30. Bundeswettbewerb Informatik, 2. Runde

Lösungshinweise und Bewertungskriterien

Allgemeines

Es ist immer wieder bewundernswert, wie viele Ideen, wie viel Wissen, Fleiß und Durchhaltevermögen in den Einsendungen zur zweiten Runde eines Bundeswettbewerbs Informatik stecken. Um aber die Allerbesten für die Endrunde zu bestimmen, müssen wir die Arbeiten kritisch begutachten und hohe Anforderungen stellen. Von daher sind Punktabzüge die Regel und Bewertungen über die Erwartungen (5 Punkte) hinaus die Ausnahme. Lassen Sie sich davon nicht entmutigen! Wie auch immer Ihre Einsendung bewertet wurde: Allein durch die Arbeit an den Aufgaben und den Einsendungen hat jede Teilnehmerin und jeder Teilnehmer einiges dazu gelernt; den Wert dieses Effektes sollten Sie nicht unterschätzen.

Bevor Sie sich in die Lösungshinweise vertiefen, lesen Sie doch bitte kurz die folgenden Anmerkungen zu Einsendungen und den beiliegenden Unterlagen durch.

Terminprobleme Einige Einsender gestehen ganz offen, dass Ihnen die Zeit zum Einsendeschluss hin knapp geworden ist, worauf wir bei der Bewertung leider keine Rücksicht nehmen können. Abiturienten macht der Terminkonflikt mit der Abiturvorbereitung Probleme. Der ist für eine erfolgreiche Teilnahme sicher nicht ideal. In der zweiten Jahreshälfte läuft aber die zweite Runde des Mathewettbewerbs, dem wir keine Konkurrenz machen wollen. Also bleibt uns nur die erste Jahreshälfte. Aber: Sie hatten etwa vier Monate Bearbeitungszeit für die zweite BwInf-Runde. Rechtzeitig mit der Bearbeitung der Aufgaben zu beginnen war der beste Weg, Konflikte mit dem Abitur zu vermeiden.

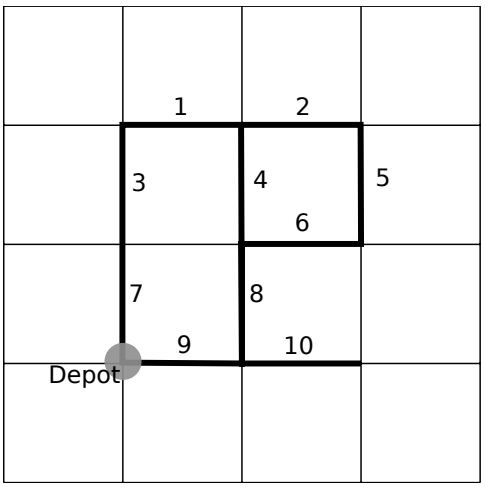
Dokumentation Es ist sehr gut nachvollziehbar, dass Sie Ihre Energie bevorzugt in die Lösung der Aufgaben, die Entwicklung Ihrer Ideen und die Umsetzung in Software fließen lassen. Doch ohne eine gute Beschreibung der Lösungsideen, eine übersichtliche Dokumentation der wichtigsten Komponenten Ihrer Programme, eine gute Kommentierung der Quellcodes und eine ausreichende Zahl sinnvoller Beispiele (die die verschiedenen bei der Lösung

des Problems zu berücksichtigenden Fälle abdecken) ist eine Einsendung wenig wert. Bewerterinnen und Bewerter können die Qualität Ihrer Einsendung nur anhand dieser Informationen vernünftig einschätzen. Mängel können nur selten durch gründliches Testen der eingesandten Programme ausgeglichen werden – wenn diese denn überhaupt ausgeführt werden können: Hier gibt es häufig Probleme, die meist vermieden werden könnten, wenn Lösungsprogramme vor der Einsendung nicht nur auf dem eigenen, sondern auch einmal auf einem fremden Rechner getestet würden. Insgesamt sollte die Erstellung des schriftlichen Materials die Programmierarbeit begleiten oder ihr teilweise sogar vorangehen: Wer nicht in der Lage ist, Idee und Modell präzise zu formulieren, bekommt keine saubere Umsetzung in welche Programmiersprache auch immer hin.

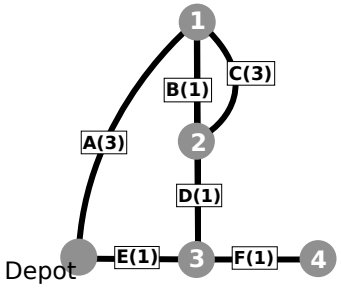
Bewertungsbögen Auf Ihrem Bewertungsbogen sind die Bewertungskriterien als Stichpunkte in den Tabellenzeilen aufgeführt. Kein Kreuz in einer Zeile bedeutet, dass das Kriterium den Erwartungen entsprechend erfüllt wurde. Vermerkt wird also in der Regel nur, wenn davon abgewichen wurde – nach oben oder nach unten. Ein Kreuz in der Spalte „+“ bedeutet Zusatzpunkte, ein Kreuz unter „–“ bedeutet Minuspunkte für Fehlendes oder Unzulängliches. Dabei haben die Marken nicht immer den gleichen Einfluss auf die Gesamtbewertung, sondern können je nach Einsendung und Kriterium unterschiedlich gewichtig sein. Die Schattierung eines Feldes bedeutet, dass die entsprechenden Zusatz- bzw. Minuspunkte in der Regel nicht vergeben wurden.

Lösungshinweise Bei den folgenden Erläuterungen handelt es sich um Vorschläge, nicht um die einzigen Lösungswege, die wir gelten ließen. Wir akzeptieren in der Regel alle Ansätze, die die gestellte Aufgabe vernünftig lösen und entsprechend dokumentiert sind. Einige Dinge gibt es allerdings, die – unabhängig vom gewählten Lösungsweg – auf jeden Fall diskutiert werden müssen. Zu jeder Aufgabe gibt es deshalb einen Abschnitt, indem gesagt wird, worauf bei der Bewertung letztlich geachtet wurde, zusätzlich zu den grundlegenden Anforderungen an Dokumentation (insbesondere: klare Beschreibung der Lösungsidee, genügend aussagekräftige Beispiele) und Quellcode (insbesondere: Strukturierung und Kommentierung, gute Übersicht durch Programm-Dokumentation).

Danksagung An der Erstellung der Lösungsideen haben mitgewirkt: Karl Bringmann (Aufgaben 1 und 3), Marvin Künnemann (Aufgaben 1 und 3) und Wolfgang Pohl. Die Aufgaben wurden vom Aufgabenausschuss des Bundeswettbewerbs Informatik entwickelt, und zwar aus Vorschlägen von Ralf Punkenburg (Aufgabe 1), Rainer Gemulla (Aufgabe 2) und Konstantinos Panagiotou (Aufgabe 3).



(a) Beispiel eines kleinen Straßennetzes. Die zu streuenden Straßenabschnitte sind durchnummeriert.



(b) Zusammengefasste Darstellung des Straßennetzes als gewichteter Graph. Die Kanten sind mit einem Bezeichner und der Länge des Segementes beschriftet, die Knoten sind indiziert.

Abbildung 1: Verschiedene Darstellungen eines einfachen Straßennetzes

Aufgabe 1: Ned

Kein Wunder, dass Konrad die Unterstützung von BwInf-lern bekommen soll: Seine Streuaufgabe gestaltet sich als komplexes algorithmisches Problem. Er muss nicht nur einen Weg finden, zu einer gegebenen Karte des Straßennetzes alle Straßenabschnitte abzufahren; er muss auch entscheiden, wann genau welcher Straßenabschnitt gestreut werden soll. Dabei ist insbesondere die Ladekapazität des Streuwagens zu berücksichtigen. Für den Aufgabenteil 1 genügt ihm jeder zulässige Weg; für Aufgabenteil 2 soll die Länge der abgefahrenen Strecke möglichst gering werden. Wir diskutieren zunächst Möglichkeiten zur Darstellung eines Streuplans, woraus sich direkt eine Lösung für Teil 1 ergibt, und benutzen diese Ergebnisse dann, um einen möglichst kurzen Streuplan zu erstellen.

1.1 Darstellung der Streupläne

Eine größere Schwierigkeit beim Lösen der Aufgabe stellt bereits die Wahl einer geeigneten Repräsentation eines Streuplans dar. Wir sehen einen *Streuplan* als eine Menge von *Rundfahrten* an. Jede Rundfahrt startet im Depot, fährt die Straßen im Straßennetz ab und kehrt anschließend zum Depot als letzter Station zurück. Dabei darf das Depot zwischendurch nicht besucht werden, sondern muss in jedem Rundweg genau zweimal, am Anfang und am Ende, angefahren werden. Die Anzahl an Rundwegen hängt im Allgemeinen von der Ladekapazität des Streumobils ab. Auch die Länge der Rundwege hängt stark von dem Straßennetz und der Ladekapazität des Wagens ab.

Im folgenden werden wir häufig die Entfernungen zwischen beliebigen Punkten im Straßennetz benötigen. Wir bezeichen mit $\text{dist}(u,v)$ die Länge des kürzesten Weges zwischen den Rasterpunkten u und v.

Reihenfolgenbasierte Darstellung

Da jeder Straßenabschnitt mindestens einmal abgefahren werden soll, liegt folgende Repräsentation nahe: Wir zählen alle Straßenabschnitte in der Reihenfolge auf, in der sie gestreut werden sollen, wobei wir ein weiteres Symbol $\perp$ für eine Rückkehr zum Depot einfügen. Weiterhin muss die Richtung, in der jeder Straßenabschnitt beim Streuen durchfahren werden soll, festgelegt werden. Die jeweils zwei Möglichkeiten, die es dafür gibt, stellen wir durch $\uparrow$ und $\downarrow$ im Fall vertikaler Straßen sowie durch $\leftarrow$ und $\rightarrow$ im Fall horizontaler Straßen dar. Im Beispiel in Abbildung 1(a) könnte für die Ladekapazität $T = 4$ ein möglicher Streuplan durch die folgende Zeichenfolge dargestellt werden:

$$(\rightarrow 2)(\downarrow 5)(\leftarrow 6)(\uparrow 4)\perp \quad (\uparrow 8)(\leftarrow 1)(\downarrow 3)(\downarrow 7)\perp \quad (\leftarrow 10)(\leftarrow 9)\perp$$

Der resultierende Streuweg ergibt sich nun daraus, dass ausgehend vom Depot immer ein kürzester Weg zum nächsten Ziel, d.h. der Startpunkt der nächsten gerichteten Kante, gewählt wird. Diese Kante wird dann gestreut, womit man zum Endpunkt der gerichteten Kante gelangt. Am Anfang jedes Rundwegs, also am Depot, lädt Konrad seinen Streuwagen vollständig mit Sand auf. Es ist gewährleistet, dass es immer einen optimalen Streuweg gibt, den wir auf diese Weise darstellen können: Für die Verbindungswege zwischen den zu streuenden Straßen und dem Depot kann immer ein beliebiger kürzester Pfad gewählt werden, und auch das volle Beladen des Schneemobils ist keine Einschränkung.

Mit dieser Darstellung können wir direkt einen beliebigen gültigen Streuplan erstellen: Wir zählen alle zu streuenden Straßenabschnitte in beliebiger Reihenfolge aus und fügen nach jedem T-ten Straßenabschnitt das Rückkehrsymbol $\perp$ ein.

Segmentbasierte Darstellung

Der obige Ansatz ist fokussiert auf die Reihenfolge einzelner Straßenabschnitte. Beim Planen eines Streuweges orientiert man sich jedoch häufig an den Kreuzungen und plant eine Tour, auf der man möglichst viele Straßen streuen kann, bis man zurückkehren muss. Die Entscheidungen werden praktisch nur an den Kreuzungen gefällt. Es liegt also gleichfalls nahe, das Straßennetz etwas zu vereinfachen und anders darzustellen: Als Graph, der als Knoten das Depot und alle Kreuzungen besitzt und durch gewichtete Kanten verbunden ist. Ein Beispiel dafür findet sich in Abbildung 1(b). Hintereinander folgende Straßenabschnitte, die eine Straße bilden, auf der man nicht abbiegen kann, werden dabei zu Straßensegmenten zusammengefasst. Bei dem entstehenden Graph muss beachtet werden, dass auch Mehrfachkanten und Schleifen entstehen können. Nun kann ein Streuplan durch eine Tour durch diesen Graphen dargestellt werden, wobei man für jede Tour entscheidet, welche der besuchte Straßenabschnitte gestreut werden sollen.

Stellt man den Graph wie vorgeschlagen dar, muss man auf ein paar Besonderheiten achten. Um den kürzest möglichen Streuplan zu finden, muss man zulassen, dass die Segmente auch unvollständig abgefahren werden können: Es muss erlaubt sein, in eine Straße einzubiegen, gewisse Straßenabschnitte zu streuen, dann aber zurückzukehren ohne das Segment komplett zu durchfahren.

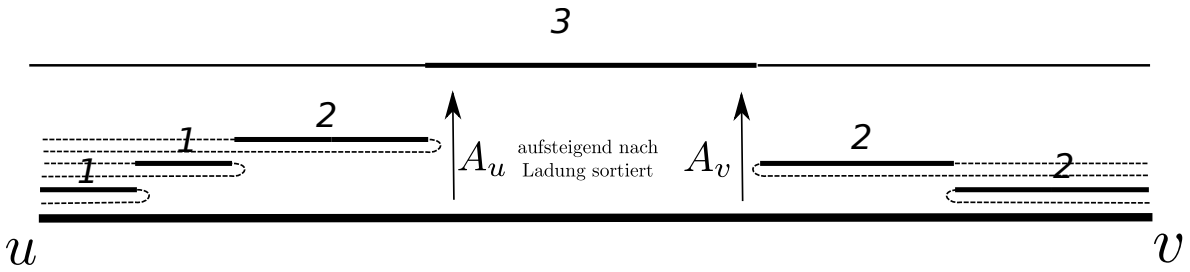


Abbildung 2: Die Aufteilung der Fahrten auf zu streuende Abschnitte. In diesem Beispiel sind Streufahrten $u \xrightarrow{S} u : 1$, $u \xrightarrow{S} u : 1$, $u \xrightarrow{S} u : 2$, $u \xrightarrow{S} v : 1$, $v \xrightarrow{S} u : 2$, $v \xrightarrow{S} v : 2$ und $v \xrightarrow{S} v : 2$ gegeben.

Nun können die Streupläne wie folgt dargestellt werden: Für jeden Rundweg eines Streuplans führen wir nur die Segmente an, auf denen etwas gestreut wird. Um die Richtung zu kodieren und Schleifen zu ermöglichen, geben wir dazu (1) zu jedem Segment den Knoten u an, über den wir auf das Segment einbiegen, (2) den Index S des Segments, auf dem wir streuen, (3) die Anzahl an Straßen l, die wir dort streuen und (4) den Knoten v, über den wir das Segment wieder verlassen. Dies schreiben wir dann $u \xrightarrow{S} v : l$. Den Streuplan von vorhin können wir also auch wie folgt darstellen:

1. Rundfahrt: $1 \xrightarrow{C} 2 : 3$, $2 \xrightarrow{B} 1 : 1$
2. Rundfahrt: $3 \xrightarrow{D} 2 : 1$, $1 \xrightarrow{A}$ **depot** $: 3$
3. Rundfahrt: $4 \xrightarrow{F} 3 : 1$, $3 \xrightarrow{E}$ **depot** $: 1$

Der Vorteil dieser Darstellung ist das Abstrahieren der einzelnen Straßenabschnitte zu Segmenten. Dadurch, dass wir für jedes Segment nur die Anzahl an Straßenabschnitten angeben, die gestreut werden sollen, überlassen wir die genaue Verteilung der folgenden Überlegung, die in Abbildung 2 illustriert ist: Angenommen das Segment S der Länge l von u nach v wird mehrfach besucht. Die Schleifen ausgehend von u, also Einträge im Streuplan der Form $u \xrightarrow{S} u : l$, bezeichnen wir mit A_u. Genauso sei A_v die Menge aller Schleifenfahrten ausgehend von v. Weiterhin interessiert uns nur die Gesamtanzahl an Abschnitten, die beim kompletten Durchfahren des Segmentes gestreut wurde, also die Summe aller Ladungen l_i von Fahrten $u \xrightarrow{S} v : l_i$ oder $v \xrightarrow{S} u : l_i$. Da letztere Fahrten ohnehin das gesamte Segment abfahren, ist es für sie komplett irrelevant, welche Abschnitte durch sie gestreut werden. Deswegen können wir zunächst festlegen, welche Schleifenfahrt welche Abschnitte streut.

Für die Schleifenfahrten fällt uns auf, dass es die kürzeste Fahrtstrecke ergibt, wenn alle Schleifenfahrten aufsteigend nach der Ladung sortiert abgefertigt werden. Dabei fährt der Streuwagen immer zum nächsten ungestreuten Abschnitt, streut dann die folgenden Abschnitte (so weit wie die Ladung reicht) und kehrt dann zum Anfang zurück. Intuitiv gesprochen versuchen wir, die „vielen kurzen“ zusammenhängenden Strecken nahe am Rand zu streuen, damit wir bei den „wenigen langen“ zusammenhängenden Strecken möglichst wenig zusätzlichen Weg haben. Auf den Beweis dafür verzichten wir aus Platzgründen.

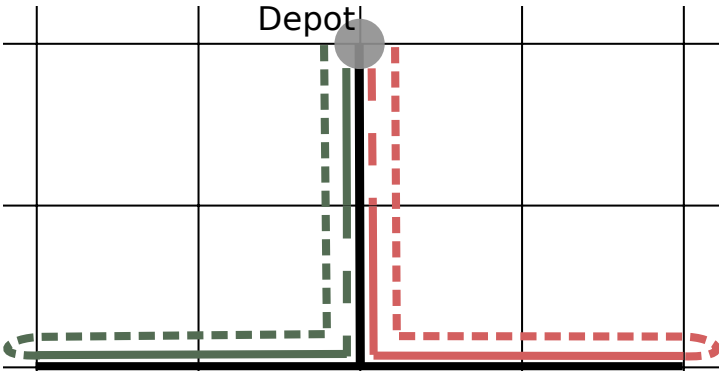


Abbildung 3: Ein Straßennetz mit optimalen Weg von 16 km, um alle Straßen mit Ladekapazität $T = 3$ zu streuen. Gestrichelte Linien zeigen Fahrten ohne gleichzeitiges Streuen an. Jede Lösung, bei der zuerst die beiden vertikalen Straßen gestreut werden, erzeugt einen Gesamtweg von mindestens 18 km. Also darf eine der beiden Straßen bei der ersten Tour nicht gestreut werden.

Vorsicht bei Vereinfachungen

Man ist schnell geneigt zu glauben, dass das Problem noch weiter vereinfacht werden kann. Reicht es nicht aus, wenn Konrad einfach den richtigen Weg rät und sofort jede bisher ungestreute Kante, auf die er trifft, streut und zum Depot zurückkehrt, sobald seine Ladung verbraucht ist? Die Antwort lautet leider Nein; es gibt wirklich Kanten, die er beim ersten Antreffen auslassen muss, wie das Beispiel in Abbildung 3 verdeutlicht.

1.2 Optimalität

Wie kann Konrad nun vorgehen, um seinen Streuweg möglichst gering zu halten?

Aufzählung aller Möglichkeiten

Mit der Wahl geeigneter Darstellungen wie oben könnte das Problem durch das Durchprobieren aller möglichen Streupläne gelöst werden. Da jedoch die Anzahl an Möglichkeiten in unserem Fall enorm groß ist, muss eine solche Lösung geschickte Pruning-Methoden verwenden, die frühzeitig Teilbäume des Entscheidungsbaums kappen können.

Eine vollständige Enumeration der Streupläne scheint nämlich aussichtslos: Für eine Instanz mit m Kanten gibt es $m!2^{2m}$ viele Streupläne in der reihenfolgenbasierten Darstellung. Für die gegebenen Beispiele mit $m = 33$ und $m = 44$ sind dies Zahlen mit 57 bzw. 81 Stellen.

Vereinfachung der Darstellung

Glücklicherweise können wir die reihenfolgenbasierte Darstellung von oben noch weiter vereinfachen: Es ist nicht einmal nötig, die Richtung der Kanten und die Rückkehr zum Depot

festzulegen! Der Grund dafür liegt im folgenden Trick: Mithilfe dynamischer Programmierung können wir aus einer gegebenen Reihenfolge Straßenabschnitte $S_1, \ldots, S_m$ schnell den optimalen Streuplan, der diese Reihenfolge berücksichtigt, berechnen. Gehen wir zunächst davon aus, dass die Richtung, in der die zu streuenden Kanten zu durchfahren sind, festgelegt ist. Dazu bezeichnen wir mit u_i den Rasterpunkt, von dem aus S_i durchfahren und gestreut werden soll, und mit v_i den Zielrasterpunkt. Wir definieren das Feld D, das zu jeder Stelle $1 \leq i \leq m$ in der Reihenfolge und jeder Ladung $0 \leq t < T$ die *minimale Weglänge* $D[i][t]$ speichert, mit der wir alle Straßenabschnitte $S_1, \ldots, S_i$ streuen konnten, uns nun am Ende von Straßenabschnitt S_i befinden (also in v_i) und noch Sand für mindestens t km geladen haben. Diese Werte können wir wie folgt bestimmen: Zunächst initialisieren wir die Werte $D[0][t] := 0$ für $t \in \{0, \ldots T-1\}$. Nun lassen wir die Laufvariable i von 1 bis m und eine weitere Laufvariable t von $T-1$ bis 0 laufen und wenden die folgenden Gleichungen an:

$$\begin{aligned} D[i][T-1] &= \min_{t=0,\ldots,T-1} D[i-1][t] + \mathrm{dist}(v_{i-1}, \mathbf{depot}) + \mathrm{dist}(\mathbf{depot}, u_i) + 1 \\ D[i][t] &= D[i-1][t+1] + \mathrm{dist}(v_{i-1}, u_i) + 1 \end{aligned}$$

Diese Gleichungen leiten sich aus folgenden Beobachtungen zum Zeitpunkt, zu dem der Abschnitt S_i gerade gestreut wurde, ab:

1. Das Schneemobil Ned kann nur dann eine Ladung Sand für mindestens $T-1$ km geladen haben, wenn es zwischen Abschnitt S_{i-1} und S_i im Depot war. Dabei spielt es keine Rolle, wieviel Ladung es am Ende von Abschnitt S_{i-1} noch geladen hatte.
2. Für alle anderen Ladungen t gilt, dass Ned vorher S_{i-1} mit einer Ladung für $t+1$ km gestreut hat und danach direkt zu S_i gefahren ist.

Der Wert $\min_{t=0,\ldots,T-1} D[m][t] + \mathrm{dist}(v_m, \mathbf{depot})$ ist nun die minimale Wegstrecke, die wir benötigen, um alle Straßenabschnitte in der angegebenen Reihenfolge zu streuen. Den genauen Streuplan kann man durch das „Rückwärtsverfolgen“ der obigen Gleichungsanwendungen bestimmen.

Dieser Algorithmus kann auch angepasst werden, um nicht nur die Einteilung in Rundfahrten (also die Positionen des Rückkehrsymbols $\perp$), sondern auch die optimale Richtung der Kanten festzulegen. Dazu erweitert man das Feld D um einen weiteren Index mit zwei möglichen Werten (beispielsweise 0 für Vorwärts, 1 für Rückwärts) mit der Bedeutung: $D[i][t][r]$ ist die minimale Weglänge, mit der die Straßenabschnitte $S_1, \ldots, S_i$ gestreut werden können, so dass noch wir noch Sand für mindestens t km geladen haben und Abschnitt S_i in der Richtung r durchfahren haben. Die anzuwendenden Gleichungen ergeben sich dann ähnlich wie oben.

Heuristiken

Mit der Vereinfachung der reihenfolgenbasierten Darstellung des obigen Abschnittes muss nur noch die optimale Reihenfolge der zu streuenden Straßenabschnitte gefunden werden! Dazu können wir lokale Suchmethoden verwenden – eine typische Strategie bei schwierigen Problemen. Dabei beginnt man beispielsweise mit einer zufälligen Reihenfolge der Straßenabschnitte und bestimmt mit dem obigen Algorithmus die Länge des optimalen Streuplans.

Von der aktuellen Reihenfolge aus versuchen wir eine ähnliche, neue Reihenfolge zu finden, die wir dann erneut untersuchen können. Eine Möglichkeit dies zu tun, ist es, zwei Elemente aus der Reihenfolge auszuwählen und ihre Nachfolger in der Reihenfolge auszutauschen. Damit dies Sinn ergibt, müssen wir dazu die Richtung des Abschnitts zwischen den beiden ausgewählten Elementen umkehren.

Stellen wir uns also vor, wir wählen die Elemente $1 \leq i \leq m-2$ und $i+2 \leq j \leq m$ aus. Ursprüngliche und resultierende Reihenfolge sehen dann beispielsweise so aus:

$$\begin{array}{lll} 1,\ldots,i-2,i-1, & i,i+1,\ldots j-2,j-1, & j,j+1,\ldots,n \\ 1,\ldots,i-2,i-1, & j-1,j-2,\ldots i+1,i, & j,j+1,\ldots,n \end{array}$$

Sollte die somit erzeugte Reihenfolge einen besseren Streuplan erzeugen, so können wir mit diesem weiter arbeiten. Man könnte beispielsweise die beste der so erzeugten Reihenfolgen auswählen und das Verfahren von ihr aus wiederholen; alternativ kann man in unserem Fall auch *alle* diese Reihenfolgen rekursiv bearbeiten, da das Verfahren schnell in einem lokalen Optimum landet. Dies bedeutet, dass eine Reihenfolge gefunden wurde, von der aus jede der oben beschriebenen Änderungen nur schlechtere Reihenfolgen erzeugt.

Durch die zufällige Wahl einer Reihenfolge am Anfang erhalten wir einen randomisierten Algorithmus, dessen Ergebnis und Laufzeit von dieser Zufallswahl abhängt. Er betrachtet zu einer Reihenfolge immer nur eine Nachbarschaft anderer Reihenfolgen, durch die falsche Wahl der Anfangsreihenfolge können wir so schnell in einem lokalen Optimum landen, das weit vom globalen Optimum entfernt ist. Um diese Wahrscheinlichkeit zu verkleinern, führen wir den Algorithmus mehrfach auf und merken uns das beste Ergebnis.

Diesen Algorithmus kann man weiter verbessern, etwa indem man eine bessere Startreihenfolge wählt oder weitere Änderungsoperationen an den Reihenfolgen zulässt.

1.3 Einfluss der Ladekapazität

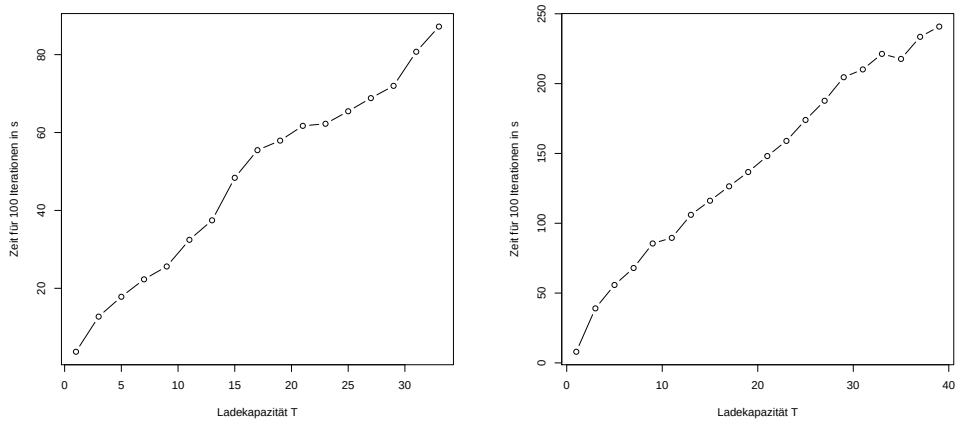
Betrachten wir den Fall der Ladekapazität $T = 1$. Wenn das Streumobil für jeden einzelnen Straßenabschnitt neu zum Depot zurückkehren muss, um die nötige Ladung Sand aufzunehmen, ist ein optimaler Streuweg leicht bestimmt: Mit unserer reihenfolgenbasierten Darstellung wird klar, dass jede Reihenfolge der zu streuenden Straßen dieselbe Weglänge erzeugt. Für jeden Straßenabschnitt S zwischen den Rasterpunkten u und v benötigt Konrad eine Tour, die (auf dem kürzesten Weg) vom Depot zu u geht, den Abschnitt nach v streut und (auf dem kürzesten Weg) zum Depot zurückkehrt (alternativ ist auch die Umkehrung des Weges möglich). Der Straßenabschnitt S trägt also um $\text{dist}(\textbf{depot}, u) + 1 + \text{dist}(v, \textbf{depot})$ zum Gesamtweg bei. Damit ist der optimale Streuweg gegeben durch:

$$\sum_{\text{Straßenabschnitt } S=\{u,v\}} \text{dist}(\textbf{depot}, u) + \text{dist}(\textbf{depot}, v) + 1$$

Im anderen Extremfall gilt $T \geq m$; Ned hat also genug Ladekapazität, um alle Straßenabschnitte auf einer Rundfahrt zu streuen. Dann würden wir nach der kürzesten Rundtour durch den Segmentgraphen suchen, die alle Segmente mindestens einmal enthält. Genau dieses Problem

ist auch als „Briefträgerproblem“ bzw. auf Englisch als „Chinese Postman Problem“ bekannt[1]. Und für dieses Problem existiert eine effiziente Lösung.

Für Kapazitäten $T = 1$ und $T \geq m$ ist das Problem also relativ leicht zu lösen. Gibt es vielleicht auch für Zwischenwerte bei den Kapazitäten eine einfache Lösung, die wir nur nicht finden können? Nein, die gibt es nicht. Teilaufgabe 2 entspricht nämlich recht genau einem in der Informatik bereits bekannten Problem, nämlich dem „Capacitated Arc Routing Problem“ (CARP) [2]. Wikipedia kennt CARP als Teilproblem der in der Entsorgungswirtschaft wichtigen Revierplanung. CARP ist als NP-schwer bekannt; ohne das $P = NP$ Problem zu lösen, können wir also keine grundsätzlich effiziente Lösung dafür finden.



(a) Beispiel 1 (b) Beispiel 2

Abbildung 4: Zeitaufwand unseres Verfahrens in Abhängigkeit der Ladekapazität für die vorgegebenen Straßennetze

Analysiert man hingegen konkrete Algorithmen für unser Streuproblem, so könnte sich ein anderes Bild ergeben – insbesondere natürlich bei Heuristiken, die nicht garantiert optimale Ergebnisse liefern. Für unsere Heuristik, die vor allem darauf basiert, aus einer Lösung neue verwandte Lösungen zu generieren und deren Qualität zu berechnen, liegt es nahe zu vermuten, dass die Laufzeit vor allem von der Berechnung der Qualität (die Komplexität der Zielfunktion) abhängt. Da der Ansatz der dynamischen Programmierung eine Laufzeit von $O(mT)$ ergibt, die Berechnungszeit also linear mit T steigt, erwarten wir also auch ein bzgl. der Laufzeit grob lineares Verhalten des Algorithmus', was in Abbildung 4 bestätigt wird.

1.4 Beispielinstanzen

Die kompakte Darstellung der mitgelieferten Beispiele findet sich in Abbildung 5. Unser Ansatz (mit klüger gewählten zufälligen Startreihenfolgen) liefert die folgenden Streuwege, hier in der segmentbasierten Darstellung und in Abbildung 6 auch grafisch angegeben:

[1] http://de.wikipedia.org/wiki/Briefträgerproblem

[2] Golden, B. L. and Wong, R. T. (1981), Capacitated arc routing problems. Networks, 11: 305–315

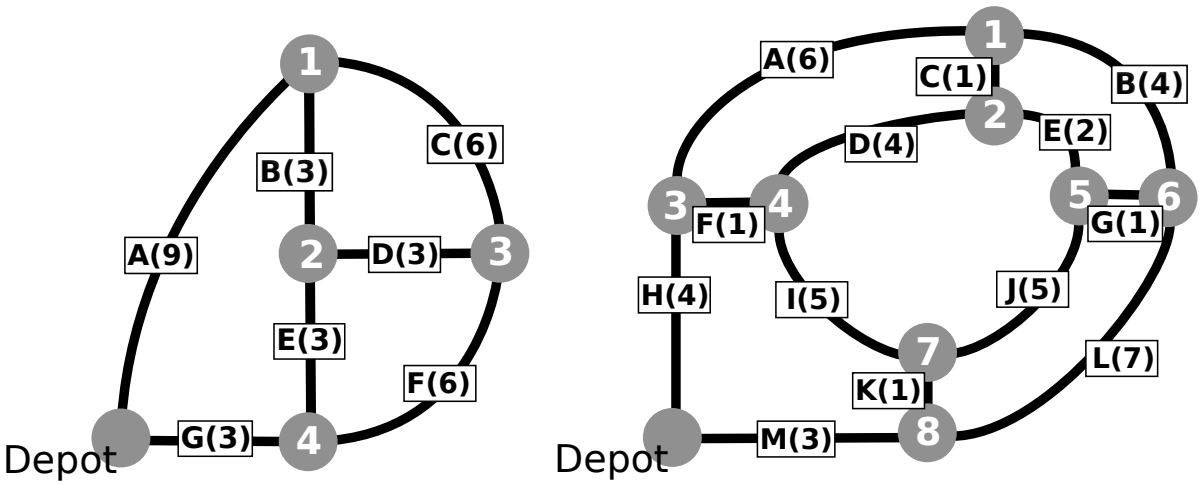


Abbildung 5: Kompakte Darstellung der Beispiele aus der Aufgabenstellung.

Beispiel 1: Streuweg 60 km

1. Rundfahrt: $2 \xrightarrow{B} 1:3, \quad 1 \xrightarrow{A} \textbf{depot}:9$
2. Rundfahrt: $2 \xrightarrow{D} 3:3, \quad 3 \xrightarrow{F} 4:6, \quad 4 \xrightarrow{G} \textbf{depot}:3$
3. Rundfahrt: $4 \xrightarrow{E} 2:3, \quad 1 \xrightarrow{C} 3:6$

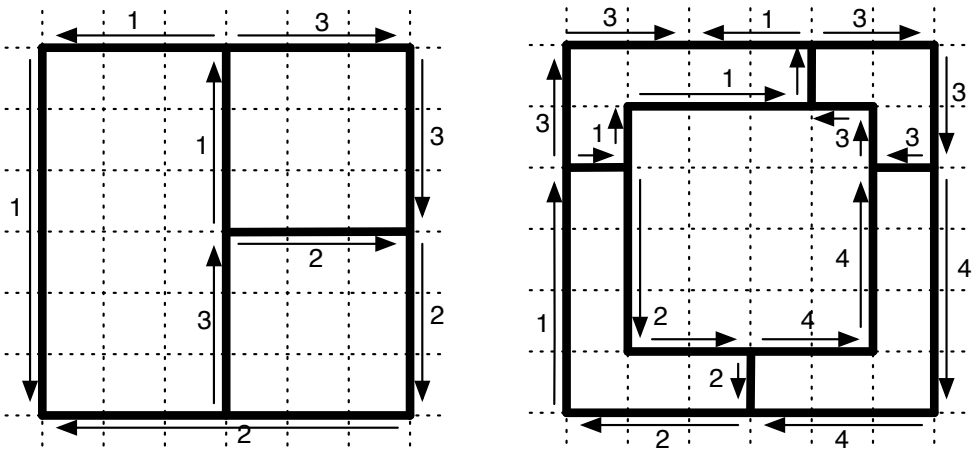
Beispiel 2: Streuweg 80 km

1. Rundfahrt: $\textbf{depot} \xrightarrow{H} 3:4,\ 3 \xrightarrow{F} 4:1,\ 4 \xrightarrow{D} 2:4,\ 2 \xrightarrow{C} 1:1,\ 1 \xrightarrow{A} 3:2$
2. Rundfahrt: $4 \xrightarrow{I} 7:5,\ 7 \xrightarrow{K} 8:1,\ 8 \xrightarrow{M} \textbf{depot}:3$
3. Rundfahrt: $3 \xrightarrow{A} 1:4,\ 1 \xrightarrow{B} 6:4,\ 6 \xrightarrow{G} 5:1,\ 5 \xrightarrow{E} 2:2$
4. Rundfahrt: $7 \xrightarrow{J} 5:5,\ 6 \xrightarrow{L} 8:7$

1.5 Mögliche Erweiterungen

Sinnvolle Erweiterungen könnten folgende sein:

- Man könnte die Möglichkeit einschränken, das Streuen beim Fahren jederzeit an- und auszustellen.
- Die Entscheidungsmöglichkeiten für den Streuwagen liegen nur in den Rasterpunkten. Lässt man hingegen zu, dass die Ladekapazität kein Vielfaches der Länge der Straßenabschnitte ist und die Entscheidung zum Umkehren oder Streuen auch innerhalb eines Straßenabschnitts gefällt werden kann, könnte das Problem weitaus schwieriger werden. Weitergehend könnte man sogar Kurven (also Straßenabschnitte mit einer Straßenlänge, die kein Vielfaches von 1 km ist) erlauben.



(a) Beispiel 1 (b) Beispiel 2

Abbildung 6: Streupläne mit minimaler Gesamtfahrstrecke für die Beispiele aus der Aufgabenstellung. Die bezifferten Pfeile zeigen, welche Straßenabschnitte bei den einzelnen Rundfahrten gestreut werden.

- Stark befahrene oder mehrspurige Straßen könnten einen höheren Streubedarf haben. Eine solche Erweiterung, die eine unterschiedliche Gewichtung der Straßenabschnitte erlaubt, ist interessant, wenn sie nicht einfach durch eine Vervielfachung der Abschnitte in der Eingabe auf das Ursprungsproblem zurückzuführen ist.
- Eine Verallgemeinerung der Straßennetze ist nur dann eine interessante Erweiterung, wenn die Lösung dadurch tatsächlich schwieriger wird. Wenn die Schwierigkeit lediglich darin besteht, das komplexere Straßennetz einzulesen und im Rechner darzustellen, der Algorithmus jedoch trivial erweitert werden kann, ist die Erweiterung nicht besonders interessant.
- Anstelle des Gesamtweges kann auch die längste Rundfahrt minimiert werden. Dies entspricht der Interpretation, dass ausreichend Streuwagen vorhanden sind, die parallel arbeiten können und man die nötige *Zeit* zu Streuen minimieren möchte. Hier gilt: Ist der Lösungsansatz für diese neue Zielfunktion identisch bis auf die Auswertung der Zielfunktion, so ist die Erweiterung nicht besonders interessant.

1.6 Bewertungskriterien

- Das Lösungsverfahren muss immer einen gültigen Streuplan ausgeben können. Es muss mit jeder sinnvollen Ladekapazität umgehen können, und das Depot soll grundsätzlich in jedem Punkt des Straßennetzes platzierbar sein.
- Die Lösung bzw. deren Umsetzung sollte auch sonst keine grundlegenden Fehler aufweisen.

- Vereinfachungen des Modells sollten erläutert werden. Sollte eine Einschränkung getroffen werden, die das optimale Ergebnis ausschließt, so muss das zumindest erkannt worden sein. Beispiele für solche Einschränkungen sind:
  - Für den ausgegebenen Streuplan gibt es keine Möglichkeit, Schleifenfahrten[3] durchzuführen.
  - Ein Straßenabschnitt wird immer dann gestreut, wenn er das erste Mal durchfahren wird.
  - Es wird immer oder bis auf das letzte Mal die vollständige Ladung Sand verstreut, bevor der Streuwagen zum Depot zurückkehrt.
  - Iterativ wird immer als nächstes der ungestreute Straßenabschnitt gestreut, der am weitesten vom Depot entfernt ist.
- Eine Lösung, die offensichtlich unnötige Wege fährt und auch in einfacheren Fällen (etwa den vorgegebenen Beispielen) nicht den optimalen Streuplan findet, erhält Punktabzug.
- Sollte ein optimaler Algorithmus implementiert worden sein, der auf kleinen Eingaben funktioniert, die Beispiele aufgrund der Größe aber nicht mehr schafft, wird nur leicht abgezogen.
- Der Einfluss der Ladekapazität auf die Schwierigkeit des Problems muss diskutiert worden sein. Dazu genügt es, wenn die Schwierigkeit des Problems in den Randfällen $T = 1$ (effizient lösbar) und $T = m$ (Briefträgerproblem) erkannt worden ist oder die Abhängigkeit des eigenen implementierten Ansatzes von der Ladekapazität aufgezeigt und analysiert wird.
- Für die beiden gegebenen Beispielinstanzen und weitere eigene müssen nachvollziehbare Streupläne angegeben sein.

[3] Fahrten, bei denen ein Straßensegment nicht vollständig durchfahren wird.

Aufgabe 2: Kool

Raus aus dem Kindergarten – rauf aufs Spielbrett Natürlich ist die Vorstellung von Kindern, die glücklich durch ein Kool-Labyrinth toben und dabei ihre Ortungs- und Orientierungssinne schärfen, ganz bezaubernd. Schweren Herzens wollen wir aber hier doch nicht von Labyrinthen und Würfeln sprechen, sondern ein Informatikerinnen und Informatikern vertrauteres Vokabular verwenden. Die Aufgabenstellung einschließlich der Abbildung der Würfel legt bereits nahe, dass es sich bei diesen um ein eigentlich zweidimensionales Phänomen handelt. Da diese „Flachwürfel“ (vulgo: Quadrate) in einem (rechteckigen) Raster zu platzieren sind, drängt sich das Bild eines *Spielbrettes* auf, auf dem wir verschiedenartige *Spielsteine* passend ablegen wollen. Die Eingänge der Würfel (bzw. Spielsteine) wollen wir als *Kontakte* bezeichnen. Damit lassen sich die Bedingungen an eine *gültige Belegung* eines Spielbretts so formulieren: (a) Jeder Kontakt eines Spielsteines muss *geschlossen* sein, also den Kontakt eines anderen Spielsteines berühren; die einzige Ausnahme ist der Kontakt des „Eingangssteins“ links oben, der am Rand des Spielbretts liegen muss. (b) Die Belegung muss *zusammenhängend* sein, d.h. von jedem Stein muss ein Weg zum Eingangsstein führen.

2.1 Spielsteine und Spielbrett

Egal, auf welche Weise wir mögliche Belegungen eines Spielbrettes berechnen wollen: eine ordentliche Repräsentation von Spielsteinen und Spielbrett wird benötigt. Beginnen wir mit den Steinen: Da die Ausrichtung der Steine, wie vorgegeben, eine Rolle spielt, gibt es nicht nur fünf, sondern fünfzehn verschiedene Steinsorten. Eine Steinsorte ist dadurch charakterisiert, an welchen der vier möglichen Stellen – nämlich links, rechts, oben und unten – der Stein einen Kontakt aufweist. Mathematisch könnte ein Stein[4] also als 4-Tupel $[l,r,o,u]$ angegeben werden, mit $l,r,o,u \in \{0,1\}$. Dabei könnte 1 für „Kontakt“ und 0 für „keinen Kontakt“ stehen – oder umgekehrt. Ein kurzer informatischer Blick auf die Situation ergibt, dass jedes Element des Tupels durch ein Bit repräsentiert werden kann. Für eine Implementierung liegt damit die Repräsentation eines Steines als 4-Bit-Feld nahe: Bit 3 (bzw. 2, 1 oder 0) ist gesetzt, wenn der Spielstein links (bzw. rechts, oben oder unten) einen Kontakt hat.

Diese Bitdarstellung kann gleichzeitig auch als ganze Zahl aufgefasst werden. Ein Stein, der einen einzigen Kontakt links (bzw. rechts, oben oder unten) hat, entspricht dann in Bitdarstellung der Zahl 8 (bzw. 4, 2 oder 1). Ein Stein mit mehreren Kontakten lässt sich dann als Summe der „Ein-Kontakt-Steine“ angeben. Ist $L = 8, R = 4, U = 2$ und $O = 1$, entspricht z. B. ein Stein mit Kontakten links, rechts und oben der Zahl $s = L + R + O = 13$. Mit Bit-Operationen lassen sich dann auch manche Tests leicht realisieren: $s\&L > 0$ z. B. prüft, ob der Stein s links einen Kontakt hat. $s = 0$ ist der „leere“ Stein, der für frei bleibende Felder auf dem Spielbrett steht.

Das Spielbrett selbst ist schlicht ein Array mit $m \times n$ Feldern. In jeder Situation ist es mit einer Menge von Steinen belegt, wobei einige oder sogar alle leere Steine sein können. In manchen Verfahren zur Lösung des Labyrinth-Problems werden immer alle Steine auf dem Spielbrett untergebracht und nur die übrigen Plätze mit leeren Steinen aufgefüllt. Bei Verfahren, die nach und nach eine gültige Belegung aufzubauen suchen, liegen einige Steine auf dem Spielbrett,

[4] Im folgenden wird „Stein“ gleichwertig mit „Steinsorte“ verwendet

während der Rest sich noch im Vorrat befindet. Zu einem Spielbrett gehört dann also nicht nur die aktuelle Belegung, sondern auch der aktuell noch verbleibende Vorrat an Steinen. Zu Beginn ist in einem solchen Fall das Spielbrett leer (mit leeren Steinen belegt), und alle (nicht-leeren) Steine befinden sich noch im Vorrat. Dieser kann als ein Feld mit 15 Positionen realisiert werden: eines für jede Steinsorte, in dem festgehalten wird, wie viele Steine der jeweiligen Sorte sich noch im Vorrat befinden.

2.2 Spielbrettgröße

Die meisten Lösungsverfahren funktionieren für gegebene Ausmaße (Länge und Breite) des Spielbretts. Gemäß Aufgabenstellung ist das Spielbrett (also die Größe der Raumecke im Kindergarten) nicht vorgegeben. Deshalb ist es nötig, gültige Belegungen für verschiedene Spielbrettgrößen zu ermitteln. Um den Suchraum klein zu halten, kann man systematisch mit kleinen Spielbrettern anfangen, auf die alle Spielsteine passen. Dabei ist es in Ordnung, Spielbrettgrößen auszuschließen, die allgemein oder auch mit Blick auf den Kontext der Aufgabenstellung unvernünftig erscheinen. Außerdem ist es sinnvoll, größere Felder nur in sehr engen Grenzen zu betrachten; die Aufgabenstellung fordert ja „möglichst kleine Rechteckflächen“ für das Labyrinth.

Für den in der Aufgabenstellung gegebenen Vorrat aus 26 Spielsteinen kann man also zunächst Felder mit den Ausmaßen 3×9, 4×7 und 5×6 betrachten, aber die Größen 1×26 und 2×14 ausschließen (allein das Vorhandensein mindestens eines Kreuzungsstücks erfordert Seitenlängen von mindestens 3). Diese Brettgrößen sind für die gegebene Probleminstanz *minmal* in dem Sinne, dass bei Platzierung aller Steine auf dem Spielbrett keine ganze Zeile oder Spalte des Bretts mehr vollständig frei sein kann. Nur wenn für die minimalen Felder keine gültigen Belegungen gefunden werden, sollten auch größere Felder probiert werden. Es ist legitim, die Brettgrößen in anderer Folge zu untersuchen; etwa könnte zuerst 5×6 (bzw. 6×5) untersucht werden, wenn die Seitenlängen des umfassenden Rechtecks möglichst gleich lang sein sollen.

2.3 Lösungssuche

Wie kann man dieses Problem auffassen? Bei gegebener Spielbrettgröße könnte man hier ein Optimierungsproblem entdecken, das die Anzahl der verbauten Würfel in einer gültigen Belegung maximiert (das wäre eine Verallgemeinerung unserer Aufgabenstellung) oder die offenen Kontakte in einer beliebigen Belegung minimiert (um so überhaupt eine gültige Belegung zu finden).

Allerdings soll nicht nur ein Optimum gefunden werden, sondern mehrere Lösungen. Von daher könnte die richtige Strategie doch die Suche im Lösungsraum sein. Wir schlagen zunächst eine Tiefensuche vor, die – bei gegebener Spielbrettgröße – vom Startfeld aus die Lösung(en) expandiert und aus Sackgassen automatisch zurückkehrt.

Dabei sollte sich die Expansion auf *offene* Felder beschränken; ein offenes Feld ist ein unbelegtes Feld, an dessen Rand mindestens ein Kontakt eines auf einem Nachbarfeld liegenden Spielsteines zu finden ist. Auf diese Weise wird praktisch an das bis dahin schon gebildete

Labyrinth angelegt, und der Suchraum wird – gegenüber einer wirklich vollständigen Suche durch alle Kombinationsmöglichkeiten der Steine – stark eingeschränkt. Hierfür ist es nötig, eine Liste mit den Koordinaten der offenen Felder vorzuhalten und ständig zu aktualisieren. Außerdem wird nur so expandiert, dass alle Zwischenzustände des Spielbretts zumindest potenziell noch zu einer gültigen Belegung erweitert werden können: alle offenen Kontakte zeigen auf offene Felder (können also durch passende Belegung potenziell geschlossen werden), und die aktuelle Belegung ist zusammenhängend.

Verbesserungen

Doch auch eine derart gelenkte Suche bleibt inhärent aufwändig. Für Spielbretter und Steinmengen in der Größenordnung des Beispiels aus der Aufgabenstellung lassen sich Lösungen noch in kurzer Zeit berechnen, aber schon für etwas größere Eingabedaten sind weitere Verbesserungen erforderlich. Hilfreich sind insbesondere Kriterien, mit denen schnell geprüft werden kann, ob die aktuelle, partielle Belegung des Spielbretts noch zu einer gültigen (und möglichst alle Steine verbrauchenden) Belegung erweitert werden kann. Ein wichtiges Abbruchkriterium für die Suche: Sind in mindestens einer der vier Richtungen auf dem Spielbrett mehr offene Kontakte vorhanden als Kontakte der passenden Gegenrichtung im Vorrat der verbliebenen Spielsteine, kann aus dem aktuellen Zustand heraus keine gültige Belegung mehr gefunden werden. Eine weitere Möglichkeit ist die Betrachtung von Teilbereichen des Spielbretts: Gibt es einen kleinen, von Spielsteinen komplett umschlossenen Teilbereich des Spielfeldes, der mindestens ein offenes Feld enthält (mindestens ein offener Kontakt zeigt in den Teilbereich), kann zunächst versucht werden, diesen Teilbereich gültig zu belegen. Ist das nicht möglich, brauchen andere Bereiche des Spielbretts nicht weiter betrachtet werden.

Ein weiterer Aspekt ist eine möglichst geschickte Auswahl der für die Expansion des aktuellen Zustandes genutzten Spielsteine. Ziel dabei ist, den Suchraum zu verkleinern. Auf jeden Fall kann man in jedem Suchschritt prüfen, ob es Felder gibt, für die im Vorrat nur noch ein möglicher Spielstein existiert – und dann die gefundene eindeutige Belegung auch vornehmen.

2.4 Heuristische Ansätze

Mit deutlich größeren Bausteinmengen als im vorgegebenen Beispiel bekommt aber auch eine stark verbesserte Lösungssuche Probleme; die Anzahl der zu probierenden Möglichkeiten wächst einfach zu stark. Eine interessante Alternative kann deshalb der Einsatz eines heuristischen Verfahrens sein. Die Aufgabenstellung hat Ähnlichkeiten zur Aufgabe 3 (Fehlerfrei puzzeln) der ersten Runde. Auch hier ist eine Art Puzzle zu lösen, nur mit anderen Bedingungen. Von daher bietet es sich an, es auch hier mit einem Simulated Annealing bzw. ähnlichen Verfahren mit Zufallskomponenten zu versuchen. Entscheidend sind hierbei die Bewertung (1) von „Zwischenlösungen“, also von Belegungen des Spielbretts und (2) deren Veränderung im Rahmen des Annealing-Prozesses.

Wir gehen wieder davon aus, dass die Spielbrettgröße gegeben und im oben genannten Sinne minimal ist. Nun werden alle Steine zufällig auf dem Spielbrett platziert. Damit erhalten wir eine erste Belegung. Bewertet werden können Belegungen nun mit der Zahl der Fehler, die sie – im Vergleich mit einer gültigen Belegung – enthalten:

offene Kontakte Jeder offene Kontakt eines Spielsteines (der also nicht den Kontakt eines anderen Spielsteines berührt) zählt als eigener Fehler – mit Ausnahme des Eingangssteins, bei dem ein Kontakt nach außen hin offen sein muss.

Zusammenhang Jeder Stein, der nicht vom Eingang aus erreichbar ist, zählt als Fehler.

Enthält die Lösung noch Fehler, muss sie verändert werden. Am einfachsten ist die Vertauschung zweier Steine unterschiedlicher Art (eine Vertauschung zweier gleichartiger Steine ist offensichtlich überflüssig). Für die so erzeugte neue Belegung ist insbesondere die Prüfung der offenen Kontakte schnell erledigt, da nur die acht möglichen Kontaktstellen der vertauschten Steine geprüft werden müssen, um die Veränderung der Fehlerzahl zu bestimmen.

Bei der Erstellung der ersten Belegung und der Veränderung kann darauf geachtet werden, dass die Randplätze des Feldes gleich so belegt werden, dass (bis auf einen Eingang) keine Kontakte nach außen zeigen. Dies wird typischerweise den Verlauf des Annealing beschleunigen. Laufzeitkritisch ist die Prüfung des Zusammenhangs, weshalb sie nur durchgeführt werden sollte, wenn die aktuelle Belegung bzgl. der Kontakte fehlerfrei ist.

Wie viele Einsendungen gezeigt haben, lassen sich mit Simulated Annealing oder vergleichbaren Verfahren Labyrinthe für große Mengen von Spielsteinen finden. Die Zufallskomponente dieser Verfahren sorgt nebenher auch dafür, dass in der Regel bei verschiedenen Aufrufen – wie gefordert – unterschiedliche gültige Belegungen gefunden werden.

Trotz der guten praktischen Erfolge heuristischer Ansätze bei dieser Aufgabe darf nicht vergessen werden, dass die Bestimmung einer gültigen Belegung nicht hundertprozentig sicher ist und der Erfolg von der Wahl geeigneter Parameter abhängt (bei Simulated Annealing etwa die Wahl der Temperaturfunktion oder der Bewertungsfunktion), die im Zweifel durch Ausprobieren zu ermitteln sind.

2.5 Erweiterungen

Wer sagt denn, dass Kool-Würfel quadratisch sein und vier Eingänge haben müssen? Und warum muss ein Kool-Labyrinth möglichst in ein Rechteck passen? Solche Einschränkungen sind mal wieder typisch für Informatiker; in einem Kindergarten sollte es aber doch kreativ und einfallsreich zugehen! Also ran und die Aufgabenstellung erweitern: Für den Einsatz in der Raumecke wäre es z.B. günstig, wenn die Würfel möglichst gut in ein rechtwinkliges Dreieck passen würden, das genau in der Ecke liegt. Interessante Labyrinthe würden sich ergeben, wenn die Würfel sechseckig wären, dementsprechend sechs (oder vielleicht nur drei) Eingänge hätten und in einem Wabenraster platziert werden müssten. Damit ein Labyrinth auch für die Erzieher hübscher aussieht, könnte man ja auch die Oberseiten der Würfel einfärben und daraus interessante (Pixel-)Muster legen – aber dafür würde man wohl recht große Würfelvorräte benötigen ...

2.6 Bewertungskriterien

- Die (interne) Repräsentation von Spielsteinen und Spielbrettern sollte geeignet gewählt sein, also insbesondere die Implementierung der Lösung nicht unnötig kompliziert machen.
- Das vorgeschlagene Verfahren sollte nur gültige Belegungen bzw. Labyrinthe finden: Es gibt genau einen Eingang (der gegenüber der Raumecke liegen soll, aber das ist nicht entscheidend), alle Kontakte bzw. Würfeleingänge sind mit einem anderen Kontakt verbunden, und die Belegung ist zusammenhängend.
- Für einen Steinvorrat sollten mehrere unterschiedliche Lösungen (Labyrinthvorschläge) berechnet werden können.
- Die in den meisten Ansätzen nötige Beschränkung auf bestimmte Spielbrettgrößen sollte sinnvoll und begründet sein.
- Das Lösungsverfahren sollte auch größere Spielbretter und Spielsteinmengen bearbeiten können. Hier sind heuristische Ansätze deutlich im Vorteil vor Backtracking-Verfahren.
- Umso wichtiger ist es für Backtracking-Lösungen, unnötige Fehler zu vermeiden; z.B. sollten ungültige Zwischenzustände ausgeschlossen sein. Glänzt das Backtracking aber mit gutem Pruning, kann dies zum Ausgleich belohnt werden.
- Bei einem heuristischen Ansatz (wie Simulated Annealing) sollten grundsätzliche Nachteile bzw. Schwierigkeiten (keine Optimalitätsgarantie, Abhängigkeit von geeigneter Parameterwahl) benannt sein.
- Außer der vorgegebenen Würfelmenge sollten mindestens drei weitere Beispiele bearbeitet und dargestellt sein.
- Die Darstellung von Würfeln und Labyrinthen sollte einigermaßen anschaulich und übersichtlich sein.

Aufgabe 3: Trickey

Es ist wieder einmal eine ganz normale Trickey-Saison, die unter n Mannschaften augetragen wird, die wir von 1 bis n durchnummerieren. Neugierig und ungeduldig wie wir sind, stellen wir uns nach jedem Spieltag die Frage, welchen Rang unser Favorit $F \in \{1,\ldots,n\}$ noch erreichen kann. Aber der Reihe nach.

3.1 Erste Teilaufgabe: Rangliste

Die erste Teilaufgabe, das Ausgeben der Rangliste, ist schnell erledigt: Nach Einlesen der Eingabe bestimmen wir zu jedem bereits gespielten Spiel die Punkteverteilung, also zu jeder beteiligten Mannschaft die Anzahl an ihr zustehenden Punkten. Nun addieren wir für jede Mannschaft die Anzahl an Punkten sowie die Anzahl an Toren auf, die sie in allen Spielen erreicht hat. Sei für die i-te Mannschaft die Gesamtanzahl an Punkten P_i und die Gesamtanzahl an Toren T_i. Sortieren wir nun die Mannschaften relativ zur lexikografischen Ordnung von (P_i, T_i), so ergibt sich die Rangliste, denn das Hauptkriterium ist die Gesamtanzahl an Punkten, und bei gleicher Punktanzahl wird auf die Anzahl an Toren geschaut.

Achtung ist allerdings noch bei der Vergabe der Ränge geboten. Welchen Rang bekommt eine Mannschaft, die zwei Mannschaften folgt, die sich den ersten Platz teilen? Wir wollen folgende Methode für das Vergeben von Rängen benutzen: Die Mannschaft mit maximalem (P_i, T_i), also die in der Sortierung erste Mannschaft, hat Rang 1. Die $(i+1)$-te Mannschaft in der Sortierung hat den gleichen Rang wie die i-te Mannschaft in der Sortierung, falls sie die gleiche Anzahl an Punkten und Toren erzielt haben. Sonst hat sie den Rang $i+1$. Einfacher ausgedrückt ist der Rang einer Mannschaft i gleich der Anzahl an Mannschaften j mit $(P_i, T_i) < (P_j, T_j)$. Das bedeutet z. B., dass bei zwei Mannschaften, die sich den ersten Rang teilen, kein zweiter Rang mehr vergeben wird, sondern die nachfolgende Mannschaft den dritten Rang erhält. Zur Diskussion eines anderen Verfahrens siehe Abschnitt **??**.

Für die Daten aus `beispiel1.txt` ergibt sich damit folgende initiale Rangliste:

Initiale Rangliste:				
Rang	Name	Punkte	Tore	Ungespielt
1	Mannheim	15	12	3
2	!Muenchen	1	3	3
3	Essen	1	2	4
3	Hamburg	1	2	3
5	Berlin	0	1	5

Die letzte Spalte zeigt die Anzahl noch zu spielender Spiele dieser Mannschaft. Die mit „!“ markierte Mannschaft ist unser Favorit.

3.2 Einfache Einsichten

Aus der detaillierten Beschreibung der Rangvergabe können wir ein paar interessante Schlüsse ziehen:

Relativ zum Favoriten Der Rang des Favoriten F hängt nur von der Anzahl an Mannschaften mit mehr Punkten (oder gleicher Anzahl an Punkten und mehr Toren) ab. Es spielt also keine Rolle, wie zwei andere Mannschaften (verschieden vom Favoriten) zueinander stehen, das heißt, wer mehr Punkte hat oder ob sie gleich viele Punkte haben. Es spielt nur eine Rolle, wie sie relativ zum Favoriten stehen. Das ist eine wichtige Eigenschaft, die jede vernünftige Lösung (implizit) benutzen wird.

Favorit gewinnt alles Der Rang des Favoriten wird maximal, wenn er alle noch zu spielenden Spiele mit 5 Punkten gewinnt. Denn stellen wir uns vor, die Saison wäre beendet und er hätte ein solches Spiel nicht gewonnen. Verändern wir nun den Ausgang dieses einen Spiels so, dass der Favorit 5 Punkte bekommt, so kann sein Rang am Ende der Saison nicht sinken, denn seine Punktzahl steigt, wohingegen die Punktzahlen der anderen Mannschaften gleich bleiben oder sinken. Wir können also annehmen, dass er alle Spiele gewinnt.

Für `beispiel1.txt` sieht die Rangliste folgendermaßen aus, nachdem der Favorit alle Spiele gewonnen hat. Auf die Anzahl an Toren gehen wir später noch ein.

Rangliste:				
Rang	Name	Punkte	Tore	Ungespielt
1	!Muenchen	16	??	0
2	Mannheim	15	12	2
3	Essen	1	2	3
3	Hamburg	1	2	2
5	Berlin	0	1	2

Gewinner gewinnt alles Wir können also die Eingabe nehmen und zusätzlich den Favoriten alle in der Eingabe noch nicht gespielten Spiele gewinnen lassen. Gibt es in der nun aktuellen Rangliste eine Mannschaft i vor dem Favoriten, so wird diese Mannschaft auch am Ende der Saison vor dem Favoriten stehen, denn der Favorit bekommt keine Punkte mehr, und Mannschaft i kann keine verlieren. Wir können also Mannschaft i alle kommenden Spiele mit 5 Punkten gewinnen lassen; so bekommen Mannschaften, die hinter dem Favoriten liegen, keine Punkte, während die Punkte von Mannschaft i ohnehin keine Rolle mehr spielen. Dies kommt allerdings in den vorgegebenen Beispielen nicht vor.

Favorit bekommt ∞ Tore Wir nehmen uns ein zu Beginn noch zu spielendes Spiel des Favoriten und stellen uns vor, die Saison wäre beendet. Erhöhen wir die Anzahl an Toren, die der Favorit in diesem Spiel erzielt, so kann sein Rang nicht sinken. Da er in diesem Spiel beliebig viele Tore schießen kann und wir sehen wollen, welchen Rang der Favorit maximal erreichen kann, ist es sinnvoll, ihn in diesem Spiel so viele Tore schießen zu lassen, wie keine andere Mannschaft erzielen kann. Symbolisch schreiben wir, dass der Favorit ∞ Tore schießt.

Tore spielen keine Rolle Beim Rangvergleich wird der Favorit F nun bei gleicher Punktzahl immer vorne liegen. Wir nehmen uns also wieder ein noch zu spielendes Spiel einer Mannschaft $i \neq F$ und stellen uns vor, die Saison wäre beendet. Sagen wir, wir verändern die Anzahl an Toren, die Mannschaft i in diesem Spiel schießt, aber so, dass die Punkteverteilung gleich bleibt. Dann verändert sich der Rang des Favoriten nicht, denn in jedem Vergleich zwischen F und einer anderen Mannschaft j sind die Punktzahlen P_F und P_j unverändert, und die Torsumme $T_F = \infty > T_j$.

Insgesamt spielt es also keine Rolle, wie viele Tore die Mannschaften $i \neq F$ erzielen, allein die Punkteverteilungen der noch zu spielenden Spiele sind von Belang.

Ein Randfall: endliche Toranzahl Die Argumentation der beiden obigen Absätze funktioniert nur unter einer Annahme (die in den vorgegebenen Beispielen erfüllt ist): Der Favorit muss noch mindestens ein ungespieltes Spiel haben. Hat er bereits zu Beginn alle Spiele gespielt, so können wir ihm keine ∞ Tore geben. Um diesen Randfall abzudecken, müssen wir anders argumentieren: Nehmen wir uns ein noch zu spielendes Spiel und stellen uns vor, die Saison wäre beendet. Nun verändern wir die Tore, die die beteiligten Mannschaften $A : B : C$ in diesem Spiel erzielen, und zwar so, dass sie minimal sind, während die Punkteverteilung gleich bleibt. Zur Punkteverteilung $5 : 0 : 0$ (A bekommt 5 Punkte, B und C bekommen 0 Punkte) ist beispielsweise die Torverteilung $1 : 0 : 0$ minimal, zur Punkteverteilung $2 : 2 : 0$ ist es die Torverteilung $1 : 1 : 0$ und zur Punkteverteilung $1 : 1 : 1$ ist es die Torverteilung $0 : 0 : 0$. Durch diese Veränderung kann sich der Rang des Favoriten nicht verschlechtern.

Wir können also annehmen, dass alle Torverteilungen wie oben gebildet werden. Damit kommt es auch in diesem Randfall nicht auf die erzielten Tore, sondern nur auf die erzielten Punkte an; die Torverteilung ergibt sich aus der Punkteverteilung.

3.3 Greedy-Algorithmus

Es ist nun klar, dass wir direkt den Favoriten alle noch zu spielenden Spiele gewinnen lassen können, und jede Mannschaft, die dann noch vor ihm liegt, alle Spiele gewinnen lassen können.

Für die dann noch offenen Spiele kann man sich leicht verschiedene „Greedy“-Strategien überlegen. Eine solche Strategie geht alle noch offenen Spiele durch und versucht direkt jedem Spiel einen Ausgang so zuzuordnen, dass der Rang des Favoriten möglichst klein bleibt. Beispielsweise könnte man zu so einem Spiel zu jeder beteiligten Mannschaft i die Punkte Δ_i berechnen, die sie noch bekommen darf, ohne den Favoriten zu überholen, und die Anzahl an noch offenen Partien S_i jeder beteiligten Mannschaft. Im Schnitt darf Mannschaft i in jedem ihrer S_i offenen Spiele Δ_i/S_i Punkte machen, ohne den Favoriten zu überholen.

Als einfache Variante nimmt man sich nun eine beteiligte Mannschaft, deren Δ_i/S_i maximal ist, und lässt sie gewinnen, das heißt sie erhält 5 Punkte. Intelligenter wäre es, zu einem Spiel $A : B : C$ den Vektor $v = (\Delta_A/S_A, \Delta_B/S_B, \Delta_C/S_C)$ und die Punkteverteilung (p_A, p_B, p_C) möglichst gut zu matchen. Man könnte beispielsweise diejenige der 7 möglichen Punktverteilungen nehmen, die den quadratischen Abstand zu v minimiert.

Verblüffenderweise reicht die einfache Variante dieses Greedy-Algorithmus bereits aus, um alle 6 mitgelieferten Beispiele optimal zu lösen! In allen Beispielen ist der erste Rang erreichbar und wird auch erreicht, wenn man die einfache Greedy-Strategie benutzt.

Es sollte aber klar sein, dass ein Greedy-Algorithmus nicht in allen Situationen optimale Ergebnisse liefert. Ein schweres generiertes Beispiel werden wir später sehen. Wenden wir uns nun exakten Algorithmen zu, die immer den maximalen Rang liefern.

3.4 Backtracking

Der finale Rang einer Mannschaft kann von den Ausgängen aller noch zu spielenden Spiele abhängen. Ein einfacher Ansatz wäre also, alle Ausgänge der noch zu spielenden Spiele durchzuprobieren, für jede solche Wahl den Rang des Favoriten zu berechnen und den maximalen Rang auszugeben, den man dabei entdeckt. Wir wissen ja bereits, dass wir für den Ausgang eines Spiels nur alle möglichen Punkteverteilungen probieren müssen, die Anzahl an Toren ist dann beliebig oder festgelegt. Problematisch an diesem Ansatz ist, dass jedes Spiel einen von 7 verschiedenen Ausgängen nehmen kann: Die möglichen Punktverteilungen sind $5:0:0$, $0:5:0$, $0:0:5$, $2:2:0$, $2:0:2$, $0:2:2$, $1:1:1$. Da eine Saison aus $\binom{n}{3}$ Spielen besteht, ergibt sich ein Maximalaufwand von $7^{\binom{n}{3}}$ Varianten. Alle Varianten durchzuprobieren ist also mehr als inpraktikabel.

Die Laufzeit verringert sich, wenn man das Spiel, dessen Ausgang man rät, intelligenter wählt. Eine Möglichkeit ist, das Spiel zu nehmen, bei dem die Summe der Punkte der beteiligten Mannschaften maximal ist. In dem Fall kommt es bei vielen Spielausgängen dazu, dass eine Mannschaft den Favoriten überholt, wonach man wiederum die Regeln anwenden kann, dass eine solche Mannschaft jedes zukünftige Spiel gewinnen sollte.

Dies alleine reicht aber noch nicht. Es braucht weitere Überlegungen, um damit die Beispiele und härtere Instanzen lösen zu können. Diese folgen in den nächsten Abschnitten.

3.5 Überholer raten

Wir nehmen an, dass wir unsere Vorüberlegungen bereits angewandt haben: Der Favorit gewinnt alle seine kommenden Spiele, und jede Mannschaft, die dann noch vor dem Favoriten liegt, gewinnt auch alle Spiele. Offene Spiele finden also nur zwischen Mannschaften statt, die nun hinter dem Favoriten liegen (oder gleichauf).

Wenn der Favorit seinen aktuellen Platz halten kann, dann gibt es keine Überholer, also Mannschaften, die im aktuellen Plan hinter dem Favoriten (oder gleichauf) und am Ende der Saison vor ihm liegen. Wenn er seinen Rang nicht halten kann, sondern um genau einen Rang absteigt, so gibt es genau einen Überholer, usw. Im Allgemeinen gibt es eine Menge U von Mannschaften, die den Favoriten bis zum Ende der Saison überholt haben. Wir können annehmen, dass jede Mannschaft in U jedes anfangs offene Spiel gewinnt (genauer: in einem solchen Spiel bekommt keine Mannschaft $i \notin U$ einen Punkt), denn wie viele Punkte eine Mannschaft $u \in U$ genau bekommt spielt keine Rolle, und auf diese Weise werden die Punkte der anderen Mannschaften minimiert.

Um das Problem zu lösen gehen wir nun wie folgt vor: Wir gehen alle Möglichkeiten für U durch, das sind höchstens 2^{n-1} viele. Alle Mannschaften in U gewinnen alle kommenden Spiele, wie beschrieben. Nun müssen wir testen, ob der Favorit seinen Rang halten kann: Können die restlichen Spiele so ausgehen, dass die Mannschaften, die bisher hinter dem Favoriten sind, auch wirklich hinter ihm bleiben? Wir nennen diesen Test *RangHalten* (man beachte: Mannschaften, die am Ende der Saison vor dem Favoriten liegen sollen, sind an keinem noch offenen Spiel beteiligt).

Nachdem wir nun U haben, wollen wir Backtracking benutzen, um die noch offenen Spielausgänge durchzuprobieren. Am Ende nehmen wir die kleinste Menge U, für die RangHalten klappt; diese liefert den maximal erreichbaren Rang.

Dieses Backtracking ist natürlich noch immer zu langsam. Deshalb geben wir in den nächsten zwei Abschnitten schnelle Heuristiken an, die in jedem rekursiven Aufruf der Backtracking-Prozedur aufgerufen werden sollten. Diese Heuristiken sind exakt (wir finden in jedem Fall eine optimale Lösung; die Heuristiken liegen nie falsch), aber nicht immer anwendbar. Das heißt, es gibt Instanzen von RangHalten, auf denen beide Heuristiken „Ich weiß nicht“ zurückliefern. In diesem Fall müssen wir weiter backtracken; die Heuristiken schlagen jedoch häufig an, womit wir dann mit diesem Teilbaum fertig sind.

Positive Heuristik für RangHalten Um schnell zu sehen, dass der Favorit seinen Rang halten kann, nutzen wir einfach den Greedy-Algorithmus aus Abschnitt 3.3. Verändert sich der Rang des Favoriten nach der Greedy-Zuweisung der restlichen Spiele nicht, so kann der Favorit seinen Rang halten, wir sind also fertig.

Negative Heuristik für RangHalten Zudem gibt es eine einfache Heuristik, um schnell zu sehen, dass die übrigen Spiele nicht so ausgehen können, dass der Favorit seinen Rang hält. Dazu berechnen wir einerseits die Anzahl S an noch ausstehenden Spielen. Andererseits sei M die Menge an Mannschaften, die noch nicht alle Spiele hinter sich haben. Mannschaft $i \in M$ darf in ihren ausstehenden Spielen insgesamt maximal $P_F - P_i$ Punkte bekommen, damit sie noch hinter dem Favoriten bleibt. Insgesamt dürfen die Mannschaften in M also noch $D := \sum_{i \in M} P_F - P_i$ Punkte bekommen. In jedem Spiel werden aber mindestens 3 Punkte vergeben, insgesamt werden also noch mindestens $3 \cdot S$ Punkte vergeben. Dass der Favorit seinen Rang hält kann also nur passieren, wenn $3 \cdot S \leq D$ gilt; gilt dies nicht, so können wir RangHalten sofort negativ beantworten.

3.6 Härtere Instanzen

Die mitgelieferten Beispiele werden bereits von einer Greedy-Strategie gelöst. Um zu demonstrieren, dass unser Backtracking-Ansatz mit Heuristiken gut ist, müssen wir also schwierigere Instanzen erzeugen. Es stellt sich heraus, dass das gar nicht so einfach ist. Zufällige Instanzen, nach verschiedenen Schemata generiert, sind in den meisten Fällen schon mit dem Greedy-Ansatz lösbar. Sind sie es nicht, so sind sie meistens mit dem Überholer-Raten-Ansatz lösbar, ohne auf Backtracking zurückgreifen zu müssen (d.h. die Heuristiken schlagen an). Nur sehr selten, und mit den richtigen Instanzenschemata, wird Backtracking benötigt, aber auch dann

ist obiger Algorithmus sehr schnell. Wir haben keine Instanz mit 10 Mannschaften gefunden, auf der er mehr als 10s gebraucht hätte; auf eine empirische Laufzeitanalyse wird verzichtet.

Um härtere Instanzen zu erzeugen zu gegebenem n und Favoriten $F = 1$, ordnen wir die $\binom{n}{3}$ Spiele zufällig an; die ersten $\alpha \approx 80\%$ von ihnen sollen schon gespielt sein. In jedem bisherigen Spiel hat der Favorit fast immer kein einziges Tor geschossen und jede andere Mannschaft ungefähr ein Tor. Hier kann man z.B. Poisson-Zufallsvariablen einfügen. Der Parameter α ist ungefähr so gewählt, dass der Favorit Rang 1 hat, wenn er nun alle offenen Spiele gewinnt, aber nur knapp vorne liegt, während die anderen Mannschaften noch viele ungespielte Spiele haben. Nun kann es sein, dass der Favorit seinen Rang 1 halten kann, es kann aber auch sein, dass man Überholer braucht. Beide Fälle werden oft schon von den Heuristiken erkannt, manchmal aber nicht. Auf diese Art sind die Beispieleingaben in Abschnitt 3.10 entstanden.

3.7 Erweiterungen

Alle erreichbaren Ränge: Wenn man durch obiges Programm den maximalen Rang kennt, den eine Mannschaft noch erreichen kann, so stellt sich die Frage danach, welche anderen Ränge noch erreichbar sind. Insbesondere stellt sich die Frage nach dem minimal erreichbaren Rang, dies könnte sehr ähnlich zur eigentlichen Fragestellung gelöst werden. Besonders für Zwischenränge stellt dies eine interessante Erweiterung dar.

Mannschaftspaare: Angenommen man verfolgt das Schicksal von gleich zwei Lieblingsmannschaften: Welchen Ränge können beide noch erreichen? Hier werden die Abhängigkeiten der Spielergebnisse voneinander um einiges komplexer.

Das verallgemeinerte 40-Punkte-Problem: Wie viele Punkte muss eine Mannschaft noch erhalten, damit sie auf jeden Fall einen bestimmten Platz erreicht? [5]

3.8 Bewertungskriterien

- Teil 1: Die Rangliste wird korrekt (und beschreibungsgemäß) berechnet.
- Die offensichtlichen Vereinfachungen bei der Suche nach den optimalen Spielergebnissen (Favorit gewinnt alle Spiele mit vielen Toren, danach noch höher rangierende Teams gewinnen ebenfalls alle Spiele) sollten erkannt worden sein.
- Die Lösung muss auch in speziellen Fällen korrekt arbeiten, nämlich wenn
  - der Favorit keine Spiele mehr zu spielen hat, die Anzahl geschossener Tore also entscheidend sein kann; oder
  - der Favorit nicht den ersten Rang erreichen kann.

[5] Eine (in der Theorie nicht ganz richtige) Faustregel der Fußball-Bundesliga lautet: Erreicht eine Mannschaft mindestens 40 Punkte, so kann sie nicht mehr absteigen. Siehe beispielsweise http://www.mathematik.de/ger/presse/ausdenmitteilungen/artikel/mdmv-19-3-153.pdf

- Schwächen von Greedy-Ansätzen bzw. Heuristiken sollten erkannt worden sein.
- Eine gesichert optimale Lösung kann bei speziellen Eingaben hohen Berechnungsaufwand erfordern, ist aber grundsätzlich für übliche Ligagrößen gut brauchbar. Ein Ansatz mit einem Durchprobieren der Überholer ist nicht erforderlich, sollte aber positiv bewertet werden.
- Der Algorithmus soll das Ergebnis auf den Beispieleingaben schnell (und korrekt) lösen können. Da diese durch einfache Beobachtungen effizient gelöst werden können, sind mehrere Minuten Berechnungszeit unnötig.
- Die Lösung muss nicht nur den besten erreichbaren Rang berechnen, sondern auch die passenden Ausgänge der offenen Spiele. Zumindest eine Liste mit passenden Ergebnissen muss ausgegeben werden.
- Es muss mindestens ein Beispiel angegeben sein, bei dem der Favorit nicht Rang 1 erreichen kann. Außerdem sollte mindestens ein Beispiel angegeben sein, bei dem der Favorit keine Spiele mehr zu spielen hat.

3.9 Vorgegebene Beispiele

Hier geben wir für die vorgegebenen Beispiele je eine optimale Lösung an. Mit Ausname von `beispiel1.txt` sind aus Platzgründen nur die initialen und finalen Ranglisten dargestellt.

beispiel1.txt

Eingabe:
5
Berlin Muenchen Mannheim Essen Hamburg
Muenchen
Muenchen : Mannheim : Essen - 0 : 2 : 1 Berlin : Mannheim : Hamburg - 1 : 7 : 0 Muenchen : Mannheim : Hamburg - 2 : 3 : 1 Muenchen : Essen : Hamburg - 1 : 1 : 1

Ausgang aller Spiele (ohne Eingabe):
Berlin : Muenchen : Mannheim - 0 : ∞ : 0 Berlin : Muenchen : Essen - 0 : ∞ : 0 Berlin : Muenchen : Hamburg - 0 : ∞ : 0 Berlin : Mannheim : Essen - 1 : 0 : 0 Berlin : Essen : Hamburg - 1 : 0 : 0 Mannheim : Essen : Hamburg - 0 : 1 : 0

Initiale Rangliste:			
Rang	Name	Punkte	Tore
1	Mannheim	15	12
2	!Muenchen	1	3
3	Essen	1	2
3	Hamburg	1	2
5	Berlin	0	1

Endgültige Rangliste:			
Rang	Name	Punkte	Tore
1	!Muenchen	16	∞
2	Mannheim	15	12
3	Berlin	10	3
4	Essen	6	3
5	Hamburg	1	2

beispiel2.txt

Initiale Rangliste:			
Rang	Name	Punkte	Tore
1	Muenchen	22	40
2	Essen	16	28
3	Hamburg	13	30
4	Berlin	4	10
5	Lichterfelde	2	2
6	!Harvestehude	1	30
7	Koeln	0	0

Endgültige Rangliste:			
Rang	Name	Punkte	Tore
1	!Harvestehude	36	∞
2	Berlin	34	16
3	Koeln	30	6
4	Muenchen	22	40
5	Lichterfelde	17	5
6	Essen	16	28
7	Hamburg	13	30

beispiel3.txt

Initiale Rangliste:			
Rang	Name	Punkte	Tore
1	D	106	179
2	K	95	135
3	L	82	117
4	C	33	55
5	J	30	65
6	G	20	58
7	H	18	50
8	E	17	63
9	I	17	52
10	F	17	50
11	!B	0	50
12	A	0	26

Endgültige Rangliste:			
Rang	Name	Punkte	Tore
1	!B	115	∞
2	D	111	180
3	C	103	69
4	E	97	79
5	K	95	135
6	A	95	45
7	G	85	71
8	H	83	63
9	L	82	117
10	F	82	63
11	J	60	71
12	I	57	60

beispiel4.txt

Initiale Rangliste:			
Rang	Name	Punkte	Tore
1	Muelheim	38	46
2	Mannheim	12	13
3	!Muenchen	5	12
4	Essen	2	4
4	Harvestehude	2	4
6	Berlin	0	5
7	Lichterfelde	0	4

Endgültige Rangliste:			
Rang	Name	Punkte	Tore
1	!Muenchen	45	∞
2	Muelheim	38	46
3	Essen	37	11
4	Berlin	20	9
5	Mannheim	12	13
6	Harvestehude	12	6
7	Lichterfelde	5	5

beispiel5.txt

Initiale Rangliste:			
Rang	Name	Punkte	Tore
1	Muelheim	30	47
2	Mannheim	26	30
3	!Muenchen	2	12
4	Berlin	2	11
5	Essen	2	4
5	Harvestehude	2	4
7	Lichterfelde	1	7

Endgültige Rangliste:			
Rang	Name	Punkte	Tore
1	!Muenchen	32	∞
2	Mannheim	31	31
3	Muelheim	30	47
4	Essen	27	9
5	Berlin	22	15
6	Harvestehude	12	6
7	Lichterfelde	11	9

beispiel6.txt

Initiale Rangliste:			
Rang	Name	Punkte	Tore
1	Muelheim	35	151
2	Mannheim	35	53
3	Hamburg	25	29
4	Harvestehude	13	16
5	Berlin	7	11
6	!Muenchen	4	12
7	Lichterfelde	4	11
8	Essen	4	6

Endgültige Rangliste:			
Rang	Name	Punkte	Tore
1	!Muenchen	49	∞
2	Essen	39	13
3	Muelheim	35	151
4	Mannheim	35	53
5	Berlin	32	16
6	Hamburg	30	30
7	Harvestehude	23	18
8	Lichterfelde	19	14

3.10 Weitere Beispieleingaben

Dies sind weitere, interessantere Beispiele.

In allen Beispielen gibt es die sieben Mannschaften M1, M2, ..., M7, und M1 ist der Favorit. Nachdem der Favorit alle Spiele gewonnen hat, ist er in jedem Beispiel auf Rang 1. Allerdings kann er diesen nicht in allen Beispielen halten.

beispiel-greedy-fails.txt: Dies ist ein Beispiel, bei dem Rang 1 erreichbar ist, aber die beschriebenen Greedy-Strategien dies nicht herausfinden. Es kann also möglicherweise Algorithmen enttarnen, die nicht den maximalen Rang berechnen.

Eingabe:
7
M1 M2 M3 M4 M5 M6 M7
M1
M3 : M4 : M6 - 1 : 1 : 2 M1 : M4 : M7 - 0 : 0 : 2 M3 : M4 : M7 - 0 : 0 : 0 M1 : M2 : M4 - 0 : 0 : 0 M1 : M2 : M5 - 0 : 0 : 0 M1 : M2 : M7 - 0 : 0 : 0 M1 : M3 : M4 - 0 : 1 : 0 M1 : M2 : M3 - 0 : 1 : 2 M1 : M3 : M6 - 0 : 0 : 0 M1 : M3 : M5 - 0 : 0 : 1 M3 : M5 : M7 - 1 : 1 : 0 M1 : M3 : M7 - 0 : 0 : 0 M2 : M3 : M4 - 0 : 0 : 0 M2 : M3 : M5 - 0 : 0 : 0 M2 : M5 : M6 - 0 : 0 : 1 M5 : M6 : M7 - 0 : 0 : 0 M2 : M3 : M6 - 1 : 0 : 0 M2 : M3 : M7 - 1 : 0 : 0 M1 : M4 : M6 - 0 : 1 : 0 M2 : M4 : M6 - 0 : 0 : 1 M4 : M6 : M7 - 2 : 0 : 1 M2 : M4 : M5 - 0 : 2 : 0 M4 : M5 : M6 - 0 : 0 : 0 M1 : M5 : M7 - 0 : 0 : 0 M1 : M6 : M7 - 0 : 1 : 1

Initiale Rangliste:			
Rang	Name	Punkte	Tore
1	M6	20	5
2	M4	19	6
3	M3	17	5
4	M2	15	3
5	M7	12	4
6	M5	12	2
7	!M1	6	0

Endgültige Rangliste:			
Rang	Name	Punkte	Tore
1	!M1	21	∞
2	M7	21	8
3	M3	21	6
4	M6	21	5
5	M4	20	6
6	M2	20	5
7	M5	17	3

Lösung (ohne Eingabe):
M1 : M2 : M6 - ∞ : 0 : 0 M1 : M4 : M5 - ∞ : 0 : 0 M1 : M5 : M6 - ∞ : 0 : 0 M2 : M4 : M7 - 1 : 0 : 1 M2 : M5 : M7 - 0 : 0 : 0 M2 : M6 : M7 - 1 : 0 : 1 M3 : M4 : M5 - 0 : 0 : 0 M3 : M5 : M6 - 0 : 0 : 0 M3 : M6 : M7 - 1 : 0 : 1 M4 : M5 : M7 - 0 : 1 : 1

beispiel-zweiter-rang-heuristik.txt Hier ist nur Rang 2 erreichbar, was schon eine Heuristik herausfindet.

Eingabe:
7
M1 M2 M3 M4 M5 M6 M7
M1
M3 : M4 : M6 - 0 : 2 : 1 M1 : M5 : M6 - 0 : 1 : 2 M1 : M6 : M7 - 0 : 0 : 0 M1 : M2 : M4 - 0 : 0 : 0 M1 : M2 : M5 - 0 : 1 : 1 M1 : M2 : M6 - 0 : 0 : 0 M1 : M2 : M7 - 0 : 0 : 1 M1 : M3 : M6 - 0 : 0 : 0 M1 : M3 : M5 - 0 : 1 : 0 M1 : M3 : M7 - 0 : 0 : 0 M3 : M4 : M5 - 0 : 2 : 1 M3 : M5 : M7 - 0 : 1 : 0 M2 : M3 : M4 - 1 : 1 : 1 M1 : M4 : M7 - 0 : 0 : 1 M2 : M3 : M7 - 0 : 1 : 1 M2 : M5 : M6 - 0 : 0 : 1 M2 : M3 : M6 - 0 : 0 : 0 M1 : M4 : M5 - 0 : 0 : 1 M2 : M4 : M7 - 1 : 1 : 1 M1 : M4 : M6 - 0 : 1 : 0 M2 : M6 : M7 - 1 : 0 : 2 M2 : M4 : M6 - 0 : 0 : 2 M2 : M4 : M5 - 0 : 0 : 0 M4 : M5 : M6 - 0 : 1 : 0 M2 : M5 : M7 - 2 : 1 : 0

Initiale Rangliste:			
Rang	Name	Punkte	Tore
1	M7	20	6
2	M4	19	7
3	M6	19	6
4	M5	18	7
5	M2	13	6
6	M3	11	3
7	!M1	5	0

Endgültige Rangliste:			
Rang	Name	Punkte	Tore
1	M7	45	11
2	!M1	20	∞
3	M4	19	7
4	M6	19	6
5	M2	18	7
5	M5	18	7
7	M3	16	4

Lösung (ohne Eingabe):
M1 : M2 : M3 - ∞ : 0 : 0 M1 : M3 : M4 - ∞ : 0 : 0 M1 : M5 : M7 - ∞ : 0 : 0 M2 : M3 : M5 - 1 : 0 : 0 M3 : M4 : M7 - 0 : 0 : 1 M3 : M5 : M6 - 1 : 0 : 0 M3 : M6 : M7 - 0 : 0 : 1 M4 : M5 : M7 - 0 : 0 : 1 M4 : M6 : M7 - 0 : 0 : 1 M5 : M6 : M7 - 0 : 0 : 1

beispiel-zweiter-rang-backtracking.txt Hier ist nur Rang 2 erreichbar, aber ob Rang 1 erreichbar ist, ist keine einfache Frage. Die Heuristik hilft dabei nicht.

Eingabe:
7
M1 M2 M3 M4 M5 M6 M7
M1
M4 : M5 : M7 - 0 : 1 : 0 M3 : M4 : M6 - 0 : 1 : 0 M1 : M5 : M6 - 0 : 2 : 0 M1 : M4 : M7 - 0 : 0 : 1 M1 : M2 : M3 - 0 : 1 : 0 M1 : M2 : M4 - 0 : 0 : 0 M1 : M2 : M6 - 0 : 2 : 0 M1 : M2 : M7 - 0 : 3 : 0 M1 : M3 : M4 - 0 : 0 : 0 M3 : M4 : M7 - 0 : 0 : 1 M1 : M3 : M6 - 0 : 1 : 0 M3 : M5 : M6 - 2 : 1 : 0 M3 : M5 : M7 - 1 : 2 : 1 M2 : M3 : M4 - 0 : 0 : 0 M3 : M6 : M7 - 0 : 1 : 1 M2 : M5 : M6 - 0 : 0 : 0 M2 : M3 : M6 - 0 : 1 : 0 M1 : M4 : M5 - 0 : 3 : 2 M2 : M4 : M7 - 2 : 0 : 1 M2 : M6 : M7 - 0 : 1 : 0 M2 : M5 : M7 - 0 : 0 : 1 M4 : M5 : M6 - 3 : 1 : 0 M3 : M4 : M5 - 1 : 1 : 1 M1 : M5 : M7 - 0 : 0 : 1 M1 : M6 : M7 - 0 : 0 : 0

Initiale Rangliste:			
Rang	Name	Punkte	Tore
1	M2	23	8
2	M7	23	7
3	M4	19	8
4	M3	18	6
5	M5	17	10
6	M6	9	2
7	!M1	3	0

Endgültige Rangliste:			
Rang	Name	Punkte	Tore
1	M2	43	12
2	!M1	23	∞
3	M7	23	7
4	M5	22	11
5	M4	19	8
6	M3	18	6
7	M6	14	3

Lösung (ohne Eingabe):
M1 : M2 : M5 - ∞ : 0 : 0 M1 : M3 : M5 - ∞ : 0 : 0 M1 : M3 : M7 - ∞ : 0 : 0 M1 : M4 : M6 - ∞ : 0 : 0 M2 : M3 : M5 - 1 : 0 : 0 M2 : M3 : M7 - 1 : 0 : 0 M2 : M4 : M5 - 1 : 0 : 0 M2 : M4 : M6 - 1 : 0 : 0 M4 : M6 : M7 - 0 : 1 : 0 M5 : M6 : M7 - 1 : 0 : 0

beispiel-endliche-tore.txt Dies ist ein Beispiel, bei dem der Favorit bereits in der Eingabe alle Spiele gespielt hat, bei dem es also auch auf die Anzahl der Tore ankommt.

Eingabe:
7
M1 M2 M3 M4 M5 M6 M7
M1
M3 : M4 : M6 - 0 : 1 : 0 M1 : M5 : M6 - 0 : 0 : 1 M1 : M4 : M7 - 0 : 1 : 0 M1 : M6 : M7 - 0 : 0 : 1 M1 : M2 : M4 - 1 : 1 : 1 M1 : M2 : M5 - 0 : 0 : 0 M1 : M2 : M6 - 0 : 0 : 0 M1 : M2 : M7 - 2 : 0 : 0 M1 : M3 : M4 - 1 : 0 : 0 M1 : M2 : M3 - 0 : 0 : 1 M3 : M5 : M6 - 0 : 0 : 0 M1 : M3 : M5 - 1 : 0 : 0 M3 : M5 : M7 - 0 : 1 : 1 M1 : M3 : M6 - 0 : 0 : 3 M1 : M3 : M7 - 0 : 0 : 1 M2 : M3 : M4 - 1 : 0 : 0 M3 : M6 : M7 - 0 : 0 : 0 M1 : M4 : M5 - 1 : 0 : 1 M2 : M5 : M6 - 1 : 1 : 0 M5 : M6 : M7 - 0 : 0 : 0 M2 : M3 : M6 - 0 : 1 : 0 M2 : M3 : M7 - 0 : 1 : 0 M2 : M4 : M7 - 0 : 0 : 1 M1 : M4 : M6 - 0 : 1 : 0 M2 : M4 : M6 - 0 : 0 : 0 M4 : M6 : M7 - 0 : 0 : 0 M2 : M4 : M5 - 1 : 0 : 3 M1 : M5 : M7 - 0 : 3 : 0

Initiale Rangliste:			
Rang	Name	Punkte	Tore
1	!M1	20	6
2	M7	20	4
3	M5	19	9
4	M4	18	4
5	M3	17	3
6	M6	16	4
7	M2	11	4

Endgültige Rangliste:			
Rang	Name	Punkte	Tore
1	M5	44	14
2	!M1	20	6
3	M4	20	5
4	M7	20	4
5	M3	19	4
6	M2	16	5
7	M6	16	4

Lösung (ohne Eingabe):
M2 : M3 : M5 - 0 : 0 : 1 M2 : M5 : M7 - 0 : 1 : 0 M2 : M6 : M7 - 1 : 0 : 0 M3 : M4 : M5 - 0 : 0 : 1 M3 : M4 : M7 - 1 : 1 : 0 M4 : M5 : M6 - 0 : 1 : 0 M4 : M5 : M7 - 0 : 1 : 0

Perlen der Informatik – aus den Einsendungen

Allgemeines

Worte des Wettbewerbs: boolanische Variable, Indizien *(als Plural von „Index“)*

Eine Möglichkeit besteht darin, alle möglichen Kombinationen auszuprobieren. Hierdurch würde man ein genaueres Ergebnis mit allen möglichen Kombinationsmöglichkeiten erhalten.

Die unmöglichen Möglichkeiten werden ignoriert.

Beide Teilprobleme beinhalten also jeweils eine nichtdeterministische Komponente und gehören somit zur Klasse der NP-vollständigen Probleme.

Somit gehört die Aufgabe zur Klasse der NP-vollständigen Probleme, die nicht in polynomieller Laufzeit deterministisch bzw. optimal gelöst werden können. *Sollten wir wirklich die Lösung der $P = NP$ Frage verpasst haben?*

Aufgabe 1: Ned

Da ich per Hand keine Lösung gefunden habe, die mit weniger als 60km auskommt, kann es eine solche nicht geben.

Diese Klasse stellt ein Auto dar, welches den Algorithmus enthält.

Fahren sie auf Holzweg und folgen sie dem Straßenverlauf 3 km.

Je größer jedoch die Ladekapazität, desto größer ist auch die Fehlerkonstante.

Dieser Lösungsweg ist natürlich noch nicht optimal, da er auf Grund des Zufalls auch eine zufällige Laufzeit hat.

Von einem beliebigen Knoten wird sich über seine Kante von ihm entfernt.

Aufgabe 2: Kool

Schnell fällt auf, dass, falls man nicht alle Würfel verwendet, sehr einfach extrem platzsparende Lösungen aus nur einem Teil erzeugt werden könnten.

. . . und von diesen aus iterativ bzw. rekursiv alle weiteren Felder angesetzt werden.
Wie denn nun?

Da die Anzahl der Blöcke sowie deren Beschaffenheit frei wählbar sind, ist die Laufzeit nicht konstant.

Bei größeren Bausätzen braucht der Kindergarten entweder Geduld oder einen Großrechner.

Aufgabe 3: Trickey

Ich gehe davon aus, dass [. . .] es nicht zu irgendwelchen Sportgerichtsurteilen kommt, sodass allen drei Mannschaften keine Punkte gegeben werden.

Der letztendliche Speicherverbrauch liegt im Gutdünken des Garbage Collectors und dessen Arbeitsmoral.

Leider öffnete ich die Dateien in einem ganz normalen Editor, sodass sie alle in einer Zeile erschienen. Mein erster Einlesevorgang beschäftigte sich also damit, die eine Zeile „auseinander zu pfriemeln“. Später öffnete ich die Datei aus Versehen in Word und sah zum großen Erschrecken, dass da ja Absätze waren.

`trickey_gui.py` stellt ein GUI (für das ich keine Extrapunkte verlange!!!) zur Verfügung.

Es kann die Betrachtung der Tore außer Betracht gelassen werden.

Für die Berechnung der Tabelle sind keine informationstechnischen, theoretischen Betrachtungen notwendig. Es handelt sich hierbei um reines Anwenden der jeweiligen Programmiersprache.

Das heißt also, dass alle schlechten Mannschaften plötzlich die Besten sind.

Eine Mannschaft ist ein Gebilde aus Objekten. Die Beschaffenheit und Anzahl dieser Objekte ist dabei irrelevant.