Spider Solitaire

David Anton Laubersheimer

6. August 2024

Johannes Gutenberg University Mainz
FB08
Institut für Informatik
Algorithmics

Spider Solitaire

David Anton Laubersheimer

1. Gutachter Dr. Markus Blumenstock
Institut für Informatik - Algorithmics
Johannes Gutenberg Universität Mainz

2. Gutachter Prof. Dr. Ernst Althaus
Institut für Informatik - Algorithmics
Johannes Gutenberg Universität Mainz

Betreuer Dr. Markus Blumenstock

6. August 2024

Abstract

In dieser Arbeit wird die NP-Vollständigkeit von Spider-Solitaire mit einer Farbe bewiesen. Hierzu wird der Beweis von Jesse Stern erweitert. Des Weiteren werden weitere Varianten von Spider-Solitaire formuliert und deren Π_2^p-Vollständigkeit gezeigt. Außerdem wird untersucht, wie Spider-Solitaire einschränken werden kann, um es effizient lösbar zu machen. Als Letztes werden Nichtapproximierbarkeitsresultate gezeigt.

In einem zweiten Teil der Arbeit wird untersucht, ob man den SMT-Solver Z3 verwenden kann, um Lösungen für gegebene Spider-Solitaire Instanzen zu suchen.

Inhalt

1 Einleitung

1.1 Regeln von Spider-Solitaire

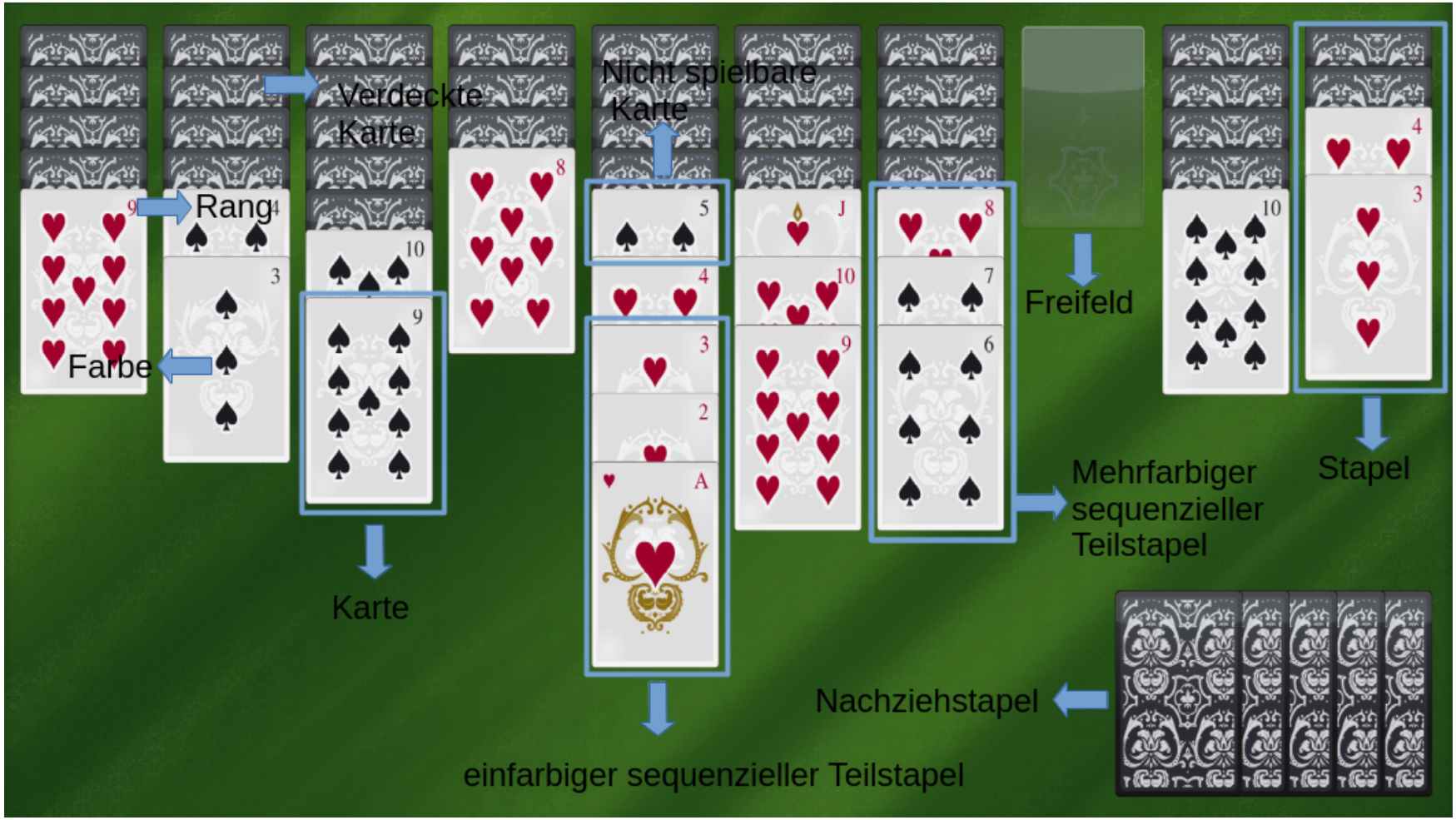
Da Spider-Solitaire ein weit verbreitetes Spiel ist, gibt es sicherlich einige Implementierungen, die in den Regeln voneinander abweichen.
Als Referenz für die Regeln wird in dieser Arbeit [9, Kpatience] verwendet.
Der Vollständigkeit halber sind hier noch einmal kurz die Regeln, wie sie von Kpatience beschrieben werden, angegeben:

> Spider is played with two card decks. The cards are dealt out into 10 playing piles, 4 of 6 cards and 6 of 5 cards each. This leaves 50 cards that can be dealt out 10 at a time, one on each playing pile.
>
> In the playing piles, a card can be placed on another card of any suit and of one higher value. A sequence of descending cards of the same suit may be moved from one playing pile to another.
>
> The goal of spider is to put all cards as real families descending from Kings anywhere in the playing piles. When such a family is built in a playing pile, it is removed to the lower-left corner of the window.
>
> The different levels determine how many suits are dealt - Easy uses 1 suit, Medium uses 2 suits, and Hard uses all 4 suits. The game is fairly easy to win at Easy level, and very difficult to win at Hard level.
>
> [13]

Die Leser:innen sind jetzt angehalten, falls noch nicht geschehen, einige Runden Spider-Solitaire zu spielen, um sich mit den Regeln vertraut zu machen.

1.2 Begriffsklärungen

Folgendes Bild gibt einen kurzen Überblick über häufig verwendete Begriffe.



1.3 Formale Definitionen

Eine Karte ist Zweitupel aus $\{1, \cdots, n\} \times \{1, \cdots, f\}$. n ist der maximale Kartenrang, f die Anzahl an Farben. Für Farben, die ≤ 4 sind sei:
1 =: ♠, 2 =: ♡, 3 =: ♣, 4 =: ♢
Die Karte mit niederwertigsten Rang heißt Ass, die Karte mit höchstwertigem Rang wird als König bezeichnet. Ein Vorkommen der Karten 1 bis n einer Farbe wird als Deck bezeichnet.
Ein (Nachzieh-)Stapel S ist eine sortierte Liste von Karten, mit Duplikaten.
Ein Spielzustand G ist ein Tupel $G = (S_1, \cdots S_k, N_1, \cdots N_s)$ aus k vielen Stapeln und s vielen Nachziehstapeln.

Solang nicht anders angegeben bezeichnet $n \in \mathbb{N}$ den maximalen Kartenrang, $k \in \mathbb{N}$ die Anzahl an Decks und $p \in \mathbb{N}$ die Anzahl an Stapeln in einer Spielinstanz G
Ein einfarbiger maximaler sequenzieller Teilstapel eines Stapels $S = (s_1, \cdots, s_i)$ ist ein Teilstapel, $T = (s_u, \cdots s_v)$ in dem die Karten eine Sequenz bilden, die gleiche Farbe haben und $(s_u, \cdots, s_{v+1})$ nicht sequenziell ist oder nicht mehr existiert und $(s_{u-1}, \cdots, s_v)$ nicht sequenziell ist.
Für jede Karte, die auf dem Nachziehstapel liegt, erhöht sich die Anzahl an maximalen sequenziellen Teilstapeln um eins.

1.3.1 Züge

Ein Zug ist ein Dreitupel, bestehend aus Ursprungsstapel, Zughöhe und Zielstapel. $(S_1, h) \to S_2$ bezeichnet einen Zug, der einen Teilstapel der Höhe h von S_1 nach S_2 bewegt.

$G \circ m$ beschreibt den Spielzustand, der aus G durch Anwenden des Zuges m entsteht.
Eine Zugfolge M ist eine geordnete Liste von Zügen. $M \circ m$ ist eine neue Zugfolge, die als letztes m ausführt und $m \circ M$ für die Zugfolge, die zuerst m, gefolgt von den Zügen in M ausführt.
Sind m, m' Züge so bezeichnet $m \circ m'$ die Zugfolge bestehend aus m gefolgt von m'
$G \circ M$ beschreibt den Zustand, der durch Anwenden der Züge in M entsteht.

m^{-1} beschreibt den Zug, der genau die Karten, die m bewegt, wieder zurückbewegt. Man beachte, dass im Allgemeinen m^{-1} im Zustand $G \circ m$ kein gültiger Zug ist. Wenn m auf den Ablegestapel legt, ist zu beachten, dass dann zum Beispiel m^{-1} Karten vom Nachziehstapel wegbewegt, was nie ein gültiger Zug ist.
m^{-1} entspricht also genau der Rückgängigtaste.
Sei $M = (m_1, \cdots, m_t)$ eine Zugfolge. Dann bezeichnet $M = (m_t^{-1}, \cdots, m_1^{-1})$ die inverse Zugfolge von M.
Sei G ein Spiel, S ein Stapel von G mit sequenziellen Teilstapel T. Ein Zug $m = (S, h) \to S'$ heißt spaltend, wenn in $G \circ m$ Teile von T in S und andere Teile in S' enthalten sind. Ein Zug m heißt einfarbig spaltend, wenn die bewegten Karten und die Zielkarten dieselbe Farbe haben.

1.3.2 Lösung, Gewinnen und Verlieren

$G_\emptyset$ ist das gewonnene Spiel. In ihm ist $S = \emptyset$ und $N = \emptyset$
Ein Spiel G ist gewinnbar oder lösbar, wenn es eine Zugfolge M an gültigen Zügen gibt, sodass $G \circ M = G_\emptyset$
Eine Lösung ist eine Zugfolge M von gültigen Zügen, sodass am Ende alle Karten auf den

Ablagestapeln liegen.

Umgekehrt gilt dementsprechend ein Spielzustand als nicht lösbar bzw. nicht gewinnbar, wenn es keine gültige Zugreihenfolge M gibt, mit der am Ende alle Karten auf dem Ablagestapel liegen.
Ein Spiel ist verloren, wenn der aktuelle Zustand (ohne den Rückgängig-Knopf) nicht lösbar ist.

1.4 Vorrausgegangene Arbeiten

Diese Arbeit würde ohne das Paper "Spider Solitaire is NP-Complete" von Jesse Stern[12] nicht existieren. In diesem zeigt Stern bereits, dass Spider-Solitaire mit vier Farben NP-Vollständig ist.
Das Kapitel 4 "NP-Vollständigkeit und mehr" basiert in großen Teilen auf dem Paper und ergänzt dies an einigen Stellen. Sterns Paper selbst basiert auf den Ideen des Papers "Complexity results for standard benchmark domains in planning" [4] welches unter anderem die NP-Vollständigkeit von Freecell zeigt.

Des Weiteren gibt es eine Masterarbeit von Mark S. Weiser mit dem Titel "How many Games of Spider Solitaire are Winnable - Explorations into the mathematics underlying spider solitaire"[14]. In dieser wird sich sowohl theoretisch als auch durch Implementierung einer heuristischen Suche, damit beschäftigt, mit welcher Wahrscheinlichkeit ein zufällig generiertes Spiel gewinnbar ist. Außerdem betrachtet er zum Beispiel auch Spiele, bei denen der Nachziehstapel unendlich groß ist. Als Endfazit kommt er zu dem Schluss, dass 99,9 % der zufällig generierten Spiele gewinnbar sind.

Natürlich müssen auch die vielen Solitaire Implementierungen erwähnt werden. In dieser Arbeit wurde auf Kpatience[9] aufgebaut. Dies implementiert sowohl ein einfaches Dateiformat als auch Patsolve[11], eine Bibliothek, die verschiedene Solitaire-Spiele lösen kann. Im Kapitel 3 "Implementierung" wird unter anderem darauf eingegangen, wie man das .kpat Dateiformat einlesen kann und der hier implementierte Löser mit dem von Kpatience verglichen.

2 Effizient lösbare Teilprobleme

In diesem Kapitel wird untersucht, ob man die Regeln von Spider-Solitaire so einschränken kann, dass es effizient entschieden werden kann.

2.1 Ein Deck, ohne Nachziehen, ohne Freifelder

Im Folgenden wird gezeigt, dass ein Spiel, in dem nur die Karten ♠1 $\cdots$ ♠n genau einmal vorkommen, in dem der Nachziehstapel bereits leer ist und keine Züge auf Freifelder erlaubt sind, in polynomieller Zeit gelöst/entschieden werden kann. Das Spiel hat verdeckte Karten, von denen aber der Wert bekannt ist. Das heißt, dass man zum Beispiel weiß, dass unter der ♠6 eine ♠7 verdeckt liegt, man die ♠7 aber nicht bewegen darf. Wären beide Karten nicht als verdeckt markiert, so wäre das Bewegen der ♠7 möglich.

Im Folgenden bezeichnet $a \to a+1$ das Bewegen der a-ten Karte auf die a+1-te Karte. Liegen auf der a-ten Karte weitere Karten, so ist die Bewegung eines Teilstapels gemeint.

Für zwei Karten $a, a+1$, die im initialen Zustand noch nicht aufeinander liegen, muss der Zug $a \to a+1$ in jeder Lösung vorkommen, da ansonsten die Lösung zu keiner Sortierung führt. [1]

Lemma: Sei G ein gewinnbares Spiel. Dann kann G mit einer beliebigen Folge von gültigen Zügen gewonnen werden.

Der Beweis erfolgt über Induktion nach der minimalen Anzahl an Zügen, in der ein Spiel gewonnen werden kann.

Induktionsanfang:
$G_\emptyset$ kann definitionsgemäß in 0 Zügen gewonnen werden.
Ein Spiel, das in einem Zug gewonnen werden kann, muss nur noch auf den Nachziehstapel abgelegt werden. Es kann also mit einem beliebigen Zug gewonnen werden.
Induktionsvoraussetzung:
Sei G ein Spiel, das in t oder weniger Zügen gewonnen werden kann. Dann kann G durch eine beliebige Folge von gültigen Zügen gewonnen werden.
Induktionsschritt:
Sei G ein gewinnbares Spiel, das in $t+1$ Zügen gewonnen werden kann.
Sei:
$(a_1 \to a_1+1, a_2 \to a_2+1, \cdots, a_{t+1} \to a_{t+1}+1)$
Eine beliebige gewinnende Zugfolge von G, mit $t+1$ Zügen

Man betrachte einen beliebigen Zug m := $b \to b+1$, der zu Beginn in G zulässig ist.
Falls $a_1 = b$ wurde der Zug der Gewinnstrategie gewählt. Der Zustand $G \circ m$ ist in t Zügen gewinnbar, kann nach I.Vor. also mit beliebiger Zugfolge gewonnen werden.

Im Fall $a_1 \neq b$ wurde ein Zug, der nicht der Lösung entspricht, gewählt. Die Karten a und $a+1$ bleiben aber weiterhin offen liegen, der Zug, der für die Lösung hätte ausgeführt werden müssen, bleibt also weiterhin möglich. Gleichzeitig kommt der Zug

[1] Bei doppelten Karten kann es zwei Möglichkeiten geben einen Zug auszuführen,z.B $a \to a+1$ und $a' \to a+1$, daher generalisiert dieses Argument nicht.

$b \to b+1$ auch in der Lösung vor. Der Zug wurde lediglich vorgegriffen.
Man betrachte:
$(a_1 \to a_1+1, a_2 \to a_2+1, \cdots, a_t \to a_t+1, b \to b+1)$
also alle Züge, die bis zu dem Zug, in dem die Karte b bewegt worden wäre, stattfinden. Keine Karte wurde so verdeckt, dass sie nicht mehr bewegt werden kann, da es nur ein Deck und eine Farbe gibt. Da alle Kartenränge eindeutig sind, wurde auch kein Zug verhindert, indem die Karte b darauf abgelegt wurde.
Ab dem Zug $a_{t+1} \to a_{t+1}+1$ erhält man wieder denselben Zustand, der auch in der ursprünglichen gewinnenden Zugfolge zu diesem Zeitpunkt erreicht worden wäre. Das Spiel ist also nach dem Zug $m := b \to b+1$ immer noch lösbar.
Es insbesondere ist $G \circ m$ in t Zügen gewinnbar, da:
$(b \to b+1, a_1 \to a_1+1, \cdots, a_t \to a_t+1, a_{t+1} \to a_{t+1}+1, \cdots)$ eine gewinnende Zugfolge ist.
Nach Induktionsvoraussetzung kann der Zustand $G \circ m$ also mit einer beliebigen Zugfolge gewonnen werden. Da m beliebig gewählt worden ist, kann somit G mit einer beliebigen Folge von gültigen Zügen gewonnen werden.
□

Daher kann auch effizient entschieden werden, ob ein Spiel lösbar ist, indem man, solange es noch gültige Züge gibt, diese ausführt, bis entweder keine Züge mehr möglich sind, oder das Spiel gewonnen ist.
Dies ist auch gleichzeitig eine Strategie, mit der man ein gewinnbares Spiel immer gewinnt. Dies macht die so eingeschränkte Spielvariante zu einem eher uninteressanten Spiel.

Unklar bleibt, ob auch mit dem Nachziehstapel weiterhin effizient entschieden werden kann, ob ein Spiel lösbar ist. Auch unklar bleibt, ob mit Freifeldern weiterhin effizient Lösungen gefunden werden können.

2.2 Inversionen

Sei G jetzt ein einfarbiges Spiel mit einem Kartendeck, in dem der Nachziehstapel leer ist. Züge auf Freifelder seien jetzt erlaubt. Eine Inversion oder Fehlstand einer Permutation[2] ist ein Paar von Werten, dessen Reihenfolge durch die Permutation vertauscht wird.

Inspiriert davon wird eine Inversion auf einem Stapel von Karten definiert. Man sagt die Karten a und b bilden eine Inversion, wenn $rang(a) = rang(b) + 1$, also die Karte b auf die Karte a gelegt werden muss, aber die Karte b über der Karte a liegt, ohne das b direkt auf a liegt, oder b verdeckt ist.[2] Ein Stapel hat eine Inversion, wenn zwei Karten in ihm eine Inversion bilden.

2.2.1 Lemma

Damit wird folgender Hilfssatz gezeigt:
Besitzt jeder Stapel eine Inversion, so ist das Spiel mit einem Deck und ohne Nachziehstapel nicht gewinnbar.

Beweis: Man betrachte den i-ten Stapel S_i. Seien a und b zwei Karten, die eine Inversion in S_i bilden. Sei o.B.d.A.

[2] Kpatience setzt eine ähnliche Idee in der Heuristik (Chaos) des Lösers um.
[3] Hierfür brauchen wir wieder die Eindeutigkeit von a

$rang(a) > rang(b)$.
Wie oben bereits begründet muss der Zug $b \rightarrow a$ stattfinden.[3] Dies ist aber nur möglich, wenn die Karte b auf einen leeren Stapel bewegt wird. Der Stapel S_i kann also nur leer werden, wenn die Karte b auf ein existierendes Freifeld bewegt werden kann.
Dies gilt aber für alle Stapel, wodurch auf keinem Stapel ein Freifeld entstehen kann. (da dafür ja bereits ein Freifeld existieren müsste.) Somit kann der Zug $b \rightarrow a$ nicht ausgeführt werden, weswegen das Spiel nicht gewinnbar ist.□

2.2.2 Ohne Freifelder

Lemma: In einem gewinnbaren Spiel mit einem Deck und ohne Freifelder hat kein Stapel eine Inversion:
Sei G ein gewinnbares Spiel. Angenommen der Stapel S_i hat eine Inversion, mit Karten a, b,$rang(a) > rang(b)$. Dann muss wie eben begründet b auf ein Freifeld bewegt werden. Widerspruch.□

Leider reichen Inversionen nicht aus, um zu entscheiden, ob ein Spiel gewinnbar ist. Man kann damit nur einige Spiele direkt als ungewinnbar markieren. Dies wird durch folgende Spielinstanz deutlich:

♠1	♠5	♠2
	♠3	♠4
		♠6

In diesem Spiel besitzt kein Stapel eine Inversion, es ist aber dennoch nicht lösbar.

2.3 Sortierte Spiele

Ein Stapel S ist sortiert, wenn der Rang der Karten von unten nach oben immer echt kleiner wird. Ein Spiel heißt sortiert, wenn alle Stapel sortiert sind und keine Karte verdeckt ist.

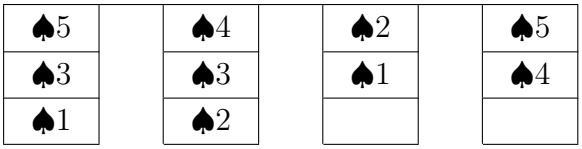


Abbildung 1: Beispiel eines sortieren Spiels

2.3.1 Sortierte Spiele sind immer Gewinnbar

Satz: Sei G ein sortiertes Spiel mit k Decks, einer Farbe und maximalem Rang n. Dann ist G gewinnbar.

G hat mindestens k Stapel, da die Karten in jedem Stapel echt kleiner werden müssen. Wäre die Anzahl an Stapeln kleiner als k müsste eine Karte in einem Stapel doppelt vorkommen.
Hat G genau k Stapel, kommt jede Karte in jedem Stapel genau einmal vor. Jeder Stapel ist also vollständig und sortiert, kann also direkt auf den Ablagestapel bewegt werden. Das Spiel ist nach k Zügen gewonnen.

Hat ein Spiel mehr als k Stapel kann diese Argumentation nicht direkt angewendet werden. Über Induktion nach dem maximalen Rang n wird eine gewinnende Zugfolge konstruiert.

Induktionsanfang: $n = 0$: Ein Spiel ohne Karten ist definitionsgemäß gewonnen.
$n = 1$: In dem Spiel kommen nur Asse vor. Diese können alle direkt auf den Ablagestapel gelegt werden. Das Spiel ist also gewinnbar.

Induktionsannahme: Ein sortiertes Spiel mit maximalem Rang n ist gewinnbar.

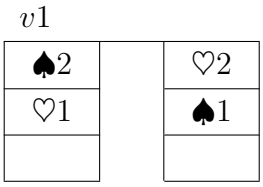
Induktionsschritt: Sei G ein sortiertes Spiel mit maximalem Rang n+1. Betrachte die ♠1. Diese Karten müssen alle an oberster Stelle eines Stapels liegen. Es gibt zwei Fälle:
1) Unter der ♠1 liegt eine ♠2. Tue nichts.
2) Unter der ♠1 liegt keine ♠2. Da es mehr als k Stapel gibt, muss es Stapel geben, die kein Ass enthalten. Da es innerhalb der Stapel keine Dopplungen geben kann, muss es einen Stapel geben, der kein Ass, aber eine zwei enthält. In diesem Stapel ist dann die Zwei an obere Stelle. Führe den Zug ♠1→♠2 aus.
Dies ist für jedes Ass möglich. Jedes Ass liegt nach diesen Zügen auf einer zwei. Die ♠2 sind, da alle Karten dieselbe Farbe haben, weiterhin bewegbar. Im weiteren Spielverlauf müssen die Asse nicht mehr von der ♠2 wegbewegt werden, da um zu gewinnen auf jeder zwei genau ein Ass liegen muss.
Da die Asse ab jetzt permanent auf der Zwei liegen, kann man sie einfach aus der Spielinstanz streichen. Durch Umnummerieren der Karte ♠a zu ♠$a-1$ erhält man ein sortiertes Spiel G' mit maximalem Rang n, das nach Induktionsvoraussetzung gewinnbar ist.
Sei $M = (m_1, m_2, \cdots m_t)$ die gewinnende Zugfolge des transformierten Spiels. Alle diese Züge sind auch, nach geeigneter Umbenennung, in G anwendbar, da die zusätzlichen Asse auf den Zweien das Bewegen der Stapel nicht verhindern. Zusammen mit den Zügen,

die die Asse auf die Zwei bewegen, bildet M, nach entsprechender Umnummerierung, eine gewinnende Zugfolge von G.□

Man stellt fest, dass keine Freifelder benötigt werden.

2.3.2 Mehrere Farben

Betrachtet man folgendes sortiertes Spiel sieht man direkt, dass der Beweis sich nicht auf mehrere Farben erweitern lässt.



Definition: Ein Spiel G heißt mehrfarbig sortiert, wenn in jedem Stapel genau eine Farbe vorkommt und jeder Stapel sortiert ist.

Satz: Ein mehrfarbig sortiertes Spiel ist gewinnbar.
Sei $\mathbb{S}$ die Menge aller Stapel von G. Wir betrachten $\mathbb{S}' \subseteq \mathbb{S}$ die Menge aller Stapel einer Farbe.
$\mathbb{S}'$ bildet ein einfarbiges sortiertes Spiel. Jede Farbe ist also für sich einzeln gewinnbar, das Spiel ist also insgesamt gewinnbar.

2.3.3 Beliebige Zugfolgen

Satz: Sei G ein einfarbiges sortiertes Spiel. Werden immer maximal große Teilstapel bewegt, also kein Stapel gespalten, so kann die Zugreihenfolge beliebig gewählt werden, um zu gewinnen.
Der Beweis erfolgt über Induktion nach der Anzahl an maximalen sequenziellen Teilstapeln t in einem Spiel G.

Induktionsanfang: t=0: $G_\emptyset$ ist definitionsgemäß gewonnen.

Induktionsvorraussetzung Ein sortiertes Spiel mit t maximalen sequenziellen Teilstapeln sei mit beliebiger Zugfolge gewinnbar.

Induktionsschritt Sei G ein sortiertes Spiel mit t+1 maximalen sequenziellen Teilstapeln. Man betrachte einen beliebigen validen Zug m, der keine Stapel spaltet.
Das Spiel ist weiterhin sortiert, bleibt also gewinnbar.
Man beachte: Durch jeden Zug werden zwei Teilstapel zu einem Stapel vereinigt. Jeder Zug verringert also die Anzahl an maximalen sequenziellen Teilstapeln um 1. Auch beim Ablegen verringert sich die Anzahl der maximalen sequenziellen Teilstapel um 1. $G \circ m$ hat also t Teilstapel, kann also nach Induktionsvoraussetzung mit einer beliebigen Zugfolge gewonnen werden. Da m beliebig gewählt wurde, kann G auch mit einer beliebigen Zugfolge gewonnen werden.

Analog können auch im mehrfarbigen Fall wieder beliebige Züge gewählt werden, solange diese immer nur auf gleichen Farben agieren.

3 Implementierung

3.1 Z3

Z3 ist ein Satisfiability modulo theories (SMT) Solver[7]. SMT-Solver sind eine Erweiterung von SAT-Solvern, die nicht nur boolesche Formeln, sondern auch mit zum Beispiel ganzzahligen oder reellwertigen Eingaben umgehen können. Z3 setzt außerdem Listen, Funktionen und quantifizierte Ausdrücke um.
Die Implementierungen wurden mit Z3py [3], einer Pythonschnittstelle für Z3 realisiert. Die genaue Funktionsweise von Z3 wird gut in Programming Z3 [8] erläutert. Alle Beispiele sind als Code für Z3Py angegeben.

3.1.1 Verwendung von Z3

Als kleines Beispiel der Funktionsweise von Z3 ist hier ein Programm abgebildet, dass ein lineares Gleichungssystem löst:

```
s = Sover
x,y = Ints("x y")
s(x+y > 200, x < 10)
s.check() #test for sat

m = m.model() #get the model

print(m.eval(x)) #get the value of x
```

In dieser Arbeit werden zwei Funktionen von Z3 verwendet. Die Erste ist, dass Z3 feststellt, ob Formeln erfüllbar sind. Die zweite Funktion ist, dass Z3 uns sogar ein Modell, also hier im Beispiel eine Belegung von x und y liefert, sodass alle Constraints erfüllt sind.

3.2 Modellierung des Spiels

3.2.1 Karten

Eine Karte wird als ein Algebraischer Datentyp dargestellt. Für jede Karte wird ihr Rang, ihre Farbe und eine ID gespeichert. Jeder Karte wird außerdem noch ein boolescher Wert zugeordnet, der angibt, ob diese Karte bewegbar ist.

3.2.2 Stapel

Jeder Stapel wird durch ein Objekt der Klasse Pile dargestellt. Im Wesentlichen wird dort eine Python-Liste von Karten gespeichert. Wichtig ist, dass für jede Karte eine eigene Variable erstellt wird und diese nicht als Z3-Array gespeichert werden. In ersten Tests stellte sich diese Variante als deutlich schneller heraus. Vermutlich liegt dies daran, dass alle Arrays total sind, Z3 also ein Modell des Arrays, dass für alle ganzzahligen Schlüssel einen Wert enthält, betrachten muss. Eine weitere Möglichkeit ist, dass Z3 so mit vielen einzelnen Objekten umgeht, anstatt direkt den Wert eines Arrays suchen zu müssen. Somit kann Z3 heuristisch die Reihenfolge, in der die Variablen angepasst werden müssen, anpassen. Der Nachteil dieser Lösung besteht darin, dass so mehr Regeln erstellt werden müssen, da Z3 keine Informationen über die Arrayeigenschaften besitzt, und somit Z3 nicht mit der Indizierung umgehen kann.

Die Karte, die an oberster Stelle eines Stapels liegt, hat immer Index 0. Dies wurde so umgesetzt, da so Regeln gespart werden können. Zum Beispiel kann die Anforderung

"die oberste Karte ist immer bewegbar, außer es ist die leere Karte" durch nur eine Regel umgesetzt werden:

```
M[0] == (A[0]!= NULLCARD)
```

Auch die Regeln, die tatsächlich das Bewegen der Karten umsetzen, werden so deutlich einfacher, da nicht herausgefunden werden muss, an welcher Stelle die oberste Karte liegt.

3.2.3 Spielzüge/Zustände

Jeder Spielzustand ist eine Ansammlung von Stapeln[4] sowie Z3 Arrays von Karten für die Nachziehstapel. Für jeden möglichen Zug muss ein Spielzustand erstellt werden. Die Funktion define_move_rules fügt dann, für zwei aufeinanderfolgende Zustände alle Regeln ein, die kodieren, was ein valider Zug ist.
Folgende Constraints erlauben zum Beispiel einen ablegenden Zug nur, wenn es einen Teilstapel gibt, auf dem Karten bereits als ablegbar markiert wurden.

```
solver.add(Implies(
    move == move_type.clear,
    Or(*[state_0.piles[x]
    .clearability[0]
    for x in range(NUM_PILES)])
))
```

Genauso gibt es auch für das Bewegen von Karten oder das Nachziehen Constraints. Außerdem gibt es Constraints, die dafür zuständig sind, den folgenden Zustand aufzubauen.
Folgende Regel definiert zum Beispiel die obersten Karten in state_1 als die Karten des Nachziehstapels, wenn nachgezogen wird:

```
top_card_rules = [
    If(move == move_type.stock,
        state_1.piles[x].pile[0]
        == state_0.stocks[0].pile[x],
    True
    )
    for x in range(NUM_PILES)]
```

3.3 Optimierungen

Im Folgenden werden einige interessante Optimierungen vorgestellt:

Erste Tests haben ergeben, dass, wenn ein Zug fest definiert wird, der neue Zustand sehr schnell gefunden werden kann. Daher wurden die meisten Optimierungen am Finden der Züge vorgenommen.
Eine wichtige Designentscheidung ist, dass keine Quantoren verwendet werden. Die Dokumentation von Z3 empfiehlt dies auch so, wenn nur endlich viele Objekte betrachtet werden. Stattdessen werden über Python List-Comprehensions mehrere Regeln erstellt. Dies wird gemacht, da das Feststellen der Erfüllbarkeit einer quantifizierten Formel im Allgemeinen nicht entscheidbar ist, Z3 also auch keine gute Suchstrategie implementieren kann. Nachteil wiederum ist, dass dadurch das Erstellen der Regeln deutlich länger dauert.

3.3.1 Kontraposition

Eine Optimierung ist, dass für die Regeln, die angeben, ob ein Zug gültig ist, nicht nur die Regel selbst, sondern auch die Kontraposition der Regel eingefügt wurde. Somit kann Z3 einige Züge schneller für ungültig erklären.

[4] Für ein Standardspiel 10 Stück

3.3.2 Leere Stapel

Eine weitere Optimierung ist die Folgende:

```
Implies(
 state_0.piles[source_pile].pile[0]
 == NULLCARD,
 get_source_pile(move)
 != source_pile)
```

Sie sagt aus: Wenn ein Stapel leer ist, kann kein Zug von diesem Stapel ausgehen. Diese Regel ist eigentlich redundant, da dadurch, dass NULLCARD nicht bewegbar ist, auch bereits keine Züge von leeren Stapeln ausgehen können. Somit müssen vermutlich weniger Züge betrachtet werden, was die Laufzeit verbessert.

3.3.3 Minimaler Index

Außerdem wurde implementiert, dass Züge auf ein Freifeld nur immer auf das Freifeld mit minimalem Index stattfinden. Diese Optimierung unterscheidet sich von allen Anderen: Sie verbietet einige eigentlich zulässige Züge. Will man also nur die Korrektheit von Zügen verifizieren, müssen diese Regel entfernt werden.

3.4 Evaluation

Für die Zeitmessung werden immer zwei Zeiten angegeben: Die Zeit zum Erstellen der Regeln und die Zeit, die Z3 benötigt, um eine Lösung zu suchen. Jeder Test wurde zehnmal ausgeführt und der Durchschnitt der Zeiten gebildet. Für die Zeit, die Z3 benötigt, wird die Zeit, die Z3 beim Aufruf von s.statistics() liefert, verwendet.
Alle Zeiten wurden auf einem Lenovo Laptop mit einer AMD Ryzen 5 5500U CPU ausgeführt.

Als Datensätze wurden Spiele verwendet die in zwei bis fünf Zügen gewonnen werden können. Es wurde immer ein Zug mehr eingefügt als eigentlich zum Gewinnen nötig wäre.
Z3 wurde mit einem Zeitlimit von 10 Minuten konfiguriert. Wurde das Zeitlimit dreimal überschritten, wurden die Tests abgebrochen.

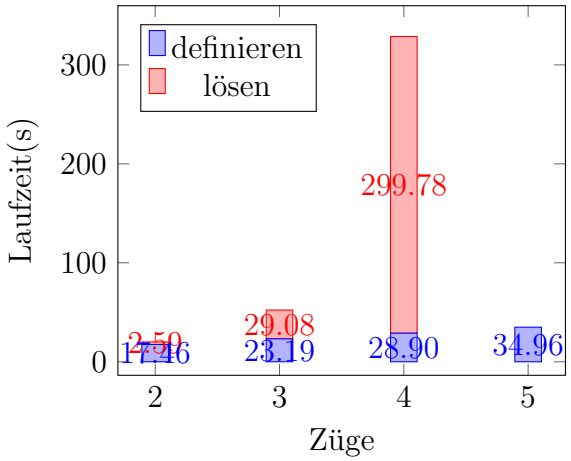


Abbildung 2: Laufzeiten des Lösens von Spielen unterschiedlicher Länge.

Man sieht direkt, dass die Laufzeit bereits für eine sehr geringe Anzahl an Zügen extrem hoch wird und bereits bei fünf Zügen das Zeitlimit von zehn Minuten überschritten wurde. Man sieht außerdem, dass für längere Spiele die Laufzeit zum Erstellen der Klauseln deutlich langsamer wächst als die Laufzeit des Lösers.

Als zweite Messreihe wurde der Effekt einzelner Optimierungen gemessen. Hierfür wurden die Optimierungen alle einzeln deaktiviert und die Zeiten dann gemessen. Alle Messungen fanden auf dem Datensatz mit drei Zügen statt.

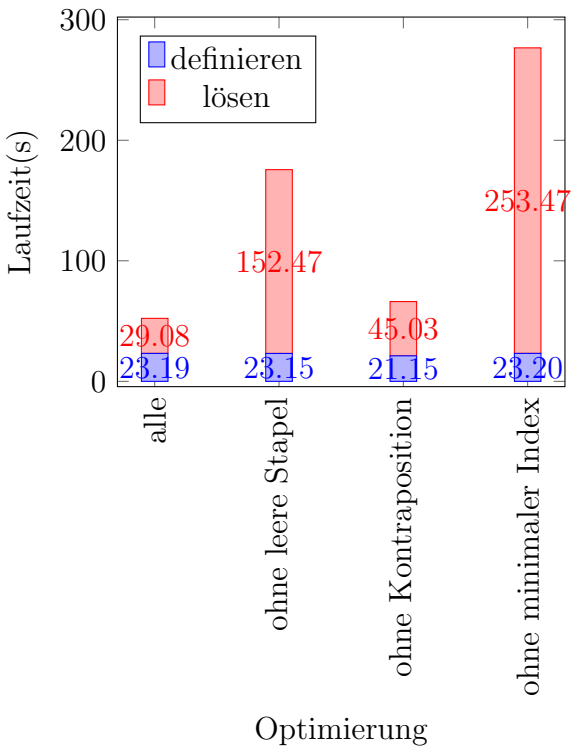


Abbildung 3: Vergleich verschiedener Optimierungen

Zwar ist die Zeit, die zum Erstellen der Regeln benötigt wird ohne die Optimierungen jeweils leicht kürzer, dies wird aber durch die deutlich geringere Zeit zum Lösen wieder deutlich verbessert.
Man sieht sehr gut, dass durch das besondere Behandeln leerer Stapel und die Optimierung "minimaler Index" sehr viel Rechenzeit gespart wird. Dies liegt vermutlich mit daran, dass in den fast gelösten Spielen bereits sehr viele Stapel leer sind, die somit nicht mehr betrachtet werden müssen.

3.4.1 Vergleich mit Kpatience

Als Vergleich dazu wurde noch die Laufzeit des Lösers von Kpatience gemessen. Diese Tests wurden auf einem Desktop-PC mit einer AMD Ryzen 5 3600 CPU ausgeführt. Es wurde die Laufzeit der Methode "Spidersolver2::patsolve" in der Quellcodedatei "./patsolve/spidersolver2.cpp" gemessen. Es wurde der Durchschnitt über 10 Instanzen gebildet. Durchschnittlich benötigte Kpatience für ein neues zufälliges Spiel, das mit c.a 150 Zügen gewonnen werden kann, 1,62 Sekunden, um eine Lösung zu finden.

Zwar ist die gemessene Zeit nicht direkt vergleichbar mit den Zeiten von Z3, da unter anderem eine andere CPU zum Messen der Zeiten verwendet wurde. Dennoch zeigt diese Messung, dass die Implementierung von KPatience deutlich schneller ist als der in dieser Arbeit implementierte Löser.

3.5 Sortierte Spiele

Wie in Kapitel 2.3 gezeigt wurde, ist ein sortiertes Spiel immer gewinnbar. Der Beweis hierfür ist konstruktiv, liefert also direkt die gewinnende Zugfolge. Insbesondere kann diese effizient bestimmt werden.
Auch Kpatience[9] setzt in der Datei "spidersolver2.cpp" innerhalb der Funktion "Deck::getMoves" eine vergleichbare Optimierung um.
In den meisten Spielen sind die letzten Zustände sortierte Spiele. Die Optimierung besteht darin, dass Z3 sortierte Spiele erkennt und für diese dann als Endzustand betrachtet beziehungsweise ab dann alle weiteren Züge als "ordered" festsetzt. Diese Züge müssen somit nicht mehr betrachtet werden.

Für das Finden der letzten Zugfolge könnte dann eine effiziente Python-Implementierung verwendet werden. Dieser Ansatz funktio-

[5] In der Implementierung des Einlesen einer Kpat-Datei findet man noch Codereste, die hierfür verwendet werden könnten

niert sicherlich, wurde aber nicht direkt implementiert.[5]

3.5.1 Evaluation

Als erste Evaluation wurde der Spielzustand aus den vorherigen Messungen, der in 5 Zügen gewonnen werden kann betrachtet. Da dieser Zustand bereits sortiert ist benötigt der Löser nur noch durchschnittlich 1,75 Sekunden um festzustellen, dass das Spiel sortiert ist und somit die Lösung trivial ist. Dies ist eine deutliche Verbesserung zu der vorher gemessenen Zeit.

Daher werden jetzt Spielstände betrachtet, bei denen noch eine gewisse Anzahl an Zügen verbleibt, bis ein sortierter Spielzustand erreicht wird. Es werden wieder die Zeit zum Aufbauen einer Lösung sowie die Laufzeit des Lösers betrachtet. Die Messungen wurden mit Instanzen durchgeführt, die noch 1 bis 4 Züge bis zu einem sortierten Zustand benötigen. Es wurde wieder ein Zeitlimit von 10 Minuten festgelegt.

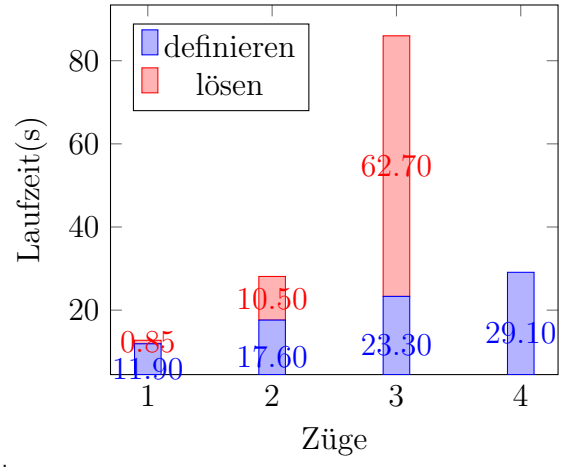


Abbildung 4: Laufzeit des Lösers bis ein sortierter Zustand erreicht wird

Bereits mit vier Zügen wird das Zeitlimit überschritten. Man erkennt wieder denselben, vermutlich exponentiellen, Anstieg der Laufzeit. Somit kann zwar deutlich früher aufgehört werden zu suchen, wodurch zwar insgesamt mehr Züge abgebildet werden können, insgesamt ist die Laufzeit dennoch weiterhin sehr schlecht.

3.6 Endfazit

Als Endfazit muss man leider festhalten, dass das, was implementiert wurde, in keiner praktischen Weise funktioniert und extrem langsam ist. Verantwortlich sind dafür vermutlich zwei Faktoren:
1. Z3 ist nicht dafür gemacht Planungsprobleme, wie zum Beispiel Spider-Solitaire effizient zu lösen.
2. Spider-Solitaire hat in jedem Schritt zu viele mögliche Züge. Dies führt dazu, dass es in jedem Schritt sehr viele möglichen Modelle/Zustände gibt, die Z3 alle betrachten muss. Dies führt dann zu einer sehr hohen Laufzeit von Z3.

3.7 Future Work

Wie man erkennt, benötigt bereits das reine Erstellen der Regeln sehr viel Zeit. Hier ist auch vermutlich noch Einsparpotential. Man kann noch untersuchen, ob mit der C++-Schnittstelle von Z3 das Erstellen deutlich schneller abläuft. Keine Regel hängt von der eigentlichen Spielinstanz ab. Daher könnte man noch untersuchen, ob es möglich ist, die Regeln Vorzugenerieren und dann nur noch einlesen zu müssen, anstatt die Regeln erst direkt vor dem Ausführen des Löser zu generieren.
Der Ansatz, der hier gewählt wurde ist, dass jeder Zug direkt ausgeführt wurde. Es gibt also für jede Kombination aus Quell und

Zielstapel und Stapelhöhe eine eigene Regel, die das Bewegen ausführt. Es gibt also $O(p^2)$ viele Regeln zum Bewegen von Karten, wobei p die Anzahl an Stapeln ist. Man könnte auch untersuchen, ob man das hochheben und ablegen getrennt voneinander betrachten kann. Man müsste dafür die hochgehobenen Karten zwischenspeichern. Somit hätte man nur noch O(k) viele Regeln zum Bewegen von Karten. Dies macht das Erstellen der Regeln sicherlich schneller. Ob dadurch die Suchzeit zunimmt oder abnimmt, bleibt offen. Die Idee hierfür ist abgeleitet daraus, wie in Freecell Stapel bewegt werden.

Man könnte auch untersuchen, ob man mit Makrozügen eine schnellere Suche erzielen kann. Ein Makrozug ist eine Zusammenfassung von häufig hintereinander auftretenden Zügen. Ein Beispiel wäre, dass man das Bewegen einer Karte und das darauffolgende Ablegen eines Stapels in einen Zug zusammengefasst.[6]

In dieser Arbeit wurde nur untersucht, ob man alleine mit Aussagenlogik eine Spider-Solitaire Lösung finden kann. Es kann noch untersucht werden, ob mit der transitiven Hüllenoperation oder der Fixpunktfunktion, die Z3 bereitstellt, ein deutlich besseres Ergebnis erzielt werden könnte. Hierfür müsste dann aber vermutlich eine neue Implementierung geschrieben werden.

Hier wurde nur untersucht, ob man eine große Menge an Regeln bauen kann, die dann direkt die Lösung liefert. Vielleicht könnte man durch mehrfaches Aufrufen von Z3 mit deutlich kleineren Instanzen eine deutlich schnelleres Ergebnis erzielen.

Es kann natürlich immer untersucht werden, ob weitere Optimierungen sinnvoll sind. Man könnte zum Beispiel noch eine aus 4.1.5 "Mehrfarbiges Spalten" abgeleitete Optimierung implementieren.

Des Weiteren könnte man untersuchen, ob der hier implementierte Ansatz so erweitert werden kann, dass entschieden werden kann, ob Spiele "Faire Spiele" oder "k-einfache Spiele" sind. Da diese Π_2^p-Vollständig sind, wird die Laufzeit vermutlich aber noch schlechter werden.

3.7.1 Verwandte Ansätze

Offen bleibt auch, ob man mit anderen Sprachen aus der logischen Programmierung wie Prolog oder Datalog bessere Ergebnisse erzielen kann. Auch die Implementierung als STRIPS-Problem könnte deutlich bessere Ergebnisse erzielen.

Das ursprüngliche Ziel dieser Arbeit Spider-Solitaire mithilfe von SAT-Solvern zu lösen bleibt auch weiterhin offen. Auch hierbei könnte man vielleicht ein deutlich besseres Ergebnis erzielen. Der hier implementierte Löser könnte dabei als Grundlage dienen, indem man alle Regeln in eine SAT-Formel übersetzt.

[6] Der Begriff des Makrozugs stammt aus der Diplomarbeit von Helmert [5] der diese in einem Beweis für Freecell definiert.

3.8 Dateiformate

Um den Löser zu testen, wurden zwei Dateiformate verwendet. Zum einen das von Kpatience bereitgestellte Dateiformat, sowie ein eigenes einfacheres JSON-Dateiformat.

3.8.1 kpat

Das .kpat-Dateiformat basiert auf XML, ist also mithilfe von Bibliotheken einfach einzulesen.
Das .kpat Format stellt alles als Züge dar. Mehrere Kartenbewegungen können in einen Zustand gruppiert werden. Hier ist beispielhaft das Bewegen eines Stapels von 5 bis zum Ass auf den dritten Stapel angefügt. Existieren diese Karten vorher nicht, bedeutet dies, dass die Karten hinzugefügt werden. Ansonsten werden die Karten von ihrem vorherigen Stapel entfernt.

```
<state>
 <move pile="stack2" position="6">
  <card id="1049349" suit="spades" rank="five"/>
  <card id="0983812" suit="spades" rank="four"/>
  <card id="0525059" suit="spades" rank="three"/>
  <card id="3801858" suit="spades" rank="two"/>
  <card id="0066305" suit="spades" rank="ace"/>
 </move>
</state>
```

Abbildung 5: Ausschnitt einer .kpat Datei

Das kpat-Format speichert den Endzustand nicht ab. Auch der Initialzustand ist als Zugfolge implementiert. Der Endzustand ergibt sich dann durch Anwenden der gespeicherten Züge. Um den Zustand zu erhalten, wird beim Einlesen einer kpat Datei erst der Initialzustand gebildet und danach alle Züge auf diesen angewendet.

Eine Eigenheit des Dateiformates sind die IDs. Wählt man diese konsekutiv oder zufällig kann die Datei nicht eingelesen werden, was die Vermutung, dass diese auch Informationen über die Karten enthalten, nahelegt. Die genaue Funktionsweise würde aus dem Code von Kpatience ersichtlich werden. Stattdessen wurde sich dazu entschieden, die IDs an allen Stellen des Programms zu übernehmen.

3.8.2 JSON

Da das .kpat Format sehr komplex ist, wurde außerdem noch ein JSON Format definiert. In diesem wird jedem Stapel, sowie dem Nachziehstapel ein Array zugeordnet. Jede Karte ist ein eigenes JSON-Objekt, das die Schlüssel suit, rank, face_down und id besitzt. Das Programm, dass die .kpat Dateien einliest, generiert eine entsprechende JSON-Datei

4 NP-Vollständigkeit und mehr

In diesem Kapitel wird der Beweis der NP-Vollständig von Spider-Solitaire von Jesse Stern angepasst, sodass dieser auch mit nur einer Farbe möglich ist. An einigen Stellen werden hierfür Dinge, auf die er nicht eingegangen ist, genauer erläutert.
Des Weiteren werden Π_2^p-Vollständigkeitsresultate gezeigt und gezeigt, dass **SPIDER** nicht approximierbar ist.

4.1 SPIDER $\in NP$

Wie bekannt liegt eine Sprache L in der Klasse NP, wenn es einen Verifizierer V mit polynomieller Laufzeit und Zertifikate t polynomieller Länge gibt, sodass:

$$x \in L \iff \exists t \in \{0,1\}^{p(|x|)} : V(x,t) = 1$$

Die folgenden Abschnitte befassen sich ausschließlich damit, dass ein Zertifikat polynomieller Länge existiert. Genauer wird gezeigt, dass jedes gewinnbare Spiel mit einer polynomiellen Anzahl an Zügen gewonnen werden kann. Dass ein Verifizierer existiert, der in Polynomialzeit arbeitet, sieht man dann leicht.

4.1.1 Fehler in Sterns Beweis

Stern argumentiert, dass es nur eine polynomielle Anzahl an Zügen, die nicht direkt rückgängig gemacht werden können, gibt. Diese sind das Aufdecken einer Karte, wenn diese verdeckte Karte nicht zur bewegten Karte passt, sowie das Bewegen eines Teilstapels auf den Ablagestapel. Hier vergisst er das Ziehen vom Nachziehstapel als Aktion.
Er argumentiert weiter, dass jede Karte auf einen Stapel gelegt werden muss, mit dem sie auf den Ablagestapel gelegt werden muss. Er argumentiert dann, dass jede Karte von höchstens $8n - 1$ Karten blockiert wird, die vorher wegbewegt werden müssen. Liegt eine Karte auf dem Stapel, mit dem sie später abgelegt werden kann, so muss sie nicht mehr wegbewegt werden. Somit kann jede der $8n$ Karten nur einmal von höchstens $8n - 1$ Karten blockiert werden, es müssen also höchstens $8n \cdot (8n - 1) \in O(n^2)$ rückgängig machbare Züge ausgeführt werden. Somit kann jedes Spiel mit einer polynomiellen Anzahl an Zügen [7] gewonnen werden.
Der Rest des Beweises würde dann, zusammen damit, dass ein Zug in Polynomialzeit auf einen Zustand angewendet werden kann, aus [5, Lemma 2.10 Plans of polynomial length] folgen. Stern erwähnt dies nicht explizit

Um zu begründen, dass sein Beweis vermutlich fehlerhaft ist, wird in Kapitel 4.6.1 "Hanoi Spider Solitaire" eine Abwandlung von Spider-Solitaire angegeben, die eine exponentielle Anzahl an Zügen zum Gewinnen benötigt. Sein Beweis kann aber genauso auf diese veränderte Variante angewendet werden.
Genauer ist das Problem, dass Stern keine Schranke dafür angibt, dass Karten, während sie von einem Stapel wegbewegt werden, nicht vielleicht exponentiell oft bewegt werden müssen.

Daher wird jetzt ein neuer Beweis dafür gegeben, dass jedes gewinnbare Spiel in einer

[7] Stern spricht von "polynomial time"

polynomiellen Anzahl an Zügen gewonnen werden kann.

4.1.2 Ohne spalten ohne Züge auf Freifelder

Anstatt jetzt direkt für ein beliebiges gewinnbares Spiel zu zeigen, dass dieses immer mit einer polynomiellen Anzahl an Zügen gewinnbar ist, wird sich jetzt schrittweise diesem Resultat genähert.

Lemma: Sei G ein beliebiges Spiel. Sei m ein Zug, der keinen sequenziellen Teilstapel spaltet. $G \circ m$ hat dann gleich viele oder einen einfarbigen maximalen sequenziellen Teilstapel weniger als G.

Beweis: Beim Ablegen eines Stapels verschwindet dieser komplett, die Anzahl an Teilstapeln verringert sich also um 1.
Beim Nachziehen können höchstens Teilstapel wachsen. Da definitionsgemäß jede Karte auf dem Nachziehstapel einen eigenen Teilstapel bildet, verringert sich die Anzahl an Teilstapeln oder bleibt gleich.
Ein anderer, so eingeschränkter Zug muss also einen maximal großen sequenziellen Teilstapel auf einen anderen Teilstapel bewegen. Haben diese die gleiche Farbe werden diese zu einem Teilstapel zusammengeführt, die Anzahl an Teilstapeln nimmt also um Eins ab. Haben Sie unterschiedliche Farben, bleibt die Anzahl an Teilstapeln gleich. Im einfarbigen Fall nimmt die Anzahl an maximalen sequenziellen Teilstapeln also immer ab.

4.1.3 Eine Farbe

In diesem Abschnitt wird jetzt gezeigt, dass die einfarbige Variante in NP liegt.

Es wird dazu folgendes gezeigt: Zwischen zwei Zügen, die maximale sequenzielle Teilstapel bewegen, können jeweils nur eine beschränkte Anzahl an eindeutigen Zuständen liegen.
Ein einfarbig spaltender Zug ist ein Zug, der nur einen Teil eines einfarbigen sequenziellen Teilstapels bewegt und auf einen anderen Stapel legt.
Da es nur eine Farbe gibt, sind Züge, die keine maximalen sequenziellen Teilstapel bewegen, einfarbig spaltende Züge.
Man betrachte alle möglichen spaltenden Züge $a \to b$, nach dem Bewegen eines maximalen Stapels. Führt man jetzt einen beliebigen spaltenden Zug aus, so wird dadurch kein Zug $c \to d$ unmöglich. Die Karte wird lediglich gegebenenfalls auf einen anderen Zielstapel bewegt. Dies liegt daran, dass zwar durch $a \to b$ ein Stapel blockiert wird, dafür aber ein anderer Stapel aufgedeckt wird mit einer Karte des gleichen Rangs.

Man betrachte einen Spielzustand G', der nur durch einfarbiges Spalten aus G entstanden ist. Es wird gezeigt: Jeder Stapel S aus G' kann in höchstens $n-1$ vielen Zügen entstehen.
Sei a der Rang der obersten Karte von S im Zustand G. Sei a' der Rang der obersten Karte von S in G'. Im Worst Case müssen $x := a' - a$ viele Karten auf S gelegt werden. Dies entspricht, dass x Karten alle einzeln auf S gelegt werden müssen. Da alle Züge im Zustand G auch schon möglich waren, müssen keine Stapel, die auf S gelegt werden, gespalten werden. Somit ist es immer möglich, den Stapel S in $n-1$ Zügen aufzubauen.
Da wie oben begründet ein spaltender Zug

keine Züge unmöglich macht kann jeder Stapel einzeln betrachtet werden. Es kann also jeder Stapel in $n-1$ Zügen aufgebaut werden, der Zustand G' kann also in höchstens $p \cdot (n-1)$ Zügen erreicht werden.

Man stellt außerdem fest, dass durch einfarbiges Spalten die Gesamtanzahl an maximalen sequenziellen Teilstapeln gleich bleibt.

Damit kann jetzt gezeigt werden, dass die einfarbige Variante von **SPIDER**, ohne Züge, die Karten auf Freifelder bewegen, mit einer polynomiell beschränkten Anzahl an Zügen gewonnen werden kann.

Wie oben begründet nimmt beim Bewegen maximaler sequenzieller Teilstapel die Anzahl an maximalen sequenziellen Teilstapeln immer ab. Es kann höchstens $\frac{n}{p}$ mal nachgezogen werden. Dabei bleibt die Anzahl an maximalen sequenziellen Teilstapeln schlimmstenfalls gleich.
Da es höchstens $n \cdot k$ maximale sequenzielle Teilstapel gibt, kann also nur $n \cdot k$ oft ein maximaler sequenzieller Teilstapel bewegt werden. Zwischen je zwei dieser Züge gibt es eine spaltende Zugfolge, die höchstens $p \cdot (n-1)$ viele spaltende Züge benötigt. Insgesamt gibt es also schlimmstenfalls $n \cdot p \cdot p \cdot (n-1) \in O(n^2 \cdot p^2)$ viele Züge. Das Spiel ist also mit einer polynomiellen Anzahl an Zügen gewinnbar.

4.1.4 Freifelder

Als Letztes müssen noch Züge, die Karten auf Freifelder bewegen betrachtet werden, da hierbei auch die Anzahl an maximalen sequenziellen Teilstapeln nicht abnimmt. Dazu wird Folgendes begründet:

Sei G ein gewinnbares Spiel mit mehreren Farben. Dann gibt es eine gewinnende Zugfolge, die jede Karte höchstens einmal auf ein Freifeld bewegt.

Beweis: Angenommen in jeder gewinnenden Zugfolge von G gibt es mindestens eine Karte, die zweimal auf ein Freifeld bewegt werden muss.
Sei a eine Karte, die mehrfach auf ein Freifeld bewegt wird. Wird sie zwischen zwei Freifeldern S, S' bewegt, so kann sie auch stattdessen liegengelassen werden und alle Züge, die auf S bzw. S' agieren können jeweils getauscht werden. Somit muss sie nur einmal auf ein Freifeld bewegt werden.
Sei a jetzt eine Karte, die von einem Freifeld auf eine Karte bewegt wird und zu einem späteren Zeitpunkt wieder auf ein Freifeld bewegt wird. Der zweite Zug auf ein Freifeld ist dann ein spaltender Zug. Im nachfolgenden Abschnitt wird gezeigt, dass es auch eine gewinnende Zugfolge ohne diesen Zug gibt.

Da jede Karte nur einmal auf ein Freifeld bewegt werden kann, können somit nur höchstens n zusätzliche Züge entstehen. Somit gibt es immer noch eine gewinnende Zugfolge mit $O(n^2 * p^2)$ Zügen.

4.1.5 Mehrfarbiges Spalten

Man betrachte jetzt Züge, die Stapel spalten und auf einen anderen Stapel legen. Haben Start- und Zielstapel die gleiche Farbe, so ändert sich die Anzahl an maximalen einfarbigen sequenziellen Teilstapeln nicht. Es wurde hierfür bereits begründet, dass diese Züge nur zu polynomiell vielen Zügen führen.

Jetzt werden Züge betrachtet, die einen ein-

farbigen sequenziellen Teilstapel spalten und auf einen Stapel einer anderen Farbe ablegen. Im Folgenden werden diese Züge als mehrfarbig spaltend bezeichnet. Bei diesen Zügen entsteht ein zusätzlicher maximaler einfarbiger sequenzieller Teilstapel.

Satz: Sei G ein gewinnbares Spiel. Dann kann G ohne mehrfarbiges Spalten und spalten auf Freifelder gewonnen werden.

Der Beweis erfolgt über Induktion nach Anzahl an maximalen einfarbigen sequenziellen Teilstapel t in einem Spiel. Der Beweis basiert darauf, dass jedes gewinnbare Spiel G als inverse Zugfolge M^{-1} mit $G = G_\emptyset \circ M^{-1}$ dargestellt werden kann.

Induktionsanfang: $G_\emptyset$ besitzt Null maximale einfarbige Teilstapel. Es ist definitionsgemäß gewonnen.
Gewinnbare Spiele mit einem maximalen einfarbigen Teilstapel enthalten nur einen Stapel, $S = (1, \cdots, n)$ dieser kann direkt abgelegt werden. Diese Spiele können also ohne spaltende Züge gewonnen werden.

Induktionsvoraussetzung: Ein gewinnbares Spiel mit t oder weniger maximalen sequenziellen einfarbigen Teilstapeln kann ohne mehrfarbig spaltende Züge gewonnen werden.

Induktionsschritt: Sei G ein gewinnbares Spiel mit t vielen maximalen einfarbigen sequenziellen Teilstapeln.
Auf G wird jetzt einen beliebiger inverser Zug m^{-1} angewendet, für den gilt, dass m in $G \circ m^{-1}$ ein gültiger Zug ist. Jetzt werden alle möglichen Zugarten, die m sein kann, betrachtet:

mehrfarbig spaltend: Wenn m^{-1} einen maximalen[8] einfarbigen sequenziellen Teilstapel auf die gleiche Farbe bewegt, so ist m ein mehrfarbig spaltender Zug oder ein auf ein Freifeld spaltender Zug. In $G' := G \circ m^{-1}$ ist die Anzahl an maximalen einfarbigen sequenziellen Teilstapeln also geringer als in G. Somit kann $G \circ m^{-1}$ nach Induktionsvoraussetzung auch ohne spaltende Züge gewonnen werden.

Für ein anderes m^{-1} ist m kein mehrfarbig spaltender Zug

Nach Induktionsvoraussetzung kann G ohne mehrfarbig spaltende Züge gewonnen werden. Sei M eine gewinnende Zugfolge von G, die nie mehrfarbig spaltet. $m \circ M$ ist eine gewinnende Zugfolge für G'.

Der Zug m kann jetzt entweder noch einfarbig spalten, auf ein Freifeld spalten, einen maximalen Teilstapel bewegen, nachziehen oder einen vollen Stapel ablegen.
Spalten: Spaltet m einfarbig oder auf ein Freifeld, so bleibt die Anzahl an maximalen Teilstapeln in G' gleich. Nach Induktionsvoraussetzung kann G' also ohne mehrfarbig spaltende Züge gewonnen werden.
Bewegen: Bewegt m einen maximalen Teilstapel auf einen Stapel gleicher Farbe, so nimmt die Anzahl an maximalen einfarbigen Teilstapeln ab. Dies bedeutet also, dass G' einen maximalen mehrfarbigen Teilstapel mehr als G besitzt. $m \circ M$ ist dann eine gewinnende Zugfolge von G', die nicht spaltet.

[8] sonst wäre m einfarbig spaltend

Bewegt m auf eine andere Farbe oder ein Freifeld, so bleibt die Anzahl an Teilstapeln konstant. Nach der Induktionsvoraussetzung kann das Spiel dann weiterhin ohne Spalten gewonnen werden.
Nachziehen: Ist m ein Nachziehen, so ist m^{-1} das wegnehmen auf den Nachziehstapel. Es kann nur weggenommen werden, wenn G keinen leeren Stapel besitzt. definitionsgemäß bildet jede Karte auf dem Nachziehstapel einen eigenen Teilstapel. Durch das inverse Nachziehen werden somit bis zu p Teilstapel hinzugefügt. Somit ist G' ein Spiel mit gleich vielen oder mehr Teilstapeln. $m \circ M$ ist eine gewinnende Zugfolge ohne mehrfarbiges spalten von G'
Ablegen: Ist m das Ablegen eines Stapels, so fügt m^{-1} einen vollen einfarbigen sequenziellen Teilstapel in das Spiel hinzu. Somit hat $G \circ m^{-1}$ $t+1$ maximale sequenzielle einfarbige Teilstapel. $m \circ M$ ist eine gewinnende Zugfolge für G', die ohne spaltende Züge auskommt
Es fehlt noch die Begründung, dass auch wirklich alle gewinnbaren Spiele so abgedeckt werden: Sei G ein gewinnbares Spiel mit $t+1$ maximalen sequenziellen Teilstapeln. Es gibt also eine gewinnende Zugfolge M, sodass $G \circ M = G_\emptyset$. G kann also aus $G_\emptyset$ aufgebaut werden, indem alle inversen Züge von M angewendet werden. Somit kann jedes gewinnbare Spiel ohne mehrfarbiges Spalten gewonnen werden. □

4.1.6 Bedeutung als Gewinnstrategie

Das, was hier bewiesen wurde, begründet erstmal, dass um ein Spiel zu gewinnen nie mehrfarbig gespalten werden muss.
Außerdem lässt sich hieraus eine Strategie, die Spieler:innen des Spiels vermutlich instinktiv genauso spielen würden ableiten. Dieser Beweis begründet, warum die Gewinnstrategie: "versuche maximal große einfarbige Stapel zu Bauen" [6] tatsächlich eine sehr effektive Strategie ist, da eben ein großer Stapel nie in zwei kleinere, mehrfarbige Stapel gespalten werden muss.

4.1.7 Mehrfarbiger Fall

Als letzte Art von Zug muss das Bewegen eines maximalen sequenziellen Teilstapels auf einen Stapel anderer Farbe oder ein Freifeld betrachtet werden. Leider ist hierfür im Rahmen dieser Arbeit kein Beweis gefunden worden, der die Anzahl dieser Züge einschränkt. Außerdem muss noch begründet werden, dass beim Ausschließen des mehrfarbigen Spaltens weiterhin eine polynomielle Zugfolge entsteht. Der Autor geht trotzdem davon aus, dass auch SPIDER mit mehr Farben in NP liegt.

4.1.8 Verifizierer

Die Existenz eines Verifizierers folgt jetzt direkt aus [5, Lemma 2.10 Plans of polynomial length]

Alternativ kann auch direkt darüber argumentiert werden, dass jeder Zug als Tupel aus (Quellstapel, Zielstapel, Stapelhöhe, Zugtyp) kodiert werden kann, wobei die Kodierungslänge der Quell, Zielstapel und Höhe logarithmisch beschränkt ist. Es gibt nur 3 Zugtypen: Nachziehen, Ablegen und ein normaler Zug. Somit erhält man direkt ein Zertifikat polynomieller Länge als Aneinan-

derreihung von Zügen.
Ein Verifizierer muss dann nur polynomiell viele Züge lesen, diese auf Gültigkeit prüfen und auf den Zustand anwenden. Das Prüfen auf Gültigkeit ist ein einfacher Vergleich der untersten bewegten Karte mit der Karte auf dem Zielstapel. Jedes anwenden entspricht "Stapelhöhe" viele Lösch- beziehungsweise Einfügeoperationen auf Zwei stapeln, die durch Darstellung der Stapel als Arrays jeweils in O(n) möglich sind. Bei maximalem Rang n kann somit jeder Zug in $O(n^2)$ ausgeführt werden. Analog können Nachziehstapel und das Weglegen einer Karte implementiert werden. Somit erhält man bei Zertifikatslänge von t Zügen eine Laufzeit von $O(t \cdot n^2)$, was polynomiell in der Eingabegröße ist.

Als Letztes folgt eine weniger formale, mit dem vorausgegangenen Kapitel kürzere, aber dafür etwas ungewöhnliche Variante eines Verifizierers
Eine .kpat Datei, die als Initialzustand die Probleminstanz enthält, ist eine mögliche Kodierung eines Zertifikates. Der Algorithmus zum Einlesen einer .kpat Datei läuft in Polynomialzeit. Er kann so erweitert werden, dass er prüft, ob die Züge valide sind. Am Ende kann effizient überprüft werden, ob alle Stapel leer sind.

Somit liegt SPIDER mit einer Farbe in NP.

4.2 Weniger Farben

Der Beweis der NP-Vollständigkeit mit vier Farben, wie er von Jesse Stern [12]geführt wurde, kann fast direkt auf lediglich eine Farbe übertragen werden. Im Folgenden wird erläutert, wie der Beweis auf zwei Decks einer Farbe übertragen werden kann. Es wird davon ausgegangen, dass der/die Leser:in den Originalbeweis grob kennt.

4.2.1 ONE-COLOR-SPIDER ist NP-Schwer

Der Beweis der NP-Schwere erfolgt durch Reduktion von **3-SAT**. Der Beweis ist eine Abwandlung des Beweises von Jesse Stern.[12] Sei $i \in \mathbb{N}$ die Anzahl an Variablen, $k \in \mathbb{N}$ die Anzahl an Klausen einer **3-SAT** Formel.

Im Folgenden wurde sich entschieden, die Nummerierung etwas weniger formal, dafür aber verständlicher als die in Sterns Paper zu wählen. Die Karten werden in Bereiche aufgeteilt. Fügt man Karten in einen Bereich hinzu, entspricht dies gleichzeitig einer Umnummerierung der folgenden Bereiche. Das Hinzufügen einer Karte t meint immer, dass die Karte ♠t zweimal im Spiel vorkommt. Zwischen zwei Bereichen liegen immer zwei Karten Puffer, damit zum Beispiel die Variablenkarten nicht auf die Karten der Klauseln gelegt werden können. Die Bereiche sind im folgenden:
[Literalauswahl] $[v_1]$ $[\neg v_1]$ $\cdots$ $[\neg v_i]$ [Klausel 1] $\cdots$ [Klausel k] [Sperrkarte C_1]$\cdots$[Sperrkarte C_k] [Reststapel][Endkarten]

Um den Beweis zu verstehen, ist es hilfreich, sich die einzelnen Teile immer bildlich vorzustellen. Dazu dient folgende Spielinstanz, die die 3-SAT Formel

$$(v_1 \vee v_2 \vee v_3) \wedge (\neg v_1 \vee v_2 \vee v_3) \wedge (\neg v_1 \vee \neg v_2 \vee \neg v_3)$$

darstellt. Den Leser:innen wird empfohlen, beim Nachvollziehen des Beweises auf folgendes Bild zu schauen:

Literalauswahlstapel:

	$v1$	$\neg v1$	v_2	$\neg v_2$	v_3	$\neg v_3$
♠7	♠11	♠17	♠23	♠27	♠33	♠36
	♠5	♠5	♠3	♠3	♠1	♠1
	♠6	♠6	♠4	♠4	♠2	♠2

Klauseln:

$v1$	$v2$	$v3$		
♠40	♠40	♠44	♠63	♠63
♠9	♠19	♠29	♠39	♠43
♠10	♠20	♠30		

$\neg v1$	$v2$	$v3$		
♠48	♠48	♠52	♠67	♠67
♠13	♠21	♠31	♠47	♠51
♠14	♠22	♠32		

$\neg v1$	$\neg v2$	$\neg v3$		
♠56	♠56	♠59	♠71	♠71
♠15	♠25	♠34	♠55	♠58
♠16	♠26	♠35		

Großer Stapel	Restestapel		Endstapel	
...	...	♠74	♠77	♠77
♠7	♠74			
...				
♠70				
♠66				
♠62				

Literalauswahlstapel: Hier kommt jede Karte genau zweimal vor. Daher wird die Auswahl der Variablen analog zu Sterns Beweis umgesetzt. Pro Variable v_i werden zwei Karten zur Variablenauswahl hinzugefügt. Am Ende wird noch eine weitere Karte zur Kartenauswahl hinzugefügt. Die höchstwertige Karte des Literalauswahlbereichs heißt "initiale Literalauswahlkarte".

Jedes positive sowie negative Literal hat einen eigenen Bereich. Es wird mindestens eine Karte eingefügt. Für jedes Vorkommen von v_i in einer Klausel werden zwei Karten zu ihrem jeweiligen Bereich hinzugefügt.

Klauseln: Ersetzt man bei den Klauseln einfach alles durch eine Farbe, erhält man zum Beispiel folgende Stapel:

$v1$	$v2$	$v3$	
♠61	♠61	♠61	♠43
♠44	♠44	♠44	
♠13	♠25	♠35	
♠14	♠26	♠36	

Wie man direkt erkennt, ist hier das Problem, dass die Karte ♠44 beziehungsweise ♠61 dreimal vergeben wurde, obwohl sie nur zwei Mal zur Verfügung steht. Dies kann man beheben, indem man für jede Klausel mehr Karten einfügt. Jede Klausel hat einen eigenen Sperrkartenbereich, in dem genau zwei Karten liegen. Die niederwertigere liegt an oberster Stelle auf dem großen Stapel, die höherwertige wird in den Karten der Klausel selbst verwendet.
Für die Klausel $v1 \vee v2 \vee v3$ erhält man dann zum Beispiel folgendes Gadget:

$v1$	$v2$	$v3$		
♠40	♠40	♠44	♠63	♠63
♠9	♠19	♠29	♠39	♠43
♠10	♠20	♠30		

Pro Klausel werden also sechs Karten im Klauselbereich hinzugefügt. Sei i der Wert der ersten Karte des Bereichs. Auf den ersten und zweiten Stapel liegt unten die Karte $i + 1$ auf dem dritten $i + 5$. Der 4-te Stapel enthält die Karten i, sowie eine der Sperrkarten, der fünfte die Karten $i + 4$ sowie das zweite Vorkommen der Sperrkarte.

Großer Stapel:
In seinem Beweis geht Stern nicht genau darauf ein, wie der große Stapel angeordnet sein muss. Er sagt lediglich, wie die obersten Karten aussehen müssen. Daher folgt hier jetzt nochmal, nur für eine Farbe, wie die Karten angeordnet sein müssen, inklusive ausführlicher Begründung. Anstatt nur einem großen Stapel wird hier einen großen Stapel und einen Restestapel eingefügt
Im Bereich des Reststapels gibt es genau eine Karte. Eine dieser Karten liegt an oberster Stelle des Restestapels. Die zweite Karte liegt auf einem ansonsten leeren Stapel.

Es gibt genau eine Endkarte. Beide Endkarten liegen jeweils auf ansonsten leeren Stapeln.

Die niederwertige Sperrkarte jeder Klausel liegt an oberster Stelle des großen Stapels. Direkt unter den Sperrkarten liegen die Trennkarten zwischen Sperrkarten und den Endkarten, die nicht verwendeten Sperrkarten und die Trennkarten zum Bereich des Restestapels, in aufsteigender Reihenfolge.

Unter dieser wird das zweite Vorkommen der initialen Literalauswahlkarte platziert. Darunter liegen alle bisher nicht verwendeten Karten in absteigender Reihenfolge.

Restestapel: Auf dem Restestapel liegt an oberster Stelle eine Karte des Restestapelbereichs. Darunter liegt die zweite Version aller Karten, von denen kein Vorkommen verwendet wurde, in absteigender Reihenfolge sortiert.

4.2.2 Lemma zum großen Stapel

Eine aus der Reduktion stammendes Spiel ist gewinnbar $\iff$ Alle Sperrkarten können vom großen Stapel wegbewegt werden.
" $\implies$ "
Um das Spiel zu gewinnen, müssen die Karten mit Rang n-1, auf die Karten mit Rang n bewegt werden. Die Karte mit Rang n-1 ist eine Trennkarte zwischen Endkarte und Sperrkarten. Diese werden von den Sperrkarten blockiert, somit müssen alle Sperrkarten vom großen Stapel wegbewegt werden.

" $\impliedby$ " Hat man alle Sperrkarten vom großen Stapel wegbewegt, so liegen dort als Nächstes die Trennkarten zwischen den Sperrkarten, Restestapel und nicht die nicht verwendete Restekarte. Die Karten liegen in aufsteigender Reihenfolge sortiert. Die zweithöchste Karte liegt also jetzt oben auf dem großen Stapel und kann somit auf die höchstwertige Karte gelegt werden. Dach können die Karten n-2 bewegt werden usw. Die Sperrkarten können somit auch alle auf die Endstapel gelegt werden.
Man erhält durch die zweite Karte des Restestapels ein Freifeld.
Jetzt kann die zweite initiale Literalauswahlkarte auf das Freifeld der zweiten Karte des Restestapels bewegt werden.
Die nicht gewählten Literalauswahlkarten können jetzt auf diese Karte bewegt werden.
In jeder Klausel können jetzt alle Karten, die zu den Literalbereichen gehören, entfernt werden. Die Sortierung der Literalstapel bleibt erhalten.

In den Klausen sind jetzt alle Literalkarten entfernt. Somit kann das zweite Feld für die Sperrkarte freigeräumt werden.
Der Restestapel und große Stapel sind jetzt bereits sortiert.

Das Spiel ist jetzt ein sortiertes Spiel und damit gewinnbar.

In Sterns Beweis erhält man kein mehrfarbig sortiertes Spiel. Somit kann dieser Beweis auch nicht direkt auf seine Beweisidee übertragen werden. Ob seine Konstruktion dennoch gilt, wurde nicht genauer geprüft.

4.2.3 Beweis der Reduktion

3-SAT Formel erfüllbar $\implies$ Spiel gewinnbar

Wie oben gezeigt wurde, ist das Spiel nur gewinnbar, wenn man die obersten Karten vom großen Stapel wegbewegen kann.
Man stellt fest: Die erfüllbare Variablenbelegung kann durch Bewegen von Karten ausgewählt werden.

Der einzige Weg, wie man die Sperrkarten wegräumen kann, ist, indem man in den Gadgets der Klauseln Karten bewegt. Für jedes Literal, das die Klausel erfüllt, kann

eine der unteren Karten freigespielt werden. Durch Bewegen eines der beiden Gadgets auf dem 4. bzw. 5. Stapel kann diese wegbewegt werden und dadurch eine Sperrkarte entfernt werden.
Außerdem stellt man fest: Die nicht bewegten Karten des Gadgets (im Beispiel 49 und 48) können auch auf das Freifeld bewegt werden. Wir die Klausel von mindestens zwei Variablen erfüllt, kann dadurch ein neues Freifeld geschaffen werden. Die gesamte Anzahl an Freifeldern bleibt jedoch gleich.
Somit können, wenn jede Klausel erfüllt ist, die obersten Karten vom großen Stapel wegbewegt werden, das Spiel ist also gewinnbar

" $\Leftarrow$ "

Wenn das Spiel gewinnbar ist, können die obersten Karten vom großen Stapel entfernt werden. Dies ist aber nur möglich, wenn für jede Klausel ein der Sperrkarten freigeräumt wurde. Dies ist nur möglich, wenn die Karten eines Literals wegbewegt wurden. Dies wiederum ist nur möglich, wenn die Auswahlkarten entsprechend bewegt wurden. Man erhält also aus den Auswahlkarten eine erfüllende Belegung.

Laufzeit

Stern geht an keiner Stelle darauf ein, dass die Laufzeit seiner Reduktion polynomiell ist. Auch hier wird dies nur knapp begründet.
Betrachtet man sich die Reduktion, stellt man fest, dass pro Literal und pro Klausel jeweils nur eine konstante Anzahl an Karten eingefügt werden muss.
Die Literalauswahlstapel können in linearer Zeit aufgebaut werden.

Jede Klausel kann dann aufgebaut werden, indem für jedes Literal einen Zähler erstellt. Erstellt man jetzt eine Klausel, so muss nur geprüft werden, wo der Zähler steht. Somit können die entsprechenden Karten für die Literale gebildet werden. Die restlichen Karten können dann in Abhängigkeit der Anzahl der bereits bearbeiteten Klauseln aufgebaut werden.

Indem man alle bereits verwendeten Karten in einer Hashmap verwaltet, kann man den großen Stapel effizient aufbauen.

Somit ist Spider-Solitaire mit einer Farbe NP-Vollständig. □

4.3 Faire Spiele

Für diesen Abschnitt ist der Unterschied zwischen verdeckten und nicht spielbaren Karten sehr wichtig. Eine nicht spielbare Karte, ist eine Karte, von der der Spieler weiß, welchen Rang und welche Farbe sie hat, er sie aber nicht bewegen kann. Bei einer verdeckten Karte weiß der Spieler nicht, welchen Wert die Karte hat. Genauer kann der Spieler zwar darüber, dass er weiß, welche Karten bereits aufgedeckt im Spiel liegen wissen, welche Karten möglich sind. Er kann aber keine Aussage über die Permutation der verdeckten Karten treffen.
Verdeckte Karten können direkt im Startzustand spielbar sein, wenn über ihnen keine anderen Karten liegen.

Ein Spiel heißt fair, wenn es unabhängig von der Permutation der verdeckten Karten gewinnbar ist. Idee dieser Definition ist es, dass hier der Spieler nicht raten muss, wie die verdeckten Karten liegen. Bei einem fairen Spiel weiß der Spieler, dass er immer eine Lösung finden kann. Die Menge aller fairen Spiele wird als **FAIR** bezeichnet.

4.3.1 FAIR $\in \Pi_2^p$

Die Polynomialzeithierachie[1, Kapitel 5] ist eine Erweiterung der Klasse NP. **FAIR** liegt in Π_2^p, wenn es zwei Zertifikate polynomieller Länge, u_1, u_2, sowie einen Verifizierer M mit polynomieller Laufzeit gibt, sodass:

$$x \in FAIR \Leftrightarrow \forall u_1 \exists u_2 : M(x, u_1, u_2) = 1$$

Mit der Umformulierung: "Für alle Permutationen der verdeckten Karten, gibt es eine Zugfolge, mit der das Spiel gelöst wird." ist die Zugehörigkeit zur Klasse Π_2^pfast Trivial. Das erste Zertifikat besteht aus einer Permutation der verdeckten Karten. Da es höchsten $O(n)$ verdeckte Karten (also alle Karten sind verdeckt) gibt, ist das Zertifikat auch nur polynomiell lang.
Das zweite Zertifikat ist eine Zugfolge. Wie in 4.1 gezeigt wurde, besitzt dieses Zertifikat polynomielle Länge und kann effizient verifiziert werden.
Daher gilt **FAIR** $\in \Pi_2^p$

4.3.2 Expandieren von Freifeldern

Für den Beweis wird erst folgendes Gadget konstruiert, dass erlaubt k viele Freifelder zu schaffen, wenn eine Karte x wegbewegt wird. Die Karte x heißt Aktivierungskarte des Gadgets. Sie kann frei gewählt werden. Für dieses Gadget wird ein eigener Kartenbereich mit $2 \cdot k + 1$ Karten hinzugefügt.

♠20	♠18	♠16	$\cdots$	♠4
♠x	♠19	♠17	$\cdots$	♠5

Abbildung 6: Gadget zum expandieren eines Freifelds

Man erkennt in der Abbildung direkt: Wird die Karte $\spadesuit x$ auf einen anderen Stapel bewegt, so können alle Felder des Gadgets freigeräumt werden.
Wenn es ansonsten keine Freifelder in dem Spiel gibt, können die Karten auch nur bewegt werden, wenn die Karte $\spadesuit x$ bewegt wurde.
Die Reduktion aus 4.2.1 erschafft keine Freifelder, solange es noch Sperrkarten auf dem großen Stapel gibt. Liegt eine "20" unter den Sperrkarten, aber über dem sortierten Teil des großen Stapels, kann wieder ein sortiertes Spiel erreicht werden. Das "Lemma zum Großen Stapel" gilt also weiterhin.

4.3.3 FAIR ist Π_2^p-Vollständig

Um die Π_2^p-Vollständigkeit von **FAIR** zu zeigen, wird im Folgenden das Π_2^p-Vollständige Problem Π_2^p-SAT auf **FAIR** reduziert. Π_2^p-SAT ist die Menge aller wahren, quantifizierten 3-SAT Formeln $\Phi(x_1, x_2, \cdots, x_n)$ der Form:

$$\forall x_1, x_2, \cdots x_t \exists x_{t+1} \cdots x_n : \Phi(x_1, x_2, \cdots, x_n)$$

Für die Reduktion wird die Reduktion aus 4.2.1 angepasst.
Die mit Existenzquantor versehen Variablen werden wie gehabt behandelt.
Die mit Allquantor quantifizierten Variablen werden anders behandelt.

Auch die mit Allquantor quantifizierten Karten haben einen Bereich, in dem initial eine Karte liegt. Analog zu den existenzquantifizierten Karten werden pro Vorkommen in Klauseln zwei Karten hinzugefügt. Die Gadgets der Klauseln werden wie in der NP-Vollständigen Variante mit diesen Karten aufgebaut.

Grobe Idee: Die Grundidee ist jetzt diese: Jedes positive und jedes negative allquantifizierte Literal erhält eine Auswahlkarte. Alle Auswahlkarten werden verdeckt hingelegt, sodass t viele aufgedeckt werden können.
Ist von jeder Variable genau eine Auswahlkarte aufgedeckt, so wird dadurch eine Variablenbelegung kodiert und das Spiel muss analog zum NP-Vollständigkeitsbeweis genau dann gewinnbar sein, wenn die so kodierte Belegung für die SAT-Formel eine erfüllende Belegung ist.
Ist für eine Variable sowohl die Auswahlkarte des positiven als auch die des negativen Literals aufgedeckt, so wird durch die Auswahlkarten keine Belegung der Variablen kodiert. Das Spiel ist dann so konstruiert, dass es trivial gewinnbar ist. Dies wird so gemacht, da das Spiel für alle Permutationen der verdeckten Karten gewinnbar sein muss, also auch solche, die keine Variablenbelegung kodieren.

Für jedes allquantifizierte Literal v_i und $\neg v_i$ wird ein neuer Kartenbereich eingefügt. In ihm liegen zwei Karten.
Die höherwertige dieser Karten ist ihre Auswahlkarte. Mit den Auswahlkarten aller Variablen werden t viele, zwei Karten hohe Stapel von verdeckten Karten aufgebaut. In einer spezifischen Anordnung der verdeckten Karten liegen also genau t viele Karten an oberer stelle und t viele Karten an unterer Stelle.
Liegt die Auswahlkarte eines Literals oben, so bedeutet das, dass dieses ausgewählt wurde.
Liegen sowohl die Auswahlkarte von v_i als auch $\neg v_i$ für ein i an oberer Stelle, so ist dies eine unzulässige Anordnung der Karten
Für jede Variable wird außerdem ein wie in

4.3.2 beschriebenes Gadget mit k vielen Freifeldern eingefügt.

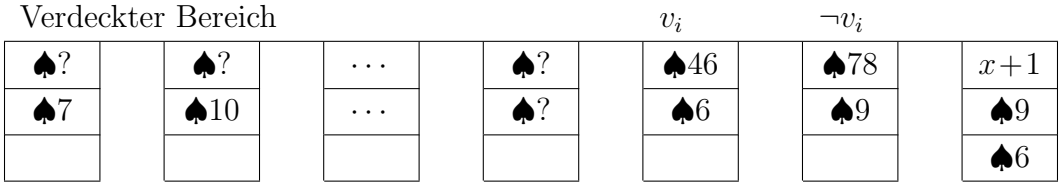


Abbildung 7: Verdeckter Stapel und gadget für eine Variable, unzulässige anordnung

Pro Literal wurde ein Stapel eingefügt. Hier kann, analog zur Kartenauswahl die Karte freigeräumt werden, wenn die Auswahlkarte oben liegt und die entsprechenden Klauseln gewählt wurden.

Der letzte Stapel kann bei einer unzulässigen Anordnung freigeräumt werden. Die Karte $x+1$ ist aus dem Aktivierungsbereich für ein aus 4.3.2 beschriebene Gadget. Es können dann k viele Freifelder geschaffen werden.

4.3.4 Beweis

Sei zunächst die quantifizierte Formel wahr. Jetzt muss begründet werden, dass das Spiel unabhängig der Anordnung der verdeckten Karten gewinnbar ist.

Zuerst werden unzulässige Anordnung, also Anordnungen, in der sowohl die Auswahlkarten v_i als auch $\neg v_i$ an oberer Stelle der verdeckten Karten liegen, betrachtet. Dadurch, dass beide Auswahlkarten an oberer Stelle liegen, kann, wie oben in der Abbildung erkennbar, die Karte x+1 freigeräumt werden. Dadurch kann die Aktivierungskarte x eines Gadgets aus 4.3.2 auf die Karte x+1 geräumt werden, wodurch k viele Freifelder entstehen. Auf diese Freifelder können dann alle Sperrkarten gelegt werden. Nach 4.2.2 "Lemma zum großen Stapel" ist das Spiel dann gewinnbar.

Kodieren die verdeckten Karten eine zulässige Belegung, liegt also für jedes v_i genau nur die Auswahlkarte des positiven oder negativen Literals an oberer Stelle, so entspricht dies genau der Auswahl einer spezifischen Variablenbelegung der Allquantifizierten Variablen. Analog zu 4.2.1 "ONE-COLOR-SPIDER ist NP-Schwer" ist das Spiel also gewinnbar. Die existenzquantifizierten Variablen werden dabei wie gehabt gehandhabt.
Sei jetzt das Spiel fair. Insbesondere ist das Spiel dann für jede Anordnung der verdeckten Karten, die eine Variablenbelegung kodiert, gewinnbar. Alle Variablenbelegung der allquantifizierten Variablen werden auch kodiert. Für eine spezifische Kartenanordnung lässt sich dann auch analog zu 4.2.1 aus einer gewinnenden Zugfolge eine erfüllbare Belegung rekonstruieren. Da alle Variablenbelegungen kodiert werden, ist die quantifizierte Formel also wahr.
Da **FAIR** in Π_2^p liegt und Π_2^p-Schwer ist es Π_2^p-Vollständig.

4.4 k-einfache Spiele

Ein Spiel heißt k-einfach, wenn unabhängig der ersten k Züge das Spiel immer noch gewonnen werden kann.
Für den Spieler bedeutet dies, dass er sich nicht schnell "verbauen" kann und das Spiel daher erst im späten Spielverlauf verloren werden kann. Spielt der Spieler mit der Rückgängigtaste bedeutet dies dann auch, dass der Spieler nie ganz bis zum Anfang zurückgehen muss.

Eine Instanz eines k-einfaches Spiel ist ein Paar (G,k) aus Spielzustand und einer Zahl k.

4.4.1 k-Einfach $\in \Pi_2^p$

Die Zugehörigkeit zur Klasse Π_2^p wird über Angabe eines Verifizierers gezeigt.
Man teilt eine Zugfolge in zwei Teile auf. Die ersten k Züge werden in Zertifikat u_1 , die restlichen in u_2 kodiert.
Der Verifizierer geht jetzt wie folgt vor:

1. Prüfe, ob u_1 genau k Züge kodiert.

2. Prüfe, ob $u_1 \circ u_2$ eine gewinnende Zugfolge kodiert.

Werden beide Bedingungen erfüllt, so wird das Zertifikat akzeptiert. Wenn es für alle Zertifikate u_1 ein Zertifikat u_2 gibt, sodass $u_1 \circ u_2$ eine gewinnende Zugfolge ist, dann ist das Spiel k-einfach.
Bedingung 2 ist nichts anderes als ein Verifizierer für die NP-Vollständigen Varianten. Somit hat der Verifizierer polynomielle Laufzeit. Wie in Kapitel 4.1 gezeigt, hat das Zertifikat u_2 polynomielle Länge.

4.4.2 k-Einfach ist Π_2^p-Schwer

Um die Π_2^p-Schwere zu zeigen wird wieder das Π_2^p-Vollständige Problem Π_2^p-SAT auf **k-Einfach** reduziert.

Für die Reduktion wird wieder eine abgewandelte Version der Reduktion $\mathbf{3-SAT} \leq_p$ **SPIDER** aus Kapitel 4.2.1 angegeben.

Seien im Folgenden $v_1 \cdots v_t$ die mit Allquantor quantifizierten Variablen.
Die neue Reduktion sieht wie folgt aus:
Zuerst wird wie in 4.2.1 eine SPIDER-Instanz generiert. Hierbei wird darauf geachtet, dass $v_1 \cdots v_t$ die höchstwertigen Literalauswahlkarten erhalten.
Sei s die Anzahl an Stapeln in der reduzierten Instanz. Man fügt jetzt noch $2 \cdot (s - 2n)$ Karten, wobei n die Gesamtanzahl an Variablen ist, im Bereich der Variablenauswahl hinzu.
Die neue hinzugefügten Karten werden jetzt wie folgt über allen Stapeln außer den Literalauswahlstapeln platziert. Damit wird dafür gesorgt, dass am Anfang nur die Literalauswahlstapel bewegt werden können. Alle anderen Stapel sind vorerst durch diese Karten blockiert. Erst, wenn alle Variablen ausgewählt wurden, können die anderen Stapel freigeräumt werden.

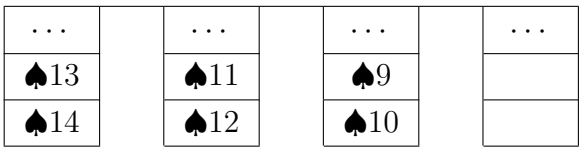


Abbildung 8: Beispiel wie die Stapel umgebaut werden

Das Spiel muss 2t einfach sein. t ist die Anzahl an allquantifizierten Variablen.

Laufzeit: In der ursprünglichen Reduktion werden nur polynomiell viele Stapel hinzugefügt. Somit werden jetzt auch noch nur polynomiell viele Karten eingefügt. Die restliche Laufzeit ist dann genau die der Reduktion $SAT \leq_p SPIDER$

4.4.3 Beweis

" $\Longrightarrow$ "
Sei zunächst die quantifizierte SAT-Formel wahr.
Die ersten $k = 2 \cdot t$ Züge sind genau das Bewegen der ersten t Literalauswahlstapel. Weitere Züge sind wegen der neu hinzugefügten Karten über allen anderen Stapeln nicht möglich. Da die SAT-Formel wahr ist, gibt es zu jeder Auswahl an $v_1 \cdots v_t$ also Variablen $v_{t+1} \cdots v_n$ sodass die SAT-Formel erfüllbar ist. Nachdem eine Variablenbelegung durch Bewegen der Literalauswahlstapel ausgewählt wurde, können die Karten, die die anderen Stapel blockieren wegbewegt werden.
Für eine feste Wahl von $v_1 \cdots v_t$ ist das Spiel also, analog zu Kapitel 4.1 gewinnbar. Somit ist es für alle Möglichkeiten $v_1 \cdots v_t$ auszuwählen gewinnbar.
Das Spiel ist also $2 \cdot t = k$-Einfach

" $\Longleftarrow$ "
Sei jetzt das Spiel k-einfach. Es kann also unabhängig der ersten k Züge gewonnen werden. Die ersten $k = 2 \cdot t$ Züge entsprechen genau dem Auswählen einer beliebigen Variablenbelegung der allquantifizierten Variablen. Danach kann eine Zugfolge gefunden werden, die genau die existenzquantifizierten Variablen auswählt. Analog zur Reduktion $SAT \leq SPIDER$ kann dann daraus eine erfüllende Belegung für die SAT-Formel konstruiert werden. Da genau die allquantifizierten Variablen beliebig ausgewählt wurden, ist die quantifizierte Formel also wahr.
Da k-Einfach in Π_2^p liegt und Π_2^p-Schwer ist, ist es Π_2^p-Vollständig

4.5 Approximierbarkeit

Sei G ein Spiel mit maximalem Kartenrang n sowie k Kartendecks. Im Folgenden ist die Anzahl an Kartendecks für jede Spielinstanz variabel. val(G) bezeichnet die maximale Anzahl an Stapeln, die in G abgelegt werden können.
Also ist val(G) := argmax$_M${# Stapel die in $G \circ M$ auf dem Ablagestapel liegen}, wobei M eine in G gültige Zugfolge ist.
Sei $0 < \rho \leq 1$. Eine ρ-Approximierung zu einer Spielinstanz G ist eine Zugfolge M, sodass in $G \circ M$ mindestens $\rho \cdot val(G)$ Decks auf dem Ablagestapel liegen.
Ein Algorithmus A heißt ρ-Approximationsalgorithmus, wenn für jede Spielinstanz G A(G) eine ρ-Approximierung liefert und A polynomielle Laufzeit hat.[1, PCP and Hardness of Approximation]

4.5.1 SPIDER ist nicht approximierbar

Satz: Gibt es einen ρ-Approximationsalgorithmus für **SPIDER** dann ist **P = NP**.

Man betrachte erneut die Reduktion $\mathbf{3-SAT} \leq_\mathbf{p} \mathbf{SPIDER}$ aus Kapitel 4.1.
Sei G eine Spielinstanz, die durch die Reduktion entstanden ist. G enthält genau zwei Decks. Ein ρ-Approximationsalgorithmus A muss, wenn G gewinnbar ist, immer eine Approximation finden, die mindestens einen der Stapel auf den Ablagestapel legt, da dann $val(G) = 2$ und $\rho > 0$ gilt.

Um ein Deck auf den Ablagestapel zu legen, müssen mindestens die Sperrkarten entfernt werden, da höherwertige Karten unter den Sperrkarten liegen. Nach 4.2.2 "Lemma zum großen Stapel" ist dann aber das Spiel gewinnbar. Es können also wieder beide Stapel auf den Ablagestapel gelegt werden.

Angenommen $val(G) = 1$. Dann gibt es also eine Zugfolge M, die nur einen Stapel auf den Ablagestapel legt. Diese muss aber die Sperrkarten entfernen, wodurch nach 4.2.2 das Spiel gewinnbar ist. Damit ist $val(G) = 2$, Widerspruch.

Somit kann $val(G)$ nur 0 oder 2 sein.
Daher gilt folgendes:
Der Approximationsalgorithmus findet eine Zugfolge, die einen Stapel auf den Ablagestapel bewegt
$\iff$
G ist gewinnbar.
$\iff$
Die Ursprüngliche 3-SAT Instanz ist erfüllbar

Somit liefert A ein Entscheidungsverfahren für $3-SAT$, dass in Polynomialzeit läuft:

1. Führe die Reduktion $\mathbf{3-SAT} \leq_\mathbf{p} \mathbf{SPIDER}$ aus
2. Führe A(G) aus.
3. Liefert A(G) eine Zugfolge, die mindestens einen Stapel ablegt?

Damit wäre **P = NP**
□

Dieses Resultat schließt die Existenz eines OPT - 2 Approximationsalgorithmus, also eins Algorithmus, der immer ein Ergebnis, das kleiner als val(G) − 2 ist, liefert, noch nicht aus. Hier kann ein solcher Algorithmus immer eine Lösung liefern, die kein Deck auf den Ablagestapel legt.

4.5.2 Ein Stapel

Satz: Sei G ein einfarbiges Spiel mit k Kartendecks und maximalem Rang n. Sei außerdem $k \leq p(n)$ für ein beliebiges, festes Polynom p. Das Entscheidungsproblem "Es kann in G mindestens ein Stapel auf den Ablagestapel gelegt werden" ist NP-Schwer.

Beweis: Die Reduktion $\mathbf{3-SAT} \leq_{\mathbf{p}}$ **SPIDER** wird angepasst. Mit zwei der Stapel wird die Reduktion wie gehabt durchgeführt. Die Restlichen $k-2$ Kartendecks werden jeweils absteigend sortiert unter den großen Stapel gelegt. Da $k \leq p(n)$ werden nur $n \cdot p(n)$ Karten eingefügt. Dies ist also in Polynomialzeit möglich.
Diese Stapel können nur abgelegt werden, wenn der große Stapel in der ursprünglichen Reduktion leer ist.

Wie im Beweis des "Lemma zum großen Stapel" begründet wurde, kann, wenn der große Stapel leer ist, mindestens ein Stapel auf den Ablagestapel gelegt werden. Der Beweis muss nur an einer Stelle leicht angepasst werden: Streng genommen handelt es sich durch die eingefügten Karten unter dem großen Stapel nicht mehr um sortierte Spiele. Trivialer Weise ändern diese Karten aber nichts an der Gewinnbarkeit, da sie, sobald sie spielbar sind, direkt abgelegt werden können.
Insbesondere können sie nur abgelegt werden, wenn bereits ein anderer Stapel ablegbar ist. Somit sind sie für das hier definierte Entscheidungsproblem irrelevant und müssen nicht betrachtet werden.

Analog zu 4.5.1 folgt die NP-Schwere. Somit kann, wenn $\mathbf{P} \neq \mathbf{NP}$ insbesondere auch kein val(G)-2 Algorithmus gefunden werden.

4.5.3 Maximale Stapelhöhe

Als Nächstes wird eine weitere Optimierungsvariante von **SPIDER** betrachtet: Gegeben eine Spielinstanz G, was ist der höchste einfarbige sequenzielle Teilstapel der gebildet werden kann?
Sei $maxHeight(G) := argmax_M\{$Höhe des höchsten einfarbigen sequenziellen Teilstapels in $G \circ M\}$, wobei M eine in G zulässige Zugfolge ist.

Die zugehörige Entscheidungsvariante lautet dann: geben ein Spiel G und eine Zahl h, ist $maxHeight(G) \geq h$
Die NP-Schwere folgt direkt aus 4.5.2. Ist $h = n$ also gleich dem maximalen Rang, so entspricht dies genau dem, dass ein Stapel abgelegt werden kann. Daher ist die Entscheidungsvariante NP-Schwer.

Sei $0 < \rho \leq 1$. Ein ρ-Approximationsalgorithmus ist ein Algorithmus A mit polynomieller Laufzeit, der bei Eingabe G eine Zugfolge M ausgibt, sodass die Höhe des größten Stapels in $G \circ M$ mindestens $\rho \cdot$maxHeight(G) ist.

Satz: Gibt es einen ρ-Approximationsalgorithmus für die größte Stapelhöhe, dann ist $\mathbf{P} = \mathbf{NP}$.
Auch hierfür wird die Reduktion $\mathbf{3-SAT} \leq_{\mathbf{p}}$ **SPIDER** angepasst.
Sei n der maximale Rang der reduzierten Instanz der ursprünglichen Reduktion. Konstruiere eine neue Instanz, in der $\frac{1}{\rho} \cdot n$ viele Karten in einem Bereich hinter der Endkarte eingefügt werden. Da ρ konstant ist, werden hier nur $O(n)$ Karten eingefügt. Die Laufzeit

bleibt polynomiell.

Die Karten werden, aufsteigend sortiert, unter den großen Stapel und den Restestapel eingefügt.
Dadurch, dass die Karten aufsteigend sortiert sind, bilden sie zusammengenommen keinen sequenziellen Teilstapel.[9]

Es muss jetzt wieder die Reduktionseigenschaft begründet werden.
" $\Longrightarrow$ "
Sei zunächst die **3-SAT**-Formel erfüllbar. Dann ist die ursprünglichen **SPIDER**-Instanz lösbar. Insbesondere können in der ursprünglichen Instanz der große Stapel und Reststapel freigeräumt werden. In der transformierten Instanz liegen damit dann die hinzugefügten Karten offen. Mithilfe zwei Freifeldern können dann die Karten absteigend sortiert werden. Das "Lemma zum großen Stapel" gilt also weiterhin. Somit ist dann das Spiel gewinnbar.
" $\Longleftarrow$ "
Sei jetzt das Spiel gewinnbar. Damit war auch das ursprüngliche Spiel gewinnbar, da der große Stapel im ursprünglichen Spiel leergeräumt werden muss. Analog zu Kapitel 4.2 folgt die Erfüllbarkeit der SAT-Formel.

Ist das Spiel gewinnbar, so kann ein Stapel der Höhe $n + \frac{1}{\rho} \cdot n$ gebildet werden. Ein ρ-Approximationsalgorithmus muss also eine Lösung liefern, in der ein Stapel der Höhe

$$\rho \cdot (n + \frac{1}{\rho} \cdot n) = n + \rho \cdot n$$

gebildet wird. Der so gebildete Stapel muss also zwangsweise Karten aus dem hinzugefügten Teil enthalten. Dafür muss aber entweder der große Stapel oder Restestapel aufgedeckt werden, was nur möglich ist, wenn alle Sperrkarten vom großen Stapel entfernt wurden. Somit ist nach 4.2.2 das Spiel gewinnbar.
Ist das Spiel nicht gewinnbar, können nicht alle Sperrkarten vom großen Stapel entfernt werden. Also können die neu hinzugefügten Karten nicht bewegt werden. Insbesondere kann der Approximationsalgorithmus also höchstens eine Lösung mit maximaler Höhe n-1 liefern.

Daraus lässt sich also folgendes Entscheidungsverfahren für **3-SAT** ableiten:

1. Führe die hier beschrieben Reduktion aus
2. Führe A(G) aus
3. Bildet die gefundene Lösung einen Teilstapel mit Mindesthöhe n?

Somit wäre **P** = **NP**□

4.5.4 Bedeutung für das Spiel

Aus algorithmischer Sicht bedeuten diese Resultate erst einmal, dass es vermutlich keinen effizienten Algorithmus gibt, der eine Zugfolge liefert, mit der überhaupt ein Stapel auf den Ablagestapel gelegt werden kann. Genauso gibt es vermutlich keinen effizienten Algorithmus, der eine Mindeststapelhöhe finden kann.

Aus spielerischer Sicht lässt sich daraus ableiten, dass es kein "perfektes" Kriterium geben kann, mit dem man überhaupt einen Stapel immer ablegen kann. Genauso gibt es

[9] Hierfür müssten die Karten absteigend sortiert sein

kein gutes Kriterium, mit dem man immer eine Mindeststapelhöhe erreicht. Einfache Spielstrategien müssen also auch hierfür auf Heuristiken zurückgreifen.

Man muss aber klar feststellen: In den allermeisten Spielen funktionieren dennoch gängige Spielstrategien, wie sie zum Beispiel in [6] beschrieben wurden, gut genug.

4.6 Future Work

4.6.1 Hanoi Spider Solitaire

Inspiriert von dem Blogartikel "Tower of Hanoi" [10] wird Hanoi-Solitaire definiert.

Hanoi-Solitaire ist eine Variation von Spider-Solitaire, in der nicht nur das Ablegen von Karten auf eine Karte mit einem Rang, der genau eins größer ist, erlaubt ist. Eine Karte kann auf andere Karten, deren Rang echt größer als der eigene Rang ist, abgelegt werden. Alle weiteren Regeln bleiben unverändert.

Man betrachte folgendes Spiel:

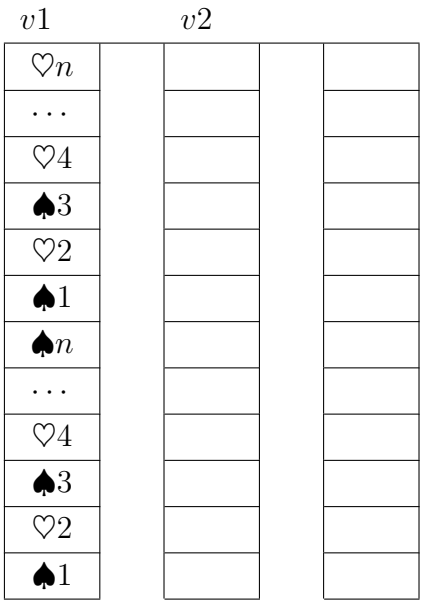


Abbildung 9: Türme von Hanoi als Spider-Solitaire Spiel

Man erkennt direkt: Das Spiel kann nur gewonnen werden, wenn der obere Teil vollständig wegbewegt wird.
Da die Farben alternierend sind, kann dabei immer nur eine Karte einzeln bewegt werden. Dies entspricht genau den Türmen von Hanoi.
Hat man den oberen der beiden Stapel auf den zweiten/dritten Stapel bewegt, so kann man durch Verwenden des dritten freien Stapels das Spiel gewinnen.

Die Türme von Hanoi können in mindestens $2^n - 1$ Zügen gelöst werden. Also werden hier auch eine exponentielle Anzahl an Zügen benötigt. Somit kann nicht mithilfe der minimalen Zuglänge argumentiert werden, um die Zugehörigkeit zur Klasse NP zu beweisen.
Es könnte aber sein, dass die Gewinnbarkeit der Hanoi-Variante vielleicht sogar effizient entschieden werden kann.
Genauso gut könnte es aber auch sein, dass durch das Zulassen von Makrozügen, die das Umstapeln eines sequentiellen Teilstapels als einen Zug kodieren, die Zugehörigkeit zur Klasse NP gezeigt werden kann.Es ist aber auch möglich, dass auch mit Makrozügen weiterhin exponentiell viele Makrozüge zum Gewinnen benötigt werden. Die genauere Betrachtung liegt nicht mehr im Rahmen dieser Bachelorarbeit und wird daher nicht durchgeführt.
Fest steht eine Sache: Da die gewinnenden Zugfolgen exponentiell lang sind, würde das Spiel vermutlich wenig Spaß machen.

5 Endfazit

Nachdem jetzt sehr viele Seiten über Spider-Solitaire geschrieben wurden, stellt sich natürlich die abschließende Frage: Was wurde jetzt eigentlich über Spider-Solitaire gelernt?

Besonders hervorgehoben wird noch einmal das Resultat aus 4.1.5 "Mehrfarbiges Spalten", da hiermit tatsächlich die Korrektheit einer Gewinnstrategie gezeigt wurde.

Aus den NP-Vollständigkeits- und Π_2^p-Vollständigkeitsresultaten kann man leider keine Gewinnstrategie erkennen. Insbesondere mit den in 4.5 gezeigten nicht-Approximierbarkeitsresultaten wurde gezeigt, dass es vermutlich keine effizienten Gewinnstrategien geben kann.

Vermutlich ist aber Spider-Solitaire, genau dadurch, dass es keine gute Gewinnstrategie gibt, man aber eine Lösung leicht nachvollziehen kann, ein so interessantes Spiel.

6 Quellen

Literatur

[1] Sanjeev Arora und Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. ISBN: 978-0-521-42426-4.

[2] S. Bosch. *Lineare Algebra*. 978-3-540-76438-0. Springer Berlin Heidelberg, 2008. ISBN: 9783642552601. DOI: https://doi.org/10.1007/978-3-540-76438-0.

[3] Microsoft Corporation. *Z3 API in Python*. URL: https://microsoft.github.io/z3guide/programming/Z3%20Python%20-%20Readonly/Introduction/ (besucht am 01. 08. 2024).

[4] Malte Helmert. „Complexity results for standard benchmark domains in planning“. In: *Artificial Intelligence* 143.2 (2003), S. 219–262. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(02)00364-8. URL: https://www.sciencedirect.com/science/article/pii/S0004370202003648.

[5] Malte Helmert. „On the Complexity of Planning in Transportation and Manipulation Domains“. Diplomarbeit. Albert-Ludwigs-Universität Freiburg, Fakultät für Angewandte Wissenschaften, Institut für Informatik, 2001. URL: https://ai.dmi.unibas.ch/papers/helmert-diplom-2001.pdf.

[6] Nils Hesse. „Spider Solitär“. In: *Spielend gewinnen: Gewinnstrategien für die 50 bekanntesten Karten-, Würfel-, Brett- und Gewinnspiele*. Wiesbaden: Springer Fachmedien Wiesbaden, 2015, S. 195–197. ISBN: 978-3-658-04441-1. DOI: 10.1007/978-3-658-04441-1_42.

[7] Leonardo de Moura und Nikolaj Bjørner. „Z3: an efficient SMT solver“. In: Bd. 4963. Apr. 2008, S. 337–340. ISBN: 978-3-540-78799-0. DOI: 10.1007/978-3-540-78800-3_24.

[8] Nikolaj Bjørner/ Leonardo de Moura/ Lev Nachmanson/ Christoph Wintersteiger. *Programming Z3*. URL: https://theory.stanford.edu/~nikolaj/programmingz3.html (besucht am 01. 08. 2024).

[9] The KDE Project Paul Olav Tvete. *KPatience*. URL: https://apps.kde.org/kpat/ (besucht am 04. 06. 2024).

[10] spidergm. *Tower of Hanoi*. 21. Juni 2020. URL: https://spidersolitaireaddict.blog/2020/06/21/tower-of-hanoi/ (besucht am 03. 08. 2024).

[11] Tom Holroyd Stephan Kulow. *Patsolve*. unknown. URL: https://cards.fandom.com/wiki/Patsolve (besucht am 04. 06. 2024).

[12] Jesse Stern. „Spider Solitaire is NP-Complete“. In: (2011). DOI: https://doi.org/10.48550/arXiv.1110.1052.

[13] KDE Project/Author Unknown. *Rules for Individual Games*. year Unknown. URL: https://docs.kde.org/stable5/en/kpat/kpat/rules-specific.html#spider (besucht am 12. 06. 2024).

[14] Mark S. Weisser. „HOW MANY GAMES OF SPIDER SOLITAIRE ARE WINNABLE? EXPLORATIONS INTO THE MATHEMATICS UNDERLYING SPIDER SOLITAIRE“. Masterarbeit. Wesleyan University, 2012. URL: http://weissersolutions.com/media/cc362089536c6195ffff85e9ffffe905.pdf (besucht am 15. 05. 2024).

Kolophon

Diese Arbeit wurde mit LaTeX 2_ε gesetzt.
Die Titleseite verwendet den von Ricardo Langner entwickelten Stil *Clean Thesis*. Das Design des Stils *Clean Thesis* ist von Benutzerhandbüchern von Apple Inc. inspiriert. Laden Sie den Stil *Clean Thesis* unter http://cleanthesis.der-ric.de/ herunter. Anpassungen an den Stil des Instituts für Informatik sind unter https://gitlab.rlp.net/institut-fur-informatik/cleanthesis-jgu zu finden.

Eigenständigkeitserklärung

Hiermit erkläre ich,

dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel (dazu zählen auch KI-basierte Anwendungen oder Werkzeuge1) benutzt habe. Sämtliche wörtlichen oder sinngemäßen Übernahmen und Zitate sind kenntlich gemacht und nachgewiesen. Ich versichere, dass ich keine Hilfsmittel verwendet habe, deren Nutzung die Prüferin oder der Prüfer explizit ausgeschlossen hat.

Im Anhang „Nutzung KI-Tools“ habe ich die verwendeten KI-Tools dokumentiert. Zudem habe ich im Anhang „KI-Outputs“ sämtliche KI-generierten Outputs einzeln aufgeführt, die relevant für die Aufgabe waren.

Mit Abgabe der vorliegenden Leistung übernehme ich die Verantwortung für das eingereichte Gesamtprodukt. Ich verantworte damit auch jegliche KI-generierten Inhalte, die ich in meine Arbeit übernommen habe. Die Richtigkeit übernommener (KI-generierter) Aussagen und Inhalte habe ich nach bestem Wissen und Gewissen geprüft.

Ich habe die Arbeit nicht zum Erwerb eines anderen Leistungsnachweises in gleicher oder ähnlicher Form eingereicht.

Mir ist bekannt, dass ein Verstoß gegen die genannten Punkte prüfungsrechtliche Konsequenzen hat und insbesondere dazu führen kann, dass die Studien- und Prüfungsleistung als mit „nicht bestanden“ bewertet wird. Die Einschreibung kann für bis zu zwei Jahre widerrufen werden, wenn Studierende zweimal oder häufiger bei Prüfungsleistungen täuschen (§ 69 Abs. 4 und 5 HochSchG).

Ort, Datum und Unterschrift