

---

# Graphbasierte Chunkssuche in Schachdatenbanken

---

Bachelor-Thesis von Igor Cherepanov

Tag der Einreichung:

1. Gutachten: Prof. Dr. Fürnkranz
2. Gutachten:



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Fachbereich Informatik  
Knowledge Engineering Group

## Graphbasierte Chunkssuche in Schachdatenbanken

Vorgelegte Bachelor-Thesis von Igor Cherepanov

1. Gutachten: Prof. Dr. Fürnkranz
2. Gutachten:

Tag der Einreichung:

---

# Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 19. November 2015

---

(Igor Cherepanov)

---

---

## Zusammenfassung

---

Mehrere in Relation zueinander stehende Schachfiguren bilden einen Cluster, der auch als Chunk bezeichnet wird. Das Thema der vorliegenden Bachelorarbeit beschäftigt sich mit der Suche nach häufigsten Chunks in Schachdatenbanken. Die Relationen, auf denen ein Chunk basiert, sind unter anderem das Decken der Figuren gleicher Farben sowie der Angriff zwischen Figuren unterschiedlicher Farben. Die Stellungen der Schachpartien werden als Graphen dargestellt. Das Extrahieren der Informationen aus der Schachstellung erfolgt mittels Bitboards. Kanten stellen die Relationen und Knoten die Schachfiguren bei den erstellten Graphen dar. Die Position einer Figur wird nicht berücksichtigt. Der Grund dafür ist, dass ein Muster, unabhängig von der Position, überall auf dem Schachfeld auftreten kann.

---

---

---

## Inhaltsverzeichnis

---

<b>1</b>	<b>Einleitung</b>	<b>4</b>
1.1	Motivation und Zielsetzung . . . . .	4
1.2	Struktur der Arbeit . . . . .	4
<b>2</b>	<b>Hintergrund</b>	<b>5</b>
2.1	Schach . . . . .	5
2.2	Chunks . . . . .	5
2.3	Graph . . . . .	6
2.4	Bitboards . . . . .	6
2.5	Schachnotationen . . . . .	10
2.6	Frequent Pattern in Graphen . . . . .	12
2.7	gSpan . . . . .	12
<b>3</b>	<b>Konzept</b>	<b>18</b>
<b>4</b>	<b>Implementierung</b>	<b>19</b>
4.1	Schachpartien . . . . .	19
4.2	Extraktion der Schachstellungen . . . . .	19
4.3	Generierung der Graphen . . . . .	19
4.4	Ausgabe der Graphen für das gSpan-Programm . . . . .	22
<b>5</b>	<b>Experimente</b>	<b>23</b>
5.1	Experiment 1 . . . . .	23
5.2	Experiment 2 . . . . .	28
5.3	Experiment 3 . . . . .	29
5.4	Experiment 4 . . . . .	30
5.5	Auswertung . . . . .	31
<b>6</b>	<b>Zusammenfassung und Ausblick</b>	<b>32</b>
	<b>Literaturverzeichnis</b>	<b>33</b>

---

## 1 Einleitung

---

### 1.1 Motivation und Zielsetzung

---

Gedächtnismeister, Großmeister im Schach oder Experten eines Gebiets haben schon immer die Menschen mit ihrem Talent fasziniert. Deren außerordentliche Leistung war auch für die Wissenschaft als Forschungsobjekt ziemlich interessant. Nachdem Psychologen mehrere Experimente mit den Genies eines Gebiets durchgeführt hatten, waren sie der Meinung, dass eine Begabung nicht angeboren ist, sondern durch systematisches Training erzielt werden kann. Danach folgten Experimente, bei denen vollkommene Anfänger im bestimmten Bereich trainiert wurden. Sie erreichten ein Weltklassenniveau. Die bekanntesten Experimentkinder eines Psychologen sind Zsuzsa Polgár und ihre zwei Schwestern, die Spitzenschachspielerinnen sind. Ihr Vater, László Polgár, behauptete ebenfalls, dass die Praxis der signifikante Faktor ist und wie das Wissen näher gebracht wird und nicht die angeborene Intelligenz, die einen Genie ausmachen. Heute gibt es Thesen, die aussagen, dass um die 10000 Trainingsstunden [1] notwendig sind um ein Niveau eines Experten zu erreichen, egal um welche Disziplin es geht. Die Studien zeigten, dass der Erfolg dieser Meister in den extrem großen Bibliotheken von Mustern in ihren Langzeitgedächtnissen liegt. Schachgroßmeister haben um die 100000 Chunks, behauptet der Psychologe Herbert A. Simon. Die Kognition und die Interpretation der erkannten Information ist ein wichtiges Element in verschiedenen Hochleistungsdomänen, insbesondere mit strategischen Aufgaben. Im Februar 1996 fand in Philadelphia ein Match zwischen Kasparov und Deep Blue<sup>1</sup> statt. Kasparov gewann drei Partien, zwei Spiele sind als Remis ausgegangen und ein Spiel verlor er. Das ist ein weiteres Beispiel, in dem die über 20 Jahre dedizierter Schacherfahrungen einen Hochleistungsrechner besiegen konnten.

Die gefundenen Muster können zum Lernerfolg im Schach beitragen. Ein weiterer Grund des Aufsuchens von Chunks sind Heuristiken. Aus den Chunks können neue Heuristiken aufgestellt werden, die dabei helfen bestimmte Abzweige im Suchbaum nicht zu berücksichtigen, da der Spielbaum vom Schach enorm groß ist.

Ziel dieser Arbeit ist das graphbasierte Aufsuchen der häufigsten Muster in Schachpartien.

---

### 1.2 Struktur der Arbeit

---

In Kapitel 2 werden zuerst die Grundlagen für das Verständnis der vorliegenden Arbeit gelegt. Dazu gehören die wichtigsten Punkten wie:

- Die Definition von Chunks und ein kurzer Einblick in die Experimente der Psychologen
- Die wichtigsten Punkte aus der Graphentheorie
- Die Bitboards, durch die Informationen aus den Schachpartien extrahiert werden
- Kurzer Einblick in die Schachnotationen, aufgrund der Formatierung der Schachspiele
- Frequent Pattern in Graphen und die Erklärung des Algorithmus, der in dieser Arbeit verwendet wird

Kapitel 3 erklärt das Konzept des Vorgehens

Kapitel 4 befasst sich mit der Implementierung

Kapitel 5 stellt die Experimente vor

---

<sup>1</sup> ein von IBM entwickelter Schachcomputer

---

## 2 Hintergrund

---

### 2.1 Schach

---

Schach ist ein strategisches Brettspiel, bei dem zwei Personen auf einem Spielfeld, das in 64 quadratische Felder unterteilt ist, gegeneinander spielen. Die genauen Regeln des Spiels werden hier nicht erklärt, jedoch befindet sich ein Verweis auf die offiziellen Regeln des Spiels auf der Homepage der FIDE (Fédération Internationale des Échecs) [2].

Mit der Frage „wie viele Partien im Schach möglich sind“, wird die hohe Komplexität des Schachspiels ersichtlich. Hier werden die ersten fundierten Schätzungen der Mathematiker genannt, die sich mit dieser Frage befasst haben. Nenad Petrovic kam im Jahr 1948 auf eine unvorstellbare Zahl von  $10^{18900}$  Partien, der britische Mathematiker John E. Littlewood errechnete im Jahr 1953 einen Wert von  $10 * 10^{70.5}$  [3].

---

### 2.2 Chunks

---

Das Wort „Chunks“ ist von dem Psychologen George Armitage Miller eingeführt worden. Als er das Gedächtnis eines Menschen untersuchte, entdeckte er, dass unsere Fähigkeit bestimmte Informationen zu verarbeiten und Entscheidungen zu treffen beschränkt ist. Bei der Untersuchung der Kurzzeitgedächtnisspanne zeigte das Experiment, dass die menschliche Kurzzeitgedächtnisspanne circa  $7 \pm 2$  Informationseinheiten behalten kann [4]. Die Kurzzeitgedächtnisspanne ist begrenzt, jedoch kann der Informationsgehalt eines Chunks (Bedeutungseinheit) variabel sein. Ein Versuch, der über 20 Monate ging und ungefähr 230 Stunden dauerte, zeigte, dass in der Kurzzeitgedächtnisspanne durch Training von 7 bis zu 230 zufälliger Informationen gespeichert werden können [5]. Das Experiment führte zu der Entdeckung, dass Informationen zu Clustern verknüpft werden können. Das Zusammenspiel der Chunks und der Informationseinheiten lässt sich am besten über Sprache erklären. Wenn die folgenden 48 Buchstaben „Neredeyseherzamankigibigünümükütüphanedegeçirdim“ behalten werden sollen, wird eine Person sicherlich nicht alle Buchstaben in der richtigen Reihenfolge merken, es sei denn die Person spricht diese Sprache. Der äquivalente Satz im Deutschen ist „FastwieimmerverbrachteichmeinenTaginderBibliothek“. Jedes Wort ist eine Informationseinheit und diese Einheit kann unterschiedlich groß sein. Der Mensch nutzt das, was er bereits weiß, um neue Informationen abzuspeichern. Chunks sind im Alltag ebenfalls anzutreffen. Telefon- und Kreditkartennummern werden, um sie sich besser merken zu können, in Blöcke bis zu 4 Zahlen mit einem Leerzeichen getrennt „Tel.: +49 - 030 123 123 12“ , „KNr.: 2341 1234 3423 9803“.

### Chunks und Schach

Der erste Psychologe, der extensive Experimente mit Schachspielern unterschiedlicher Spielstärke durchgeführt hatte, war der Adriaan de Groot. Er untersuchte kognitive Anforderungen und Überlegungsprozesse für die besten Schachzüge der Schachspieler. Er zeigte den Spielern unterschiedliche Schachstellungen und führte ein Interview mit ihnen und stellte dabei fest, dass es keine Unterschiede in deren Gedankenprozessen gibt. Die Tiefe der Zugberechnungen, die Heuristiken und die in Betracht genommenen Zugmöglichkeiten bei der Suche waren identisch, sowie die benötigte Zeit, jedoch tätigte der stärkere Spieler immer den besseren Zug als der Spieler mit einer mittleren Stärke. Es zeigte sich auch, dass die guten Schachspieler den besten Zug in der Regel meistens auf dem ersten Blick erkannten.

Die Großmeister und die Meister erwiesen sehr gute Resultate in Memorieren einer Stellung einer unbekanntem Partie, die sie nur fünf Sekunden anschauen konnten. Sie konnten die Stellung mit einer Genauigkeit von 95% nachbauen und lagen nur bei den Figuren falsch, die auf die Stellung fast keinen Einfluss hatten. Das Resultat der Spieler, die dem Meistertitel nahe waren, war ca. 72%, die Amateure kamen auf 50% und die Anfänger auf 33%.

Dieser Test wurde von H. A. Simon und W. G. Chase erweitert [6]. Unter den gleichen Bedingungen wurde den Spielern ein Brett mit zufällig platzierten Schachfiguren gezeigt und dabei sind überraschende Resultate entstanden. Bei diesem Experiment waren die Großmeister nicht besser als die Anfänger. Alle Spieler, von Großmeister bis Anfänger konnten im Durchschnitt die gleiche Anzahl der Figuren richtig aufstellen. Das bedeutet, dass die Schachspieler im Langzeitgedächtnis über eine riesige Bibliothek von Schachmustern verfügen, die sie im Experiment benutzt haben um die Informationen

---

auf dem Feld sinnvoll zu clustern und diese im Kurzzeitgedächtnis abzuspeichern. Auch der Blickverlauf der Schachspieler wurde analysiert. Es hat sich herausgestellt, dass der Blick schnell über großen Distanzen verläuft und sich auf eine geringere Anzahl der unterschiedlichen Stellen fokussiert, jedoch auf die relevanten für den nächsten Zug. Diese Experimente haben gezeigt, dass das Finden des nächsten Zuges durch die Wahrnehmung einer sinnvollen Aufstellung einer Schachpartie erfolgt und nicht durch tiefe Kalkulation. Die Weltklasseschachspieler haben jahrelange Erfahrung und das macht ihr Talent aus.

## 2.3 Graph

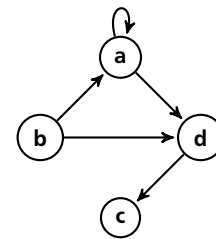
Ein Graph  $G := (V, E)$  hat eine endliche Menge von Knoten  $V$  und eine Relation  $E \subseteq V \times V$ , die eine Menge von Kanten darstellt.

$$V := \{a, b, c, d\}$$

$$E := \{(b, a), (a, a), (a, d), (d, c), (b, d)\}$$

Eine Kante  $(b, a) \in E$  stellt eine gerichtete Kante von  $b$  zu  $a$  dar.

Ist die Relation  $E$  symmetrisch, so wird der Graph als ungerichteter Graph bezeichnet.



### Labeling

In der Graphentheorie repräsentiert Labeling (*Markierung*) die Zuweisung der Labels zu den Kanten und den Knoten.<sup>2</sup>

Ein labeled Graph kann durch ein 4-Tupel repräsentiert werden  $G := (V, E, L, l)$ . Dabei ist  $V$  die Menge der Knoten und  $E \subseteq V \times V$  ist die Relation, die die Menge der Kanten repräsentiert.

$V$  ist die Menge der Labels und  $l$  ist eine Funktion  $l : (V \cup E) \rightarrow L$ .

Beispiel eines ungerichteten labeled Graphen  $G := (V, E, L, l)$

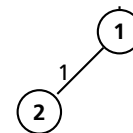
$$V := \{1, 2\}$$

$$E := \{(1, 2), (2, 1)\}$$

$$L := \mathbb{N}$$

$$l \in (V \cup E) \rightarrow L$$

$$l(1) = 1, l(2) = 2, l((1, 2)) = 1, l((2, 1)) = 1$$



Labeling kann unterschiedliche Relationen zwischen den Daten repräsentieren.

### Isomorphie der Graphen

Zwei ungerichtete Graphen  $G := (V, E)$  und  $G' := (V', E')$  sind isomorph, wenn es eine bijektive Funktion  $f : V \rightarrow V'$  gibt, sodass  $\forall u \in V : l_G(u) = l_{G'}(f(u))$  und  $\forall (u, v) \in E : (f(u), f(v)) \in E'$  und  $l_G(u, v) = l_{G'}(f(u), f(v))$  gilt. Diese Funktion wird als Isomorphismus zwischen den Graphen  $G$  und  $G'$  bezeichnet.

## 2.4 Bitboards

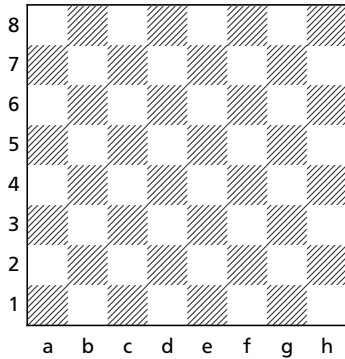
Für die menschliche Vorstellung ist die beste Datenstruktur eines Schachfelds ein zweidimensionales Array, denn es bildet visuell am nächsten das Schachfeld ab, jedoch sind bei einem zweidimensionalen Array die Operationen auf dem Schachfeld nicht ressourceneffizient. Als eine bessere Datenstruktur bieten sich die sogenannten Bitboards an [19].

<sup>2</sup> Die Begriffe wie Label, labeled sowie Labeling werden weiterhin aus dem Englischen unübersetzt verwendet.

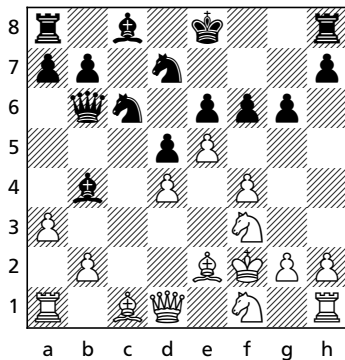


Ein Bitboard ist eine 64 Bit lange binäre Zahl. Es gibt eine bijektive Korrespondenz zwischen den Bits eines Bitboards und dem Schachfeld.

Eine mögliche Zuordnung zwischen einem Schachfeld und einem Bitboard ist in der unteren Abbildung dargestellt.



56	57	58	59	60	61	62	63
48	49	50	51	52	53	54	55
40	41	42	43	44	45	46	47
32	33	34	35	36	37	38	39
24	25	26	27	28	29	30	31
16	17	18	19	20	21	22	23
8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7



Ein Bitboard ist eine binäre Zahl. Jede Position dieser Zahl kann nur eine 1 und 0 annehmen. Das bedeutet, dass ein Bitboard nicht viele Informationen übertragen kann, weswegen es einen Bedarf gibt mehrere Bitboards anzulegen um alle Informationen zu sichern. Für jede Art der Figuren mit jeweils einer Farbe wird ein Bitboard angelegt, in dem die 1 in einer Position des Bitboards die Information trägt auf welchem Feld sich die Figur sich befindet.

0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1
0	0	0	0	1	1	1	0
0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	1	0	1	0	0
1	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1
0	0	0	0	0	0	0	0

Die Bitboards für schwarze und weiße Bauern der oben angegebenen Spielstellung. Es gibt weitere zehn Bitboards für jede Figurart in einer Farbe (es werden hier nicht alle Bitboards gezeigt). Jede Änderung auf dem Spielbrett impliziert eine Änderung der Bitboards.

Durch diese Datenstruktur wird es möglich die Operationen der booleschen Algebra nutzen zu können. Hierdurch entsteht ein Vorteil gegenüber anderen Datenstrukturen für ein Schachfeld, da booleschen Operationen wie AND, ODER und Shift kleinere Prozessorberechnung für die Ausführung benötigen im Vergleich zur Multiplikation oder Modulorechnung.

### Generierung der Züge

Die Generierung der Züge basiert auf der booleschen Algebra. Bei der Generierung der Deck- und Angriffszüge der Dame, des Läufers und des Turms, da die Reichweite eines Zuges länger als ein Feld ist und in dieser Reichweite andere Figuren stehen können, unterscheidet sich die Erzeugung der Züge von den restlichen Figurarten.

Der Grundgedanke für die Kalkulation der Züge ist die Schnittmenge zweier Bitboards. Der eine trägt die Information der Gegnerfiguren und der andere beinhaltet die Angriffsreichweite der Figur, für die die Kalkulation erfolgt. Ein kurzer Einblick ist im folgenden Bild gegeben.

von der Dame

attackierte Felder & Gegnerfiguren = attackierte Figuren

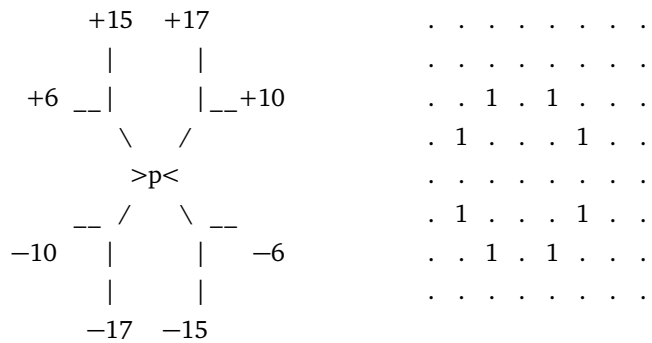
```

. . . . . 1 . . 1 1 . . 1 . . . . .
. . . 1 . . 1 . 1 . 1 1 1 1 1 . . . . 1 . . 1 .
. 1 . 1 . 1 . . . 1 . . . . . 1 . . . . .
. . 1 1 1 . . . . . . . . . . . . . . . . . . .
1 1 1 * 1 1 1 . & . . . * . . 1 . = . . . * . . 1 .
. . 1 1 1 . . . . . . . . . . . . . . . . . . .
. . . 1 . 1 . . . . . . . . . . . . . . . . . . .
. . . 1 . . . . . . . . . . . . . . . . . . .

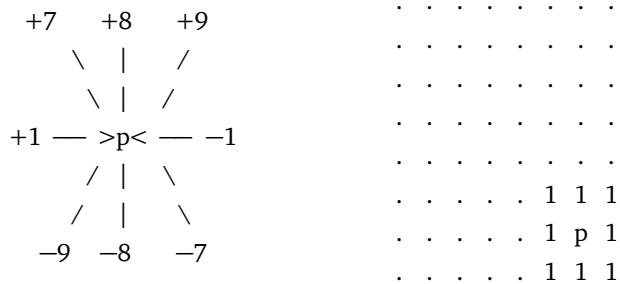
```

### Bauer-, Springer-, Königszüge

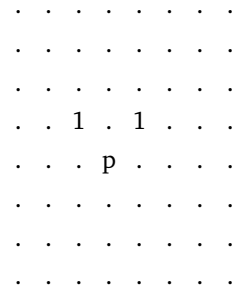
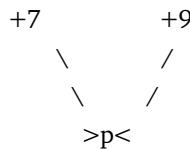
Das erste Bild veranschaulicht, wie ein Bitboard mit Angriffszügen für die Springerfigur vorkalkuliert werden kann. Die Position p stellt die Position des Springers dar. Für jeden neuen möglichen Zug wird ein Bitboard mit der geschifteten Position des Springers instanziiert. Um das Bitboard gesamt aller Angriffszüge zu erhalten, werden die neuen geschifteten Bitboards zu einem konjugiert (logisch bitweise). Das erste Bild enthält alle Schiftnumern für die Zugkalkulation des Springers. Das Bitboard daneben illustriert ein Beispiel der kalkulierten Züge.



Shift-Schema für die Kalkulation der Angriffe für die Dame und ein Beispiel des Bitboards aller vorkalkulierten Züge.

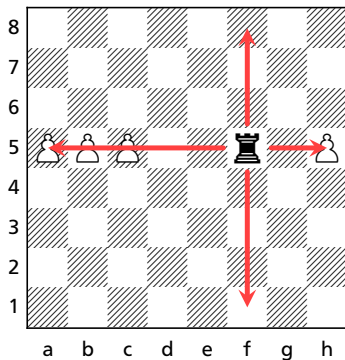


Die Bauernfigur hat nur die Möglichkeit, ein Feld nach vorne zu rücken oder eine andere Figur diagonal anzugreifen. Das heißt, dass das Shiften um 8 bei den Angriffen nicht berücksichtigt wird.



Alternativ können diese Angriffsbitboards für die oben genannten Figuren für jede Schachfeldposition vorkalkuliert werden, um die Kalkulation nicht mehr machen zu müssen, sondern nur das richtige Board für bestimmte Position und eine Figur abrufen. Die vorkalkulierten Bitboards werden als Lookup-Tabellen bezeichnet. Die Bitboards werden herkömmlich in einem Array gespeichert.

### Turm-, Läufer-, Damenzüge



Bei Figuren, deren Angriffsbereich länger als ein Feld ist und die eine Figur nicht überspringen können, reicht die Schnittmenge zweier Bitboards nicht aus, um das richtige Resultat zu erreichen. Die Bauern auf den Feldern a5 und b5 können vom Turm, aufgrund des Bauern auf dem Feld c5, nicht geschlagen werden. Resultat der Schnittmenge der möglichen Angriffszüge des Turms mit den vorgegebenen Bauern würde sie als angreifbar ausgeben. Bei den Figuren erfolgt die Berechnung ebenfalls mit der booleschen Algebra, jedoch etwas anders.

Die binäre Zahl 11100101 (*occupied* genannt) ist die Linie 5 von a5 bis h5 des Schachfeldes und repräsentiert alle Figuren, die sich auf dieser Linie befinden.

00000100 (*slider* genannt) repräsentiert den Turm, der sich auf f5 befindet.

Die Berechnung der Züge für den Turm muss separat für jede seiner Angriffsrichtungen erfolgen. In diesem Fall sind es a5-f5, f5-h5, f5-f8 und f1-f5. Es wird gezeigt wie eine Angriffslinie berechnet wird. Die anderen werden nach dem gleichen Prinzip berechnet. Folgende Schritte sind notwendig:

I.

occupied - (2\*slider)

$$11100101 - (2 \cdot 00000100) = 11011101$$

Die Zahl, die die Position des Turmes darstellt wird mit zwei multipliziert und von occupied subtrahiert.

II.

occupied XOR (occupied - (2\*slider))

$$11100101 \text{ XOR } 11011101 = 00111000$$

Auf das Resultat des ersten Schrittes wird occupied mit dem bitweisen exklusiven ODER (XOR) angewendet. Somit erhält man die linken horizontalen Angriffsfelder des Turms (a5-e5).

Um die rechte Seite zu erhalten müssen die Zahlen reversiert werden. Unter Reversieren ist das Umdrehen einer Binär-

zahl gemeint (Beispiel: 100111 hat folgende reversierte Form 111001). Die Funktion reverse steht für das Reversieren einer Zahl.

`reverse(occupied) = 10100111`

`reverse/slider) = 00100000`

Es folgen die gleichen Schritte mit reversierten Zahlen:

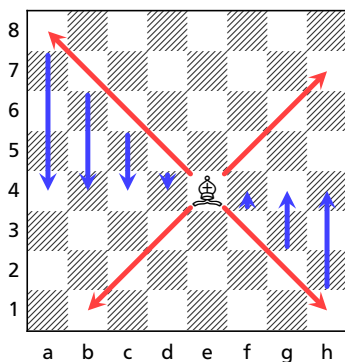
`reverse(reverse(occupied) XOR (reverse(occupied) - (2*reverse/slider))))`

`reverse(10100111 XOR (10100111 - (2*00100000))) =`

`reverse(10100111 XOR 01100111) =`

`reverse(11000000) = 00000011`

Das Resultat für die Angriffslinie g5-h5 ist 00000011. Das gleiche Vorgehen wird auf gleiche Weise für die restlichen Angriffslinien angewendet. Das Bitboard muss jedoch umgedreht werden, um die gewünschte Linie (occupied und slider) zu erhalten.



Die Kalkulationen der Angriffslinien für den Läufer erfolgt auf die gleiche Weise. Ein Läufer bewegt sich und greift nur auf der Diagonale seiner Farbe an. Die Diagonale muss ebenfalls als eine Zahl repräsentiert werden. Durch die bereits vorgestellten Shiftoperationen für die Königszüge ist schon bekannt, wie eine Figur sich diagonal bewegen kann. Mit dem gleichen Vorgehen werden alle Zahlen auf der Diagonale erreicht. Das Feld d5 wird von der Position des Läufers mit einem Shift um 9 Bits erreicht. Um auf das Feld c6 zu kommen wird die 9 mit 2 multipliziert und dann geshiftet. Für b7 erfolgt ein Shift um  $9 \cdot 3$  usw. Nachdem alle Zahlen der Diagonale bekannt sind, kann die occupied-Zahl konstruiert werden.

Die Dame vereint die Züge sowohl eines Turms als auch eines Läufers in sich. Das heißt, dass für die Damenangriffsberechnung die Zügberechnungen eines Turms und eines Läufers enthält.

---

## 2.5 Schachnotationen

---

Die gespielten Schachpartien sind im PGN Format gespeichert. Aus diesen Partien werden FEN-formatierte Schachstellungen extrahiert. Es folgen kurze Erläuterungen der genannten Formatierungen.

### Algebraische Notation

Die Algebraische Notation ist von der FIDE die einzig anerkannte Aufzeichnung der Züge während der Schachpartien. Die Schachfiguren werden mit dem ersten Buchstaben der englischen Figurbenennung bezeichnet, der groß geschrieben wird, außer des Springers (der Springer wird mit dem N bezeichnet) und der Bauernfigur, die keinen Buchstaben in der Notation enthält. Die acht vertikalen Linien des Schachbretts sind mit ihren Bezeichnungsbuchstaben gekennzeichnet (von a bis h). Die Reihen sind ebenfalls mit ihren Bezeichnungsziffern gekennzeichnet. Die Bezeichnung eines Feldes besteht aus zwei Zeichen. Das erste ist der Buchstabe für die Vertikale und das zweite ist die Ziffer für die Horizontale.

In der Notierung eines Zuges steht an der ersten Position die Kennzeichnung einer Figur, danach folgt das Feld, wohin die Figur bewegt wurde. Wenn die gezogene Figur eine Figur schlägt, wird es mit einem „x“ gekennzeichnet und steht zwischen der Figur- und der Feldbezeichnung. Wenn ein Bauer eine Figur schlägt, steht vor dem „x“ der Buchstabe aus

---

welcher Vertikale der Zug geschah. Bei einem en passant kann „ e.p“ an den Zug angefügt werden. Wenn zwei gleiche Figuren auf das gleiche Feld gezogen werden können, wird die gezogene Figur

1. falls beide Figuren auf derselben Reihe stehen, mit

- a) der Abkürzung ihren Namens
- b) der Herkunftslinie
- c) dem Ankunftsfield gekennzeichnet.

2. Falls beide Figuren auf derselben Linie stehen, mit

- a) der Abkürzung ihren Namens
- b) der Herkunftsreihe
- c) dem Ankunftsfield gekennzeichnet.

Wenn zwei Bauern gleiche Figur des Gegners angreifen können, der Bauer der gezogen wird, kann durch folgendes identifiziert werden

- a) durch den Buchstaben der Herkunftslinie
- b) durch ein „x“ (Kennung fürs Schlagen)
- c) durch das Ankunftsfield

Die Umwandlung eines Bauern wird durch das Feld, in dem die Umwandlung passiert und die Kennung der Figur, zu der der Bauer sich umwandelt, gekennzeichnet. Ein Angebot zu einem Remis wird mit (=) markiert. [7]

### Portable Game Notation

Der Begriff Portable Game Notation bezeichnet ein Format für die Darstellung der Schachpartien auf ASCII-basierten Textdateien. Das Format ist für das menschliche Auge gut lesbar, sowie leicht zu parsen. Es besteht aus zwei Teilen. Im ersten befinden sich Angaben betreffend des Turniers, des Orts, des Datums, der Runde, des Namens der Spiele. Der zweite Teil besteht aus dem Verlauf des Spiels, das in der algebraischen Notation formatiert ist.

### Forsyth-Edwards Notation

Die FEN (Forsyth-Edwards Notation) Notation liefert alle Information bezüglich einer einzelnen Stellung einer Partie. Die Notation wird mit der folgenden Grammatik näher gebracht.

<FEN> → <Figurenstellung><AmZug><Rochade><enPassant><Halbzüge><Zugnummer>

<FEN> besteht aus 6 Nichtterminalsymbolen, die voneinander mit einem Leerzeichen getrennt sind.

<Figurenstellung> → <Reihe> / <Reihe> / <Reihe> / <Reihe> / <Reihe> / <Reihe> / <Reihe> / <Reihe>

<Reihe> → <Ziffer><Figur> {<Ziffer><Figur>} [<Ziffer>] | '8'

Das Nichtterminalsymbol <Figurenstellung> besteht aus acht Reihen. Die Reihen sind mit einem Schrägstrich voneinander getrennt. Eine <Reihe> besteht aus Figuren oder Leerfeldern. Die Anzahl der Figuren plus die Summe der Ziffern muss gleich acht sein (Beispiel: 4B1r1 - vier Leerfelder, weißer Läufer, ein Leerfeld, schwarzer Turm, ein Leerfeld).

<Figur> → <WFigur> | <BFigur>

<Ziffer> → 1 | 2 | 3 | 4 | 5 | 6 | 7

<WFigur> → P | N | B | R | Q | K

---

<BFigur> → p | n | b | r | q | k

Die weißen Figuren werden mit großen Buchstaben dargestellt.

<AmZug> → w | b

Dieser Teil gibt an welcher Spieler am Zug ist, "w" steht für Weiß, "b" für Schwarz.

<Rochade> → - | K | Q | k | q | [K][Q][k][q]

"-" - Die Rochade bei beiden Farben ist nicht mehr möglich.

K - kurze Rochade (Weiß); Q - Lange Rochade (Weiß)

k - kurze Rochade (Schwarz); q - lange Rochade (Schwarz)

[K][Q][k][q] - stellt die Variation der möglichen Rochaden dar.

<enPassant> → - | <epFeld>

<epFeld> → <LinienB> <epReihe>

<LinienB> → a | b | c | d | e | f | g | h

<epReihe> → 3 | 6

<Halbzüge> → 0 | <n>

n ist eine positive ganze Zahl. Diese Zahl gibt die Anzahl der Halbzüge seit dem letzten Bauernzug oder Schlagen einer Figur.

<Zugnummer> → <n>

n ist eine positive ganze Zahl und gibt die Zugnummer an.

Die zwei vorgestellten Formate sind vom Steven J. Edwards entwickelt worden. [8]

---

## 2.6 Frequent Pattern in Graphen

---

### Frequent Subgraphs

Die Gewinnung von Wissen auf Basis der computergestützten Methoden aus großen Datenmengen wird als Data Mining bezeichnet. Graph Mining ist eine Erweiterung des Data Minings, bei der Informationen aus Graphen gewonnen werden. Die Untersuchung der Graphen auf Muster im Vergleich zu anderen Sequenzen wie Zahlen, Tupel oder Zeichenketten ist, aufgrund des Isomorphietests, NP-vollständig. Das bedeutet, dass es keinen Polynomialzeitalgorithmus gibt, der prüft, ob ein Subgraph in einem anderen Graph vorkommt.

Die folgende Zusammenfassung des gSpan-Algorithmus basiert auf der wissenschaftlichen Veröffentlichung von Xifeng Yan und Jiawei Han [10].

---

## 2.7 gSpan

---

### Lexikographische Ordnung der Graphen

Das Ziel der lexikographischen Ordnung, die unten eingeführt wird, ist die effiziente Entdeckung der häufig auftretenden Subgraphen. Mithilfe dieser Ordnung wird ein hierarchischer Suchraum aufgestellt. Jeder Knoten dieses Suchraums ist ein Graph mit einem eindeutigen lexikographischen Label. Die Suche startet vom kleinsten Graphen, der keine Kanten enthält und wächst mit jeder hinzugefügten Kante.

Es wird ein DFS-<sup>3</sup>Code für einen gegebenen DFS-Baum definiert. Mithilfe des DFS-Codes wird ein kanonisiertes Labeling-System für eine eindeutige Abbildung eines Graphen zu einer eindeutigen Sequenz erreicht. Der entwickelte DFS-Code

---

<sup>3</sup> Depth-First Search ist der englischer Begriff für die Tiefensuche

Baum stellt eine Relation zwischen den gesamten Graphen dar.

Ein Graph enthält mehrere Knoten und Kanten und es gibt keine Information welcher Knoten als Startknoten gelten soll. Deswegen gibt es mehrere Möglichkeiten aus einem Graphen einen DFS-Baum zu bauen. Ein DFS-Baum eines Graphen kann erzeugt werden, jedoch muss dieser DFS-Baum nicht eindeutig sein.

Die Traversierung eines Graphen  $G$  erfolgt linear mit folgenden Regeln:

sei ein DFS-Baum  $T$  gegeben mit  $\forall i, j : v_i \prec_T v_j \iff i < j$ . Das bedeutet, dass der Knoten  $v_i$  vor dem Knoten  $v_j$  entdeckt war. Jede Kante wird aufsteigend mit einer Ziffer von 0 bis  $n-1$  gekennzeichnet, wobei  $n$  für die Anzahl der Kanten in einem Graph steht.  $v_0$  wird als Wurzelkante bezeichnet und  $v_{n-1}$  als zuletzt entdeckte Kante. Das Ergebnis wird als  $G_T$  bezeichnet.

### Forward/Backward-Mengen von Kanten

Gegeben sei ein  $G_T$ , in dem die Forward-Kantenmenge alle Kanten, die ein DFS-Baum hat, enthält. In der Backward-Kantenmenge sind alle Kanten, die nicht in dem DFS-Baum enthalten sind.  $(v_i, v_j)$  repräsentiert eine Kante. Falls  $i < j$  gilt, so ist die Kante ein Element der Forward-Kantenmenge, ist  $i > j$  so ist die Kante in der Backward-Kantenmenge.  $E_{f,T} := \{e \mid \forall i, j : i < j, e = (v_i, v_j) \in E\}$  ist die Menge der Forward-Kanten in  $G_T$  und  $E_{b,T} := \{e \mid \forall i, j : i > j, e = (v_i, v_j) \in E\}$  die Menge der Backward-Kanten. Es werden zwei partielle Ordnungen  $\prec_{f,T}$  und  $\prec_{b,T}$  definiert. Angenommen  $e_1 = (v_{i1}, v_{j1})$  und  $e_2 = (v_{i2}, v_{j2})$ , dann gilt  $\forall e_1, e_2 \in E_{f,T} : e_1 \prec_{f,T} e_2 \iff v_{j1} < v_{j2}$ , sowie  $\forall e_1, e_2 \in E_{b,T} : e_1 \prec_{b,T} e_2 \iff$  i.  $v_{i1} < v_{i2}$  ii.  $v_{i1} = v_{i2}$  und  $v_{j1} < v_{j2}$ .

### DFS Code

Bei einem gegebenen DFS-Baum  $T$  eines Graphen  $G$  wird eine Sequenz für eine Kante basierend auf der  $\prec_{E,T}$  Ordnung konstruiert, sodass  $e_i \prec_{E,T} e_{i+1}$  mit  $i \in \{0, \dots, (|E| - 1)\}$  gilt und der Graph die folgende Notierung  $code(G, T)$  erhält. Grundsätzlich kann der DFS Code auf folgende Weise konstruiert werden: Zuerst wird ein neuer Knoten hinzugefügt, dann alle zu ihm führende Forward-Kanten vom alten Knoten, dann alle Backward-Kanten, die zu dem neu hinzugefügten Knoten führen. Das Beispiel aus Abbildung 1, Graph 2(b):  $((v_0, v_1), (v_1, v_2), (v_2, v_0), (v_2, v_3), (v_3, v_1), (v_1, v_4))$ .

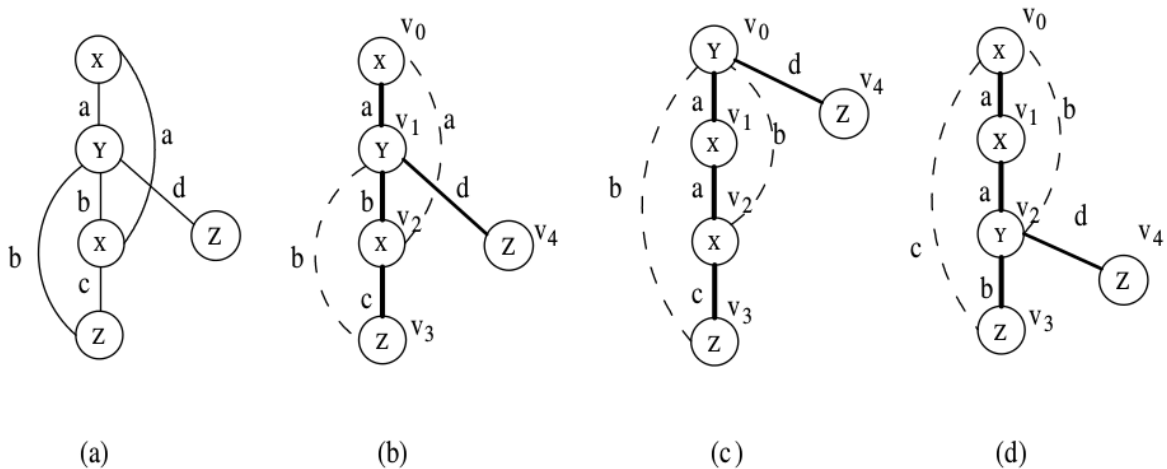


Abbildung 1: Tiefensuche und Forward/Backward-Kantenmengen

Kantennummer	(b) $\alpha$	(c) $\beta$	(d) $\gamma$
0	$(0, 1, X, a, Y)$	$(0, 1, Y, a, X)$	$(0, 1, X, a, X)$
1	$(1, 2, Y, b, X)$	$(1, 2, X, a, X)$	$(1, 2, X, a, Y)$
2	$(2, 0, X, a, X)$	$(2, 0, X, b, Y)$	$(2, 0, Y, b, X)$
3	$(2, 3, X, c, Z)$	$(2, 3, X, c, Z)$	$(2, 3, Y, b, Z)$
4	$(3, 1, Z, b, Y)$	$(3, 0, Z, b, Y)$	$(3, 0, Z, c, X)$
5	$(1, 4, Y, d, Z)$	$(0, 4, Y, d, Z)$	$(2, 4, Y, d, Z)$

**Tabelle 1:** DFS Codes für die Graphen 2(b), 2(c) und 2(d) aus Abbildung 1

Eine Kante wird als ein Tupel zweier Knoten repräsentiert  $(v_i, v_j)$ . Diese zwei Knoten verfügen jeweils über eigenes Label  $l(v_i), l(v_j)$  und es existiert eine Kante zwischen den beiden Knoten, die ebenfalls über ein Label verfügt  $l((v_i, v_j))$ . Diese Information wird in einem 5-Tupel zusammengefasst:  $(v_i, v_j, l(v_i), l(v_i, v_j), l(v_j))$ . Ein DFS Code eines Graphen besteht aus solchen 5-Tupeln. Tabelle 1 zeigt die DFS Codes für die Graphen 2(b), 2(c) und 2(d) aus Abbildung 1. Es folgen noch weitere Beschränkungen für den DFS Code.

### DFS Code's Neighborhood Restriction

Gegeben sei ein Graph  $G$ , ein DFS-Baum  $T$  und ein DFS Code  $\alpha = code(G, T)$ . Angenommen sei  $\alpha = (a_0, a_1, \dots, a_m)$ ,  $m \geq 2$  und zwei Nachbarn  $a_k = (v_i, v_j, l(v_i), l(v_i, v_j), l(v_j))$  und  $a_{k+1} = (v_l, v_m, l(v_l), l(v_l, v_m), l(v_m))$ . Dann müssen  $a_k$  und  $a_{k+1}$  folgende Regeln erfüllen:

Regel 1. falls  $a_k$  eine Backward-Kante ist, dann muss eine der folgenden Bedingungen gelten:

- i. falls  $a_{k+1}$  eine Forward-Kante ist, dann gilt  $v_l \leq v_i$  und  $v_m = v_{i+1}$ .
- ii. falls  $a_{k+1}$  eine Backward-Kante ist, dann gilt  $v_l = v_i$  und  $v_j \leq v_m$ .

Regel 2. falls  $a_k$  eine Forward-Kante ist, dann

- i. falls  $a_{k+1}$  eine Forward-Kante ist, dann gilt  $v_l \leq v_j$  und  $v_m = v_{j+1}$ .
- ii. falls  $a_{k+1}$  eine Backward-Kante ist, dann gilt  $v_l = v_j$  und  $v_m < v_i$ .

### DFS-lexikographische Ordnung

DFS-lexikographische Ordnung stellt eine Relation zwischen den Graphen dar.

Angenommen  $Z := \{code(G, T) \mid T \text{ ist ein DFS-Baum von } G\}$  ist die Menge aller DFS Codes und es gibt eine lineare Ordnung der Menge der Labels  $\prec_L$ . Die Kombination von  $\prec_L$  und  $\prec_{E,T}$  ist eine lineare Ordnung  $\prec_e$  auf der Menge  $E_T \times L \times L \times L$ . DFS-lexikographische Ordnung ist eine lineare Ordnung, die folgend definiert ist:

Falls  $\forall \alpha, \beta \in Z$ :  $\alpha = Code(G_\alpha, T_\alpha) = (a_0, a_1, \dots, a_m)$  und  $\beta = Code(G_\beta, T_\beta) = (b_0, b_1, \dots, b_m)$  ist, dann gilt  $\alpha \leq \beta$  genau dann, wenn eine der Aussage wahr ist:

- i.  $\exists t, 0 \leq t \leq \min(m, n), a_k = b_k$  für  $k < t, a_t \prec_T b_t$
- ii.  $a_k = b_k$  für  $0 \leq k \leq m$  und  $n \geq m$ .

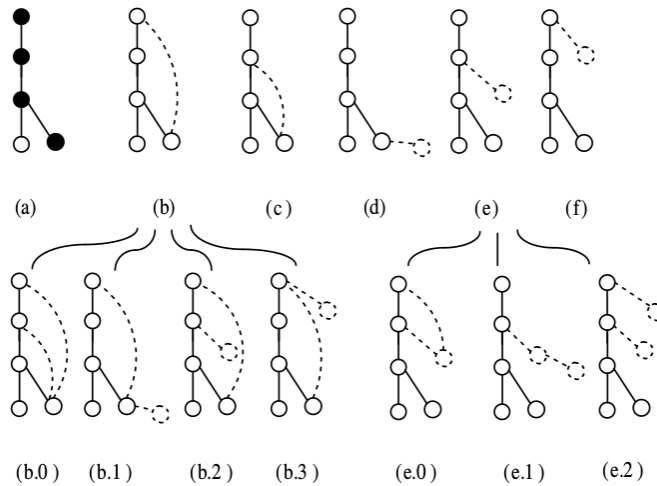
In Abbildung 1 ist der erste Graph in drei weiteren Variationen aufgezeichnet. Wenn die DFS Codes der Graphen b,c und d aufgestellt werden und die DFS-lexikographisch sortiert werden, erhält man  $\gamma \prec \alpha \prec \beta$ .



## minimaler DFS Code

Gegeben ist ein Graph  $G$ ,  $Z(G) := \{code(G, T) \mid \forall T, T \text{ ist ein DFS-Baum von } G\}$  basierend auf der DFS-lexikographischen Ordnung. Das minimale dieser Menge  $min(Z(G))$  wird als minimaler DFS Code des Graphen  $G$  bezeichnet und ist ebenfalls eine Kanonisierung dieses Graphen. Sind zwei Graphen isomorph zueinander, so sind deren Kanonisierungen ebenfalls isomorph zueinander. In den generierten Codes der Tabelle 1 für den gegebenen Graphen aus Abbildung 1 ist der minimale DFS Code  $\gamma$ .

## DFS Code Baum



**Abbildung 2:** DFS Code / Das Wachsen der frequenten Graphen

In einem DFS-Code-Baum repräsentiert jeder Knoten einen DFS-Code. Bei einem gegebenen DFS Code  $\alpha = (a_0, a_1, \dots, a_m)$ , falls es einen DFS Code  $\beta = (a_0, a_1, \dots, a_m, b)$  gibt, so wird er ein Kind von  $\alpha$  genannt und  $\alpha$  ist der Elterncode von  $\beta$ . Diese Relation stellt die Relation zwischen den Eltern- und Kinderknoten in einem DFS Code Baum dar. Die Relation zwischen den Geschwisterknoten ist durch DFS-lexikographische Ordnung konsistent. Ein DFS Code Baum enthält den minimalen DFS Code für alle Graphen.

## gSpan

Im gSpan-Algorithmus werden die Graphen als Adjazenzlisten verarbeitet. Die unteren Pseudo-Codes veranschaulichen die Funktionsweise des gSpan-Algorithmus.

### GraphSet\_Projection(GS, S)

1. sort labels of the vertices and edges in GS by their frequency
2. remove infrequent vertices and edges
3. relabel the remaining vertices and edges in descending frequency;
4.  $S' \leftarrow$  all frequent 1-edge graphs in GS
5. sort  $S'$  in DFS lexicographic order
6.  $S \leftarrow S'$
7. **for each** edge  $e \in S'$  **do**
8.     initialize  $s$  with  $e$ , set  $s.GS = \{g \mid \forall g \in GS : e \in E(g)\}$
9.     Subgraph\_Mining(GS, S,  $s$ )
10.     $GS \leftarrow GS - e$
11.    **if**  $|GS| < \text{minSup}$
12.    **break**

In den Zeilen von 1 bis 6: Die selten vorkommenden Knoten und Kanten werden aus der Graphmenge GS entfernt. In der absteigenden Reihenfolge werden die verbliebenen nach der DFS lexikographischen Ordnung sortiert und es werden neue Labels vergeben.

Angenommen die Menge der Labels für die Knoten sei  $\{A, B, C, \dots\}$  und  $\{a, b, c, \dots\}$  für die Kanten. Seien die Kanten  $(A,a,A)$ ,  $(B,b,C)$ ,  $(C,c,C)$  in GS frequent. Zuerst (Zeile 7) werden mithilfe der Methode Subgraph\_Mining(GS, S,  $s$ ) alle frequenten Subgraphen, die  $(A,a,A)$  enthalten, gefunden. Dann werden alle frequenten Subgraphen mit der Kante  $(B,b,C)$  aber ohne der Kante  $(A,a,A)$  gesucht. In der dritten Runde sucht die Funktion Subgraph\_Mining(GS, S,  $s$ ) die Graphen, die nur die Kante  $(C,c,C)$  enthalten. Mit diesem Vorgehen wird GS immer kleiner, sodass der Algorithmus immer schneller wird.

### Subgraph\_Mining(GS, S, s)

1. **if**  $s \neq \text{min}(s)$
2.     textbf{return}
3.      $S \leftarrow S \cup \{s\}$
4.     generate all  $s'$  potential children with one edge growth
5.     Enumerate( $s$ )
6.     **for each**  $c$ ,  $c$  is  $s'$  child **do**
7.         **if**  $\text{support}(c) \geq \text{minSup}$
8.              $s \leftarrow c$
9.             Subgraph\_Mining(GS, S,  $s$ )

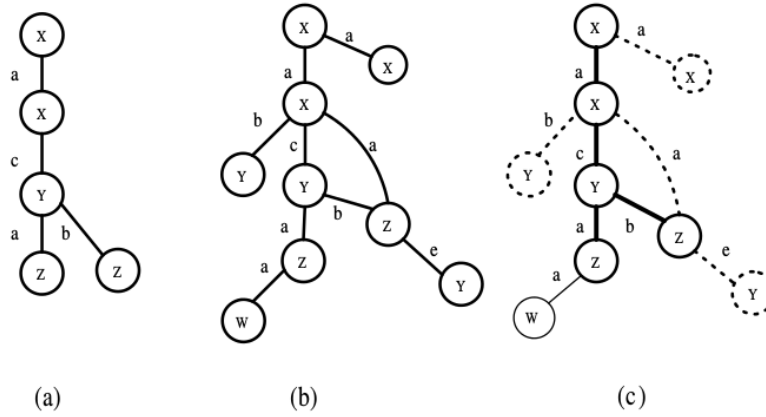
In jeder Rekursion wächst Subgraph\_Mining(GS, S,  $s$ ) von  $s$  um eine Kante und entdeckt alle frequenten Kindergraphen von  $s$  ( $s$  ist ein DFS Code, der ein Knoten des DFS Code Baumes ist).

### Enumerate( $s$ )

1. **for each**  $d \in s.GS$  **do**
2.     enumerate the next occurrence of  $s$  in  $g$
3.     **for each**  $c$ ,  $c$  is  $s'$  child and occur in  $g$  **do**
4.          $c.GS \leftarrow c.GS \cup \{g\}$
5.     **if**  $g$  covers all children of  $s$
6.     **break**

Die Methode Enumerate zählt alle Kookkurrenzen aller Kindergraphen von  $s$  in  $g$ . Als Beispiel zeigt Abbildung 3(a) einen, schon durch die vorigen Routinen entdeckten, Graphen. Abbildung 2(b)-2(f) veranschaulicht möglichen Kinder

dieses Graphen. In Abbildung 3(b) befindet sich der Graph in der Graphendatenmenge und die Abbildung 3(c) zeigt das Vorkommen des Graphen aus der Abbildung 3(a) in 3(b). Die gepunkteten Linien zeigen potentielle Kandidaten von 3(a) mit einer zusätzlichen Kante. Wegen der DFS Code's Neighborhood Restriction fällt der Knoten W mit der Kante a als Kandidat aus. Durch diese Methode werden die Kindergraphen auf die Häufigkeit geprüft.



**Abbildung 3:** Wachstum eines Subgraphen

### 3 Konzept

Wie bereits bekannt ist, sind die Chunks bestimmte Muster im Langzeitgedächtnis. Es stellt sich die Frage, worauf diese Chunks basieren. Diese Frage hat das Konzept der Arbeit festgelegt.

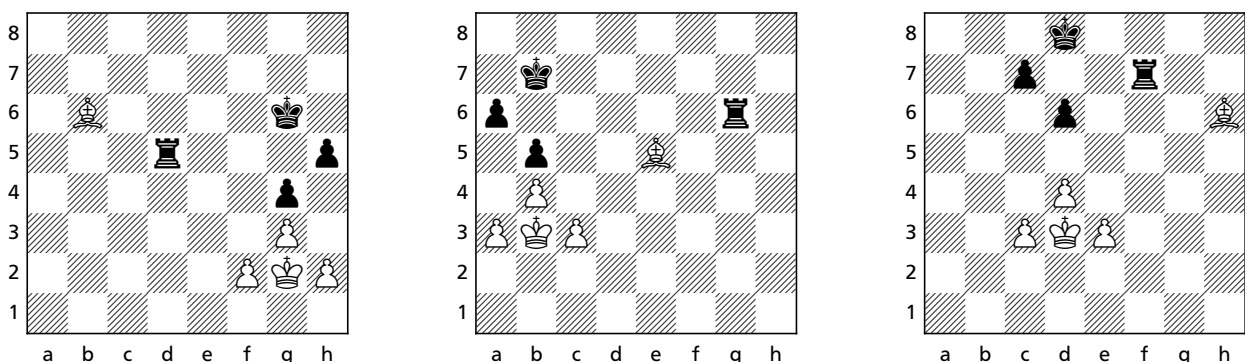
Bei den schon genannten Experimenten (2.2), wurde versucht die Logik hinter den Chunks und ebenfalls die Größe eines Chunks festzustellen. Bei der Rekonstruktion der gemerkten Schachstellung aus den Echtpartien wurden die Spieler gebeten die Stellungen zu beschreiben. Die Positionen, die für die Spieler wenig Sinn ergaben, waren am schwersten zu merken [9]. Die Spieler erzählten von bestimmten Figurenkonstruktionen wie Rochade, Bauernstrukturen, Fianchetto. Diese Konstellationen sind bekannte Chunks im Schach. Forscher haben ebenfalls die Zeitintervalle zwischen der Aufstellungen der Figuren gemessen, mit einer Annahme, dass in einer kurzen Überlegungszeit platzierte Figuren zu einem Chunk gehören. Bei Experimenten russischer Psychologen wurden die ersten fünf Sekunden der Augenfixierungen eines Spielers aufgenommen. Es zeigte sich, dass die meisten Blicke auf solche Relationen wie Attackieren und Decken der Figuren fokussiert waren. Es gab nahezu keine Fokussierung auf die Ecken des Schachbretts, sowie auf die leeren Felder [11]. Sicher kann gesagt werden, dass Relationen wie

- Attacken der Figuren
- Decken der Figuren
- gemeinsame Farbe
- gemeinsamer Typ einer Figur

in der Natur des Schachspiels liegen und die Chunks teilweise auf diesen aufbauen. [9]

Die oben genannten Relationen können in einem Graphen abgebildet werden. Die Schachfiguren werden als Knoten dargestellt, die Figurenrelationen als Kanten. Durch das Labeling werden die Relationen und die Figuren kodiert, sodass die Farben einer Figur, das Decken und die Attacken der Figuren dargestellt werden können.

Die Positionen der Figuren werden nicht berücksichtigt. Das heißt, dass ein Graph auf den genannten Relationen aufbaut. Der Grund dafür, die Position einer Figur bei der Suche nach häufigen Mustern außer Acht zu lassen ist, dass ein Chunk unabhängig von seiner Position überall auf dem Feld auftreten kann. Die unteren drei Schachstellungen veranschaulichen die gleichen Chunks auf unterschiedlichen Positionen, sowie auch ein symmetrisches Vorkommen eines Chunks. Die erste und die zweite Stellung zeigen ein symmetrisches Vorkommen einer Schachstellung. Ebenfalls kann die Position einer Figur in bestimmten Stellungen irrelevant sein. Als Beispiel dient die Platzierung des Läuders oder des Turms in den folgenden Stellungen.



Eine gerichtete Kante stellt einen Angriff oder eine Deckung einer Figur am besten dar. Deswegen war am Anfang dieser Arbeit vorgesehen, gerichtete Graphen aus den Schachstellungen zu erzeugen. Die Suche nach Programmen, die frequente Mustern in gerichteten Graphen suchen war erfolglos. Es gab einige Projekte und wissenschaftliche Arbeiten [12] [13], jedoch keine geprüften Realisierungen. Es ist gelungen mit einem Mitglied eines Projektes Kontakt aufzunehmen und Herr Washio [13] empfahl den gSpan-Algorithmus. Der gSpan-Algorithmus arbeitet auf vollständigen ungerichteten Graphen ohne Mehrfachkanten. Die Ergebnisse der Performanz [10] des Algorithmus stellen gute Resultate dar. Aus diesem Grund wurde das Konzept komplett auf ungerichtete Graphen umstrukturiert.

---

## 4 Implementierung

---

Während der Planung dieser Arbeit war beabsichtigt mit C/C++ zu programmieren, da es einige Vorteile in der Effizienz gibt. Aus diesem Grund war die Suche nach fremden Programmen, die in C/C++ geschrieben sind, beschränkt, um sie besser in eigenen Code einbinden zu können.

---

### 4.1 Schachpartien

---

Als Alltagsmedium hat sich das Internet schon seit einigen Jahren etabliert. Es gibt mehrere Schachportale, die das Online-Schachspielen anbieten und gespielte Partien speichern, auf die ein freier Zugriff möglich ist. Ebenfalls gibt es kommerzielle Schachdatenbanken, die alle FIDE-Partien aufzeichnen und ebenfalls Partien aus den alten Archiven zur Verfügung stellen. Da die Seriosität der Partien aus den Online-Portalen nicht eingeschätzt werden kann, außerdem ist die Spielstärke der Spieler nur auf das Portal beschränkt und die Möglichkeit der Manipulation nicht ausgeschlossen werden kann, sind die Spiele aus der kommerziellen Schachdatenbank ChessBase [14] entnommen worden. Die Spiele sind im PGN-Format und von Spielern mit einer Spielstärke von 2500 bis 3000 Elo gefiltert worden.

---

### 4.2 Extraktion der Schachstellungen

---

Die Extraktion der Schachstellungen aus den Schachpartien erfolgte mit einem Open-Source Programm namens pgn-extract [15]. Das Programm ist in C geschrieben. Der Code dieses Programms wurde angepasst, sodass aus einem Spiel von einem bestimmten Intervall der Spielzüge zu jedem Zug oder auch in einem n-Schritt eine FEN-Formatierung der Stellung erfolgt. Diese FEN-Stellungen werden in eine Textdatei für die weitere Verarbeitung geschrieben.

---

### 4.3 Generierung der Graphen

---

Bei der Generierung der Graphen wird auf die Methoden der Open-Source Software namens Stockfish [16] zugegriffen, die zu dem Implementierungscode integriert sind. Das Programm liest die FEN formatierten Stellungen ein und stellt jegliche Bitboards dieser Stellung zur Verfügung.

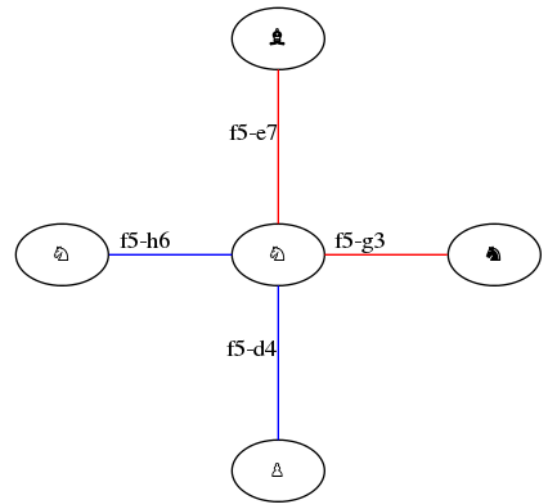
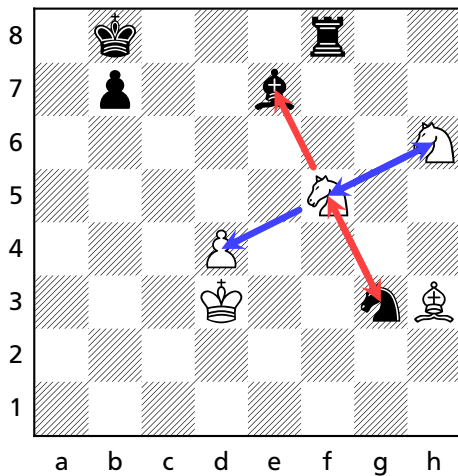
Der Pseudocode erklärt das grobe Vorgehen der Erstellung der Graphen. Bei den Punkten I. und II. basiert die Kalkulation auf den Bitboards, die im Abschnitt 2.4 beschrieben sind.

#### Generation of Graphs

1. Initialize a graph, chess\_position
2. Read a FEN from the input file and store into the chess\_position
3. **for each** square of chess\_position **do**
4.     **if** there is a piece on the square
5.         I. calculate attacks
6.         II. calculate defends
7.         III. generate vertices and edges
8.         IV. store edges and vertices into the graph

Punkt III befasst sich mit der Erzeugung der Kanten und der Knoten für den Graphen. Das Schachbrett ist zu dem Bitboard, das Informationen über alle Figuren trägt, bijektiv und jede Figur hat eine eindeutige Position, die auch den Knoten im Graphen identifiziert. Zu jeder Figur auf einer Position ist ebenfalls eine eindeutige Kodierung zugeordnet (die Kodierungen sind in der Tabelle 2 dargestellt). Die Kodierungen sind Labels für Knoten in einem Graph. Der gSpan-Algorithmus kann nur ungerichtete und vollständige Graphen verarbeiten und erlaubt ebenfalls keine Mehrfachkanten.

Aus diesem Grund muss die Kante so kodiert werden, dass alle Informationen aus der Relationen zweier Knoten (Figuren) aus der Kodierung extrahiert werden können. Die Kodierungen sind Labels für Kanten und Knoten.



Die obere Schachstellung zeigt die Angriffs- und Deckungszüge der Springerfigur auf f5. Daneben ist der Subgraph des Gesamtgraphen, der diesen Knoten und die von ihm ausgehende Relationen zu anderen Figuren darstellt. Es sind folgende Relationen, die unterschieden werden sollen, möglich:

- I. eine Figur greift eine andere Figur an
- II. eine Figur deckt eine andere Figur
- III. die Figuren decken sich gegenseitig (da die Mehrfachkanten nicht erlaubt sind, wird dies mit einer Kante dargestellt)
- IV. die Figuren greifen sich gegenseitig an
- V. eine Figur steht zu keiner anderen Figur in Relation (aufgrund der vollständigen Graphen)

Der Springer in der oberen Stellung steht zu dem Läufer auf e7 in Relation I, zu dem Bauern auf d4 in Relation II, zu dem Springer auf h6 in Relation III und zu dem Springer auf g3 in Relation IV.

Figur	weiß	schwarz
Bauer	0001	1001
Springer	0010	1010
Läufer	0011	1011
Turm	0100	1100
Dame	0101	1101
König	0110	1110

**Tabelle 2:** binäre Kodierungen der Schachfiguren / Labels für die Knoten

### Beschreibung der Kantenkodierung

I. Relation: diese Relation stellt die Beziehung eines Angriffs einer Figur auf eine andere Figur dar. Bei einem Angriff von Weiß auf Schwarz werden die Bits von 8 bis 9 auf 00 gesetzt, bei dem Angriff von Schwarz auf Weiß wird der 8 Bit auf 1 gesetzt und der 9 auf 0.

II. Relation: bei dieser Relation geht es um die Deckung einer Figur durch einer anderen. Die Ordnungsrelationskodierung dient zu erfahren, welche Figur gedeckt wird. Wenn eine größere Figur gedeckt wird, wird die Ordnungsrelationskodie-

zung auf 00 gesetzt, andersherum wird das 8 Bit auf 1 gesetzt und das 9 auf 0.

III. Relation: es gibt Figuren die sich gegenseitig immer decken, wie Figuren gleicher Art oder positionsabhängig decken können (z.B. Turm und Dame auf einer Linie). Diese Kodierung stellt solche Relationen dar. Alle Bits der Kodierung werden auf 0 gesetzt.

IV. Relation: diese Relation ist analog zu der III. Sie bezieht sich jedoch auf die Angriffe von Figuren.

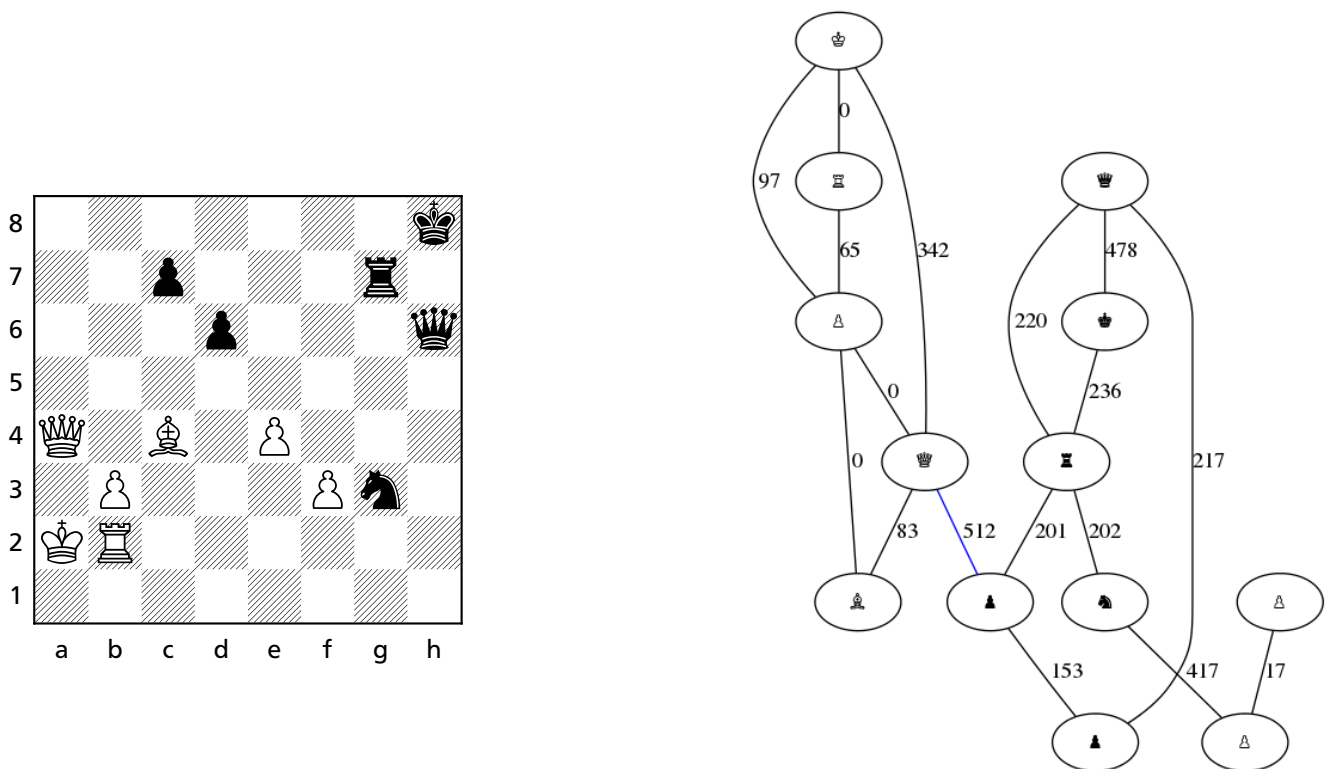
V. Relation: Es muss ein geschlossener Graph sein. Das heißt, dass es keine isolierten Knoten geben darf. Diese Kodierung dient dazu zu zeigen, dass ein Knoten (Figur) zu keiner Figur in einer Relation steht. Diese Relation kommt vermehrt im Endspiel vor, da es dort weniger Figuren gibt.

Die Relationen werden in Tabelle 3 veranschaulicht.

Die Ordnungsrelationskodierung basiert auf der Kleiner-Relation. Die Elemente dieser Relation sind die Labels für die Schachfiguren (Tabelle 2).

Relation	8-9 Bit	4-7 Bits	0-3 Bits
I.	Farbenkodierung	Figurkodierung	Figurkodierung
II.	Ordnungsrelationskodierung	Figurkodierung	Figurkodierung
III.	00	0	0
IV.	00	0	0
IV.	10	0	0

**Tabelle 3:** binäre Kodierung einer Kante / Labels für die Kanten



**Abbildung 4:** Graph einer Schachstellung

Bei der Deckung eines Bauern von einem anderen Bauern ermöglicht die Kodierung der Kante nicht zu erfahren, von welchem Bauern die Deckung ausgeht.

---

Das Setzen der Kanten mit dem Label 512 folgt keiner Logik. Die Kanten werden nur gesetzt wenn der Graph nicht vollständig ist.

In Abbildung 4 ist ein Graph gezeigt, der die nebenstehende Schachstellung illustriert. Ohne die Kante mit dem Label 512 (Relation IV) wäre der Graph nicht vollständig. Diese Kante wird bei den Auswertungen der Resultate nicht berücksichtigt.

---

#### 4.4 Ausgabe der Graphen für das gSpan-Programm

---

Nach der Erzeugung der Graphen erfolgt eine Generierung einer Inputdatei für das gSpan-Programm [17]. Darüber hinaus wurde für die Auswertung der Resultate ein Parser für die Outputdatei geschrieben.



---

## 5 Experimente

---

Der Verlauf eines Schachspiels wird in drei Phasen aufgeteilt. Die Phasen eines Schachspiels sind nicht klar getrennt, jedoch verfügt jede Phase über bestimmte Eigenschaften. Die Eröffnung ist die Bezeichnung der ersten Phase, in der der Spieler seine Figuren für die nächste Phase mobilisiert und strategisch aufstellt. Das heutige Wissen der Schachspieler über Eröffnungen geht sehr weit bis zum Mittelspiel. Deswegen werden die ersten 15 Züge bei den kommenden Experimenten ausgelassen. Die Eröffnung endet, indem die Figurenentwicklung abgeschlossen ist. Als Mittelspiel wird die nächste Phase bezeichnet. Im Gegensatz zu den einstudierten Eröffnungen macht die hohe Variation der Mittelspielphase das Auswendiglernen unmöglich und der Spieler ist auf sein tiefes Positionsverständnis angewiesen. Bei den Experimenten liegt die Phase zwischen dem 15. und dem 40. Zug einer Partie. Danach folgt die Endspielphase. In dieser Phase ist die Anzahl der Figuren reduziert, der König wird ins Spiel gebracht. Diese Phase hat das Intervall vom 45. bis zum 70. Zug in den Experimenten.

Spielphase	Nummer der Züge
Mittelspiel	15 - 40
Endspiel	45 - 70

**Tabelle 4:** Aufteilung der Spielphasen bei den Experimenten

Ein getätigter Spielzug kann große Änderungen im Spiel verursachen. Es bewegt sich nur eine Figur, somit geschieht keine große Veränderung zwischen den Relationen mehrerer Figuren auf dem gesamten Schachfeld. Aus diesem Grund wird nicht jede Schachstellung extrahiert, sondern jede Stellung, die nach fünf Halbzügen passiert ist. Somit fließen vielfältigere Schachstellungen einer Partie in die Graphenmenge ein.

---

### 5.1 Experiment 1

---

Im ersten Experiment werden 100000 Schachstellungsgraphen aus den Schachpartien der Spieler mit einer Stärke zwischen 2500 und 3000 Elo extrahiert. In Tabelle 5 sind die Resultate der Suche der häufigsten Subgraphen mittels gSpan-Algorithmus vorgestellt. Die Spalten sind nach der relativen Häufigkeit unterteilt. Die Zeilen stellen die Anzahl der Graphenvorkommnisse, mit der in ersten Spalte vorgegebene Knotenanzahl im Graph, dar. Die Suche der häufigsten Subgraphen startet mit einem minimalen Support von 30%. Dann wird der Wert auf 20% gesetzt und als letztes auf 10%.

Anzahl der Knoten	Anzahl der Graphen (10%)	Anzahl der Graphen (20%)	Anzahl der Graphen (30%)
1	12	12	12
2	60	36	22
3	115	27	15
4	89	9	2
5	7	-	-

**Tabelle 5:** Ergebnisse des Experiments 1

Die Anzahl der Graphen mit einem Knoten ist bei allen drei minimalen Supportwerten zwölf. Diese zwölf Knoten stellen die sechs Schachfiguren jeweils in Weiß und Schwarz dar.

Figur	absolute Häufigkeit
♘ / ♞	68560
♙ / ♚	75878
♖ / ♗	92071
♔ / ♕	68519

**Tabelle 6:** absolute Häufigkeiten N, B, R, Q Figuren

Die häufigsten Graphen mit einer Kante sind Paare, in denen ein Bauer in Relation zu einer anderen Figur steht. Die unteren beiden Tabellen zeigen die absoluten Häufigkeiten dieser einkantigen Graphen. Tabelle 7 bezieht sich auf die Graphen, in denen ein Bauer gedeckt wird. In Tabelle 8 sind die Graphen, in denen ein Bauer angegriffen wird. Die häufigste Figur, die von einem Bauern gedeckt wird, ist der Springer (ca. 32% relativer Häufigkeit).

In einem Intervall zwischen 18% und 25% relativer Häufigkeit liegen jegliche Paare, die sich gegenseitig decken. Dazu gehören zum Beispiel solche Relationen, in denen ein Turm von einem Läufer oder eine Dame von einem Springer gedeckt werden. Die weniger frequenten Paare waren die, in denen ein Springer einen Läufer deckt oder ein König von einem Springer gedeckt wird.

Das gegenseitige Decken (Kanten mit dem Label 0) war bei den Graphen am häufigsten, in denen zwei Türme (ca. 23%), Turm und König, Turm und Dame, Läufer und Bauer sich gegenseitig decken. Weniger häufig war die Kombination aus Dame und Läufer (ca. 10%). Sich gegenseitig deckende Springer waren unter den Graphen nicht zu finden.

Figur	absolute Häufigkeit
♖ / ♗	89236
♙ / ♚	83439
♖ / ♗	64121
♔ / ♕	58740
♘ / ♞	44953
♙ / ♚	41149

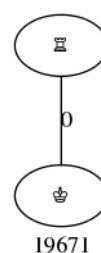
**Tabelle 7:** Deckung der Bauernfigur

Figur	absolute Häufigkeit
♔ / ♕	35770
♙ / ♚	32284
♘ / ♞	28953
♖ / ♗	28056
♙ / ♚	15678

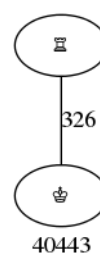
**Tabelle 8:** Angriff auf die Bauernfigur

### Einkantige Graphen

Der Graph mit der Relation, in der der Turm den König deckt kommt häufiger als das gegenseitige Decken beider Figuren vor. Der Grund ist der aktive Turm im Mittelspiel.



**Abbildung 5:**  $R \leftrightarrow K$



**Abbildung 6:**  $R \rightarrow K$

Die zweikantigen Graphen, die einen Angriff von Figuren wie Turm, Läufer, Springer auf einen Bauern und umgekehrt, darstellten, erwiesen eine relative Häufigkeit von ca. 30%. Die Kombination, in der eine Dame einen Bauern angreift war die häufigste. Der Angriff der Dame auf einen Springer war wenig frequent.

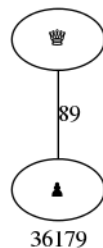


Abbildung 7:  $Q \rightarrow P$

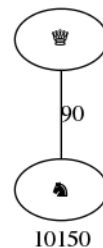


Abbildung 8:  $Q \rightarrow N$

### Graphen mit drei Knoten

Drei verbundene Bauern waren ebenfalls eine häufige Sequenz. Die unteren Graphen (Abbildungen von 10 bis 15) weisen auf eine Rochadenstellung mit zwei verbundenen Türmen, in denen die Positionen der Türme, des Königs und der Bauern leicht variieren.

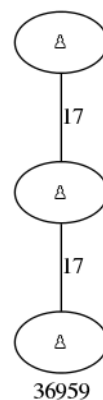


Abbildung 9:  $P \rightarrow P \rightarrow P$

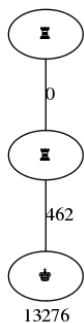


Abbildung 10:  $r \leftrightarrow r \rightarrow k$

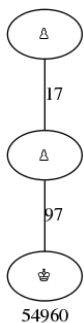


Abbildung 11:  $K \rightarrow P \rightarrow P$



Abbildung 12:  $R \leftrightarrow R \rightarrow P$

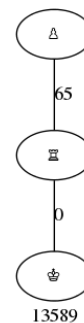


Abbildung 13:  $K \leftrightarrow R \rightarrow P$

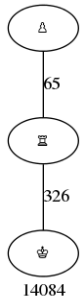


Abbildung 14:  $R \rightarrow K; R \rightarrow P$

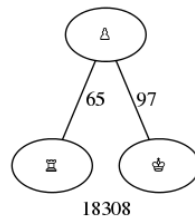
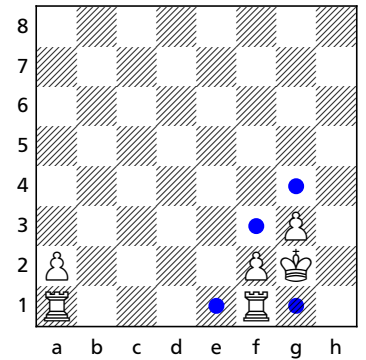


Abbildung 15:  $R \rightarrow P; K \rightarrow P$



In Abbildung 16, in der der letzte Bauer einer Kette aus zwei Bauern vom Turm gedeckt wird. Mit einer ähnlichen Frequenz waren die gleichen Graphen, in denen anstatt eines Turmes eine Dame, Läufer, König oder ein Springer den Bauern deckten. Ähnlich frequente Graphen, in der der letzte Bauer von einer Figur angegriffen wird, zeigt die Abbildung 17.

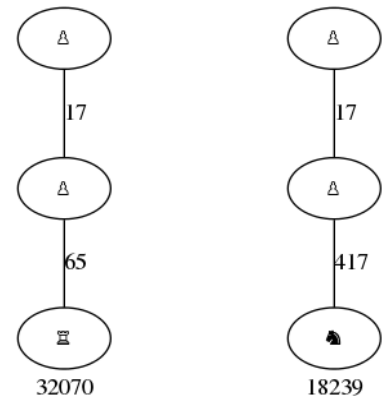


Abbildung 16:  $R \rightarrow P \rightarrow P$     Abbildung 17:  $n \rightarrow P \rightarrow P$

Die in Abbildungen 18 und 19 vorgestellten Graphen treten ebenfalls mit den Figuren wie Bauer, Turm, Läufer, König anstatt der Dame auf.

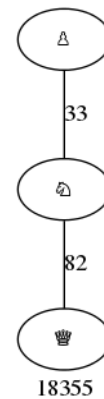


Abbildung 18:  $Q \rightarrow N \rightarrow P$

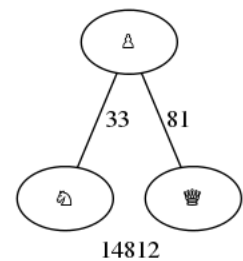


Abbildung 19:  $N \rightarrow P; Q \rightarrow P$

Die unteren Abbildungen zeigen eine Zusammenfassung der frequenten Graphen mit einer Variierung der Figuren. In ähnlichen Graphen zu Abbildung 21, nur anstatt des Läufers traten andere Figuren wie eine Dame oder ein Springer frequent auf. Abbildung 22 mit einem Wechsel des Königs mit anderen Figuren war ebenfalls frequent. Identisch zu Abbildung 23, gab es Graphen, in denen die Dame einen Turm, einen Bauern oder einen Springer deckt oder einen Angriff auf einen Bauern tätigt.

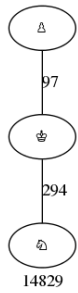


Abbildung 20:  $K \rightarrow P; N \rightarrow K$

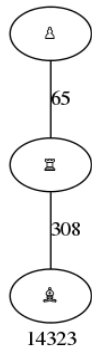


Abbildung 21:  $B \rightarrow R; R \rightarrow P$

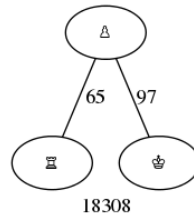


Abbildung 22:  $R \rightarrow P; K \rightarrow P$

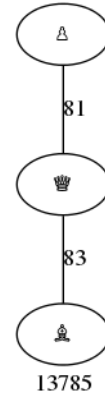


Abbildung 23:  $B \leftarrow Q \rightarrow P$

### Graphen mit vier Knoten

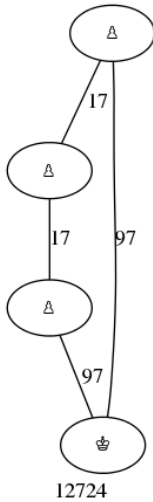


Abbildung 24: 4 Knoten

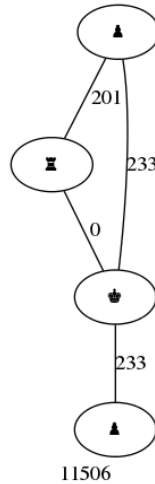


Abbildung 25: 4 Knoten

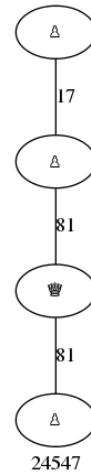


Abbildung 26: 4 Knoten

Drei verbundene Bauern mit einem König, der diese zwei Bauern deckt und weitere auf eine Rochade deutende Stellungen wie in Abbildungen 24 und 25 waren frequent.

Weitere Graphen mit vier Knoten waren eine Mischung aus den häufigsten ein-, zwei- oder dreikantigen Graphen. Ein Paar Fragmente sind in den Abbildungen von 26 bis 29 dargestellt.

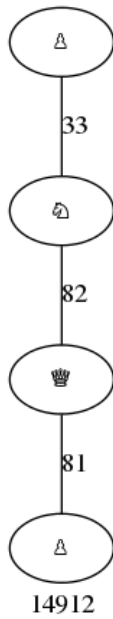


Abbildung 27: 4 Knoten

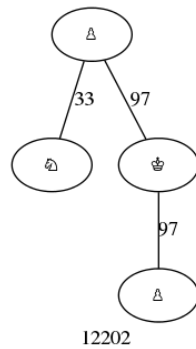


Abbildung 28: 4 Knoten

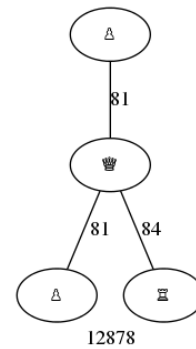


Abbildung 29: 4 Knoten

### Graphen mit fünf Knoten

Die Anzahl der Graphen mit fünf Knoten war sehr gering. Entweder deuteten die Graphen auf einen, nach einer Rochade positionierten, König mit nebenstehenden Bauern oder einem Turm oder es waren Graphen, die eine Mischung der vorherigen Graphen darstellten.

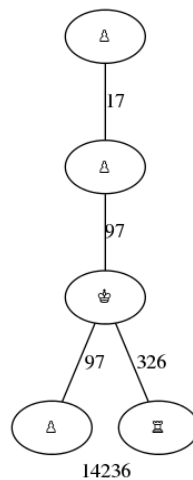


Abbildung 30: 5 Knoten



Abbildung 31: 5 Knoten

## 5.2 Experiment 2

Im letzten Experiment wurden sehr offensichtliche Relationen zwischen den Figuren entdeckt und die Anzahl der Graphen mit mehr als fünf Knoten war nicht groß. In diesem Experiment wird der minimale Support auf 5% gesetzt. Die zeitliche Ausführung des gSpan-Algorithmus ist stark gestiegen.

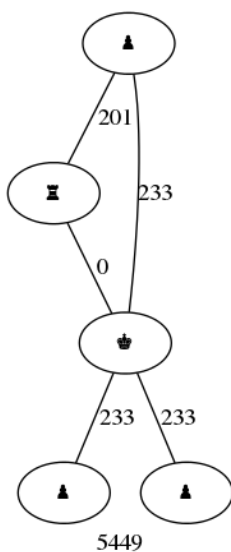
Tabelle 9 veranschaulicht die Ergebnisse der gefundenen Graphen.

Anzahl der Knoten	Anzahl der Graphen (5%)
1	12
2	79
3	320
4	416
5	250
6	15

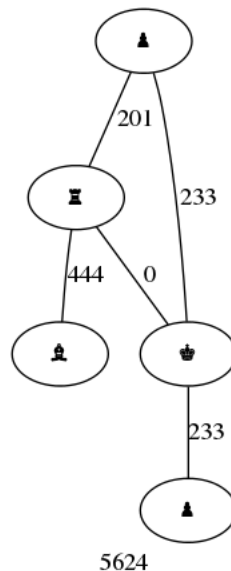
**Tabelle 9:** Ergebnisse des Experiments 2

### Beobachtungen

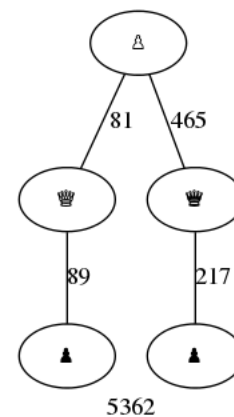
Die Anzahl der gefundenen Graphen ist deutlich gestiegen, jedoch stellen die Graphen eine Mischung aus frequenten Graphen mit kleinerer Anzahl der Knoten dar. Das Wachstum der frequenten Graphen basiert auf der Eltern-Kind-Beziehung. Das heißt, dass der Anstieg der Graphen mit vier oder fünf Knoten durch den Anstieg der kleineren Graphen verursacht worden ist. Die Resultate der Graphen mit drei Knoten zeigen eine hohe Variation der Deckungen gleichfarbiger Figuren und die wachsenden Graphen sind ebenfalls weitere Variationen einer Deckung. Die Interaktionen zweier Farben bei Graphen mit vier oder fünf Knoten sind sehr selten und in keiner hohen Variation zu sehen.



**Abbildung 32:** 5 Knoten



**Abbildung 33:** 5 Knoten



**Abbildung 34:** 5 Knoten

Oben Abbildungen 32, 33 und 34 sind drei ausgewählte Graphen mit fünf Knoten. Die ersten beiden Graphen zeigen die Deckvariationen der gefundenen Graphen und der letzte zeigt die Interaktionsrelationen beider Farben.

### 5.3 Experiment 3











Bei diesem Experiment handelt es sich um die Endspielphase. Die Partien sind wie in den vorigen Experimenten von Schachspielern der Elostärke von 2500 bis 3000. Die Graphen sind aus den Schachstellungen, die nach dem 45. Zug passieren, extrahiert. Die Anzahl der erstellten Graphen beträgt 100000.

Die Resultate der gefundenen Graphen sind in Tabelle 10 illustriert.











Anzahl der Knoten	Anzahl der Graphen (5%)	Anzahl der Graphen (10%)	Anzahl der Graphen (20%)
1	12	12	12
2	38	22	8
3	40	9	2
5	6	-	-

**Tabelle 10:** Ergebnisse des Experiments 3

Im Endspiel sind weniger Figuren auf einem Schachfeld anwesend, dies könnte die Erklärung für die Senkung der gefundenen Graphen sein. Die gefundenen Graphen sind zum ersten Experiment sehr ähnlich jedoch mit einer kleineren Frequenz. Es handelt sich dabei meistens um Konstruktionen aus einem König, Bauern, Turm oder Dame. Es gibt keine Graphen, in denen ein Springer und ein Läufer zueinander in einer Relation stehen.

Figur	absolute Häufigkeit
 / 	96634
 / 	62344
 / 	41450
 / 	33072
 / 	20870

**Tabelle 11:** absolute Häufigkeiten N, B, R, Q Figuren

Figur	absolute Häufigkeit
 / 	99587
 / 	92071
 / 	75878
 / 	68560
 / 	68519

**Tabelle 12:** aus dem Experiment 1

Tabelle 11 zeigt die absoluten Häufigkeiten der Schachfiguren in den Endspielen. Daneben sind die Resultate für die gleichen Figuren aus dem Experiment 1. Die beiden Tabellen veranschaulichen, dass das Schachbrett mit weniger Figuren besetzt ist.

## 5.4 Experiment 4











Bei einem relevanten Materialunterschied gehen einige Partien früher zu Ende und es gibt ebenfalls Partien, in denen ein Remis vorzeitig angeboten wird. Das führt dazu, dass die Anzahl der Züge nicht weit über 45 Züge hinausgeht. In diesem Experiment sind 81509 Graphen aus den Partien, die mit über 65 Zügen lang sind, extrahiert worden.

Die Resultate sind ähnlich zum vorigen Experiment mit noch mehr gesunkener Häufigkeit der Figuren und der Graphen. Die untere Tabellen zeigen die Resultate des Experimentes. Bei dem minimalen Support von 20% sind Damen nicht mehr im Endspiel zu finden.

Anzahl der Knoten	Anzahl der Graphen (5%)	Anzahl der Graphen (10%)	Anzahl der Graphen (20%)
1	12	12	10
2	37	16	8
3	31	8	2
4	4	-	-

**Tabelle 13:** Ergebnisse des Experiments 4



Figur	absolute Häufigkeit
 / 	76959
 / 	49064
 / 	31439
 / 	23922
 / 	13322

**Tabelle 14:** absolute Häufigkeiten N, B, R, Q Figuren

---

## 5.5 Auswertung

---

Das erste Experiment beschreibt ausgewählte frequente Graphen. Im zweiten Experiment werden die wesentliche Unterschiede der Ergebnisse aufgeführt und ebenfalls einige Graphen illustriert, die im ersten Experiment aufgrund des minimalen Supports nicht aufgesucht wurden. Bei den letzten beiden Experimenten sind nur die Tabellen mit den resultierenden Graphen angegeben. Der Grund dafür ist, dass die gefundenen frequenten Graphen schon im ersten Experiment beschrieben sind. Aufgrund der Eigenschaften des Endspiels war die Anzahl der gefundenen Graphen und die Frequenz dieser Graphen niedrig.

Die Ergebnisse der Tests zeigen unterschiedliche Variationen der Deckungen zwischen den Figuren gleicher Farbe. Angriffsinteraktionen zwischen den Figuren waren nicht vielfältig und diese Graphen waren an der unteren Grenze des minimalen Supportwertes. Der Grund dafür ist möglicherweise die große Anzahl an Variationen in einem Schachspiel und die Grundprinzipien des Schachspielens, bei den Figurendeckung ein wichtiges Element ist. Die gefundenen Graphen stellten schon bekannte Informationen bezüglich der Figurenrelationen wie Rochade, verbundene Bauern und welche Figuren miteinander öfters oder seltener agieren dar.

---

## 6 Zusammenfassung und Ausblick

---

Im Rahmen dieser Bachelorarbeit wurden Spiele aus einer Schachspieldatenbank mittels Graph Mining untersucht.

Zu Beginn der Arbeit erfolgte eine Einführung in die Forschungsexperimente aus der kognitiven Psychologie, bei denen es unter anderem um die Experimente mit Schachspielern ging. Die Resultate dieser Experimente initiierten den Kerngedanken dieser Bachelorarbeit. Daraufhin wurden die Grundlagen, die für diese Arbeit notwendig sind, beschrieben. Ein Teil der Grundlagen befasst sich mit der Graphentheorie, der andere mit Bitboards, die eine Grundlage der Informationsextraktion der Schachpartien darstellen. Im letzten ging es um das Graph Mining, in dem der gSpan-Algorithmus vorgestellt wurde.

Auf den Resultaten der Forschungsexperimente und den Gegebenheiten, die aus einem Schachspiel extrahiert werden können baut das Konzept dieser Arbeit auf, das im Kapitel 3 erläutert wird. Es wird ebenfalls begründet welche Elemente ein Graph einer Schachstellung abbildet und was mit Absicht nicht berücksichtigt wird. Der Grund der Wahl der ungerichteten Graphen für das Graph Mining ist ebenfalls in diesem Kapitel genannt.

Im Kapitel 4 werden die Komponenten der Implementierung beschrieben. Aus Performanzgründen sind zwei fremde Programme in das Projekt integriert worden, die bei der Extraktion der Schachpartien mitwirken. Es wird eingeführt, wie ein Graph konstruiert ist. Das Labeling ermöglicht es, viele Informationen in einer Kante sowie in einem Knoten zu übertragen. Die Kodierung wird näher gebracht.

Bei den Experimenten wurde die Eröffnungsphase ausgelassen, aus dem Grund, da sie einstudiert ist. Der Kern der Experimente war die Gewinnung von Wissen darüber, welche frequenten Muster in Schachpartien großer Vielfalt vorkommen. Die Experimente sind in zwei Teile aufgeteilt. In Experimenten I und II wird das Mittelspiel einer Partie untersucht. Die letzten beiden Experimente befassen sich mit dem Endspiel. Die Gründe der Aufteilung sind die unterschiedlichen Eigenschaften dieser Phasen.

Zukünftig wäre eine weitere Betrachtung dieser Herangehensweise interessant. Die Graphen können basierend auf einer bestimmten Situation zielgerichteter erstellt werden. Das Schachprogramm, das in diesem Projekt integriert war, bietet ebenfalls Funktionen, die bei der Graphenerzeugung hilfreich sein können. Dazu gehören beispielsweise die Bewertungsfunktionen oder die Betrachtung der Fesselungen. Darüber hinaus kann mit einer parallelisierten Version eines Graphenalgorithmus eine Suche mit einem sehr niedrigen minimalen Supportwert durchgeführt werden [18].

---

## Literaturverzeichnis

---

- [1] Daniel J. Levitin; This Is Your Brain on Music: The Science of a Human Obsession
- [2] FIDE, Schachregeln <http://www.max-maier.de>,  
letzter Zugriff am November 2015
- [3] Stefan Klein; Wie berechenbar ist das Schachspiel?, 2011
- [4] George A. Miller, The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information, 1956
- [5] K. Anders Ericsson; William G. Chase; Steve Faloan, Acquisition of a Memory Skill, 1980
- [6] Herbert A. Simon; William G. Chase, Skill in Chess, 1973
- [7] FIDE rules on algebraic notation,  
<http://portablegamenotation.com/FIDE.html>  
letzter Zugriff am November 2015
- [8] Standard: Portable Game Notation Specification and Implementation Guide,  
<http://www6.chessclub.com/help/PGN-spec>  
letzter Zugriff am November 2015
- [9] William G. Chase; Herbert A. Simon; Perception in Chess, Carnegie-Mellon University, 1973
- [10] Xifeng Yan; Jiawei Han; gSpan: Graph-Based Substructur Pattern Mining, 2002
- [11] O. K. Tichomirov, E. D. Poznyanskaya; An Investigation of Visual Search as a Means of Analyzing Heuristics. 1966
- [12] Yuhua Li; Quan Lin; Gang Zhong; Dongsheng Duan; Yanan Jin; Wei Bi; A Directed Labeled Graph Frequent Pattern Mining Algorithm based on Minimum Code, College of Computer Science & Technology Huazhong University of Science and Technology, 2009
- [13] Akihiro Inokuchi; Takashi Washio; Hiroshi Motoda; Complete Mining of Frequent Patterns from Graphs: Mining Graph Data, Institute for Scientific and Industrial Research, Osaka University, 2003
- [14] Mega Database 2012 [http://shop.chessbase.com/de/cat\\_root](http://shop.chessbase.com/de/cat_root)  
letzter Zugriff am November 2015
- [15] A Portable Game Notation (PGN) Manipulator for Chess Games,  
<https://www.cs.kent.ac.uk/people/staff/djb/pgn-extract/>  
letzter Zugriff am November 2015
- [16] Stockfish Chess, Homepage und Git-Repository  
<http://stockfishchess.org/>  
<http://stockfishchess.org/mac/>  
<https://github.com/daylen/stockfish-mac>  
letzter Zugriff am November 2015
- [17] SOFTWARE - gSpan: Frequent Graph Mining, Package <https://www.cs.ucsb.edu/~xian/software/g-Span.htm>  
letzter Zugriff am November 2015

- 
- [18] Marc Sebastian Wörlein; Extension and parallelization of a graph-mining-algorithm; Friedrich-Alexander-Universität Erlangen–Nürnberg, 2006
- [19] Bitboards, The Chess Programming Wiki  
<http://chessprogramming.wikispaces.com/Bitboards>  
letzter Zugriff am November 2015