Seminararbeit

Sorting Colored Balls in Colored Tubes von Ernst Althaus et. al

Philipp Geier

Semester:
Abgabedatum: 28. März 2025
Betreuer: Jun.-Prof. Dr. Philipp Kindermann

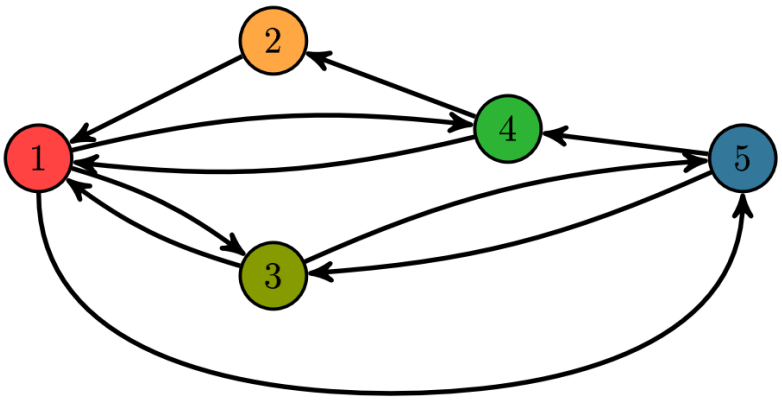
Universität Trier
Fachbereich IV - Informatik
Algorithmik

1 Einleitung

In dieser Ausarbeitung geht es um das Thema „Sorting Colored Balls in Colored Tubes“ von Ernst Althaus, Markus Blumenstock, Nick Johannes Peter Rassau, Felix Schuhknecht und Anton Quentin Zimdars. Ein Spiel aus der deutschen Fernsehserie „Frag doch mal die Maus“ dient als dient als Inspiration für das Problem. Wir wollen jeden Ball in die Tube seiner Farbe sortieren, indem wir einen oberen Ball von einer Tube in eine andere legen, und dürfen dabei eine extra farblose Tube benutzen. Dieses Problem ist das Sorting Colored Balls into colored Tubes Problem (SCBT).

1.1 Idee

Wir zeigen generelle Eigenschaften zur Berechenbarkeit und, dass das Problem, die Anzahl an Zügen zu minimieren, NP-schwer ist. Da das Spiel ohne Höhenbegrenzung dem Feedback Arc Set Problem (FAS), konstruieren wir das SCBT Problem als Graphen und zeigen die Korrektheit dessen. Dadurch reduzieren wir das SCBT Problem auf das FAS Problem. Weil das FAS Problem NP-vollständig ist, ist somit auch

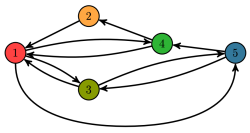


a The graph of a Feedback Arc Set instance.

Abb. 1: ⇒-linepushes
Quelle: [**akitaya2022pushing**]

1.2 Anwendung

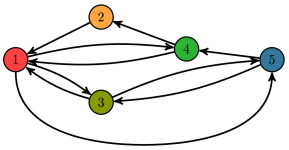
Das Konzept vom Manipulieren von Objekten durch uniforme externe Kräfte kommt in mehrere Spielen vor.
Beispielsweise kommt das Konzept im Computer-Spiel „2048“ vor [Abb. 2]. Hier gibt es ein quadratisches Feld mit 16 Feldern, auf welchem Tokens drauf sein können. Zu Beginn sind 2 Tokens mit der Aufschrift „2“ zufällig auf dem Feld verteilt. Durch Schiebeoperationen verschiebt man alle Tokens in eine Richtung, sodass diese an den Rand oder an Tokens, welche den Rand berühren, angrenzen. Diese Operationen können wir uns als mehrere linepush-Operationen in eine Richtung vorstellen, also so lange einen linepush machen bis die Konfiguration sich nicht mehr verändert. Nach jeder Verschiebung kommt ein weiteres Token mit der Aufschrift „2“ oder „4“ hinzu. Das besondere an 2048 ist jedoch, dass Tokens mit der selben Nummer beim Verschieben zusammen kommen und die Nummern addiert werden. Gespielt wird bis eine Verschiebung die Konfiguration nicht verändern kann.



a The graph of a Feedback Arc Set instance.

Abb. 2: Spiel 2048
Quelle: [**2048**]

Auch in Geschicklichkeitsspielen wie „Labyrinth“ [Abb. 3] wird das Konzept vom Manipulieren von Objekten genutzt. In diesem Spiel ist die uniforme externe Kraft die Gravitation. Eine Kugel liegt in einem Labyrinth, welches über 2 Räder an den Seiten in unterschiedliche Richtungen geneigt werden kann. Durch die Kombination beider Räder sind nicht nur die Richtungen $\Leftarrow, \Uparrow, \Rightarrow, \Downarrow$ möglich, sondern auch diagonale Richtungen. Durch die Neigung des Feldes fängt die Kugel an zu rollen. Das Ziel des Spieles ist, die Kugel an das Ende des Labyrinths zu führen, ohne dass diese in ein Loch fällt.



a The graph of a Feedback Arc Set instance.

Abb. 3: Geschicklichkeitsspiel Labyrinth
Quelle: [**labyrinth**]

Zum anderen wird das Prinzip für den Selbstzusammenbau und in der robotischen Bewegungsplanung genutzt. Hier wird oftmals mit Teilchen im Micro- und Nanobereich gearbeitet, sodass eine individuelle Bewegung der Teilchen nicht vorstellbar ist. Will man die Teilchen nun bewegen, ist die Herangehensweise mit Hilfe von linepushes sinnvoller, da uniforme externe Kräfte wie ein linepush realisierbar sind.

2 Puzzle

Für den weiteren Verlauf der Ausarbeitung sind folgende Definitionen sehr wichtig und werden durchgängig benutzt.

2.1 Äquivalenz

Definition 1. *Zwei Konfigurationen sind äquivalent zueinander, falls die beiden Konfigurationen identisch im Begrenzungsrechteck sind.*

Die Konfigurationen werden also bezogen auf das Begrenzungsrechteck betrachtet und bei gleicher Größe des Begrenzungsrechtecks und gleicher Position der Tokens in diesem sind die Konfigurationen äquivalent.

2.2 Kanonische Konfiguration

Definition 2. *Eine Konfiguration heißt kanonisch, falls mehrere $\Rightarrow$-linepushes und $\Uparrow$-linepushes eine äquivalente Konfiguration erzeugen.*

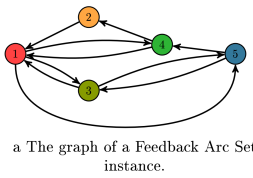
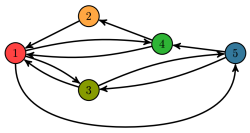


Abb. 4: Kanonische Konfiguration
Quelle: [**akitaya2022pushing**]

Eine kanonische Konfiguration ist in Abbildung 4 zu sehen. Hier würde ein $\Rightarrow$-linepush alle Tokens um eine Einheit nach rechts verschieben, da jede Reihe an Tokens an die linke Seite des Begrenzungsrechteck angrenzen und jedes weitere Token unmittelbar von rechts an diese Tokens angrenzt und somit mit nach rechts verschoben werden. Analog ist dies auch für ein $\Uparrow$-linepush und alle Tokens würden um eine Einheit nach oben verschoben werden. Insgesamt bleibt die Konfiguration äquivalent.

2.3 Kompakte Konfiguration

Definition 3. *Eine Konfiguration heißt kompakt, falls ein beliebiger linepush durch einen oder mehrere linepushes rückgängig gemacht werden kann.*

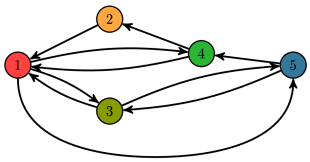


a The graph of a Feedback Arc Set instance.

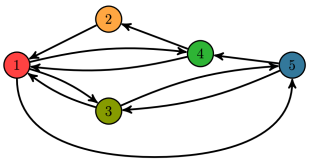
Abb. 5: Kompakte Konfiguration
Quelle: [**akitaya2022pushing**]

Ein Beispiel einer kompakten Konfiguration ist in Abbildung 5 abgebildet. Jeder linepush in eine beliebige Richtung kann durch einen oder mehrere linepushes zu einer äquivalenten Konfiguration gemacht werden.

Schauen wir uns beispielsweise ein $\Leftarrow$-linepush an [Abb. 6]:



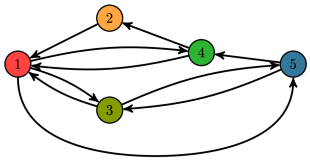
a The graph of a Feedback Arc Set instance.



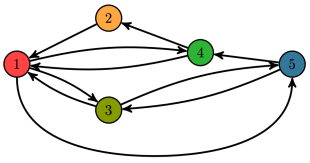
a The graph of a Feedback Arc Set instance.

Abb. 6: $\Leftarrow$-linepush am Beispiel

Um die Ausgangsposition zu erreichen, können wir durch abwechselnde $\Rightarrow$-linepushes und $\Uparrow$-linepushes eine kanonische Konfiguration erzeugen [Abb.7 (links)]. Darauf erreichen wir die Ausgangsposition [Abb.7 (rechts)] mit folgender Sequenz $\langle \Leftarrow^5 \Downarrow^4 \Uparrow \rangle$.



a The graph of a Feedback Arc Set instance.

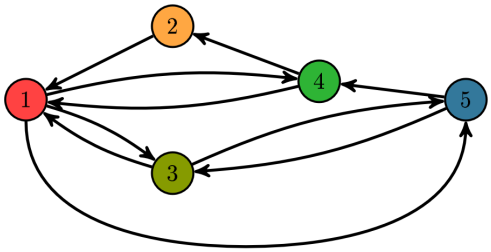


a The graph of a Feedback Arc Set instance.

Abb. 7: Wiederherstellung der Ausgangsposition

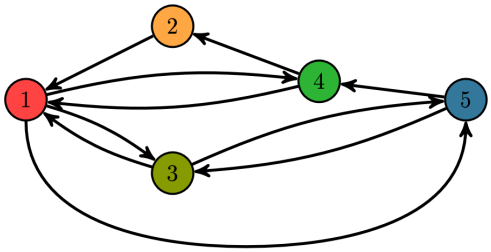
2.4 Kompatibilität

Definition 4. *Zwei Konfigurationen heißen kompatibel, falls die Anzahl von Reihen gleicher Länge in beiden Konfigurationen gleich ist und die Anzahl von Spalten gleicher Länge in beiden Konfigurationen gleich ist.*



a The graph of a Feedback Arc Set instance.

Abb. 8: Kompatibilität von Reihen
Quelle: [**akitaya2022pushing**]



a The graph of a Feedback Arc Set instance.

Abb. 9: Kompatibilität von Spalten
Quelle: [**akitaya2022pushing**]

Die beiden Konfiguration, die in Abbildung 8 und Abbildung 9 zu sehen sind, sind kompatibel. Denn die Abbildung 8 zeigt, dass die Anzahl Reihen gleicher Länge gleich ist und die Abbildung 9 zeigt, dass die Anzahl Spalten gleicher Länge gleich ist. Somit sind sie kompatibel.

2.5 Unbeschriftete/Beschriftete Puzzle

Wir betrachten uns in dieser Ausarbeitung unbeschriftete und beschriftete Puzzle.
Unbeschriftete Puzzle sind Puzzle mit Tokens, die alle entweder keine Beschriftung oder eine einheitliche Beschriftung haben. Hierbei nehmen wir an, dass wir die Tokens dann nicht unterscheiden können. Auch falls Tokens ihre Positionen tauschen wird die Konfiguration als äquivalent bezeichnet.

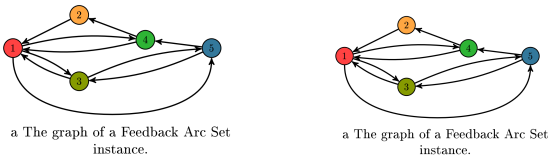


Abb. 10: Unbeschriftete Konfigurationen
Quelle: [**akitaya2022pushing**]

Beschriftete Puzzle haben eine nicht einheitliche Beschriftung. Die Beschriftung kann über Farben, Zahlen und auch anderweitigen Symbolen erfolgen, solange es mindestens zwei unterschiedliche Beschriftungen sind. Tokens können die selbe Beschriftung zu einem anderen haben, aber werden dann als zueinander äquivalent angesehen. Dies bedeutet, dass die Tokens ohne die Konfiguration zu verändern ausgetauscht werden können.

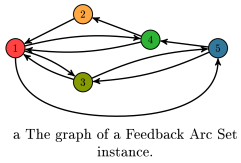


Abb. 11: Beschriftete Konfiguration
Quelle: [**akitaya2022pushing**]

3 Verdichtungspuzzle

Sei eine unbeschriftete verteilte Ausgangskonfiguration gegeben mit n vielen Tokens und $n = ab$. Gibt es eine Folge von linepushes, die eine Konfiguration erzeugt, bei der die Tokens ein $a \times b$ Feld bilden?

Mit anderen Worten versuchen wir das Begrenzungsrechteck der Anfangskonfiguration auf die Größe $a \times b$ mit Hilfe von linepushes zu verkleinern.

3.1 Komprimierbare Konfiguration

Definition 5. *Eine unbeschriftete Konfiguration heißt komprimierbar, falls ein linepush das Begrenzungsrechteck verkleinern kann.*

In Abbildung 12 ist eine komprimierbare Konfiguration zu sehen, da das Begrenzungsrechteck der Konfiguration beispielsweise durch einen $\Uparrow$-linepush verkleinert werden kann.

3.2 Verteilte Konfiguration

Definition 6. *Eine unbeschriftete Konfiguration heißt verteilt, falls in jeder Reihe und Spalte maximal ein Token liegt.*

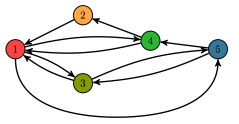


Abb. 12: Komprimierbare Konfiguration
Quelle: [**akitaya2022pushing**]

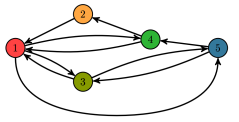


Abb. 13: Verteilte Konfiguration
Quelle: [**akitaya2022pushing**]

In Abbildung 13 ist eine verteilte Konfiguration. Jede Reihe hat maximal ein Token und gleichermaßen gilt dies für jede Spalte.

3.3 Theorem

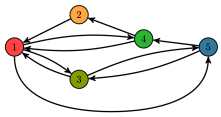
Alle verteilten Konfigurationen von $n = a \cdot b$ Tokens mit $a, b \in \mathbb{N}$ kann in eine $a \times b$ Box gedrückt werden, falls und nur falls $a \leq 2$ oder $b \leq 2$ oder $a = b = 3$.

Beweis.
Da wir von einer verteilten Konfiguration ausgehen, gilt das Theorem für $1 \leq a \leq 2$. Weil in jeder Reihe nur maximal ein Token sein kann, können wir $\Uparrow$-linepushes machen bis wir eine Reihe mit b vielen Tokens haben.
Falls $a = 2$ führen wir $\Downarrow$-linepushes aus bis wir eine weitere Reihe mit b vielen Tokens haben und diese beide Reihen werden danach mit $\Uparrow$-linepushes oder $\Downarrow$-linepushes zusammengeführt, sodass wir nur noch a viele Reihen haben. Schließlich führen wir $\Rightarrow$-linepushes aus bis unsere Begrenzungsrechteck eine $a \times b$ Box ist. Analog gilt dies für $1 \leq b \leq 2$.

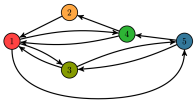
Für $a = b = 3$ gehen wir zuerst gleichermaßen vor und erhalten zwei Reihen mit jeweils drei Tokens. Da wir zu Beginn eine verteilte Konfiguration hatten, ist in jeder Spalte maximal ein Token. Dazu sind genau 3 Tokens nicht in der oberen oder unteren Reihe. Dadurch lassen sich die Tokens mit weiteren linepushes zu einer 3×3 Box zusammenführen.

Für jedes anderweitige a und b gibt es eine mögliche Startkonfiguration, bei welcher keine $a \times b$ Box durch linepushes entstehen kann:

Bei dieser Startkonfiguration sind die Tokens in zwei Diagonalen unterteilt wie in Abbil-



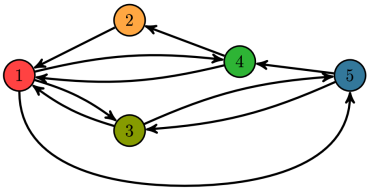
(a) Start

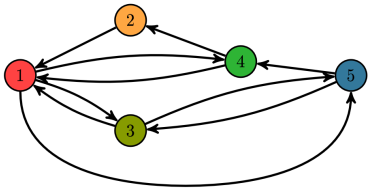


(b) L-Form

(c) Ende

Abb. 14: Ablauf für die doppelte diagonale Ausgangskonfiguration
Quelle: [**akitaya2022pushing**]

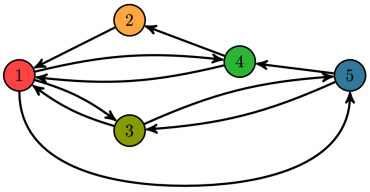


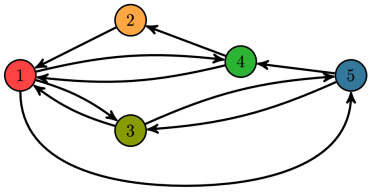


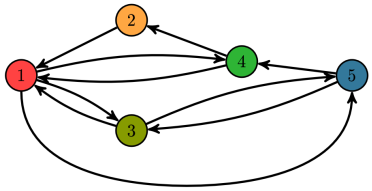
dung 14 (a) zu sehen ist. Durch linepushes können wir keine $a \times b$ Box erhalten, da diese L-Formen [Abb. 14 (b)] entstehen, welche a (bzw. b) viele Tokens in der Reihe (bzw. Spalte) haben. Dadurch kann die Komprimierung am Ende nicht mehr funktionieren, da mehr als a (bzw. b) viele Tokens in der Reihe (bzw. Spalte) sind [Abb. 14 (c)]. □

3.4 Programm

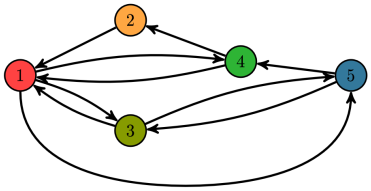
Um das Verdichtungspuzzle besser zu verstehen, habe ich ein JavaFX Programm geschrieben, um mit diesem linepushes zu simulieren.
Auf einer Gridpane werden rote beschriftete Tokens in einem Raster angezeigt und durch das Drücken auf die Knöpfe kann ein linepush in die Richtung des Pfeiles auf den Knöpfen ausgeführt werden.
Solange die Konfiguration nicht kompakt ist, kann ein linepush das Begrenzungsrechteck verkleinern. Dieses passt sich automatisch an die Konfiguration an.







a The graph of a Feedback Arc Set instance.



a The graph of a Feedback Arc Set instance.

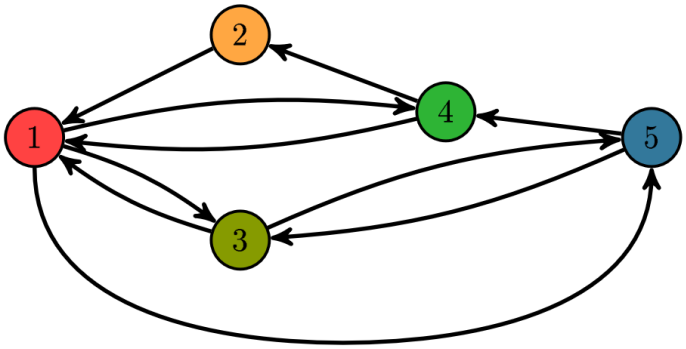
Abb. 17: Programm für das Verdichtungspuzzle

4 Permutationspuzzle

Sei eine beschriftete kanonische Ausgangskonfiguration gegeben. Gibt es eine Folge von linepushes, die eine bestimmte Permutation erzeugt?

4.1 Selbe Form

Definition 7. *Eine beschriftete Konfiguration K_1 hat die selbe Form wie eine beschriftete Konfiguration K_2, falls die beiden Konfigurationen ohne die Beschriftung der Tokens äquivalent sind.*



a The graph of a Feedback Arc Set instance.

Abb. 18: Konfigurationen der selben Form
Quelle: [**akitaya2022pushing**]

Abbildung 18 zeigt zwei Konfigurationen der selben Form mit einer unterschiedlichen Beschriftung.

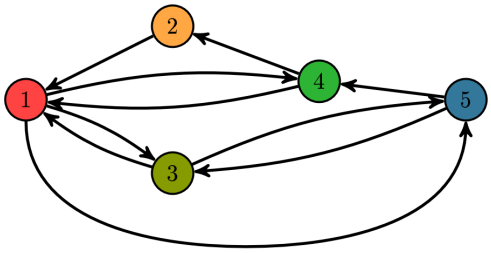
4.2 Permutation

Definition 8. *Für die Permutation betrachten wir nur beschriftete kanonische Konfigurationen. Eine Permutation einer beschrifteten Konfiguration ist eine Konfiguration der*

selben Form.

4.3 Dualpuzzle

Ein Dualpuzzle besteht aus einem Primärpuzzle mit einer kompakten Ausgangskonfiguration und dem Dual. Das Dual wird aus dem Inversen des Primärpuzzles gebildet. Alle leeren Felder im Begrenzungsrechteck des Primärpuzzles werden ausgefüllt und alle gefüllten Felder leer gemacht. Dann werden die Tokens sinnvoll angeordnet, sodass diese eine kompakte Konfiguration bilden. Um diese Konfiguration entsteht schließlich noch ein neues kleineres Begrenzungsrechteck. Ist das Primärpuzzle kanonisch, so nennt man das Dualpuzzle auch kanonisch.

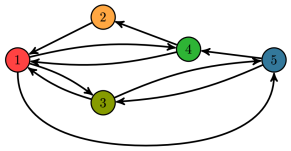


a The graph of a Feedback Arc Set instance.

Abb. 19: Primärpuzzle (links) / Dual (rechts)
Quelle: [**akitaya2022pushing**]

4.4 Unbeweglicher Zentraler Kern

Bei einem Permutationspuzzle schauen wir uns Permutationen einer Ausgangskonfiguration an. Jedoch sind unter bestimmten Umständen manche Tokens nicht beweglich. Falls mehr als die Hälfte der Reihen und mehr als die Hälfte der Spalten voll ist, so existiert ein unbeweglicher zentraler Kern. Die Tokens in diesem Kern lassen sich durch keinen linepush innerhalb des Begrenzungsrechtecks verschieben.



a The graph of a Feedback Arc Set instance.

Abb. 20: Konfiguration mit einem unbeweglichen zentralen Kern
Quelle: [**akitaya2022pushing**]

Da die doppelte Anzahl ausgefüllter Reihen in Abbildung 20, also Reihen die von einer Seite des Begrenzungsrechtecks zur anderen Seite reichen, größer als die Anzahl der Reihen ist und dies gleichermaßen für die Spalten gilt, haben wir einen unbeweglichen zentralen Kern. Dieser Kern ist mit einem blauen Rechteck in der Abbildung 20 gekennzeichnet.

4.5 Theorem

Alle möglichen Permutationen sind nur mit einer geraden Anzahl von Transpositionen erreichbar. Das heißt alle möglichen Permutationen sind gerade.

Beweis.
Zuerst bilden wir das Dual eines Dualpuzzle und erhalten ein Puzzle, das den Regeln eines Primärpuzzles folgt. Das Begrenzungsrechteck des Duals eines Puzzles ist dazu immer kleiner als das Begrenzungsrechteck des Puzzles selber. Daher können wir nun mit Induktion über die Größe des Begrenzungsrechtecks argumentieren.

Falls das Begrenzungsrechteck ausgefüllt ist, so haben wir nur die Permutation der Identität vorliegen, da linepushes nichts bewirken. Die Identität ist eine gerade Permutation.

Allgemein gilt für ein Primärpuzzle, dass auf jeden linepush, der eine nicht äquivalente Konfiguration zum Vorherigen erzeugt, ein entgegengesetzter linepush, der auch eine nicht äquivalente Konfiguration erzeugt, innerhalb der Sequenz folgen muss. Denn die Endkonfiguration muss die selbe Form zur Ausgangskonfiguration haben.

Es sei nun eine nicht verdichtete kanonische Konfiguration C gegeben. Dieses Puzzle bildet ein kanonisches Dualpuzzle. Das Dual des Dualpuzzles ist eine kanonische Konfiguration D mit einem kleineren Begrenzungsrechteck als das von C. Durch die kleinere

Größe des Begrenzungsrechteck gilt die Induktionshypothese. Nach Durchführung von mehreren linepushes und dem Wiederherstellen von einer kanonischen Konfiguration haben wir nun eine Permutation π von D und eine Permutation σ von C vorliegen. Da C und D den Regeln des Primärpuzzles folgen, ist die gesamte Permutation $\pi\sigma$ gerade. Durch die Induktionshypothese ist σ gerade und daher auch π. □

4.6 Generierung von Zyklen

Alle möglichen Permutationen können mit folgenden Sequenzen erreicht werden. Diese Sequenzen haben die Variable $k \in \mathbb{N}_0$, mit der die unterschiedlichen Permutationen erreicht werden. Die Richtungen in der Sequenz geben dabei die Richtung der linepushes an.

- Typ-A k-Sequenz: $\langle \Leftarrow^k \Leftarrow \Downarrow \Rightarrow \Uparrow \Rightarrow^k \rangle$
- Typ-B k-Sequenz: $\langle \Downarrow^k \Downarrow \Leftarrow \Uparrow \Rightarrow \Uparrow^k \rangle$
- Typ-C k-Sequenz: $\langle \Leftarrow^k \Downarrow \Leftarrow \Uparrow \Rightarrow \Rightarrow^k \rangle$

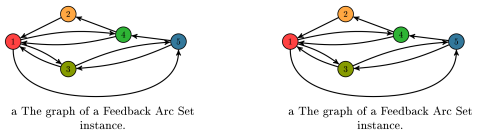


Abb. 21: Typ-A 4-Sequenz
Quelle: [**akitaya2022pushing**]

4.7 Theorem

Sei G_C unsere Permutationsgruppe für unser Puzzle.

Falls kein Feld frei ist, dann ist G_C die triviale Gruppe.
Das Begrenzungsrechteck ist gefüllt und linepushes verändern die Konfiguration nicht und daher liegt die triviale Gruppe, auch Identität genannt, vor. Dies erkennt man auch in Abbildung 22.

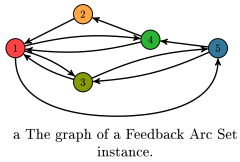


Abb. 22: Dichte Konfiguration
Quelle: [**akitaya2022pushing**]

Falls genau ein Feld frei ist, dann wird G_C zyklisch über die nicht-Kern Tokens generiert, da Permutationen nur durch das zyklische Rotieren der Tokens am Rand des Begrenzungsrechteck möglich sind wie in Abbildung 23 zu sehen ist.

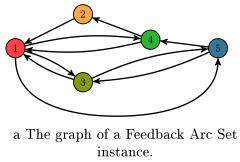
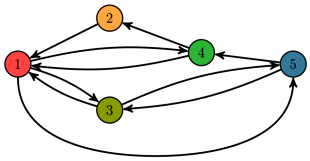


Abb. 23: zyklische Permutation
Quelle: [**akitaya2022pushing**]

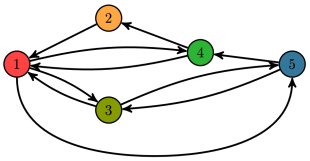
Ansonsten ist G_C entweder isomorph zu der alternierenden Gruppe und hat somit eine ähnliche Struktur und Eigenschaften wie diese oder die alternierende Gruppe über die nicht-Kern Tokens. Die alternierende Gruppe ist dabei die Permutationsgruppe, die alle geraden Permutationen beinhaltet.

4.8 Programm

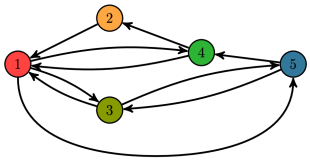
Für das Permutationspuzzle hab ich ebenso ein JavaFX Programm geschrieben, um mit diesem linepushes zu simulieren und mögliche Permutationen anzuschauen.
Hier wird, wie beim Verdichtungspuzzle, ein Raster mit roten beschrifteten Tokens angezeigt und durch das Drücken auf die Knöpfe kann ein linepush in die Richtung des Pfeiles auf den Knöpfen ausgeführt werden. Dazu ist es möglich, die Beschriftung der leeren Felder sich auch anzeigen zu lassen, wie in den oberen beiden Bildern in Abbildung 26 zu sehen ist.
Da wir uns nur kompakte Konfigurationen anschauen, wird hier die Anpassung des Begrenzungsrechtecks nicht benötigt und daher bleibt dieses bei jedem linepush gleich.



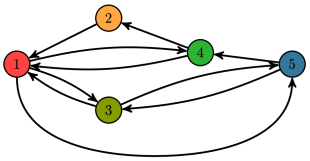
a The graph of a Feedback Arc Set instance.



a The graph of a Feedback Arc Set instance.

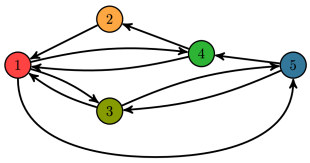


a The graph of a Feedback Arc Set instance.

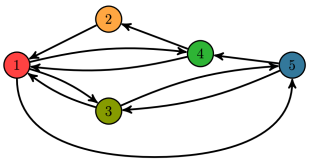


a The graph of a Feedback Arc Set instance.

Auch für das Dualpuzzle habe ich ein JavaFX Programm geschrieben. Dieses gleicht der Funktionsweise des Permutationspuzzles, aber dazu wird eine weitere GridPane angezeigt, auf der das zum Primärpuzzle entsprechende Dual abgebildet ist. Die für das Dual gespeicherten Werte werden folgenden berrechnet: Im ersten Schritt werden die inversen Werte zum Primärpuzzle gebildet. Dies bedeutet, dass jede „1“ aus dem Primärpuzzle zur „0“ im Dual wird und die „0“ zur „1“. Die „0“ zeigt dabei ein Rasterfeld ohne Token und die „1“ ein Rasterfeld mit einem Token an. Im zweiten Schritt werden die Tokens durch Verschiebungen zu einer kompakten Konfiguration geformt und schließlich wird ein berechnetes blaues Begrenzungsrechteck für das Dual hinzugefügt.

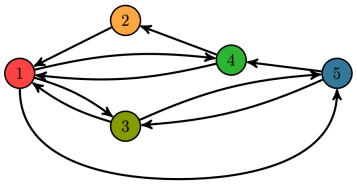


a The graph of a Feedback Arc Set instance.

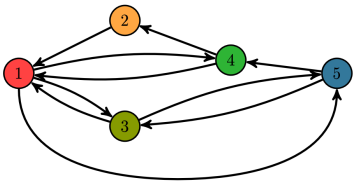


a The graph of a Feedback Arc Set instance.

Abb. 26: Programm für das Permutationspuzzle



a The graph of a Feedback Arc Set instance.



a The graph of a Feedback Arc Set instance.

Abb. 27: Programm für das Dualpuzzle