Edge Hop

Ein Modell zur Komplexitätsanalyse von kombinatorischen Spielen

Moritz Gobbert
(Unter Betreuung von Prof. Dr. Henning Fernau und Prof. Dr. Peter Sturm)

Masterarbeit zur Erlangung des akademischen Grades eines Master of Science

Universität Trier
FB IV - Informatikwissenschaften
6. Mai 2015

Zusammenfassung

In dieser Masterarbeit werde ich ein Spiel vorstellen, bei dem es darum geht einen markierten Spielstein auf einem Graphen von einem Startknoten zu einem Zielknoten zu bewegen. Manche Knoten des Graphen sind mit weiteren Spielsteinen besetzt, die der Spieler unter passenden Bedingungen ebenfalls bewegen kann. Weiterhin hat der Graph bestimmte Eigenschaften, wie z. B., dass auf jedem Knoten nur eine bestimmte Anzahl an Spielsteinen liegen darf oder dass jede Kante vom markierten Spielstein nur einmal passiert werden kann. Im weiteren Verlauf der Masterarbeit werde ich durch eine Reduktion von *3-SAT* zeigen, dass die Frage, ob der markierte Spielstein den Zielknoten erreichen kann, *NP-vollständig* ist. Für die Reduktion werde ich verschiedene Gadgets konstruieren, die ein Gerüst für die Komplexitätsanalyse anderer (kombinatorischer) Spiele bzw. ähnlicher Fragestellungen bieten. Damit werde ich dann die Komplexität von spezifischen Varianten von *Latrunculi*, *Minecraft*, *2048* und *Game about Squares* bestimmen.

Keywords: Komplexität, Kombinatorische Spiele, Latrunculi, Minecraft, 2048, Game about Squares.

Eidesstattliche Erklärung

Ich versichere an Eides Statt durch meine eigenhändige Unterschrift, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt habe. Alle Stellen, die wörtlich oder dem Sinn nach auf Publikationen oder Vorträgen anderer Autoren beruhen, sind als solche kenntlich gemacht. Ich versichere außerdem, dass ich keine andere als die angegebene Literatur verwendet habe. Diese Versicherung bezieht sich auch auf alle in der Arbeit enthaltenen Zeichnungen, Skizzen, bildlichen Darstellungen und dergleichen.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt.

Datum und Ort

Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Spiele sind ein in der Komplexitätstheorie stark untersuchter Bereich. Hierbei werden verschiedene Arten von Spielen, bspw. kombinatorische Geduldsspiele wie *Rush Hour* oder *Numberlink*, Zweipersonen-Spiele wie *Dame* oder *Phutball* und sogar Videospiele wie *Pokémon* oder *Lemmings*, betrachtet. Selbst „Keinpersonen"-Spiele, z. B. *Conway's Game of Life*, werden untersucht.[1]

Spiel	*Komplexität*	*Literatur*
Rush Hour	PSPACE-vollständig bzw. FPT	[A11, A10]
Numberlink	NP-vollständig	[A1]
Dame	EXPTIME-vollständig	[A23]
Phutball	PSPACE-schwer	[A7]
Pokémon	NP-vollständig	[A2]
Lemmings	PSPACE-vollständig	[A25]
Conway's Game of Life	Turing-vollständig	[A22]

Tabelle 1.1: Verschiedene Spiele und die Komplexitäten der zugehörigen Entscheidungsfragen.

Die unterschiedlichen Ergebnisse bezüglich Komplexität der Spiele, bzw. der in den entsprechenden Arbeiten untersuchten Entscheidungsfragen, (siehe Tabelle 1.1) geben Hinweis darauf, dass die jeweiligen Beweise unterschiedlich geführt wurden. Darum haben Hearn und Demaine ein Modell entwickelt, welches als Rahmen für Schwere-Beweise[2] fungiert – die so genannte *Nondeterministic Constraint Logic (NCL)* [A14]. Wir wollen allerdings in dieser Arbeit ein weiteres Modell entwickeln, welches als Rahmen für NP-Schwere-Beweise genutzt werden kann. Unser Modell soll sich im Gegensatz zur NCL „näher" an Problemen der Bewegungsplanung orientieren, so dass sich die Komplexitäten verschiedener Probleme natürlicher mit unserem Modell zeigen lassen. Grob wird diese Arbeit aus zwei Teilen bestehen. Im ersten Teil werden wir das Modell entwickeln und zeigen, dass es NP-vollständig ist und im zweiten Teil werden wir das Modell nutzen, um die Komplexität anderer Probleme zu zeigen.

[1] Eine umfangreiche Liste von NP-vollständigen Puzzles findet sich bspw. bei Graham Kendall, Andrew Parkes und Kristian Spoerer [A16]. Ebenfalls hat Édouard Bonnet in seiner Doktorarbeit ein Kapitel der Komplexität von Spielen gewidmet [A4].

[2] Die NCL ist in ihrer ursprünglichen Version ein Modell für PSPACE-Schwere-Beweise. Aber es gibt Varianten, die zum Zeigen vom NP-Schwere gezeigt werden können.

1.1 Motivation

Das Bestimmen der Komplexität eines Problems ist manchmal relativ simpel (vor allem, wenn man „ähnliche“ Probleme aufeinander reduziert – bspw. lässt sich das *Hamiltonkreis-Problem* auf das *Problem des Handlungsreisenden* reduzieren, indem man jeder Kante Kosten von 0 zuweist), aber oft auch ziemlich aufwändig (z. B. Cooks Beweis zur NP-Vollständigkeit von SAT [A5]). Häufig werden Komplexitäts-Beweise ähnlicher Probleme grob gleich geführt und darum stellt sich die Frage, ob man den Beweisprozess nicht vereinfachen kann, indem man die „aufwändigen“ Argumentationen der Reduktion in einem *eigenen* und *allgemeinen* Beweis führt und mit diesem Beweis „Schnittstellen“ (*Gadgets*) zur Verfügung stellt, um die Komplexität anderer Probleme zu zeigen. Im Idealfall sollte das Konstruieren der Gadgets weniger aufwändig sein, als den Beweis der Komplexität direkt zu führen und zusätzlich sollten die (selben) Gadgets für die Reduktionen mehrerer Probleme funktionieren. Hearn und Demaine haben 2005[3] die *Nondeterministic Constraint Logic* entwickelt um ein solches Modell für (in ihrer ursprünglichen Version) PSPACE-vollständige Probleme zu bieten. Auch wenn es mittlerweile Variationen dieses Modells für andere Komplexitätsklassen gibt, wollen wir dennoch ein eigenes (NP-vollständiges) Modell entwickeln. Wir werden unser Modell als (abstraktes) Spiel namens *Edge Hop* konstruieren, in dem es darum geht, Pfade auf einem Graphen mit besonderen Eigenschaften zu finden. Durch diese Fragestellung können wir Gadgets bereitstellen, welche mit Problemen der Bewegungsplanung zusammenhängen. Wir bieten also ein Modell, mit welchem sich auf natürliche Weise die NP-Schwere von Bewegungsplanungs-Problemen zeigen lässt.

Weiter wollen wir in dieser Arbeit mithilfe von *Edge Hop* die Komplexität von vier verschiedenen Spielen (bzw. spezifischen, zugehörigen Fragestellungen) zeigen. Wir werden uns mit einem alten römischen Brettspiel namens *Latrunculi*, einem aktuellen Videospiel namens *Minecraft* und zwei Browserspielen namens *2048* und *Game about Squares* beschäftigen.

Latrunculi wurde gewählt, weil dort einzelne Züge mächtiger sind, als z. B. bei Schach oder Dame. Ein Spieler kann mit einem einzelnen Zug potentiell jede Position auf dem Spielfeld erreichen und deswegen ist die Frage, ob er einen bestimmten Stein des Gegenspielers erobern[4] kann, nicht trivial. Wir werden uns also mit genau dieser Frage beschäftigen. Außerdem wurde Latrunculi als Beispiel gewählt, weil die Bestimmung der Komplexität mithilfe von Edge Hop weniger aufwändig ist als die explizite Reduktion von 3-SAT (siehe [A13]). Wir haben also an dieser Stelle ein Beispiel dafür, dass die Bestimmung von Komplexität mithilfe von allgemeinen Modellen weniger aufwändig ist, als explizite Reduktionen von verschiedenen anderen Problemen.

Minecraft wurde als Beispiel gewählt, weil das Bauen Teil des Spiels ist und sich somit passende Gadgets im Spiel selbst konstruieren lassen, ohne auf einen externen Editor angewiesen zu sein. Wir werden allerdings sehen, dass wir sonst keine Besonderheiten des Spiels verwenden und es als Vertreter für jegliche Videospiele in einer dreidimensionalen Umgebung steht, die die Möglichkeit bieten, den Spieler in eine Richtung zu zwingen und zusätzlich noch die Eigenschaft haben, dass es (verschlossene) Türen gibt, welche man von einem anderen Ort aus öffnen kann

[3] Siehe [A15].

[4] Bei Latrunculi *erobert* man Steine des Gegenspielers, statt sie zu schlagen.

bzw. muss.

2048 wurde gewählt, weil es die interessante Eigenschaft hat, dass jeder Zug globale Auswirkungen hat. Dadurch, dass ein Zug prinzipiell das ganze Spielfeld verändern kann, ist es hier nicht ausreichend, einfach nur die passenden Gadgets zu konstruieren. Bei 2048 ist es notwendig, dass angegeben wird, wie man die Gadgets auf dem Spielfeld platziert, so dass diese nicht durch Züge zerstört werden, die an einer anderen Stelle des Spielfelds ausgeführt werden. Weiter ist 2048 ein gutes Beispiel dafür, dass man auch die Komplexität von Problemen mit Edge Hop bestimmen kann, bei denen es nicht explizit darum geht, ein Objekt zu einer bestimmten Position zu bewegen. Man muss die globalen Züge und ihre Auswirkungen auf das Spielfeld passend interpretieren.

Game about Squares wurde als weiteres Beispiel gewählt, weil es ein Bewegungsplanungs-Problem ist und damit sehr gut zu unserem Modell passt. Interessant bei Game about Squares ist, dass es ähnlich zu Sokoban und Push-∗-X ist, welche in unterschiedlichen Komplexitätsklassen liegen. Bei diesem Beispiel kommt es also darauf an, wie man das Spiel verallgemeinert.

1.2 Unterschiede zur NCL

In diesem Abschnitt wollen wir uns kurz die ursprüngliche Variante der NCL ([A15]) anschauen und dann erläutern, worin die Unterschiede zu unserem Modell liegen. [5]

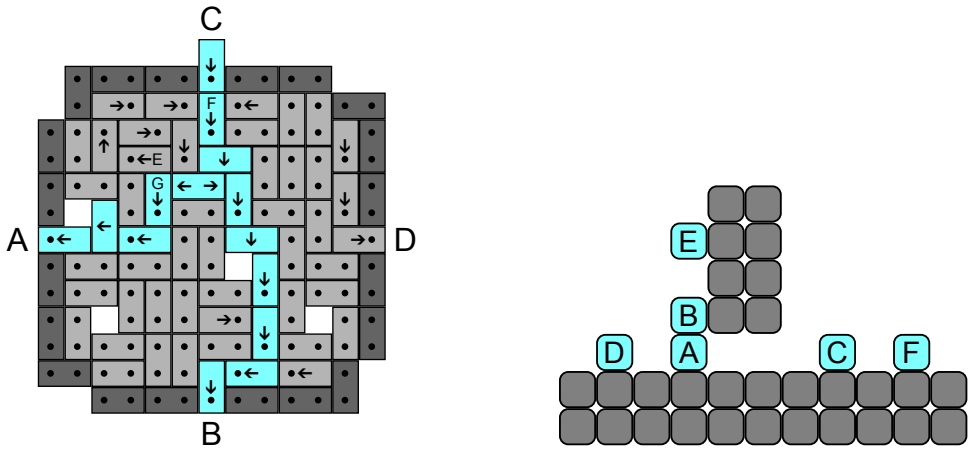
Eine *NCL-„Maschine“* („NCL machine“) ist ein ungerichteter Graph mit (nichtnegativen) Kantengewichten und Minimum-Inflow-Bedingungen an den Knoten. Eine Konfiguration einer solchen Maschine ist eine Orientierung aller Kanten, so dass die jeweiligen Inflow-Bedingungen an den Knoten erfüllt sind. Ein Übergang von einer Konfiguration in eine andere ist das Umkehren der Orientierung *einer* Kante, so dass die Inflow-Bedingungen erfüllt bleiben. Die essentielle Frage bei diesem Problem ist es, ob man zu einer gegebenen Konfiguration mit einer Folge von Konfigurationsübergängen die Orientierung einer bestimmten Kante umkehren kann.[6] Robert A. Hearn und Erik D. Demaine haben gezeigt, dass dieses Problem PSPACE-vollständig ist. Hierfür haben sie eine Art Normalform eingeführt, bei der jeder Knoten der NCL-Maschine eine Valenz von 3 hat[7], die Inflow-Bedingung jedes Knotens 2 beträgt und zusätzlich jede Kante ein Gewicht von entweder 1 oder 2 hat. Mit Verwendung der Normalform haben sie zwei Grundgadgets konstruiert, so genannte AND- und OR-Knoten. Aus diesen Grundgadgets haben sie dann andere Gadgets (Quantor-Gadgets, Kreuzungs-Gadgets) konstruiert, um letztendlich eine Reduktion vom Erfüllbarkeitsproblem für quantifizierte boolesche Formeln (QBF) durchzuführen.

Um nun die Komplexität eines Bewegungsplanungs-Problems $\mathcal{P}$ mithilfe der NCL zu zeigen müssen AND- und OR-Knoten als Instanz von $\mathcal{P}$ konstruiert werden. Zusätzlich muss man noch die einzelnen Kanten modellieren, die die Gadgets untereinander verbinden. Da es bei einer NCL-Konfiguration um Orientierungen von Kanten

[5] Abgesehen davon, dass die ursprüngliche NCL für Probleme in PSPACE und Edge Hop für Probleme in NP gedacht ist.

[6] Alternativ kann man sich die Frage stellen, ob man aus einer gegebenen Konfiguration durch Konfigurationsübergänge eine bestimmte, andere Konfiguration erreichen kann.

[7] Genauer hat ein Knoten entweder drei unterschiedliche inzidente Kanten, oder eine inzidente Kante und eine inzidente Schleife.



(a) Schiebepuzzle. (Vgl. [A15], Abbildung 12, Seite 16.)

(b) Sokoban-Spielsituation. (Vgl. [A15], Abbildung 15, Seite 21.)

Abbildung 1.1: Ein AND-Knoten konstruiert als Schiebepuzzle und als Sokoban-Spielsituation.

und nicht explizit um Bewegung in dem zugehörigen Graphen geht, gibt es keine „natürliche“ Entsprechung der Konfigurationsübergänge (in der NCL) zum Bewegen eines Objekts bei Problem $\mathcal{P}$. Hier kommt es stark darauf an, wie man Konfigurationen und Konfigurationsübergänge von $\mathcal{P}$ interpretiert, damit diese zur NCL passen. Abbildung 1.1a zeigt, wie ein AND-Knoten als Schiebepuzzle konstruiert wurde. Dass der Stein bei C ein Feld aus dem Gadget hinausragt, bedeutet, dass die zugehörige Kante zum Gadget hin zeigt. Die Steine bei A und B ragen nicht aus dem Gadget hinaus, was bedeutet, dass die zugehörigen Kanten vom Gadget weg zeigen. Damit der Stein bei C komplett ins Gadget gezogen werden kann (also die zugehörige Kante vom Gadget weg zeigen kann), müssen sowohl der Stein bei A als auch der Stein bei B aus dem Gadget gezogen werden (also die zugehörigen Kanten zum Gadget hin zeigen). Abbildung 1.1b zeigt wie das gleiche Gadget als Sokoban-Spielsituation konstruiert werden kann. Die dargestellte Situation entspricht wieder einer zum Gadget hin zeigenden Kante (die Block C entspricht) und zwei vom Gadget weg zeigenden Kanten (die den Blöcken A und B entsprechen). Das Umkehren der Orientierung einer Kante entspricht in diesem Fall dem Verschieben des Blockes. Block A und C müssen jeweils nach links und Block B nach oben verschoben werden.

Bei beiden Beispielen wurde also ein Logik-Problem in ein Bewegungsplanungs-Problem eingebettet. Das ist schon der erste Unterschied zu unserem Modell – Edge Hop. Wir möchten Gadgets bereitstellen, die nicht explizit auf Logik-Gattern beruhen, sondern Aspekte der Bewegungsplanung beinhalten. Dadurch ergibt sich eine eventuell einfachere Möglichkeit, diese Gadgets aus Bewegungsplanungs-Problemen zu konstruieren. Da unser Modell ebenfalls auf einem Graph-Problem beruht, müssen wir auch Gadgets zur Modellierung der Kanten entwerfen. Bei den meisten Bewegungsplanungs-Problemen sind diese allerdings problemlos zu konstruieren, weil es Teil dieser Art von Problemen ist, ein Objekt über Wege (Pfade, Felder auf einem Spielbrett, Gänge, ...) zu bewegen und diese intuitiv den Kanten des Graphen entsprechen. Weiter werden wir noch ein Gadget konstruieren, welches einen Weg in zwei Wege aufspaltet, also eine einfache Abzweigung. Auch diese ist bei vielen Bewegungsplanungs-Problemen ohne Aufwand zu konstruieren, da die Möglichkeit zwischen mehreren Wegen zu wählen ebenfalls Teil dieser Art von Problemen ist. Zusätzlich benötigen wir auch noch das „Gegenstück“ – ein Gadget, dass zwei We-

ge zu einem zusammenführt. Dieses ist in den meisten Fällen nur ein gespiegeltes Verzweigungs-Gadget und somit auch ohne größeren Aufwand konstruierbar. Dann werden wir noch ein Gadget konstruieren, welches ein zu bewegendes Objekt nur in eine bestimmte Richtung ziehen lässt – eine Einbahnstraße oder Diode. Diese Funktionsweise ist nicht bei jedem Bewegungsplanungs-Problem inhärent vorhanden, kann aber in vielen Fällen ohne Aufwand konstruiert werden. Zuletzt werden wir noch ein Gadget konstruieren, bei welchem ein bestimmter Weg nur dann passierbar ist, wenn zuvor ein anderer Weg im selben Gadget benutzt wurde. Wir nennen dieses Gadget Entsperr-Gadget. Dies kann z. B. einem Gang entsprechen, in dem sich ein Schlüssel befindet und einem anderen Gang, in dem sich die zugehörige Tür befindet. Bei diesem Gadget kommt es also auch auf das zu betrachtende Problem an – wenn das Problem eine „Entsperr-Mechanik“ besitzt, dann ist es klar, wie das Gadget konstruiert werden muss.

Das heißt allerdings, dass für einen Beweis über Edge Hop mehr essentielle Gadgets konstruiert werden müssen. Sind es bei der NCL nur zwei (AND- und OR-Knoten), müssen bei Edge Hop vier verschiedene konstruiert werden (*Dioden*-Gadget, *Verzweigungs*-Gadget, *Zusammenführ*-Gadget und *Entsperr*-Gadget).[8] Da in den meisten Fällen das Zusammenführ-Gadget ein gespiegeltes Verzweigungs-Gadget ist, muss man dieses nicht explizit konstruieren. Damit bietet unser Modell ein Grundgadget mehr, aber diese sind dafür näher an Problemen der Bewegungsplanung und damit eventuell weniger aufwändig zu konstruieren.

Ein weiterer Unterschied zwischen beiden Modellen ist der, dass die NCL-Graphen planar sind und man somit keine Kreuzungs-Gadgets konstruieren muss. Bei manchen Problemen kann es aufwändig sein, eine Kreuzung zu konstruieren. Aber bei vielen anderen Problemen (bspw. solche bei denen das zu bewegende Objekt nicht von sich aus die Bewegungsrichtung ändern kann oder solche, bei denen man Bewegungen im dreidimensionalen Raum betrachtet) kann man problemlos Kreuzungs-Gadgets entwerfen.

[8] Bei beiden Modellen müssen noch die Kanten des zugehörigen Graphen und eventuell noch Gadgets für Anfang und Ziel der Bewegungsabfolge konstruiert werden.

Kapitel 2

Theorie

Im Folgenden werden wir die Regeln des Spiels und die Eigenschaften des zugehörigen Graphen kennen lernen. Danach werden wir sehen, dass die Frage, ob der Spielstein den Zielknoten erreichen kann, *NP-vollständig* ist indem wir eine Reduktion von *3-SAT* vornehmen.

2.1 Edge Hop

Edge Hop ist ein Knobelspiel für eine Person – also nach den üblichen Definitionen ein *kombinatorisches Geduldsspiel* („combinatorial puzzle“) [A6]. Das Spiel wird auf einem ungerichteten Graphen $G = (E, V)$ gespielt. Ziel von Edge Hop ist es, einen bestimmten Stein zu einem bestimmten Knoten zu ziehen. Hierbei darf nur zwischen adjazenten Knoten gezogen werden. Anders ausgedrückt ist das Ziel des Spiels einen Pfad zwischen zwei bestimmten Knoten zu finden.

Definition 2.1.1 (Aktiver Stein)**.** Der aktive Stein ist ein speziell markierter Stein, mit dem der Spieler zwischen Knoten des Graphen zieht. Der Spieler darf den aktiven Stein nur von einem Knoten u zu einem Knoten v ziehen, wenn $uv \in E$. Wir nennen dies einen *aktiven Zug*. Für einen aktiven Zug von einem Knoten u zu einem Knoten v schreiben wir kurz $u \xrightarrow{a} v$. Zusätzlich nennen wir den Knoten, auf dem sich der aktive Stein zu Beginn des Spiels befindet *Startknoten*.

Definition 2.1.2 (Passiver Stein)**.** Passive Steine sind weitere Steine auf dem Graphen. Ein passiver Stein darf nur von einem Knoten u zu einem Knoten v gezogen werden, wenn $uv \in E$ und zusätzlich entweder $x \xrightarrow{a} v$ der letzte aktive Zug war (für ein beliebiges $x \in V$) oder v der Startknoten ist und noch kein aktiver Zug ausgeführt wurde. Wir nennen dies einen *passiven Zug*. Für einen passiven Zug von einem Knoten u zu einem Knoten v schreiben wir kurz $v \xleftarrow{p} u$.

Definition 2.1.3 (Zielknoten)**.** Der Zielknoten ist ein speziell markierter Knoten $z \in V$, den der Spieler versucht zu erreichen. Der Spieler gewinnt das Spiel, wenn er den Zug $u \xrightarrow{a} z$ für ein $u \in V$ ausführt.

Definition 2.1.4 (Maximale Belegtheit)**.** Die Maximale Belegtheit $k_v \in \mathbb{N}$ eines Knotens v gibt an, wie viele Spielsteine (sowohl der aktive als auch passive Steine) sich auf diesem Knoten v höchstens befinden dürfen. Für einen Knoten v, auf dem sich k Spielsteine befinden, ist ein Zug $u \xrightarrow{a} v$ oder $v \xleftarrow{p} u$ nur dann möglich, wenn $k < k_v$.

Definition 2.1.5 (Benutzte Kanten)**.** Zu einem beliebigen Zeitpunkt $t \geq 0$ des Spielverlaufs ist $X_t \subseteq E$ die Menge aller Kanten uv, für die ein aktiver Zug $u \xrightarrow{a} v$ im bisherigen Spielverlauf existiert. Wir definieren induktiv:

$$X_0 := \emptyset$$
$$X_t := \begin{cases} X_{t-1} \cup \{uv\}, \text{ falls der } t\text{-te Zug } u \xrightarrow{a} v \\ X_{t-1}, \text{ sonst} \end{cases}$$
$$X := \bigcup_{t \geq 0} X_t$$

X nennen wir die Menge *aller* benutzten Kanten.

Definition 2.1.6 (Gültige Zugfolge)**.** Eine Zugfolge ist eine Folge von beliebigen aktiven und passiven Zügen (die den Definitionen 2.1.1, 2.1.2 und 2.1.4 genügen). Eine Zugfolge $Z = Z_1, Z_2, ..., Z_m$ heißt gültig, wenn Folgendes gilt:

(1) Für $1 \leq i \leq m$ gilt: Z_i ist ein aktiver Zug $\Rightarrow$ Z_i nutzt keine Kante aus X_i.

(2) Der letzte Zug ist $u \xrightarrow{a} z$ für ein $u \in V$.

Falls eine gültige Zugfolge $Z = Z_1$ nur aus einem Zug besteht, so sagen wir auch: *Der Zug Z_1 ist gültig.*

Definition 2.1.7 (EdgeHop-Graph)**.** Ein EdgeHop-Graph ist ein 5-Tupel $\mathcal{EH} = (G, k_G, a, z, P_G)$. Hierbei ist $G = (V, E)$ ein einfacher (zusammenhängender) Graph[1], $k_G : V \to \mathbb{N}$, $k_G(v) = k_v$ eine Funktion, die jedem Knoten seine maximale Belegtheit zuordnet, $a \in V$ der Startknoten, $z \in V$ der Zielknoten und P_G eine Multimenge über V, welche für jeden passiven Stein den Knoten beinhaltet, auf dem er liegt. Wenn es klar ist über welchen Graphen wir sprechen, werden wir statt $k_G(v)$ einfach nur $k(v)$ und statt P_G einfach nur P schreiben. Ebenfalls setzen wir $V(\mathcal{EH}) := V(G)$ und $E(\mathcal{EH}) := E(G)$.

Definition 2.1.8 (EdgeHop)**.** Sei $\mathcal{EH}$ ein beliebiger EdgeHop-Graph. Gibt es in diesem eine gültige Zugfolge von a nach z? Wir nennen dieses Entscheidungsproblem EdgeHop.

Abbildung 2.1 zeigt einen Beispiel-Graphen $\mathcal{EH}$, auf dem Edge Hop gespielt werden kann. Die Knotenmenge V ist $\{a, b, c, d, z\}$ wobei z den Zielknoten bezeichnet. Die Kantenmenge E ist $\{ab, ad, bc, bd, cz\}$. Der grüne Kreis auf Knoten a ist der aktive Stein, die roten Kreise auf Knoten b und c sind die passiven Steine. Die Zahlen unter den Knoten geben die jeweilige maximale Belegtheit an. Formal gilt:

$$\begin{aligned} \mathcal{EH} = (&(\{a, b, c, d, z\}, \{ab, ad, bc, bd, cz\}), \\ &\{(a, 2), (b, 2), (c, 1), (d, 2), (z, 1)\}, \\ &a, z, \{b, b, c\}) \end{aligned}$$

Eine gültige Zugfolge für dieses Beispiel ist $a \xleftarrow{p} b, a \xrightarrow{a} d, d \xleftarrow{p} b, d \xrightarrow{a} b, b \xleftarrow{p} c, b \xrightarrow{a} c, c \xrightarrow{a} z$. Abbildung A.1 im Anhang A zeigt diese Zugfolge.

[1] Den Zusammenhang können wir o. B. d. A. annehmen, denn wenn sich Start- und Zielknoten nicht in der selben Zusammenhangskomponente befinden, gibt es keine Zugfolge von a nach z und wenn sich ein beliebiger anderer Knoten nicht in der selben Zusammenhangskomponente wie der Startknoten befindet, kann dieser durch keine Zugfolge erreicht werden und muss somit nicht betrachtet werden.

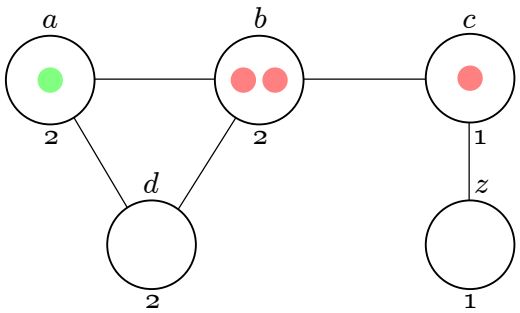


Abbildung 2.1: Ein Beispielgraph.

2.2 Komplexität

In diesem Abschnitt werden wir sehen, dass EdgeHop NP-vollständig ist. Zuerst werden wir die Zugehörigkeit zu NP zeigen und danach dann die NP-Schwere. Für die Charakterisierung von NP werden wir auf die Polynomialzeithierarchie zurückgreifen (vgl. [A26]):

$$L \in NP$$
$$\Leftrightarrow$$
es gibt ein Polynom p und eine Sprache $L' \in P$, so dass für $A = \{0,1\}^{p(|x|)}$ gilt:
$$L = \{x : \exists y \in A : (x,y) \in L'\}$$

Wir wollen also eine Sprache konstruieren, die zu jedem Wort $x \in$ EdgeHop mit Hilfe eines polynomiell langen Wortes y (auch *Zeuge* genannt) EdgeHop entscheiden kann.

Lemma 2.2.1. EdgeHop $\in NP$.

Beweis. Wir konstruieren L' wie folgt:

$$L' := \{(x,y) : x \in \text{EdgeHop}, y \text{ ist eine gültige Zugfolge für } x.\}$$

Hierbei nehmen wir an, dass ein $x \in$ EdgeHop wie folgt aussieht:

$$x = \$e_1, e_2, ..., e_m \$ k_1, ..., k_n \$ v \$ z \$ v_1, ..., v_p \$$$

Wobei $e_1, ..., e_m$ die Kanten des Graphen, $k_1, ..., k_n$ die maximalen Belegtheiten der Knoten $1, ..., n$, v die Startposition des aktiven Steins, z den Zielknoten und $v_1, ..., v_p$ die Positionen der passiven Steine bezeichnet. Das Dollarzeichen \$ wird als Trennsymbol genutzt.[2] Der Graph in Abbildung 2.1 würde beispielsweise folgendermaßen kodiert sein:

$$\$ab, ad, bc, bd, cz \$ 2, 2, 1, 2, 1 \$ a \$ z \$ b, b, c \$$$

Eine gültige Zugfolge y kann nicht beliebig lang sein, denn für jede Kante kann es höchstens einen aktiven Zug geben, da sonst eine Kante doppelt genutzt werden

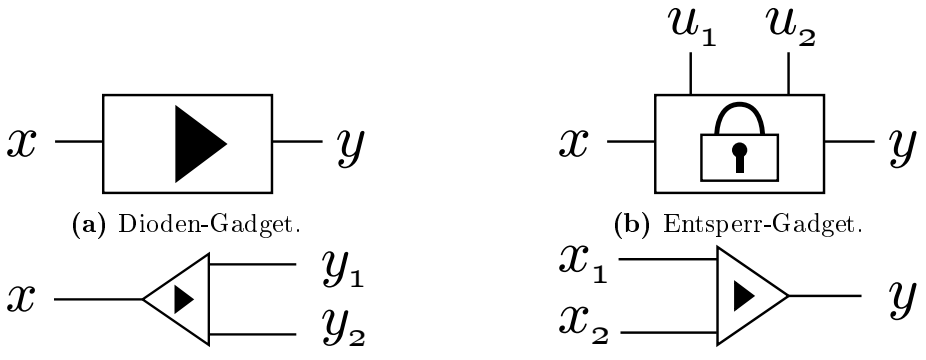
[2] Ob wir die Zahlen unär oder binär kodieren, macht in unserer Betrachtung keinen Unterschied, da wir über die Anzahl der Kanten (und passiven Steine) und nicht über deren kodierter Länge argumentieren.

würde, was den Regeln von Edge Hop widerspricht. Weiter können auf jeden aktiven Zug höchstens so viele passive Züge folgen, wie es passive Steine gibt, da diese nur *zum* aktiven Stein gezogen werden können. Sei p die Anzahl der passiven Steine. Einen Zug kann man durch (t, e) kodieren, wobei $t \in \{\text{aktiv}, \text{passiv}\}$ angibt, von welchem Typ der Zug ist und $e \in E$ angibt, welche Kante genutzt wird.[3] Sei $e_{max} = max_{e \in E}(|e|)$, also die Länge einer längsten Kodierung einer Kante, und sei $c \in \mathbb{R}$ so, dass $max_{e \in E}(|(t, e)|) \leq c \cdot e_{max}$. Dann gilt:

$$\begin{aligned} |y| &\leq c \cdot e_{max} \cdot |E| + c \cdot e_{max} \cdot |E| \cdot p \\ &= (p+1) \cdot c \cdot e_{max} \cdot |E| \\ &\leq |x| \cdot c \cdot e_{max} \cdot |E| \\ &\leq c \cdot e_{max} \cdot |x|^2 \\ &\leq c \cdot |x|^3 \end{aligned}$$

Wir sehen also, dass eine Zugfolge y eine zu $|x|$ polynomielle Länge hat. Damit ist $L' \in P$, da wir einfach testen können, ob die Zugfolge y für die in x kodierte Spielsituation gültig ist. Da $x \in$ EdgeHop genau dann, wenn x eine gültige Zugfolge besitzt, gilt also EdgeHop $= \{x : \exists y \in \{0,1\}^* : (x, y) \in L'\}$ und damit auch, dass EdgeHop $\in NP$. □

Um zu zeigen, dass EdgeHop NP-schwer ist, werden wir eine Reduktion von *3-SAT* vornehmen. Da *3-SAT* selbst NP-schwer ist (siehe [A26]), ist dann auch EdgeHop NP-schwer. Im folgenden werden wir Gadgets konstruieren, um zu einer gegebenen aussagenlogischen Formel in *3-KNF* einen Graphen zu konstruieren, auf dem Edge Hop gespielt werden kann. Hierzu konstruieren wir erst vier Grundgadgets: Dioden-, Entsperr-, Verzweigungs- und Zusammenführungs-Gadget. Die Symbole für diese Gadgets sind in Abbildung 2.2 zu sehen.



(a) Dioden-Gadget. **(b)** Entsperr-Gadget.

(c) Verzweigungs-Gadget. **(d)** Zusammenführungs-Gadget.

Abbildung 2.2: Die Symbole für die einzelnen Gadgets, die wir nutzen werden, damit die folgenden Abbildungen übersichtlicher sind.

Lemma 2.2.2 (Dioden-Gadget)**.** *Innerhalb des Dioden-Gadgets (Abbildung 2.3) gibt es eine gültige Zugfolge von x nach y und keine von y nach x.*[4]

[3] An dieser Stelle reicht die Angabe einer Kante, da die Position des aktiven Steins den Anfangs- und Endknoten eines Zuges vorgibt. Aktive Züge starten immer auf dem Knoten, auf dem sich der aktive Stein befindet und passive Züge enden immer auf dem Knoten, auf dem sich der aktive Stein befindet.

[4] Wir werden das Gadget so in den kompletten Graphen einbauen, dass es auch außerhalb des Gadgets keine gültige Zugfolge von x nach y gibt.

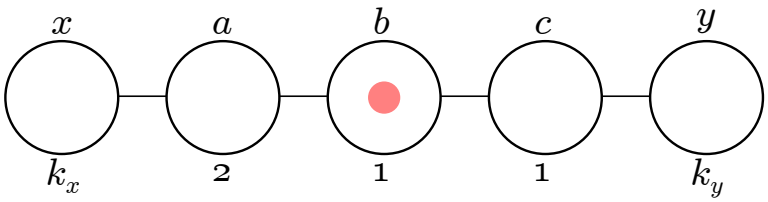


Abbildung 2.3: Das Dioden-Gadget als EdgeHop-Teilgraph. Dem aktiven Stein ist es nur möglich, dieses Gadget in einer bestimmten Richtung zu passieren.

Beweis. Eine gültige Zugfolge von x nach y ist $x \xrightarrow{a} a, a \xleftarrow{p} b, a \xrightarrow{a} b, b \xrightarrow{a} c, c \xrightarrow{a} y$. Damit ist der erste Teil des Lemmas gezeigt.

Nun bleibt noch zu zeigen, dass es keine gültige Zugfolge von y nach x gibt. Zuerst betrachten wir das Gadget im Ursprungszustand. Da wir Zugfolgen innerhalb des Gadgets betrachten, muss der aktive Stein über die Knoten c, b und a ziehen. Da auf b ein passiver Stein liegt und $k_b = 1$ ist $c \xrightarrow{a} b$ kein gültiger Zug. Ebenfalls ist $k_c = 1$ und somit ist auch $c \xleftarrow{p} b, c \xrightarrow{a} b$ nicht möglich. Es ist also nicht möglich, dass der aktive Stein im Ursprungszustand des Gadgets von c nach b zieht, also insbesondere auch nicht von y nach x. Der zweite Fall ist der, dass der passive Stein nicht mehr auf Knoten b liegt. Da $k_c = 1$ kann der passive Stein nicht über Knoten c gezogen worden sein. Er muss also über Knoten a gezogen wurden sein. Dies bedeutet allerdings, dass der aktive Stein zu einem vorherigen Zeitpunkt t Knoten a besucht haben muss und damit ist die Kante $xa \in X_t$ (und auch den nachfolgenden Mengen benutzter Kanten). In diesem Fall ist zwar die Zugfolge $c \xrightarrow{a} b, b \xrightarrow{a} a$ gültig, aber da die Kante xa schon benutzt wurde, kann der aktive Stein Knoten x nicht erreichen. Also gibt es auch in diesem Fall keinen gültige Zugfolge von y nach x und somit leistet das Dioden-Gadget die geforderten Eigenschaften. □

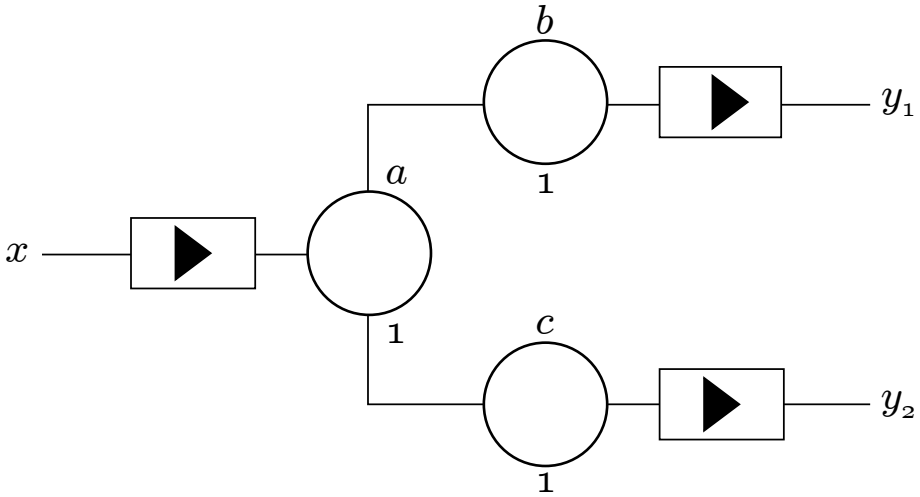


Abbildung 2.4: Ein Verzweigungs-Gadget konstruiert als EdgeHop-Teilgraph. Der aktive Stein kann das Gadget über einen von zwei möglichen Ausgängen zu verlassen.

Lemma 2.2.3 (Verzweigungs-Gadget). *Für alle $i, j \in \{1, 2\}$ mit $i \neq j$ gibt es innerhalb des Verzweigungs-Gadgets (Abbildung 2.4) gültige Zugfolgen von x nach y_i und keine gültigen Zugfolgen von y_i nach x oder von y_i nach y_j.*

Beweis. Wegen Lemma 2.2.2 folgt, dass es keine gültige Zugfolge von y_1 nach b und keine gültige Zugfolge von y_2 nach c gibt. Damit gibt es insbesondere keine gültige

Zugfolge von y_i nach x und keine gültige Zugfolge von y_i nach y_j (für $i, j \in \{1, 2\}$ mit $i \neq j$).

Ebenfalls folgt aus demselben Lemma, dass es gültige Zugfolgen von x nach a, von b nach y_1 und von c nach y_2 gibt. Um den ersten Teil des Lemmas zu zeigen reicht es also den Untergraphen $U_V = (\{a, b, c\}, \{ab, ac\})$ zu betrachten. Hier erfüllen die Zugfolgen $a \xrightarrow{a} b$ und $a \xrightarrow{a} c$ die geforderten Eigenschaften. □

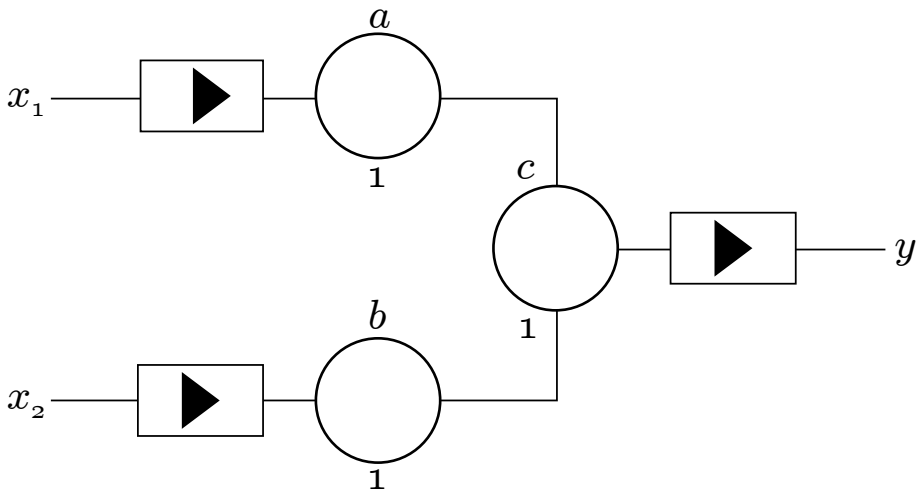


Abbildung 2.5: Ein Zusammenführungs-Gadget konstruiert als EdgeHop-Teilgraph. Der aktive Stein kann das Gadget nur über einen bestimmten Knoten verlassen, unabhängig davon, über welchen Eingangs-Knoten er das Gadget betritt.

Lemma 2.2.4 (Zusammenführungs-Gadget)**.** *Für alle $i, j \in \{1, 2\}$ mit $i \neq j$ gibt es innerhalb des Zusammenführungs-Gadget (Abbildung 2.5) gültige Zugfolgen von x_i nach y und keine gültigen Zugfolgen von y nach x_i oder von x_i nach x_j.*

Beweis. Wieder folgt aus Lemma 2.2.2, dass es keine gültige Zugfolge von a nach x_1 und keine gültige Zugfolge von b nach x_2 gibt. Damit gibt es insbesondere keine gültige Zugfolge von y nach x_i und keine gültige Zugfolge von x_i nach x_j (für $i, j \in \{1, 2\}$ mit $i \neq j$).

Weiter folgt aus demselben Lemma, dass es gültige Zugfolgen von x_1 nach a, von x_2 nach b und von c nach y gibt. Um den ersten Teil des Lemmas zu zeigen reicht es also den Untergraphen $U_Z = (\{a, b, c\}, \{ac, bc\})$ zu betrachten. Hier erfüllen die Zugfolgen $a \xrightarrow{a} c$ und $b \xrightarrow{a} c$ die geforderten Eigenschaften. □

Lemma 2.2.5 (Entsperr-Gadget)**.** *Innerhalb des Entsperr-Gadgets (Abbildung 2.6) gilt für alle $v, w \in \{u_1, u_2, x, y\}$ mit $v \neq w$: Es gibt eine gültige Zugfolge von v nach w genau dann, wenn $v = u_1$ und $w = u_2$ oder $v = x$ und $w = y$ und der aktive Stein hat das Gadget zu einem vorigen Zeitpunkt über Knoten u_1 betreten.*

Beweis. Zuerst zeigen wir die Existenz der gültigen Zugfolgen. Wegen Lemma 2.2.2 gibt es gültige Zugfolgen von u_1 nach p, von s nach u_2, von x nach a und von c nach y. Deswegen reicht es, wenn wir den Untergraphen $U_E = (\{a, b, c, p, q, r, s\}, \{ab, bc, br, pq, pr, qr, rs\})$ betrachten. Eine gültige Zugfolge von p nach s ist $p \xleftarrow{p} r, p \xrightarrow{a} r, r \xrightarrow{a} s$.

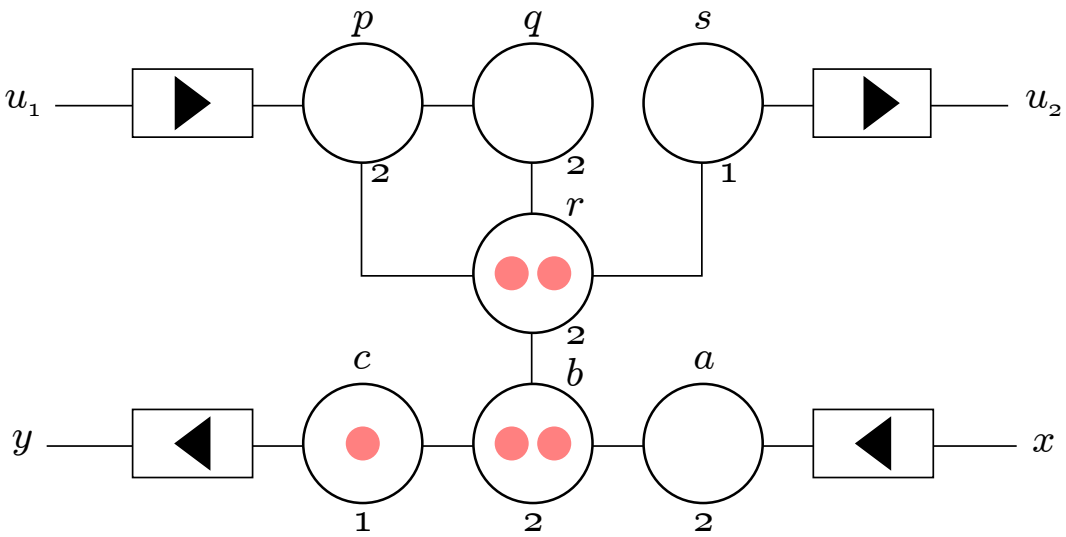


Abbildung 2.6: Ein Entsperr-Gadget. Der aktive Stein kann nur von x nach y ziehen, wenn er zu einem früheren Zeitpunkt das Gadget über u_1 betreten hat.

Für jede gültige Zugfolge von a nach c muss der der sich auf c befindende passive Stein verschoben werden (da $k_c = 1$). Wegen Lemma 2.2.2 kann der Stein nicht ins Dioden-Gadget gezogen werden[5] und deswegen muss $b \overset{p}{\leftarrow} c$ ausgeführt werden. Da $k_b = 2$ darf kein passiver Stein auf b liegen, wenn $b \overset{p}{\leftarrow} c$ ausgeführt wird. Da $k_a = 2$ kann höchstens *einmal* $a \overset{p}{\leftarrow} b$ ausgeführt werden. Der andere passive Stein auf b kann also weder nach a noch nach c gezogen werden. Es muss also mindestens einmal $r \overset{p}{\leftarrow} b$ ausgeführt werden, um beide passiven Steine von b zu verschieben. Auf r liegen ebenfalls zwei passive Steine. Um diese zu verschieben müssen die Züge $p \overset{p}{\leftarrow} r$ und $q \overset{p}{\leftarrow} r$ ausgeführt werden (da $k_p = k_q = 2$ und $k_s = 1$). Also muss der aktive Stein zu Knoten p ziehen. Er kann nicht von a nach p und dann innerhalb des Gadgets nach c ziehen, da er hierbei die Kante br mehr als einmal nutzen würde.[6] Damit der aktive Stein Knoten p erreichen kann, muss er das Gadget über u_1 betreten. Also gibt es keine Zugfolge von a nach c, wenn der aktive Stein nicht zuvor das Gadget über u_1 betreten hat. Angenommen die Züge $p \overset{p}{\leftarrow} r$, $p \overset{a}{\rightarrow} q$, $q \overset{p}{\leftarrow} r$, $q \overset{a}{\rightarrow} r$, $r \overset{p}{\leftarrow} b$, $r \overset{a}{\rightarrow} s$ wurden ausgeführt, bevor der aktive Stein Knoten a erreicht hat. Dann ist $a \overset{p}{\leftarrow} b$, $a \overset{a}{\rightarrow} b$, $b \overset{p}{\leftarrow} c$, $b \overset{a}{\rightarrow} c$ eine gültige Zugfolge von a nach c.

Nun müssen wir noch zeigen, dass es innerhalb des Gadgets keine gültigen Zugfolgen zwischen den anderen Knoten gibt. Wegen Lemma 2.2.2 gibt es keine gültigen Zugfolgen von u_1 nach x oder von x nach u_1. Ebenfalls folgt aus dem Lemma, dass es keine gültige Zugfolge von y oder von u_2 ausgehend gibt. Es bleibt also zu zeigen, dass es keine gültige Zugfolge von u_1 nach y und keine gültige Zugfolge von x nach u_2 gibt.

Zuerst zeigen wir, dass es keine gültige Zugfolge von u_1 nach y gibt. Wie oben angesprochen reicht es zu zeigen, dass es keine gültige Zugfolge von p nach c gibt. Wie vorhin argumentiert müsste jede solche Zugfolge $c \overset{p}{\leftarrow} b$ ausführen. Das wiederum heißt, dass sowohl $a \overset{p}{\leftarrow} b$, als auch $r \overset{p}{\leftarrow} b$ ausgeführt werden müssen. Angenommen

[5] Prinzipiell kann der passive Stein zwar zu Feld a des Dioden-Gadgets gezogen werden, aber dann kann der aktive Stein das Dioden-Gadget nicht mehr verlassen.

[6] Der aktive Stein kann auch nicht von a nach p, dann außerhalb des Gadgets nach x und dann nach c ziehen, da hierbei die Kante ab mehr als einmal genutzt werden würde.

der aktive Stein befindet sich zum Zeitpunkt t auf Knoten p. Wir betrachten nun zwei Fälle: Entweder der passive Zug $a \overset{p}{\leftarrow} b$ wurde zu einem vorigen Zeitpunkt ausgeführt oder nicht.

Fall 1: $a \overset{p}{\leftarrow} b$ wurde noch nicht ausgeführt. Das bedeutet, dass der aktive Stein erst von p nach a ziehen muss, um $a \overset{p}{\leftarrow} b$ auszuführen und dann von a nach c ziehen muss. Da allerdings sowohl jede Zugfolge von p nach a als auch jede Zugfolge von a nach c die Kante ab nutzt, ist dies nicht möglich.

Fall 2: $a \overset{p}{\leftarrow} b$ wurde zu einem vorherigen Zeitpunkt ausgeführt. Angenommen $ab \notin X_t$. Das bedeutet, dass $a \overset{a}{\rightarrow} b$ nicht ausgeführt wurde. Wegen Lemma 2.2.2 allerdings kann der aktive Stein auch nicht von a nach x gezogen sein, also muss sich der aktive Stein auf Knoten a befinden. Dies ist ein Widerspruch zu unserer Annahme, dass sich der aktive Stein auf Knoten p befindet. Deswegen muss $ab \in X_t$ gelten. Daraus folgt, dass der Zug $a \overset{a}{\rightarrow} b$ ausgeführt wurde. Da sich der aktive Stein auf Knoten p befindet, wurde entweder $b \overset{a}{\rightarrow} c$ oder $b \overset{a}{\rightarrow} r$ ausgeführt. Falls nun $bc \in X_t$, dann gibt es keine gültige Zugfolge von p nach c, da jede solche Zugfolge den Zug $b \overset{a}{\rightarrow} c$ beinhaltet. Falls $br \in X_t$, dann gibt es ebenfalls keine gültige Zugfolge von p nach c, da jede gültige Zugfolge den Zug $r \overset{a}{\rightarrow} b$ beinhaltet.[7]

Zuletzt zeigen wir noch, dass es keine gültige Zugfolge von x nach u_2 gibt. Auch hier reicht es wieder zu zeigen, dass es keine gültige Zugfolge von a nach s gibt. Es ist offensichtlich, dass jede solche Zugfolge nur dann gültig ist, wenn mindestens einer der beiden auf r liegenden passiven Steine verschoben wurde. Hier unterscheiden wir ob $p \overset{p}{\leftarrow} r$ oder $b \overset{p}{\leftarrow} r$ ausgeführt wurde.[8]

Fall 1: $p \overset{p}{\leftarrow} r$ wurde ausgeführt. Wegen Lemma 2.2.2 und der Tatsache, dass es keine gültige Zugfolge von p nach c gibt, muss der aktive Stein von p nach s gezogen sein um das Gadget verlassen zu können. Das heißt insbesondere, dass der Zug $r \overset{a}{\rightarrow} s$ ausgeführt wurde. Wenn der Stein nun zu einem Zeitpunkt t wieder zu Knoten a zieht, dann gibt es keine gültige Zugfolge mehr von a nach s, da jede solche Zugfolge den Zug $r \overset{a}{\rightarrow} s$ beinhaltet, aber $rs \in X_t$ gilt.

Fall 2: $b \overset{p}{\leftarrow} r$ wurde ausgeführt. Dies ist nur möglich, wenn *beide* passiven Steine von b verschoben wurden. Dies ist allerdings nicht möglich, da $k_c = 1$ und $k_a = 2$. □

Wir haben nun alle Gadgets konstruiert, die wir benötigen, um eine Spielsituation aus einer Formel in *3-KNF* zu konstruieren. Abbildung 2.2 zeigt die Symbole, welche wir im Folgenden für die jeweiligen Gadgets verwenden werden. Abbildung 2.7 zeigt den schematischen Aufbau unseres Graphen. Wir müssen als nächstes die Variablen- und Klausel-Gadgets aus unseren vorher eingeführten Gadgets konstruieren und danach die einzelnen Gadgets untereinander passend verbinden.

Abbildung 2.8 zeigt ein Variablen-Gadget. Die jeweiligen Variablen-Gadget-Teilgraphen werden im gesamten EdgeHop-Graph so eingebettet sein, dass die Knoten t_{out} und f_{out} mit den jeweils ersten Klausel-Gadgets verbunden werden, welche die zugehörigen Literale enthalten. Die Knoten t_{in} und f_{in} werden mit den jeweils letzten Klausel-Gadgets verbunden, welche die zugehörigen Literale enthalten. Durch den Aufbau der Klausel-Gadgets bzw. des gesamten Graphen wird es so sein, dass es von t_{out} nur eine gültige Zugfolge nach t_{in} und von f_{out} nur eine gültige Zugfolge nach f_{in} gibt.[9]

[7] Eine Zugfolge von r nach u_2, dann nach x und dann nach b wäre ebenfalls nicht gültig, da hier die Kante ab mehrfach benutzt werden würde.

[8] Der Fall, dass $q \overset{p}{\leftarrow} r$ ausgeführt wurde ist gleich zu dem Fall, dass $p \overset{p}{\leftarrow} r$ ausgeführt wurde.

[9] Natürlich gibt es dann auch gültige Zugfolgen von t_{out} zu allen Knoten auf dem Pfad zu t_{in}

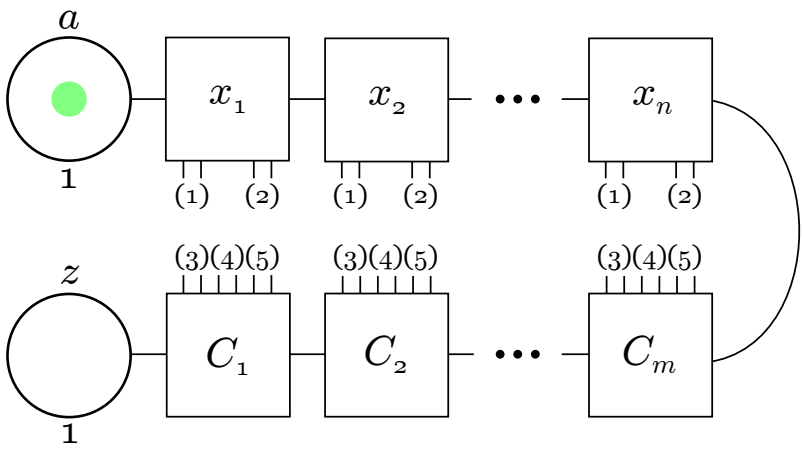


Abbildung 2.7: Der schematische Aufbau des Graphen mit dem aktiven Stein auf Knoten a, dem Zielknoten z, Variablengadgets x_1 bis x_n und Klauselgadgets C_1 bis C_m. Die Kanten bei (1) verbinden die Variablengadgets mit den Klauselgadgets, die die jeweilige Variable in nicht-negierter Form enthalten, die Kanten bei (2) verbinden die Variablengadgets mit den Klauselgadgets, die die jeweilige Variable in negierter Form enthalten. Die Kanten bei (3), (4) und (5) entsprechen dem ersten, zweiten und dritten Literal der zugehörigen Klausel und sind mit den entsprechenden Variablengadgets verbunden.

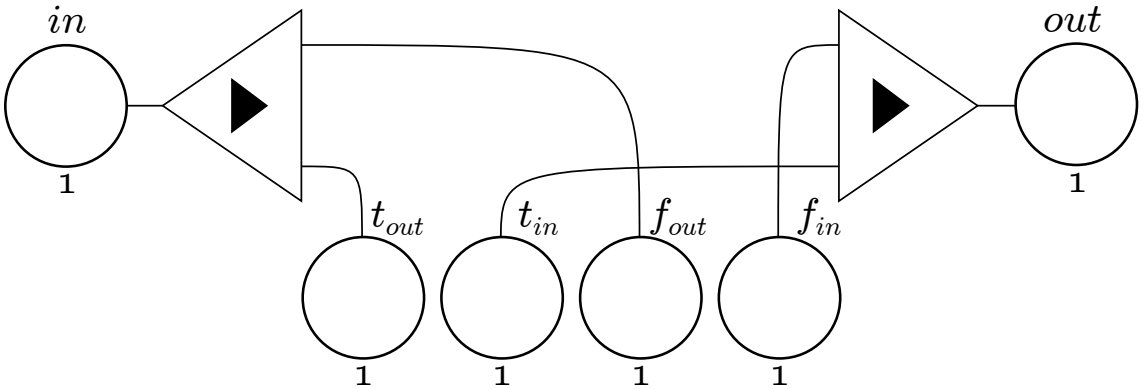


Abbildung 2.8: Ein Variablen-Gadget. Der aktive Stein betritt das Gadget über den Knoten *in* und verlässt es letztendlich über den Knoten *out*. Die Aufgabe dieses Gadgets ist es, den Spieler dazu zu bringen, sich für eine Variablenbelegung zu entscheiden. Hierbei bedeutet ein Ziehen über den Knoten t_{out}, dass die zugehörige Variable den Wahrheitswert *„wahr“* zugewiesen bekommt. Analog bedeutet ein Ziehen über f_{out}, dass die Variable den Wahrheitswert *„falsch“* zugewiesen bekommt.

Lemma 2.2.6 (Variablen-Gadget). *Im Variablen-Gadget (Abbildung 2.8) gibt es gültige Zugfolgen von in nach t_{out} und nach f_{out}. Und es gibt gültige Zugfolgen von t_{in} und von f_{in} nach out.*[10] *Weiter gibt es keine gültigen Zugfolgen von t_{out} nach f_{out} (und umgekehrt) und von t_{in} nach f_{in} (und umgekehrt).*

Beweis. Die gewünschten Eigenschaften folgen aus den Funktionsweisen der Verzweigungs- und Zusammenführungs-Gadgets. (Lemma 2.2.3 und Lemma 2.2.4). □

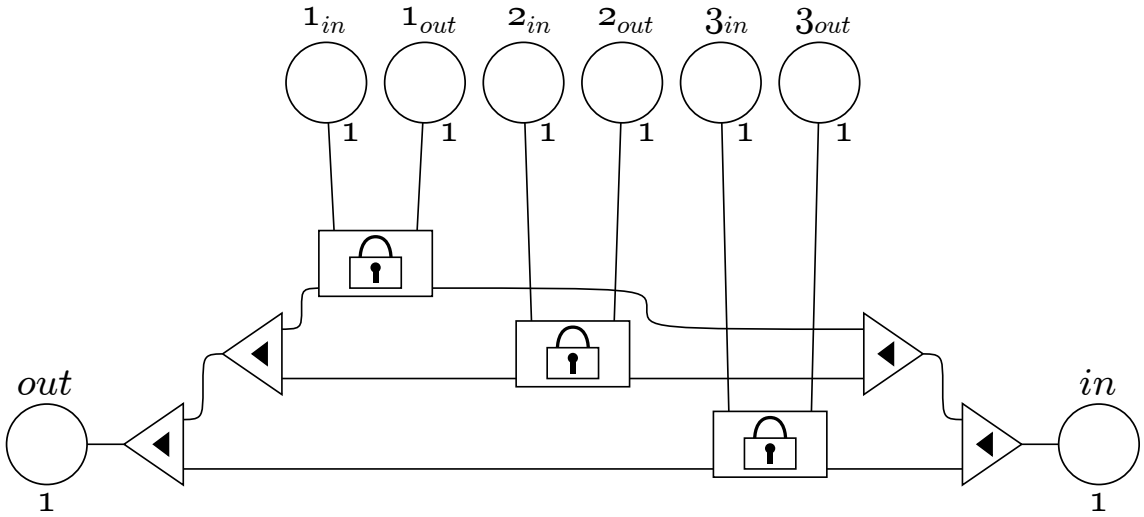


Abbildung 2.9: Ein Klausel-Gadget. Der aktive Stein betritt das Gadget über den Knoten *in* und verlässt es letztendlich über den Knoten *out*. Die Aufgabe dieses Gadgets ist es, zu überprüfen, ob die korrespondierende Klausel erfüllt ist. Hierbei entsprechen die Knoten 1_{in} und 1_{out} dem ersten Literal der Klausel, 2_{in} und 2_{out} dem zweiten Literal und 3_{in} und 3_{out} dem dritten Literal.

Abbildung 2.9 zeigt ein Klausel-Gadget. Die jeweiligen Klausel-Gadget-Teilgraphen werden im gesamten EdgeHop-Graph so eingebettet sein, dass die Knoten 1_{in}, 2_{in} und 3_{in} entweder mit dem korrespondierenden Variablen-Gadget verbunden werden, wenn keine frühere Klausel das Literal enthielt, oder sonst mit dem vorherigen Klausel-Gadget, welches das korrespondierende Literal enthielt. Analog werden die Knoten die Knoten 1_{out}, 2_{out} und 3_{out} mit dem nächsten Klausel-Gadget verbunden, welches das zugehörige Literal enthält oder, wenn kein anderes Klausel-Gadget mehr das Literal enthält, mit dem zugehörigen Variablen-Gadget. Wir nennen die Knoten 1_{in}, 2_{in} und 3_{in} *entsperrende* Knoten.

Lemma 2.2.7 (Klausel-Gadget). *Im Klausel-Gadget (Abbildung 2.9) gibt es gültige Zugfolgen von 1_{in} nach 1_{out}, von 2_{in} nach 2_{out} und von 3_{in} nach 3_{out}. Ebenfalls gibt es eine gültige Zugfolge von in nach out, wenn der aktive Stein zuvor über 1_{in} und 1_{out}, oder über 2_{in} und 2_{out} oder über 3_{in} und 3_{out} gezogen ist.*

Beweis. Die Existenz einer gültigen Zugfolge von 1_{in} nach 1_{out} (und nur nach 1_{out}) folgt aus Lemma 2.2.5. (Analog folgt die Existenz der Zugfolgen von 2_{in} nach 2_{out} (und nur nach 2_{out}) und von 3_{in} nach 3_{out} (und nur nach 3_{out}) aus demselben Lemma.)

und gültige Zugfolgen von f_{out} zu allen Knoten auf dem Pfad zu f_{in}.

[10] Durch den restlichen Aufbau des Graphen ergibt sich damit, dass es eine gültige Zugfolge von x_{in} nach x_{out} gibt bei der der aktive Stein entweder über t_{out} oder über f_{out} zieht.

Damit der aktive Stein von *in* nach *out* ziehen kann, muss er eines der Entsperr-Gadgets durchqueren. Ebenfalls liefert Lemma 2.2.5 wieder, dass dies nur möglich ist, wenn der aktive Stein vorher über einen der Knoten 1_{in}, 2_{in} oder 3_{in} gezogen ist. Lemma 2.2.3 liefert, dass es eine gültige Zugfolge von *in* zu jedem der Entsperr-Gadgets gibt und aus Lemma 2.2.4 folgt, dass es eine gültige Zugfolge von jedem der Entsperr-Gadgets zu *out* (und nur nach *out*) gibt. □

Damit haben wir nun alle Einzelteile, um aus einer gegebenen *3-SAT*-Formel einen EdgeHop-Graphen zu konstruieren und damit die NP-Schwere zu zeigen. Hierzu wollen wir noch die folgenden Schreibweisen einführen:

Sei ϕ eine aussagenlogische Formel in 3-KNF mit m Klauseln und n Variablen.

- $\hat{x}^i$ bezeichnet das zur Variable x_i korrespondierende Variablen-Gadget.
- $\hat{x}^i_v$ bezeichnet Knoten v in dem durch $\hat{x}^i$ induzierten (Teil-)Graphen.
- $\hat{C}^i$ bezeichnet das zur i-ten Klausel korrespondierende Klausel-Gadget.
- $\hat{C}^i_v$ bezeichnet Knoten v in dem durch $\hat{C}^i$ induzierten (Teil-)Graphen.

Lemma 2.2.8 (NP-Schwere)**.** EdgeHop *ist NP-schwer.*

Beweis. Sei ϕ eine aussagenlogische Formel in 3-KNF mit m Klauseln und n Variablen. Also $\phi = \bigwedge_{k=1}^{m} (x_{k_1} \vee x_{k_2} \vee x_{k_3})$ mit $x_{k_1}, x_{k_2}, x_{k_3} \in \{x_j, \neg x_j : 1 \leq j \leq n\}$. Wir definieren $f(\phi) := (G, k, a, z, P)$. a ist hierbei der Start- und z der Zielknoten. Weiter gilt für die Funktion der maximalen Belegtheiten $k_G = k : V(G) \to \mathbb{N}$:

$$k(v) := \begin{cases} 1, & v \in \{a, z\} \\ k_{\hat{x}^i}(v), & v \in V(\hat{x}^i) \\ k_{\hat{C}^j}(v), & v \in V(\hat{C}^j) \end{cases}$$

Die neuen Knoten haben also eine maximale Belegtheit von 1 und alle anderen Knoten übernehmen die maximale Belegtheit, die sie in ihrem Gadget-Teilgraphen besitzen. Die Multimenge $P_G = P$ ist ebenso die große Vereinigung aller Multimengen der einzelnen Gadget-Teilgraphen, da auf a oder z keine passiven Steine liegen. Es gilt also:

$$P := \biguplus_{i=1}^{n} P_{\hat{x}^i} \ \uplus \ \biguplus_{j=1}^{m} P_{\hat{C}^j}$$

Die Knotenmenge unseres EdgeHop-Graphen $\mathcal{EH}$ ist die Vereinigung der Knotenmengen aus den jeweils zu einer Variablen oder Klausel korrespondierenden Gadget-Graphen. Zusätzlich nehmen wir noch den Startknoten a und den Zielknoten z mit auf. Es gilt also:

$$V(\mathcal{EH}) := \{a, z\} \ \cup \ \bigcup_{k=1}^{m} V(\hat{C}^m) \ \cup \ \bigcup_{k=1}^{n} V(\hat{x}^k)$$

Weiter setzen wir die Kantenmenge unseres EdgeHop-Graphen auf $E(G) := E_s \cup E_g \cup E_v \cup E_C \cup E_r$. Hierbei enthält E_s eine Kante, die den Startknoten mit dem Variablen-Gadget $\hat{x}^1$ ganz links verbindet, eine Kante, die den Zielknoten mit dem

Klausel-Gadget $\hat{C}^1$ ganz links verbindet und eine Kante, die das Variablen-Gadget $\hat{x}^n$ ganz rechts mit dem Klausel-Gadget $\hat{C}^m$ ganz rechts verbindet. E_g enthält die Kanten der durch die jeweiligen Gadgets induzierten Teilgraphen. E_v enthält die Kanten, die die Variablen-Gadgets untereinander verbinden. E_C enthält die Kanten, die die Klausel-Gadgets untereinander verbinden. E_r enthält die Kanten, die Variablen-Gadgets mit den entsprechenden Klausel-Gadgets verbinden. Formal gilt also:

$$
\begin{aligned}
E_s &= \{a\hat{x}^1_{in}, z\hat{C}^1_{out}, \hat{x}^n_{out}\hat{C}^m_{in}\} \\
E_g &= \bigcup_{k=1}^{m} E(\hat{C}^m) \ \cup \ \bigcup_{k=1}^{n} E(\hat{x}^k) \\
E_v &= \bigcup_{i=1}^{n-1} \{\hat{x}^i_{out}\hat{x}^{i+1}_{in}\} \\
E_C &= \bigcup_{i=1}^{m-1} \{\hat{C}^i_{in}\hat{C}^{i+1}_{out}\}
\end{aligned}
$$

Um die Menge E_r zu definieren brauchen wir eine Funktion $Lit : \{1, ..., m\} \times \{1, 2, 3\} \to \bigcup_{i=1}^{n}\{x_i, \neg x_i\}$, die folgendermaßen aufgebaut ist:[11]

$$
Lit(i, p) := \begin{cases} x, & C_i = (x \vee y \vee z), p = 1 \\ y, & C_i = (x \vee y \vee z), p = 2 \\ z, & C_i = (x \vee y \vee z), p = 3 \end{cases}
$$

Da die Menge E_r drei Arten von Kanten beinhaltet, setzen wir $E_r = E_f \cup E_b \cup E_l$. E_f enthält die Kanten zwischen einem Variablen-Gadget $\hat{x}^i$ und den ersten Klausel-Gadgets, welche x_i oder $\neg x_i$ als Literal enthalten. E_b enthält die Kanten zwischen den Klausel-Gadgets, die die gleichen Literale enthalten und E_l enthält die Kanten zwischen einem Variablen-Gadget $\hat{x}^i$ und den letzten Klausel-Gadgets, welche x_i oder $\neg x_i$ als Literal enthalten. Formal gilt also:

$$
\begin{aligned}
E_f = \{&\hat{x}^i_{t_{out}}\hat{C}^j_{p_{in}} : (Lit(j, p) = x_i) \\
&\wedge (\forall k \in [1, j) \forall p' \in [1, p] \ Lit(k, p') \neq x_i) \\
&\wedge (\forall p' \in [1, p) \ Lit(j, p') \neq x_i)\} \\
\cup \{&\hat{x}^i_{f_{out}}\hat{C}^j_{p_{in}} : (Lit(j, p) = \neg x_i) \\
&\wedge (\forall k \in [1, j) \forall p' \in [1, p] \ Lit(k, p') \neq \neg x_i) \\
&\wedge (\forall p' \in [1, p) \ Lit(j, p') \neq \neg x_i)\}
\end{aligned}
$$

Hierbei entspricht die linke Seite der Vereinigung den nicht-negierten und die rechte Seite den negierten Variablen. In E_f sind also die Kanten zwischen dem *wahr* entsprechenden Knoten t_{out} eines Variablen-Gadgets $\hat{x}^i$ und einem Klausel-Gadget $\hat{C}^i$ an Knoten p_{in}, wenn das Literal in Klausel C_i an Position p genau x_i ist und wenn bei allen früheren Klauseln an jeder Position nicht x_i als Literal steht und in Klausel

[11] Die Funktion *Lit* gibt an, welches Literal an Position p der Klausel C_i steht. Für eine Formel $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_4 \vee \neg x_5 \vee \neg x_6)$ ist beispielsweise $Lit(1, 3) = x_3$ und $Lit(2, 2) = \neg x_5$.

C_i an den Positionen vor p ebenfalls nicht x_i steht. (Analog: Der Knoten f_{out}, der der Belegung *falsch* entspricht, mit dem Literal $\neg x_i$ in den Klauseln.)

$$\begin{aligned} E_l = \{\hat{x}^i_{t_{in}}\hat{C}^j_{p_{out}} : (&Lit(j,p) = x_i) \\ &\wedge (\forall k \in (j,m] \forall p' \in [1,p]\ Lit(k,p') \neq x_i) \\ &\wedge (\forall p' \in (p,3]\ Lit(j,p') \neq x_i)\} \\ \cup \{\hat{x}^i_{f_{in}}\hat{C}^j_{p_{out}} : (&Lit(j,p) = \neg x_i) \\ &\wedge (\forall k \in (j,m] \forall p' \in [1,p]\ Lit(k,p') \neq \neg x_i) \\ &\wedge (\forall p' \in (p,3]\ Lit(j,p') \neq \neg x_i)\} \end{aligned}$$

Hierbei entspricht wieder die linke Seite der Vereinigung den nicht-negierten und die rechte Seite den negierten Variablen. In E_l sind also die Kanten zwischen dem *wahr* entsprechenden Knoten t_{in} eines Variablen-Gadgets $\hat{x}^i$ und einem Klausel-Gadget $\hat{C}^i$ an Knoten p_{out}, wenn das Literal in Klausel C_i an Position p genau x_i ist und wenn bei allen späteren Klauseln an jeder Position nicht x_i als Literal steht und in Klausel C_i an den Positionen nach p ebenfalls nicht x_i steht. (Analog: Der Knoten f_{in}, der der Belegung *falsch* entspricht, mit dem Literal $\neg x_i$ in den Klauseln.)

$$\begin{aligned} E_b = \{\hat{C}^i_{p_{out}}\hat{C}^j_{q_{in}} : (&Lit(i,p) = Lit(j,q)) \\ &\wedge (\forall k \in (i,j) \forall p' \in [1,3]\ Lit(k,p') \neq Lit(i,p))\} \end{aligned}$$

Diese Menge enthält die Kanten zwischen zwei Klausel-Gadgets $\hat{C}^i$ und $\hat{C}^j$ an den Knoten p_{out} und q_{in} wenn das Literal in Klausel C_i an Position p gleich dem Literal in Klausel C_j an Position q ist und zusätzlich in allen Klauseln zwischen C_i und C_j an keiner Position dieses Literal erneut vorkommt.

Um nun zu zeigen, dass EdgeHop *NP-schwer* ist, müssen wir zum einen zeigen, dass es im konstruierten Graphen genau dann eine gültige Zugfolge von a nach z gibt, wenn die zugehörige Formel ϕ erfüllbar ist und zum anderen müssen wir zeigen, dass ein solcher Graph in Polynomzeit konstruiert werden kann.

3-SAT $\leq_p$ EdgeHop. Angenommen ϕ ist erfüllbar. Dann gibt es eine Belegung α der Variablen $x_1, ..., x_n$ so dass $\hat{\alpha}(\phi) = 1$.[12] Aus Lemma 2.2.6 und Lemma 2.2.7 folgt, wie der aktive Stein ziehen muss.[13] Die einzige Zugmöglichkeit, die nicht fest vorgegeben ist, ist ob der aktive Stein bei einem Variablen-Gadget $\hat{x}^i$ über $\hat{x}^i_{t_{out}}$ oder $\hat{x}^i_{f_{out}}$ zieht. An dieser Stelle kommt nun die Belegung α ins Spiel: Falls $\alpha(x_i) = 1$, dann zieht der aktive Stein über $\hat{x}^i_{t_{out}}$ und falls $\alpha(x_i) = 0$, dann zieht der aktive Stein über $\hat{x}^i_{f_{out}}$.

Da α eine erfüllende Belegung für ϕ ist, ist jede Klausel $C_1, ..., C_m$ erfüllt. Dies bedeutet, dass innerhalb jeder dieser Klauseln C_j mindestens ein Literal ℓ *wahr* ist. Falls ℓ von der Form x_i (für $x_i \in \{x_1, ..., x_n\}$), dann zieht der aktive Stein über

[12] Siehe für diese Schreibweise [A18].

[13] Der aktive Stein zieht von a zum ersten Variablen-Gadget. Von dort zu allen korrespondierenden Klausel-Gadgets (abhängig von der Belegung der Variable). Dann zieht er zurück zum selben Variablen-Gadget und von dort zum nächsten Variablen-Gadget. Dann wieder zu allen korrespondieren Klausel-Gadgets usw. bis der aktive Stein wieder zum letzten Variablen-Gadget zurückkehrt. Von dort muss er zum letzten Klausel-Gadget, von diesem zum vorletzten, usw. bis der aktive Stein beim ersten Klausel-Gadget angekommen ist. Von dort kann er nur nach z ziehen.

$\hat{x}^i_{t_{out}}$. Aus unserer Konstruktion des Graphen folgt allerdings, dass der aktive Stein auch $\hat{C}^j$ (über einen der entsperrenden Knoten) betritt und ermöglicht somit eine gültige Zugfolge von $\hat{C}^j_{in}$ nach $\hat{C}^j_{out}$. Das heißt, dass Gadget $\hat{C}^j$ durchquert werden kann. (Analog, falls ℓ von der Form $\neg x_i$ (für $x_i \in \{x_1, ..., x_n\}$).) Dies bedeutet, dass jedes Klausel-Gadget mindestens einmal über einen entsperrenden Knoten betreten wird und somit sind alle Klausel-Gadgets durchquerbar und der aktive Stein kann Knoten z erreichen. Zusammengefasst gilt also: Wenn ϕ erfüllbar ist, dann gibt es eine gültige Zugfolge von a nach z in unserem Graphen.

Nehmen wir nun an, in unserem (aus einer Formel konstruierten) Graphen gibt es eine gültige Zugfolge von a nach z. Dann definieren wir für diese:

$$\alpha(x_i) := \begin{cases} 1, & \text{wenn der aktive Stein von } \hat{x}^i_{in} \text{ nach } \hat{x}^i_{t_{out}} \text{ zieht} \\ 0, & \text{wenn der aktive Stein von } \hat{x}^i_{in} \text{ nach } \hat{x}^i_{f_{out}} \text{ zieht} \end{cases}$$

Diese Funktion ist wohldefiniert, denn: Angenommen es gibt ein x_i, so dass $\alpha(x_i)$ undefiniert ist. Dann gilt für alle $w \in \{t, f\}$, dass der aktive Stein nicht von $\hat{x}^i_{in}$ nach $\hat{x}^i_{w_{out}}$ zieht. Das bedeutet weiter, dass der aktive Stein nicht von $\hat{x}^i_{in}$ nach $\hat{x}^i_{out}$ zieht. Dies ist ein Widerspruch dazu, dass die Zugfolge gültig ist, denn nach unserer Konstruktion des Graphen zieht der aktive Stein bei jeder gültigen Zugfolge von a nach z durch jedes Variablen-Gadget. Angenommen es gibt ein x_i, so dass $\alpha(x_i)$ nicht eindeutig ist. Das heißt, es gibt ein $1 \leq i \leq n$ so, dass der aktive Stein von $\hat{x}^i_{in}$ nach $\hat{x}^i_{t_{out}}$ und nach $\hat{x}^i_{f_{out}}$ zieht. Dies ist ein Widerspruch dazu, dass die Zugfolge gültig ist, da das Verzweigungs-Gadget innerhalb des Variablen-Gadgets nur *einmal* durchquert werden kann. Das heißt, die Zugfolge definiert eine eindeutige und vollständige Belegung der Variablen. Ebenfalls folgt aus der Gültigkeit der Zugfolge, dass jedes Klausel-Gadget durchquerbar ist. Dies bedeutet, dass innerhalb jeder Klausel ein Literal *wahr* ist und somit ist α eine erfüllende Belegung.

Polynomzeit-Konstruktion. Die Laufzeit der Konstruktion des Graphen hängt von der Laufzeit der Konstruktionen der Knotenmenge und der oben verwendeten, einzelnen Kantenmengen ab. Ein einzelnes Variablen-Gadget und auch ein einzelnes Klausel-Gadget kann jeweils in konstanter Zeit konstruiert werden, also gilt, dass $V(\mathcal{EH})$ in Zeit $\mathcal{O}(n+m)$ konstruiert werden kann. Aus dem gleichen Grund kann auch E_g in Zeit $\mathcal{O}(n+m)$ konstruiert werden. E_s beinhaltet immer diese drei Kanten und kann deswegen in konstanter Zeit konstruiert werden. Da für jede Variable (bis auf eine) eine Kante zu E_v hinzugefügt wird, kann diese Menge in Zeit $\mathcal{O}(n)$ konstruiert werden. Analog gilt, dass E_C in Zeit $\mathcal{O}(m)$ konstruiert werden kann. Um die Laufzeit abzuschätzen, die wir für die Konstruktion der Menge E_r benötigen, schauen wir uns auch dort an, wie die Laufzeiten zur Konstruktion der Teilmengen E_f, E_b und E_l sind. Ein naiver Ansatz, um E_f zu konstruieren wäre, für jede Variable einmal die Formel von links nach rechts abzusuchen, bis man das erste Vorkommen der Variable (und das erste Vorkommen der negierten Variable) innerhalb der Formel findet. Hierbei zählt man mit, in welcher Klausel man gerade sucht und auch, welche Position innerhalb der Klausel man gerade betrachtet. Die Laufzeit für die Konstruktion von E_f ist also $\mathcal{O}(n \cdot m)$.[14] Für die Konstruktion von E_l kann man analog vorgehen. Hier sucht man die Formel allerdings von rechts nach links ab, um das jeweils letzte Vorkommen zu finden. Diese Laufzeit ist also

[14] Die Länge der Formel ϕ ist im Wesentlichen von der Anzahl der Klauseln abhängig, da die Formel in 3-KNF vorliegt und somit jede Klausel gleichlang ist. Ihre Länge ist also $\mathcal{O}(m)$.

auch $\mathcal{O}(n \cdot m)$. Zuletzt müssen wir und noch ansehen, in welcher Laufzeit man E_b konstruieren kann. Auch hier müssen wir (in einem naiven Ansatz) für jede Variable (und ihre Negation) die Formel absuchen um zwei aufeinanderfolgende Klauseln zu finden, die dasselbe Literal enthalten. Hierbei muss man wieder zählen, welche Klausel man gerade betrachtet, welche Position innerhalb einer Klausel man betrachtet und auch speichern, in welcher Klausel an welcher Position das gerade bearbeitete Literal zuletzt vorkam. Auch hier ist die Laufzeit $\mathcal{O}(n \cdot m)$, da man für jede Variable die Formel einmal (oder zweimal – je nach dem, ob man negierte und nicht-negierte Variablen separat verarbeitet) absuchen muss. Es können also alle Teilmengen in Polynomzeit konstruiert werden. Es gilt:

$$\begin{aligned}&\mathcal{O}(n+m) + \mathcal{O}(n+m) + \mathcal{O}(1) + \mathcal{O}(n) + \mathcal{O}(m) + \mathcal{O}(n \cdot m) + \mathcal{O}(n \cdot m) + \mathcal{O}(n \cdot m)\\ &= \mathcal{O}(n \cdot m)\end{aligned}$$

Und deshalb ist die Konstruktion des gesamten Graphen auch in Polynomzeit möglich. □

Satz 2.2.9 (NP-Vollständigkeit)**.** EdgeHop *ist NP-vollständig.*

Beweis. Nach Lemma 2.2.8 is EdgeHop NP-schwer und nach Lemma 2.2.1 ist EdgeHop $\in NP$. Damit ist EdgeHop nach Definition NP-vollständig. □

2.3 Alternativer Beweis

In diesem Abschnitt werden wir uns eine Alternative anschauen, die die NP-Schwere von EdgeHop zeigt. Wir werden eine Reduktion vom (gerichteten) Hamiltonkreisproblem vornehmen, welches bekanntermaßen auch NP-vollständig ist. [A26] Wir werden also einen EdgeHop-Graphen aus einem beliebigen Graphen G so konstruieren, dass es genau dann eine gültige Zugfolge (von a nach z) gibt, wenn G einen Hamiltonkreis besitzt. Die Idee hierbei ist, jeden Knoten des Graphen G durch ein (modifiziertes) Entsperr-Gadget zu ersetzen, so dass der aktive Stein für jeden Knoten des ursprünglichen Graphen ein Gadget entsperren muss, bevor er danach durch die Entsperr-Gadgets zum Zielknoten ziehen kann. Wir werden also die gleichen Grund-Gadgets nutzen wie im vorigen Abschnitt, werden sie allerdings anders zusammen bauen. Abbildung 2.10 zeigt den schematischen Aufbau des Graphen.

Bis jetzt haben unsere Verzweigungs- und Zusammenführ-Gadgets zwei Aus- bzw. Eingänge gehabt. Für den jetzigen Beweis ist es einfacher, wenn wir diese Gadgets mit beliebig vielen Aus- bzw. Eingängen nutzen können. Durch Kaskadieren unserer alten Gadgets ist dies allerdings problemlos möglich (Abbildung 2.11 zeigt dazu ein Beispiel).

Wie angesprochen wollen wir jeden Knoten des ursprünglichen Graphen G durch ein (modifiziertes) Entsperr-Gadget, genannt *Knoten-Gadget*, ersetzen. Die Ausgangsknoten eines Knoten-Gadgets werden genau dann mit den Eingangsknoten eines anderen Knoten-Gadgets verbunden, wenn es eine Kante vom ersten Knoten zum zweiten gibt. Abbildung 2.12 zeigt ein Knoten-Gadget. Wir nennen die Knoten $x_1, ..., x_k$ und $y_1, ..., y_l$, wie auch beim Klausel-Gadget zuvor, *entsperrende* Knoten.

Lemma 2.3.1 (Knoten-Gadget)**.** *Seien $i, i' \in \{1, ..., l\}$ und $j, j' \in \{1, ..., k\}$ (mit $i \neq i'$ und $j \neq j'$) beliebig. Für das Knoten-Gadget (Abbildung 2.12) gilt:*

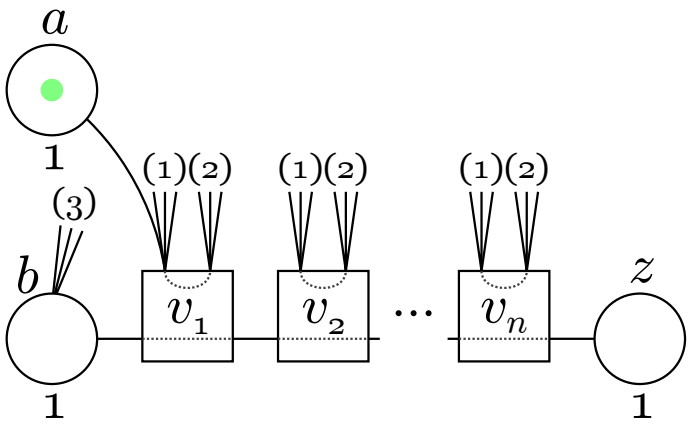
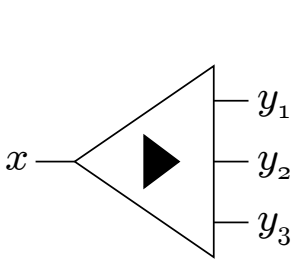
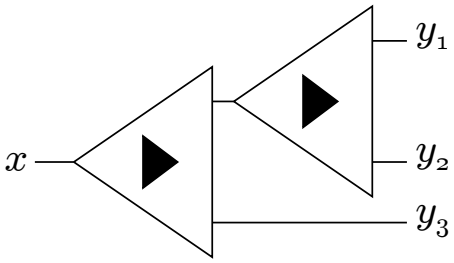


Abbildung 2.10: Der schematische Aufbau des EdgeHop-Graphen, konstruiert aus einem beliebigen Graphen G. Wie üblich ist a der Startknoten und z der Zielknoten. Die Kanten bei (1) entsprechen den jeweils eingehenden Kanten und die Kanten bei (2) den jeweils ausgehenden Kanten des zugehörigen Knotens. Die Kanten bei (3) führen zu allen Knoten, die eine ausgehende Kante zu a haben.



(a) Das Entsperr-Gadget wie wir es nutzen werden.



(b) Das Entsperr-Gadget aufgebaut aus zwei üblichen Entsperr-Gadgets.

Abbildung 2.11: Ein Verzweigungs-Gadget mit drei Ausgängen, konstruiert aus Verzweigungs-Gadgets mit je zwei Ausgängen. Ein Zusammenführ-Gadget kann man analog konstruieren.

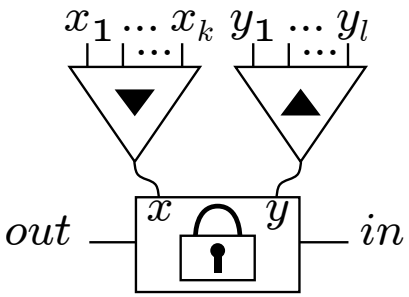


Abbildung 2.12: Ein Knoten-Gadget. Damit der aktive Stein im Gadget von *in* nach *out* ziehen kann, muss er es zuvor über einen der Knoten $x_1, ..., x_k$ betreten haben. Die Knoten $x_1, ..., x_k$ entsprechen den eingehenden Kanten und die Knoten $y_1, ..., y_l$ entsprechen den ausgehenden Kanten des korrespondieren Knotens. x entspricht u_1 und y entspricht u_2 aus Abbildung 2.6

1. *Es gibt nur dann eine gültige Zugfolge von in nach out, wenn das Gadget zuvor über einen Knoten x_i betreten wurde.*

2. *Es gibt keine gültige Zugfolgen von y_i nach x_j, von x_i nach $x_{i'}$, oder von y_j nach $y_{j'}$.*

3. *Es gibt nur dann eine gültige Zugfolge von x_i nach y_j, wenn das Gadget nicht zu einem vorigen Zeitpunkt über Knoten $x_{i'}$ betreten wurde.*

Beweis. Eigenschaft 1 folgt aus Lemma 2.2.5, Eigenschaft 2 folgt aus Lemma 2.2.3 und aus Lemma 2.2.4. Eigenschaft 3 folgt ebenfalls aus Lemma 2.2.5 und der Tatsache, dass keine Kante in einem EdgeHop-Graphen mehrfach benutzt werden kann. □

Nachdem wir das Knoten-Gadget konstruiert haben, können wir einen EdgeHop-Graphen aus einem beliebigen Graphen G konstruieren. Also gilt:

Lemma 2.3.2 (Reduktion vom gerichteten Hamiltonkreisproblem)**.** *Sei $G = (V, E)$ ein beliebiger zusammenhängender gerichteter Graph. Es gibt eine in Polynomzeit berechenbare Funktion f, so dass gilt:*

$$\begin{gathered} G \text{ besitzt einen Hamiltonkreis.} \\ \Leftrightarrow \\ \text{Es gibt eine gültige Zugfolge von } a \text{ nach } z \text{ in } f(G). \end{gathered}$$

Beweis. Sei G ein beliebiger zusammenhängender gerichteter Graph. Wir definieren $f(G) := (G', k, a, z, P)$. Hierbei ist $V(G')$ die Menge aller Knoten, die durch die einzelnen Knoten-Gadgets $\hat{K}^v$ induziert wird, vereinigt mit dem Start-, dem Ziel- und einem *Zwischen*knoten. Für die Ein- und Ausgangsknoten der Knoten-Gadgets gilt[15]:

$$\hat{K}^v_{x_i} \in V(\hat{K}^v) \Leftrightarrow i \in \{1, ..., d^-_G(v)\}$$
$$\hat{K}^v_{y_i} \in V(\hat{K}^v) \Leftrightarrow i \in \{1, ..., d^+_G(v)\}$$

Die Menge der Knoten ist also:

$$V(G') = \{a, z, b\} \cup \bigcup_{v \in V(G)} V(\hat{K}^v).$$

Die Kantenmenge $E(G')$ besteht aus den Kantenmengen, die die einzelnen Knoten-Gadgets induzieren (mit der oben angesprochenen Anzahl an Ein- und Ausgangsknoten), der Kante zwischen dem *out*-Knoten des letzten Knoten-Gadgets und dem Zielknoten, aus der Kante zwischen dem Start-Knoten und einem beliebigen Knoten-Gadget[16], aus der Kante zwischen dem *in*-Knoten des ersten Knoten-Gadgets und dem Zwischenknoten und aus den Kanten zwischen den *in* und *out*-Knoten der einzelnen Knoten-Gadgets. Zusätzlich gehören zur Menge $E(G')$ noch die

[15] An dieser Stelle können wir auch definieren, dass alle Knoten-Gadgets gleich viele Ein- und Ausgänge haben, in dem wir festlegen, dass jedes Knoten-Gadget genau so viele Ein- bzw. Ausgänge hat wie die größte Eingangs- bzw. Ausgangsvalenz ist, also $|\bigcup_{i \in \mathbb{N}} \hat{K}^v_{x_i}| = |\bigcup_{i \in \mathbb{N}} \hat{K}^v_{y_i}| = max(\{max_{v \in V(G)}(d^-_G(v)), max_{v \in V(G)}(d^+_G(v))\})$

[16] An dieser Stelle nutzen wir das Knoten-Gadget, das zu Knoten $1 \in V(G)$ korrespondiert.

Kanten zwischen einem Knoten x_i eines (zu Knoten v korrespondierenden) Knoten-Gadgets und einem Knoten y_j eines anderen (zu Knoten u korrespondierenden) Knoten-Gadgets, wenn u der i-te Nachfolger von v und v der j-te Vorgänger von u ist. Zuletzt gehören noch die Kanten zwischen dem Zwischenknoten und dem Knoten x_i eines (zu Knoten v korrespondierenden) Knoten-Gadgets zur Menge, wenn der Startknoten der i-te Nachfolger von v ist. Wir definieren $d := d_G^-(1)$. Formal gilt:

$$\begin{aligned}
E(G') = & \bigcup_{v \in V(G)} (E(\hat{K}^v)) \cup \{\hat{K}^n_{out} z\} \cup \{a\hat{K}^1_{x_{d+1}}\} \cup \{b\hat{K}^1_{in}\} \\
& \cup \bigcup_{i=1}^{n-1} \hat{K}^i_{in}\hat{K}^{i+1}_{out} \\
& \cup \{\hat{K}^v_{y_i}\hat{K}^u_{x_j} : ((v,u) \in E(G)) \\
& \quad \wedge (|\{u' : (v,u') \in E(G) \wedge u' \leq u\}| = i) \\
& \quad \wedge (|\{v' : (v',u) \in E(G) \wedge v' \leq v\}| = j)\} \\
& \cup \{\hat{K}^v_{y_i} b : ((v,1) \in E(G)) \\
& \quad \wedge (|\{v' : ((v',1) \in E(G)) \wedge (v' \leq v)\}| = i)\}
\end{aligned}$$

Damit ist G' konstruiert. Zusätzlich ist a der Startknoten und z der Zielknoten. Für die Funktion der maximalen Belegtheiten $k_{f(G)} = k : V(G') \to \mathbb{N}$ gilt:

$$k(v) := \begin{cases} 1, & v \in \{a, z, b\} \\ k_{\hat{K}^i}(v), & v \in V(\hat{K}^i) \end{cases}$$

Die neuen Knoten haben also eine maximale Belegtheit von 1 und alle anderen Knoten übernehmen die maximale Belegtheit, die sie in ihrem Gadget-Teilgraphen besitzen. Die Multimenge P ist ebenso die große Vereinigung aller Multimengen der einzelnen Knoten-Gadget-Teilgraphen, da auf a, z oder b keine passiven Steine liegen. Es gilt also: $P := \biguplus_{v \in V(G)} P_{\hat{K}^v}$.

Nehmen wir nun an, dass G hamiltonisch ist, dann gibt es eine Folge von paarweise verschiedenen Knoten $(v_1, ..., v_n) \in V(G)^n$ mit $(v_i, v_{i+1}) \in E(G)$ für $1 \leq i < n$ und $(v_n, v_1) \in E(G)$. Da die Folge einen Kreis beschreibt können wir o. B. d. A. annehmen, dass $v_1 = 1$. Aus unserer Konstruktion (und Lemma 2.2.4) folgt nun, dass es eine gültige Zugfolge von a nach $\hat{K}^1_x$ gibt. Ebenfalls gibt es aufgrund unserer Konstruktion (zusammen mit Lemma 2.2.4 und 2.2.3) gültige Zugfolgen von $\hat{K}^{v_i}_y$ nach $\hat{K}^{v_{i+1}}_x$ (für $1 \leq i < n$). Außerdem gibt es eine gültige Zugfolge von $\hat{K}^{v_n}_y$ nach b und eine Zugfolge von b nach $\hat{K}^1_{in}$. Aus Lemma 2.2.5 folgt, dass in einem Knoten-Gadget $\hat{K}^v$ gültige Zugfolgen von x nach y und von *in* nach *out* existieren. Zuletzt gibt es noch eine gültige Zugfolge von $\hat{K}^{v_n}_{out}$ nach z. Eine gültige Zugfolge von a nach z setzt sich nun zusammen aus

- einer gültigen Zugfolge von a nach $\hat{K}^1_x$,
- gültigen Zugfolgen von $\hat{K}^{v_i}_x$ nach $\hat{K}^{v_i}_y$ und dann nach $\hat{K}^{v_{i+1}}_x$ für $1 \leq i < n$,
- einer gültigen Zugfolge vom $\hat{K}^{v_n}_x$ nach $\hat{K}^{v_n}_y$,
- einer gültigen Zugfolge von $\hat{K}^{v_n}_y$ nach b und dann nach $\hat{K}^1_{in}$,

- gültigen Zugfolgen von $\hat{K}_{in}^{v_i}$ nach $\hat{K}_{out}^{v_i}$ und dann nach $\hat{K}_{in}^{v_{i+1}}$ für $1 \leq i < n$,
- einer gültigen Zugfolge von $\hat{K}_{in}^{v_n}$ nach $\hat{K}_{out}^{n}$ und dann nach z.

Nehmen wir nun an, in unserem Graphen gibt es eine gültige Zugfolge von a nach z. Aufgrund unserer Konstruktion und Lemma 2.2.5 muss diese gültige Zugfolge von b nach z ziehen. Dazu muss jedes Knoten-Gadget entsperrt werden. Das bedeutet, dass der aktive Stein in jedem Gadget mindestens einmal über Knoten x ziehen muss. Da jedes Gadget nur einmal benutzbar ist, wird jedes Knoten-Gadget *genau* einmal durchzogen, damit es eine gültige Zugfolge von b nach z gibt. Da es zwischen zwei verschiedenen Knoten-Gadgets genau dann eine gültige Zugfolge gibt, wenn diese im ursprünglichen Graphen G adjazent sind, induziert die Zugfolge einen Hamiltonkreis.

Die Funktion ist offensichtlich in Polynomzeit berechenbar. Auch wenn die Größe eines Knoten-Gadgets abhängig von dem zugehörigen Knoten (und somit nicht konstant) ist, ist diese beschränkt durch $max_{v \in V(G)}(d_G^-(v)) + max_{v \in V(G)}(d_G^+(v))$, also insbesondere auch linear beschränkt in der Anzahl aller Knoten des Graphen G. Wir müssen n Knoten-Gadgets der Größe $\mathcal{O}(n)$ konstruieren und haben zusätzlich noch konstanten Mehraufwand die restlichen Knoten (a, b und z) zum Graphen hinzuzufügen. Die gesamte Laufzeit liegt somit bei $\mathcal{O}(n^2)$.

Somit gilt unsere Behauptung. □

2.4 Bemerkungen

Wir haben mithilfe weniger Gadgets gezeigt, dass EdgeHop NP-vollständig ist. Um die NP-Schwere andere Probleme zu zeigen, können wir unsere Reduktion nutzen. Wir müssen nur die folgenden Gadgets als Teilinstanzen des zu betrachtenden Problems konstruieren:

- ein Gadget, dass einen „Weg“[17] von x nach y erlaubt (aber nicht umgekehrt)
- ein Gadget, dass einen Weg von x nach y_1 oder nach y_2 erlaubt (wegen der Dioden-Gadgets müssen wir nicht darauf achten, dass die anderen Wege nicht existieren)
- ein Gadget, dass einen Weg von x_1 nach y oder von x_2 nach y (auch hier müssen wir die anderen Wege nicht betrachten)
- ein Gadget, dass einen Weg von u_1 nach u_2 erlaubt, und einen Weg von x nach y wenn vorher von u_1 nach u_2 gezogen wurde (hier muss sichergestellt werden, dass es keinen Pfad von u_1 nach y und keinen Pfad von x nach u_2 gibt).
- ein Gadget für den Startknoten
- ein Gadget für den Zielknoten
- ein Gadget für die Kanten des Graphen
- ein Gadget für Kreuzungen

[17] *Weg* kann hier entweder tatsächlich einen Pfad meinen (wenn man ein Bewegungsproblem betrachtet), oder es kann auch einfach nur eine Folge von Aktionen und Entscheidungen beschrieben.

Im Beweis zu Satz 2.2.9 haben wir eine eingeschränkte Variante unseres definierten Problems betrachtet (eine Normalform sozusagen). Wir haben nur Knoten mit einer maximalen Belegtheit von 1 oder von 2 verwendet – also Knoten, auf denen man *keinen* passiven Zug ausführen kann und Knoten, auf denen man *höchstens einen* passiven Zug ausführen kann. Dementsprechend haben wir auch keinen Knoten verwendet, auf dem (zu Beginn) mehr als 2 passive Steine liegen. Ebenfalls haben wir keinen Knoten mit einer größeren Valenz als 4 verwendet.

Kapitel 3

Beispiele

3.1 Latrunculi

Abbildung 3.1: Zwei Römer spielen Latrunculi.
Quelle: `http://eventi.langhe.net/event/alba-ce-un-latrunculus-in-museo/`, letzter Zugriff: 02. Januar 2015

In diesem Abschnitt werden wir die 2014 gezeigten Ergebnisse [A13] verstärken, in dem wir mithilfe von Edge Hop zeigen, dass *1-Zug-Latrunculi* NP-vollständig ist. Zusätzlich ist die Reduktion von EdgeHop ausgehend (also die Konstruktion kleiner, simpler Gadgets) weniger aufwändig, als die Reduktion im bisherigen Beweis, in dem große Gadgets als Latrunculi-Spielsituationen konstruiert wurden.

3.1.1 Über Latrunculi

Latrunculi (auch Ludus Latrunculorum) ist ein aus der Römerzeit bekanntes Zweispieler-Brettspiel, welches allerdings ältere (ägyptische) Wurzeln hat. [A9] Das Spiel war zur damaligen Zeit beliebt – der Senator Cnaeus Calpurnicus Piso z. B. lockte aufgrund seines spielerischen Könnens viele Zuschauer an. [A24] Bis heute wurden allerdings noch keine explizit aufgezeichneten, vollständigen Regelwerke

gefunden, weswegen verschiedene Historiker verschiedene Auslegungen der Regeln vorschlagen. Ein Unterschied liegt z. B. in der Anzahl verschiedener Spielfiguren. Während einige Historiker vorschlagen, dass jeder Spieler nur eine einzelne Art von Spielsteinen (genannt *latrones*[1]) hat (ähnlich wie bei *Dame*), schlagen andere vor, dass jeder Spieler neben den *latrones* noch einen weiteren Stein (genannt *dux*[2]) hat, den es zu erobern bzw. beschützen gilt. Ein anderer Unterschied liegt in der Startaufstellung: Hier gibt es zum einen die Annahme, dass die Steine zu Beginn des Spiels abwechselnd von beiden Spielern beliebig auf dem Spielbrett verteilt wurden und zum anderen die Annahme, dass die Steine zu Spielstart eine feste Position hatten. Wir werden uns im Folgenden an die Regeln halten, wie sie bei Masters Traditional Games [A12] oder Katharina Uebel und Peter Buri [A21] zu finden sind, andere Regelauslegungen findet man aber z. B. bei Bell [A3], Schädler [A24] oder Kowalski [A17]. Für die Betrachtung der Komplexität machen die verschiedenen Regelauslegungen keinen Unterschied, wie wir später sehen werden. Die Regeln, die wir betrachten werden im Folgenden erklärt.

Abbildung 3.2: Ein Spielbrett, auf dem Latrunculi gespielt werden kann.
Quelle: `http://english.lasindias.com/was-go-played-in-ancient-europe`, letzter Zugriff: 02. Januar 2015

- Das Spiel wird auf einem rechteckigen Spielbrett (siehe Abbildung 3.2) gespielt, welches in $n \times m$ Felder unterteilt ist. [A24] Typische Spielfeldgrößen waren $n = m = 8$ und $n = 8$, $m = 12$. Diese Wahlmöglichkeit bei der Spielfeldgröße bietet einen idealen Ansatz zur Verallgemeinerung des Spiels.

- Zu Beginn des Spiels besetzt jeder Spieler jedes Feld der beiden, ihm nächsten Reihen mit seinen Steinen.

- Die Spieler ziehen abwechselnd je einen ihrer Steine, bis ein Spieler keine Steine mehr hat, oder er keine gegnerischen Steine mehr erobern kann.

[1] *Latrones* bedeutet *Soldaten* oder *Söldner*.
[2] *Dux* bedeutet *Anführer* oder *Imperator*.

- Ein Stein kann so weit horizontal oder vertikal ziehen, wie sich freie Felder in der entsprechenden Richtung befinden.[3]
- Wenn in einem Zug ein Stein des Gegenspielers erobert wird, dann darf der erobernde Spieler mit dem zuletzt gezogenen Stein erneut ziehen. Das einmalige Bewegen nennen wir *Teilzug* und ein *Zug* besteht damit aus aufeinanderfolgenden Teilzügen eines Spielers.
- Ein Stein des Gegenspielers wird *erobert*, wenn ein eigener Stein so gezogen wird, dass der Stein des Gegenspielers auf zwei gegenüberliegenden Seiten von eigenen Steinen umstellt ist.
- Erobern ist nur aktiv möglich – man darf zwischen zwei Steine des Gegenspielers ziehen, ohne dass der eigene Stein automatisch erobert wird.

Zur Verdeutlichung dieser Regeln siehe Abschnitt A.2 im Anhang A.

3.1.2 Komplexität

Da im Gegensatz zum Schach und vielen anderen Spielen ein Zug in Latrunculi aus vielen Teilzügen bestehen kann, ist es klar, dass man mit einem Zug viel mehr erreichen kann als in anderen Spielen. Darum werden wir uns im Folgenden überlegen, ob es in Latrunculi möglich ist, in einem Zug einen bestimmten Stein des Gegenspielers zu erobern.[4] Wir definieren also:

Definition 3.1.1 (1-Zug-Latrunculi)**.** Seien $n, m \in \mathbb{N}$ Seitenlängen des Spielfelds, $B, O \subseteq \{1, ..., n\} \times \{1, ..., m\}$ Positionen der blauen bzw. orangefarbenen Steine (mit $B \cap O = \emptyset$), $(a_1, a_2) \in B$ Position des *aktiven Steins* und $(z_1, z_2) \in O$ Position des *Zielsteins*. Gibt es eine Teilzugfolge, die den *Zielstein* erobert und nur mit dem *aktiven Stein* zieht?

Wir nennen dieses Problem *1-Zug-Latrunculi*.

Zuerst werden wir uns überlegen, dass *1-Zug-Latrunculi* in NP liegt und danach werden wir mithilfe von EdgeHop die NP-Schwere zeigen. Hierzu werden wir die Kanten des EdgeHop-Graphen, den Start- und Zielknoten und die vier Gadgets (Abbildungen 2.3, 2.4, 2.5 und 2.6) aus Latrunculi-Spielsituationen modellieren.

Lemma 3.1.1. *1-Zug-Latrunculi* $\in NP$.

Beweis. Da auf einen blauen Teilzug A nur ein weiterer Teilzug folgen kann, wenn Teilzug A damit endet, dass ein orangefarbener Stein erobert (und somit aus dem Spiel entfernt) wird ist es klar, dass ein blauer Zug nicht aus mehr als o Teilzügen bestehen kann. Es ist also möglich, eine Folge von Teilzügen zu raten und in Polynomzeit zu überprüfen, ob dieser Zug regelkonform ist und ob er dazu führt, dass der Zielstein erobert wird. (Siehe hierzu auch [A13].) □

Im folgenden werden wir für Abbildungen die Latrunculi-Spielsituationen darstellen nur die Felder einzeichnen, auf denen ein Stein liegt oder über die der aktive Stein zieht. Weiter gilt die übliche Konvention, dass Felder von links nach rechts mit

[3] Wie der Turm im Schach.

[4] In Spielen wie z. B. Schach wäre diese Frage trivial.

aufeinander folgenden Kleinbuchstaben und von oben nach unten mit natürlichen Zahlen ab 1 bezeichnet werden. Außerdem werden wir neben blauen und orangefarbenen Kreisen noch orangefarbene Kreise mit schwarzem Rand nutzen. Diese stehen für Spielsteine, welche beim Durchqueren des Gadgets nicht erobert werden (oder beim Erobern beteiligt sind) – diese Steine agieren als Blockaden, um zu verhindern, dass der *aktive Stein* in eine ungewollte Richtung zieht. Spieltechnisch sind diese Steine reguläre orangefarbene Steine, die spezielle Darstellung gilt nur der Übersichtlichkeit.

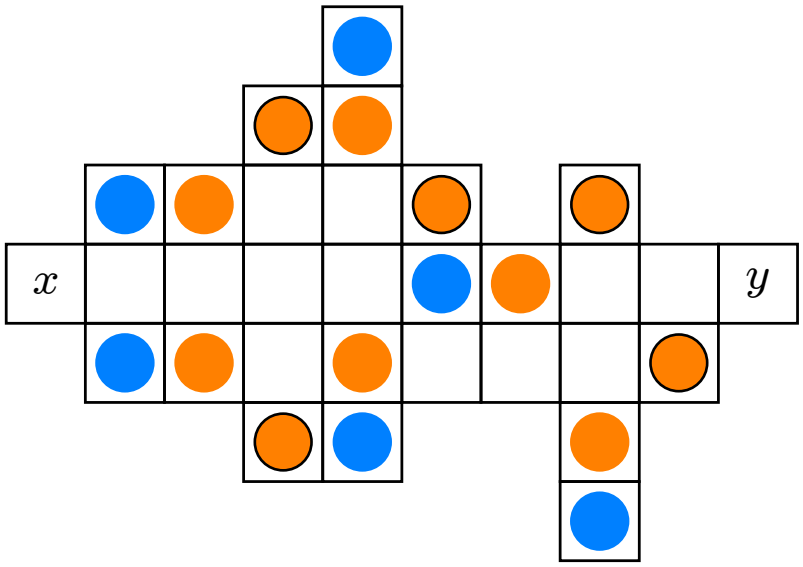


Abbildung 3.3: Ein Dioden-Gadget konstruiert als Latrunculi-Spielsituation. Der aktive Stein betritt das Gadget über Feld x und verlässt es über Feld y.

Lemma 3.1.2 (Dioden-Gadget (Latrunculi))**.** *Dem aktiven Stein ist es im Dioden-Gadget (Abbildung 3.3) möglich von Feld $x = a4$ zu Feld $y = j4$ zu ziehen aber nicht umgekehrt. Weiter kann der aktive Stein das Gadget über kein anderes Feld als y (und trivialerweise x) verlassen.*

Beweis. Angenommen der aktive Stein startet auf $a4$. Eine Folge von Teilzügen nach $j4$ ist: $e4$ (erobert $e3$), $e5$ (erobert $e6$), $d5$ (erobert $c5$), $d3$ (erobert $c3$), $h3$ (erobert $h2$), $h4$ (erobert $g4$), $j4$. (Siehe Abschnitt A.3 im Anhang A.)

Falls der aktive Stein auf $j4$ startet muss er, um nach $a4$ zu ziehen die folgenden Teilzüge ausführen: $h4$ (erobert $g4$), $h3$ (erobert $h2$). Von $h3$ aus müsste er nun weiter nach links ziehen, über $e3$. Dies ist allerdings nicht möglich, da dort ein orangefarbener Stein liegt. Der aktive Stein kann also im Ursprungszustand nicht von $j4$ nach $a4$ ziehen. Für den Fall, dass der aktive Stein das Gadget zu einem vorigen Zeitpunkt von $a4$ nach $j4$ durchzogen hat und somit den Stein auf $e4$ erobert hat gilt allerdings, dass es keine gültige Teilzugfolge nach $h3$ gibt, da der orangefarbene Stein auf $j4$ schon erobert wurde.

Zuletzt ist es ersichtlich, dass der aktive Stein von keinem der Felder $e5$, $d5$, $d3$ oder $h3$ eine Möglichkeit hat, das Gadget zu verlassen. Von Feld $e4$ kann er es über $a4$ verlassen und von $h4$ kann er es über $j4$ verlassen. Falls ein Teilzug auf einem anderen Feld endet, dann endet auf diesem Feld auch der Zug (da auf keinem anderen Feld ein orangefarbener Stein erobert werden kann). Somit kann der aktive Stein das Gadget nur über $a4$ und $j4$ verlassen. □

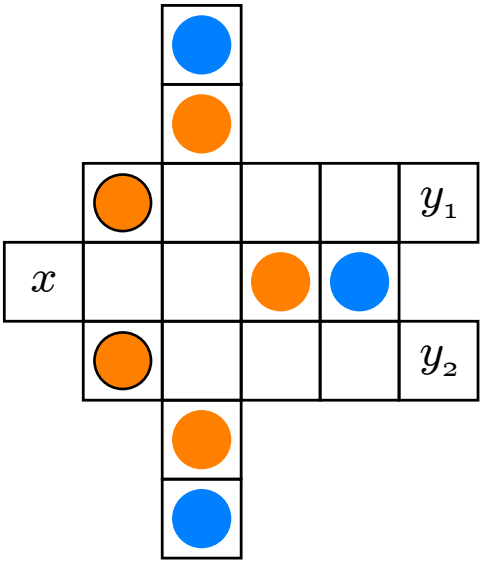


Abbildung 3.4: Ein Verzweigungs-Gadget konstruiert als Latrunculi-Spielsituation. Der aktive Stein betritt das Gadget über Knoten x und verlässt es über Feld y_1 oder über Feld $y2$.

Lemma 3.1.3 (Verzweigungs-Gadget (Latrunculi))**.** *Der aktive Stein kann im Verzweigungs-Gadget (Abbildung 3.4) von Feld $x = a4$ nach Feld $y_1 = f5$ oder nach Feld $y_2 = f3$ ziehen. Weiter kann der aktive Stein das Gadget über kein anderes Feld außer den drei genannten Feldern verlassen.*

Beweis. Angenommen der aktive Stein startet auf $a4$. Eine Folge von Teilzügen nach $f5$ ist: $c4$ (erobert $d4$), $c5$ (erobert $c6$), $f5$. Eine Folge von Teilzügen nach $f3$ ist: $c4$ (erobert $d4$), $c3$ (erobert $c2$), $f3$.

Es gilt, dass der Zug endet, wenn der aktive Stein auf ein anderes Feld als $c4$, $c5$ oder $c3$ zieht. Und von $c4$ kann der aktive Stein das Gadget nur über $a4$, von $c5$ nur über $f5$ und von $c3$ nur über $f3$ verlassen. □

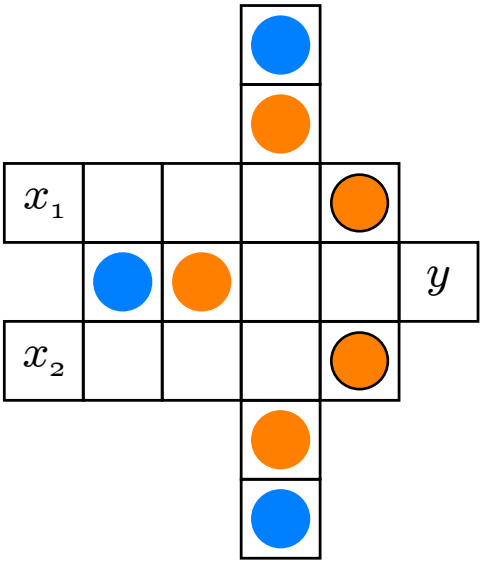


Abbildung 3.5: Ein Zusammenführungs-Gadget konstruiert als Latrunculi-Spielsituation. Der aktive Stein verlässt das Gadget über Feld y, egal ob er es über x_1 oder x_2 betritt.

Lemma 3.1.4 (Zusammenführungs-Gadget (Latrunculi)). *Der aktive Stein kann im Zusammenführungs-Gadget (Abbildung 3.5) von Feld $x_1 = a5$ nach Feld $y = f4$ oder von Feld $x_2 = a3$ nach Feld y ziehen. Weiter kann der aktive Stein das Gadget über kein anderes Feld außer den drei genannten Feldern verlassen.*

Beweis. Angenommen der aktive Stein startet auf $a5$. Eine Folge von Teilzügen nach $f4$ ist: $d5$ (erobert $d6$), $d4$ (erobert $c4$), $f4$. Angenommen der aktive Stein startet auf $a3$. Eine Folge von Teilzügen nach $f4$ ist: $d3$ (erobert $d2$), $d4$ (erobert $c4$), $f4$.

Wieder gilt, dass der Zug endet, wenn der aktive Stein auf ein anderes Feld als $d5$, $d4$ oder $d3$ zieht. Und von $d5$ kann der aktive Stein das Gadget nur über $a5$, von $d4$ nur über $f4$ und von $d3$ nur über $a3$ verlassen. □

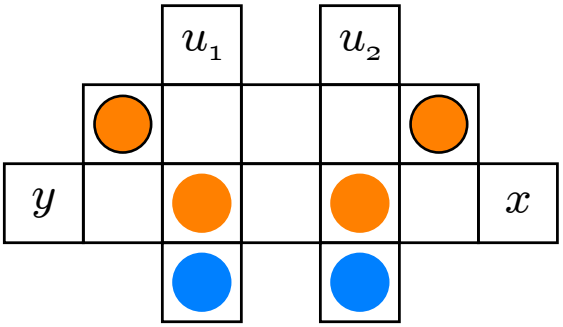


Abbildung 3.6: Ein Entsperr-Gadget konstruiert als Latrunculi-Spielsituation. Der aktive Stein kann nur dann von Feld x nach Feld y ziehen, wenn er zuvor von Feld u_1 nach u_2 gezogen ist.

Lemma 3.1.5 (Entsperr-Gadget (Latrunculi)). *Der aktive Stein kann im Entsperr-Gadget (Abbildung 3.6) von Feld u_1 nach Feld u_2 ziehen. Weiter kann der aktive Stein nur dann von Feld x nach Feld y ziehen, wenn er zuvor von Feld u_1 nach Feld u_2 gezogen ist. Zusätzlich darf der aktive Stein weder von u_1 nach y noch von x nach u_2 ziehen. Außerdem kann der Stein das Gadget über kein anderes Feld als x, y, u_1 oder u_2 verlassen.*

Beweis. Angenommen der aktive Stein steht auf $c4$. Eine Teilzugfolge nach $e4$ ist: $c3$ (erobert $c2$), $e3$ (erobert $e2$), $e4$.

Angenommen der aktive Stein steht auf $g2$. Um $a2$ zu erreichen kann der aktive Stein nur direkt nach $a2$ ziehen. Dazu muss er allerdings über $c2$ und $e2$ ziehen. Dies ist nur möglich, wenn die auf diesen Feldern liegenden, orangefarbenen Steine zuvor erobert wurden. Damit diese erobert werden muss der aktive Stein von $c4$ nach $e4$ ziehen.

Angenommen der aktive Stein steht auf $c4$. Um nach $a2$ zu ziehen muss der aktive Stein die Teilzugfolge $c3$ (erobert $c2$), $c2$ ausführen. Auf $c2$ endet allerdings der Zug, da kein weiterer Stein erobert wird. Also kann der aktive Stein nicht $a2$ erreichen.

Angenommen der aktive Stein steht auf $g2$. Um $e4$ zu erreichen muss der aktive Stein auf $e2$ ziehen. Falls dort ein orangefarbener Stein liegt, kann der aktive Stein nicht dort hin ziehen. Falls dort kein orangefarbener Stein liegt, kann der aktive Stein zwar nach $e2$ ziehen, aber dann endet der Zug. Somit kann der aktive Stein auch dann nicht $e4$ erreichen.

Die einzigen Felder innerhalb des Gadgets, bei denen der Zug nicht zwangsweise endet sind $c3$ und $e3$. Von dort kann der aktive Stein das Gadget nur über $c4$ bzw. $e4$ verlassen. Damit gilt das Lemma. □

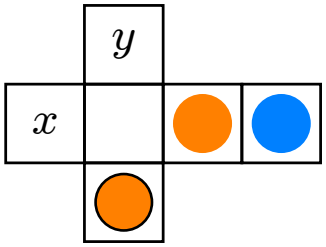
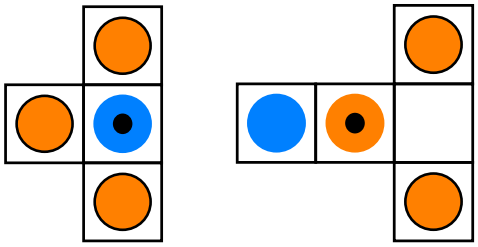


Abbildung 3.7: Ein Knick-Gadget konstruiert als Latrunculi-Spielsituation. Dieses wird benötigt, um Kanten zu simulieren, die nicht vollständig horizontal oder vertikal verlaufen.

Lemma 3.1.6 (Knick-Gadget (Latrunculi))**.** *Das Knick-Gadget (Abbildung 3.7) ermöglicht dem aktiven Stein seine Bewegungsrichtung von* horizontal *zu* vertikal *(oder umgekehrt) zu ändern, in dem er von $x = a2$ nach $y = b3$ (oder umgekehrt) zieht.*

Beweis. Angenommen der aktive Stein startet auf $a2$. Die Teilzugfolge $b2$ (erobert $c2$), $b3$ bringt ihn nach $b3$, wobei der erste Teilzug in *horizontaler* und der zweite Teilzug in *vertikaler* Richtung ausgeführt wurde.

Wenn der aktive Stein auf $b3$ startet, bringt ihn die Teilzugfolge $b2$ (erobert $c2$), $a2$ nach $a2$. Hierbei ist der erste Teilzug in *vertikaler* und der zweite Teilzug in *horizontaler* Bewegungsrichtung ausgeführt wurden. □



(a) Start. **(b)** Ziel.

Abbildung 3.8: Start- und Ziel-Gadget als Latrunculi-Spielsituationen. Der blaue Stein mit dem schwarzen Punkt ist der aktive Stein, der orangefarbene Stein mit dem schwarzen Punkt ist der Zielstein.

Lemma 3.1.7 (NP-Schwere)**.** *1-Zug-Latrunculi ist NP-schwer.*

Beweis. Satz 2.2.9 hat uns gezeigt, dass EdgeHop NP-vollständig ist. Mithilfe der gerade konstruierten Gadgets (Lemmata 3.1.2, 3.1.3, 3.1.4 und 3.1.5) können wir zu jedem Gadget des EdgeHop-Graphen eine passende Latrunculi-Spielsituation konstruieren.

Für Kanten, die komplett horizontal oder vertikal verlaufen brauchen wir keine speziell konstruierte Spielsituation, da der aktive Stein so weit in eine Richtung ziehen kann, wie freie Felder vor ihm sind. Wir müssen nur darauf achten, dass das Ausgangs-Feld auf der selben horizontalen bzw. vertikalen Position liegt, wie das Eingangs-Feld der beiden zu verbindenden Gadgets. Alle anderen Kanten müssen wir durch orthogonale Polygonzüge ersetzen, die wir dann mithilfe von Knick-Gadgets (Lemma 3.1.6) als Latrunculi-Spielsituation konstruieren. Da weiter gilt, dass ein ziehender Stein in Latrunculi nicht beliebig seine Bewegungsrichtung ändern kann, brauchen wir auch kein spezielles Crossover-Gadget.

Zuletzt müssen wir noch ein Start-Gadget und ein Ziel-Gadget als Spielsituation konstruieren – das Start-Gadget beinhaltet den aktiven Stein und sorgt dafür, dass er sich nur in eine Richtung bewegen kann und das Ziel-Gadget beinhaltet den Zielstein und sorgt dafür, dass dieser nur aus einer Richtung kommend erobert werden kann. Diese Gadgets sind in Abbildung 3.8 abgebildet. Es ist offensichtlich, dass die Gadgets die Forderungen erfüllen.

Damit ist 1-Zug-Latrunculi NP-schwer. □

Satz 3.1.8 (NP-Vollständigkeit)**.** *1-Zug-Latrunculi ist NP-vollständig.*

Beweis. Nach Lemma 3.1.7 is 1-Zug-Latrunculi NP-schwer und nach Lemma 3.1.1 ist 1-Zug-Latrunculi $\in NP$. Damit ist 1-Zug-Latrunculi nach Definition NP-vollständig. □

Wir haben nun gesehen, dass 1-Zug-Latrunculi NP-vollständig ist. Es ist auch klar, dass unsere Konstruktionen nicht von den spezifischen Regeln abhängen, die wir zu Anfang gewählt haben. Bei anderen Regelvorschlägen bleiben die Ergebnisse gleich. Wenn man die Steine zu Beginn des Spiels auf beliebige Felder legen darf, dann ist unsere Konstruktion auf eine „natürlichere“ Weise erreichbar, als wenn die Spieler das Spiel erst so lange spielen müssen, bis die Steine auf den passenden Positionen liegen. Aber das ändert nichts an der Komplexität unserer Frage, da wir nicht betrachten, wie es zu der Spielsituation kam. Wenn wir die Regelvariante betrachten, bei der jeder Spieler einen *dux*-Stein besitzt, ist 1-Zug-Latrunculi sogar gleichbedeutend zu der Frage, ob der blaue Spieler in einem Zug gewinnen kann, in dem man festlegt, dass der Zielstein der *dux*-Stein ist. Aber auch hier ändert sich nichts an der Komplexität. Einen gravierenden Unterschied macht allerdings die Festlegung, ob ein Zug aus beliebig vielen Teilzügen bestehen kann, oder ob ein Zug aus einem Teilzug besteht. Im zweiten Fall würde unsere Entscheidungsfrage uninteressant sein, da man sie ganz trivial beantworten kann, wenn man die höchstens erreichbaren $n+m$ Felder darauf überprüft, ob der Zielstein erobert wird, wenn der aktive Stein dort hin zieht.

Eine weitere Idee ist es, 1-Zug-Latrunculi darauf zu erweitern, dass man keinen aktiven Stein voraussetzt, sondern nur den Zielstein vorgibt. Allerdings kann man hier alle Gadgets so anpassen, dass keiner der inneren blauen Steine ein Gadget (in einem Zug) verlassen kann und somit wäre die Frage gleich zu 1-Zug-Latrunculi.

3.2 Minecraft

(a) Die Spielfigur baut mit einer Spitzhacke Golderz ab.

(b) Die Spielfigur bekämpft einen gegnerischen NPC, in diesem Fall einen *Creeper*.

Abbildung 3.9: Zwei Screenshots aus dem Spiel Minecraft (Version 1.8.1).

In diesem Kapitel werden wir zeigen, dass das populäre Computerspiel *Minecraft* in einer eingeschränkten Variante NP-schwer ist.[5] Minecraft soll in diesem Fall als Stellvertreter für alle Computerspiele stehen, in denen eine Spielfigur durch eine dreidimensionale Welt navigieren muss. Deswegen sind unsere Einschränkungen, obwohl sie bezogen auf Minecraft sehr stark sind, gerechtfertigt.

[5]Die NP-Zugehörigkeit werden wir nicht explizit zeigen, aber Argumente bzw. Beweisideen anbringen.

3.2.1 Über Minecraft

Minecraft ist ein 2011[6] erschienenes *Sandbox*-Computerspiel. Ursprünglich wurde das Spiel alleine von *Markus Persson* entwickelt, seit 2009 allerdings wird es von der schwedischen Firma *Mojang AB* weiterentwickelt. Minecraft ist ein sehr populäres Spiel – neben den Versionen für die drei „großen" PC-Betriebssystem-Familien (Linux, OS X, Windows) gibt es mittlerweile Versionen des Spiels für andere Geräte. Beispielsweise die *Pocket Edition* für Android, iOS und Windows Phone, oder Versionen für Playstation 3 (und Playstation 4) und Xbox 360 (und Xbox One). Die PC bzw. Mac-Version des Spiels hat sich aktuell über 18 Millionen mal verkauft. [B1] Ein weiterer Anhaltspunkt für die Popularität des Spiels, bzw. des gesamten Franchises ist, dass Microsoft im September 2014 Mojang AB für 2,5 Milliarden USD aufgekauft hat. [B4]

Wie üblich für Sandbox-Games hat Minecraft kein direktes Ziel. Der Spieler kann seiner Kreativität freien Lauf lassen, um die (prozedual generierte) Welt zu verändern. Je nach Spielmodus muss der Spieler auf verschiedene Dinge achten. Im sogenannten *Creative*-Modus ist der Spieler unsterblich, kann fliegen, hat Zugriff auf alle Blockarten bzw. Gegenstände. und kann Blöcke ohne Einschränkungen zerstören. Im *Survival*-Modus hat der Spieler einige Einschränkungen. Er ist nicht mehr unsterblich und kann nicht mehr fliegen. Ebenfalls muss der Spieler die Blöcke, die er zum Bauen nutzen möchte selbst sammeln. Je nach Art des Blocks braucht er hierfür spezielle Werkzeuge.[7] Weiter kann man sich in diesem Spielmodus entscheiden, ob man mit gegnerischen NPCs (also Monstern) spielen möchte, die unter bestimmten Bedingungen erscheinen und die Spielfigur angreifen. Die offizielle Webseite beschreibt das Spiel wie folgt: „Minecraft is a game about breaking and placing blocks. At first, people built structures to protect against nocturnal monsters, but as the game grew players worked together to create wonderful, imaginative things." [B1]

Eine spezielle Blockart in Minecraft ist *Redstone*-Erz. Wenn der Spieler diesen Block abbaut bekommt er *Redstone*-Staub, welcher als „elektrischer" Leiter funktioniert. Weiter kann man aus Redstone-Staub Redstone-Fackeln bauen, die wie ein Inverter arbeiten. Das bedeutet, dass es in Minecraft möglich ist grundlegende Logikgatter, also NICHT-, UND- und ODER-Gatter, und damit auch komplexere Schaltkreise zu bauen. (Siehe hierzu A.4 im Anhang A.) Dadurch ist Minecraft (bis auf technische Einschränkungen[8]) *Turing-vollständig*. Ein Beispiel für eine 8-bit CPU implementiert in Minecraft findet man bei [B6], ein Beispiel für einen, in Minecraft implementierten Brainfuck-Interpreter bei [B12] (auch wenn diese Konstruktion mehr als nur Redstone nutzt).

Mit Version *1.3.1* im Jahr 2012 wurde dem Spiel ein weiterer Spielmodus hinzugefügt, der sogenannte *Adventure*-Modus. [B7] Die Spielweise in diesem Modus entspricht der eines typischen *Action-Adventures*. Hierbei navigiert der Spieler durch vorher gebaute Welten und kann nur Blöcke zerstören oder setzen, die der Autor der Welt vorher als zerstörbar oder setzbar markiert hat. Wir werden uns im folgen-

[6] Spielbare Pre-Alpha-Versionen des Spiels wurden schon 2009 veröffentlicht. [B11]

[7] Stein kann der Spieler bspw. mit einer beliebigen Spitzhacke abbauen, Eisenerz kann der Spieler nur mit einer aus Stein gefertigten Spitzhacke abbauen und Diamanterz kann der Spieler nur mit einer aus Eisen gefertigten Spitzhacke abbauen.

[8] Die maximale Größe einer Welt entspricht einem Quader mit einer Grundfläche von 60 Millionen × 60 Millionen Blöcken und einer Höhe von 256 Blöcken.

den mit diesem Spielmodus beschäftigen. Allerdings werden wir weiter nur solche Welten betrachten, in denen der Autor keinen Redstone-Staub (und andere damit herstellbare Blöcke) verwendet hat. Da Minecraft ein Spiel ohne fest definiertes Ziel ist, ist es Sache des Autors einer Welt, ein Ziel zu definieren. Dies kann z. B. das Erreichen eines bestimmten Blocks, das Finden eines bestimmten Gegenstandes oder das Besiegen eines bestimmten Gegners sein. Wir werden uns darauf einigen, dass wir solche Welten betrachten, in denen das vorgegebene Ziel ist, eine mit Diamanten gefüllte Kiste zu finden. Damit ist unser betrachtetes Problem ein Stellvertreter für andere Action-Adventures oder allgemein Spiele, in der eine Spielfigur sich in einer dreidimensionalen Welt bewegt und versucht einen bestimmten Ort zu erreichen.

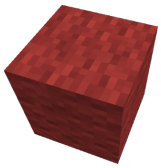
3.2.2 Komplexität

Definition 3.2.1 (Adventure-Modus-Minecraft (ohne Redstone))**.** Ist es dem Spieler in einer beliebigen vorgefertigten Minecraft-Welt ohne Redstone möglich, eine Kiste mit Diamanten zu finden, wenn sich die Spielfigur im Adventure-Modus befindet?

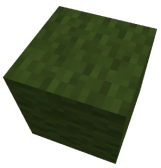
Wir nennen dieses Problem *Adventure-Modus-Minecraft (ohne Redstone)* und kürzen es ab mit *AMC-R*.

(a) „Oak Wood“-Block.

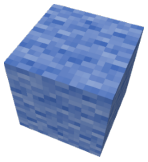
(b) „Polished Andesite“-Block.



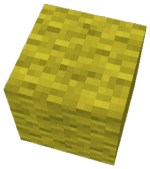
(c) „Red Wool“-Block.



(d) „Green Wool“-Block.



(e) „Light Blue Wool“-Block.



(f) „Yellow Wool“-Block.

(g) „Iron Door“-Block.

(h) „Lever“-Block.

Abbildung 3.10: Abbildungen der einzelnen Blöcke, die wir in unseren Gadgets verwenden.

In den folgenden Abbildungen werden Wände und Decken durch *„Polished Andesite“*-Blöcke und vom Spieler begehbare Flächen durch *„Oak Wood“*-Blöcke dargestellt. *„Green Wool“*-Blöcke markieren die Eingänge der Gadgets und *„Red Wool“*-Blöcke die Ausgänge. Zusätzlich markieren *„Light Blue Wool“*-Blöcke und *„Yellow*

Wool“-Blöcke die entsperrenden Ein- bzw. Ausgänge des Unlock-Gadgets. „*Lever*“-Blöcke und „*Iron Door*“-Blöcke nutzen wir als Entsperr-Mechanik, da diese typisch für Action-Adventures ist. „*Iron Door*“-Blöcke sind im ursprünglichen Zustand geschlossen und können nur durch *aktivieren* eines benachbarten Blockes geöffnet werden. Ein Block wird aktiviert, wenn ein an ihm platzierter „*Lever*“-Block betätigt wird.[9] Siehe für die Blöcke Abbildung 3.10. Weiter werden wir für die folgenden Konstruktionen bzw. Beweise verwenden, dass die Spielfigur nur 1 Block hoch und 4 Blöcke weit springen kann und dass sie erst dann Fallschaden nimmt, wenn die Falltiefe 5 Blöcke oder größer ist.

Um die Bewegungsabfolge der Spielfigur zu notieren, geben wir die Koordinaten $(x, y, z) \in \mathbb{R}^3$ der Blöcke an, die die Spielfigur betritt. Zur Vereinfachung nehmen wir an, dass die Spielfigur nicht auf mehreren Blöcken gleichzeitig stehen kann. Eine Bewegungsabfolge $(x_0, y_0, z_0), ..., (x_n, y_n, z_n)$ ist nur dann erlaubt, wenn für alle $0 \leq i < n$ gilt:

- Die Spielfigur steht zu Beginn auf (x_0, y_0, z_0)
- $\sqrt{(x_i - x_{i+1})^2 + (z_i - z_{i+1})^2} \leq 5$
- $y_i - 4 \leq y_{i+1} \leq y_i + 1$
- Für alle $0 \leq j \leq n$ gilt: Es gibt keinen Block mit Koordinaten $(x_j, y_j + 1, z_j)$ oder $(x_j, y_j + 2, z_j)$.

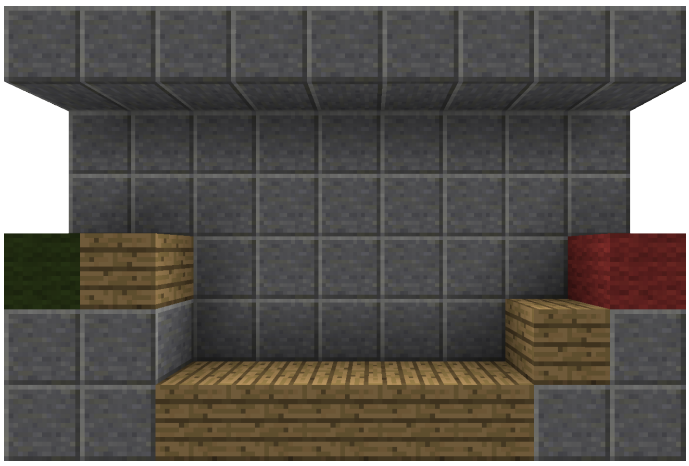


Abbildung 3.11: Ein Dioden-Gadget gebaut aus Minecraft-Blöcken (Seitenansicht). Die vordere Wand fehlt, damit das Innenleben des Gadgets sichtbar ist.

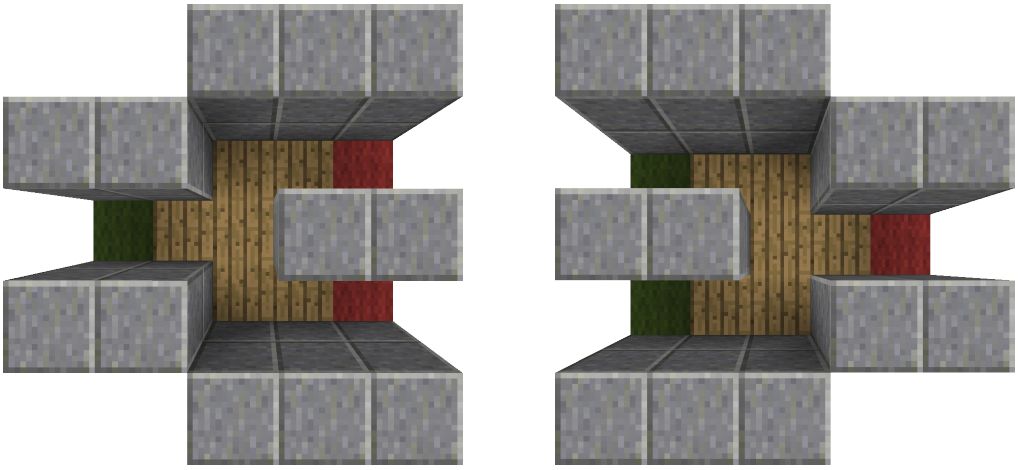
Lemma 3.2.1 (Dioden-Gadget (Minecraft))**.** *Die Spielfigur kann sich im Dioden-Gadget (Abbildung 3.11) vom grünen Block zum roten Block aber nicht vom roten Block zum gründen Block bewegen.*

Beweis. Wir nehmen o. B. d. A. an, dass der grüne Block die Koordinaten $(0, 0, 0)$ hat und der rote Block die Koordinaten $(8, 0, 0)$ hat. Dann ist $(0, 0, 0)$, $(1, 0, 0)$, $(2, -2, 0)$, $(3, -2, 0)$, $(4, -2, 0)$, $(5, -2, 0)$, $(6, -2, 0)$, $(7, -1, 0)$, $(8, 0, 0)$ eine erlaubte Bewegungsabfolge.

[9] Der „*Lever*“-Block selbst wird auch aktiviert, wenn er betätigt wird.

Da $0 \not\leq -1$ kann sich die Spielfigur nicht von $(x, -2, 0)$ nach $(1, 0, 0)$ bewegen (mit $x \in \{2, ..., 6\}$). Weiter kann sich die Spielfigur nicht von $(7, -1, 0)$ nach $(1, 0, 0)$ bewegen, da $\sqrt{(7-1)^2 + (0-0)^2} = 6 \not\leq 5$. Außerdem kann die Spielfigur sich auch nicht von $(8, 0, 0)$ nach $(1, 0, 0)$ bewegen, da $\sqrt{(8-1)^2 + (0-0)^2} = 7 \not\leq 5$. Es ist leicht zu sehen, dass die Spielfigur $(0, 0, 0)$ nicht erreichen kann, wenn sie schon nicht $(1, 0, 0)$ erreichen kann.

Damit gilt das Lemma. □



(a) Ein Verzweigungs-Gadget. **(b)** Ein Zusammenführ-Gadget.

Abbildung 3.12: Verzweigungs- und Zusammenführ-Gadget gebaut aus Minecraft-Blöcken (Draufsicht). Die Decke fehlt, damit das Innenleben des Gadgets sichtbar ist.

Lemma 3.2.2 (Verzweigungs-Gadget (Minecraft))**.** *Die Spielfigur kann sich im Verzweigungs-Gadget (Abbildung 3.12a) vom grünen Block zu einem beliebigen der beiden roten Blöcke bewegen.*

Beweis. Wir nehmen o. B. d. A. an, dass der grüne Block die Koordinaten $(0, 0, 0)$ hat und die roten Blöcke die Koordinaten $(4, 0, -1)$ und $(4, 0, 1)$ haben. Dann sind $(0, 0, 0)$, $(1, 0, 0)$, $(2, 0, 0)$, $(2, 0, 1)$, $(3, 0, 1)$, $(4, 0, 1)$ und $(0, 0, 0)$, $(1, 0, 0)$, $(2, 0, 0)$, $(2, 0, -1)$, $(3, 0, -1)$, $(4, 0, -1)$ erlaubte Bewegungsabfolgen. □

Lemma 3.2.3 (Zusammenführ-Gadget (Minecraft))**.** *Die Spielfigur kann sich im Zusammenführ-Gadget (Abbildung 3.12b) von jedem der beiden grünen Blöcke zum roten Block bewegen.*

Beweis. Wir nehmen o. B. d. A. an, dass die grünen Blöcke die Koordinaten $(0, 0, -1)$ und $(0, 0, 1)$ haben und der rote Block die Koordinaten $(4, 0, 0)$ hat. Dann sind $(0, 0, -1)$, $(1, 0, -1)$, $(2, 0, -1)$, $(2, 0, 0)$, $(3, 0, 0)$, $(4, 0, 0)$ und $(0, 0, 1)$, $(1, 0, 1)$, $(2, 0, 1)$, $(2, 0, 0)$, $(3, 0, 0)$, $(4, 0, 0)$ erlaubte Bewegungsabfolgen. □

Lemma 3.2.4 (Entsperr-Gadget (Minecraft))**.** *Im Entsperr-Gadget kann sich die Spielfigur vom hellblauen Block zum gelben Block bewegen. Ebenfalls gibt es nur einen Weg vom grünen Block zum roten Block, wenn die Spielfigur vorher vom hellblauen Block zum gelben Block gegangen ist. Weiter kann sich die Spielfigur nicht vom hellblauen Block zum roten Block und nicht vom grünen Block zum gelben Block bewegen.*

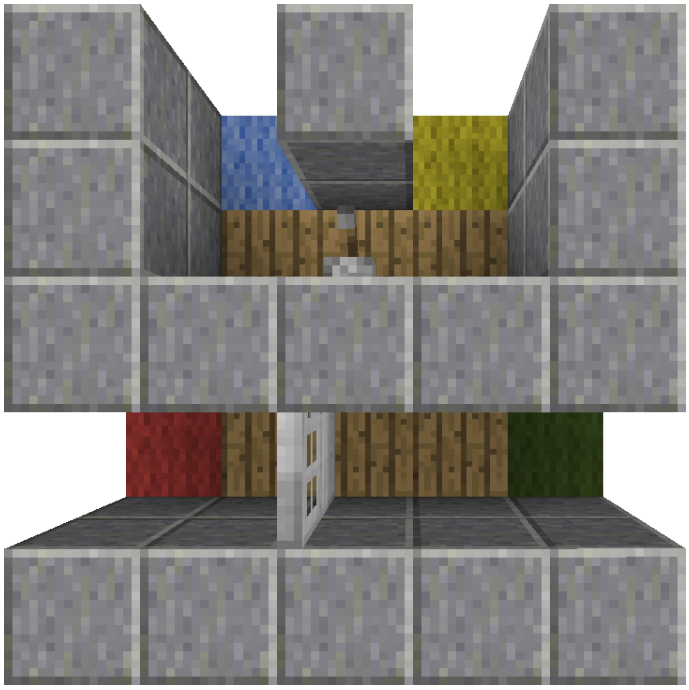


Abbildung 3.13: Ein Entsperr-Gadget gebaut aus Minecraft-Blöcken (Draufsicht). Die Decke fehlt, damit das Innenleben des Gadgets sichtbar ist.

Beweis. Wir nehmen o. B. d. A. an, dass der grüne Block die Koordinaten $(4, 0, 0)$, der rote Block die Koordinaten $(0, 0, 0)$, der hellblaue Block die Koordinaten $(1, 0, 3)$ und der gelbe Block die Koordinaten $(3, 0, 3)$ hat. Dann ist $(1, 0, 3)$, $(1, 0, 2)$, $(2, 0, 2)$, $(3, 0, 2)$, $(3, 0, 3)$ eine erlaubte Bewegungsabfolge vom hellblauen zum gelben Block. Weiter ist $(4, 0, 0)$, $(3, 0, 0)$, $(2, 0, 0)$, $(1, 0, 0)$, $(0, 0, 0)$ eine erlaubte Bewegungsabfolge vom grünen zum roten Block. Diese ist allerdings nur dann möglich, wenn der *„Iron Door“*-Block geöffnet wurde, was bedeutet, dass die Spielfigur zu einem vorigen Zeitpunkt den *„Lever“*-Block aktiviert haben muss, also vom hellblauen zum gelben Block gelaufen sein muss.

Es ist ebenfalls leicht zu sehen, dass es keine Wege vom hellblauem zum roten Block oder vom grünen zum gelben Block gibt, da die beiden Gänge durch die Wand in der Mitte voneinander getrennt sind. □

Damit haben wir die größeren EdgeHop-Gadgets aus Minecraft-Blöcken nach gebaut. Im folgenden müssen wir nun noch das Start- und Ziel-Gadget und die Kanten (inklusive Kreuzungen) modellieren und können somit zeigen, dass *AMC-R* NP-schwer ist.

Lemma 3.2.5 (NP-Schwere)**.** *Adventure-Modus-Minecraft (ohne Redstone) ist NP-schwer.*

Beweis. Nach Satz 2.2.9 ist EdgeHop NP-vollständig. Mithilfe der gerade konstruierten Gadgets (Lemmata 3.2.1, 3.2.2, 3.2.3 und 3.2.4) können wir zu jedem Gadget des EdgeHop-Graphen ein passendes Gadget aus Minecraft-Blöcken konstruieren.

Für das Start-Gadget müssen wir einen Raum konstruieren, bei dem an drei von vier Seiten eine Wand ist und nur eine Seite frei ist. Damit kann das Gadget nur in eine Richtung verlassen werden. Wir nehmen an, dass sich die Position, auf der die Spielfigur startet (der sogenannte „world spawn“) innerhalb dieses Raumes

befindet.[10] Das Ziel-Gadget wird ähnlich konstruiert – ein Raum, bei dem nur auf eine Seite keine Wand ist. Zusätzlich müssen wir in diesen Raum noch die Kiste mit den Diamanten stellen.

Wenn wir (wie bei 1-Zug-Latrunculi auch) orthogonale Graphen voraussetzen, müssen wir nur gerade Kanten und 90-Grad-Knicke modellieren. Gerade Kanten werden als Gänge konstruiert. Diese können beliebig lang sein, da die Spielfigur beliebig weit laufen kann. Knicke können wir als Räume modellieren, bei denen auf zwei nicht-gegenüberliegenden Seiten keine Wände sind. Da EdgeHop-Graphen nicht planar sind müssen wir noch Kreuzungen modellieren. Wir können annehmen, dass sich zwei Kanten immer nur in genau einem Punkt schneiden[11]. Das bedeutet, dass bei jedem Schnittpunkt eine Kante horizontal und eine vertikal verläuft. Dann können wir die Kreuzungen so modellieren, dass die horizontal verlaufende Kante *über* die vertikal verlaufende geführt wird. Dadurch bleibt die gesamte Konstruktion (bis auf die Kreuzungen) „flach“. (Alle diese Gadgets sind in Abschnitt A.5 im Anhang A zu sehen.) □

Wir haben nun gesehen, dass Adventure-Modus-Minecraft (ohne Redstone) NP-schwer ist.[12] Um die Zugehörigkeit zu NP leichter zu sehen können wir weitere Einschränkungen einführen. Eine Idee wäre es, die Zeit, die der Spieler zum Erreichen des Ziels hat, zu beschränken.

Wenn bei allen Gadgets die Decke aus *„Glass“*-Blöcken konstruiert wird, dann ist der „light level“[13] auf den betretbaren Blöcken abhängig von der Tageszeit im Spiel selbst. Da ab einem gewissen „light level“ (welcher dann von der Tageszeit abhängt) gegnerische NPCs erscheinen[14], müsste der Spieler das Ziel erreichen, bevor dies eintritt, da die Spielfigur ohne Rüstung und Schwert den Gegnern unterlegen ist und es schwierig hat, das Ziel zu erreichen. In diesem Falle könnte man die Gadgets so gestalten, dass es für die Spielfigur unmöglich wird, das Ziel zu erreichen, wenn gegnerische NPCs erscheinen und somit müsste das Ziel in einer bestimmten Zeit erreicht werden.[15]

Eine andere Rechtfertigung für ein Zeitlimit ist das *Hunger*-System [B8]. Jede Aktion, die die Spielfigur ausführt reduziert den „Food level“ des Spielers. Dieser kann durch Essen wieder erhöht werden. Wenn der Level den Wert 0 erreicht, dann beginnt die Spielfigur Schaden zu nehmen. Die Anzahl an Aktionen, die der Spieler ausführen kann ist somit beschränkt. Wenn man nun die Gadgets so gestaltet, dass

[10] Der „world spawn“ befindet sich bei neuen Welten in der Nähe vom Mittelpunkt $(x, y, z) = (0, 0, 0)$, kann aber vom Autor der Welt auf einen beliebigen Punkt gesetzt werden. [B10]

[11] Wenn sich zwei Kanten e und f mehrmals schneiden, dann gibt es entweder *gerade* viele oder *ungerade* viele Schnittpunkte. Wenn es gerade viele Schnittpunkte gibt, dann sind allerdings beide zu o. B. d. A. e inzidenten Knoten auf der selben Seite (von f aus betrachtet) und es ist möglich die Kante e so zu zeichnen, dass sie f gar nicht schneidet. Wenn es ungerade viele Schnittpunkte gibt, dann sind beide zu e inzidenten Knoten in verschiedenen Gebieten. Hierbei ist es allerdings möglich die Kante so zu zeichnen, dass sie f nur einmal schneidet.

[12] Für andere Action-Adventures, bzw. Computerspiele in dreidimensionaler Umgebung, mit einer Entsperrmechanik und einer Möglichkeit, eine Bewegungsrichtung zu erzwingen kann man die NP-Schwere analog zeigen.

[13] In Minecraft ist der „light level“ eine natürliche Zahl im Intervall $[0, 15]$, wobei 0 dunkel und 15 hell ist.

[14] Gegnerische NPCs erscheinen ab einem „light level“ von 7 oder niedriger. [B9]

[15] Hierbei müsste dann allerdings die Zeit, bis es Nacht wird, von der Distanz zwischen Start und Ziel abhängen.

es in jedem Gadget eine Kiste mit Nahrung gibt, dann hat der Spieler eine genügend hohe Anzahl an Aktionen, um das Ziel zu erreichen.

Eine andere Argumentation für die Zugehörigkeit zu NP ist, dass bei unseren Einschränkungen kein Block mehrmals betreten werden muss, um vom Start zum Ziel zu kommen. Wir nutzen keinen Redstone und damit gibt es keinen Möglichkeit, dass *„Lever“* mit beliebig vielen und beliebig weit entfernten Blöcken interagieren können. Damit haben wir eine implizite Schranke an die Anzahl der Schritte, die die Spielfigur ausführen muss, um das Ziel zu erreichen. (Wir können unsere Entsperr-Gadgets auch umgestalten, dass wir keine *„Lever“*-Blöcke benötigen. Dann ist diese Argumentation noch eindeutiger, da die Spielfigur nicht mehr (beliebig oft) mit der Welt interagieren kann. Siehe hierzu Abschnitt A.6 im Anhang A.)

3.3 2048

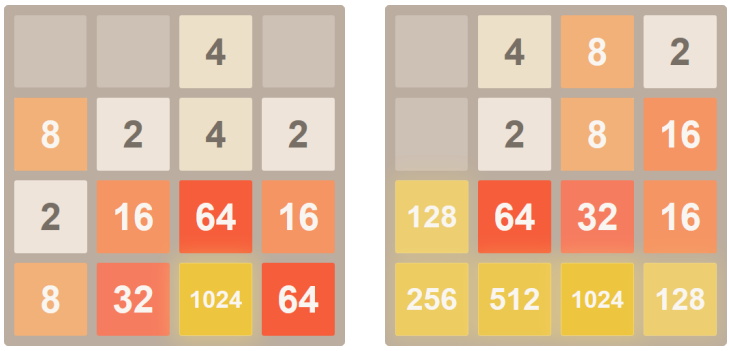
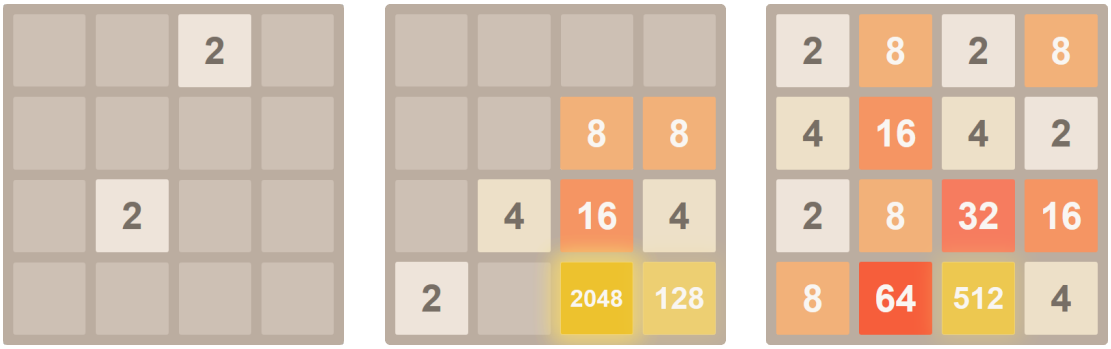


Abbildung 3.14: Zwei Screenshots aus der Browser-Version von 2048.

In diesem Abschnitt werden wir mithilfe von Edge Hop zeigen, dass das 2014 erschienene Browser- und Handy-Spiel *2048* NP-vollständig ist. Da der Spieler bei 2048 nur eine (implizit) beschränkte Anzahl von Zügen zur Verfügung hat und sich jeder Zug auf das gesamte Spielfeld auswirkt, unterscheidet sich 2048 von anderen Schiebepuzzles, welche üblicherweise PSPACE-vollständig sind. [A14]

3.3.1 Über 2048



(a) Anfangssitation.

(b) In dieser Situation hat der Spieler eine Kachel mit Wert 2048 erreicht und das Spiel somit gewonnen.

(c) In dieser Situation kann der Spieler keinen Zug mehr ausführen und hat das Spiel verloren.

Abbildung 3.15: Anfangs- und Endsituationen in 2048.

2048 ist ein im März 2014 von Gabrielle Cirulli entwickeltes Schiebepuzzle [B3], in dem der Spieler auf einem 4×4 Felder großen Spielfeld versucht, durch Verschieben von Kacheln eine bestimmte Situation zu erreichen. Jede Kachel in 2048 hat eine natürliche Zahl als Wert. Wenn sich zwei Kacheln mit gleichem Wert durch Verschieben berühren, dann verschmelzen diese zu einer Kachel, die als Wert die Summe der beiden verschmolzenen Kacheln hat. Ziel des Spiels ist es, eine Situation zu erreichen, in der mindestens eine Kachel den Wert 2048 hat. Zu Beginn des Spiels befinden sich zwei Kacheln mit je einem Wert von 2 oder 4 auf zufälligen Feldern und alle anderen Felder sind leer. Zusätzlich wird nach jedem Zug des Spielers eine

Kachel mit Wert 2 oder 4 auf ein zufällig ausgewähltes, leeres Feld gesetzt.[16] Dadurch, dass das Spiel nur Kacheln mit Wert 2 oder 4 generiert und nur Kacheln gleichen Wertes verschmelzen ergibt sich, dass jede Kachel eine Zweierpotenz als Wert hat. Wenn der Spieler keine Züge mehr ausführen kann, dann hat er das Spiel verloren. Abbildung 3.15 zeigt Beispiele der Situationen. Ein Zug in 2048 hat die Besonderheit, dass der Spieler keine bestimmte Kachel in eine Richtung schiebt, sondern sich immer alle Kacheln so weit in die gewählte Richtung bewegen, wie freie Felder vorhanden sind. Abbildung 3.16 zeigt zwei Beispielzüge.

(a) Ausgangssituation.

(b) Der Spieler hat alle Kacheln nach oben bewegt.

(c) Der Spieler hat alle Kacheln nach links bewegt.

Abbildung 3.16: Der Spieler bewegt alle Kacheln nach oben und dann nach links.

3.3.2 Komplexität

Die verallgemeinerte Version des Spiels wird beliebige Spielfeldgrößen erlauben. Wir werden aber zusätzlich noch die Siegbedingung, also den zu erreichenden Wert, als von der Feldgröße abhängig annehmen, da dadurch die Konstruktion der Gadgets einfacher wird. Es ist allerdings auch möglich, durch Konstruktion spezieller Gadgets, den zu erreichenden Wert von 2048 beizubehalten.

Definition 3.3.1 (2048-Plus)**.** Sei $n \geq m$. Ist es dem Spieler möglich, aus einer beliebigen Spielsituation auf einem $n \times m$ großen 2048-Spielfeld eine Kachel mit Wert 2^{3n-1} zu erzeugen, wenn alle neu generierten Kacheln einzigartige Werte $w < 2^{3n-1}$ besitzen?

Wir nennen dieses Problem *2048-Plus* und schreiben kurz *2048*$^+$.

Im Folgenden werden wir die Felder in den Abbildungen als Paare $(x, y) \in \mathbb{Z}$ angeben, wobei wir dem Feld unten links das Paar $(0, 0)$ zuordnen. Da alle Werte (die für die Funktionsweise unserer Gadgets wichtig sind) Zweierpotenzen sind, werden wir nur die zugehörigen Exponenten als Wert auf die jeweiligen Kacheln schreiben. Zusätzlich werden wir noch dunkel gefärbte Kacheln verwenden, auf die wir keinen Wert schreiben. Wir nehmen an, dass diese Kacheln niemals mit anderen Kacheln verschmelzen können und als Blockaden agieren. Allerdings werden wir später noch sehen, wie wir die Werte auf diesen blockierenden Kacheln wählen müssen, damit diese unsere Konstruktion nicht beeinträchtigen. Zusätzlich werden wir die Gadgets aus Gründen der Übersichtlichkeit so gestalten, dass die Eingänge immer durch

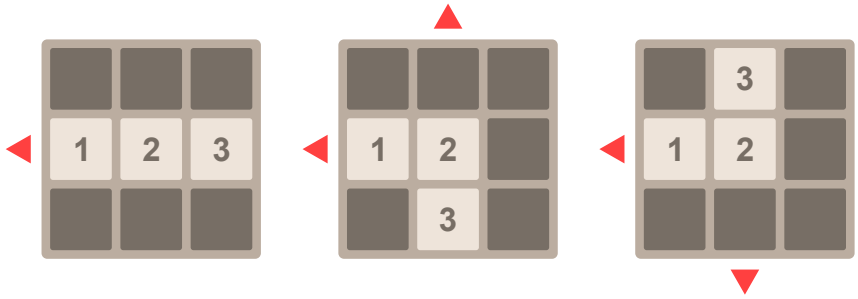
[16] Sowohl zu Spielbeginn als auch beim Setzen einer neuen Kachel nach einem Zug wird zu 90 % eine Kachel mit Wert 2 und zu 10 % eine Kachel mit Wert 4 gesetzt.

Kacheln mit einem Wert von 2^1 dargestellt werden. Auch hier werden wir uns später überlegen, wieso das kein Problem darstellt.

Außerdem werden wir unsere Gadgets so aufbauen, dass in jedem Zug höchstens eine Verschmelzung stattfindet. Daher können wir die zuletzt aus einer Verschmelzung entstandene Kachel eindeutig identifizieren und nennen die Position der Kachel *aktives Feld*. Dadurch ist es einfacher, unsere Gadgets zu beschreiben, da wir die Folge von Verschmelzungen als Bewegung durch das Spielfeld interpretieren können und die Konstruktion somit natürlicher zu Edge Hop passt. Zuletzt nehmen wir an, dass der Wert einer neuen Kachel immer so gewählt wird, dass er verschieden von den Werten der vier Nachbarkacheln ist.[17] Wir werden Kacheln, die niemals verschmelzen, dunkel und Kacheln, die für die Funktionsweise des Gadgets relevant sind, hell färben. Wir verzichten also darauf, Kacheln unterschiedlicher Werte unterschiedlich zu färben, wie es im Originalspiel der Fall ist.

Da sich Züge in 2048 nicht nur lokal auswirken, sondern sich ganze Teile einer Zeile oder Spalte verschieben, könnte es passieren, dass das Ziehen durch ein Gadget ein anderes Gadget zerstört. Wir müssen also später darauf achten, wie wir die Gadgets auf dem Spielfeld platzieren, damit dies nicht eintritt. Deshalb werden wir in allen Abbildungen durch rote Dreiecke markieren, welche Teile von Zeilen und Spalten durch das jeweilige Gadget benutzt werden. Dadurch sehen wir, wo wir keine anderen Gadgets platzieren dürfen, die zu einem späteren Zeitpunkt durchzogen werden sollen.

Um die Bewegungsabfolge auf dem Spielfeld zu notieren, geben wir eine Folge von Richtungen $r \in \{\uparrow, \rightarrow, \downarrow, \leftarrow\}$ an, in die der Spieler alle Kacheln verschiebt. Eine Zugfolge $(r_0, ..., r_n)$ ist nur dann erlaubt, wenn für alle $0 \leq i \leq n$ gilt, dass es höchstens ein aktives Feld nach dem Zug gibt.



(a) Diode. **(b)** Knick (rechts). **(c)** Knick (links).

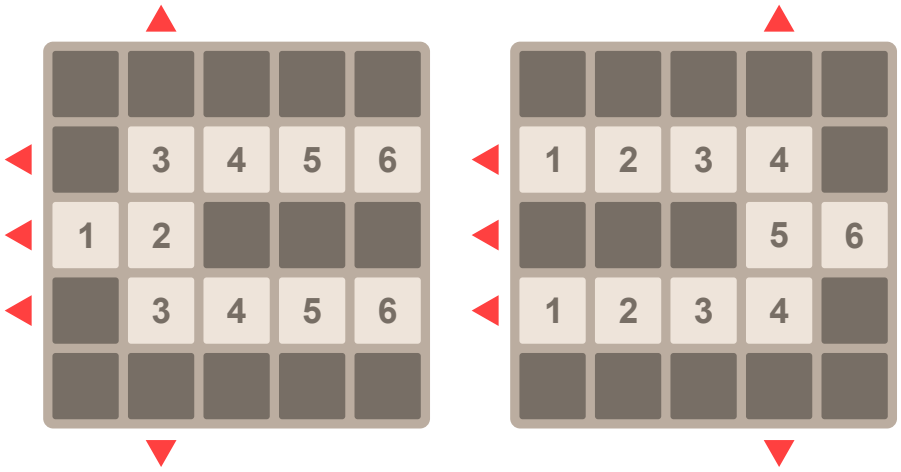
Abbildung 3.17: Dioden- und Knick-Gadgets konstruiert aus 2048-Kacheln.

Lemma 3.3.1 (Dioden-Gadget und Knick-Gadgets (2048))**.** *Angenommen es befindet sich sowohl im Dioden-Gadget (Abbildung 3.17a) als auch in den Knick-Gadgets (Abbildungen 3.17b und 3.17c) eine Kachel mit Wert* 2^1 *auf Position* $(-1, 1)$*. Dann gibt es je eine Zugfolge, so dass am Ende* $(2, 1)$ *im Dioden-Gadget,* $(1, 0)$ *im ersten Knick-Gadget und* $(1, 2)$ *im zweiten Knick-Gadget das aktive Feld ist.*

[17] Dass dies unter Verwendung der ursprünglichen Regeln im Allgemeinen nicht funktioniert ist klar: Wenn auf $(0, 0)$ eine Kachel mit Wert 2 und auf $(2, 0)$ eine Kachel mit Wert 4 liegt, dann kann auf Kachel $(1, 0)$ weder eine Kachel mit Wert 2 noch eine Kachel mit Wert 4 liegen, ohne unsere Bedingung zu verletzen. Allerdings können im originalen Regelwerk keine Kacheln mit anderem Wert erstellt werden.

Beweis. Die Zugfolge $(\rightarrow, \rightarrow, \rightarrow)$ erfüllt die Behauptung für das Dioden-Gadget. Nach dem ersten Zug ist $(0, 1)$ das aktive Feld und die sich darauf befindende Kachel hat den Wert 2^2, da sowohl die Kachel auf $(-1, 1)$ als auch die Kachel auf $(0, 1)$ zuvor den Wert 2^1 hatten. Nach dem zweiten Zug ist $(1, 1)$ das aktive Feld und die zugehörige Kachel hat (mit gleicher Argumentation) den Wert 2^3. Und nach dem letzten Zug ist $(2, 1)$ das aktive Feld, da die Kacheln auf Feld $(1, 1)$ und $(2, 1)$ verschmelzen.

Analog sind $(\rightarrow, \rightarrow, \downarrow)$ und $(\rightarrow, \rightarrow, \uparrow)$ Zugfolgen, die die Behauptung für die beiden Knick-Gadgets erfüllen. □



(a) Verzweigungs-Gadget. **(b)** Zusammenführ-Gadget.

Abbildung 3.18: Verzweigungs- und Zusammenführ-Gadgets konstruiert aus 2048-Kacheln.

Lemma 3.3.2 (Verzweigungs-Gadget (2048))**.** *Angenommen es befindet sich im Verzweigungs-Gadget (Abbildung 3.18a) eine Kachel mit Wert* 2^1 *auf Position* $(-1, 2)$*. Dann gibt es Zugfolgen, so dass entweder* $(4, 1)$ *oder* $(4, 3)$ *das aktive Feld ist.*

Beweis. Die Zugfolgen $(\rightarrow, \rightarrow, \downarrow, \rightarrow, \rightarrow, \rightarrow)$ und $(\rightarrow, \rightarrow, \uparrow, \rightarrow, \rightarrow, \rightarrow)$ erfüllen die Behauptung. Im ersten Fall ist $((0, 2), (1, 2), (1, 1), (2, 1), (3, 1), (4, 1))$ die Folge von aktiven Feldern und nach jedem Zug gilt, dass die Kachel auf dem jeweiligen aktiven Feld durch Verschmelzen einen um eins erhöhten Exponenten (also doppelt so großen Wert) hat, so dass diese mit der Kachel auf dem nachfolgenden aktiven Feld verschmelzen kann. Im zweiten Fall ist $((0, 2), (1, 2), (1, 3), (2, 3), (3, 3), (4, 3))$ die Folge von aktiven Feldern. Abbildung A.7 im Anhang A zeigt die erste der beiden Zugfolgen.

Damit gilt das Lemma. □

Lemma 3.3.3 (Zusammenführ-Gadget (2048))**.** *Angenommen es befindet sich im Zusammenführ-Gadget (Abbildung 3.18b) eine Kachel mit Wert* 2^1 *auf Position* $(-1, 1)$ *oder auf Position* $(-1, 3)$*. Dann gibt es je eine Zugfolge, so dass* $(4, 2)$ *das aktive Feld ist.*

Beweis. Die Zugfolgen $(\rightarrow, \rightarrow, \rightarrow, \rightarrow, \uparrow, \rightarrow)$ und $(\rightarrow, \rightarrow, \rightarrow, \rightarrow, \downarrow, \rightarrow)$ erfüllen die Behauptung. Im ersten Fall ist $((0, 1), (1, 1), (2, 1), (3, 1), (3, 2), (4, 2))$ die Folge von aktiven Feldern und wie im Beweis zu Lemma 3.3.2 gilt nach jedem Zug, dass die

Kachel auf dem jeweiligen aktiven Feld durch Verschmelzen einen um eins erhöhten Exponenten hat, so dass diese mit der Kachel auf dem nachfolgenden aktiven Feld verschmelzen kann. Im zweiten Fall ist $((0,3),(1,3),(2,3),(3,3),(3,2),(4,2))$ die Folge von aktiven Feldern.

Damit gilt das Lemma. □

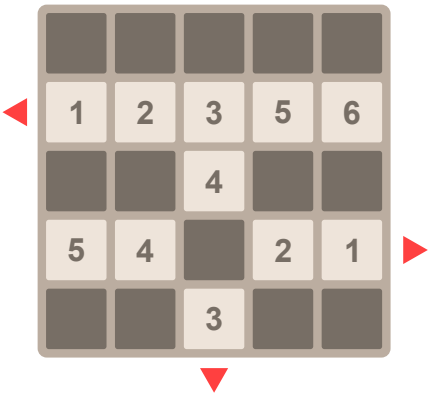


Abbildung 3.19: Ein Entsperr-Gadget konstruiert aus 2048-Kacheln. Der vertikale Abstand zwischen der zweiten und vierten Zeile kann beliebig groß sein, ohne die Funktionsweise des Gadgets einzuschränken. Wichtig ist nur, dass sich die Kachel mit Wert 4 (auf Feld $(2,2)$) direkt unter der Kachel mit Wert 3 (auf Feld $(2,3)$) befindet.

Lemma 3.3.4 (Entsperr-Gadget (2048))**.** *Angenommen im Entsperr-Gadget (Abbildung 3.19) liegt eine Kachel mit Wert* 2^1 *auf Feld* $(-1,3)$*. Dann gibt es eine Zugfolge, so dass* $(4,3)$ *das aktive Feld ist.*

Angenommen auf Feld $(5,1)$ *liegt eine Kachel mit Wert* 2^1*. Dann gibt es eine Zugfolge, die damit endet, dass* $(0,1)$ *das aktive Feld ist, wenn zu einem vorigen Zeitpunkt* $(4,3)$ *das aktive Feld war.*

Außerdem gibt es keine Zugfolge, bei der erst $(0,3)$ *und dann* $(0,1)$ *das aktive Feld ist und keine Zugfolge, bei der erst* $(4,1)$ *und dann* $(4,3)$ *das aktive Feld ist.*

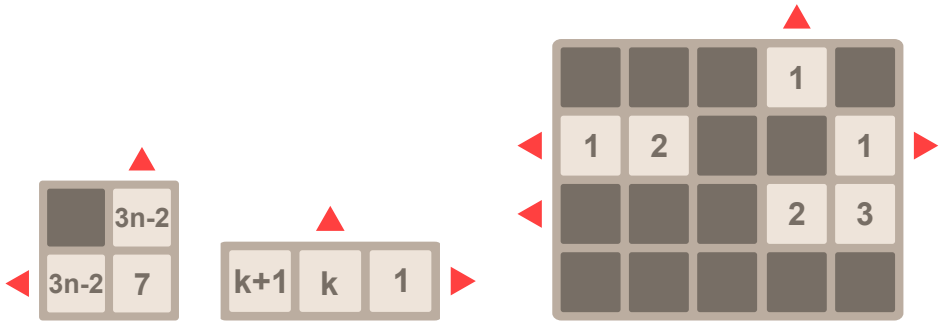
Beweis. Angenommen es liegt auf Feld $(-1,3)$ eine Kachel mit Wert 2^1. Dann ist $(\rightarrow,\rightarrow,\rightarrow,\uparrow,\rightarrow,\rightarrow)$ eine Zugfolge, die damit endet, dass Feld $(4,3)$ aktiv ist. Damit gilt die erste Aussage des Lemmas.

Damit $(1,0)$ das aktive Feld am Ende einer Zugfolge sein kann, muss diese mit $\downarrow$, $\uparrow$ oder $\leftarrow$ enden. Da die Kacheln auf $(0,0)$ und $(2,0)$ Werte haben, die nirgendwo anders vorkommen, können diese nicht mit der Kachel auf Feld $(1,0)$ verschmelzen. Damit ist der einzig mögliche letzte Zug der gesuchten Zugfolge $\leftarrow$. Die beiden Kacheln verschmelzen aber nur dann, wenn der Exponent der Kachel auf $(1,1)$ um eins erhöht ist. Das bedeutet, dass zu einem vorigen Zeitpunkt Feld $(1,1)$ aktiv sein muss. Die einzige Kachel, die durch Verschieben innerhalb des Gadgets auf ein Nachbarfeld von $(1,1)$ gelangen kann ist die Kachel auf Feld $(2,0)$ mit Wert 2^3. Diese Kachel kann nur dann auf ein Nachbarfeld von $(1,1)$, nämlich $(2,1)$ gelangen, wenn zu einem vorigen Zeitpunkt die oben genannte Zugfolge ausgeführt wurde, die damit endete, dass Feld $(4,3)$ aktiv war. Wenn diese Zugfolge ausgeführt wurde und auf Feld $(5,1)$ eine Kachel mit Wert 2^1 liegt, dann ist $(\leftarrow,\leftarrow,\leftarrow,\leftarrow,\leftarrow)$ die gesuchte Zugfolge, die damit endet, dass Feld $(0,1)$ aktiv ist.

Da es keine Möglichkeit gibt, eine Kachel mit Wert 2^3 auf Feld $(1,2)$ zu schieben, gibt es keine Zugfolge innerhalb des Gadgets, bei der erst $(0,3)$ und dann $(0,1)$ das

aktive Feld ist. Da es keine Möglichkeit gibt, eine Kachel mit Wert 2^4 auf Feld $(3, 2)$ und eine Kachel mit Wert 2^3 auf Feld $(2, 1)$ oder $(3, 0)$ zu schieben, gibt es keine Zugfolge, bei der erst Feld $(4, 1)$ und dann $(4, 3)$ aktiv ist. □

Damit haben wir die vier grundlegenden Gadgets konstruiert, die wir für eine Reduktion von EdgeHop benötigen. Nun müssen wir nur noch Start- und Ziel-Gadget konstruieren und Gadgets zur Modellierung der Kanten entwerfen. Das *Start-Gadget* ist eine einzelne Kachel mit Wert 2^1. Dies entspricht unseren Annahmen, die wir zu den jeweiligen Lemmata der Gadgets getroffen haben. Abbildung 3.20a zeigt, wie wir das *Ziel-Gadget* gestalten können. Da im EdgeHop-Graph das letzte verwendete Gadget vor dem Zielknoten ein Zusammenführ-Gadget ist und unsere Zusammenführ-Gadgets (mit 2048-Kacheln) eine Kachel mit Wert 2^6 auf dem „Ausgangsfeld“ haben, müssen wir im Zielgadget auf Feld $(1, 0)$ eine Kachel mit Wert 2^7 bereitstellen, damit diese mit der Kachel auf dem letzten aktiven Feld des letzten Zusammenführ-Gadgets verschmelzen kann. Das heißt, die letzten beiden Züge in unserer Zugfolge, um eine Kachel mit Wert 2^{3n-1} zu erreichen sind $\rightarrow, \downarrow$.



(a) Ziel. **(b)** Zurücksetzung. **(c)** Sprung.

Abbildung 3.20: Ziel-, Zurücksetz-, und Sprung-Gadgets konstruiert aus 2048-Kacheln.

Wir nehmen wieder – wie auch in den anderen Reduktionen – an, dass der zugehörige EdgeHop-Graph orthogonal ist. Abbildungen 3.17b und 3.17c zeigen, wie wir unsere Knicke mit 2048-Kacheln gestalten können. Abbildung 3.20b zeigt ein *Zurücksetz-Gadget*. Dieses funktioniert wie folgt: Wenn $(0, 1)$ das aktive Feld ist, muss die Kachel zu einer Kachel mit Wert 2^{k+1} verschmolzen sein. Dann führt der Zug $\leftarrow$ dazu, dass sich auf Feld $(0, 1)$ eine Kachel mit Wert 2^1 befindet. Diese kann dann mit $\downarrow$ ins nächste Gadget gezogen werden.

Das Dioden-Gadget (Abbildung 3.17a) kann in Verbindung mit dem Zurücksetz-Gadget beliebig weit fortgesetzt werden um (gerichtete) Kanten zu modellieren. Da die Dioden-Gadgets und die Tatsache, das jede Kante im EdgeHop-Graphen nur einmal genutzt werden kann, implizit eine Richtung der Kanten vorgeben, ist es kein Problem, dass wir gerichtete Kanten modellieren. Das Problem an dieser Konstruktion der Kanten ist allerdings, dass auf keinem Feld über bzw. unter einer horizontal verlaufenden Kante Kacheln mit $\uparrow$ bzw. $\downarrow$ verschmolzen werden dürfen, da sonst die Kachel-Kette, die diese Kante modelliert, auseinander gebrochen wird. (Analog für vertikal verlaufende Kanten und Kacheln rechts bzw. links davon.) Daher konstruieren wir ein *Sprung-Gadget* (Abbildung 3.20c) als Alternative. Beim Sprung-Gadget kann, wie der Name vermuten lässt, der horizontale Abstand zwischen Feld $(1, 2)$ und Feld $(3, 2)$ beliebig groß sein. Das bedeutet, in diesen Spalten kann beliebig oft $\downarrow$ oder $\uparrow$ ausgeführt werden.

Lemma 3.3.5 (Sprung-Gadget). *Angenommen es liegt im Sprung-Gadget (Abbildung 3.20c) eine Kachel mit Wert* 2^1 *auf Feld* $(-1, 2)$*. Dann gibt es eine Zugfolge, so dass Feld* $(4, 1)$ *aktiv ist. Diese Zugfolge existiert, unabhängig davon, wie oft Kacheln in Spalte* 2 *verschmolzen wurden.*

Beweis. Eine Zugfolge, die damit endet, dass Feld $(4, 1)$ aktiv ist, ist $(\rightarrow, \leftarrow, \downarrow, \downarrow, \rightarrow)$. Durch den zweiten Zug in dieser Folge $(\leftarrow)$ wird die Kachel von Feld $(4, 2)$ auf Feld $(3, 2)$ gezogen und damit kann die darüber liegende Kachel durch $\downarrow$ mit dieser verschmelzen und Feld $(3, 2)$ ist das aktive Feld. Dadurch, dass keine Kachel aus Spalte 2 für diese Zugfolge benötigt wird, ist es klar, dass es egal ist, welche Kacheln dort liegen und es also auch egal ist, wie oft in dieser Spalte Kacheln nach oben oder unten verschoben werden.

Es ist ebenfalls leicht zu sehen, dass diese Zugfolge auch dann funktioniert, wenn der Abstand zwischen der Kachel auf Feld $(1, 2)$ und der Kachel auf Feld $(1, 3)$ beliebig groß ist, da der zweite Zug in unserer Zugfolge $(\leftarrow)$ *alle* Kacheln rechts von $(1, 2)$ (die sich in der gleichen Zeile befinden) ein Feld nach links verschiebt. □

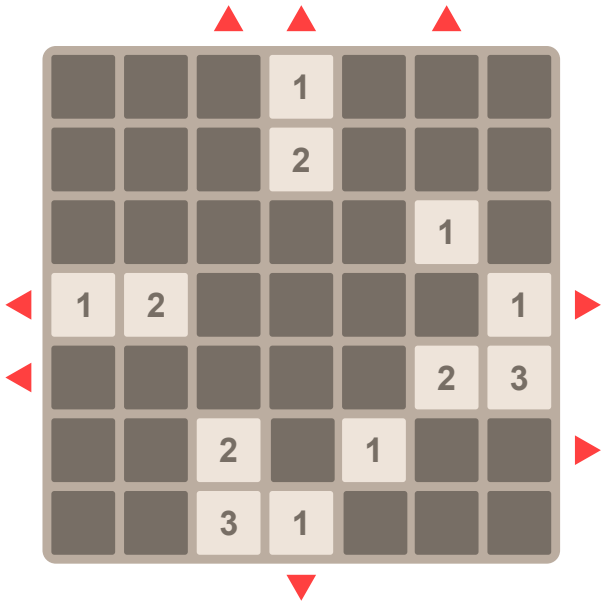


Abbildung 3.21: Ein Kreuzungs-Gadget konstruiert aus 2048-Kacheln.

Da der EdgeHop-Graph, aus dem wir eine Spielsituation konstruieren, nicht planar ist müssen wir noch das Kreuzungs-Gadget konstruieren um Kanten zu modellieren, die sich kreuzen. Da wir allerdings schon gesehen haben, dass wir ein Sprung-Gadget konstruieren können, ist es nicht überraschend, dass das Kreuzungs-Gadget aus zwei Sprung-Gadgets besteht. Abbildung 3.21 zeigt ein Kreuzungs-Gadget. Dass das Kreuzungs-Gadget funktioniert folgt unmittelbar aus Lemma 3.3.5.

Lemma 3.3.6 (NP-Schwere). *2048⁺ ist NP-schwer.*

Beweis. Lemmata 3.3.1, 3.3.2, 3.3.3 und 3.3.4 zeigen, wie die grundlegenden EdgeHop-Gadgets konstruiert werden können. Ebenfalls haben wir uns oben überlegt, wie Start- und Ziel-Gadget (Abbildung 3.20a) aussehen und wie wir die Kanten modellieren müssen (Lemma 3.3.5 und Abbildungen 3.17b, 3.17c, 3.20b und 3.21). Damit können wir prinzipiell einen EdgeHop-Graphen konstruieren. Ein Problem

bleibt bei einer naiven Konstruktion allerdings bestehen: Da sich Züge in 2048 global auswirken, können Gadgets, die sich in der selben Zeile oder Spalte befinden durch einen Zug zerstört werden. Wir haben in allen Abbildungen markiert, welche Teile einer Zeile oder Spalte durch Durchqueren eines Gadget zerstört werden. Wenn wir ein Gadget auf dem Spielfeld platzieren, dann dürfen wir es nicht so platzieren, dass es auf einer markierten Teilspalte oder -zeile eines Gadgets liegt, welches zu einem früheren Zeitpunkt besucht werden kann. Falls eine Kante des Graphen eine markierte Zeile oder Spalte kreuzt, so müssen wir an dieser Stelle ein Sprung-Gadget setzen, um die markierte Zeile (bzw. Spalte) zu überspringen.[18] Nachdem alle Gadgets platziert wurden, müssen wir noch blockierende Kacheln auf die leeren Felder setzen.

Da wir angenommen haben, dass jede neu generierte Kachel einen einzigartigen Wert (der keine Zweierpotenz ist) besitzt, werden neu generierte Kacheln niemals mit einer Kachel auf dem Spielfeld verschmelzen können. Dadurch können diese Kacheln unsere Konstruktion nicht beeinträchtigen. Weiter sind all unsere Gadgets so konstruiert, dass sie nur mit einer bestimmte Zugfolge durchquert werden können, welche dem Ziehen durch den EdgeHop-Graph entspricht. Das heißt, dass eine Kachel mit dem Zielwert 2^{3n-1} nur dann erzeugt werden kann, wenn durch die gesamte Konstruktion gezogen wird.

Da EdgeHop NP-schwer ist, ist auch 2048^+ NP-schwer. □

Für die NP-Vollständigkeit müssen wir nun noch zeigen, dass $2048^+ \in NP$ gilt. Hierfür werden wir allerdings keinen genauen Beweis angeben, sondern nur eine grobe Beweisskizze, die verdeutlicht, wieso die Eigenschaft gilt.

Lemma 3.3.7. *$2048^+ \in NP$.*

Beweisidee. Wir nehmen an, dass neu generierte Kacheln immer einen einzigartigen Wert haben, der keine Zweierpotenz ist. Dann gilt, dass keine neu generierte Kachel jemals mit einer anderen Verschmelzen kann und damit ein Feld blockiert. Da unser Spielfeld $n \times m$ Felder groß ist, ist nach maximal $n \cdot m$ Zügen das Spielfeld voll mit blockierenden Kacheln und der Spieler hat verloren. Das bedeutet allerdings, dass keine Zugfolge aus mehr als $n \cdot m < n^2$ Zügen bestehen kann und somit kann in Polynomzeit überprüft werden, ob eine gegebene Zugfolge eine Kachel mit Wert 2^{3n-1} erzeugt oder nicht. □

In unseren Konstruktionen haben wir angenommen, dass die dunklen Kacheln nicht mit anderen Kacheln verschmelzen können. Diese Eigenschaft ist gewährleistet, wenn beispielsweise alle dunklen Kacheln einzigartige Werte haben. Wenn wir ein beliebiges Feld mit beliebigen Kacheln (mit Werten kleiner als 2^{3n-1}) zulassen, dann gibt es $2^{3n-1} - 1$ verschiedene Kachelwerte, von denen wir nur diejenigen für unsere Gadgets nutzen, die Zweierpotenzen sind. Die anderen $2^{3n-1} - 1 - (3n - 1)$ Werte können beliebig auf die dunklen Kacheln verteilt werden. Unser Spielfeld ist $n \times m$ Felder groß (mit $n \geq m$) und da $n^2 \leq 2^{3n-1} - 3n$ (für $n \in \mathbb{N}$) kann jede dunkle Kachel mit einem einzigartigen Wert versehen werden.

Eine andere Idee, die blockierenden Kacheln zu gestalten ist es, unsere Problemstellung so zu modifizieren, dass die Frage lautet, ob es möglich ist eine *weitere*

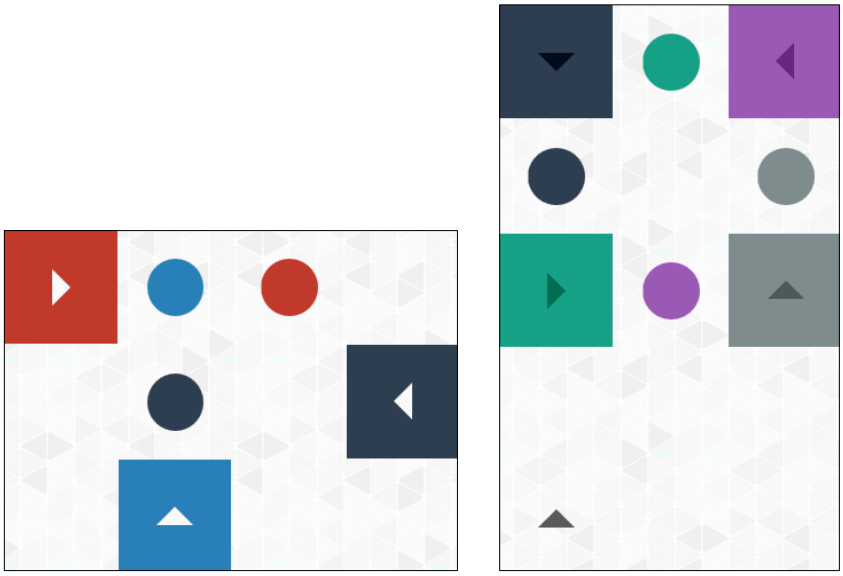
[18] Bei den meisten unserer Gadgets ist es so, dass sich nur Kacheln links vom Gadget verschieben, weswegen es bei einem Durchqueren der Gadgets von links nach rechts (vgl. die Konstruktion auf Abbildung 2.7) zu keinem Problem führt, diese Gadgets in einer Reihe zu platzieren.

Kachel mit einem Wert von 2048 zu erreichen. Dann können wir unsere Gadgets so gestalten, dass nur Kacheln mit Werten $w \in \{2^k : 1 \leq k \leq 11\}$ für die Funktionsweise der Gadgets genutzt werden und die dunklen Kacheln beliebige Werte (sogar beliebige Zweierpotenzen) größer als 2048 haben. Dass dies möglich ist, ist leicht zu sehen: Abgesehen vom Ziel-Gadget ist der höchste Wert, der auf einer Kachel vorkommt 2^7. Das Ziel-Gadget kann so angepasst werden, dass die beiden Kacheln mit Wert 2^{3n-1} durch Kacheln mit Wert 2048 ersetzt werden. Sowohl das Zurücksetz-Gadget als auch das Sprung-Gadget sorgen dafür, dass die Werte auf modellierten Kanten nicht beliebig wachsen. Da wir zusätzlich in unserer Konstruktion nur einen ein Block breiten Pfad nutzen, können wir problemlos die Einschränkung treffen, dass nur Kacheln mit Wert 2^1 und 2^2 neu generiert werden, da Pfade 2-färbbar sind. [A8]

Eine noch offene Frage die sich stellt ist, ob ein „echtes“ 2048 NP-schwer ist. Echt in diesem Sinne meint, dass auch die blockierenden Kanten Werte $w \in \{2^k : 1 \leq k \leq 10\}$ besitzen. Das Problem an dieser Fragestellung ist, dass sich Züge in 2048 global auswirken. In unseren Überlegungen konnten wir diese Eigenschaft „umgehen“, in dem wir blockierende Steine auf dem gesamten Spielfeld platziert hatten, um die Auswirkungen eines Zuges gering zu halten. Dennoch haben die Züge auch bei unserer Konstruktion unweigerlich blockierende Kacheln verschoben. Bei einem echten 2048 hat man allerdings weniger verschiedene Werte zu Verfügung, um blockierende Kacheln zu konstruieren, so dass sich diese eventuell schon durch eine geringe Anzahl an Zügen neben anderen (blockierenden) Kacheln mit selbem Wert befinden können. Für diese Problemstellung müsste man sowohl die Anzahl der blockierenden Kacheln in den Gadgets, als auch die Anzahl der benötigten Züge in benachbarten Reihen und Spalten minimieren, damit es nicht vorkommen kann, dass die blockierenden Kacheln so oft verschmelzen, dass diese zu einer Kachel mit Wert 2048 verschmelzen.

Andere Komplexitätsbetrachtungen zu 2048 gibt es bei [B2] und [A20], wobei die Autoren die Spielregeln etwas anders auslegen. Christopher Chen betrachtet die Komplexität einer 2048-Version, bei der *keine* neuen Kacheln generiert werden und zeigt, dass dieses Problem in NP liegt. Rahul Mehta interpretiert 2048 als ein Zweipersonen-Spiel, bei dem Spieler $\mathcal{A}$ die Kacheln verschiebt (wie auch in unserer Version) und Spieler $\mathcal{B}$ k neue Kacheln mit Werten aus $\{2^0, ..., 2^k\}$ (für $k \in \mathbb{Z}_{>0}$) auf beliebige leere Positionen des Spielfelds setzt. Weiter wird die neue Kachel von Spieler $\mathcal{B}$ gesetzt, nachdem Kacheln verschmelzen, aber bevor sich die restlichen Kacheln auf dem Spielfeld verschieben. Dadurch ergibt sich ein PSPACE-vollständiges Problem. Zusätzlich zeigt Rahul Mehta noch, dass für *„2048-k-Moves“*, also die für Frage, ob eine Spielsituation in k Zügen erreicht werden kann, ein FPT-Algorithmus existiert.

3.4 Game About Squares



(a) Level 2. **(b)** Level 20.

Abbildung 3.22: Zwei Screenshots aus der Browser-Version von Game about Squares.

In diesem Abschnitt werden wir mithilfe von Edge Hop zeigen, dass das 2014 erschienene Browser-Spiel *Game about Squares* NP-schwer ist. Wir geben also einen alternativen Beweis für die 2014 veröffentlichen Ergebnisse von Jens Maßberg an. [A19] Da Jens Maßbergs Beweis direkt von SAT reduziert ist unser alternativer Beweis weniger aufwändig.

3.4.1 Über Game about Squares

Game about Squares (GAS) ist ein 2014 von Andrey Shevchuk entwickeltes Schiebepuzzle [B5], in dem der Spieler auf einem unbeschränkten, in Felder eingeteilten Spielfeld versucht, farbige Quadrate auf bestimmte Felder zu schieben. Jedes dieser farbigen Quadrate kann nur in eine bestimmte Richtung verschoben werden. Die Richtung, in welche ein Quadrat geschoben werden kann, ist zu Spielbeginn festgelegt, kann aber durch Verschieben auf speziell gekennzeichnete Felder geändert werden. Im Gegensatz zu z. B. *Sokoban* kann in GAS ein Quadrat beliebig viele andere Quadrate verschieben. Ebenfalls gibt es in GAS keine Spielfigur, die sich auch auf dem Spielfeld befindet. Der Spieler kann beliebige Quadrate verschieben, indem er sie (mit dem Mauszeiger) anklickt. Jeder Klick auf ein Quadrat verschiebt dieses um genau ein Feld in die dem Quadrat zugewiesene Richtung. Abbildung 3.22 zeigt Level 2 und Level 20 aus der Browser-Version des Spiels. Die farbigen Quadrate sind die Quadrate, die der Spieler verschieben muss. Die farbigen Kreise sind die jeweiligen Zielpositionen der gleichfarbigen Quadrate. Die Dreiecke auf den Quadraten geben an, in welche Richtung das Quadrat verschoben werden kann.[19] Wenn

[19] Wenn das Dreieck auf dem Quadrat dunkel gefärbt ist, dann zeigt dies an, dass sich auf dem Feld, auf dem sich das Quadrat befindet, selbst auch ein Dreieck befindet. Wenn das Dreieck hell gefärbt ist, dann bedeutet dies, dass das Feld, auf dem sich das Quadrat befindet, keine besondere Eigenschaft hat.

ein Quadrat auf das Dreieck im unteren Teil von Abbildung 3.22b geschoben wird, ändert sich die Bewegungsrichtung des Quadrates auf △. Abbildung A.8 im Anhang A zeigt eine Beispielzugfolge zur Lösung eines Levels aus GAS.

3.4.2 Komplexität

Wir definieren eine *„Game about Squares"*-Instanz als 5-Tupel $\mathcal{G} = (C, r, p, z, d)$ mit $C \subset \mathbb{N}$ eine endliche Menge von Farben (die somit den Quadraten im Spiel entsprechen), $r : C \to \{\triangle, \triangleright, \triangledown, \triangleleft\}$ eine Funktion, die jedem Quadrat eine initiale Bewegungsrichtung zuweist, $p : C \to \mathbb{Z}^2$ eine Funktion, die jedem Quadrat seine initiale Position zuweist, $z : C \to \mathbb{Z}^2$ eine partielle Funktion, die Quadraten ihre Zielposition (also die Position der gleichfarbigen Kreise) zuweist und $d : \{\blacktriangle, \blacktriangleright, \blacktriangledown, \blacktriangleleft\} \to \mathcal{P}(\mathbb{Z}^2)$ eine Funktion, die die Positionen der Dreiecke angibt. Weiter soll gelten:

- p ist injektiv (D. h. die Quadrate haben paarweise verschiedene Positionen.)
- z ist injektiv (D. h. die Zielpositionen sind paarweise verschieden.)
- $\forall x, y \in \{\blacktriangle, \blacktriangleright, \blacktriangledown, \blacktriangleleft\}\ x \neq y \Rightarrow d(x) \cap d(y) = \emptyset$ (D. h. verschiedenartige Dreiecke haben paarweise verschiedene Positionen.)
- $\forall c \in C, x \in \{\blacktriangle, \blacktriangleright, \blacktriangledown, \blacktriangleleft\} : p(c) \in d(x) \Rightarrow r(c) = x$ (D. h. wenn ein Quadrat auf einem Feld mit einem Dreieck startet, dann muss die initiale Richtung des Quadrates dem Dreieck entsprechen.)

Definition 3.4.1 (Game about Squares)**.** Gibt es zu einer beliebigen Instanz von *Game about Squares* eine Folge von Zügen, so dass alle Kreise von Quadraten entsprechender Farbe besetzt sind?

Wir nennen dieses Problem *Game about Squares* und schreiben kurz GAS.

(a) Ein leeres Feld.

(b) Ein Feld das die Zielposition eines blauen Quadrates anzeigt.

(c) Ein Dreieck (▼). Wenn ein Quadrat dieses Feld betritt, ändert es seine Bewegungsrichtung auf ▽.

(d) Ein Quadrat mit Bewegungsrichtung ▷.

(e) Ein Quadrat auf einem Feld mit einem Dreieck (▶).

(f) Ein Quadrat auf einem Feld mit einem orangefarbenen Kreis.

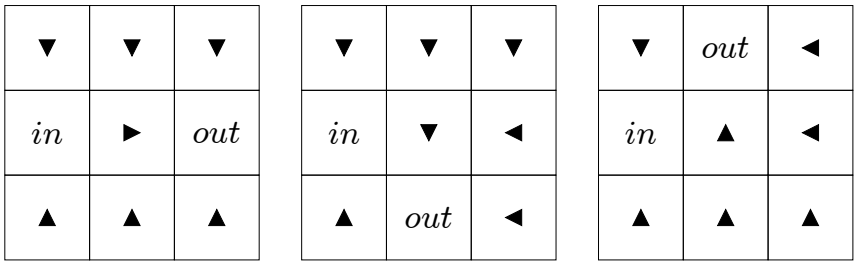
Abbildung 3.23: Symbole, die wir in unseren Abbildungen verwenden.

Da jedes Quadrat zu jedem Zeitpunkt eine eindeutige Bewegungsrichtung hat und alle Quadrate voneinander unterscheidbar sind, werden wir zur Notation der Zugfolge eine Folge von Farben angeben, die der Reihenfolge entspricht, in der der Spieler die verschiedenen Quadrate anklickt. Wir werden das orangefarbene Quadrat mit o, das blaue Quadrat mit b und das rote Quadrat mit r bezeichnen. Für lange

Zugfolgen oder Teilfolgen, die aus dem selben Zug bestehen, werden wir die folgende abkürzende Schreibweise (für $n > 0$) nutzen:

$$(x^n) := \begin{cases} (x), & n = 1 \\ (x^{n-1}, x), & n > 1 \end{cases}$$

Weiter werden wir wie üblich die einzelnen Felder durch Paare $(x, y) \in \mathbb{Z}^2$ angeben, wobei wir dem Feld unten links das Paar $(0, 0)$ zuordnen. Außerdem werden wir annehmen, dass keines der Gadgets über ein anderes als die gekennzeichnet Eingangs- und Ausgangsfelder verlassen werden kann. Hierfür werden wir jedes Gadget mit einem Rahmen von ins Gadget zeigenden Dreiecken umgeben. Damit diese Eigenschaft sichergestellt ist, muss die Breite des Rahmens ein Feld größer als die Anzahl der Quadrate im Gadget sein. Aus Gründen der Übersichtlichkeit werden wir allerdings den Rahmen immer nur mit einer Breite von einem Feld einzeichnen. Abbildung 3.23 zeigt, welche Symbole wir in unseren Abbildungen verwenden und was diese bedeuten.



(a) Dioden-Gadget. **(b)** Knick-Gadget (rechts). **(c)** Knick-Gadget (links).

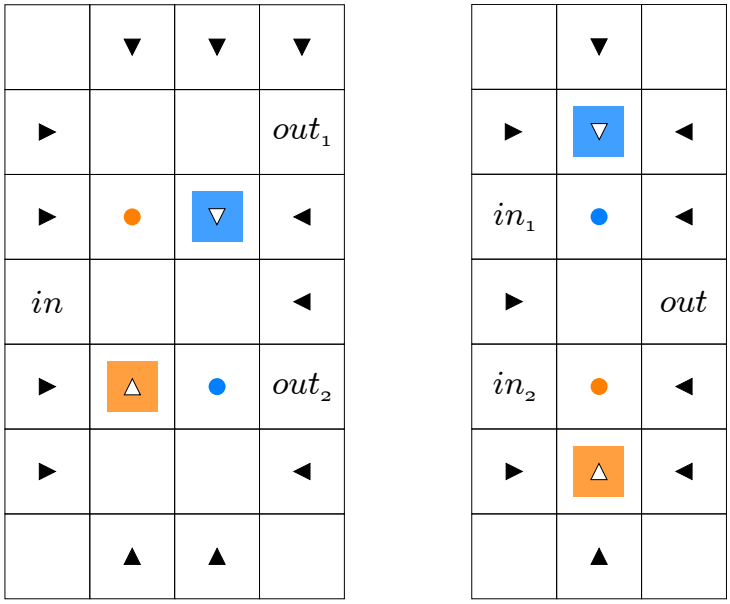
Abbildung 3.24: Dioden- und Knick-Gadgets konstruiert als *„Game about Squares"*-Spielsituationen.

Lemma 3.4.1 (Dioden-Gadget und Knick-Gadgets (GAS))**.** *Angenommen es befindet sich sowohl im Dioden-Gadget (Abbildung 3.24a) als auch in den Knick-Gadgets (Abbildungen 3.24b und 3.24c) ein rotes Quadrat mit Bewegungsrichtung $\triangleright$ auf Feld in. Dann gibt es eine Zugfolge, so dass dieses Quadrat Feld out erreicht.*

Beweis. In allen drei Fällen führt die Zugfolge (r^2) zum Feld *out*. Bei beiden Knick-Gadgets sorgt das Dreieck auf Feld $(1, 1)$ dafür, dass das Quadrat die nötige Bewegungsrichtung ($\triangledown$ bzw. $\triangle$) erhält, so dass es beim zweiten Zug in die richtige Richtung geschoben wird. □

Lemma 3.4.2 (Verzweigungs-Gadget (GAS))**.** *Angenommen es befindet sich ein rotes Quadrat mit Bewegungsrichtung $\triangleright$ auf Feld in. Dann gibt es Zugfolgen, so dass dieses Quadrat zu Feld out_1 bzw. out_2, das orangefarbene Quadrat zum orangefarbenen Kreis und das blaue Quadrat zum blauen Kreis gelangt.*

Beweis. Eine Zugfolge, die ein rotes Quadrat von *in* nach out_1 führt ist (b^2, r, o^2, r^2). Die ersten beiden Züge sorgen dafür, dass das blaue Quadrat nach der Zugfolge auf dem blauen Kreis steht. die nächsten drei Züge sorgen dafür, dass das rote Quadrat auf Feld $(1, 5)$ geschoben wird. Zusätzlich befindet sich nach diesen drei



(a) Verzweigungs-Gadget. **(b)** Zusammenführ-Gadget.

Abbildung 3.25: Verzweigungs- und Zusammenführ-Gadgets konstruiert als *„Game about Squares“*-Spielsituationen.

Zügen das orangefarbene Quadrat auf dem orangefarbenen Kreis. Die letzten beiden Züge führen dazu, dass das rote Quadrat auf Feld out_1 geschoben wird. (Abbildung A.9 im Anhang A zeigt diese Zugfolge.) Analog ist (o^2, r^2, b, r, b) eine Zugfolge, die ein rotes Quadrat von *in* nach out_2 führt. □

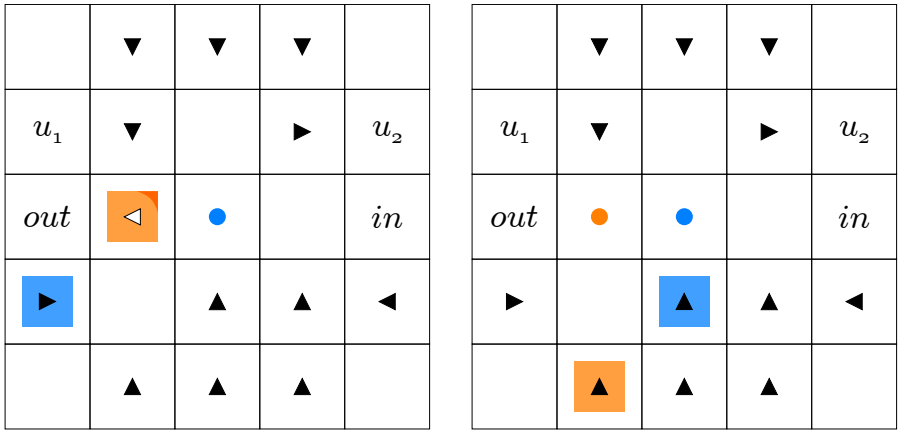
Lemma 3.4.3 (Zusammenführ-Gadget (GAS)). *Angenommen es befindet sich ein rotes Quadrat mit Bewegungsrichtung $\triangleright$ auf Feld in_1 oder auf Feld in_2. Dann gibt es Zugfolgen, so dass dieses Quadrat zu Feld out, das orangefarbene Quadrat zum orangefarbenen Kreis und das blaue Quadrat zum blauen Kreis gelangt.*

Beweis. Angenommen, es befindet sich ein rotes Quadrat auf Feld in_1. Dann ist (o, r, b, r) eine Zugfolge, die dieses Quadrat zu Feld *out* führt. Nach dem ersten Zug dieser Zugfolge befindet sich das orangefarbene Quadrat auf dem gleichfarbigen Kreis. Nach den nächsten beiden Zügen ist das rote Quadrat auf der selben Höhe wie Feld *out*. Außerdem ist zu diesem Zeitpunkt auch das blaue Quadrat auf dem blauen Kreis. Nach dem letzten Zug befindet sich das rote Quadrat auf Feld *out*. Analog ist (b, r, o, r) eine Zugfolge, die ein rotes Quadrat von Feld in_2 nach *out* führt. □

Lemma 3.4.4 (Entsperr-Gadget (GAS)). *Angenommen im Entsperr-Gadget (Abbildung 3.26a) befindet sich ein rotes Quadrat mit Bewegungsrichtung $\triangleright$ auf Feld u_1. Dann gibt es eine Zugfolge, so dass dieses Quadrat Feld u_2 erreicht.*

Angenommen es befindet sich ein rotes Quadrat mit Bewegungsrichtung $\triangleleft$ auf Feld in. Dann gibt es eine Zugfolge, so dass dieses Quadrat Feld out erreicht, wenn zu einem vorigen Zeitpunkt ein Quadrat von Feld u_1 zu Feld u_2 gezogen ist. Zusätzlich befindet sich nach dieser Zugfolge das orangefarbene Quadrat auf dem orangefarbenen Kreis und das blaue Quadrat auf dem blauen Kreis.

Außerdem kann kein Quadrat mit Bewegungsrichtung $\triangleright$ von Feld u_1 nach Feld out und kein Quadrat mit Bewegungsrichtung $\triangleleft$ von Feld in nach u_2 ziehen.



(a) Ursprungszustand.

(b) Zustand, nach dem ein Quadrat von u_1 nach u_2 gezogen ist.

Abbildung 3.26: Ein Entsperr-Gadget konstruiert als *„Game about Squares"*-Spielsituation.

Zuletzt gibt es eine Zugfolge, so dass das blaue Quadrat zum blauen Kreis und das orangefarbene Quadrat zum orangefarbenen Kreis gelangt.

Beweis. Angenommen es befindet sich ein rotes Quadrat mit Bewegungsrichtung $\triangleright$ auf Feld u_1. Dann ist (r^3, b^2, r^3) eine Zugfolge, bei der das rote Quadrat Feld u_2 erreicht. Nach dem ersten Zug ändert sich die Bewegungsrichtung des Quadrats auf $\triangledown$. Die nächsten beiden Züge schieben das orangefarbene Quadrat auf Feld $(1, 0)$, so dass sich dessen Bewegungsrichtung auf $\triangle$ ändert. Durch den vierten und fünften Zug wird das rote Quadrat auf Feld $(3, 1)$ geschoben. Zusätzlich ändern sich die Bewegungsrichtungen des blauen und des roten Quadrates auf $\triangle$. Im vorletzten Zug ändert sich die Bewegungsrichtung des roten Quadrates auf $\triangleright$ und nach dem letzten Zug befindet sich das rote Quadrat auf Feld u_2.

Angenommen es befindet sich ein rotes Quadrat mit Bewegungsrichtung $\triangleleft$ auf Feld *in*. Damit dieses Quadrat Feld *out* erreichen kann, muss das orangefarbene Quadrat auf ein anderes Feld geschoben werden. Das orangefarbene Quadrat kann nicht nach links geschoben werden, weil es dann nicht mehr den orangefarbenen Kreis erreichen kann. Ebenfalls kann es nicht nach rechts geschoben werden, weil seine Bewegungsrichtung $\triangleleft$ ist und sich auf Feld *out* kein Quadrat mit Bewegungsrichtung $\triangleright$ befindet. Weiter kann das orangefarbene Quadrat nicht nach oben geschoben werden, da es nicht die passende Bewegungsrichtung hat und weder ein Quadrat auf $(0, 0)$, $(1, 0)$ noch auf $(1, 1)$ liegt. Das bedeutet, das orangefarbene Quadrat muss nach unten verschoben werden, damit ein anderes Quadrat von *in* nach *out* geschoben werden kann. Das orangefarbene Quadrat kann aber nur von einem sich auf Feld $(1, 3)$ befindenden Quadrat nach unten verschoben werden. Feld $(1, 3)$ kann nur erreicht werden, wenn ein Quadrat von Feld u_1 zieht. Zusammen bedeutet das, dass das orangefarbene Quadrat nur dann nicht den Weg von *in* nach *out* blockiert, wenn ein anderes Quadrat zuvor von u_1 nach u_2 gezogen ist. Sei nun ein rotes Quadrat mit Bewegungsrichtung $\triangleleft$ auf Feld *in* und sei die Zugfolge (r^3, b^2, r^3) zu einem früheren Zeitpunkt ausgeführt worden (siehe Abbildung 3.26b), dann ist (r^4, b, o^2) eine Zugfolge, bei der das rote Quadrat Feld *out* erreicht und sowohl das orangefarbene als auch das blaue Quadrat die jeweils gleichfarbigen Kreise erreichen.

Weiter ist es leicht zu sehen, dass kein Quadrat mit Bewegungsrichtung $\triangleright$ von Feld u_1 nach *out* und kein Quadrat mit Bewegungsrichtung $\triangleleft$ von Feld *in* nach Feld

u_2 ziehen kann.

Eine Zugfolge, die dafür sorgt, dass sowohl das blaue, als auch das orangefarbene Quadrat auf die jeweils gleichfarbige Kreise gelangen ist (b^3). Im Ursprungszustand ist das orangefarbene Quadrat schon auf dem gleichfarbigen Kreis und die Zugfolge schiebt das blaue Quadrat auf den blauen Kreis. □

Lemma 3.4.5 (NP-Schwere)**.** GAS *ist NP-schwer.*

Beweis. Satz 2.2.9 hat uns gezeigt, dass EDGEHOP NP-vollständig ist. Mithilfe der gerade konstruierten Gadgets (Lemmata 3.4.1, 3.4.2, 3.4.3 und 3.4.4) können wir zu jedem Gadget des EDGEHOP-Graphen eine passende „Game about Squares"-Spielsituation konstruieren.

Ein spezielles Start-Gadget müssen wir nicht konstruieren. Wir setzen nur ein rotes Quadrat mit Bewegungsrichtung $\triangleright$ auf Feld *in* des ersten Verzweigungs-Gadgets. Ein spezielles Ziel-Gadget brauchen wir ebenfalls nicht. Wir setzen einen roten Kreis auf Feld *out* des letzten Entsperr-Gadgets. Mit dieser Konstruktion können alle Kreise innerhalb der Gadgets durch gleichfarbige Quadrate innerhalb der selben Gadgets erreicht werden und nur das rote Quadrat muss durch die gesamte Konstruktion ziehen, damit es den roten Kreis erreichen kann. Um nun weiter einen EDGEHOP-Graphen zu konstruieren brauchen wir noch Kanten und Kreuzungen. Wenn wir einen orthogonalen Graphen zugrunde legen, dann brauchen wir wieder (wie in den vorigen Reduktionen auch) nur horizontale bzw. vertikale Kanten und Knicke. Die Knicke sind in Lemma 3.4.1 beschrieben. Für horizontale und vertikale Kanten brauchen wir keine speziellen Gadgets, weil ein Quadrat nicht von sich aus die Richtung wechseln kann. Aus dem gleiche Grund müssen wir auch keine Kreuzungen konstruieren. (Vgl. die Argumentation zu Latrunculi.) Da ebenfalls nur ein einzelnes Quadrat außerhalb der Gadgets auf den Kanten zieht, können sich auch keine Quadrate an Kreuzungen in ungewollte Richtungen verschieben.

Damit ist GAS NP-schwer. □

Wir haben nun also gesehen, dass GAS NP-schwer ist und damit die Ergebnisse von Jens Maßberg verfestigt. Die Frage, die offen bleibt ist allerdings, ob das Problem *NP-vollständig* ist. Bei GAS gibt es keine implizite (oder explizite) Beschränkung der Züge, wie bei unseren vorigen Problemen. Daher können wir an dieser Stelle nicht argumentieren, dass nur Zugfolgen gewisser Länge geprüft werden müssen. Auf der einen Seite ist GAS ähnlich zu Spielen wie *Sokoban*, welche *PSPACE-vollständig* sind, aber auf der anderen Seite gibt aus auch Ähnlichkeiten zu Spielen wie *Push-∗-X*, die *NP-vollständig* sind. [A14] Der grobe Unterschied zwischen beiden Problemen ist, dass bei Sokoban eine komplette Konfiguration des Spielfelds gesucht wird, während bei Push-∗-X nur ein Block (bzw. die schiebende Spielfigur) zu einer gewissen Position gelangen muss. Bei GAS kommt es darauf an, wie viele Kreise auf dem Spielfeld sind. Wenn es nur ein Kreis auf dem Spielfeld gibt, dann muss auch nur ein Quadrat zu einer Zielposition gelangen (wie bei Push-∗-X), wohingegen das Spiel sich eher wie Sokoban spielt, wenn mehrere Kreise auf dem Spielfeld sind.

Kapitel 4

Fazit

4.1 Zusammenfassende Worte

In dieser Arbeit haben wir das Spiel *Edge Hop* kennengelernt und gesehen, dass die zugehörige Entscheidungsfrage, EdgeHop, NP-vollständig ist. Für die NP-Schwere haben wir sogar zwei verschiedene Beweise angegeben. Im zweiten Teil haben wir uns mit vier verschiedenen Spielen beschäftigt und auch ihre Komplexität betrachtet. Alle vier Spiele hatten gemeinsam, dass sie NP-schwer sind und dass dies mithilfe von EdgeHop gezeigt wurde. Sowohl bei Latrunculi (bzw. 1-Zug-Latrnuculi), als auch bei 2048 (bzw. 2048^+) haben wir neben der NP-Schwere auch gesehen, dass die Probleme in NP liegen, in dem wir argumentiert haben, dass eine zum Erfolg führende Zugfolge nicht beliebig lang sein kann. Bei Minecraft (also AMC-R) haben wir nicht explizit gezeigt, dass unser Problem in NP liegt, haben aber argumentiert, welche Änderungen wir an dem Problem treffen müssen, damit es einen NP-Algorithmus gibt. Bei Game about Squares (also GAS) haben wir keinen Algorithmus angeben können, haben uns aber kurz überlegt, dass das Spiel Ähnlichkeit zu Sokoban und auch zu Push-∗-X hat und somit in PSPACE aber vielleicht auch in NP liegt.

	EdgeHop	1-Zug-Latrunculi	*AMC-R*	2048^+	GAS
Kante	1	1	∗	5	∗
Knick	1	2	2	3	2
Diode	5	7	8	3	2
Verzweigung	1	3	5	6	7
Zusammenführung	1	3	5	6	4
Entsperren	6	3	5	6	8
Überprüfen	4	1	4	5	7

Tabelle 4.1: Anzahl der Züge, die bei den jeweiligen Problemen benötigt werden, um die genannten Gadgets zu durchqueren. Die Zugfolge im Entsperr-Gadget wurde zu zwei einzelnen Zugfolgen aufgeteilt – die Zugfolge zum Entsperren des Gadgets und die Zugfolge, die überprüft, ob das Gadget entsperrt wurde. ∗ bedeutet, dass die Anzahl variabel ist und von der Länge der Kante anhängt.

Interessant ist auch, dass fast alle Zugfolgen zum Durchqueren der Gadgets bis auf einen konstanten Faktor gleich lang sind. Tabelle 4.1 zeigt, wie viele Züge jeweils benötigt werden, um die einzelnen Gadgets zu durchqueren. Die einzigen beiden Ausnahmen sind die Kanten-Gadgets bei *AMC-R* und GAS. Diese hängen in unserer Konstruktion von der Länge der jeweiligen Kante ab.

4.2 Offene Fragen

Auch wenn wir die NP-Vollständigkeit von EdgeHop gezeigt haben gibt es noch einige offene Fragen. Bei den von uns gewählten Beispielen war es unproblematisch, dass der EdgeHop-Graph nicht planar war. Aber bei anderen Problemen ist es eventuell aufwändig oder unmöglich, ein Kreuzungs-Gadget zu konstruieren. Darum stellt sich die Frage, ob es möglich ist mit den Gadgets selbst eine Kreuzung zu konstruieren, so dass der EdgeHop-Graph planar ist, bzw. zu einem planaren Graphen umgeformt werden kann.

Ebenfalls offen ist, wie sich die Komplexität von EdgeHop ändert, wenn man die Regeln modifiziert:

- Wie ändert sich die Komplexität, wenn jede Kante beliebig oft genutzt werden kann?
- Macht es einen Unterschied, ob man mehrere *aktive* Steine mit eventuell eigenen Zielknoten hat?
- Ändert sich die Komplexität, wenn man statt einfachen Graphen andere Graphen (gerichtete Graphen, Graphen mit Mehrfachkanten, ...) zugrunde legt?

Wenn man zulässt, dass jede Kante beliebig oft genutzt werden kann, dann ist es eventuell möglich „Speicherbausteine“ zu konstruieren und hat damit die Möglichkeit Allquantoren zu modellieren. Damit könnte man eine Reduktion von QBF durchführen. Wenn man mehrere Zielsteine hat, dann ergibt sich ein Spiel, welches ähnlich zu Sokoban ist und damit ergibt sich eventuell eine andere Komplexität. Bei anderen Graphen kommt es auf die Graphklassen an. Gerichtete Graphen kann man eventuell mit Dioden-Gadgets in unserem Modell modellieren und Mehrfachkanten kann man konstruieren, in dem man jede „Mehrfachkante“ durch eine Kante-Knoten-Kante-Konstellation in unserem Modell ersetzt.

Bei unseren gewählten Beispiele bleiben auch noch einige Fragen offen. Eine natürliche Erweiterung von 1-Zug-Latrunculi wäre k-Zug-Latrunculi. Dies könnte man dann sogar erweitern zur Frage, ob aus einer beliebigen Spielsituation heraus gewonnen werden kann. Hier wäre es dann interessant zu sehen, ob diese Frage schwieriger zu beantworten ist als z. B. bei Schach oder ob sie genau so schwierig ist, obwohl ein einzelner Zug in Latrunculi mächtiger ist. Außerdem könnte man auch bei Latrunculi die Regeln modifizieren und dann die Komplexität der neuen Variante betrachten.

Bei AMC-R gibt es viele Möglichkeiten, das Problem zu ändern. Da Minecraft in seiner Reinform Turing-vollständig ist, kommt es hier ganz stark darauf an, wie man das Problem definiert. Es stellt sich z. B. die Frage, wie es sich mit der Komplexität unserer Variante verhält, wenn es dem Spieler möglich ist, Blöcke abzubauen und zu setzen. Oder ob es möglich ist, statt einer Tür, die von einem Schalter geöffnet wird, eine Situation zu simulieren, in der jede Tür nur durch einen zugehörigen Schlüssel geöffnet werden kann, den die Spielfigur vorher finden muss. Eine sehr natürliche Erweiterung unserer Fragestellung ist es auch, andere Blockarten zusätzlich zuzulassen. Auch wenn viele Blockarten nur das Aussehen unserer Gadgets ändern würden, kann es bei einigen Blockarten eventuell dazu führen, dass neue Spielmachaniken hinzu kommen.

Bei 2048^+ stellt sich die Frage, wie es mit einem *echten* 2048 aussieht, also der Frage, ob es möglich ist aus einer gegebenen Situation heraus eine Kachel mit dem Zielwert zu erzeugen, wenn alle neu generierten Kacheln Wert 2 oder 4 haben und die blockierenden Kacheln ebenfalls nur Zweierpotenzen (die kleiner als der Zielwert sind) als Werte haben. Eine andere Frage ist, ob es in unserer Variante einen NP-Algorithmus gibt, wenn neue Kacheln zwar beliebige Werte haben können, aber diese nicht zwingend verschieden sein müssen von Werten, die sich schon auf dem Feld befinden. An dieser Stelle ist unsere Argumentation, dass eine Zugfolge höchstens n^2 Züge haben kann ungültig.

Bei GAS bleibt die Frage nach der Komplexität ebenfalls offen. Wir haben zwar gesehen, dass das Problem NP-schwer ist, aber hier stellt sich die Frage, ob es nicht sogar PSPACE-schwer ist. Oder die Frage, ob es einen NP-Algorithmus gibt. Außerdem kann man auch bei GAS fragen, wie die Komplexität aussieht, wenn die Regeln etwas modifiziert werden. Wir haben eine Variante betrachtet, in der jedes Quadrat ein zugehöriges Zielfeld hatte. Ändert sich die Komplexität, wenn nicht zu jedem Quadrat ein gleichfarbiger Punkt auf dem Spielfeld sein muss?

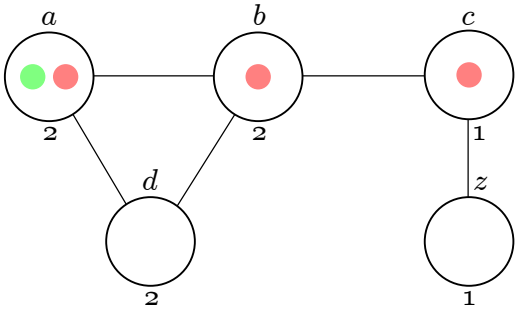
Ebenfalls stellt sich sowohl bei GAS als auch bei AMC-R die Frage, ob man die Kanten-Gadgets so gestalten kann, dass diese mit einer konstanten Anzahl an Zügen durchquert werden können. Oder ob es alternativ möglich ist, die Fragestellungen so zu erweitern, dass es möglich ist, solche Kanten-Gadgets zu konstruieren.

Weiter haben alle Probleme gemeinsam, dass man sich nicht nur mit der reinen Entscheidungsfrage beschäftigen kann, sondern sich die Frage nach Optimierung stellt. Wie schwierig ist es, die Länge der zum Erfolg führenden Zugfolgen zu minimieren? Auch wenn Tabelle 4.1 Hinweis darauf gibt, dass sich die Probleme nicht grundlegend unterscheiden, ist es dennoch sinnvoll, unsere definierten Probleme von diesem Standpunkt aus zu betrachten.

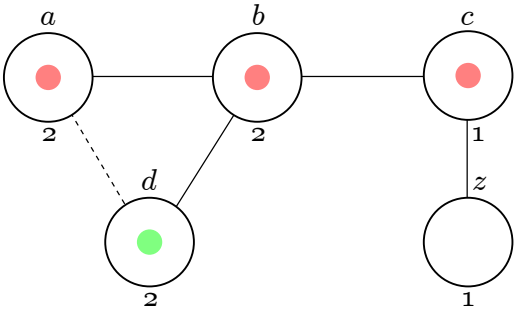
Anhang A

Anhang

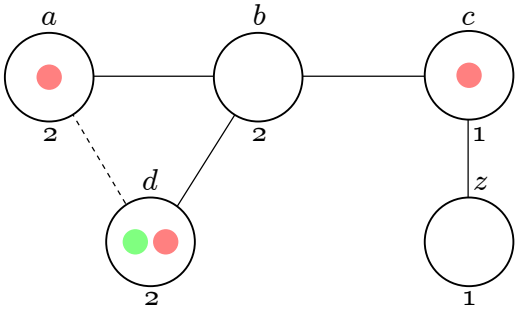
A.1 Zugfolge im Beispielgraphen



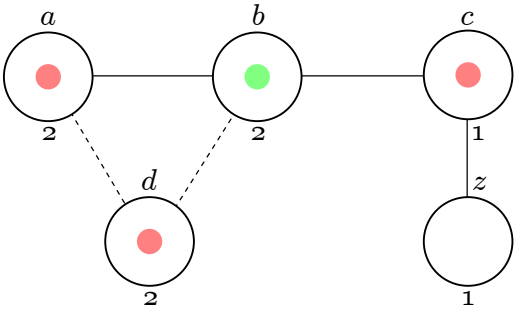
(a) Zug 1, $a \overset{p}{\leftarrow} b$. $X_1 = \emptyset$.



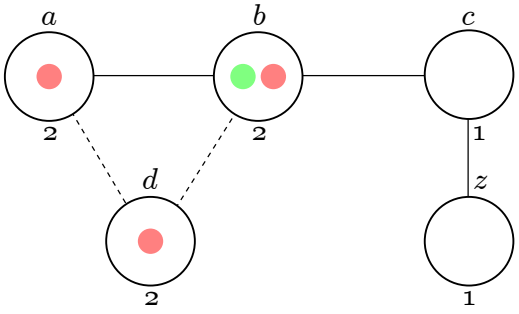
(b) Zug 2, $a \overset{a}{\rightarrow} d$. $X_2 = \{ad\}$.



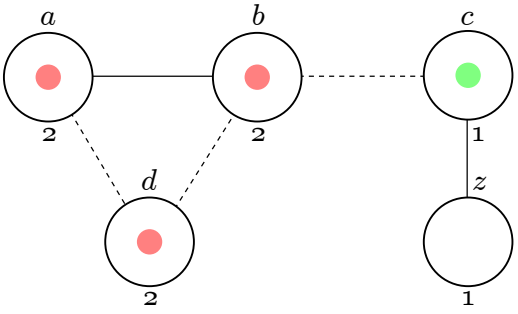
(c) Zug 3, $d \overset{p}{\leftarrow} b$. $X_3 = \{ad\}$.



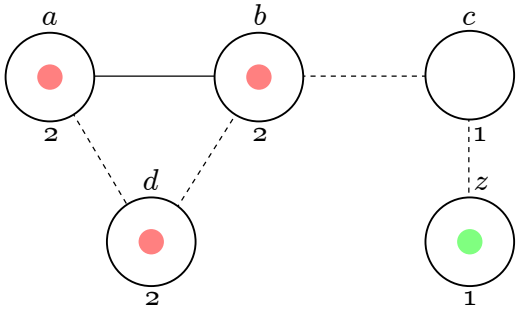
(d) Zug 4, $d \overset{a}{\rightarrow} b$. $X_4 = \{ad, db\}$.



(e) Zug 5, $b \overset{p}{\leftarrow} c$. $X_5 = \{ad, db\}$.



(f) Zug 6, $b \overset{a}{\rightarrow} c$. $X_6 = \{ad, db, bc\}$.

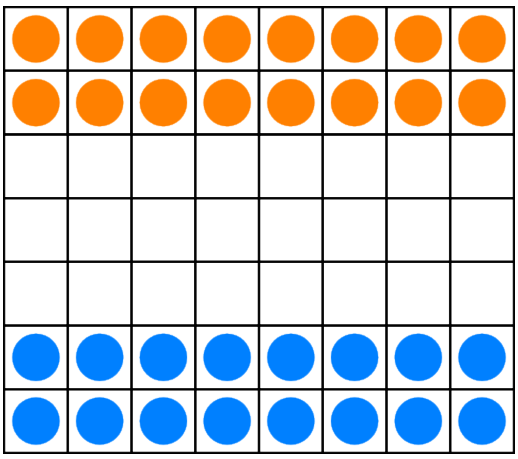


(g) Zug 7, $c \xrightarrow{a} z$. $X_7 = \{ad, db, bc, cz\}$.

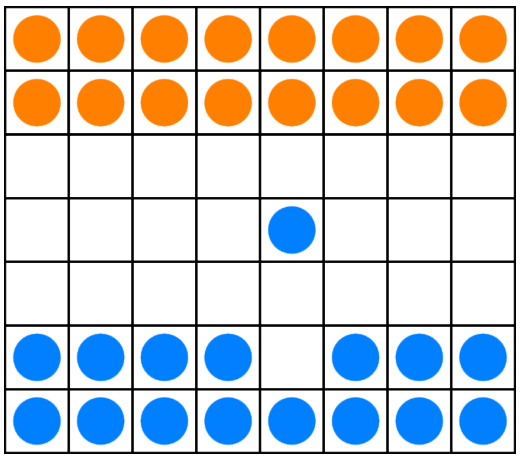
Abbildung A.1: Zugfolge, die das Spiel gewinnt. Zu jedem Bild ist die jeweilige Menge der benutzten Kanten angegeben. Benutzte Kanten werden als gestrichelte Linien dargestellt.

A.2 Beispieleröffnung einer Latrunculipartie

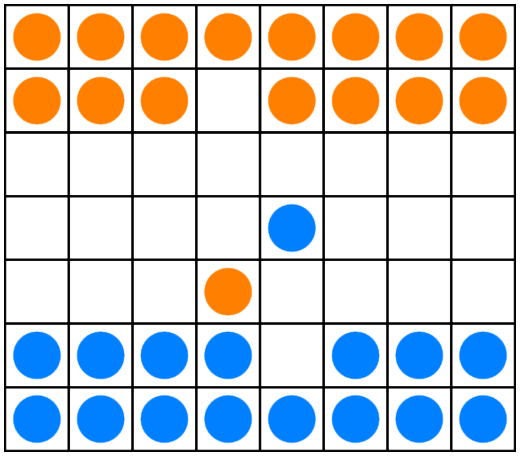
Im Folgenden werden die Felder wie üblich von links nach rechts mit a bis h benannt und von unten nach oben mit 1 bis 7.



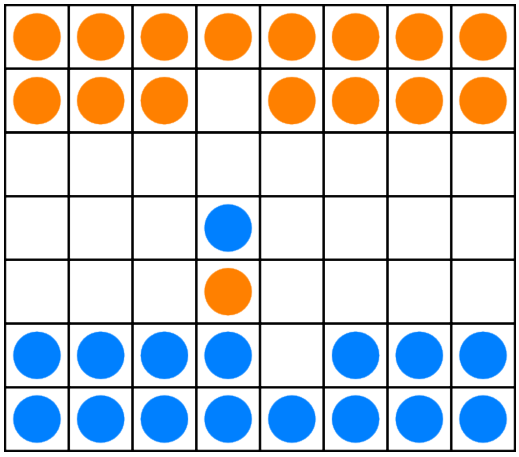
(a) Ausgangssituation. Das Feld hat eine Größe von 8×7.



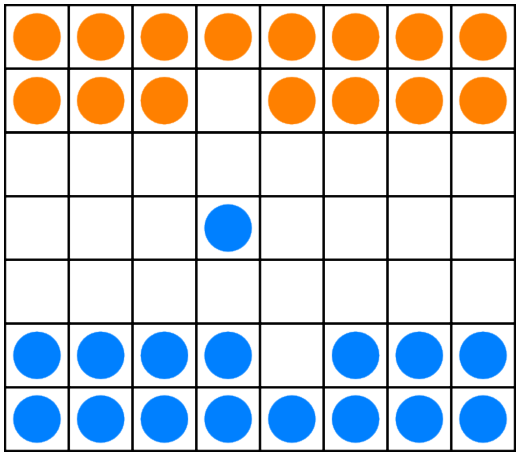
(b) Zug 1, *Blau* zieht $e2$ nach $e4$.



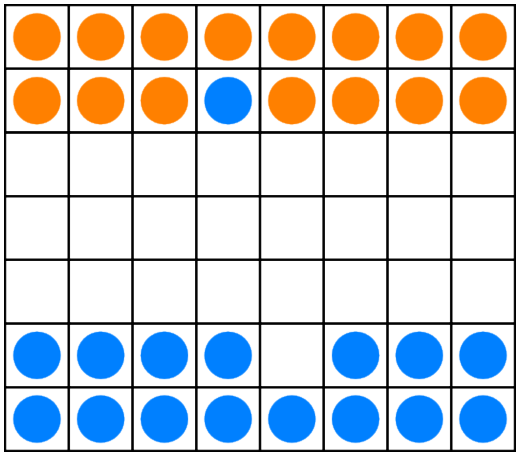
(c) Zug 2, *Orange* zieht $d6$ nach $d3$.



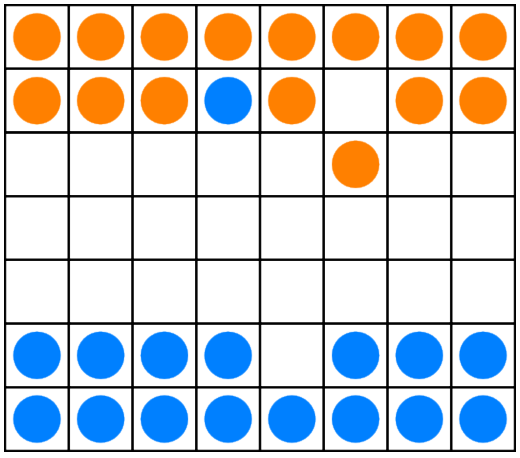
(d) Zug 3, *Blau* zieht $e4$ nach $d4$.



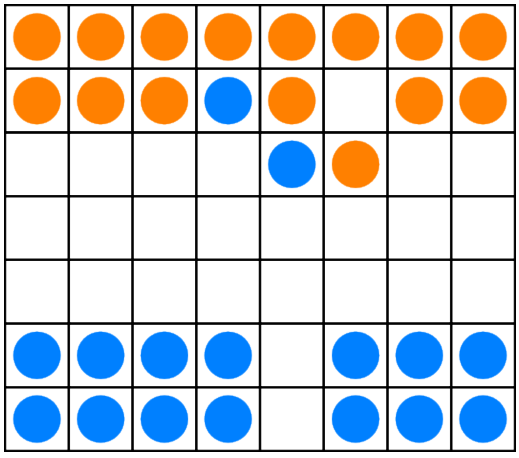
(e) Zug 3, *Blau* erobert $d3$, da $d3$ auf zwei gegenüberliegenden Seiten ($d2$ und $d4$) umstellt war.



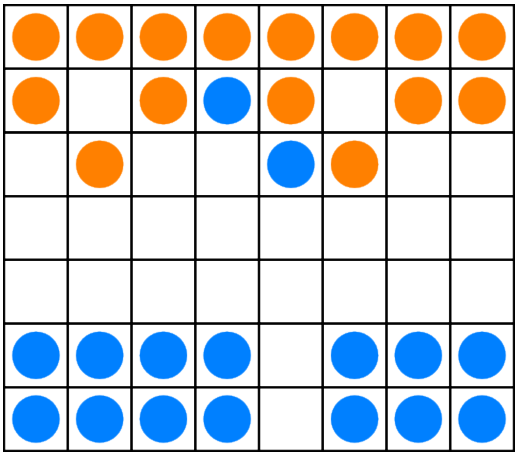
(f) Zug 3 (Teilzug 2), *Blau* zieht $d4$ nach $d6$. Obwohl $d6$ von zwei Seiten umstellt ist, wird der Stein nicht erobert, da *Blau* selbst zwischen die beiden orangefarbenen Steine gezogen ist.



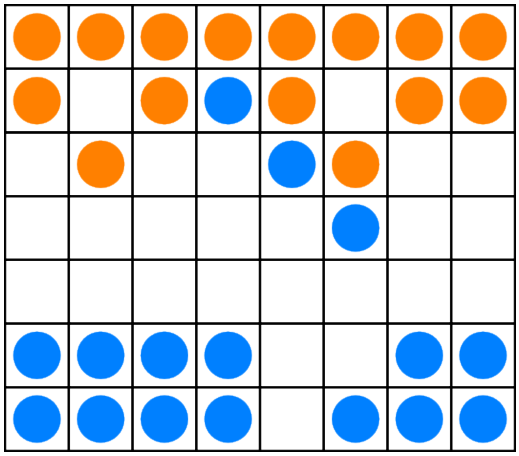
(g) Zug 4, *Orange* zieht $f6$ nach $f5$.



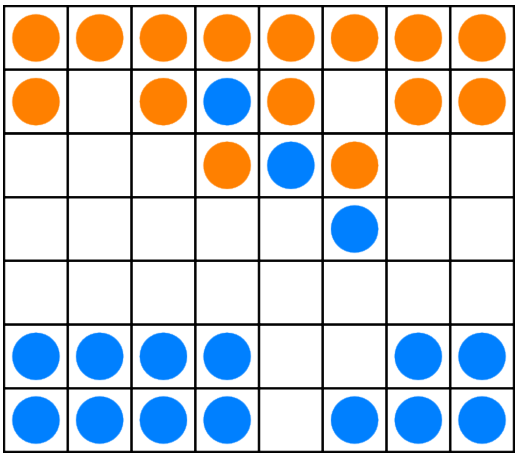
(h) Zug 5, *Blau* zieht $e1$ nach $e5$.



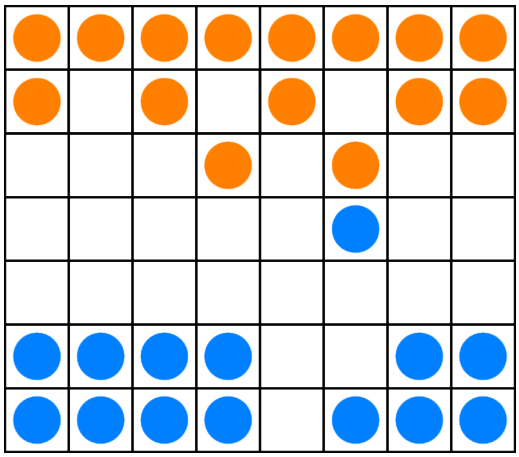
(i) Zug 6, *Orange* zieht $b6$ nach $b5$.



(j) Zug 7, *Blau* zieht $f2$ nach $f4$.



(k) Zug 8, *Orange* zieht $b5$ nach $d5$.

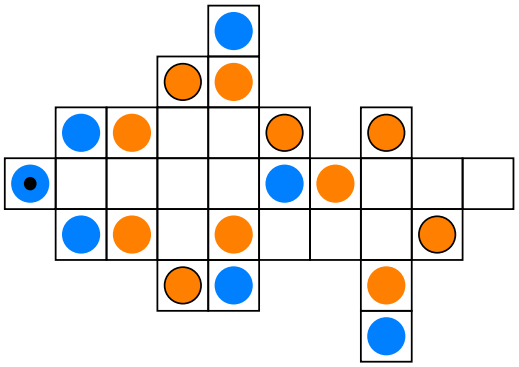


(l) Zug 8, *Orange* erobert $d6$ und $e5$, da sowohl $d6$ als auch $e5$ auf je zwei gegenüberliegenden Seiten ($d5$ und $d7$ bzw. $d5$ und $f5$) umstellt war.

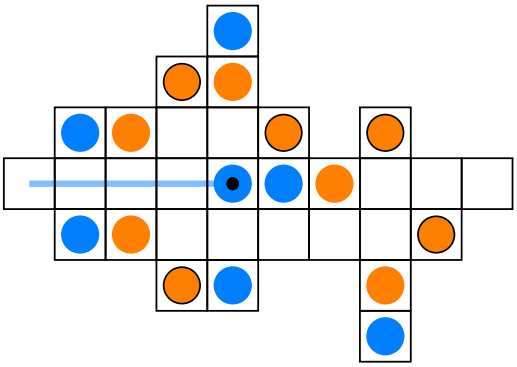
Abbildung A.2: Beispieleröffnung einer Latrunculipartie, in der *Orange* einen Stein und *Blau* zwei Steine verliert.

A.3 Teilzugfolge im Dioden-Gadget (Latrunculi)

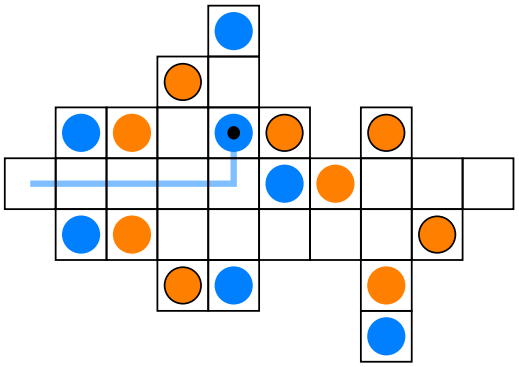
Im Folgenden werden die Felder wie üblich von links nach rechts mit a bis j benannt und von unten nach oben mit 1 bis 7. Zusätzlich zur angesprochenen Notation wird der aktive Stein durch einen schwarzen Kreis in der Mitte gekennzeichnet.



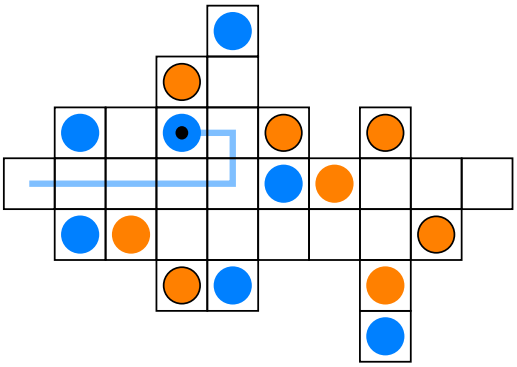
(a) Ausgangssituation. Der aktive Stein startet auf $a4$



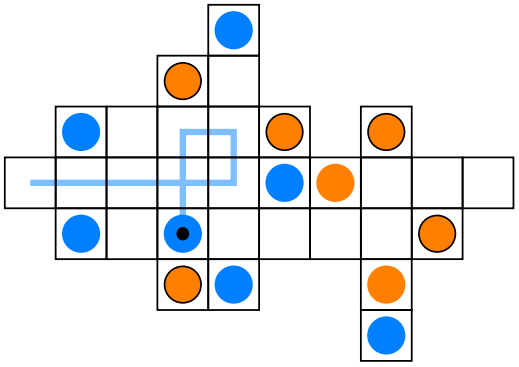
(b) Teilzug 1, $e4$ (erobert $e3$).



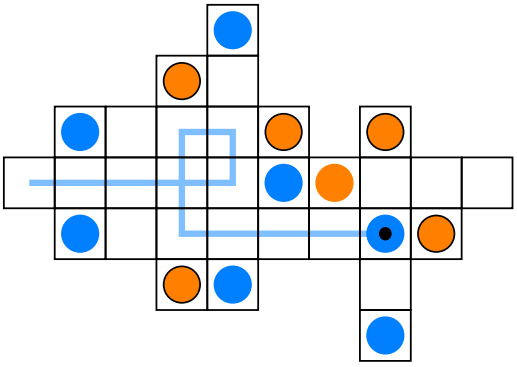
(c) Teilzug 2, $e5$ (erobert $e6$).



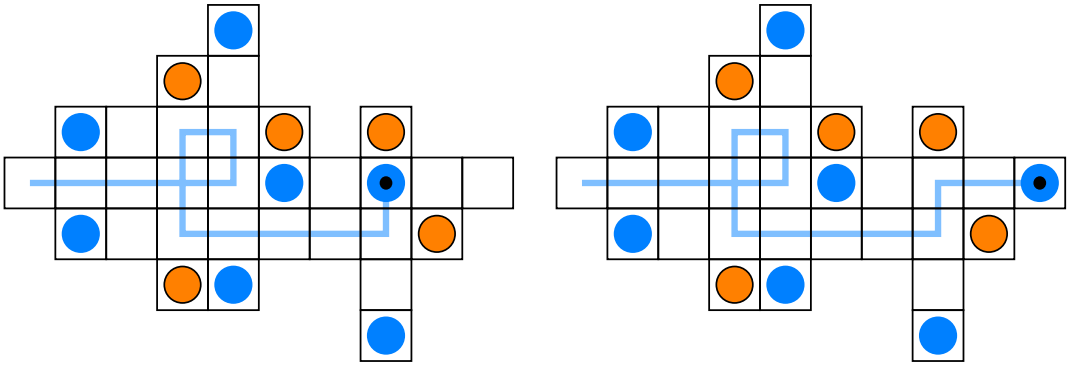
(d) Teilzug 3, $d5$ (erobert $c5$).



(e) Teilzug 4, $d3$ (erobert $c3$).



(f) Teilzug 5, $h3$ (erobert $h2$).

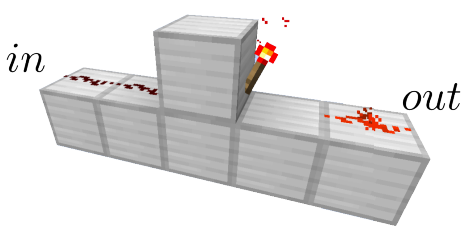


(g) Teilzug 6, $h4$ (erobert $g4$). **(h)** Teilzug 7, $j4$.

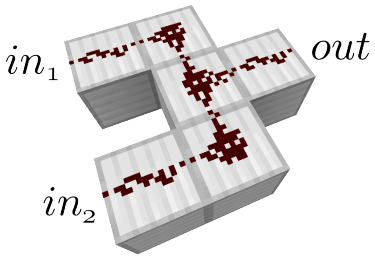
Abbildung A.3: Eine mögliche Teilzugfolge von $a4$ nach $j4$.

A.4 Logik-Gatter (Minecraft)

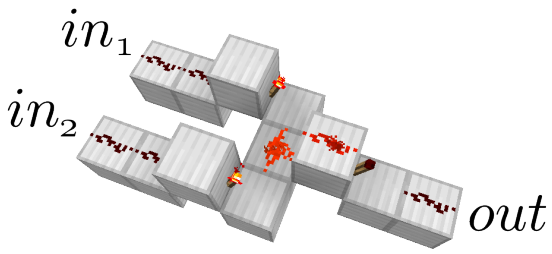
Die folgenden drei Abbildungen zeigen wie man die grundlegenden Logik-Gatter in Minecraft konstruieren kann. Die Eisenblöcke können durch beliebige andere Blöcke ersetzt werden – diese sind nur Teil des Gatters, da Redstone-Staub und -Fackeln nur auf (bzw. an) Blöcke gesetzt werden können.



(a) Ein NICHT-Gatter.



(b) Ein ODER-Gatter

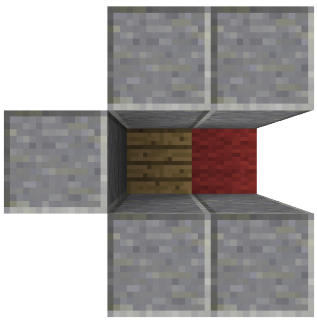


(c) Ein UND-Gatter.

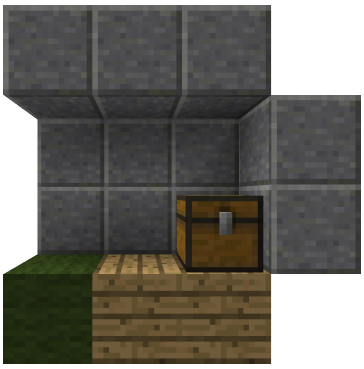
Abbildung A.4: Grundlegende Logik-Gatter konstruiert aus Minecraft-Redstone.

A.5 EdgeHop-Kanten (Minecraft)

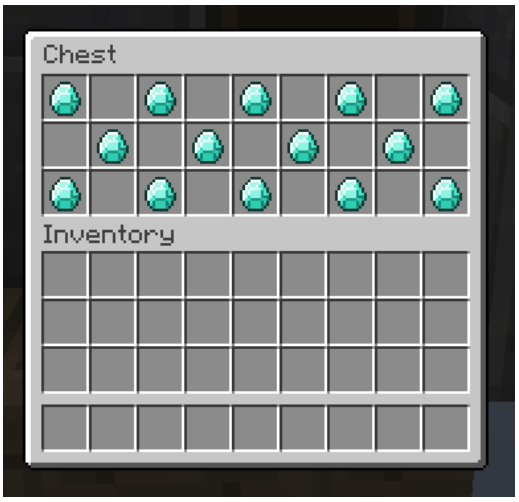
Die folgenden Abbildungen zeigen wie man Start- und Ziel-Gadget und Kanten aus Minecraft-Blöcken bauen kann. Wie üblich fehlen die jeweils vorderen Wände oder Decken der Konstruktionen, damit das Innenleben sichbar ist.



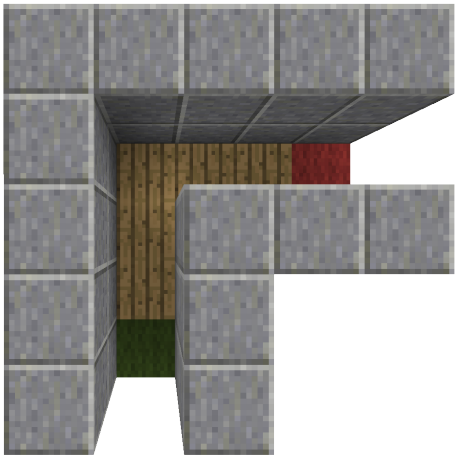
(a) Start-Gadget (Draufsicht).



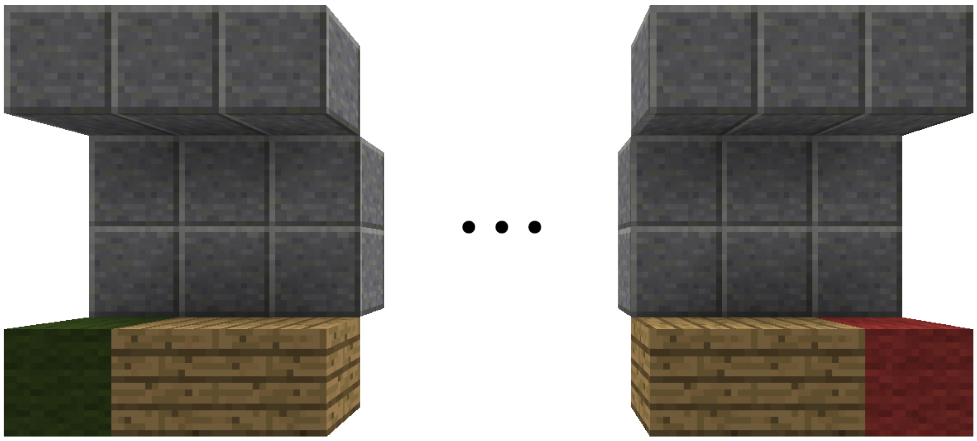
(b) Ziel-Gadget (Seitenansicht).



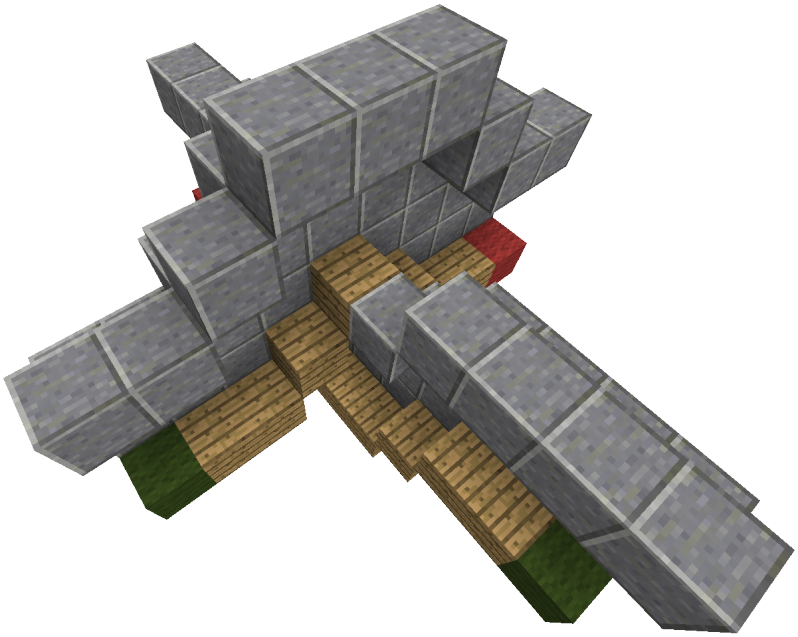
(c) Ziel-Gadget (Inhalt der Kiste).



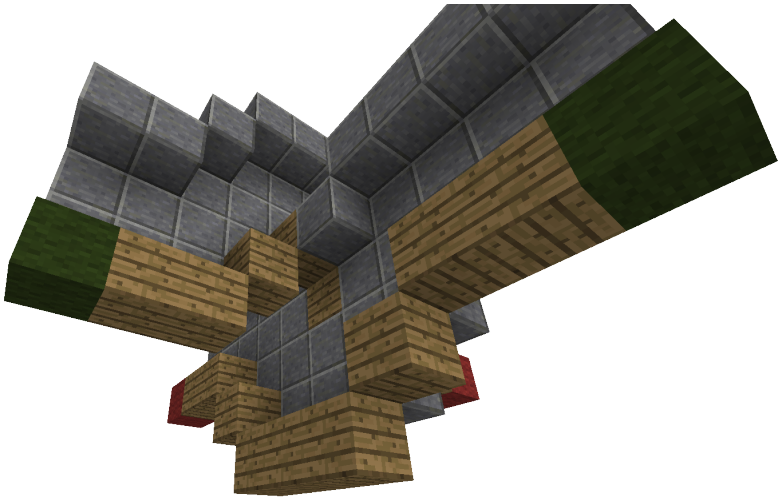
(d) Knick-Gadget (Draufsicht).



(e) Kanten-Gadget (Seitenansicht).



(f) Kreuzungs-Gadget (Sicht von oben).



(g) Kreuzungs-Gadget (Sicht von unten).

Abbildung A.5: Die einzelnen Gadgets um die Kanten des EdgeHop-Graphen zu modellieren, konstruiert aus Minecraft-Blöcken.

A.6 Alternatives Entsperr-Gadget (Minecraft)

Die folgenden Abbildungen zeigen wie man das Entsperr-Gadget ohne *„Lever“*- und *„Iron door“*-Blöcke gestalten kann. Dieses Gadget macht sich zu nutze, dass der *„Sand“*-Block in Minecraft einer der wenigen Blöcke ist, der Gravitation nicht ignoriert und nach unten fällt, wenn unter ihm kein Block ist. Weiter macht sich das Gadget zu nutze, dass *„Torch“*-Blöcke zerstört werden, wenn sich der Block, an den sie gesetzt sind, bewegt.

Dieses Gadget arbeitet wie folgt: Damit der Spieler sich vom hellblauen zum gelben Block bewegen kann, muss er den *„Packed ice“*-Block zerstören. Dadurch wird der adjazente *„Torch“*-Block zerstört und der darüber liegende *„Sand“*-Block fällt herunter. Damit wird der zum diesem Block adjazente *„Torch“*-Block zerstört und der darüber liegende *„Sand“*-Block fällt schlussendlich hinunter. Damit kann die Spielfigur vom grünen zum roten Block gehen, da der *„Sand“*-Block dafür sorgt, dass die Spielfigur keinen zu hohen Sprung mehr ausführen müsste.

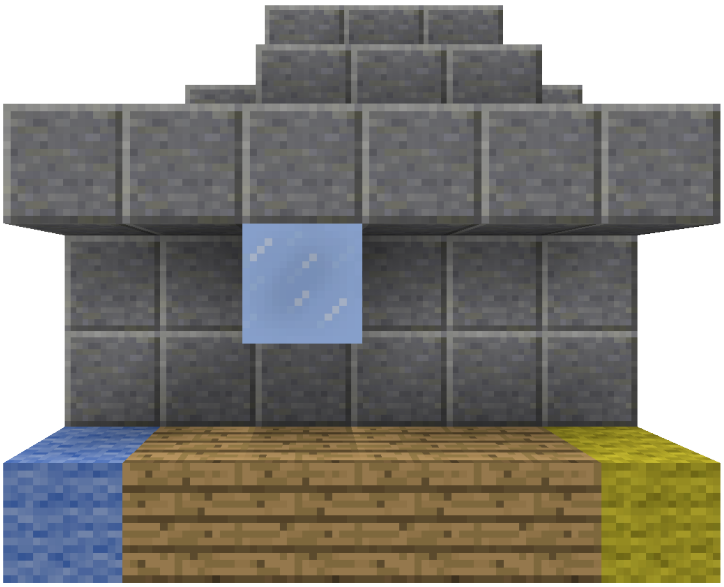
Da wir den Adventure-Modus betrachten, kann der Spieler keine beliebigen Blöcke zerstören. Damit er den *„Packed ice“*-Block zerstören kann müssen wir der Spielfigur ein beliebiges Item ohne Haltbarkeit (bspw. einen *„Stick“*) geben, welches einen CanDestroy:[“minecraft:packed_ice”]-Tag besitzt.

(a) „Sand“-Block.

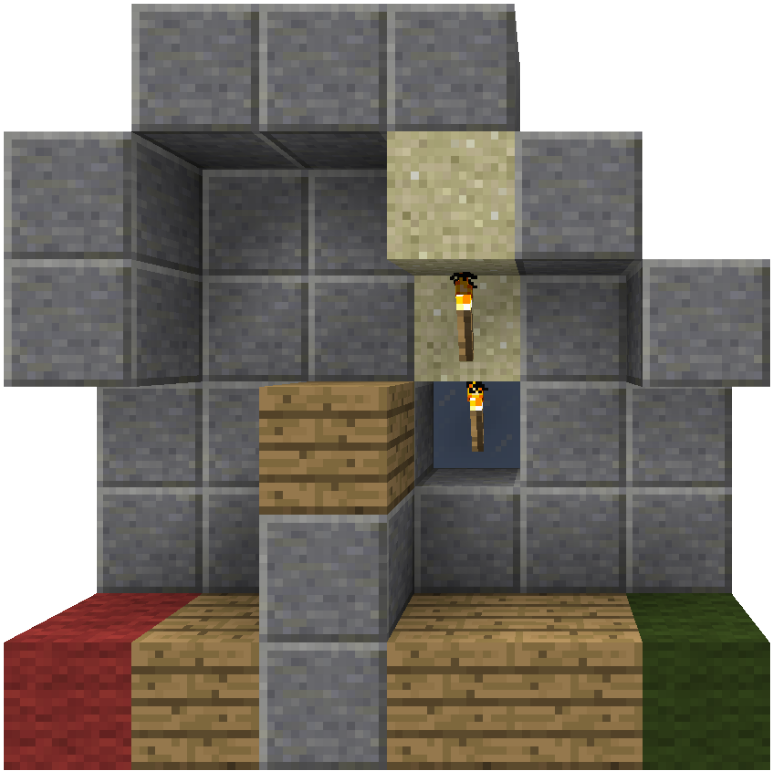
(b) „Torch“-Block.

(c) „Packed ice“-Block.

(d) „Stick“ mit einem „CanDestroy“-Tag.



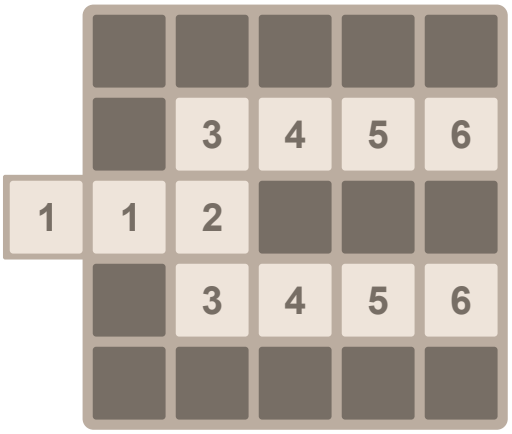
(e) Alternatives Entsperr-Gadget, entsperrender Ein- und Ausgang (Seitenansicht).



(f) Alternatives Entsperr-Gadget, überprüfender Ein- und Ausgang (Seitenansicht).

Abbildung A.6: Alternative Möglichkeit, das Entsperr-Gadget zu modellieren. Hierbei werden keine *„Lever“*-Blöcke oder *„Iron door“*-Blöcke verwendet.

A.7 Zugfolge im Verzweigungs-Gadget (2048)

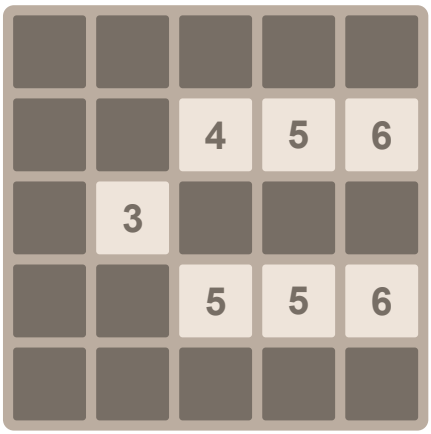


(a) Ausgangssituation.

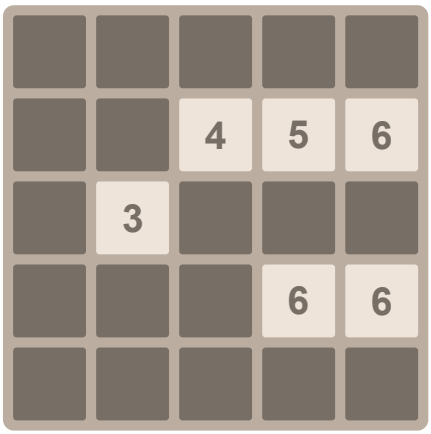
(b) Nach dem ersten Zug (→). Das aktive Feld ist nun $(0, 2)$.

(c) Nach dem zweiten Zug (→). Das aktive Feld ist nun $(1, 2)$.

(d) Nach dem dritten Zug (↓). Das aktive Feld ist nun $(1, 1)$.



(e) Nach dem vierten Zug (→). Das aktive Feld ist nun $(2, 1)$.

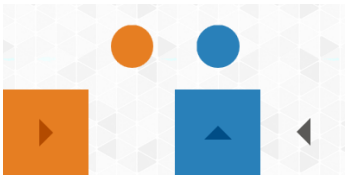


(f) Nach dem fünften Zug (→). Das aktive Feld ist nun $(3, 1)$.

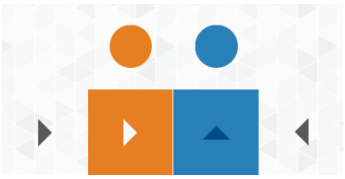
(g) Nach dem sechsten Zug (→). Das aktive Feld ist nun $(4, 1)$.

Abbildung A.7: Eine Folge im Verzweigugns-Gadget, die damit endet, dass $(4, 1)$ das aktive Feld ist.

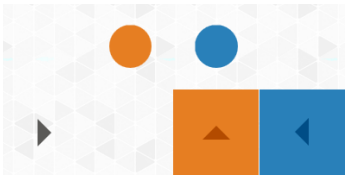
A.8 Zugfolge für Level 9 (Game about Squares)



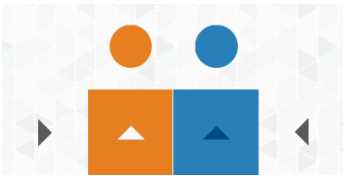
(a) Ausgangssituation.



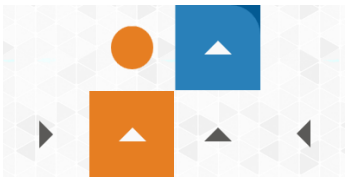
(b) Zug 1, Verschieben des orangefarbenen Quadrates.



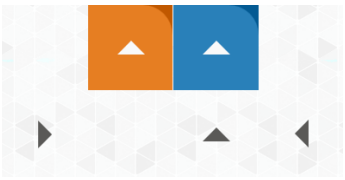
(c) Zug 2, Verschieben des orangefarbenen Quadrates.



(d) Zug 3, Verschieben des blauen Quadrates.



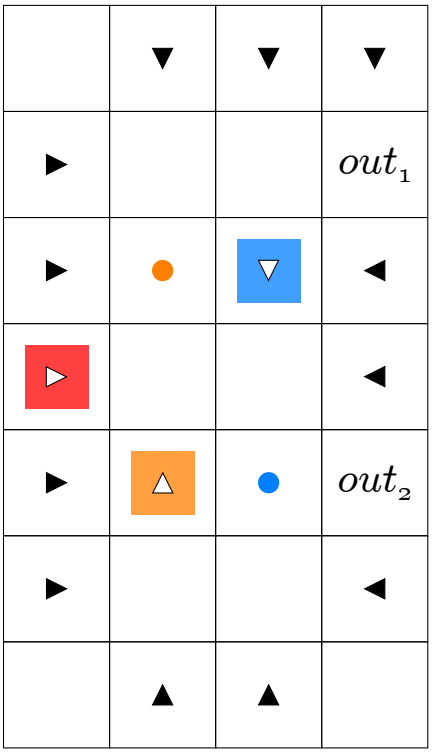
(e) Zug 4, Verschieben des blauen Quadrates.



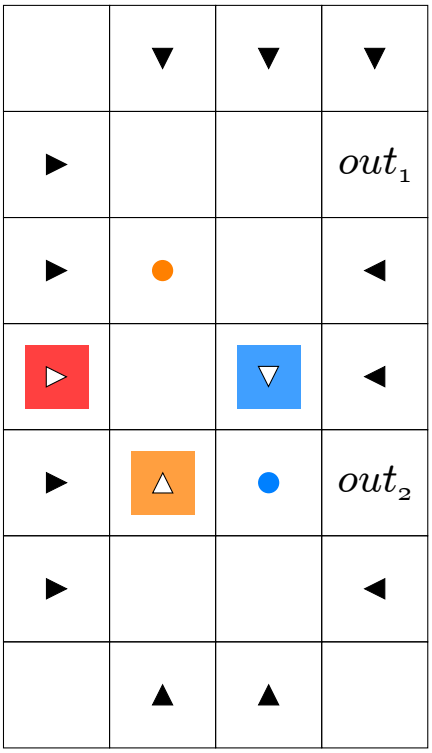
(f) Zug 3, Verschieben des orangefarbenen Quadrates.

Abbildung A.8: Eine Folge von Zügen, die zur Lösung von Level 9 aus *Game about Squares* führt.

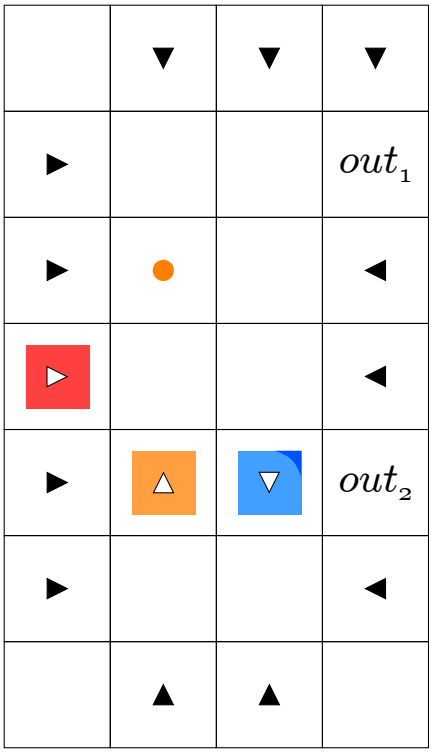
A.9 Zugfolge für das Verzweigungs-Gadget (Game about Squares)



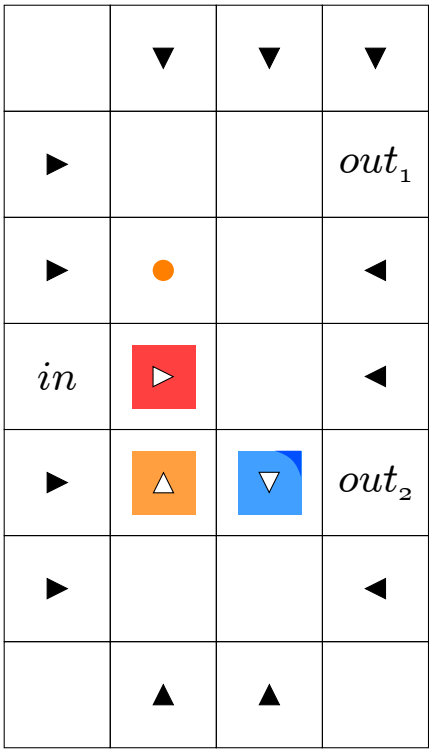
(a) Ausgangssituation.



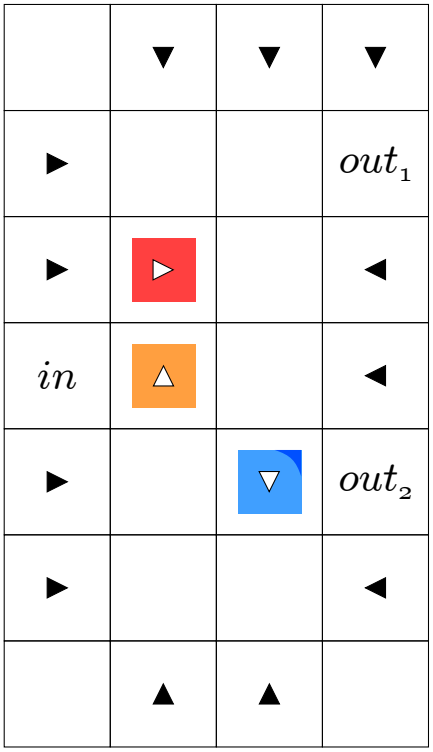
(b) Zug 1, Verschieben des blauen Quadrates.



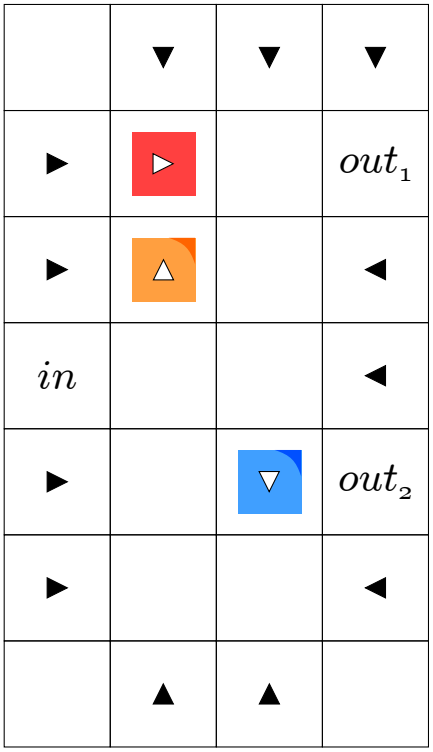
(c) Zug 2, Verschieben des blauen Quadrates.



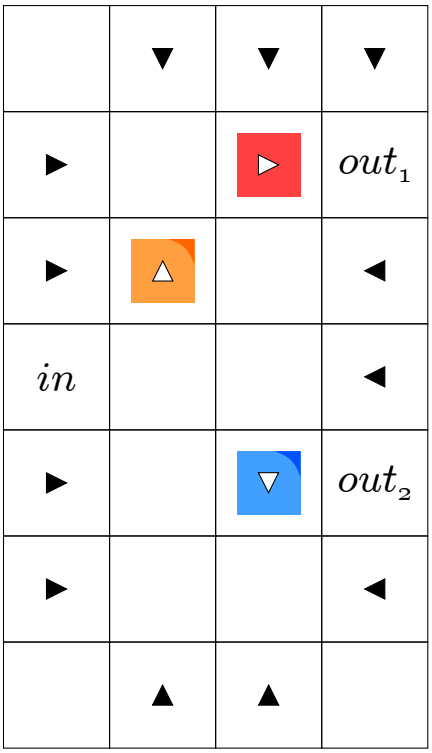
(d) Zug 3, Verschieben des roten Quadrates.



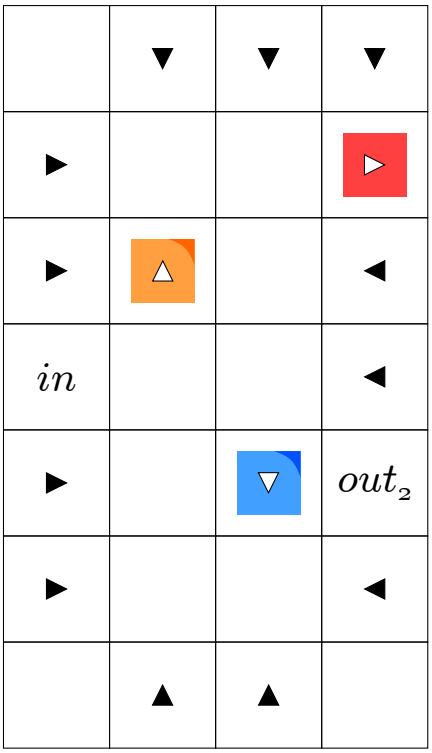
(e) Zug 4, Verschieben des orangefarbenen Quadrates.



(f) Zug 5, Verschieben des orangefarbenen Quadrates.



(g) Zug 6, Verschieben des roten Quadrates.



(h) Zug 7, Verschieben des roten Quadrates.

Abbildung A.9: Eine Folge von Zügen, die das rote Quadrat von Feld *in* zu Feld *out* führt.

Anhang B

Definitionen

B.1 Allgemeine Definitionen

In diesem Abschnitt folgen einige Definitionen zu Begriffen, die in der Arbeit auftauchen. Für weitere Definitionen werden auf die Bücher von R. Diestel [A8] und Ingo Wegener [A26] verwiesen.

Definition (Einfacher Graph)**.** Ein *einfacher Graph* ist ein Paar $G = (V, E)$ mit $V \subset \mathbb{N}$ und $E \subset \{\{u, v\} : u, v \in V, u \neq v\}$. Wir nennen V die Menge der *Knoten* und E die Menge der *Kanten* des Graphen. Wenn es nicht eindeutig ist, zu welchem Graphen die Mengen gehören schreiben wir auch $V(G)$ und $E(G)$. Zwei Knoten $u, v \in V$ heißen *adjazent* wenn gilt $\{u, v\} \in E$. Wir schreiben für die Kante $\{u, v\}$ auch kurz uv. Wir nennen u *Nachbar* von v (und umgekehrt).

Definition (Gerichteter Graph)**.** Ein *gerichteter Graph* ist ein Paar $G = (V, E)$ mit $V \subset \mathbb{N}$ und $E \subset \{(u, v) : u, v \in V, u \neq v\}$. Wie auch beim einfachen Graphen nennen wir V die Menge der *Knoten* und E die Menge der *Kanten* des Graphen. Ebenfalls verwenden wir die Schreibweisen $V(G)$ und $E(G)$, wenn es nicht eindeutig ist, zu welchem Graphen die jeweiligen Mengen gehören. Zwei Knoten $u, v \in V$ heißen *adjazent* wenn gilt $(u, v) \in E$. Wir nennen u *Vorgänger* von v und v *Nachfolger* von u.

Definition (Valenz)**.** Die *Valenz* eines Knotens u in einem ungerichteten Graphen G gibt an, wie viele Nachbarn dieser Knoten hat. Sie ist definiert als $d_G(u) = |\{v \in V(G) : uv \in E(G)\}|$.

Für einen gerichteten Graphen H unterschieden wir zwischen *Eingangsvalenz* (Anzahl der Vorgänger) und *Ausgangsvalenz* (Anzahl der Nachfolger). Wir definieren die Eingangsvalenz als $d_H^-(v) = |\{u \in V(H) : (u, v) \in E(H)\}|$ und die Ausgangsvalenz als $d_H^+(v) = |\{w \in V(H) : (v, w) \in E(H)\}|$.

Definition (Pfad)**.** Ein *Pfad* von u nach v ist eine Folge von paarweise verschiedenen Knoten $(x_0, ..., x_k)$ mit $x_0 = u$, $x_k = v$ und $x_i x_{i+1} \in E$ (bzw. $(x_i, x_{i+1}) \in E$) für $0 \leq i < k$.

Definition (Zusammenhang)**.** Ein Graph $G = (V, E)$ heißt zusammenhängend, wenn für alle $u, v \in V$ ein Pfad von u nach v existiert.

Definition (NP)**.** *NP* ist die Komplexitätsklasse der Entscheidungsprobleme, die nichtdeterministisch in Polynomzeit lösbar sind, bzw. für die eine gegebene Lösung in Polynomzeit überprüft werden kann.

Definition (PSPACE)**.** *PSPACE* ist die Komplexitätsklasse der Entscheidungsprobleme, die (deterministisch) in polynomiellem Platz lösbar sind.

Definition ($\mathcal{C}$-schwer)**.** Sei $\mathcal{C}$ eine Komplexitätsklasse. Ein Problem $\mathcal{P}$ heißt *$\mathcal{C}$-schwer*, wenn für jedes $\mathcal{P}' \in \mathcal{C}$ gilt, dass sich dieses (in Polynomzeit) auf $\mathcal{P}$ reduzieren lässt, also $\mathcal{P}' \leq_p \mathcal{P}$.

Definition ($\mathcal{C}$-vollständig)**.** Sei $\mathcal{C}$ eine Komplexitätsklasse. Ein Problem $\mathcal{P}$ heißt *$\mathcal{C}$-vollständig*, wenn $\mathcal{P} \in \mathcal{C}$ und $\mathcal{P}$ zusätzlich $\mathcal{C}$-schwer ist.

Anmerkung. Für die üblichen Komplexitätsklassen gilt:
$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXPTIME$

B.2 Entscheidungsprobleme

Problem: 1-Zug-Latrunculi
Eingabe: $n, m \in \mathbb{N}$ Seitenlängen des Spielfelds, $B, O \subset \{1, ..., n\} \times \{1, ..., m\}$ Positionen der blauen bzw. orangefarbenen Steine (mit $B \cap O = \emptyset$), $(a_1, a_2) \in B$ Position des aktiven Steins und $(z_1, z_2) \in O$ Position des Zielsteins.
Fragestellung: Gibt es eine Zugfolge, die den Zielstein erobert und nur mit dem aktiven Stein zieht?

Problem: 2048^+
Eingabe: $n, m \in \mathbb{N}$ mit $n \geq m$ Seitenlängen des Spielfelds und $f : \{1, ..., n\} \times \{1, ..., m\} \to \mathbb{N}_0$ Funktion die angibt, ob auf einem Feld eine Kachel liegt und welchen Wert sie hat.
Fragestellung: Kann der Spieler eine Kachel mit Wert 2^{3n-1} erreichen, wenn neue Kacheln mit beliebigen Werten $w < 2^{3n-1}$ generiert werden können?

Problem: 3-SAT
Eingabe: Boolesche Formel ϕ in 3-KNF.
Fragestellung: Gibt es eine Belegung α der Variablen, so dass $\alpha(\phi) = 1$?

Problem: Adventure-Modus-Minecraft (ohne Redstone)
Eingabe: Eine beliebige Minecraft-Welt im Adventure-Modus, ohne Redstone.
Fragestellung: Kann die Spielfigur eine mit Diamanten gefüllte Kiste erreichen?

Problem: EdgeHop
Eingabe: 5-Tupel $\mathcal{EH} = (G, k_G, a, z, P_G)$, mit $G = (V, E)$ einfacher Graph, $k_G : V \to \mathbb{N}$, $a, z \in V$ und P_G Multimenge über V.
Fragestellung: Gibt es eine gültige Zugfolge von a nach z?

Problem: GaS
Eingabe: 5-Tupel $\mathcal{GAS} = (C, r, p, z, d)$ mit $C \subset \mathbb{N}$ endlich, $r : C \to \{\triangle, \triangleright, \triangledown, \triangleleft\}$, $p, z : C \to \mathbb{Z}^2$ und $d : \{\blacktriangle, \blacktriangleright, \blacktriangledown, \blacktriangleleft\} \to \mathcal{P}(\mathbb{Z}^2)$.
Fragestellung: Gibt es eine Folge von Zügen, so dass alle Zielfelder von Quadraten entsprechender Farbe besetzt sind?

Problem: Hamiltonkreis, gerichtet
Eingabe: Gerichteter Graph $G = (V, E)$.
Fragestellung: Gibt es eine Folge $(x_1, ..., x_{|V|})$ von paarweise verschiedenen Knoten, so dass $(x_i, x_{i+1}) \in E$ für $1 \leq i < |V|$ und $(x_{|V|}, x_1) \in V$?

Problem: QBF
Eingabe: Quantifizierte Boolesche Formel ϕ.
Fragestellung: Ist ϕ wahr?

Literatur

[A1] Aaron B. Adcock u. a. „Zig-Zag Numberlink is NP-Complete“. In: *CoRR* abs/1410.5845 (2014). URL: http://arxiv.org/abs/1410.5845.

[A2] Greg Aloupis u. a. „Classic Nintendo Games Are (Computationally) Hard“. In: *Fun with Algorithms - 7th International Conference, FUN 2014, Lipari Island, Sicily, Italy, July 1-3, 2014. Proceedings*. Hrsg. von Alfredo Ferro, Fabrizio Luccio und Peter Widmayer. Bd. 8496. Lecture Notes in Computer Science. Springer, 2014, S. 40–51. DOI: 10.1007/978-3-319-07890-8_4. URL: http://dx.doi.org/10.1007/978-3-319-07890-8_4.

[A3] Robert Charles Bell. *The Boardgame Book*. London: Marshall Cavendish Ltd., 1979.

[A4] Édouard Bonnet. „Résultats Positifs et Négatifs en Approximation et Complexité Paramétrée“. Diss. Université Paris-Dauphine, Nov. 2014.

[A5] Stephen A. Cook. „The Complexity of Theorem-proving Procedures“. In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. STOC '71. Shaker Heights, Ohio, USA: ACM, 1971, S. 151–158. DOI: 10.1145/800157.805047. URL: http://doi.acm.org/10.1145/800157.805047.

[A6] Erik D. Demaine und Robert A. Hearn. „Playing Games with Algorithms: Algorithmic Combinatorial Game Theory“. In: *Games of No Chance 3*. Hrsg. von Michael H. Albert und Richard J. Nowakowski. 3, 2005, Banff, Alberta, Can.: Mathematical Sciences Research Institute Publications, 2009, S. 3–56. ISBN: 978-0-521-86134-2.

[A7] Dariusz Dereniowski. „Phutball is PSPACE-hard“. In: *Theoretical Computer Science* 411.44-46 (2010), S. 3971–3978. DOI: 10.1016/j.tcs.2010.08.019.

[A8] R. Diestel. *Graphentheorie*. Springer, 2010. ISBN: 978-3-642-14911-5.

[A9] Edward Falkener. *Games ancient and oriental and how to play them – being the games of the ancient Egyptians, the Hiera Gramme of the Greeks, the Ludus Latrunculorum of the Romans and the oriental games of chess, draughts, backgammon and magic squares*. London: Longmans, 1892.

[A10] Henning Fernau u. a. „On the parameterized complexity of the generalized rush hour puzzle“. In: *Proceedings of the 15th Canadian Conference on Computational Geometry, CCCG'03, Halifax, Canada, August 11-13, 2003*. 2003, S. 6–9. URL: http://www.cccg.ca/proceedings/2003/22.pdf.

[A11] Gary William Flake und Eric B. Baum. „Rush Hour is PSPACE-complete, or "Why you should generously tip parking lot attendants"“. In: *Theor. Comput. Sci.* 270.1-2 (2002), S. 895–911. DOI: 10.1016/S0304-3975(01)00173-6. URL: http://dx.doi.org/10.1016/S0304-3975(01)00173-6.

[A12] Masters Games. *Ludus Latrunculorum or Latrunculi – Rules for the Classic Roman Game*. 2011. URL: http://www.mastersgames.com/rules/Ludus-Latrunculorum-Rules.pdf (besucht am 02. 01. 2015).

[A13] Moritz Gobbert und Henning Fernau. „Latrunculi – on the complexity of Roman chess“. In: *Sixth Workshop on Non-Classical Models for Automata and Applications – NCMA 2014, Kassel, Germany, July 28-29, 2014. Proceedings*. 2014, S. 115–130.

[A14] Robert A. Hearn und Erik D. Demaine. *Games, puzzles and computation*. A K Peters, 2009. ISBN: 978-1-56881-322-6.

[A15] Robert A. Hearn und Erik D. Demaine. „PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation“. In: *Theoretical Computer Science* 343 (2005), S. 72–96.

[A16] G. Kendall, A. Parkes und K. Spoerer. „A Survey of NP-Complete Puzzles“. In: *International Computer Games Association Journal (ICGA)* 31 (2008), S. 13–34.

[A17] Wally J. Kowalski. *Roman Board Games*. 2000. URL: http://ablemedia.com/ctcweb/showcase/boardgameslat2.html (besucht am 06. 05. 2014).

[A18] Martin Kreuzer und Stefan Kühling. *Logik für Informatiker*. Pearson Studium, 2006. ISBN: 978-3-8273-7215-4.

[A19] Jens Maßberg. „The „Game about Squares“ is NP-hard“. In: *CoRR* abs/1408.3645 (2014). URL: http://arxiv.org/abs/1408.3645.

[A20] Rahul Mehta. „2048 is (PSPACE) Hard, but Sometimes Easy“. In: *CoRR* abs/1408.6315 (2014). URL: http://arxiv.org/abs/1408.6315.

[A21] Katharina Uebel und Peter Buri. *Römische Spiele – So spielten die alten Römer*. second. Euskirchen: Regionalia Verlag, 2011.

[A22] Paul Rendell. „A Universal Turing Machine in Conway's Game of Life“. In: *2011 International Conference on High Performance Computing & Simulation, HPCS 2012, Istanbul, Turkey, July 4-8, 2011*. Hrsg. von Waleed W. Smari und John P. McIntire. IEEE, 2011, S. 764–772. DOI: 10.1109/HPCSim.2011.5999906. URL: http://dx.doi.org/10.1109/HPCSim.2011.5999906.

[A23] J. M. Robson. „N by N Checkers is Exptime Complete“. In: *SIAM J. Comput.* 13.2 (1984), S. 252–267. DOI: 10.1137/0213018. URL: http://dx.doi.org/10.1137/0213018.

[A24] Ulrich Schädler. „Latrunculi – ein verlorenes strategisches Brettspiel der Römer“. In: *HOMO LUDENS – Der spielende Mensch* IV (1994), S. 47–67.

[A25] Giovanni Viglietta. „Lemmings Is PSPACE-Complete“. In: *Fun with Algorithms - 7th International Conference, FUN 2014, Lipari Island, Sicily, Italy, July 1-3, 2014. Proceedings*. Hrsg. von Alfredo Ferro, Fabrizio Luccio und Peter Widmayer. Bd. 8496. Lecture Notes in Computer Science. Springer, 2014, S. 340–351. DOI: 10.1007/978-3-319-07890-8_29. URL: http://dx.doi.org/10.1007/978-3-319-07890-8_29.

[A26] Ingo Wegener. *Komplexitätstheorie: Grenzen der Effizienz von Algorithmen*. Springer, 2003. ISBN: 3-540-00161-1.

Online-Ressourcen

[B1] Mojang AB. *Minecraft*. URL: https://minecraft.net/ (besucht am 26.01.2015).

[B2] Christopher Chen. *2048 is in NP | Open Endings*. 2014. URL: http://blog.openendings.net/2014/03/2048-is-in-np.html (besucht am 25.02.2015).

[B3] Gabriele Cirulli. *2048 – Gabriele Cirulli*. 2014. URL: http://gabrielecirulli.com/works/2048 (besucht am 13.02.2015).

[B4] Owen Hill. *Yes, we're being bought by Microsoft*. 2014. URL: https://mojang.com/2014/09/yes-were-being-bought-by-microsoft/ (besucht am 26.01.2015).

[B5] Andrey Shevchuk. *Game about Squares*. 2014. URL: http://gameaboutsquares.com/ (besucht am 25.02.2015).

[B6] the1laz. *Working CPU with RAM, branching, etc... (save added)*. 2010. URL: http://www.minecraftforum.net/forums/minecraft-discussion/redstone-discussion-and/348979-working-cpu-with-ram-branching-etc-save-added (besucht am 29.01.2015).

[B7] Official Minecraft Wiki. *1.3.1*. 2014. URL: http://minecraft.gamepedia.com/1.3.1 (besucht am 26.01.2015).

[B8] Official Minecraft Wiki. *Hunger*. 2015. URL: http://minecraft.gamepedia.com/Hunger (besucht am 01.02.2015).

[B9] Official Minecraft Wiki. *Light*. 2015. URL: http://minecraft.gamepedia.com/Light (besucht am 01.02.2015).

[B10] Official Minecraft Wiki. *Spawn/Multiplayer details*. 2014. URL: http://minecraft.gamepedia.com/Spawn/Multiplayer_details (besucht am 01.02.2015).

[B11] Official Minecraft Wiki. *Version history (disambiguation)*. 2014. URL: http://minecraft.gamepedia.com/Version_history_(disambiguation) (besucht am 26.01.2015).

[B12] WubbiConcepts. *Snapshot 14w07a: „Brainfuck" interpreter using the new scoreboard features*. 2014. URL: https://www.youtube.com/watch?v=4g1QknsPwIM (besucht am 29.01.2015).