more or less wasteful.

"I have 3 generators:

1) push/pull to trap squares.
2) push/pull to non-trap squares.
3) non push/pull moves.

"I generate moves in that order.

But I also use the "history heuristic" which gives big node reductions.

My current implementation of hh is like this:

1) History table indexed by [color][f-t] where f-t is the from and to square of the move.

2) After the best move is found at any node, I increment the table entry corresponding to the FIRST step made.

3) I sort the move list according to the history table. I only do this for the first step, not for all 4 steps."

Soon Omar had an interface to allow bots to play on his web site, and soon Don's program was up and running. Omar said:
(2/8/03 from Omar)
"I played Don's bot earlier today. It did a pretty good job of trying to hold back my rabbit at the end. It is definitely playing better than a beginner. It is scary to think that your bots are getting into the 4th ply already. They can find a lot of human errors at that level."

My program joined his on February 20th. Omar played it twice and wrote:
(2/22/2013)
"Hey David,

"Your bot is playing really well. It beat me the first two games I played against it. I lost on time, but I think it would have won anyways.

Omar"

6 HOW TO PLAY WELL: NOBODY KNOWS

Coding an engine for a new game is very exciting and rewarding. The engine is improving very rapidly, and most ideas make noticeable improvements. Bugs are obvious and fixes are easy to verify. Performance improvements are easy to find. No one knew how to play well and the strongest players made tactical mistakes that could be exploited.

Later engine development becomes a disciplined engineering effort. Changes must be carefully measured through self-play or test positions. Performance must be measured, and code carefully tuned to search as many positions as possible. The Arimaa web site helped since it had a rating system. I put up a version with fast time controls called bot_speedy and got lots of good feedback from players on the site.

Arimaa has a huge branching factor, with typically 20,000 distinct legal moves in any position. This limited search to about 3 ply (12 steps), so my plan was to win the prize through superior tactics. Since there were no strong players, I had no game records or publications to guide development of an evaluation function. I focused new development on trap tactics and rabbit goal-races.

3 ply search is not enough to beat strong players, so I needed some way to extend the search. The first change was to deepen the search by step rather than ply. This dramatically reduced the time for move generation and sorting, but required changes to the negamax framework and null move handling. Only one of four steps has a change of color to move. When the color does not change, negamax is called with the same alpha and best bounds, and without negating the score. A null step generated null steps for the rest of the steps in the move, so a null step might skip one to four steps.

I was inspired by the computer chess concept of a Static Exchange Evaluator to calculate the number of steps to capture each piece at a trap (one to six steps, assuming no enemy moves). This code runs once at each trap and looks for one of about 50 local piece configurations. This generates many cutoffs when the current step is not the last step in a move. This works well with null move, since a null move during the 3rd ply will enable a trap threat to be found on the first step of the 4th ply.

Once players learned to avoid blunders around traps, they won easily by sacrificing material to open a path for a rabbit to run across the board. I needed some way to extend the search to find threats to win. I statically evaluated how many steps (1 to 8, or more) it will take each rabbit to reach the goal, assuming no intervening moves by the opponent. This is a tricky 700 lines of code, since there are many cases. This allows the search to find goals four steps earlier, and enables a highly pruned search to find defenses against goal threats. The mate search is implemented as a mode flag during quiescence search. When a rabbit has a short path to the goal, that side is only allowed moves that advance toward the goal (including moves to push or pull enemy pieces out of the way). The other side is only allowed moves by pieces that are close enough to affect the goal.

When this was implemented, weak players could no longer sacrifice pieces to force a goal, and strong players complained that the program defended tenaciously against goal threats. The strong players shifted to new strategies that immobilized pieces, won material, and did not try for the goal until there was a large material advantage. For more details on bot BOMB, my Arimaa program, see Fotland (2006).

7 SEARCH DETAILS

The negamax search function takes the board, move-to-make, alpha, beta, and remaining depth (in steps). The following code highlights the basic structure, but leaves out many details involving the transposition table, time control, killer and history tables, etc. Phases (steps) in a move are numbered 0 through 3.

```
negamax(board b, move m, int alpha, int beta, int depth) {
   b.makeMove(m);
   if (b.mateCheck()) {   // static check for rabbit running
      return mateValue();
   }
   // null move applies to all steps in the current move
   if (m.isNull() && b.getPhase() != 0) {
      if (b.getPhase() == 3)
         return -negamax(b, m, -beta, -alpha, depth-1);
      else
         return negamax(b, m, alpha, beta, depth-1);
   }
   // checks for repetition and time control omitted
   // hash table lookup: check score for cutoff omitted

   // razoring check.
   // Evaluate early if remaining steps are all the same color
   if (b.getPhase() <= 2 && depth >= 1 && depth + b.getPhase() <= 3) {
      if ((val = b.evaluate()) >= beta) {
         return val;   // cut off
      }
   }

   int bestVal = -INFINITY;  // best score found

   // end of search - evaluate and check for mate or mate extensions
   if (depth <= 0) {
      val = b.evaluate();
      if (b.evalFoundMate()) {
         return mateValue();    // set to prefer shorter mates
      }
      // code omitted that starts or stops mate search
      if (val > beta)
         return val;   // cutoff
      if (val > bestVal)
         bestVal = val;
   }

   // null move if depth > 0, no razor check, no forced push
   if (b.nullMoveOK()) {
      if (b.getPhase() == 3)
         val = -negamax(b, PASS, -beta, -beta+1, max(depth-4, 0));
      else
         val = negamax(b, PASS, beta-1, beta, max(depth-4, 0));
      if (val >= beta)
         return val;
   }

   // if depth > 0 (full search) or in mate search
   // try moves in order:
   //    hash table move
   //    killer 1 or 2
   //    history table moves 1, 2, or 3
   //    pulling moves
   //    pushing moves
   //    forward steps (all pieces, in order of bits in bitmap)
   //    left or right steps
   //    backward steps
   // if in mate search, sort moves by distance to the rabbit
```

```
if (depth > 0 or b.matesearch())
   b.genMoves();
// if depth < 0 and not mate search, Quiescence moves only
// complete a push
// capture a piece in a trap
// save a piece threatened with capture in a trap
else
   b.genQuiescenceMoves();

int b1 = beta;   // PVS bound
foreach (move tryit in b.generatedMoves) {
   if (pruneMove()) {
      // prune if this step undoes the previous step
      // unless there is a pull or push involved
      continue;
   }
   if (b.getPhase() == 3) {
      val = -negamax(b, tryit, -b1, -alpha, depth-1);
      if (val >= b1 && val < beta) {
         // search again with full window
         val = -negamax(b, tryit, -beta, -alpha, depth-1);
      }
   } else {
      val = negamax(b, tryit, alpha, b1, depth-1);
   }
   if (val > bestVal) {
      if (val > beta) {
         return val;   // cutoff
      }
      bestVal = val;
      if (val > alpha) {
         alpha = val;
         if (b.getPhase() == 3 && depth > 0) {
            // adjust PVS bound only when player changes
            // no PVS during quiescence search
            b1 = alpha + 1;
         }
      }
   }
   return bestVal;
}
```

8 THE FIRST HUMAN vs COMPUTER COMPETITION

The first human versus computer competition was about a year later, in early 2004. My program, BOMB, won the computer-computer challenge, with one loss to Don Dailey's program, OCCAM. Arimaa proved popular enough that people developed good strategies and I was not able to win a single game against Omar.

I did not think I would be able to keep up with the continuing human progress. For the 2005 championship I did a little work on the rabbit mate evaluation a few months before the competition, and then stopped working on it entirely. Omar asked if BOMB could be entered in future championships and I agreed. At the time BOMB was considerably stronger than other programs so I agreed to publish a description of my approach, with much encouragement from Jaap van den Herik (see Fotland, 2006).

9 IDEAS ON ARIMAA IMPLEMENTED IN GO

In 2006 and 2007 I applied a full board alpha-beta search to THE MANY FACES OF GO, and got a significant improvement in strength, from about 8 kyu to 5 kyu. My experience with Arimaa was very helpful there. Then in 2007 Monte Carlo Tree Search became the algorithm of choice for computer Go. I spent all of 2008 rewriting MANY FACES OF GO using MCTS, then 2009 releasing and supporting a new version. This left me no time to return to Arimaa.

10 CONGRATULATIONS

I felt that some breakthrough in machine learning or new algorithms would be required to win Omar's prize. The search would need to be much more selective to reach the required depths. David Wu's win this year is very impressive, since he had a huge increase in strength within an alpha-beta framework. He deserves sincere congratulations for his achievement.

REFERENCES

Fotland, D. (2006). Building a World-Champion Arimaa Program. 4th International Conference on Computers and Games (eds. H.J. van den Herik, Y. Björnsson, and N.S. Netanyahu). *Lecture Notes in Computer Science*, Vol. 3846, pp. 175–186.

Fotland, D. (2013). World Computer Go Championships. Fetched from http://www.smart-games.com/worldcompgo.html

King, M. (2007). Snapshot of World War Chess web site. Fetched from https://web.archive.org/web/20080509132508/http://www.worldwarchess.com/

DESIGNING A WINNING ARIMAA PROGRAM

David J. Wu[1]

ABSTRACT

In 2015, twelve years after the creation of the Arimaa Challenge, the program SHARP won the Challenge, defeating three strong human opponents with a total of 7 wins and 2 losses. This paper describes the design of the winning agent and the improvements and innovations that led to its recent success. Among these are several innovations greatly increasing the quality and selectivity of the search and several major improvements to the positional evaluation.

1. INTRODUCTION

Arimaa is a game invented in 2002 by computer engineer Omar Syed. According to Syed, the original motivation for Arimaa came from the defeat of the human World Champion in Chess, Garry Kasparov, in 1997 by IBM's DEEP BLUE (Syed and Syed, 2003). Syed set out to design a game playable using a Chess set that would be as fun and interesting for human players and yet be more difficult for computers. The result was a fascinating new board game called Arimaa. By designing Arimaa, Syed hoped to encourage new interest and research in artificial intelligence for strategic games (Syed and Syed, 2003).

To this end, every year an event known as the "Arimaa Challenge" has been held, in which the top computer program using common desktop hardware plays a series of matches against three different players chosen that year to "defend" the Challenge. Since the beginning of the Challenge in 2004 up until the previous year, despite significant progress in computer Arimaa, human players have convincingly defeated the top programs each year.

However, to everyone's surprise in 2015 the tables turned. In this year our own program SHARP swept both the Computer Championship and a preliminary blitz tournament undefeated with 18 wins in a row, outperformed the strongest other competing program in preliminary screening matches against human players with a record of 28-2, and proceeded to defeat the challenge defenders 7-2, winning the Challenge!

Such a strong performance was both exciting and unexpected, and judging from the games, likely some luck was necessary. But in retrospect, the result is not entirely surprising. Since its first win of the Computer Championship in 2011, SHARP has improved greatly, especially in the last two years. In self-play testing against older versions at blitz speeds, it improved at least 200 Elo rating points going into the 2014 competition. It then leapt far ahead of all other programs by gaining an additional 400 Elo rating points by the start of the 2015 competition[2]. If not already stronger now, SHARP is at least close to being on par with top players, and it seems that there is still plenty of room for further improvement.

The goal of this paper is to explain the design of SHARP and to present the most significant innovations and improvements responsible for its jump in strength in the last two years. In Section 2 we describe the game Arimaa and some of the properties that have made it computer-resistant. In Section 3 we give an overview of other work and research that has been done in Arimaa. In Section 4 we present the basic algorithms used in SHARP and in Section 5 the recent search improvements that made its success possible. In Section 6 we describe the development and design of the positional evaluation. Finally, in Section 7 we present some conclusions and possibilities for future work.

[1]email:lightvector@gmail.com

[2]In winning chances, 200 Elo implies about 3:1 odds, and 400 Elo implies 10:1 odds, quite a large improvement.

2. THE GAME

Below we describe the rules of Arimaa (2.1) and we briefly discuss some of the properties of the game that have made it computer resistant (2.2).

2.1 Rules

Arimaa is a deterministic two-player abstract strategy game played on an 8 x 8 board, the two players being *Gold* and *Silver*. The rows and columns on the board are labeled 1. . .8 and a. . .h, respectively, as shown in Figure 1. Below we describe five parts of the rules of Arimaa, including the move notation.

2.1.1 Setup

Prior to normal play, Arimaa begins with a *setup phase* where both players place their pieces in any desired arrangement within their starting two rows with Gold placing all pieces first. In order of decreasing strength, each player has 1 Elephant, 1 Camel, 2 Horses, 2 Dogs, 2 Cats, and 8 Rabbits.

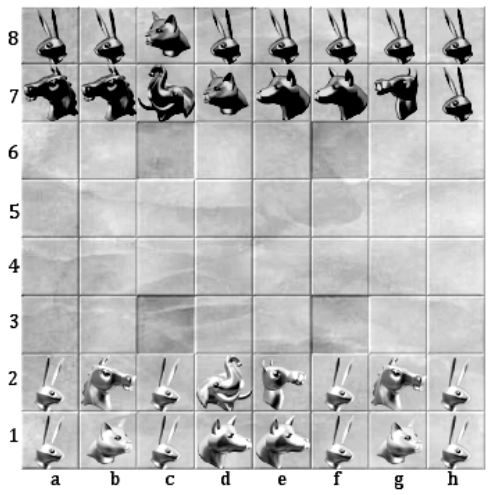


Figure 1: Example game position, just after the setup phase.

2.1.2 Movement

Following the setup phase, play begins and players take turns making moves of up to four *steps*, with Gold moving first. A *step* consists of selecting a piece of one's color and moving it to an empty adjacent square, that is, left, right, forward, or backward. All pieces move identically this way with one exception: rabbits may not move backward.

Pieces are differentiated in strength in that pieces can *push* or *pull* weaker opposing pieces using two steps at a time, as shown in Figure 2. Pushes can displace weaker opposing pieces into any adjacent empty square, and similarly pulls can displace weaker pieces from any adjacent square into the one just vacated. Simultaneously pushing and pulling with the same piece is illegal.

Additionally, stronger pieces can prevent weaker pieces from moving by *freezing* them. Whenever a piece is adjacent to a stronger opposing piece and is not *defended*, that is, when it has no friendly piece adjacent, then it is *frozen* and cannot move. (It can still be pushed or pulled by the opponent.)

Players must change the board position on their turn and also may not make any move that would result in a *third-time repetition* of the position and player-to-move.

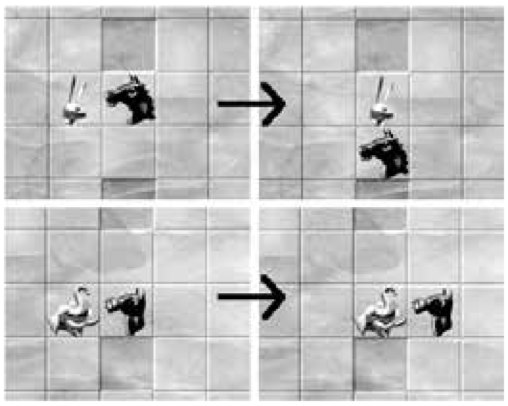


Figure 2: A silver horse steps down, pulling a gold rabbit. A gold elephant pushes a silver camel to the right.

2.1.3 Capture

The squares c3, c6, f3, and f6, are *trap* squares. Whenever a piece is on a trap square but is not defended, it is *captured* and immediately removed from the board.

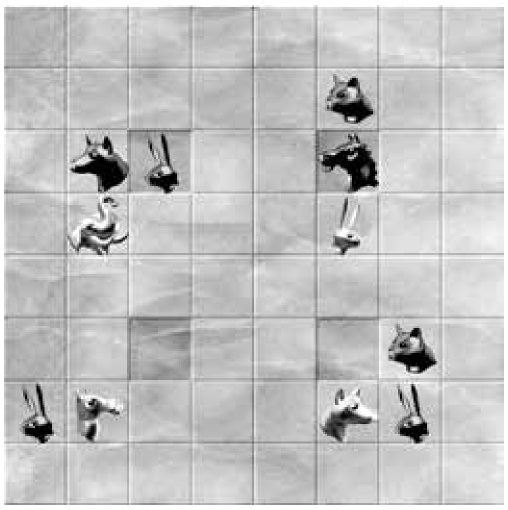


Figure 3: Examples of capturing and freezing. *Upper-left*: If the gold elephant pushes the silver dog, it captures the silver rabbit on the trap at c6. *Upper-right*: The silver horse at f6 can capture the gold rabbit by stepping left or right while pulling the gold rabbit onto the trap. *Lower-left*: The rabbit on a2 is frozen by the adjacent gold camel. *Lower-right*: The g2 rabbit is *not* frozen by the gold dog because it is guarded by the silver cat.

2.1.4 Game Objective

A player scores a *goal* and wins the game when one of that player's rabbits ends the turn on the opponent's back row, on the opposite side of the board[3]. A threat to win the game on the next move in this way is called a *goal threat*.

2.1.5 Notation

Arimaa move notation is used in a small number of places in this paper. For reference, it consists of tokens such as Ee2n indicating the piece (E $\in$ {E,M,H,D,C,R,e,m,h,d,c,r} uppercase for Gold and lowercase for Silver), the

[3]Much more rarely, a player can win by *immobilization* if the opponent is unable to make any legal move on their turn, or by *elimination* if the all of the opponent's rabbits are captured.

originating square (e2), and the direction of movement (n $\in$ {n,s,e,w,x} with x indicating a capture). An example move in the middle of a game might look like: Ef4n Ef5w cf6s Hg4n.

2.2 Computer-Resistance

Why is Arimaa computer-resistant? We can identify two major obstacles.

The first is that in Arimaa, the per-turn branching factor is extremely large due to the combinatorial possibilities produced by having four steps per turn. Even after identifying equivalent permutations of steps as the same move, on average there are about 17000 legal moves per turn (Haskin, 2006). This is a serious impediment to search.

Obviously, a high branching factor alone doesn't imply computer-resistance, particularly if the standard of comparison is with human play: high branching factors affect humans as well. However, Arimaa has a property common to many computer-resistant games: that "per amount of branching" the board changes slowly. Indeed, pieces move only one orthogonal step at a time. This makes it possible to effectively plan ahead, cache evaluations of local positions, and visualize patterns of good moves, all things that usually favor human players.

The second obstacle is that Arimaa is frequently quite positional or strategic, as opposed to tactical. Capturing or trading pieces is somewhat more difficult in Arimaa than in, for example, Chess. Moreover, since the elephant cannot be pushed or pulled and can defend any trap, deadlocks between defending elephants are common, giving rise to positions sparse in easy tactical landmarks. Progress in such positions requires good long-term judgement and strategic understanding to guide the gradual maneuvering of pieces, posing a challenge for positional evaluation.

3. HISTORY AND EARLIER WORK

Despite these difficulties with branching factor and evaluation, the strongest Arimaa programs have generally resembled strong Chess programs in design, using alpha-beta search in combination with a slew of other enhancements and carefully-tuned evaluation functions (Zhong, 2005; Fotland, 2006; Cox, 2006). Developers Fotland and Zhong were among the first to do well with this approach and detailed many basic ideas used by nearly all strong Arimaa programs today, such as static capture and goal detection. Fotland in particular pioneered so much of the early groundwork that despite abandoning work on Arimaa after 2005, his program BOMB continued to win every Arimaa Computer Championship up through 2008[4](Syed and Syed, 2015b). Since then, the Computer Championship has been lively, with the title trading back and forth between a variety of ever-improving bots every year from 2009 onwards. In every case, the winning program was a "traditional" Chess-like searcher.

A few others have tried alternative approaches without significant success. These include Kozelek (2009) and Miller (2009), who made attempts to apply Monte-Carlo tree search in various ways. In another alternative but unsuccessful approach, Trippen (2009) investigated a pattern-matching and plan-based method for performing extremely selective search, examining only a tiny number of moves (as few as five!) in each position.

More recent and successful work has focused on ways of learning better move ordering or evaluation for use within the traditional alpha-beta search paradigm, including our own work on learning move ordering from expert game records (Wu, 2011) and similar other work (Vivek Choksi, 2013). In an exciting result, Hrebejk (2013) demonstrated that it is also possible to learn a strong evaluation function from expert game records . The following year, Hrebejk's program DRAKE, while not quite a contender for the top, performed fantastically well given its status as a newcomer.

The successes of these later approaches and the steady improvement in computer strength after 2009 made it a good bet that the traditional alpha-beta search paradigm would eventually be sufficient. And indeed, this is the path that SHARP followed. We begin next in Section 4 with an overview of the basic algorithms and enhancements taken from this paradigm, and then in Sections 5 and 6 elaborate on how we navigated the two obstacles of branching factor and strategic evaluation in Arimaa - the former via new enhancements to the search greatly increasing its efficiency and selectivity, the latter via determined engineering and a robust development methodology for improving the evaluation function.

[4]Even today, surpassing BOMB is viewed as a noteworthy achievement for any new developer!

4. BASIC SEARCH ALGORITHMS AND ENHANCEMENTS

SHARP follows the same fundamental design as strong Chess programs, using an *iterative-deepening depth-limited alpha-beta search* to explore the game tree and compute its *minimax value* out to increasing depths (see 4.1). The search is augmented with a variety of heuristics and enhancements for *ordering*, *extending*, *reducing*, and *pruning* to increase its efficiency and focus it in important parts of the tree (see 4.2 and 4.3). At leaf nodes, a carefully designed *evaluation function* is called to evaluate the game state, taking into account a large variety of positional features (described later in Section 6).

4.1 Alpha-Beta Search

In alpha-beta search, one computes the *minimax value*[5]of a position and finds a move that achieves that value by recursively finding the minimax value of each move in a depth-first manner and taking the maximum of the values returned for each move, negating whenever the side-to-move changes[6]. The algorithm's name derives from an optimization: it also accepts an interval (α, β) and returns from a node immediately if it can prove that its minimax value is outside the interval. The interval is passed down recursively, and after each move is searched (letting x be the value returned), two rules are followed to apply and update the interval:

1. Beta cutoff: If $x >= \beta$, return immediately, since the maximum move value must be $>= \beta$ and outside the window (α, β).
2. Improving alpha: Else set $\alpha := max(\alpha, x)$ when recursing on the remaining moves. A move value of x means that values $<= x$ can no longer affect the maximum move value at this node.

The efficiency of alpha-beta search improves greatly if it searches better moves first, obtaining beta cutoffs and stronger alpha-beta bounds earlier. As a result, alpha-beta is almost always used with a heuristic *move ordering* function to sort moves in order of likely quality. Since searching the entire game tree is infeasible, one stops at a limited depth and calls a heuristic *evaluation function* to estimate the value of the position based on features of the board state. In actual timed play, one usually also uses *iterative deepening* - iteratively re-searching the same position with increasing depth limits - to run until the desired amount of time is consumed.

It is worth noting that a variety of subtle details in the implementation of these algorithms can make a significant difference in performance. Like other strong programs, SHARP carefully manages these details. To give two examples:

- Upon a timeout during iterative deepening, the results of the interrupted search are used. In particular, if that search finds any move superior to the first move (which by ordering is always the best move of the previous iteration), that move is chosen. This gains a significant fraction of a ply in average effective search depth.
- When sorting moves for move-ordering, often a $O(n^2)$ *selection sort* is better than a more asymptotically-efficient sort. Selection sort can be done incrementally, eliminating the need to sort more than a few elements if a beta cutoff occurs early.

One other detail is that SHARP defines the depth of a search in steps rather than in moves. That is, it uses a formulation of the game where each *ply* of the search is a single step and the side-to-move only changes every four ply. Because of Arimaa's high branching factor, this improves performance by avoiding large lists of moves. However, as part of one of the recent innovations contributing to SHARP's success, the search also does not strictly adhere to recursing one step at a time. These details are elaborated on below in Section 5.4.

[5]The *minimax value* is the best result that the player-to-move can force from that position with optimal play assuming the opponent responds optimally.

[6]This is the "negamax" formulation of minimax search, which is the formulation virtually always used in practice.

4.2 Standard Enhancements

SHARP uses a variety of search enhancements that are standard in alpha-beta searchers for other games, notably Chess. We list and describe seven specific enhancements below.

4.2.1 Transposition Table

In Arimaa, different sequences of steps or moves that lead to the same final game state, or *transpositions*, are common. Therefore, a large hashtable, or *transposition table* is used to cache results to avoid repeating work[7]. As is usual, the contents of the table are preserved between top-level iterations of the search, and the best move is also stored so that in subsequent iterations that move can be searched first to improve move ordering. In SHARP, the table also caches the values of direct calls to the evaluation function[8].

In SHARP, transposition table entries are 128 bits, composed of 64 bits recording the full hash key, 32 bits for the best move, 21 bits for the evaluation, and 11 bits for the search depth and for flag values indicating the kind of result stored. In the event of a collision, a new entry always overwrites the existing entry. This is the simplest possible *replacement scheme*. A more sophisticated replacement scheme, such as a bucketed scheme based on entry age or depth, might be an avenue for future improvement.

4.2.2 Killer Moves

In wide variety of games, a heuristic that improves move ordering amazingly well is to guess that a move that performs well in one position will, if legal, be good in other positions. Accordingly, the *killer move* heuristic records a move whenever it causes a beta cutoff and ranks that move early in the ordering at any subsequent node at the same ply-level of the tree (Marsland, 1986). In Arimaa, this heuristic alone improves the speed of search by a factor of tens or hundreds or more depending on the depth of the search.

SHARP tracks 8 killer moves at each ply (2 per ply within the quiescence search), evicting older killers to make room for newer ones according to a FIFO (first-in, first-out) scheme. Unlike standard practice in Chess bots, killer moves are allowed to be captures.

4.2.3 History Heuristic

Another technique for move ordering based on a similar idea is known as the *history heuristic*. At the start of the search, a table with a counter for every possible move is initialized. After searching each node, the best move at that node has its counter incremented, and the values of these counters are used to order future moves (Schaeffer, 1989). Whereas the killer move heuristic captures tactics common to multiple local branches of the search, the history heuristic captures broader trends across the entire search about what moves tend to be better or worse.

Since in Arimaa a table of all possible legal moves would be too large and sparse, SHARP uses a table indexed only by the first step of the move (56 * 4 = 224 possibilities), or the first two steps for pushes and pulls (approx. 56*4*4 = 896 possibilities). A separate table is maintained for every ply in the search.

4.2.4 Quiescence Search

A depth-limited tree search will often return bad results because near the leaves, it will often misevaluate useless and losing threats as good because the search will end without the opponent having turns to refute these threats. This and similar problems are referred to as the *horizon effect*.

A well-known technique to fix this is to perform a *quiescence search* at leaf nodes, where the search is extended so long as the position appears to be tactically unstable (e.g. has a hanging piece), and to limit the cost of doing

[7]Note that in alpha-beta, frequently the result of a search is not an exact value, but rather an upper or lower bound. This necessitates care in determining whether a cached result is sufficient to prune the search of a repeated node when alpha-beta bounds have changed.

[8]This not always done in Chess programs. In SHARP, the evaluation function is by far the most costly part of the program, making it a clear win to cache it.

so, by examining only moves likely to resolve the instability (e.g. capturing moves) (Marsland, 1986).

In Chess, simply searching heuristically-promising capturing moves is already quite good, making quiescence search easy to implement. Most bad tactics in Chess that falsely appear favorable with shallow depth, such as capturing a defended piece, are refuted by a capture, such as recapturing with the defender.

By contrast, in Arimaa, many bad tactics are not simply refuted with captures. In practice, resolving tactical positions frequently involves determining whether a piece can be defended in four steps or whether an effective counterthreat can be made. However, the branching factor makes it difficult to include enough such moves without exploding the size of the search tree and degrading performance.

Because of this, SHARP, atypically compared to Chess programs, uses a quiescence search that is itself severely depth-limited. The quiescence search is never longer than 3 moves (12 steps). To further limit branching, only the first layer generates the full variety of moves needed to resolve common tactics, with deeper layers being more restricted. While not ideal, these choices strike a balance between performance and tactical strength.

The following describes the classes of moves generated in each layer of the quiescence search:

- Layer 0 (if initially 1-3 steps left): Goal defense, capturing moves, rare situational moves[9], goal threats, capture defense, capture threats, pushes and pulls of trap defenders, other “tactical” moves.
- Layer 0 (if initially 4 steps left): Goal defense, capturing moves, rare situational moves, goal threats, capture defense.
- Layer 1 (next 4 steps): Goal defense, capturing moves, rare situational moves.
- Layer 2 (next 4 steps): Goal defense, capturing moves, rare situational moves. Only invoked if layer 1 consumed all 4 steps.

4.2.5 Extensions

Aside from quiescence, it can be beneficial to extend the search in other situations. SHARP’s extensions are simple. It extends the search depth by 1 step whenever a goal threat is played for which the minimum defense by the opponent requires at least 2 steps, and by 2 steps if the minimum defense requires at least 3 steps.

4.2.6 Late Move Reduction

Just as it is useful to extend the search in some cases, it is often useful to *reduce* the search depth of a move. With an effective move ordering, moves late in the ordering are unlikely to be good and therefore usually a waste of time to search. *Late move reduction* decreases the time spent on these moves by reducing the depth remaining by more than one ply when searching them, although if the move actually does appear to be good, the move is re-searched with the normal depth to verify the result (Romstad, 2007). One can view reduction as a “soft pruning” that doesn’t entirely eliminate the cost of searching a move but mostly does so and avoids errors that would result from pruning it entirely. This technique is a key component in many strong Chess programs today.

Unfortunately, effective late move reduction is difficult in Arimaa because sufficiently effective move ordering is difficult. In the past, we have had at most limited success using this technique to improve the search. We also know of no other developer or published result so far that has mentioned any success with late move reductions in Arimaa[10].

However, in one of the biggest improvements for 2015, SHARP now *does* perform late move reduction with great effectiveness. This is due to the discovery of new and effective ways to identify good moves and improve the move ordering in Arimaa, which we will discuss further in Section 5.

[9]A few types of moves are generated only in special kinds of board positions according to various heuristics to fix situation-specific weaknesses, such in “elephant blockade” situations.

[10]Although, some authors have presented some methods for pruning, such as Zhong, who described a method involving the dependency structure of moves (Zhong, 2005) that is now used by many other Arimaa bots.

4.2.7 Multithreading

As with many other programs, SHARP takes advantage of multiple cores on a machine with a multithreaded implementation of alpha-beta. The algorithm used is conceptually similar to the "Dynamic Tree Splitting" algorithm developed for Chess and described by Robert Hyatt (1994). However it differs in that its communication methods are far simpler - rather than signaling other threads to request splitting of work, threads simply operate on a shared representation of the game tree, with operations synchronized with locks at each node.

As is typical with efficient multithreaded alpha-beta search implementations, the details of this implementation are fairly complex, and we refrain from describing them here.

4.3 Standard Arimaa-Specific Enhancements

SHARP also implements several Arimaa-specific search enhancements used by most other strong Arimaa programs. We list and describe three of them below.

4.3.1 Static Goal Detection

With four steps per move in Arimaa, it's nontrivial to check whether the player-to-move can win by scoring a goal on the current turn. Naively, doing so would appear to require a four-step search. However, as first described by David Fotland and elaborated on by Haizhi, it's possible to check this more efficiently by enumerating the classes of patterns that would allow a goal in four steps and writing specialized code (such as a decision tree) to directly determine whether one of these patterns is present on the board (Zhong, 2005; Fotland, 2006). While there are many patterns, there are few enough that enumerating them is barely feasible, and testing them directly provides a huge speedup over a search[11].

SHARP contains several thousand lines of code dedicated to testing whether goal is possible in up to four steps. The resulting code is fast enough that it can be called even on leaf nodes with no noticeable cost to performance, extending the effective search depth by a full four steps for goal threats in the deep endgame.

4.3.2 Static Capture Generation

For similar reasons, all strong Arimaa programs contain code for statically detecting whether a piece is capturable and for generating capturing moves. In SHARP, static capture generation is used in quiescence search and to improve move ordering.

4.3.3 Goal Relevance Pruning

SHARP also uses a technique first mentioned by Fotland and used in his early championship-winning program Bomb - when a goal threat is present on the board, moves that are too far away to defend it or to interact in dependencies with defending moves can be pruned (Zhong, 2005). As Fotland did not describe his method in detail, likely our implementation is different, and we describe it here:

Define a set of steps S as *relevant* to a goal threat if for every game state reachable by a move that stops the goal threat, at least one move reaching it begins with a step in S. Then, one can restrict move generation only to steps in S without any loss of non-losing moves.

The task is then to choose S as small as possible. The idea is that if the opponent threatens goal, S can be reduced to only a local set of steps that do prevent it or that due to local dependencies must be played before steps that would prevent it. Farther-away steps can be pruned because even if desirable, they can always be played later after the goal threat is dealt with.

[11]If it's not obvious why, consider the 1-step case. Whereas a search might involve generating and trying potentially dozens of steps, a direct test consists only of checking if there is an unfrozen rabbit on the 7th rank with an empty square in front, doable in just a few bitboard operations.

In general, given a goal threat, if the player to move has n steps left, all steps involving only squares more than $3(n-1)$ away from any squares involved in the goal threat (including squares necessary for unfreezing or trap defense for the goal threat move to be legal) can be excluded from S. Due to interactions involving the trap squares, this bound is tight in the general case, as shown in Figure 4.

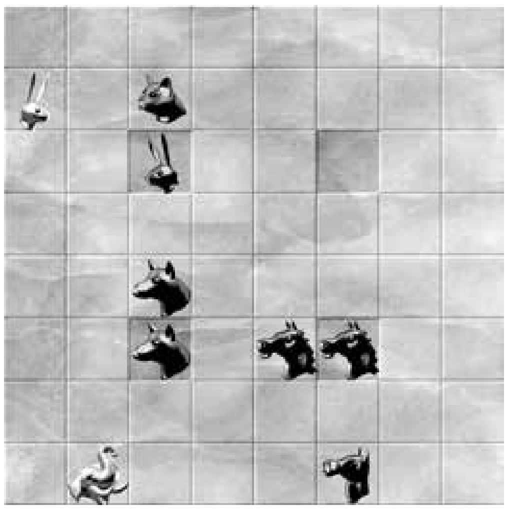


Figure 4: With 4 steps remaining, the step Mf1n, involving a square $3(4-1) = 9$ squares away from b7 (a square that if occupied could freeze the gold rabbit), cannot be pruned as irrelevant because it is part of the unique move Mf1n He3w Dc4n Cc7w that both prevents Gold's goal threat Ra7n and avoids sacrificing any pieces.

However, one can almost always restrict S much further. The "radius of relevance" can only expand by three squares per step as in Figure 4 around a trap with a singly-defended piece on that trap. Otherwise, one can use a radial bound of $2(n-1)$, or if freezing/unfreezing of pieces is irrelevant, only n. Due to a player's own blocking pieces occasionally interfering with ways to prevent a goal, often steps even closer than $n-1$ squares can be pruned, as shown in Figure 5.

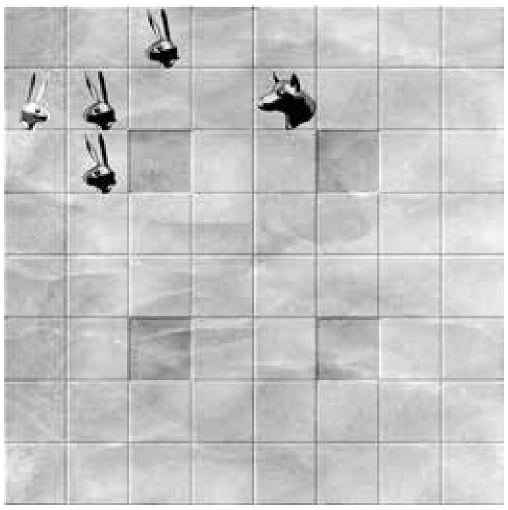


Figure 5: With 4 steps remaining, the step de7w, involving a square only 2 squares away from b7, can still be pruned as irrelevant. Silver's own rabbits prevent the dog from blocking or freezing the gold rabbit in 4 steps.

SHARP contains a couple hundred lines of bitmap-based code that takes advantage these details to compute a small S, so that when a goal threat is present, most steps can be pruned as irrelevant[12]. In the cases where a goal is actually unpreventable, this greatly speeds up the proof, and otherwise still serves to reduce transposition table load and search overhead in the endgame.

[12]If the goal threat is actually unstoppable, $S = \{\}$ by definition is a possible relevant set. And in fact, sometimes SHARP's local bitmap analysis is tight enough to generate the empty set for S and directly prove that the goal is unstoppable.

5. INNOVATIONS AND IMPROVEMENTS IN SEARCH

In just the last two years, SHARP has gained more than 600 Elo points in blitz games against its older versions[13]. Some of the gains were due to a plethora of minor changes and refinements, and some were due to large performance optimizations. However, a significant part was from a few key innovations directly aimed at solving the obstacle of Arimaa's large branching factor, one of the two major obstacles to strong computer play in Arimaa. The common thread between these innovations was improving the search's ability to order and filter good moves out from the thousands of other random and useless moves possible in any typical Arimaa position. We describe four innovations in 5.1 to 5.4.

5.1 Bradley-Terry Root Move Ordering

SHARP continues to use and benefit greatly from a move ordering function that we initially developed in 2011, detailed in Wu (2011). While not developed in the last two years unlike the other improvements discussed below, we present it again here due to the amount it continues to contribute to the playing strength.

This move ordering function is the result of training a slightly generalized *Bradley-Terry model* over thousands of expert Arimaa games to learn to predict expert players' moves. A Bradley-Terry model models the winner of a competition between two agents i and j probabilistically as:

$$P(\text{i wins}) = \frac{\gamma_i}{\gamma_i + \gamma_j}$$

where γ_i and γ_j are model parameters representing the "strengths" of i and j.

We can generalize this both to teams of agents and to competitions between more than two agents or teams by defining the strength of a team of agents $T \subset [1, ..., n]$ to be:

$$\gamma(T) = \prod_{i \in T} \gamma_i$$

And given competing teams $T_1, ..., T_n$, modeling the probability that T_j wins to be:

$$P[T_j \text{ wins}] = \frac{\gamma(T_j)}{\sum_{i=1}^{n} \gamma(T_i)}$$

We apply this model to predict expert moves by considering each position in an expert game to be a "competition" between the available moves, where the winner is the move actually played. Each move is a "team" of the various features associated with that move, where the possible features range from things like "moved a silver horse to d4" to "captured an opponent's horse" to "improves a heuristic trap-defensiveness score for c3 by 5 points".

Given a large number of training games, we then perform a maximum likelihood estimation to choose the model parameters $\gamma_i, i \in [1, ..., n]$ where γ_i is the "strength" of the ith possible feature, that maximize the likelihood of the data given the model[14]. The resulting model assigns to each move in a position a probability that the move would be chosen by an expert player. This can then be used for move ordering by ranking all moves in order of most likely to least likely.

Using a set of several thousand binary features, including features indicating source and destination and piece type, pushes and pulls, captures and goals, capture and goal threats and defense, trap defense, step dependency structure, and a few other types of features (see Wu (2011) for a more detailed list), and training on several thousand rated games by players rated 2100 or higher on arimaa.com, we obtain extremely good move prediction. The resulting move predictor captures the expert moves within the first 1% of the move ordering more than 80% of the time, and within the first 20% of the ordering more than 99% of the time!

At the root node, SHARP uses this move ordering function with great effectiveness. The search almost always finds a good move rapidly, particularly when the former best move is proven unsound on an iteration. Furthermore, with so few expert moves occurring past the first 10% or 20% of the ordering, it can reduce aggressively

[13]Implying winning chances of around 97%!

[14]One can also take logs, defining $\delta_i = \log \gamma_i$. This reveals that we are effectively performing a type of multivariable logistic regression with a linear model.

N	1	2	5	10	100	1000	1%	5%	10%	20%	40%	60%	80%
Expert%	15.0	22.5	34.2	44.4	78.3	96.8	81.0	94.4	97.5	99.2	99.8	100	100

Table 1: Out-of-sample percentage of expert moves falling within the top N moves of the root move ordering as of 2015, for different values of N. Test set is 21K game positions from players rated 2100+ on arimaa.com. Average branching factor was 16567.

with little risk of reducing good moves. All moves past the approximately first 10% of moves are reduced by 1 step and moves past the first 20% are reduced by 2 steps, gaining a large speedup.

Unfortunately, the resulting move ordering function is far too slow to be applied except at the root node. With the current implementation, ranking and ordering all legal moves in a typical position takes about 0.1 seconds, which is far too slow to apply deeper in the search tree.

Despite that, the success of this technique served as a proof of concept back in 2011 that it was possible to filter the set of legal moves down to only a tiny subset with hardly any loss of good moves based on only local features and properties of those moves. The knowledge that this was possible led to experimentation with many other ideas to see if something similar could be done with enough speed to be used within the search tree instead of only the root. Ultimately, this was what gave rise to the biggest improvements in 2014 and 2015.

5.2 Move Generation and Ordering Within-Tree

Whereas the method above solved the problem of move ordering and pruning at the root as early as 2011, it took until 2015 for SHARP to acquire similarly-effective methods suitable for use within the search tree.

The following list describes the current move generation and ordering structure for the main (non-quiescence, non-root) tree search. At each node, SHARP generates and searches the following moves in order from top to bottom[16]:

1. Transposition table move (if available)
2. Killer moves 1-8 (if legal)
3. Capture moves (ordered by size of piece captured)
4. Tactical moves (ordered by history heuristic + movetype[17])
5. Transposition table move prefix (first step, push, or pull only) (depth reduced by 1 step)
6. Other pushes, pulls, and single steps (ordered by history heuristic + movetype) (depth reduced by 1 step)

The key breakthrough was the development of a set of "tactical move generators", item 4 in the above list, which serve largely the same purpose as the Bradley-Terry ordering model for the root. They produce a set of "tactical" moves that, despite only amounting to a few percent of the legal moves in a position on average, contain a large fraction of the likely valuable moves in a position, enabling reductions of the rest.

Of particular note is the degree to which moves can be reduced. Each step, push, or pull in items 5 or 6 is reduced by only one step, but this reduction occurs at each level of recursion. As a result, moves composed entirely of steps that fail to be generated by earlier items (most moves), will be reduced by as many as four steps over the up-to-four levels of recursion for that turn!

A last thing to note is that the generated move list, rather than only containing single steps or only full moves, contains a mix of partial moves of all different lengths, mixing single steps together with two-step pushes and

[15] Having the best move almost always show up in the first few percent of searched moves also synergizes well with the minor detail in Section 4.1 about using the results of an interrupted search.

[16] In the presence of a goal threat, item 4 in this list is skipped, the moves from item 6 are generated using goal relevance pruning, and no reduction in depth is performed.

[17] A minor Arimaa-specific heuristic taking into account the direction and type of a move is used when the history heuristic is near zero and fails to distinguish moves.

pulls and potentially even longer moves, such as capture and tactical moves. This design choice itself is actually another major recent improvement, and also one of the key factors enabling the effectiveness of the late move reduction and boosting the gains from move ordering.

We describe both the tactical move generation and this design choice further in the next sections.

5.3 Tactical Move Generation

The within-tree counterpart to the Bradley-Terry root move ordering model, tactical move generation was perhaps the most significant single innovation in SHARP's search in 2015. It consists of a large set of move generators that generate a small set of "tactical" moves most critical and likely to be good in a position. These include the following classes of moves.

- Up to 4-step pushes and pulls of opponent's pieces on a square adjacent to a trap.
- Pushes and pulls of opponent's pieces that are heuristically likely to threaten them.
- Moves that add defenders to "poorly-defended" traps.
- Moves that unfreeze and/or run-away own pieces that are threatened with capture.
- Moves that freeze "important" opposing pieces.
- Steps around threatened pieces that are heuristically likely to block runaway attempts.
- Up to 4-step moves of the elephant that end the elephant on "useful" squares.
- Moves that match a variety of patterns for severe goal threats and/or are likely forced wins-in-2.
- Many more classes of moves involving further kinds of tactical conditions and patterns.

The design of these generators was directly motivated by the performance problems of the root move ordering. The root move ordering suffers in performance because it is "retroactive". It requires generating legal moves, playing them out, and sorting them after-the-fact. This is precisely analogous to the way that checking for 4-step goal or capture by searching and testing possible moves is far slower than a direct pattern-based check. It stood to reason that the performance of feature-based move ordering could be improved the same way - by directly generating candidate good moves in a forward-looking manner.

Just as with the Bradley-Terry model, we used data from expert games heavily. However, it became obvious early on that including all the kinds of good moves that strong Arimaa players play would be counterproductive. Often good moves in Arimaa are "quiet" moves, where there is no urgent fight and the precise choice of move does not make a major tactical difference. In these cases, the set of moves generated would need to be quite large to have a significant change of including the move actually played by an expert. And moreover, quiet moves are not ones that would benefit much from special search, precisely because of the absence of major tactics.

Therefore, rather than the obvious idea of maximizing the proportion of expert moves generated, we instead minimized the amount by which SHARP's evaluation of the best move produced by any generator was worse than SHARP's evaluation of the expert move[18]. In a bootstrapping fashion, this leveraged the program itself to judge when an expert move was tactically critical instead of merely one of many reasonable moves.

The development and testing cycle followed consisted of five steps.

- By hand, look at positions with large evaluation differences and identify the most common kind of critical move or tactic not being generated.
- Write a move generator that generates that class of moves.
- Observe that the generator is too inclusive and increases the branching factor too much, or that it is too restrictive and fails to produce enough instances of that class of moves.

[18]Treating the difference as 0 if SHARP believed a move generator's move was better than the expert move, and also capping it at 6000 millirabbits if it was worse by more than that. Evaluation consisted of a quick 5-step-deep search.

- Iterate on and refine the move generator and the heuristics used until it generates enough of the target class of moves while increasing the total number of legal moves generated by all generators by an acceptably small amount.
- Add the new move generator and repeat.

The results were a huge success. On average, the resulting set of move generators is capable of capturing about 97% percent of the evaluation difference between expert moves and passing, giving up an average of merely about 6 centirabbits per move[19], while only generating about 3% percent of all of the legal moves when run recursively over the course of the four steps of the turn to produce a complete move.

In the search, the presence of these move generators provides two major benefits. Just as with the root ordering, because of the effectiveness with which they identify critical moves, they allow aggressive reductions of all other moves. Secondly, some of them are used in the top layer of quiescence search, providing 1-4 steps of extra tactical depth at minimal cost. Adding these generators immediately gained 80 Elo rating points in self-play testing, the largest gain any single change to the search has produced in the last several years.

5.4 Game Formulation and Search Structure

The other major innovation in SHARP's search is a design choice that relates to the fundamental structure of the search. With four steps per move in Arimaa, there are two potential formulations of the game for the purposes of search:

- Movewise formulation: Each ply or recursion level in the search consists of a normal up-to-four-step legal move. The side-to-move changes every ply. Branching factor $\approx$ 10000.
- Stepwise formulation: Each ply or recursion level in the search consists of a single step. The side-to-move changes only every 4 ply. Branching factor $\approx$ 30.

Multiple authors have discussed the tradeoffs between these two formulations in earlier work, often favoring the stepwise formulation due to the overhead of generating and sorting large arrays of moves in the movewise formulation (Zhong, 2005). Among other things, generating tens of thousands of moves only to return due to a beta cutoff after the examining the first few is a huge waste of time, necessitating some form of batched incremental move generation, which itself poses other difficulties. Up until the last couple of years, for these reasons SHARP also used the stepwise formulation of search.

However, the stepwise formulation also has disadvantages. Since alpha-beta search is depth-first, it reduces the effectiveness of move ordering by forcing the search to consider moves that begin with the same steps together. If for example the four-step move Ee6s Ee5s Ee4s Ee3w is heuristically likely to be a good move but no other moves beginning with Ee6s are likely to be good, then in the event that the search first explores Ee6s, Ee5s, Ee4s, Ee3w and the move turns out not to be good, it will be forced to then uselessly explore every other move in the subtree beginning with Ee6s before being allowed to try better alternatives.

SHARP solved this problem in 2014 with the innovation of using a hybrid formulation. At any given node, the move list contains a mix of moves ranging from one to four steps[20]. This allows promising moves with three or four steps to be tried without committing to examining a large number of other moves with the same prefix. The transposition table move and killer moves are now always recorded with the full four steps[21], allowing them to be tried in rapid succession. Similarly, the static capture move generators also generate the entire capture sequence rather than merely the step that begins the capturing move, and pushes and pulls are always generated with both steps together.

Switching from an originally stepwise formulation to this hybrid formulation by itself resulted in about a 20% speedup for 9-step searches and much more for deeper searches. But more importantly, coupled with tactical

[19]That is, per move on average giving up an amount of advantage that SHARP judges to be 6% of the value of a rabbit in the opening. A rabbit is similar in value to a pawn in Chess.

[20]Unless reduced, moves still decrement the remaining depth of the search proportional to number of steps they contain, with pass steps decrementing by the number of steps left in the turn.

[21]This involves using a principal-variation recording mechanism to extract out the full 4 steps, since the first time these moves are discovered to be best, they are often played over more than one level of recursion.

move generation, this hybrid formulation enables effective late-move reduction. A hybrid formulation makes it easy to separate promising moves from other moves and protect them from being reduced. By contrast, a stepwise search gives only very coarse control over the search - one can only differentiate between moves in a forward manner by their first steps.

6. POSITIONAL EVALUATION

Recent improvements in the evaluation function have also played a major rule SHARP's strength and success. Whereas we managed the first obstacle to strong play in Arimaa, the branching factor, with a series of new innovations to greatly improve the move ordering and selectivity of the search, the second obstacle of crafting a positional evaluation accurate enough to guide the program through strategic positions was handled with simple dedicated engineering - a great deal of hand-tuning and testing guided by expert knowledge (see 6.3). To give the improvements a proper place we start with the description of a testing procedure (6.1) and of a developement cycle (6.2). We describe new components and improvements in 6.4.

6.1 Testing

Positional evaluation in Arimaa is difficult and complicated. Although recently there have been promising developments in automated evaluation learning and tuning (Hrebejk, 2013), the strongest Arimaa programs continue to have hand-written evaluation functions. Like them, SHARP's evaluation function is hand-written. It spans thousands of lines of code and involves dozens of tables and hundreds of numeric parameters and constants.

The only thing that made this complexity manageable was having an effective way to test and inspect the results of changes. Two methods in particular have proven highly valuable, and while these methods are well-known and by no means groundbreaking, they are important, and we present them next in Subsections 6.1.1 and 6.1.2.

6.1.1 Self-Play

Self-play was the most important way of testing changes to SHARP[22]. Every change to the evaluation function (as well almost any change to the search) was tested by playing at least several hundred games against a variety of older versions, often in multi-way tournaments. Some games used very fast controls (such as 4 seconds/move) and some used more normal time controls (such as 15 seconds/move), with faster time controls used to collect data rapidly and slower ones used to confirm differences with less bias.

To handle these multi-way tournaments and compare multiple versions of SHARP at once, we used the well-known BayesElo program (Coulom, 2010) and/or a set of custom tools[23]to analyze the game results assuming the Elo rating model and compute confidence bounds on the relative ratings of different versions. Changes were only accepted if statistically significant or if based on prior expectations the changes were likely to at least not cause harm (ex: a bugfix eliminating a source of random noise in the evaluation function).

While noisy and despite its possible biases, as far as we know, simply playing games is by far the most accurate way to evaluate the effect of a change. Development relied extensively on self-play to ensure that no changes significantly harmed the strength of the program.

6.1.2 Low-depth Play and Hand-Analysis

The second method used to test SHARP's evaluation was to play it against itself or another human with only a 4-step-depth search and watch the game by hand. Modulo details such as quiescence search, a 4-step search selects the move that directly maximizes the evaluation function. As a result, this testing method often revealed strange quirks and preferences by the evaluation function that were clearly wrong or pathological.

[22]Testing against a variety of other strong opponents rather than just oneself would likely be preferable, but at this point there are no other available programs strong enough!

[23]One such custom tool, for example, checks for any statistical intransitivities that would suggest the rating model used was a poor fit.

6.2 Development Cycle

All of these methods facilitated the basic development cycle for the evaluation function. The cycle consisted of the following five steps.

- Identify a deficiency, whether from an observed mistake in a game, expert commentary or feedback, seeing an obvious strategic misjudgment in low-depth play, etc., using search traces to trace bad play down to specific misevaluated board positions[24].
- Using instances of this problem as test positions, iterate on different possible implementations of a new evaluation term or a modification of an existing term to get the program to produce the right evaluation for those positions.
- Use low-depth-play testing to expose whether SHARP now appears to be doing unintended things as a result of side effects of the change. If so, go back and iterate further on the implementation.
- Check the performance cost of the new evaluation code, optimizing or simplifying and iterating further if the performance cost is too large.
- Once the intended evaluation is implemented, test it with a few thousands of games of self-play to determine whether it is ultimately effective or not.

6.3 Basic Evaluation Components

SHARP's evaluation function, for the most part, is linear, simply summing together terms for the various different components:

$$\text{Eval} = \text{Material} + \text{Piece-square tables} + \text{Trap control} + \ldots$$

Many of these terms are themselves sums of subterms where each subterm corresponds to a piece or square or board pattern and is a function of several features, often combined by applying lookup-table based transforms and taking a product. For example:

$$\text{Piece Threat Score} = \sum_{p} \text{Value}(p) * f(\text{TrapScore}(p)) * g(\text{ThreatDistance}(p)) * ...$$

Nine important top-level components are described below.

6.3.1 Material

In Arimaa, like in Chess, the material difference is the most important factor in evaluating a position.

One major difference between Arimaa and Chess is that in Chess, static piece values are a moderately good approximation throughout the game to the average value of a given material balance[25]. In Arimaa, this is not true. Since only relative piece strength matters for pushing, pulling, and freezing, the value of a piece depends heavily on the distribution of opposing piece strengths, with heavier pieces usually devaluing and weaker pieces usually increasing in value as pieces are traded.

To handle these nonlinearities, SHARP uses the "HarLog" material evaluation formula, developed by an anonymous arimaa.com user during a forum discussion thread (Anonymous, 2009):

$$\text{HarLog} = G * \log(\frac{\text{GR} * \text{GP}}{\text{SR} * \text{SP}}) \sum_{p \in \text{NonRabbitPieces}} (1 + C_p)/(Q + O_p)$$

[24]Having an easy-to-use search trace was invaluable, particularly one allowing interactive exploration of the game tree.

[25]Commonly, Pawn = 1, Knight = 3+, Bishop = 3+, Rook = 5, Queen = 9.

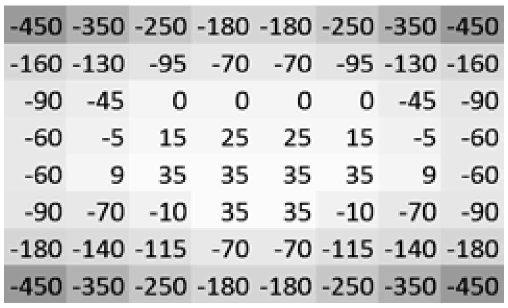
where GR and GP are the number of gold rabbits and pieces, SR and SP are the number of silver rabbits and pieces, O_p is the number of opposing pieces stronger that p, $C_p = 1$ if $O_p = 0$ and else $C_p = 0$, and $G = 0.6314442034$ and $Q = 1.447530126$ are constants.

While somewhat arbitrary, in practice this formula gives values for piece trades that agree closely with evaluations from expert players.

6.3.2 Piece-Square Tables

SHARP also uses the common technique of piece-square tables as a cheap and simple way to encourage different pieces to develop to useful locations, particularly the elephant. For each piece, a small value is added or subtracted using a table lookup based on its location and the number of stronger opposing pieces.

Additionally, for rabbits, to capture the effect that rabbit advances are of slight negative value early on but become good once pieces are traded, an additional table, indexed by the y-coordinate of the rabbit and the sum of twice the number of non-rabbit pieces of the opponent plus the number of rabbit pieces of the opponent, penalizes advancement when the opponent has a lot of material and rewards it when the opponent has less.



-450	-350	-250	-180	-180	-250	-350	-450
-160	-130	-95	-70	-70	-95	-130	-160
-90	-45	0	0	0	0	-45	-90
-60	-5	15	25	25	15	-5	-60
-60	9	35	35	35	35	9	-60
-90	-70	-10	35	35	-10	-70	-90
-180	-140	-115	-70	-70	-115	-140	-180
-450	-350	-250	-180	-180	-250	-350	-450

Figure 6: Piece-square table for the silver elephant

6.3.3 Trap Control

Perhaps the most important positional feature after material in Arimaa is trap control, referring to the ability and ease with which a player can locally defend against material threats in a trap, or conversely, the ability of a player to safely make material threats in that trap.

SHARP estimates trap control by summing over each piece the product of a factor that depends on the Manhattan distance of that piece from the trap with a strength factor that combines both the global and the local material rank of the piece. A variety of additional adjustments are applied, including:

- An adjustment for who has the strongest piece nearby for a couple different choices of radii.
- A bonus for controlling the defense squares closer to the edge of the board (pieces near the edge are less easily dislodged).
- An adjustment for having a weak piece on the trap square itself, varying from a bonus when trap control is otherwise weak (the piece helps defend) to a penalty when otherwise strong (the piece interferes with making threats).

For each trap, the resulting trap control score is fed through a logistic function $x \mapsto 1/(1 + e^{-kx})$ to produce the final evaluation. The trap control scores themselves are also used as inputs to a variety of other evaluation components.

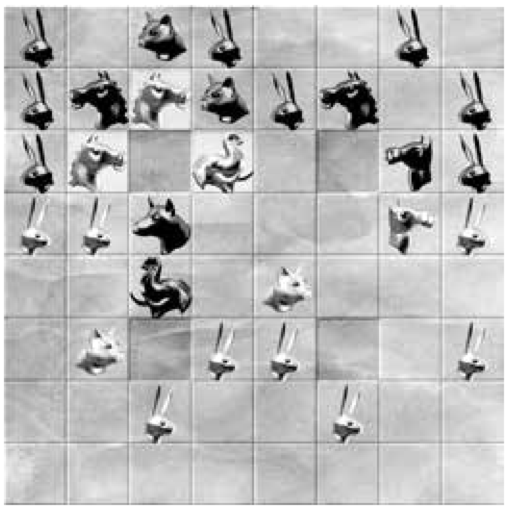


Figure 7: Silver has lost trap control at his c6 home trap, with Gold solidly holding the b6, c7, and d6 squares

6.3.4 Freeing and Domination Distance

For each piece, SHARP approximately calculates the number of steps (0-5) required to *free* a piece to move, where a piece is free if it is unfrozen and has a legal step, push, or pull available to it that does not sacrifice it[26].

Additionally, for each piece, SHARP approximately computes the number of steps (0-5) required by the opponent to *dominate* it, which is to place a stronger opposing piece adjacent to it such that the opposing piece is free.

These values are not used directly in the evaluation score, but rather are used as inputs into most other evaluation components, as they are important features to evaluate how defendable or threatenable a piece is.

6.3.5 Hostages

A common strategic pattern in Arimaa consists of one piece, frequently the elephant, holding another piece "hostage" by pinning and threatening it with capture. This ties the opponent's elephant to defense, ideally giving the hostage-holder the strongest remaining piece on the rest of the board.

SHARP evaluates hostages in a complicated manner involving a wide variety of features, some of which are:

- The trap control score of the relevant trap.
- The pieces involved and whether holding the hostage actually does result in the strongest free piece elsewhere.
- The number of steps required to capture if defense is abandoned.
- The degree of advancement of pieces by the hostaged side (a common counter against a hostage pattern is a swarm of the trap with smaller pieces to free the tied-down elephant from defense).

6.3.6 Frames

Another common strategic pattern in Arimaa is a "frame", where a piece is stuck on a trap, pinning another piece, usually the elephant, to defense. Frames are evaluated using a similarly wide variety of features, including:

- The strength of the piece being framed.

[26]An exception is made where a piece is considered free if it is unable to move only due to friendly pieces blocking it in, would still be unfrozen and unsacrificed if those pieces were to move away, and the situation is not part of a larger "blockade" formation.

- The domination distance of the pieces holding the frame.
- The strength and quantity of pieces needed to hold the frame.
- The ability of the opponent to rotate out and replace the stronger pieces with weaker ones.

6.3.7 Elephant Mobility and Blockades

Another relevant feature is how physically free the elephant is to move - having an elephant that can move and reach many parts of the board quickly is better than having one blockaded or hemmed in.

SHARP uses bitmap operations to estimate the number of squares reachable by the elephant using varying numbers of steps. These computations take into account steps required for stepping, pushing, or pulling blocking pieces, as well as the fact that the elephant cannot step through a trap by itself. The resulting counts, weighted by centrality of squares, are summed together and fed through a lookup table to produce a final score.

Additionally, in the event that the elephant is decentralized and nearly completely blockaded, additional routines are called to estimate the value of the blockade taking into account features very similar to those for a frame:

- The exact position of the elephant and the degree of blockadedness of the elephant.
- The domination distance of the pieces holding the blockade.
- The strength and quantity of opposing pieces needed to hold the blockade.
- The ability of the opponent to rotate out and replace the stronger pieces with weaker ones.

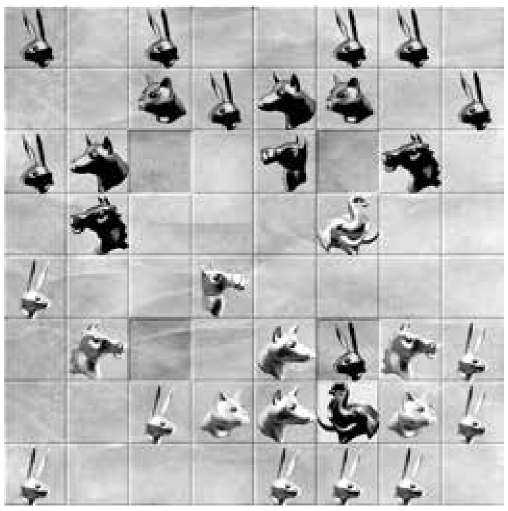


Figure 8: Silver's elephant is blockaded at f2. Silver is completely lost.

6.3.8 Piece Threats

Individual pieces, if threatened with hostaging or capture, can be liabilities. Features relevant in piece-threat evaluation include:

- The trap control score of the relevant trap.
- The number of steps required to capture the piece, including steps to push away defenders of the trap.
- Whether or not the piece is frozen, and whether there are friendly pieces in front of the piece.
- Whether or not it is a rabbit (rabbits can't retreat, making their defense uniquely difficult).

Additionally, the value of the piece itself is computed by considering what the HarLog score would be with and without the piece on the board, and is used to scale the threat. To avoid overpenalizing, in the event that the piece is already being penalized directly for another feature (such as being hostaged), only the maximum of the two penalties is used.

6.3.9 Goal Threats

Highly advanced rabbits can be extremely valuable if they are supported with other pieces or they are in sparsely-defended regions, so that they exert goal-threat pressure on the opponent. Features used in their evaluation include:

- The y-coordinate of the rabbit.
- A score indicating how well-blocked the rabbit is from advancing to goal.
- The trap control score of the nearest opponent trap to the rabbit.
- A heuristic measure of the amount of supporting friendly strength around the rabbit.

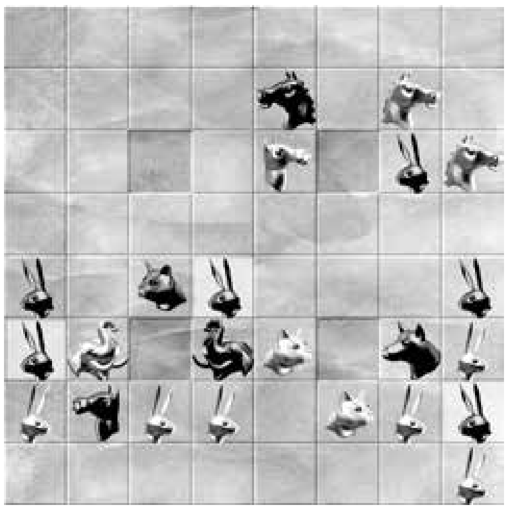


Figure 9: Silver is winning due to the severe goal pressure exerted by the marked rabbits.

6.4 New Components and Improvements

Prior to 2015, one of SHARP's major weaknesses was an understanding of how to place pieces efficiently and strategically. Such an understanding is necessary to play well in many non-tactical situations that involve gradual maneuvering and fighting for position. Although a variety of improvements were made in many areas of the evaluation function, the two largest improvements came from adding two major new components to fix this weakness, centering around the concepts of "piece alignment" (6.4.1) and "imbalances" (6.4.2). A third noteworthy new component involving nonlinearities and "swindles" is discussed in 6.4.3.

6.4.1 Piece Alignment

In Arimaa, "piece alignment" refers to the heuristic that pieces should placed near the opposing pieces that they most efficiently beat - camels should try to be near opposing horses, horses should try to be near opposing dogs, etc. Prior to 2015, a term for this did exist in the evaluation function but it was incomplete and failed to actually capture this heuristic and the nuances around it well.

The new term consists of a sum across each pair of non-rabbit pieces (p_1, p_2) where p_2 is (one of) the strongest opposing piece(s) weaker than p_1, taking into account the following features:

- The estimated number of steps needed for p_1 to dominate p_2, computed more approximately than the domination distance computation described earlier but with no cap on distance.
- The strength and quantity of the pieces of the two types.
- The y-coordinates of the pieces (attacking more advanced pieces is more valuable).

Piece alignment has long been recognized by human players to be important, and its absence from SHARP was one of the largest strategic weaknesses limiting the strength of the program. Implementing this feature properly resulted in an enormous improvement of nearly 80 Elo rating points in self-play testing. This is one of the largest gains ever observed for any single change in the last several years.

6.4.2 Imbalances

The other major new evaluation component in SHARP for strategic piece placement was a term for "imbalances", which refer to inefficiencies caused by having an overconcentration of pieces in the same region or side of the board.

SHARP computes this component of its evaluation in a very similar way to the way it computes the piece alignment value, but instead considers pairs of pieces owned by the same player rather than by different players, and instead of domination distance uses a slightly different notion of distance to assign penalties when many strong pieces are near each other or on the same side of the board.

Unfortunately, the current implementation of this feature fails to capture the strategic concept well enough to match expert opinion in some positions. It is likely that some more iteration and experimentation would produce a different design or feature set that would perform better. Nonetheless its addition was still a large improvement for 2015, gaining about 35 rating points in self-play testing.

6.4.3 Nonlinearities and Swindles

A final recent innovation in SHARP's evaluation function deals with a phenomenon of Arimaa that allowed human players to sometimes "swindle" computer programs out of otherwise won games.

In Arimaa, in situations involving goal threats and fights that threaten to end the game immediately, the values of most other positional factors, such as material, are reduced. Since pieces in Arimaa move slowly, in the event that a goal fight is occurring in a corner of the board, the game can easily be won or lost in a way determined only by the local state of that corner, making the global material balance of the board irrelevant.

Since most strong Arimaa programs, old versions of SHARP included, use evaluation functions that are essentially linear, they can sometimes be defeated in otherwise hopeless positions by sacrificing a large amount of material in order to initiate a goal attack. Often the program will happily take the offered material, oblivious to the fact it is made irrelevant by the attack.

Ideally, one would like to solve such a tactical situation primarily via search, but this is difficult. Moreover, the phenomenon still applies to a lesser degree even if the goal threats are not likely to end the game right away - they can still increase the overall volatility of the position.

SHARP now handles this by explicitly modeling the situation. In particular, the "goal threat" component of the evaluation function, in addition to outputting a term that gets linearly added into the score, also outputs probabilities g and s that Gold and Silver, respectively, win or otherwise turn the game around in the short term due to the volatility of their goal threats. The normal linear part of the evaluation, x is computed and then fed through a sigmoid to produce a probability $p = 1/(1+e^{-kx})$ that Gold wins the game if the game is instead won over the long term. The final output of the evaluation function as a whole is simply the total probability that Gold wins:

$$\text{Final eval} = 1 * g + 0 * s + p * (1 - g - s)$$

In certain positions, this change makes it noticeably harder to swindle SHARP, since when ahead, SHARP is more likely to choose a "safe" and "conservative" move as opposed to a more "greedy" move. Additionally, it is more likely to attempt to start goal fights when sufficiently behind. And indeed, in self-play testing, this change resulted in an improvement of about 15 Elo points.

7. RESULTS AND FUTURE WORK

Over the last few years, SHARP has improved greatly in all areas of play. SHARP demonstrates that it is possible to achieve effective reduction/pruning in Arimaa by leveraging expert game data using local move features and patterns, and to meet the challenge of producing an accurate positional evaluation by employing good methods for iterative tuning and testing.

Year	2008	2010	2011	2012	2014	2015
Blitz (15s/move)	N/A	2045	2139	2256	2286	N/A
CC (2m/move)	1513	2035	1980	2117	2172	2487
Self-play gain	N/A	Untested	Untested	100-250?	≈200	≈400

Table 2: Elo ratings of versions of SHARP on arimaa.com as of July 16, 2015 at different time controls, along with estimated single-threaded self-play gain over the previous version at fast testing speeds.

However, SHARP's heuristics and algorithms are far from perfect. There are multiple promising avenues for further improvement in both search and evaluation. In the search, a variety of basic things have simply not been tried yet, ranging from a more sophisticated replacement scheme for the hashtable, to razoring and other heuristic pruning of nodes near leaves, to other enhancements such as "principal variation search" (Marsland, 1986). Tuning of the heuristics in components such as the tactical move generation could also yield improvements.

In the evaluation function, particularly with recent work demonstrating the feasibility of automated learning (Hrebejk, 2013), there is huge potential to improve SHARP by tuning the hundreds of different weights and coefficients on all of the features that have been implemented. Identifiable missteps and strategic misunderstandings in some of its recent games also suggest new features to add. Evaluation of positions in the opening remains a particularly weak part of SHARP's game, and almost certainly large improvements are possible.

While SHARP may have won the Challenge, there is still a small way to go to produce a bot that is unambiguously dominant over the top human players rather than merely competitive or favored, and beyond that there is still plenty of room to experiment and grow[27]. However, with our results Arimaa now crosses the threshold, leaving the ranks of the games holding out against effective computer play and joining the ever-growing pool of games in which computers are the strongest players.

8. ACKNOWLEDGEMENTS

I would like to thank my former undergraduate thesis advisor, Professor David Parkes, for his guidance and feedback on this paper. I would also like to thank Omar Syed as well as the Arimaa community as a whole for giving me the chance to work on this fun project and be involved in this amazing game.

9. REFERENCES

Anonymous (2009). Arimaa forum discussion thread. http://arimaa.com/arimaa/forum/cgi/YaBB.cgi?board=devTalk;action=display;num=1062013358;start=60. Accessed on 2015-07-12.

Coulom, R. (2010). Bayesian Elo Rating. http://www.remi-coulom.fr/Bayesian-Elo/. Accessed on 2015-07-15.

[27]As of now, July 2015, as a fun exhibition, SHARP with some human assistance is beginning a postal game against the "Mob", a team of human players including many top players, at time controls of a week per move. At such time controls the Mob is likely the favorite, but we'll see what happens as the game unfolds over the next year!

Cox, C.-J. (2006). Analysis and Implementation of the Game Arimaa. M.Sc. thesis, Maastricht University, The Netherlands.

Fotland, D. (2006). Building a World-Champion Arimaa Program. *Proceedings 4th International Computer and Games Conference* (eds. H. van den Herik, Y. Björnsson, and N. Netanyahu), Vol. 3846 of *Lecture Notes in Computer Science*, pp. 175–186, Springer Verlag, Heidelberg.

Haskin, B. (2006). A Look at the Arimaa Branching Factor. http://arimaa.janzert.com/bf_study/. Accessed on 2015-07-12.

Hrebejk, T. (2013). Arimaa challenge - static evaluation function. M.Sc. thesis, Charles University, Prague.

Hyatt, R. (1994). The DTS high-performance parallel tree search algorithm. https://cis.uab.edu/hyatt/search.html. Accessed on 2015-07-12.

Kozelek, T. (2009). Methods of MCTS and the game Arimaa. M.Sc. thesis, Charles University of Prague, Czech Republic.

Marsland, T. A. (1986). A Review of Game-Tree Pruning. *ICGA Journal*, Vol. 9, No. 1, pp. 3–19.

Miller, S. (2009). Researching and Implementing a Computer Agent to Play Arimaa. Thesis, University of Southampton, UK.

Romstad, T. (2007). An Introduction to Late Move Reductions. http://www.glaurungchess.com/lmr.html. Accessed on 2015-07-16.

Schaeffer, J. (1989). The History Heuristic and Alpha-Beta Search Enhancements in Practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, pp. 1203–1212.

Syed, O. and Syed, A. (2003). Arimaa - A New Game Designed to be Difficult for Computers. *ICGA*, Vol. 26, No. 2, pp. 138–139.

Syed, O. and Syed, A. (2015a). Arimaa - The next challenge. http://arimaa.com/arimaa/. Accessed on 2015-07-12.

Syed, O. and Syed, A. (2015b). The Arimaa World Championship. http://arimaa.com/arimaa/wcc/. Accessed on 2015-07-12.

Trippen, G. (2009). Plans, Patterns and Move Categories Guiding a Highly Selective Search. *Proceedings 12th Advances in Computer Games Conference* (eds. H. van den Herik and P. Spronck), Vol. 6048 of *Lecture Notes in Computer Science*, pp. 111–222, Springer Verlag, Heidelberg.

Vivek Choksi, V. M., Neema Ebrahim-Zadeh (2013). Move Ranking in Arimaa. Accessed on 2015-07-12.

Wu, D. (2011). Move Ranking and Evaluation in the Game of Arimaa. Bachelor of arts, Harvard College, Cambridge, Massachusetts.

Zhong, H. (2005). Building a Strong Arimaa-playing Program. M.Sc. thesis, University of Alberta, Dept. of Computing Science.

NOTES

CHESS ENDGAME NEWS

G.M^c^C. Haworth[1]

Reading, UK

The recent focus by Newborn and Hyatt (2014) on the increasing ability of chess engines to play endgames is most welcome. They revisited a 16-position test set *TS1* derived from Fine (1941) and demonstrated that the seminal chess engine CRAFTY (Hyatt, 2015) backed only by 5-man 'EGT' endgame-tables (Nalimov et al, 2000) could now handle it with ease. They therefore considered a second 16-position test set *TS2* as a stiffer benchmark challenge for CRAFTY other chess engines.

The following engines are defined here:

- M_{k-1}: filters moves in sub-*k*-man ('s*k*m') positions by minimaxing on Depth to Mate ('DTM'),
- *F*: filters moves by minimaxing on the DTF depths defined by FINALGEN (Romero, 2012),
- C_5: filters by minimaxing on 'DTC' Depth to Conversion and using s6m DTC EGTs,[2]
- Z_6: filters by minimaxing on 'DTZ' Depth to Zeroing of the ply-count, using s7m DTZ_{50}' EGTs,[3, 4]
- $E \equiv M_7FC_5Z_6$: filters by deploying engines M_7, *F*, C_5 and Z_6 in turn,
- *H*: Hyatt's CRAFTY, unassisted by EGT support,
- HM_5: the engine used by Newborn and Hyatt (2014),
- *X*: the author's FRITZ14 engine, analysing at 3mins/position, and
- *EH*: notional engine, EGT-based but supported by CRAFTY as needed, whose performance is defined here.

FINALGEN (Romero, 2012) provides depth to mate and/or winning pawn-conversion for positions with at most one piece per side and sufficiently limited pawn mobility. Because it cannot contemplate, e.g., endgame KQQKP, DTF depths can be greater than DTM depths. FINALGEN builds its EGTs in single-thread mode and does not call on non-FINALGEN EGTs. Engine *E* considers the four depth metrics in 'most distant first' sequence, the most describable of the twenty-four sequences available! The use of the DTF/C/Z metrics does not affect 'moves to mate' but can isolate a unique optimal move when DTM alone does not. Table 3 gives example positions and moves, also illustrating the sort of unnatural move-choices that (non-DTM) metric arithmetic can dictate.

EH and HM_5 can choose different moves but where the position is beyond all available EGT-based machines, *EH* is effectively engine *H* and, here, plays the move attributed to CRAFTY by Newborn and Hyatt (2014). The objectives of deploying engine *EH* on test sets TS1 and TS2 were to:

- exercise FINALGEN and the Lomonosov 7m DTM EGTs (MVL, 2015) where possible,
- examine to what extent each '*EH* element' contributed in finding a best line from the test positions,
- compare the move-choices and 'moves to mate' of HM_5 and *EH*,
- examine the uniqueness and optimality of the moves available,
- consider what the characteristics might be of good positions in a notional test set TS3.

Table 1 details the positions of test sets TS1 and TS2, and indicates the performance of engines HM_5 and *EH* on them. Note that these are mainly wtm wins except for a wtm draw (TS2.07), a btm draw (TS2.02) and three btm wins for Black (TS1.13 and TS2.04/05). It should also be noted that TS1.15 ≡ TS2.13.

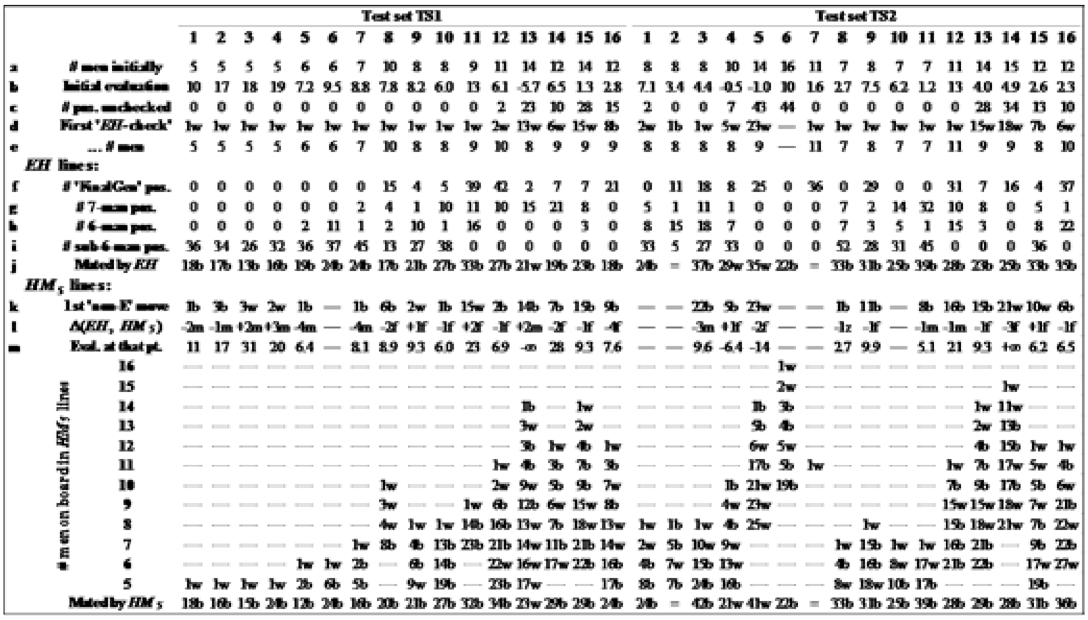
Table 2's row *a* indicates the initial number of men for each position: row *b* provides a DEEP FRITZ14 3-minute evaluation of the initial position. Row *c* gives the number of positions which are beyond the scope of engine *E*, with row *d* giving the first position checked by *EH* and row *e* giving the number of men at that point. Rows *f-i* indicate the number of positions where, respectively, FINALGEN, 7-man (7m), 6m and s6m DTM EGTS are the first endgame tools used within *EH*: this data is also illustrated graphically in more detail in Figure 1. Row *k* indicates the first position at which engines *EH* and HM_5 differ, with row *l* indicating the nature of the

[1] The University of Reading, Berkshire, UK, RG6 6AH. email: guy.haworth@bnc.oxon.org.
[2] Bourzutschky and Konoval computed all 6m DTC EGTs (Haworth, 2013) but these are not publicly available.
[3] The use of De Man's s7m DTZ_{50}' EGTs (CPW, 2013) is valid: the FIDE 50m draw-rule does not become relevant here.
[4] Further, C_n and Z_n prefer/defer a change of force or pawn-push even if there is no EGT, q.v., Table 3, #09.

'suboptimality' from *EH's* point of view. A further 3-minute evaluation by FRITZ10 at that point as given in row *m* makes it clear that, apart from position TS2.11, any engine would likely secure a result from those positions. Data beyond row *m* indicates the count of men for each subsequent position in the HM_5 line provided. Haworth (2015) provides supporting data, including extensions of the tables, pgn files and annotated lines of play.

TS#	Identity	#m	w-b	Material	FEN	Val.	'Best' move	*dtm*	Mate, ply HE	EH	Δ
1.01	Fine 25	5	3-2	KPPKP	6k1/7p/5P1K/8/8/8/7P/8 w - - 0 1	1-0	Kg5	35	35	35	0
1.02	Fine 26	5	3-2	KPPKP	8/2k5/p1P5/P1K5/8/8/8/8 w - - 0 1	1-0	Kd5	33	31	33	-2
1.03	Fine 29	5	3-2	KPPKP	4k3/4Pp2/5P2/4K3/8/8/8/8 w - - 0 1	1-0	Kf5	25	29	25	4
1.04	Fine 42	5	3-2	KPPKP	8/5p2/8/4K1P1/5Pk1/8/8/8 w - - 0 1	1-0	Ke4	33	47	31	16
1.05	Fine 51	6	3-3	KPPKPP	8/8/2pp3k/8/1P1P3K/8/8/8 w - - 0 1	1-0	d5	37	23	37	-14
1.06	Fine 53	6	3-3	KPPKPP	8/8/3pkp2/8/8/3PK3/5P2/8 w - - 0 1	1-0	Ke4	47	47	47	0
1.07	Fine 58	7	4-3	KP(3)KPP	8/8/2ppk3/8/2PPK3/2P5/8/8 w - - 0 1	1-0	d5+	43	31	47	-16
1.08	Fine 61	10	5-5	KP(4)KP(4)	8/ppp5/8/PPP3kp/8/6KP/8/8 w - - 0 1	1-0	b6	?	39	33	6
1.09	Fine 66	8	4-4	KP(3)KP(3)	8/1k3ppp/8/3K4/7P/5PP1/8/8 w - - 0 1	1-0	Kd6	?	41	41	0
1.10	Fine 67	8	4-4	KP(3)KP(3)	8/2p5/3k4/1p1p1K2/8/1P1P4/2P5/8 w - - 0 1	1-0	b4	?	53	53	0
1.11	Fine 70	9	5-4	KP(4)KP(3)	8/k7/3p4/p2P1p2/P2P1P2/8/8/K7 w - - 0 1	1-0	Kb1	?	63	65	-2
1.12	Fine 76	11	6-5	KP(5)KP(4)	8/8/p6p/1p3kp1/1P6/P4PKP/5P2/8 w - - 0 1	1-0	f4	?	67	53	14
1.13	Fine 80	14	7-7	KP(6)KP(6)	8/8/1ppk4/p4pp1/P1PP2p1/2P1K1P1/7P/8 b - - 0 1	0-1	b5	?	43	39	4
1.14	Fine 82	12	6-6	KP(5)KP(5)	8/pp5p/8/PP2k3/2P2pp1/3K4/6PP/8 w - - 0 1	1-0	c5	?	57	37	20
1.15	Fine 90	14	7-7	KP(6)KP(6)	8/7p/2k1Pp2/pp1p2p1/3P2P1/4P3/P3K2P/8 w - - 0 1	1-0	e4	?	57	45	12
1.16	Fine 100A	12	6-6	KP(5)KP(5)	8/6p1/3k1p2/2p2Pp1/2P1p1P1/1P4P1/4K3/8 w - - 0 1	1-0	Kf2	?	47	35	12
2.01	CCE4 479	8	4-4	KP(3)KP(3)	8/1p4kP/5pP1/3p4/8/4P3/7K/8 w - - 0 1	1-0	e4	?	47	47	0
2.02	CCE4 491a	8	4-4	KP(3)KP(3)	8/1pp5/3k3p/PP6/2P2K2/8/8/8 b - - 0 1	=	Kd7	=	=	=	=
2.03	CCE4 530	8	4-4	KP(3)KP(3)	2k2K2/8/pp6/2p5/2P5/PP6/8/8 w - - 0 1	1-0	a4	?	83	73	10
2.04	CCE4 608	10	5-5	KP(4)KP(4)	8/pp3p2/8/6kp/8/3K1PP1/PP6/8 b - - 0 1	0-1	f5	?	39	55	-16
2.05	CCE4 679	14	7-7	KP(6)KP(6)	8/pp2k1pp/2p5/2P1p3/2P1P2P/6P1/P7/2K5 b - - 0 1	0-1	g5	?	79	67	12
2.06	CCE4 680	16	8-8	KP(7)KP(7)	8/1p6/p1p5/P1Pp2pp/1P1P1p1k/5P1P/6PK/8 w - - 0 1	1-0	g3/g4	?	43	43	0
2.07	CCE4 765	11	6-5	KP(5)KP(4)	8/1k6/p4p2/2p2P2/p1P2P2/2P5/P1K5/8 w - - 0 1	=	Kc1	=	=	=	=
2.08	Lejeune 2	7	4-3	KP(3)KPP	k7/4p3/4p3/8/8/3P1P2/5P2/K7 w - - 0 1	1-0	Kb2	?	65	65	0
2.09	Lejeune 4	8	4-4	KP(3)KP(3)	k7/8/1p6/p1p5/2P4K/8/PP6/8 w - - 0 1	1-0	a4	?	61	61	0
2.10	Lejeune 5	7	4-3	KP(3)KPP	8/8/p7/8/1P6/7p/P4k1P/3K4 w - - 0 1	1-0	a3	?	49	49	0
2.11	Lejeune 6	7	4-3	KP(3)KPP	8/5p1p/8/6k1/8/6P1/5PP1/7K w - - 0 1	1-0	Kh2	?	77	77	0
2.12	Christmas Cracker	11	6-5	KP(5)KP(4)	3k4/3p4/3p4/p2P2p1/P2P2P1/3P4/3K4/8 w - - 0 1	1-0	Kd1	?	55	55	0
2.13	Pillsbury 1895	14	7-7	KP(6)KP(6)	8/7p/2k1Pp2/pp1p2p1/3P2P1/4P3/P3K2P/8 w - - 0 1	1-0	e4	?	57	45	12
2.14	Capablanca 1919	15	8-7	KP(7)KP(6)	8/5p2/2kp1p1p/p1p2P2/2P5/7P/PP3PP1/6K1 w - - 0 1	1-0	a4	?	55	49	6
2.15	Botvinnik 1944	12	6-6	KP(5)KP(5)	8/1p3k2/p4ppp/3P4/1P6/4K2P/1P4P1/8 w - - 0 1	1-0	g3/g4	?	61	65	-4
2.16	Botvinnik 1958	12	6-6	KP(5)KP(5)	8/4pk2/1p4p1/1P2p3/3pP2P/3K2P1/4P3/8 w - - 0 1	1-0	Kc4	?	71	69	2

Table 1. Key data for the positions of test sets TS1 and TS2.



		Test set TS1															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
a	# men initially	5	5	5	5	6	6	7	10	8	8	9	11	14	12	14	12
b	Initial evaluation	10	17	18	19	7.2	9.5	8.8	7.8	8.2	6.0	13	6.1	-5.7	6.5	13	2.8
c	# pos. unchecked	0	0	0	0	0	0	0	0	0	0	0	2	23	10	28	15
d	First '*EH*-check'	1w	1w	1w	1w	1w	1w	1w	1w	1w	1w	1w	2w	13w	6w	15w	8b
e	... # men	5	5	5	5	6	6	7	10	8	8	9	10	8	9	9	9
EH lines:																	
f	# 'FinalGen' pos.	0	0	0	0	0	0	0	15	4	5	39	42	2	7	7	21
g	# 7-man pos.	0	0	0	0	0	0	2	4	1	10	11	10	15	21	8	0
h	# 6-man pos.	0	0	0	0	2	11	1	2	10	1	16	0	0	0	3	0
i	# sub-6-man pos.	36	34	26	32	36	37	45	13	27	38	0	0	0	0	0	0
j	Mated by *EH*	18b	17b	13b	16b	19b	24b	24b	17b	23b	27b	33b	27b	21w	19b	23b	18b
HM_5 lines:																	
k	1st 'non-E' move	1b	3b	3w	2w	1b	—	1b	6b	2w	1b	15w	2b	14b	7b	15b	9b
l	$\Delta(EH, HM_5)$	-2m	-1m	+2m	+3m	-4m	—	-4m	-2f	+1f	-1f	+2f	-1f	+2m	-2f	-1f	-4f
m	Eval. at that pt.	11	17	31	20	6.4	—	8.1	8.9	9.3	6.0	23	6.9	-∞	28	9.3	7.6
# men on board in HM_5 lines	16	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
	15	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
	14	—	—	—	—	—	—	—	—	—	—	—	—	1b	—	1w	—
	13	—	—	—	—	—	—	—	—	—	—	—	—	3w	—	2w	—
	12	—	—	—	—	—	—	—	—	—	—	—	—	3b	1w	4b	1w
	11	—	—	—	—	—	—	—	—	—	—	—	1w	4b	3b	7b	3b
	10	—	—	—	—	—	—	—	1w	—	—	—	2w	9w	5b	9b	7w
	9	—	—	—	—	—	—	—	3w	—	—	1w	6b	12b	6w	15w	8b
	8	—	—	—	—	—	—	—	4w	1w	1w	14b	16b	13w	7b	18w	13w
	7	—	—	—	—	—	—	1w	8b	4b	13b	23b	21b	14w	11b	21b	14w
	6	—	—	—	—	1w	1w	2b	—	6b	14b	—	22w	16w	17w	22b	16b
	5	1w	1w	1w	1w	2b	6b	5b	—	9w	19b	—	23b	17w	—	—	17b
	Mated by HM_5	18b	16b	15b	24b	12b	24b	16b	20b	23b	27b	32b	34b	23w	29b	29b	24b

		Test set TS2															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
a	# men initially	8	8	8	10	14	16	11	7	8	7	7	11	14	15	12	12
b	Initial evaluation	7.1	3.4	4.4	-0.5	-1.0	10	16	2.7	7.5	6.2	12	13	4.0	4.9	2.6	2.3
c	# pos. unchecked	2	0	0	7	43	44	0	0	0	0	0	0	28	34	13	10
d	First '*EH*-check'	2w	1b	1w	5w	23w	—	1w	1w	1w	1w	1w	1w	15w	18w	7b	6w
e	... # men	8	8	8	8	9	—	11	7	8	7	7	11	9	9	8	10
EH lines:																	
f	# 'FinalGen' pos.	0	11	18	8	25	0	36	0	29	0	0	31	7	16	4	37
g	# 7-man pos.	5	1	11	1	0	0	0	7	2	14	32	10	8	0	5	1
h	# 6-man pos.	8	15	18	7	0	0	0	7	3	5	1	15	3	0	8	22
i	# sub-6-man pos.	33	5	27	33	0	0	0	52	28	31	45	0	0	0	36	0
j	Mated by *EH*	24b	=	37b	29w	35w	22b	=	33b	31b	25b	39b	28b	23b	25b	33b	35b
HM_5 lines:																	
k	1st 'non-E' move	—	—	22b	5b	23w	—	—	1b	11b	—	8b	16b	15b	21w	10w	6b
l	$\Delta(EH, HM_5)$	—	—	-3m	+1f	-2f	—	—	-1z	-1f	—	-1m	-1m	-1f	-3f	+1f	-1f
m	Eval. at that pt.	—	—	9.6	-6.4	-14	—	—	2.7	9.9	—	5.1	21	9.3	+∞	6.2	6.5
# men on board in HM_5 lines	16	—	—	—	—	—	1w	—	—	—	—	—	—	—	—	—	—
	15	—	—	—	—	—	2w	—	—	—	—	—	—	—	1w	—	—
	14	—	—	—	—	1b	3b	—	—	—	—	—	—	1w	11w	—	—
	13	—	—	—	—	5b	4b	—	—	—	—	—	—	2w	13b	—	—
	12	—	—	—	—	6w	5w	—	—	—	—	—	—	4b	15b	1w	1w
	11	—	—	—	—	17b	5b	1w	—	—	—	—	1w	7b	17w	5w	4b
	10	—	—	—	1b	21w	19b	—	—	—	—	—	7b	9b	17b	5b	6w
	9	—	—	—	4w	23w	—	—	—	—	—	—	15w	15w	18w	7w	21b
	8	1w	1b	1w	4b	25w	—	—	—	1w	—	—	15b	18w	21w	7b	22w
	7	2w	5b	10w	9w	—	—	—	1w	15b	1w	1w	16b	21b	—	9b	22b
	6	4b	7w	15b	13w	—	—	—	4b	16b	8w	17w	21b	22b	—	17w	27w
	5	8b	7b	24b	16b	—	—	—	8w	18w	10b	17b	—	—	—	19b	—
	Mated by HM_5	24b	=	42b	21w	41w	22b	=	33b	31b	25b	39b	28b	29b	28b	31b	36b

Table 2. The benchmarking of HM_5 lines by engine *EH* in the context of the force-profiles.

Some headlines from the results:

- The 4 5m and 2 6m tests TS.01-06 are solvable using accessible Nalimov s7m DTM EGTs,
- a further 4 tests, TS1.07 and TS2.08/10/11, are solvable using MVL (2015) 7m DTM EGTs,
- a further 9 tests, TS1.08-11 and TS2.02/03/07/09/12 are solvable if only FINALGEN is also used,
- the remaining 13 tests (TS1.12-16 and TS2.01/04-06/13-16) require the initial use of CRAFTY,
- across TS1/2, CRAFTY, FINALGEN, 7m, 6m and s6m DTM EGTs are *EH's* lead evaluator as follows: CRAFTY 78/181, FINALGEN 142/222, 7m DTM 82/97, 6m 46/112 and 5m 324/290 times, i.e., in % terms, CRAFTY 12/20, FINALGEN 21/25, 7m DTM 12/11, 6m 7/12 and 5m 48/32, q.v., Figure 1 for a graphical representation of the breakdown per position,
- HM_5 and *EH* played identically on 6 tests, TS1.06 and TS2.01/02/06/07/10.

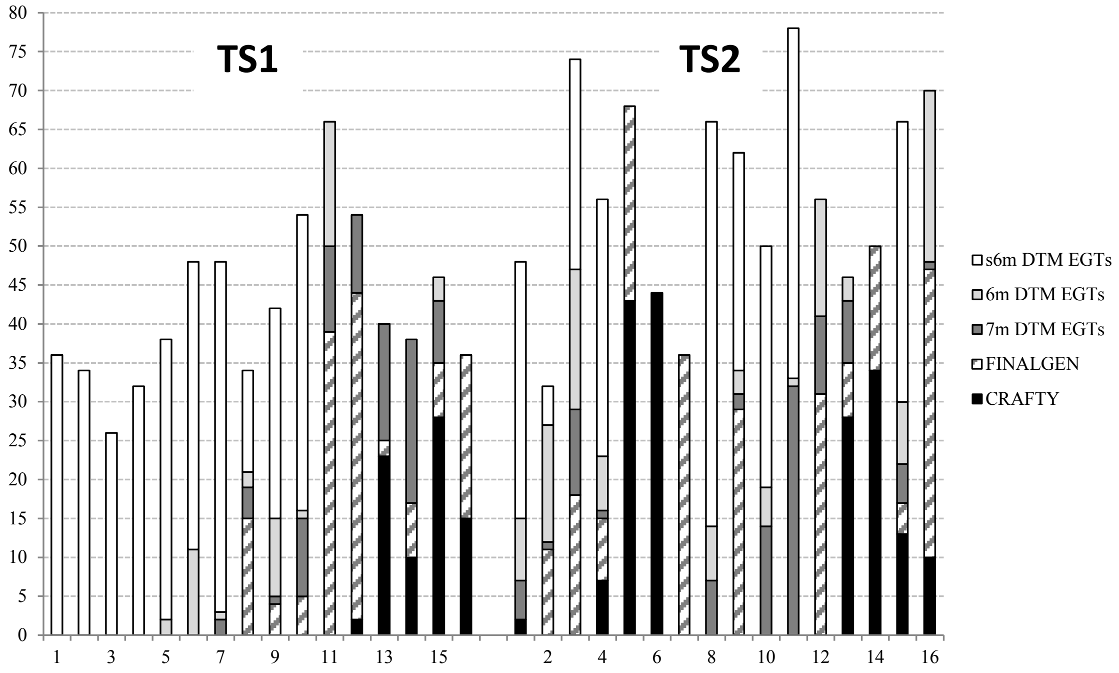


Figure 1. TS1/2: number of plies for which CRAFTY, FINALGEN or 7m/6m/s6m EGTs are *EH*'s lead evaluator.

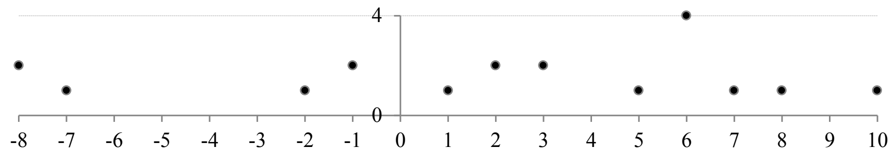


Figure 2. TS1/2: distribution of the difference of CRAFTY'S and *EH*'s '#moves to mate'.

Figure 2 shows the frequency of mate-length differences other than zero, of which there are thirteen. HM_5 in fact only takes 21 more moves across the 32 positions of TS1/2 than *EH*: the four outliers are TS1.05/TS1.07/TS2.04 where HM_5 concedes for the defense, and TS1.14 where HM_5 takes 10 moves more to mate. From TS1.05, a total of 7 moves are conceded with moves 1b (4m), 3b (2m) and 4b (1m). From TS.07, a net total of 8 moves are conceded, see moves 1b/2w/2b/4b/6w/8w. From TS2.04, the three moves **10. Ke2 Ke4 11. Kd2** concede 4, 4 and then crucially 7 moves respectively in DTM terms. For TS1.14, White's moves 12/13/14/16 concede 1/1/1/5 moves in DTM terms.

HM_5 resolves all the TS1/2 tests whereas, for 13 positions, *E* is as useful as a Mark 1 Dalek in a stairwell and requires CRAFTY's initial help. Although all 5m EGTs can be created in nine phases[5] as ancillary, parallel threads of computation within CRAFTY's hour, it would be interesting to see the performance of CRAFTY, completely unassisted by M_5, both playing itself and with *EH* taking the other side.

[5] The 9 phases are: 3-man (2 phases ≡ 0/1 pawns), 4-man (3 phases ≡ 0/1/2 pawns) and 5-man (4 phases ≡ 0/1/2/3 pawns). Phases 1-5 take ~30 seconds and phases 6-9 take ~30m each, times easily improved with more parallelism.

#	TS	move	FEN	E-DTM	E-DTF	E-DTC	E-DTZ	Notes
01	1.01	1w	6k1/7p/5P1K/8/8/8/7P/8 w - - 0 1	**Kg5′′′′**	—	—	—	Absolutely unique winning move
02	1.11	3w	8/2k5/3p4/p2P1p2/P2P1P2/8/8/2K5 w - - 0 1	?	Kd1‴	—	—	Effectively unique: alternative Kb1 merely retracts move 2w
03	1.01	1b	6k1/7p/5P2/6K1/8/8/7P/8 b - - 0 1	**Kf7″**	—	—	—	The DTM EGT decides
04	1.01	10w	8/5k2/5P2/6KP/8/8/8/8 w - - 0 1	h6′	**h6″**	—	—	The DTF EGT decides
05	1.01	2w	5k2/7p/5P2/6K1/8/8/7P/8 w - - 0 1	Kf5′	Kf5′	**Kf5″**	—	The DTC EGT decides
06	2.08	7w	8/4p3/8/4Pk2/2K2P2/8/5P2/8 w - - 0 1	Kc5′	Kc5′	?	**Kc5″**	The DTZ EGT decides
07	1.12	14w	8/6k1/p7/1p3PKp/1P5P/P7/8/8 w - - 0 1	?	Kxh5′	**Kxh5″**	—	DTC decides without EGT: immediate capture ⇒ *dtc* = 1p
08	1.09	2w	8/1k4pp/3K4/5p2/7P/5PP1/8/8 w - - 0 1	?	g4′	?	**g4″**	DTZ decides without EGT: immediate P-push ⇒ *dtz* = 1p
09	1.12	24b	8/5k2/3Q4/pp4K1/1P6/P7/8/8 b - - 0 1	Kg7′	Kg7′	Kg7′	**Kg7″**	DTC/Z decide without EGT: axb4 and a4 are both rejected
10	1.14	15b	7Q/pP6/Pk6/1P6/6K1/8/8/8 b - - 0 1	Kc5′	**Kc5″**	—	—	*dtm* < *dtf*: *dtm* = 8p and *dtf* = 16p
11	1.14	16w	7Q/pP6/P7/1Pk5/6K1/8/8/8 w - - 0 1	b8Q′	Qb2′	**b8Q″**	—	FINALGEN cannot contemplate endgame KQQPPKP
12	2.03	25w	8/2P5/pK1k4/p7/8/1P6/8/8 w - - 0 1	**c8Q″**	—	—	—	… and not CRAFTY's 25. Kxa5, reaching for the 5m EGTs
13	1.12	20w	6k1/8/p4P1P/1p4K1/1P6/P7/8/8 w - - 0 1	?	**f7″**	—	—	Unnatural: 20. Kg6 promotes a pawn quickly
14	1.12	20b	6k1/5P2/p6P/1p4K1/1P6/P7/8/8 b - - 0 1	?	**Kf8″**	—	—	Unnatural: 20. … Kxf7 clearly prolongs the line
15	1.09	3b	8/1k4pp/3K4/8/6PP/8/8/8 b - - 0 1	Kb6′	Kb6′	?	**Kb6″**	DTF-excluded, DTZ-u-optimal Kc8! requires Ke(6/7) ′′′′
16	1.15	17b	1Q6/2k5/4P3/5P2/7p/6p1/7P/K7 b - - 0 1	?	**Kc6″**	—	—	Kc6 '⇒' *dtf* = 24p though *dtm* = ~12p. Kxb8 is more natural

Table 3. Some illustrative positions and decisions taken by engine *E*.

Considerations of the two key resources, space and time, suggest that the *EH*/HM_5 performance comparison is one of unlike ‘apples and pears’. In the latter days of man-machine contests, the use of 6-man EGTs was banned for these reasons. Engine *E* inherits the unlimited space/time resources used to compute EGTs whereas HM_5 is using predetermined space/time and only one hour of real-time solving time. The use of ‘WDL’ win/draw/loss EGTs on ever-greater GBytes of low-latency SSD memory will facilitate access to EGTs deeper into the forward search. Figure 1 shows how far down the search-tree CRAFTY would have to progress before invoking a ‘FINALGEN’ to create some EGTs: 2 ply in TS2.01, 3 ply in TS1.12 but 44 ply in TS2.05/06.

There are at least two measures of the ‘difficulty’ of a position. One is the time taken to identify and adopt what is the best move but this will reduce as hardware evolves. A second, more hardware-independent measure, is the apparent relative merit of ‘the best move’ at various depths of search by one or more engines, though it is not yet clear how this can be turned into a single number even for one engine.

As a footnote, the criticality of the position and value *v* of ‘the move’ can be assessed by analyzing the TS1/2 positions with the other side to move (ostm). Verdicts after ‘◈’ are not purely EGT-based but required some tree-search and evaluation:

- *v* = 1 point, win becomes a loss: TS2.01; ◈ TS1.12/15, TS2.06/13
- *v* = ½ point, win becomes a draw: TS1.01/05-11, TS2.03/08-11; ◈ TS1.14, TS2.04-05/15-16
- *v* = ½ point, draw becomes a loss: TS2.02
- *v* = 0 point, result unchanged:
  - TS1.02-04, TS2.12. Note that TS1.03/04 and TS2.12 are type-BM zugs;[6]
  - ◈ TS1.13/16, TS2.07/14

Clearly, as chess engines search more deeply and therefore improve, the creation of challenging test sets becomes harder. Their purpose is primarily to test chess-engines’ search and evaluation rather than their use of pre- or even runtime-computed EGTs. Therefore, while the value of positions should be known, they should not be clearly decisive, have best opening moves which are quickly found or be in an EGT or in range of FINALGEN. Only a few TS1/2 positions, including TS2.01/09/10/12, distinguish themselves in this regard today. The focus on pawns, especially those with restricted movement, and the initial exclusion of pieces is helpful to both FINALGEN and chess-engine search, so the exclusive use of KP-endgames is an onerous restriction but one which is fortunately unnecessary.

The Chess Study epitomizes the ‘hard to solve’ position and TS1/2 used 16 of these. Some other investigations of ‘anti-computer’, even ‘impossible to solve’, positions, have drawn entirely on the corpus of studies, currently represented without peer by van der Heijden’s HHDBIV (2010). Three notable articles are those by van der Heijden himself (2006, 2014) and Vlasák (2013). However, it should be said that many of their choices look more like ‘game’ than ‘endgame’ positions, one having as many as 22 men on the board. This suggests that there should be separate accolades for the most difficult *m*-man positions. Do those positions of most marginal advantage with the greatest metric-depth (Haworth, 2013a/b) provide the greatest challenge to chess-engines if they do not have access to the relevant endgame table?

[6] A type-BM zug is one in which DTM is greater with the move than without it (Bleicher and Haworth, 2010)

#	id	HHdbIV		Author(s)	Year	m	w-b	GBR code	s8m	F	Position: 'FEN' Forsyth Extended Notation	Val.	DEEP FRITZ 14, 2-core, 3m				
		#	pos.										Eval.	Δ	Depth	Choice	✓/×
01	H01	4728	1w	Behting	1906	9	4-5	0002.14	n	n	8/8/7p/3KNN1k/2p4p/8/3P2p1/8 w - - 0 1	=	-4.10	1.12	21	1. Ng7+	×
02	H02	25700	6w	Gurvich	1952	7	4-3	0044.10	Y	n	1N6/6k1/1B6/8/1P6/8/b1K5/n8 w - - 0 1	1-0	4.35	0.44	25	6. Kd1	×
03	H03	70286	3w	Antonini	2003	7	3-4	4001.02	Y	n	8/8/2K5/4N3/p5Q1/4k2p/8/q7 w - - 0 1	1-0	12.20	0.00	19	3. Qf3+	✓
04	H04	56516	4w	Pervakov & Grin	1988	7	3-4	1400.02	Y	n	8/8/8/8/5R2/8/kr1p2p1/3K3Q w - - 0 1	1-0	∞	∞	21	4. Qh2	✓
05	H05	56539	4w	Kuryatnikov	1988	7	4-3	0041.11	Y	n	1k6/2p4B/4b3/4N3/2K5/8/P7/8 w - - 0 1	1-0	7.41	6.80	27	4. Kb5	✓
06	H06	35997	6w	Mitrofanov	1967	9	5-4	4033.30	n	n	kb4Q1/P7/1PP5/K6q/8/8/8/4n3 w - - 0 1	1-0	12.65	12.65	21	6. Qg5	✓
07	H07	8860	6w	Zepler	1923	7	4-3	0400.21	Y	n	3k4/3P4/P2R4/8/6p1/r7/4K3/8 w - - 0 1	1-0	3.06	0.00	29	6. Ke1	✓
08	H08	67602	1w	Smyslov	2000	8	4-4	0000.33	n	Y	8/8/6p1/5p2/p1k2P2/8/P6P/4K3 w - - 0 1	=	-0.66	11.04	26	1. a3	✓
09	H09	39037	1w	Bazlov	1971	7	4-3	0405.00	Y	n	n7/2N4K/8/k7/7R/2r5/8/N7 w - - 0 1	1-0	6.88	6.02	20	1. Rh5+	✓
10	H10	67600	2w	Smyslov	2000	9	4-5	0130.23	n	Y	8/5k1p/4p3/2K4P/5P2/3b4/p7/6R1 w - - 0 1	=	0.00	0.62	22	2. h6	✓
11	H11	32098	1w	Fritz	1961	7	4-3	0310.21	Y	Y	B7/P7/P7/k6K/8/8/7p/r7 w - - 0 1	1-0	3.40	10.93	29	1. Bh1	✓
12	H12	67945	2w	v. d. Heijden & Beasley	2000	7	5-2	0013.30	Y	Y	8/8/8/8/8/K1P5/PBPn4/1k6 w - - 0 1	1-0	6.98	6.62	21	2. c4	✓
13	H13	57384	1w	Arestov	1989	12	6-6	0053.33	n	n	2K5/4p1B1/4k1P1/1b3pP1/p3n3/3P4/4B3/8 w - - 0 1	1-0	1.21	0.46	21	1. Bb2	×
14	H14	58049	5w	Neishtadt	1989	16	8-8	3213.45	n	n	7k/p4p1B/p4P2/P3qP2/7R/p1p2R2/P7/Kn6 w - - 0 1	=	0.00	4.93	22	1. Re3	✓
15	H15	69180	6w	Fabiano	2002	12	6-6	3001.44	n	Y	3N4/K3p3/4p3/k1P5/p1P3p1/P7/4P3/7q w - - 0 1	1-0	10.96	10.97	19	1. e3	✓
16	H16	69110	4w	Kralin	2002	13	6-7	0130.45	n	Y	5bRK/6p1/2p4k/2P1p1p1/4p1P1/4P3/2P5/8 w - - 0 1	1-0	0.00	0.00	34	1. c3	✓
17	V01	32412	1w	Zaitsev	1962	8	4-4	0107.11	n	n	5Kn1/4n3/8/5P2/8/3k4/3p4/1N3R2 w - - 0 1	1-0	0.69	0.21	24	1. Rd1	×
18	V02	71075	1w	Gurgenidze & Kalandadze	2004	12	7-5	0300.63	n	Y	8/pPPp1p2/3P1PPr/8/2P5/2k5/8/2K5 w - - 0 1	1-0	16.04	16.04	23	1. Kd1	✓
19	V03	64369	1w	Gurgenidze & Kalandadze	1997	13	7-6	0801.33	n	n	4R3/k6r/P3p3/K7/5P2/P5pp/8/RN1r4 w - - 0 1	=	0.00	10.40	18	1. Nc3	✓
20	V04	55531	1w	Gurgenidze	1987	12	7-5	0613.51	n	n	5Bk1/1PP5/5P2/7P/n6p/4r2P/r7/6K1 w - - 0 1	1-0	6.09	∞	24	1. Bb4	✓
21	V05	—	1w	Salai	2011	12	7-5	0040.53	n	Y	4K1k1/8/1p5p/1Pp3b1/8/1P3P2/P1B2P2/8 w - - 0 1	1-0	1.24	0.30	26	1. Kd7	×
22	V06	71074	1w	Benno	2004	10	7-3	3411.30	n	n	k1N5/1r1P4/8/KP6/7q/P7/6RB/8 w - - 0 1	1-0	15.79	0.00	22	1. Rg8	✓
23	V07	69009	1w	Visokosov	2002	14	7-7	0071.44	n	n	B2k4/3Pp3/4P1P1/5p1P/1pK5/7b/5p2/2b2N2 w - - 0 1	1-0	0.28	0.28	23	1. g7	×
24	V08	72886	1w	Kovalenko	2006	13	6-7	0000.56	n	Y	8/6Pp/p4K1p/P6p/8/P6p/Ppk4P/8 w - - 0 1	1-0	6.38	5.98	17	1. g8Q	✓
25	V09	72995	1w	Sochniev	2006	8	3-5	0034.12	n	n	2n5/1p2p2N/6P1/8/7k/2K5/8/6b1 w - - 0 1	1-0	2.42	3.59	23	1. g7	✓
26	V10	72386	1w	Csengeri	2005	11	6-5	0700.42	n	n	7K/1p1R4/1pP5/8/P7/P7/P1k3r1/3r4 w - - 0 1	1-0	3.29	4.08	23	1. c7	✓
27	V11	57418	1w	Pervakov & Sumbatyan	1989	22	12-10	3812.66	n	n	1N6/pq3P2/p2p4/P3k1B1/2RNp1r1/P1P1P3/2P1p1rp/4R2K w - - 0 1	1-0	9.11	9.11	17	1. Bf4+	✓
28	V12	38172	1w	Alekseev	1970	18	10-8	4825.23	n	n	1K1n3B/r6q/pR6/k1pN1R2/p1P4r/P1N4B/Q7/8 w - - 0 1	1-0	12.66	0.32	19	1. Qb1	✓
29	V13	66438	1w	Fiedler	1999	21	11-10	3111.78	n	n	8/1p2Pq1B/3p2p1/3N1kp1/1P1P4/2pP2P1/p1Pp1pP1/5R1K w - - 0 1	1-0	3.69	13.55	22	1. Bg8	✓
30	V14	73873	1w	Katsnelson & Sochniev	2007	8	3-5	0113.03	n	n	8/K7/1B1n4/8/1p5k/p4p2/6R1/8 w - - 0 1	=	-3.43	4.09	18	1. Bf2+	✓
31	V15	73067	1w	Ryabinin	2006	12	7-5	0300.63	n	Y	8/2P3P1/1pPp4/p7/4P1PP/7K/4k3/5r2 w - - 0 1	1-0	9.28	6.96	21	1. g5	✓
32	V16	75276	1w	Didukh & Masimov	2009	12	6-6	0311.34	n	n	8/5p2/5P2/P7/3B4/8/pp2KPp1/r1k1N3 w - - 0 1	1-0	0.00	18.84	17	1. Be3+	✓
33	V17	61165	1w	Pervakov & Selivanov	1993	12	5-7	4070.23	n	n	b2q4/1kbP2p1/1B6/1QP5/8/7p/p7/7K w - - 0 1	1-0	5.49	∞	16	1. c6+	✓
34	V18	59763	1w	Elkies	1991	14	8-6	3002.54	n	n	8/1p6/1p6/kPp2P1K/2P5/N1Pp4/q2P4/1N6 w - - 0 1	=	-0.17	0.00	23	1. f6	✓
35	H17	3477	5b	Saavedra & Barbier	1895	4	2-2	0300.10	Y	Y	8/2P5/8/8/8/3r4/2K5/k7 b - - 0 1	1-0	∞	0.00	28	5... Rf3	×
36	H18	69110	1w	Kralin	2002	16	8-8	4130.55	n	n	4R2K/3qb1p1/2p3kP/2P1p1p1/4p3/4P2P/2P5/6Q1 w - - 0 1	1-0	0.00	∞	24	1. Rg8	✓
37	H19	—	1w	Krug	2012	18	8-10	4062.46	n	n	Q3b3/1p2q3/1p6/bp5P/1p6/1p6/1P1N1PPp/4NK1k w - - 0 1	1-0	0.00	0.00	21	1. f3	×

Table 4. The studies cited by van der Heijden (2006, 2014) and Vlasák (2013).

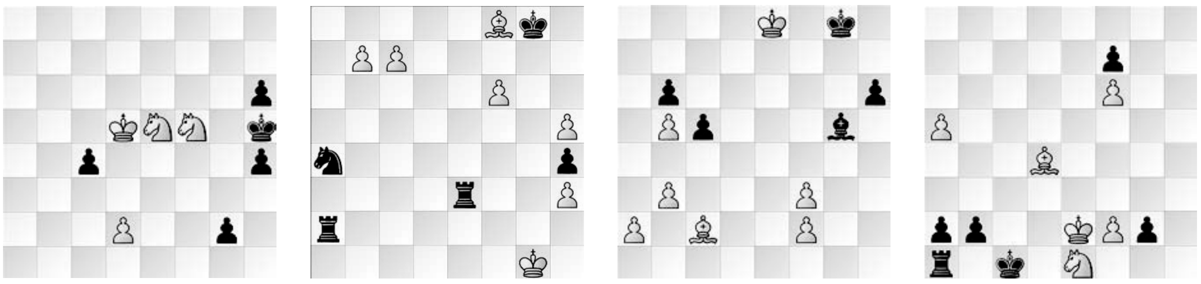


Figure 3. Four studies from Table 4: H01 (White to draw); V04, V05 and V16 (White to win).

Table 4 lists the 37 positions that van der Heijden and Vlasák chose. Authors are credited and serial numbers in the HHDBIV corpus are given.[7] The status of the positions vis-à-vis sub-8-man EGTs and FINALGEN is indicated. Some of these positions may contribute in part to a future, hypothetical test set *TS3*.

The author lightly tested 'DF14' DEEP FRITZ 14, i.e., Horváth's engine PANDIX (CPW, 2015) against the first, recommended move. The right-hand columns show its evaluation based on a 2-core, 3-minute run, the 'Δ' evaluation-difference to the next-best move, the move chosen and whether this agrees with the composition's author or not. But *DF14* is just one engine and it should be noted that different engines have different blind spots, can succeed or fail in finding 'best moves' and can certainly vary widely in their efficiency. The lesson is perhaps to stress-test studies with a battery of significantly different engines. Vlasák used HIARCS and HOUDINI, HIARCS usually but not always being slower. Of his studies, the ones which defeated an engine or occupied it for more than one hour are V04-05, V12 and V16-18. Others, such as V03/07/08/10/14, are in the second, 'useful engine-performance benchmark' class while the rest are quickly solved by engines.

The author's *DF14* found the recommended first move in all but 8 of the 37 positions, but note that one of these, *H17*, is included only to emphasise that the study composer presumes a fallible opponent who can be

[7] HHDBIV indices for the 16-study subset of TS1/2 are: TS1.01-04 (#7988, 7316, 1983, 18467), TS1.06-07 (1842, 20109), TS1.10-11 (3970, 4175), TS2.01 (15174), TS2.03 (15590), TS2.07-12 (51741, 66283, 18012, 31619, 66284, 7093).

tricked whereas a computer engine presumes its opponent is no more fallible than itself.[8] The full Saavedra study behind *H17* also emphasises that the first move is not necessarily the crux of the study. Therefore, finding the first move is no guarantee that the engine will reproduce the intended and presumably correct[9] solution, within a prescribed time or at all. CRAFTY did not reproduce the identical mainlines of all studies in TS1/2.

H01, now proved sound (Nunn, 2012) after years of debate, continues to defeat the best engines despite its short solution. *H02* is only 7-man but also defeats *DF14*. *H18* and *H19* also provide a significant challenge. Fortresses, perpetual check, zugzwangs (especially if engines' 'null move feature' cannot be switched off) and the 50-move draw rule continue to be factors which pose difficulties for chess engines.

My thanks to the authors cited, and particularly to Harold van der Heijden and Emil Vlasák for their test sets of compositions, and to future readers who contribute suggestions for test set TS3.

References

Bleicher, E. and Haworth, G. M^{c}C. (2010) 6-man chess and zugzwangs. Advances in Computer Games 12, Pamplona. LNCS 6048. pp. 123-135. Springer. ISSN 0302-9743 doi: 10.1007/978-3-642-12993-3_12.

CPW (2013). https://chessprogramming.wikispaces.com/Ronald+de+Man. Ronald de Man.

CPW (2015). https://chessprogramming.wikispaces.com/Gyula+Horváth. Gyula Horváth.

Fine, R. (1941). *Basic Chess Endings*. David McKay. Revised edition, ISBN 978-0812934939 (2003).

Haworth, G. M^{c}C. (2013a). Haworth's Law. *ICGA Journal*, Vol. 36, No. 4, p. 230.

Haworth, G. M^{c}C. (2013b). Chess Endgame News. *ICGA Journal*, Vol. 36, No. 4, pp. 228-9.

Haworth, G. M^{c}C. (2015). This note plus supporting datasets. http://centaur.reading.ac.uk/39946/.

Hyatt, R. (2015). https://cis.uab.edu/hyatt/crafty/. Hyatt's CRAFTY website.

MVL (2015). http://tb7.chessok.com/probe/250/3232. Lomonosov server of s8m DTM data.

Nalimov, E. V., Haworth, G. M^{c}C., and Heinz, E. A. (2000). Space-efficient Indexing of Chess Endgame Tables. *ICGA Journal*, Vol. 23, No. 3, pp. 148-162.

Newborn and Hyatt (2014). Computer Chess Endgame Play with Pawns: Then and Now. *ICGA Journal*, Vol. 37, No. 4, pp. 195-206.

Nunn, J. (2012). The Behting study is sound. Chessbase's Chess News: http://preview.tinyurl.com/ns57pkl.

Romero, P. P. (2012). FINALGEN: download, tutorial and examples. http://www.mtu-media.com/finalgen.

Tamplin, J. (2015). http://chess.jaet.org/endings/. Multi-metric endgame table service.

Van der Heijden, H. M. J. F. (2006). Schaaksoftware: eindspelstudietest. *EBUR* Vol. 18, No. 4, pp. 19-21.

Van der Heijden, H. M. J. F. (2010). http://www.hhdbiv.nl/. HHDBIV, ENDGAME STUDY DATABASE IV.

Van der Heijden, H.M.J.F. (2014). Onoplosbare (?) eindspelstudies [Insoluble (?) endgame studies]. *Computerschaak*. Vol. 33, No. 1, pp. 10-11.

Vlasák, E. (2013) Computer News: Looking far ahead. *EG*, Vol. 19, No. 193, pp. 219-226.

[8] This is a reflection on the concept of 'correctness'. For *H17*, engines choose **5. … Rf3** postponing mate, but the swindle **5. … Rd4** hopes for **6. c8Q??**, a mere draw. Black's most powerful piece is the white queen by the side of the board.

[9] Perhaps more than 30% of studies have a flaw in their solution. Vlasák, e.g., notes that *V12* is cooked by **1. Rff7**. HHDBIV notes these flaws where known and includes repaired versions of studies where available.

Research on the Computational Complexity of $n \times n$ Chinese Chess[1]

Qiang Gao[2] *and Xinhe Xu*[3]

Shenyang, 110004, China

ABSTRACT

Computational complexity of a computer game is an almost insurmountable obstacle in the process of looking for an ideal solution. Nevertheless researchers in the field of computer games aim to find such solutions, which may range from tractable to intractable. From the perspective of computational complexity, this article illustrates that $n \times n$ Chinese chess is intractable. The article starts to introduce the notion of an EXPTIME-complete problem of computational complexity and give as an example the G_3 game. Then an $n \times n$ Chinese chess position (one rook and two queens) is constructed, which consists of three essential components, viz. Boolean controller, switch, and the crossing of clause-channel. The G_3 game is simulated on the position, and it is proved that G_3 is reducible to the $n \times n$ Chinese chess position in polynomial time. From this result it is implied that $n \times n$ Chinese chess is EXPTIME-complete.

1. INTRODUCTION

Computer game programmers have the task to let computers think on how to make moves that win the game. Computer games are a challenging research field in artificial intelligence. The biggest hurdle of computer games lies in the computational complexity. By classifying the complexity of problems we may understand how hard it is to solve a game. Even if a problem is proved to be solvable with great difficulty (for example, it is NP-complete, PSPACE-complete or EXPTIME-complete), seasoned researchers frequently need to spend too much effort in looking for an efficient solving algorithm. Therefore, they are often looking for approximate solutions. Yet, many scholars are still performing investigations on the computational complexity of computer games; for instance, Chess (Fraenkel and Lichtenstein, 1981) and Checkers (Robson, 1984) were proved to be EXPTIME-complete. The two publications used the G_3 game, since this game had been proved to be complete in exponential time (Stockmeyer and Chandra, 1979). In both publications (on chess and checkers) the G_3 game was simulated on an $n \times n$ chessboard and it was proved that G_3 is reducible to generalized chess (checkers) in polynomial time. As a third instance, we mention Go. Go was proved to be PSPACE-hard (Lichtenstein, 1980). (Still, Go was also suspected to be EXPTIME-complete (Fraenkel and Lichtenstein, 1981)). Furthermore, Gobang (Reisch, 1980), Connect6 (Hsieh and Tsai, 2007) and Othello (Iwata, 1994) were proved to be PSPACE-complete. Later on, the Generalized Geography Game (Sipser, 2006) was simulated on these generalized chessboards separately. Amazons was proved to be PSPACE-complete (Hearn, 2005), and the Formula Game (Sipser, 2006) was adopted during the process of its proof. Chinese chess is a time-honored board game and it is quite similar to chess. The state complexity and game-tree complexity of 9×10 Chinese chess (Shi-Jim, Jr-Chang, and Tai-Ning, 2004) is contrasted to those of chess (Chinchalkar, 1996; Allis, 1994) in Table 1.

Game	State complexity	Game-tree complexity
Chinese chess	48	150
chess	46	123

Table 1: Contrasting complexity of Chinese chess and chess (data is given as powers of 10)

[1]Supported by National Natural Science Foundation of China, ID61370153

[2]College of Information Science and Engineering, Northeastern University, email:tommy_06@163.com

[3]College of Information Science and Engineering, Northeastern University, email:xuxinhe@mail.neu.edu.cn

Table 1 shows that the state complexities of Chinese chess and chess are very close, but the game-tree complexity of Chinese chess is much higher. It is thereby implied that Chinese chess is more difficult to solve. Now that chess was proved to be EXPTIME-complete, it is believed that Chinese chess is EXPTIME-complete too.

1.1 EXPTIME-complete

EXPTIME indicates that a set of decision problems can be solved by a deterministic Turing machine in $O(c^{p(n)})$ time (for some c and p being a polynomial where n is the length of the input). EXPTIME is not easier than other complexities such as P, NP, NP-complete, and space complexities (i.e., PSPACE and PSPACE-complete) (cf. Papadimitriou, 1994). A problem that is EXPTIME-complete is the hardest problem in EXPTIME. For the precise definition of problems that are EXPTIME-complete we use the definitions of other EXPTIME complete problems as given by Fraenkel and Lichtenstein (1981). They read as follows.

Definition 1: A decision problem B is EXPTIME-complete if it satisfies the following two conditions:

(1) B is in EXPTIME, and

(2) every A in EXPTIME is polynomial time reducible to B.

A decision problem is intractable if it cannot be decided by a polynomial time algorithm. For instance, the halting problem is a well-known problem that is indecidable (cf. Sipser, 2006). Moreover, the succinctly circuit problem is NEXP-time complete (cf. Papadimitriou, 1994). Below we discuss the G_3 game.

1.2 G_3 game

The ideas below are adopted from Fraenkel and Lichtenstein (1981, Section 2.2). The G_3 game is a two-player game. The players are R (Red) and B (Black). Every position in the G_3 game is a 4-tuple (τ, R-LOSE(*X*,*Y*), B-LOSE(*X*,*Y*), α), where τ denotes the player whose turn it is to play on the position, R-LOSE=$C_{11} \vee C_{12} \vee C_{13} \vee \cdots \vee C_{1p}$ and B-LOSE=$C_{21} \vee C_{22} \vee C_{23} \vee \cdots \vee C_{2q}$ are Boolean formulas in 12DNF, that is, each C_{1i} or C_{2j} is a conjunction of at most 12 literals ($1 \leq i \leq p$) ($1 \leq j \leq q$). Each literal is a variable in *X*(or *Y*); α is an assignment (0 or 1) of values to the set of variables $X \cup Y$. The players play alternately. Player R (or B) moves by changing the value of precisely one variable in *X* (or *Y*). In particular, passing is not permitted. Precisely, R can move from (R, R-LOSE, B-LOSE, α) to (B, R-LOSE, B-LOSE, α') iff α and α' differ in the assignment of exactly one variable in *X*, and R-LOSE is false under the assignment α. R(or B) loses if the formula R-LOSE(or B-LOSE) is true after some moves of player R(or B). Stockmeyer and Chandra (1979) proved that this game is EXPTIME-complete. The G_3 game will be played on a position of *n* × *n* Chinese chess.

The idea is to reduce an arbitrary instance of a G_3 game to a *n* × *n* chess position so that each variable in the G_3 game generates one rook and two queens. There are other pieces on the board that cannot be moved due to a deadlock to form channels that the queens can pass to attack the kings. The rook can only be in two different places which corresponds to the truth assignment of that variable. With the rook in either one place, one of the channels opens up for the queen to attend. The challenge in Chinese chess is how to group pieces together so that they are in a deadlock and form channels. Hence, the introduction of the cannon and the nine palaces is redundant for our case. The introduction of the piece exchange zone is also unnecessary. In Section 2, we outline the course of our proof and give the rules of *n* × *n* Chinese chess. In Section 3, we describe the construction of the special position. In Section 4, we provide conclusions.

2. OUTLINE OF PROOF AND RULES OF *N* × *N* CHINESE CHESS

This article aims to prove that the problem of testing whether B(R) wins or not in a given arbitrary position is EXPTIME-complete. Based on the definition of computational complexity's complete problem (Fraenkel and Lichtenstein, 1981), the steps of the proof are as follows:

1) Prove that the *n* × *n* Chinese chess is in EXPTIME;

2) construct a position and show that the G_3 game is reducible to the Chinese chess position;

3) prove that the transformation (i.e., construction of the given position) is completed in polynomial time.

2.1 Definition of $n \times n$ Chinese chess

We remark that only a finite number of different game positions may occur on a chessboard with a fixed size (e.g., 9×10 Chinese chess or 8×8 chess). This means that the number of possible positions is a constant. Current methods of research on computational complexity are asymptotic. Hence, they apply only to the rate of growth of the complexity as the problem size increases. Therefore the size of the problem must be generalized. This comes down to the fact that the size is arbitrary (cf. Sipser, 2006).

Definition 2: Let $n \times n$ Chinese chess be defined as follows. Given an arbitrary position of a generalized Chinese chess on an $n \times n$ chessboard, can Red (Black) win from that position? The rules of pieces of $n \times n$ Chinese chess are the same as those of 9×10 Chinese chess, which are listed below.

(i) The chariot moves and captures in any distance orthogonally, but may not jump over intervening pieces.

(ii) The horse moves and captures one square orthogonally and then one square diagonally away from its former position.

(iii) The cannon moves like chariot, any distance orthogonally without jumping, but can only capture by jumping a single piece along the path of attack.

(iv) The soldier moves and captures by advancing only one square before it crosses the river. Once the river is crossed, it may also move and capture one square horizontally.

(v) The elephant moves and captures exactly two squares diagonally. It cannot cross the river.

(vi) The general and advisor cannot leave the nine palaces and the advisor moves (or captures) one square diagonally.

(vii) The river is at the (*n/2*)th line on the $n \times n$ board (the river does not appear in the constructed position because that does not affect the construction and the proof in this article).

Moreover, the number of all kinds of pieces (except for the only one general of each side) will be generalized to m. Red (Black) wins the $n \times n$ Chinese chess iff Black (Red)'s general has been captured. The main difference between Chinese chess and chess are the nine palaces. The size of the nine palaces will be expanded because the chessboard is generalized (shown in Figure 1). □

2.2 The computational complexity of $n \times n$ Chinese chess

Theorem 1 $n \times n$ Chinese chess is in EXPTIME.

Proof. Assume that the computer needs a unit time when it handles a position of Chinese chess. Then the sum of possible positions of $n \times n$ Chinese chess is the upper limit of time to solve $n \times n$ Chinese chess. Because the force of Chinese chess is fourteen, the upper limit of possible positions sized by $n \times n$ is $15^{n \times n}$. Thus $n \times n$ Chinese chess is in EXPTIME.

3. CONSTRUCTION OF THE POSITION

In this section, a position (i.e., instance) of $n \times n$ Chinese chess will be given and an arbitrary G_3 game will be simulated in it. The main idea of the construction is that those and only those truth-assignments to the variables which win the G_3 game for R (B) lead the chariot of Red (Black) to win the $n \times n$ Chinese chess from the constructed position. Both sides use the chariot to capture or exchange pieces. R (B) wins the game if R (B) needs fewer moves to capture the general of the other side than B (R). This means that, R (B) of the G_3 game has a winning strategy if and only if R (B) of the $n \times n$ Chinese chess has a winning strategy in the given position. In 3.1 we describe the gadgets, and in 3.2 we give the winning strategy.

3.1 Illustration of gadgets

The features of the G_3 game will be considered. Changes are called gadgets. Obviously, the rules of Chinese chess cannot be violated during the construction. Some gadgets refer to construction ideas of chess (cf. Fraenkel and Lichtenstein, 1981) and checkers (cf. Robson, 1984). The Boolean controller and the Switch (3.1.2) are designed to realize the assignment of the values true or false to a variable of the G_3 game. The crossing of clause-channel and literal-channel (3.1.3) is designed to calculate the Boolean value of the formula R-LOSE(B-LOSE) of the G_3 game. Because the rules of Chinese chess differ from those of chess, the layout and pieces used in the gadgets are quite different. Moreover, according to the rules of Chinese chess there should be some moves that produce captures, therefore the Piece-exchanging zone and the Delay-area of the constructed position can be used to realize this process. (Not to complicate matters we leave the process out of the contribution). The most prominent difference between Chinese chess and other chess games lies in the nine palaces (see Figure 1). What is the general's activity? Each piece must pass the general to capture him, so a gadget of the position is designed to simulate it. However, it is not necessary for our problem at hand and therefore it is left out. Above all, the given position contains a Boolean controller (Figure 2), a Switch (see Figure 3) and a Crossing of the clause-channel and literal-channel (Figure 4). These three gadgets form the essentials for the position of $n \times n$ Chinese chess under investigation in this contribution.

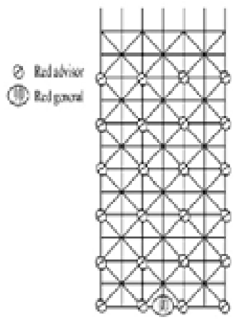


Figure 1: The initial nine palaces

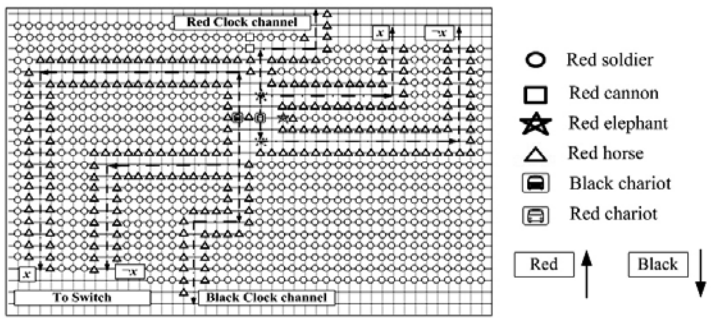


Figure 2: Red Boolean controller

3.1.1 Boolean controller

The Boolean controller (called BC for short) is designed to realize the assignment of the values true or false to a variable of the G_3 game. Figure 2 shows the structure of the red Boolean controller (RBC). RBC includes red soldiers (RS), red elephants (RE), a red chariot (RCT), red horses (RH), red cannons (RC), and a black chariot (BCT). It also contains (1) a literal (i.e., x and $\neg x$)-channel (corresponding to the literal that the clause of G_3 game contains, if BCT leaves the RBC via x-channel, the variable x is assigned to 1; if BCT leaves the RBC via $\neg x$-channel, the value of the variable x is 0) and (2) a red Clock channel. A black Boolean controller (BBC) is obtained from a RBC by an interchange $x \leftrightarrow y$, $\neg x \leftrightarrow \neg y$, red $\leftrightarrow$ black throughout, followed by a 180° rotation. There is one R (B) Boolean controller for each $x \in X (y \in Y)$. In normal play, R (B) can move one RE (BE), one RCT and one BCT in any R (B) Boolean controller. All other pieces are deadlocked, but they may capture the opponent's pieces passively according to the rule. The positioning of the deadlocked pieces forces the chariot to move through predefined channels in order to reach the opponent's general. At the beginning R must move RE to the north or south intersection with the dashed outline (RE's first move has only two possible choices), the two moves determine via which literal-channel BCT leaves RBC. The irrelevant moves in the BC are ignored in this contribution.

3.1.2 Switch

The Switch is designed to make sure that (1) only one opponent's chariot can reach the clause-channel and (2) the opponent's chariots at a clause-channel cannot go back to Boolean controller via literal-channels. Figure 3 shows the structure of the Switch. When a BCT comes via an RBC to an R Switch, it captures the RH on the longer longitudinal path and then proceeds down unperturbed to the C_{1i}-channel. If a BCT attempts to pass the R Switch in the opposite direction, aiming to reach the north corner of the longer longitudinal path, the RE at

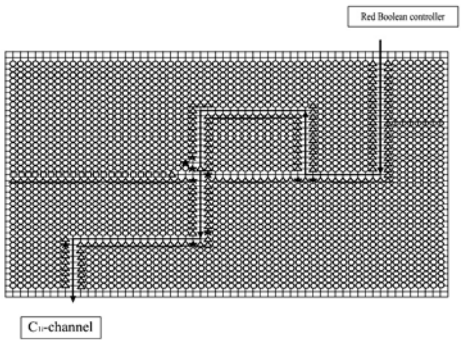


Figure 3: Red Switch

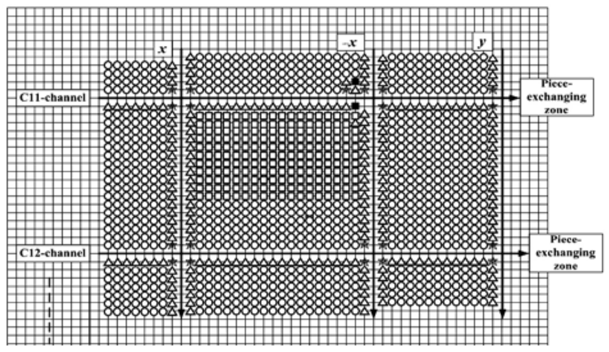


Figure 4: crossing of C_{1i}-channel and literal-channels

the southeast of the captured RH will go to the northwestern intersection and thus will open up a line of many RCs effectively covering the their eastern horizontal shorter path of the Switch, making the BCT impassable. Similarly, the BCTs at the C_{1i}-channels cannot go back to the Boolean controller.

3.1.3 C_{1i}-channels

The clause-channel corresponds to C_{1i} (the ith clause) or C_{2j} (the jth clause) of the Boolean formula R-LOSE (B-LOSE) in the G_3 game. According to the definition of the G_3 game, every clause contains some literals and the value of the conjunction which is the value of the clause. If the number of BCTs that stop unperturbed at the clause-channel equals to that of the literals contained in this clause, the value of this clause is true, i.e., the formula R-LOSE (B-LOSE) is true. Figure 4 shows the crossing structure of clause-channels and literal-channels.

As mentioned above, BCT gets to the C_{1i}-channel from the Switch via the literal-channel (e.g., x-channel). Only if $x(\neg x) \in C_{1i}$ holds, then BCT can stop unperturbed correspondingly at the intersection of the C_{1i}-channel and literal-channel, and finally get to the Piece-exchanging zone via C_{1i}-channel. Otherwise, the BCT cannot stop at the intersection because it will be captured by RH at the southwest of the intersection (shown in Figure 4), and then the number of the BCTs that reach the Piece-exchanging zone is not sufficient to capture the opponent's general. The irrelevant moves of this gadget will be ignored in this contribution.

3.2 Winning strategy

To make the global structure clear, at the south (north) of Figure 5 the overall nine palaces are divided into several channels which the BCTs must pass. The general appears at the southernmost (northernmost) row. In fact, the overall nine palaces consist of all these channels and one general. Figure 5 shows the overall structure of the construction. It forms a position of $n \times n$ Chinese chess on which the G_3 game can be simulated.

3.2.1 B's winning strategy

Assume that the RCT leaves an RBC via the Red Clock channel after the BCT has left the same RBC via the x ($\neg x$) channel. The BCT needs 3 moves to leave RBC (shown in Figure 2) and another 6 moves for reaching the C_{1i}-channel via the Switch (one move for reaching the Switch and one move for leaving the RBC refer to the same move), thus it spends $m = 9$ moves reaching the C_{1i}-channel. When there are p (p is the number of literals that the C_{1i} contains, and $1 \leq p \leq k \leq 12$) BCTs that stop unperturbed at the C_{1i}-channel, B will move each of them down the C_{1i}-channel towards the Piece-exchanging zone. As mentioned above, these BCTs have to exchange pieces with a line containing precisely $p - h$ (h is odd and $1 \leq h < p$) RCs at this zone, so B spends $p - h + 1$ moves in exchanging pieces. For the rest of the BCTs holds: each BCT needs another one move (they have to pass an inflection point) to get to the nine palaces. There are $2 \times (h-1)$ advisors and every two advisors can capture one BCT, so the BCTs have to exchange pieces with these advisors. At the nine palaces, the BCTs need one more move to capture another advisor after capturing two advisors, thus B needs $2 \times (h-1) + 2 \times (h-1) \div 2 - 1 = 3h - 4$ moves at the nine palaces. At last, the only one BCT checkmates the

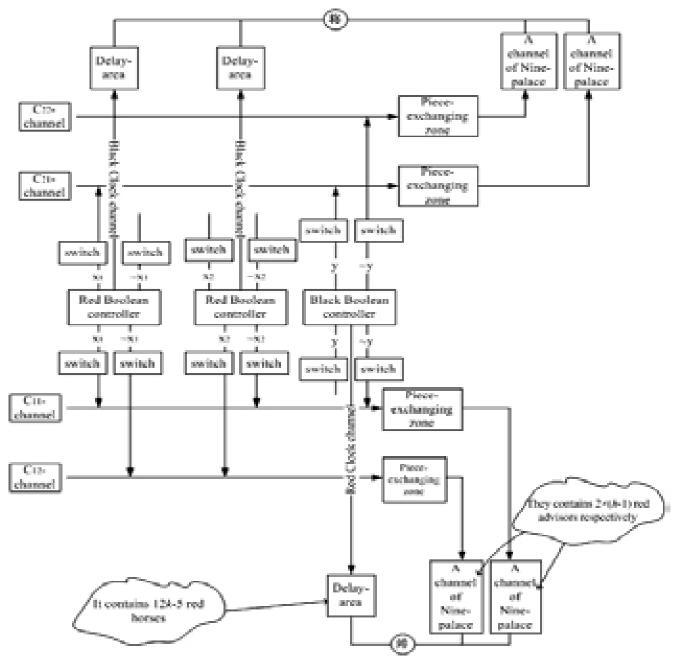


Figure 5: Global view of the construction for the case R-LOSE = $C_{11} \vee C_{12}, C_{11} = x_1 \wedge \neg x_2 \wedge \neg y, C_{12} = \neg x_1 \wedge x_2$; B-LOSE = $C_{21} \vee C_{22}, C_{21} = x_1 \wedge y, C_{22} = \neg y$

opponent's general after two moves. Above all, B spends $m \times p + p - h + 1 + h + 3h - 4 + 2 = (m+1) \times p + 3h - 1$ moves in playing the game, because $m = 9$, the totality is $10p + 3h - 1$.

Below, we examine in how many moves the BCT leaves the RBC. RCT needs 3 moves to leave the BC; the RE in the RBC makes one move according to the rule. At the Piece-exchanging zone, R spends $p - h$ moves in capturing the opponent's $p - h$ BCTs. At the nine palaces, RAs capture $h - 1$ BCTs, so R needs $h - 1$ moves in that area. At the Delay-area which consists of $12k - 5$ RHs, R needs $12k - 5$ moves to capture these RHs; at last R needs two moves to capture the opponent's general. Accordingly, the total moves R needs are $3 + 1 + p - h + h - 1 + 12k - 5 + 2 = 12k + p$, because $1 \le p \le k \le 12$ and $1 \le h \le p - 1$, R needs one more move than B to play the game. Above all, B wins with a margin of one more move.

3.2.2 R's winning strategy

B needs p BCTs that can stop unperturbed at one C_{1i}-channel and every BCT needs m moves to reach the channel, but it is possible that $C_{1i} = 0$. For example, $C_{11} = x_1 \wedge x_2 \wedge y_1$, the move that the BE in the BBC makes determines the value of the variable y_1. If the move makes BCT leave the same BBC via $\neg y_1$-channel, the value of y_1 is 0 (false), i.e., $C_{11} = 0$. Because $C_{11} = 0$ (that is to say, the value of R-LOSE is false), every of BCTs (assume that the number of the BCTs is z) that stops at the C_{11}-channel requires at least one move to reach another C_{1i}-channel, where C_{1i} = 1. If the number of literals that the C_{1i} contains is k (the k is the maximum number of literals that a clause contains and $1 \le k \le 12$) and the Piece-exchanging zone connected to the C_{1i} contains only one RC (i.e., $h = k - 1$), the total moves that B needs to play the game is $10k + 3(k-1) - 1 + z = 13k - 4 + z$, because the total moves of R is $12k + k = 13k$, thus R wins the game when $z > 4$.

4. CONCLUSIONS

Based on Section 3, we arrive at the following.

Lemma 1 G_3 game is reducible to $n \times n$ Chinese chess in polynomial time.

Proof. According to Section 3, an arbitrary instance of the G_3 game was simulated and solved on the given position. This means, that a given position of $n \times n$ Chinese chess can solve an arbitrary instance of the G_3 game. That is to say, there exits such reducibility that can convert an arbitrary G_3 game to the given position of $n \times n$ Chinese chess. According to definition 3 of reducibility (see below) we will make the next step in the proof.

Definition 3 (Sipser, 2006) Problem A is reducible to problem B. If there is a computable function f: where for every w, $w \in$ A $\Leftrightarrow f(w) \in$ B, the function f is called the reduction of A to B.

Based on the above definition, it follows that the G_3 game is reducible to $n \times n$ Chinese chess. Next it is necessary to estimate how long it takes to construct the position in Section 3. Assume that m is the sum of the variables contained in the 4-tuple and in the constructed position every variable corresponds to one BC, four literal-channels, four Switches, one Clock channel and four clause-channels (as shown in Figure 5, if certain literal is not in any clause, its channel is truncated prior to reaching the crossing of its channel and a clause-channel). As mentioned above, every channel-segment or shield around each channel is at the length of $13k$ (i.e., $O(k)$) which is a constant, and the sum of the variables is m, so the total size of the construction is $O(m \times k)$. Thus the construction can be completed in polynomial time. □

Theorem 2 $n \times n$ Chinese chess is EXPTIME-complete.

Proof. Firstly, according to Theorem 1, it follows that $n \times n$ Chinese chess is in EXPTIME. Secondly, because it has been proved that the G_3 game is EXPTIME-complete (Stockmeyer and Chandra, 1979) and based on Definition 1, it can be proved that all problems that are in EXPTIME may be reducible to the G_3 game in polynomial time. On the base of the transferable feature of reducibility (cf.Sipser, 2006) and Lemma 1, it can be proved that all problems that are in EXPTIME may be reducible to $n \times n$ Chinese chess.

From the above two points, it proves that $n \times n$ Chinese chess is EXPTIME-complete. □

5. REFERENCES

Allis, L. V. (1994). *Searching for Solutions in Games and Artificial Intellingence.* Ph.D. Thesis, Maastricht University, Maastricht, the Netherlands. ISBN 90–9007488–0.

Chinchalkar, S. (1996). An Upper Bound for the Number of Reachable Positions. *ICCA Journal*, Vol. 19, No. 3, pp. 181–183.

Fraenkel, A. S. and Lichtenstein, D. (1981). Computing a perfect strategy for n × n chess requires time exponential in n. *Journal of Combinatorial Theory*, Vol. 31, No. 2, Series A, pp. 199–214.

Hearn, R. A. (2005). Amazons is PSPACE-complete. arXiv:cs.CC/0502013v1.

Hsieh, M. Y. and Tsai, S.-C. (2007). On the fairness and complexity of generalized k-in-a-row games. *Theoretical Computer Science*, Vol. 385, pp. 88–100.

Iwata, S. and Kasai, T. (1994). The Othello game on an n × n board is PSPACE-complete. *Theoretical Computer Science*, Vol. 123, pp. 329–340.

Lichtenstein, D. and Sipser, M. (1980). Go is polynomial-space hard. *Journal of the ACM*, Vol. 27, pp. 393–401.

Papadimitriou, C. (1994). *Computational Complexity.* Addison-Wesley. ISBN 0–201–53082–1.

Reisch, S. (1980). Gobang ist PSPACE-vollständig (Gobang is PSPACE-complete). *Acta Informatica*, Vol. 13, No. 1, pp. 59–66.

Robson, J. (1984). N by N Checkers is EXPTIME complete. *SIAM Journal on Computing*, Vol. 13, pp. 252–267.

Shi-Jim, Y., Jr-Chang, C., and Tai-Ning, Y. (2004). Computer Chinese Chess. *ICGA Journal*, Vol. 27, No. 1, pp. 3–18.

Sipser, M. (2006). *Introduction to the Theory of Computation(Second Edition).* China Machine Press. ISBN 7–111–10840–x.

Stockmeyer, L. J. and Chandra, A. K. (1979). Provably difficult combinatorial games. *SIAM Journal on Computing*, Vol. 8, pp. 151–174.

INFORMATION FOR CONTRIBUTORS

Submission of material
Contributions to the Journal of up to 4000 words are welcome in any form. Short contributions (2000 - 3000 words) are preferred. Longer papers of striking originality and cogency may be considered, provided the authors consent to publication in instalments. Authors are urged to supply full references, including the names and location of publishers and ISBN or ISSN numbers where applicable.

While any form is welcome, efficiency of production imposes an editorial preference for some forms usual in electronic communications. The preferred forms are (in that order) Word (version 7.0 or convertible), LaTeX (information on the ICGA LaTeX style file can be found at: http://icga.leidenuniv.nl/?page_id=26).

Figures not conveniently accommodated in any of the above formats may be submitted either in PostScript format or as camera-ready hard copy. We urge contributors not to hesitate to submit their material in the form which most clearly represents the authors' wishes as to lay-out and presentation. More information is available at our homepage at http://www.icga.org.

Abstracting and indexing
Contributors may be interested to know that the *ICGA Journal* (and previously the *ICCA Journal* as of Vol. 10, No. 1 (1987)), is a source for Thomson Scientific for inclusion in the CompuMath Citation Index®, the Science Citation Index Expanded®, ISI Web of Science®, SciSearch®, and Personal Alert®. Paper abstracts and author information are accessible online through ISI Web of Science®.

The journal is also a source of information for R.R. Bowker for inclusion in the International Serials Database which is a source for Ulrich's Periodicals Directory™. Since 2006, the *ICGA Journal* is included in the abstract and citation database Scopus® by Elsevier B.V.

The Editorial Board
Broadening the scope of the Journal from chess to games has resulted in an extended Editorial Board, which now possesses specific knowledge on a variety of games. We mention Amazons, Backgammon, Bridge, Checkers, Chess, Chinese Chess, Chinese Dark Chess, Clobber, Dots and Boxes, Draughts, Ein Stein Würfelt Nicht, Go 9x9, Go 13x13, Go 19x19, Havannah, Hex, LOA, NoGo, Nonograms, Othello, Phantom Go, RoShamBo, Shogi, Sudoku and Surakarta. Contributors are encouraged to bring their submissions to the attention of the Editor-in-Chief via a game-knowledgeable member of the Editorial Board. Their names and email addresses are:

NEWS, INFORMATION, TOURNAMENTS, AND REPORTS

GAME OVER, ARIMAA?[1]

Andy Lewis[2]

Manningtree, Essex, UK

ABSTRACT

In an earlier article (Lewis, 2015) I introduced Kingpin readers to Arimaa, a recently invented and hitherto almost unknown chess variant. I argued that Arimaa has a comparable beauty and complexity to chess, but a much higher potential (given the lengthy history and extensive theory of chess) for creativity and originality.

This article recounts the extraordinary events of the Arimaa 2015 Computer Challenge. It offers a third-party, non-technical account of one of the key innovations that led to the triumph of SHARP & David Wu. It poses the question whether there is any reason left for Arimaa to exist, suggesting that even to survive it must rapidly emulate the technical innovations introduced within Chess in the past two decades. Finally, it reflects on how the community might regroup following Omar Syed's sad announcement (May 2015) that he will step down as leader of the Arimaa world.

1. THE RISE OF THE MACHINES

Supporters of the humanoid team for the 2015 Arimaa Computer Challenge (see Website 1) could have been forgiven for a sense of complacency. The Challenge match is between the world's strongest bot (computer program) and a team of 3 elite human players. The competition consists of a 3-game match between the bot and each of the humans, and the bot is required to win all three of the individual matches. A significant incentive is a $12,000 prize for the winning programmer to first accomplish the Challenge.

Although, this competition has been running annually since 2004, there have only been two occasions in which even one member of the human team has been defeated over 3 games. That was in 2010 and 2012, but even then silicon suffered a convincing defeat, each time by an overall 3-6 margin.

Could 2015 really be any different? Portents prior to the start of this year's Computer Challenge were ominous. In the Computer World Championship (March 2015) (see Website 2) bot SHARP flattened 9-0 the rival pretenders to the silicon crown. David Wu is the human intelligence behind SHARP and had previously taken the Computer title in 2011 and 2014, but by the more seemly margin of 7-2 both times.

Fritz Juhnke (twice world champion): tied 1-1 with SHARP.

Next up were the Screening Games (March-April 2015). This is an elimination contest between the top two bots from the Computer World Championship. Any human brave (or foolish) enough could take the field, and the bot which does best against the cannon fodder gets a shot against the elite humans. This was simple target practice for SHARP who demolished all-comers by 28-2. Its remaining rival, bot Z, was less accurate, with a meagre 18-10 score line against the same opposition.

The honorable exception was Karl 'Fritz' Juhnke (former world champion) who scored 50% over two games. His marathon 93 move victory over SHARP (see Website 3) is a masterful exemplar of anti-computer strategy.

[1] This contribution (now titled: Game Over, Arimaa?) is a slightly adapted version of Andy Lewis' article titled Arimaa: Game Over? published in Kingpin Chess Magazine, July 11, 2015. The ICGA Journal Editor is grateful for the permission by the author and the Editor-in-Chief of Kingpin Chess Magazine, Jonathan Manley, to include this version in our Journal.
[2] Andy Lewis is a former Executive Director with Nomura Securities, Japan. He holds a PhD in Philosophy (UCL 1988) and was 1980 British Under 21 Chess Champion.

The only other blow to SHARP was delivered by the rising star of Arimaa-land, John ‘deep_blue’ Smith of Germany, who demonstrated that the bot’s ability to assess material imbalance is still hardly flawless (see Website 3 for both games).

Its silicon rivals now bested, SHARP then took its seat in the 2015 Computer Challenge. The *Imperial Guards* were led by Mathew Brown (newly crowned 2015 world champion) and Jean Daligault (2014 world champion). The third team member was Lev Ruchka who, though never placed on the podium in a world championship, is an experienced and respected player within the Arimaa community. This is one of the strongest humanoid teams ever to have faced the bot onslaught. It could hardly have gone worse for the humans.

Jean Daligault (six times world champion): defeated 1-2 by SHARP.

Arimaa-land was dismayed to find the humanoids trailing 0-3 after Round 1, then staggered by the 0-6 deficit after Round 2. Only a late rally of human intelligence in Round 3 provided the sour consolation of a couple of elegant wins for Mathew Brown and Jean Daligault. But an overall 2-7 defeat left no room for doubt – or hope. On 18 April 2015, the Computer Challenge, which had stood for 11 years, was now lost.

The *Holy Grail* of Arimaa-land had been surrendered almost without a fight: what on earth happened here?

At the Computer Challenge Award Ceremony (9 May 2015), Jean dismissed any notion of a statistical fluke, and was fulsome in his praise of his computer opponent, describing it as “very, very strong, but in a few positions, it’s making big mistakes”. Commenting on whether SHARP had now better assimilated the lessons of his 2011 work (see Daligault, 2012) than any of his other (human) opponents, Jean responded that SHARP has actually *expanded* our conception of Arimaa techniques: to the point where now it is us humans who should start to learn from SHARP’s play. Gasp!

The explanation of why SHARP won the 2015 Computer Challenge is depressingly familiar to anyone who witnessed mankind’s toppling from the chess-throne in 1997: *face it, guys, it’s better than us!*

2. GENESIS OF A TERMINATOR

The bot SHARP and David Wu are no strangers to Arimaa Computer tournaments. SHARP first entered the lists in the 2008 Computer Championship, and achieved a respectable second place behind bot BOMB (programmed by David Fotland) – the dominant engine of the early Arimaa years.

Indeed, as reigning 2014 Computer World Champion, SHARP could have represented silicon honour in the Computer Challenge last year. However, bot ZILTOID (programmed by Riccardo Barreira) performed fractionally better in the Screening Games. ZILTOID lost the 2014 Challenge by the convincing margin of 2-7, so the implication is that even as late as 2014 SHARP would also have lost, by a similar or even greater score line. What did David Wu do between the 2014 and 2015 Computer Challenges that was so radically different? The simple and surprising answer seems to be: *not very much!*

Top Arimaa (and chess) engines use a move search technique known as *“iterative-deepening, depth-limited alpha-beta”* (or “alpha-beta” for short) (see Wu, 2011, p. 13). What this means roughly is that an engine looks at all variations to a given depth, discards the obviously rubbish ones, then does a deeper dive on the remainder. This process is iterated a number of times (assuming a forced end to the game is not found) until a time or depth limit is reached, when the residual node positions are compared using an evaluation function.

Now, the efficiency of any alpha-beta search can be dramatically improved, by looking at the more promising candidate moves first (see Wu, 2011, p.17). But the problem of how to arrive at a move selection ordering for an Arimaa engine is even more pressing than that for chess (or Go): since there are so many alternatives (approximately 16,064 for the average position (see Wu, 2011, p. 9). How could a computer ever rank such a vast number of alternative moves, without first doing some in-depth analysis?

In 2011, for his Harvard BA thesis in Computer Science, David defines an expert prediction methodology for Arimaa (see Wu, 2011, pp. 17-38) – based on one already successfully deployed for Go. David’s methodology specifies several hundred "features" of an Arimaa move, which might be ascertained by direct inspection, rather than in-depth analysis. Example types of features are material (whether a piece is won/lost), mobility (whether a piece is frozen/freed), and direct threats (whether a goal is threatened on the next move). A large sample of "expert" games (initially those played by humans rated above 1800) was then surveyed to determine to what extent these features are present in actual moves played by the experts.

This data can then be used to attempt to predict what move an expert is likely to make in a position outside the sample, based on the extent to which these features are present in the moves available from that position. David looked at various algorithms for generating predictions, and showed that accurate prediction of expert moves is indeed possible: for the most promising algorithm he examined in 2011; 90% of the time the experts' actual move fell within the top 5% ranked by feature (cf. Wu, 2011, pp. 37-38).

The power of such an approach is immediately obvious. It would combine the human expert's intuitive, immediate judgements about what candidate moves "make sense" with the computer's remorseless ability to grind-out zillions of variations.

Golly! It's that simple! But does this approach actually work? Yes it does. This move order function described in his thesis was the basis of the SHARP engine which won the 2011 Computer Championship, but, David tells me, has not been vastly altered in the 2015 version. The prediction engine has, however, been increasingly hardened by comparing the output of the selective move generator with actual moves chosen by the strongest human players – which would, of course, improve in quality over the 2010-15 period.

The overall architecture, therefore, appeared to have been established in 2011, and what David has been doing since then is basically refining and perfecting his initial concept. David's own metric describing the 2010-15 development process is that while numerous discrete improvements were effective, no single enhancement contributed more than 10% to the total improvement.

But, even if the initial conception was fundamentally sound, is it really credible that small improvements could, all of a sudden, result in bot SHARP just blowing away the opposition? That sounds just crazy!

Well, not at any rate, to the early eighteenth-century philosopher, Georg Hegel (1814) it looked as follows.
"The common view has it that when we speak of a growth or a destruction, we always imagine a gradual growth or disappearance. Yet we have seen cases (...) *which* (...) *involve* (...) *not only a transition from one proportion to another, but also a transition* (...) *an interruption of a gradual process, differing qualitatively from the* (...) *former state.''*

A series of incremental, *quantitative* improvements has morphed a dangerous, but (at the highest level) eminently beatable, SHARP of 2011-14 into the terrifying, *qualitatively* unrecognizable, Terminator of 2015.

David has now delivered his paper to the *ICGA Journal* describing how he achieved his success in the Computer Challenge. (That was a condition of his $12,000 prize.) I should probably shut-up now, and apologize to David for pre-empting his detailed & canonical explanation. Let the final word on this matter be his!

3. JUDGEMENT DAY

Does Arimaa even have a future? Will it survive bot SHARP's sudden and dramatic dethroning of humanity from the Arimaa summit? Received wisdom among Arimaa hard-core devotees is: yes, of course it will – just as chess survived, even flourished, after Kasparov's defeat by DEEP BLUE in 1997. It's a comforting line only for the faithful.

Chess had a rich culture, history and literature prior to its computerization. The number of chess-players has been variously estimated; however a hard figure to focus on is the 520,000 or so named on the 2015 FIDE rating list (see Website 4).

Of course, this metric does not do justice to the popularity of chess, and will not include all club, correspondence, internet & social players. Indeed, if you really want to ramp up the number, you might as well throw in even those who can just about set up the initial position without too many mistakes. This may even stretch the total to the jaw-dropping headline of 600 million players worldwide arrived at by the 2012 YouGov survey commissioned by AGON, and endorsed authoritatively on the FIDE website (see Website 5). But honestly, surely not even FIDE needs to descend to statistical manipulation to demonstrate the global cultural entrenchment of chess?

Arimaa's roots are somewhat shallower, a twelve year history, two published books (Juhnke, 2009; Daligault, 2012) (or three if you count the Wiki-book, see Website 6) – with perhaps around 500 active players worldwide. Why bother to learn a chess-like game with only an eclectic, scattered band of followers: when you can just play chess, with perhaps almost a million times as many opponents?

Arimaa has set its stall out as "the game of real intelligence" supposedly requiring a type of abstract reasoning ability available to humans, but not to computers. Bot SHARP has now dismantled this advantage: Arimaa is no longer anymore "computer-resistant" than chess.

And, with the benefit of hind-sight, perhaps the "computer-resistance" of Arimaa was always over-egged. Research on chess-playing software commenced in the 1950s and, with considerable commercial backing, eventually resulted in Kasparov's debacle against DEEP BLUE some four decades later. Research on Arimaa-playing software commenced around 2003, with zero commercial backing and a considerably smaller group of programmers, resulting only 12 years later in SHARP's victory.

Clearly, this data is susceptible of multiple interpretations, but could be used to argue that Arimaa was actually *less* computer-resistant than chess! Perhaps, it just took one exceptionally talented software engineer, working on the problem persistently over seven years, to land a solution.

Moreover, while detracting nothing from David Wu's triumph, let's be clear about what has *not* been achieved. What characteristics define human thought? Self-awareness, creativity, abstraction, the freedom to formulate rules, but also to modify or even break them? If it set out to create a *thinking machine*, the Chess-AI project failed laughably.

But has the Arimaa Computer Challenge, by this standard, done any better? No: SHARP has none of these qualities, or if it does, they are irrelevant to its success. Just like any other Arimaa (or chess) engine: SHARP is nothing more than a highly superior mechanized abacus. However, computer resistance was only one plank in the Arimaa USP: even if removed, isn't it still a game with excellent potential and interest? Certainly! But to achieve that potential, Arimaa-land needs to step up at least a couple of gears.

4. WHAT CHESS CAN TEACH ARIMAA

Arimaa must radically expand its franchise. It has nowhere near the critical mass it needs to ensure long-term survival. It needs to go mainstream. Face-tournaments with elite players are always a great way to generate publicity.

Want chess-GMs to participate in Arimaa tournaments? Easy: *show 'em the money!*

But this means normalizing Arimaa playing conventions, standards and conduct to that of the wider gaming community: something for which Arimaa-land demonstrates little appetite. And it badly needs some sponsorship deals, something which the paltry number of current followers – dedicated though they are – renders somewhat implausible. Even if Arimaa-land could survive purely confined to the on-line medium, the opportunities for improvement are self-evident. The front-end needs a refresh, and the creaking web-services on which the Arimaa game-room is hosted require a significant overhaul. These will require volunteer technical effort and/or funding.

Perhaps we will soon see bot SHARP or its counterpart available from the App Store and distributable to any smart phone. This means that to retain its credibility as a mind-sport, Arimaa, like chess, must introduce significant anti-cheating technology: a problem made still more pressing for Arimaa, if there is no face-to-face play to validate an individual's playing strength.

Bizarrely for a community so obviously rich in software expertise, Arimaa-land appears somewhat *under-computerized*. The Arimaa community has reliably provided an abundant harvest of new and improved bots each year: around 200 bots are available from the Game Room. But how about leveraging the engines to provide practical, targeted help for us patzers?

For example, an ancillary function might point an Arimaa position at a specified engine and generate an immediate set of candidate moves with evaluations. Perhaps the function could even automatically annotate an entire game. And wouldn't Arimaa players like to annotate and learn from their own and each other's games, not just with text comments, but with representations of the other likely paths the game might have taken? How about an *"ArimaaBase"* containing all games of note, and those between the top players annotated in depth, with a flexible GUI-based querying capacity?

Of course, the equivalent tools have been available to chess-players since the mid-1990s, and are now an essential part of any serious player's armoury. And this stuff is cheap and easy to use: anyone with a decent laptop and $200 to spend can have a +3000 ELO engine at their command and more shrink-wrapped chess knowledge than you can shake a stick at.

Is the chess-player simply expected to forgo these tools when he journeys into Arimaa-land? *Really?*

5. A CHAPTER CLOSES

Yet Arimaa *might* surmount all of these challenges. But not, perhaps, without someone with Omar Syed's vision, integrity and zeal at the helm.

Omar started work on designing an original game in 1997 following Kasparov's defeat by DEEP BLUE. That creative process led to invention of Arimaa around 2002. He developed the Arimaa website and Game Room and financed the hardware used to host it. He devised and stumped up the lion's share of the prize for the Computer Challenge, which commenced in 2004. He boot-strapped the Arimaa strategy. He led mankind in the early bot wars, only standing down when sufficiently strong humanoid players had emerged to relieve him.

Everything unique about Arimaa has Omar's imprint on it: from the *sui generis* time-control rules, to the unique tournament format for the World Championship. I have no specific evidence for it, but I suspect Omar is still the guy doing most of the production and user support on the game room servers and website. All this is done without pay or profit in the service of a vision. If only electronic distribution of these was possible, he would probably provide free beer and pizzas to the Game Room players!

In some ways, it would be helpful if Omar could be less modest and specifically credit his contributions to every aspect of the game on the Arimaa website. This will be useful when he decides to step down from his lead role in the game, so we understand what jobs now need to be filled. Because, sadly, this is now what has actually happened. In a further blow to the beleaguered Arimaa community, Captain Omar announced to a stunned audience at the Computer Challenge Award Ceremony that, after the best part of two decades, he is standing down from the wheel.

Arimaa without Omar is almost unimaginable: how shall Arimaa even survive, still less flourish, without Omar?

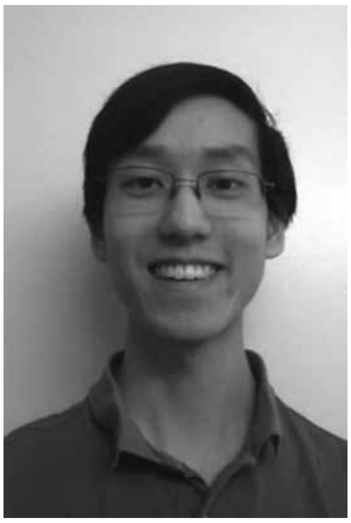
6. THE WHITE KNIGHT?

David Wu is clearly on board Arimaa for the long-haul. His association with the Arimaa community commenced in 2007, and although this decade he has spent most of his Arimaa-related time on bot development, he is still a strong player who can hold his own in the world's top 10.

Harvard educated, and now working in New York, David currently balances a mixture of software development and research within the financial industry. With the massive achievement of the Computer Challenge now on his CV, surely David will soon have Wall Street Heads of Algo Trading beating on his door, demanding his services. The new poster boy of Arimaa-land would certainly be well positioned to look for that much needed Corporate Sponsorship.

Is David the new White Knight? Commanding respect from geek and gamer alike, David is ideally placed to lead the faithful in the next phase of the Arimaa voyage.

In a gracious acceptance speech at the Computer Challenge Award ceremony, David announced that he plans to plough back his $12,000 award into the game, unveiling plans for a new and improved Arimaa Game Room, with other ideas to follow. There is so much that can be done for Arimaa, but the Game Room is the shop window: this is the right place to begin.


David Wu: dark destroyer or new champion of Arimaa-land?

Within only a couple of days of the announcement of the Game Room re-write, David had already assembled a crack team of six volunteers to work on this project. The collaborative culture of Arimaa-land, together with its depth of technical expertise, generate a sense of optimism that this project will succeed. The post-Omar epoch is already off to a promising start. One can only wish David and co every success!

7. REFERENCES

Daligault, J. (2012). *Arimaa Strategies and Tactics.* CreateSpace Independent Publishing Platform. ISBN 10-1452884-17X.

Hegel, G.W.F. (1814). *Science of Logic.* Republished (1969) Allen & Unwin, London England. Cited in Robert L. Carneiro (2000) in The transition from quantity to quality: A neglected causal mechanism in accounting for social evolution. Proc. Natl. Acad. Sci, USA, 10.1073/prias.240462397. Also published in PNAS, Vol. 97, No. 23, pp. 12926-12931.

Juhnke, F. (2009). *Beginning Arimaa: Chess Reborn Beyond Computer Comprehension*. Flying Camel Publications. ISBN 0-9824274-0-9.

Lewis, A. (2015). Arimaa, Computers and The Future of Chess. *Kingpin Chess Magazine*, February 2015.

Wu, D. (2011). *Move Ranking and Evaluation in the Game of Arimaa*. B.A. thesis, Harvard College, Cambridge, Mass.

WEBSITES

1) Arimaa Computer Challenge

2) Arimaa Website: follow link for "Championship" for human and computer world championship and "Challenge" for computer challenge and screening games

3) 2015 Arimaa Computer Challenge – Screening Games – bot SHARP vs F. Juhnke & DEEP BLUE vs bot SHARP.

4) FIDE Ratings List

5) Press Release "AGON releases new chess player statistics from YouGov"

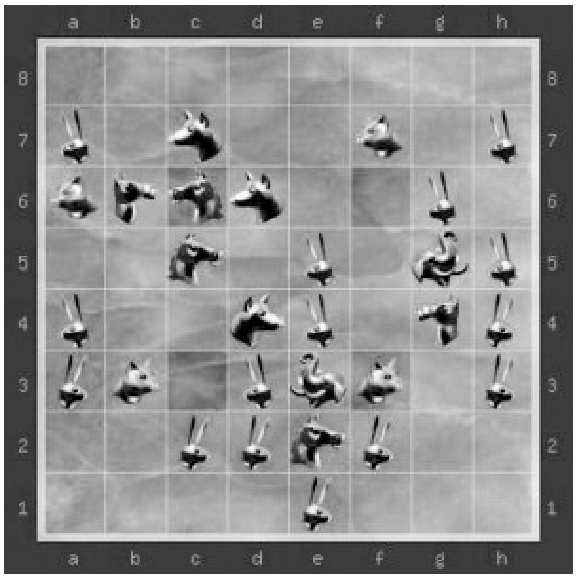
6) Arimaa Wiki-book

7) 2015 Arimaa World Championship R13 – M. Brown vs F. Juhnke

8. APPENDIX: 2015 WORLD CHAMPIONSHIP – THE CLINCHER

The 2015 World Championship was decided in an elimination battle between Karl Juhnke *(*aka *"Fritz")* World Champion in 2005 and 2008 and Mathew Brown (aka *"Browni"*) runner up in 2014 and currently the world's highest rated humanoid player.

The final game (see Website 7) reached Position 1 below after 40 moves.



Position 1: Mathew Brown – Fritz Juhnke
World Championship 2015
Gold to play 41g

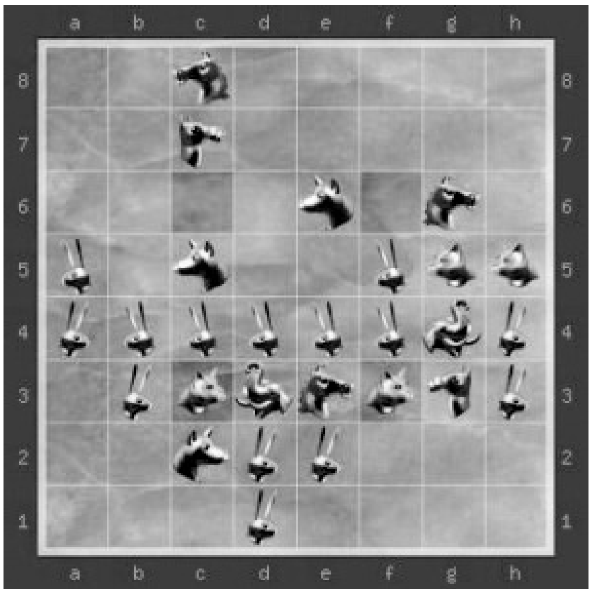
The material balance is slightly in Gold's favour with Silver having only a Dog and Rabbit to show for his Horse. However, this is the very least of Silver's problems.

Note first the over-ambitious Silver Elephant on e3, cut off from the rest of the Silver army, almost *smothered* (cf. Juhnke, 2009, pp. 95-105) by a combination of Gold and Silver pieces, and nearly out of moves. Secondly, Silver's best hope of relieving the beleaguered Elephant, the Camel on g4 is frozen by the Gold's Elephant on g5. If the Silver Elephant did make a bolt for freedom, by suiciding the Silver Rabbit on d3 into the c3 trap, then darting back via d3 and d4 to join the rest of the Silver forces, the g4 Silver Camel would rapidly get dumped in the f3 trap. Thus, the Silver Camel is said to be *hostaged* (cf. Juhnke, 2009, pp. 111-128) by the Gold Elephant.

But if Silver's Elephant can hardly move, and Gold's Elephant is required to keep Silver's Camel under lock and key, does this equate to an impasse? No. Because, thirdly, Gold's Camel on b6 is now the *strongest free piece* (cf. Juhnke, 2009, p. 97). Its main adversary, the Horse on c6, can do nothing except defend passively against the Camel's threats: active counter-attack would result in quick material loss. The accumulation of the above 3 factors, equates to an exceptionally difficult position for Silver.

Rather than try to force home his advantage, Browni choose a subtler and more poisonous method. 3-time repetition is not a draw in Arimaa (as it is in chess): rather the side that might seek to repeat for the third time is forced to *vary* the position. Gold has far more freedom in the above diagram than Silver, therefore by nimbly pirouetting his Dog and Cat around the c3 trap, Browni slowly and insidiously wrested a series of almost intangible concessions from Fritz.

This torture continue for a further 40 moves culminating in Position 2 below.



Position 2: Mathew Brown – Fritz Juhnke
World Championship 2015
Gold to play 80g

At first glance it might seem that not so much has changed about the position: the only material transition is the loss of a solitary Silver Rabbit. However, the experienced Arimaa player immediately notices the key differences. Firstly, the Silver Elephant (now on d3) has no legal moves whatsoever. Secondly, Gold Camel has now frozen the sole Silver Horse on c8 – indeed it is only the Silver Dog on c5 which prevents an immediate capture. Thirdly, all of the Silver Rabbits have been lured forward to the 5th rank or beyond, where they can play only a limited part in defending the Silver home traps (c6 and f6) against captures, also leaving Silver vulnerable to Rabbit advances to the goal-line.

The resulting position is now hopelessly lost for Silver, and there are perhaps any number of winning methods. Merciful at last, Browni dispatched Fritz in the most economical fashion, displacing the Silver Cats on g5 and h5, then marching his unopposed h-Rabbit to the 8th rank.

A cruel end to his hopes of regaining the World Championship, Fritz's role at the conclusion was reduced almost to that of a teaching-aid.

Browni's play is a text-book demonstration of a number of key Arimaa strategic themes uncovered over the past decade, many by Fritz himself. But least the younger generation of Arimaa players give themselves airs, they need to remind themselves that only an exceptional pupil could deliver such a stringent lesson in Arimaa fundamentals.

Congratulations to Mathew Brown, the 6th humanoid Arimaa World Champion, crowned 12 April 2015.

ACKNOWLEDGEMENTS
I would like to thank Fritz Juhnke, Jean Daligault, Mathew Brown, Omar Syed & David Wu for their original photos, reproduced in this and the 2015 Kingpin articles. I thank David also for his help (via email) in writing this article. I thank Matthew Craven and John Smith for their feedback (via Arimaa Forum) on the 2nd KingPin article. I thank and applaud Jon Manley, as the first editor of a popularist Chess Magazine to publish not only one, but two articles in a year on Arimaa.

Mathew Brown: 2015 Arimaa World Champion.

THE ICGA REFEREES 2014

The Editorial Board

The Editor-in-Chief and the Editorial Board would like to acknowledge the expert assistance of the following persons in refereeing submissions to the Journal during the year 2014. We hope to meet an equal willingness among the referees for the year to come.

Ingo Althöfer	Clyde P. Kruskal	Wen-Jie Tseng
Cameron Browne	Marc Lanctot	Peter van Emde Boas
David Fotland	Shun-Shii Lin	Jiao Wang
Dap Hartmann	Simon Lucas	Mark Watkins
Guy Haworth	Pim Nijssen	I-Chen Wu
Bob Hearn	Christian Posthoff	Xinhe Xu
Kai Himstedt	Salcedo Sanz Sancho	Jan van Zanten
Robert Hyatt	Jonathan Schaeffer	
Akihiro Kishimoto	Gerry Tesauro	

CALENDAR FOR EVENTS 2015

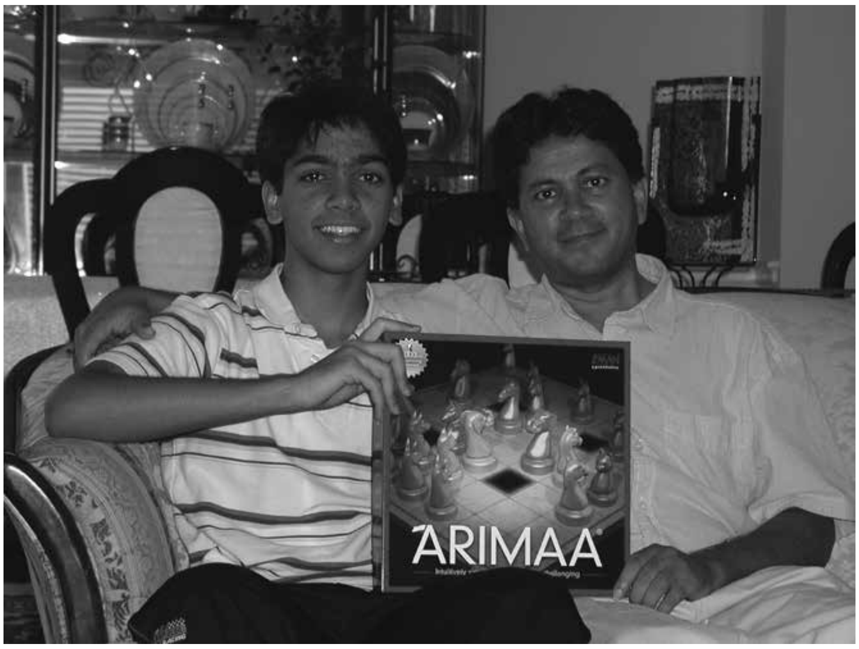
May 29 - 30, 2015
TCGA 2015 Workshop and Tournaments. More information: http://tcga2015.math.cycu.edu.tw

June 13, 2015
30th CSVN 'Gebruikers' Tournament, Leiden, the Netherlands. Information: http://www.computerschaak.nl/

June 29 – July 5, 2015
ICGA events in Leiden, the Netherlands. Information: www.icga.org. Email: info@icga.org.

July 25, 2015
European Go Congress and Computer Go Tournament, Liberec, Czech Republic. Information: http://www.egc2015.cz/ and http://computer-go.org/pipermail/computer-go/2014-Cecember/007007.html

August 10-15, 2015
19th World Computer-Bridge Championship to be held at the ACBL Chicago summer NABC. Information: http://computerbridge.co.



Omar Syed (right) with his son Amir after whom the game is named (A+AMIR spelt backwards) (see pp. 3-12).

HOW THE ICGA JOURNAL REACHES YOU

The *ICGA Journal* appears four times a year. In order to receive all issues of the 2015 *ICGA Journal*, you should subscribe as an ICGA member (the Triennial Meeting in Maastricht in July 2002 officially has changed the name ICCA into ICGA). The (renewal) fee is now € 40 for Europe, UK £ 25.--, or US $ 50.--. This annual fee has been agreed upon at the Triennial Meeting in Paderborn in June 1999. For subscriptions, renewals and orders of back issues, readers in **Asia, Australia, and Africa** should send their orders and payment to

ICGA
c/o Prof.dr. H. Iida
Research Unit for Computers and Games
Japan Advanced Institute of Science and Technology,
1-1, Asahidai, Nomi, Ishikawa 923-1292, Japan
Email: iida@jaist.ac.jp

Payment may be made as follows: Japanese Yens or US dollars, cash; as of January 2011 we do not accept cheques since the transition cost exceeds 23 Euro. For the readers (and others who wish to use our credit-card services, e.g., for books) payment by credit card (**Visa/Mastercard**) is possible. Send your credit-card details (Visa or Mastercard, full name on card, card number, expiry date, and address), state the amount to be paid and signed by the credit card holder to **Hiroyuki Iida, Email: iida@jaist.ac.jp, Business Fax: +81 (761) 51 1149.**

Alternatively: It is possible to pay by PayPal service via the Internet, too. All information is available at http://www.icga.org/ under ICGA Journal/Subscribe/Payment Options.

For **all other** subscriptions, renewals and orders of back issues, readers should send their orders and payment to

ICGA
c/o Prof. dr. H.J. van den Herik
Leiden University / Faculty of Science / LIACS
Leiden Centre of Data Science
P.O. Box 9512, 2300 RA Leiden / The Netherlands
Email: journal@icga.org.

Payment may be made as follows: Euros in cash, direct bank transfer from abroad: € 40. Credit-card payments should go to Hiroyuki Iida (see above). Though it is primitive, valid banknotes in UK pounds, US dollars, or Euros are still the most effective way of paying your dues neatly and safely. Also it is the cheapest way to transfer your dues free of all charges to the ICGA. Of course, postal regulations require that banknotes are sent by *registered* mail.

For the convenience of readers in countries allowing *direct transfer* to Dutch banks, payment may be made to the ING Bank (NEW BANKACCOUNT), the Netherlands, IBAN code: NL34INGB0003988921, BIC code: INGBNL2A, Tilburg, the Netherlands. While being acceptable, this method imposes charges on the ICGA and payment should be increased by Euro 7 to compensate for transfers between countries outside the European Community. Please include a correct statement of your mailing address. (We keep receiving payments without addresses!) Should your mailing label be erroneous or when changing residence, please return an amended label to Prof.dr. Iida or Prof.dr. Van den Herik as the case may require.

While stocks last, back issues of the *ICCA Journal* / *ICGA Journal* as from Vol. 6, No. 3 (August 1983) up to Vol. 37, No. 4 (2014) are still available. At this moment we offer back issues and some ICGA proceedings and books at a special price. For more information see http://www.icga.org. We can provide the pdf files from Vol. 1, No. 1 as well. Information at Johanna Hellemons: info@icga.org.

Institutional Membership is € 160 per year and includes six copies of each issue of the Journal. Library subscriptions are € 80 per year. A replacement copy for libraries is rated at € 20 each. For the American countries the amounts are US $ 190, US $ 95, and US $ 25 respectively. The currency differences are due to the differences in mailing costs.

Other enquiries should be directed to the Secretary/Treasurer, Prof.dr. Hiroyuki Iida, at the address above.

ADDRESSES OF AUTHORS

Omar Syed
Murphy, Texas / USA

David Fotland
San Jose, California / USA

Qiang Gao and Xinhe Xu
College of Information Science and Engineering,
Northeastern University, Shenyang, 11004 / China

Guy M^{c}C. Haworth
University of Reading, Berkshire / UK

Andrew Lewis
Manningtree, Essex / UK

David Wu
New York, NY / USA

The addresses of authors not mentioned above will be found elsewhere in this issue.

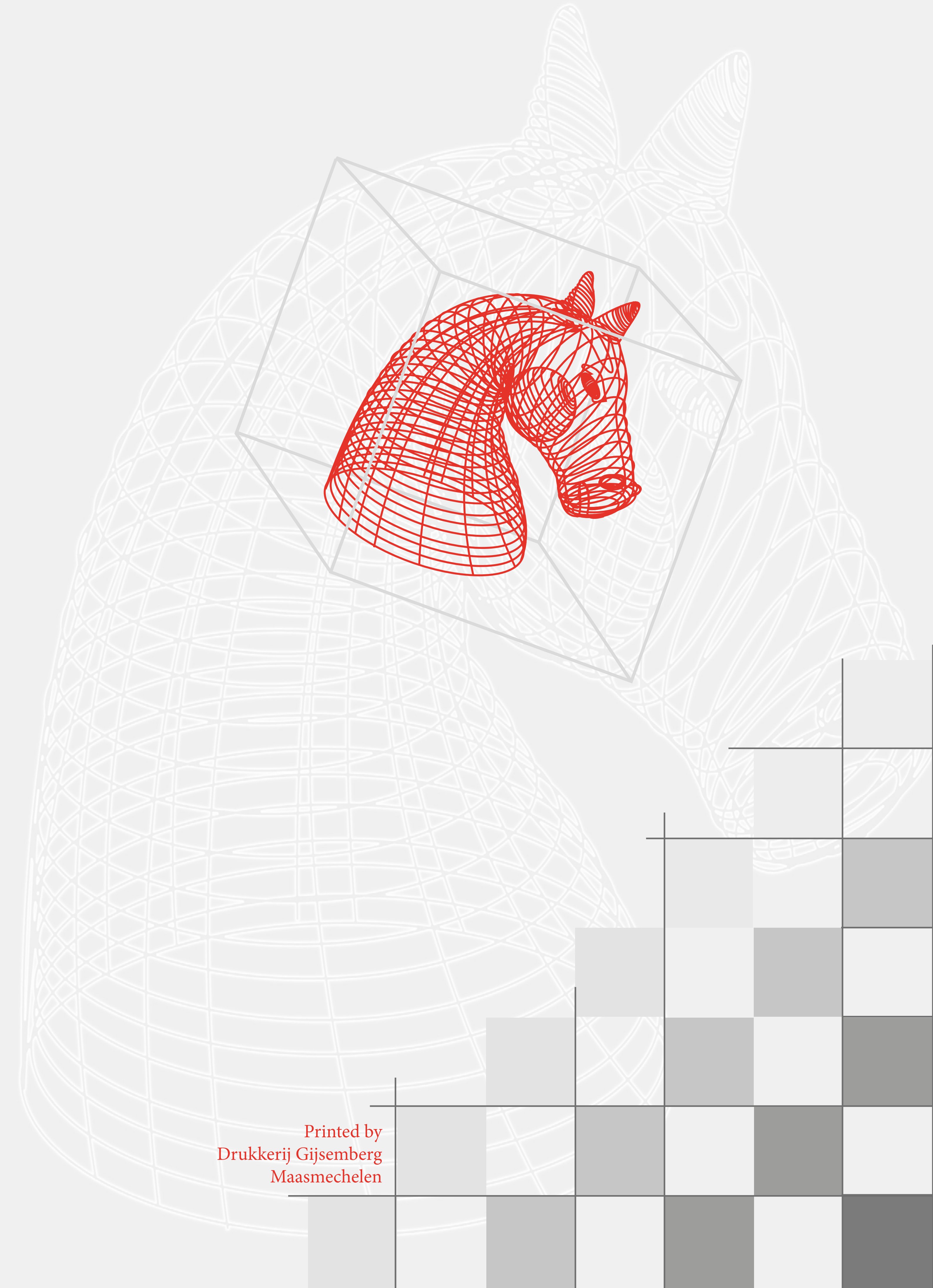
The deadline for copy for the next issue is June 1, 2015.

Please submit specific contributions to the appropriate Section Editors (see p. 54) and general contributions and correspondence to:

Prof. dr. H.J. van den Herik
Leiden University
LCDS

Correspondence address:	P.O. Box 9512 2300 RA Leiden / The Netherlands
Residential address:	Niels Bohrweg 1, Building (Snellius), Room 164 2333 CA Leiden / The Netherlands
Telephone:	+31 71 527 7045
Email:	journal@icga.org

Please address all material to be reviewed, digested or abstracted, to Dap Hartmann, Delft University of Technology, Faculty of Technology, Policy and Management, P.O. Box 5015, 2600 GA Delft, The Netherlands. Email: L.Hartmann@tudelft.nl.



Printed by
Drukkerij Gijsemberg
Maasmechelen