34. 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 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 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.

Bewertungsbogen Aus der ersten Runde oder auch aus früheren Wettbewerbsteilnahmen kennen Sie den Bewertungsbogen, der angibt, wie Ihre Einsendung die einzelnen Bewertungskriterien erfüllt hat. Auch in dieser Runde können Sie den Bewertungsbogen im Anmeldesystem PMS einsehen. In der ersten Runde ging die Bewertung noch von 5 Punkten aus, von denen bei Mängeln dann abgezogen werden konnte. In der zweiten Runde geht die Bewertung nun von 20 Punkten aus; dafür gibt es deutlich mehr Bewertungskriterien, bei denen Punkte abgezogen oder auch hinzuaddiert werden konnten.

Terminlage der zweiten Runde Für Abiturienten ist der Terminkonflikt zwischen Abiturvorbereitung und zweiten Runde sicher nicht ideal. Doch leider bleibt uns nur die erste Jahreshälfte für die zweite BwInf-Runde: In der zweiten Jahreshälfte läuft nämlich die zweite Runde des Mathewettbewerbs, dem wir keine Konkurrenz machen wollen. 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 der Dokumentation 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.

Bewertungskriterien Bei den im Folgenden beschriebenen Lösungsideen 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, wesentliche Auszüge aus dem Quellcode enthalten), Quellcode (insbesondere: Strukturierung und Kommentierung, gute Übersicht durch Programm-Dokumentation) und Programm (keine Implementierungsfehler).

Danksagung An der Erstellung der Lösungsideen haben mitgewirkt: Robert Denkert und Karl Schrader (Aufgabe 1), Jan-Niklas Brandes und Adrian Lison (Aufgabe 2) sowie Robert Czechowski und Nikolai Wyderka (Aufgabe 3). Die Aufgaben wurden vom Aufgabenausschuss des Bundeswettbewerbs Informatik entwickelt, und zwar aus Vorschlägen von Holger Schlingloff (Aufgabe 1) und Peter Rossmanith (Aufgaben 2 und 3).

Aufgabe 1: Geburtstagskuchen

1.1 Abstrahierung der Aufgabenstellung und Definitionen

Gegeben sind eine Fläche und eine Menge an Punkten. Zuerst ist ein Maß zu finden, welches beschreibt, wie gleichmäßig verteilt die Punkte auf der Fläche sind, und ein Programm zu schreiben, welches das Maß effizient berechnet.

Zusätzlich ist für eine fest definierte, herzförmige Fläche ein Algorithmus zu entwickeln, der eine Menge an Punkten so darauf platziert, dass sie nach dem definierten Maß möglichst gleichmäßig verteilt sind.

Mögliche Flächen

Naheliegend ist es, die Fläche auf Polygone einzuschränken, da sich mit diesen gut arbeiten lässt. Auch andere Darstellungsmöglichkeiten wie die Beschreibung der Form durch (Radial-) Funktionen ist möglich, macht aber die spätere Implementierung und Arbeit mit der Fläche unnötig kompliziert.

Definition der Herzform

Solange die gewählte Form als Herz zu erkennen ist, ist sie gut genug. Um eine hübsche Herzform zu erhalten, bietet es sich an, einige (hier: 42) Punkte einer Herzfunktion (siehe unten sowie Abb. 1 und 2) zu berechnen, und das resultierende Polygon zu nutzen.

$$x(t) = 12\sin(t) - 4\sin(3t), \qquad t \in \{0, 2\pi\},$$
$$y(t) = 13\cos(t) - 5\cos(2t) - 2\cos(3t) - \cos(4t), \qquad t \in \{0, 2\pi\}.$$

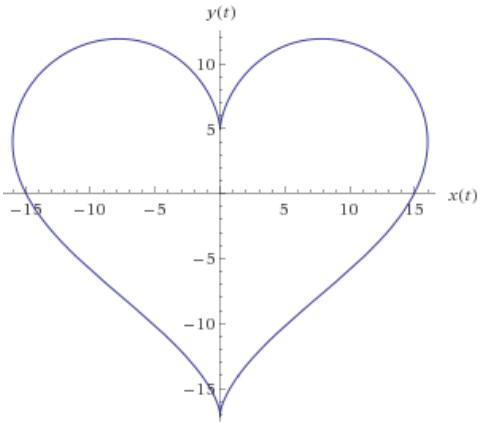


Abb. 1: Funktion mit Achsen

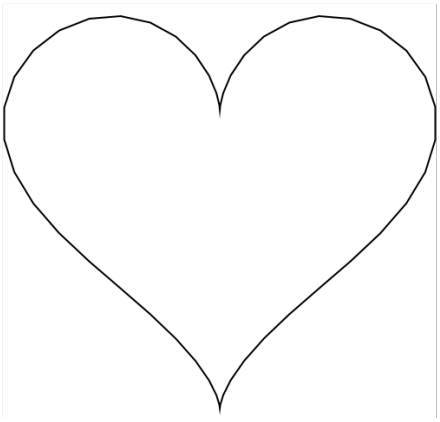


Abb. 2: Polygon aus 42 Punkten

1.2 Wahl des Maßes

An das Maß für die Gleichmäßigkeit einer Punkteverteilung werden drei Anforderungen gestellt:

1. Es muss mit der Intuition übereinstimmen.
2. Es muss einen effizienten Algorithmus geben, der das Maß für gegebene Punkte und Fläche berechnet.
3. Es muss einen effizienten Algorithmus geben, der für gegebene Punkte und Fläche eine Verteilung der Punkte auf der Fläche berechnet, welche von dem Maß als möglichst gleichmäßig verteilt angesehen wird.

Bei der Lösung der Aufgabe bietet es sich an, zuerst Punkt 3 zu erfüllen, da meist Punkt 2 automatisch mit abfällt.

An dieser Stelle können unmöglich alle sinnvollen Maße angegeben werden. Wir wollen nur kurz zwei naheliegende Maße angeben und deren Schwächen diskutieren. In beiden Fällen handelt es sich um Maße, deren Werte maximiert werden sollen.

- *Das Maß sei das Minimum der Abstände von zwei Punkten zueinander.* Problem: Der Rand wird nicht mit einbezogen. Dadurch liegen bei der besten Verteilung die Punkte auf dem Rand, was von der Intuition nicht als gleichmäßig verteilt angesehen wird.
- *Das Maß sei das Minimum der Abstände von zwei Punkten zueinander oder von einem Punkt und dem Rand.* Problem: Dieses Maß funktioniert zwar gut für die bestmögliche Verteilung, liefert aber beim Vergleich von zwei nicht perfekten Verteilungen unter Umständen unintuitive Ergebnisse: Betrachtet man in den nächsten Abbildungen die beiden Verteilungen Imperfekt 1 und Imperfekt 2, sind bei Imperfekt 1 die Punkte der Intuition nach besser verteilt. Jedoch beträgt der kleinste Abstand im linken Bild $3LE$ (2 Punkte oben links), im rechten hingegen $20LE$. Damit würde hier entgegen der Intuition entschieden werden.

Damit wir im Folgenden mit diesem Ansatz präzise arbeiten können, benötigen wir ein paar Definitionen:

Voronoi-Diagramm Gegeben ist eine Fläche und eine Menge von Punkten. Für jeden Punkt bilden die Punkte der Fläche, die näher an diesem Punkt liegen als an jedem anderen, die zugehörige Voronoi-Region. Zusammen bilden die Voronoi-Regionen also eine Zerlegung der Fläche. Diese wird Voronoi-Diagramm genannt.

Centroidal Voronoi Tesselation (CVT) Eine CVT (auf Deutsch etwa: Schwerpunktbezogene Voronoi-Zerlegung) ist ein spezielles Voronoi-Diagramm, bei dem die Kerzen jeweils im geometrischen Schwerpunkt ihrer zugehörigen Region liegen. CVTs haben die Eigenschaft, dass sie die optimale Verteilung von Punkten auf der Fläche darstellen, sodass der durchschnittliche Abstand aller Punkte der Fläche zu der jeweils nächstgelegenen Kerze minimal ist – was natürlich genau das Gesuchte ist. An dieser Stelle wird auf einen Beweis der Eigenschaft verzichtet.

Zur Ermittlung einer sehr guten Approximation einer CVT nutzen wir Lloyd's Algorithmus, den wir im nächsten Abschnitt vorstellen.

Die beste Verteilung der Punkte ist erreicht, wenn jede Kerze im Schwerpunkt ihrer Zelle liegt. Mit dieser Idee lassen sich auch alle anderen Verteilungen bewerten:

Das Maß sei definiert als der durchschnittliche Abstand einer Kerze von dem Schwerpunkt ihrer Region.

$$\text{Das Maß beträgt } 0 \quad \Leftrightarrow \quad \text{Die Kerzen sind bestmöglich verteilt.} \tag{1}$$

Als netter Nebeneffekt ergibt sich bei diesem Maß eine Aufteilung in wohlgeformte Kuchenstücke, sodass jedes Kuchenstück eine Kerze in ihrem Schwerpunkt trägt.

Beispiele für die Qualität des Maßes

Hier soll gezeigt werden, dass das Maß wirklich mit der Intuition übereinstimmt.

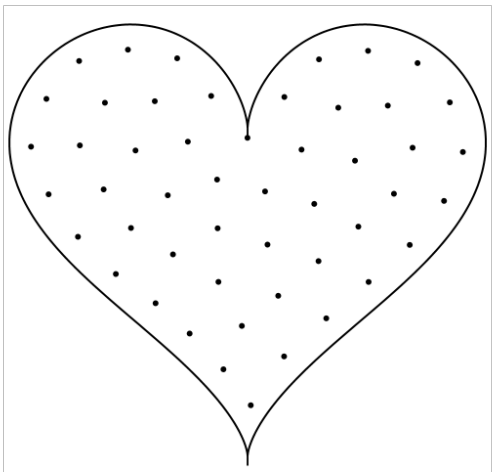


Abb. 3: Perfekt: Score 10^{-7}.

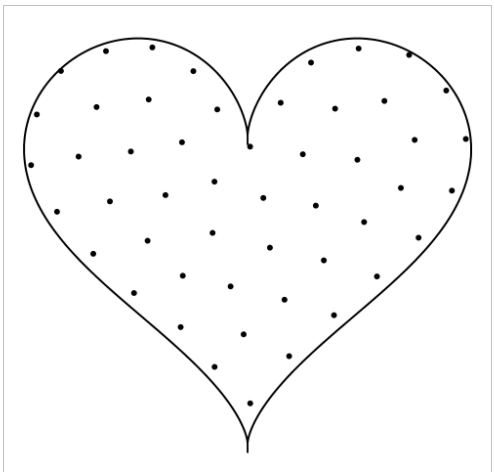


Abb. 4: Am Rand, Score 3.44.

Man erkennt in Abb. 4, dass es bestraft wird, wenn Kerzen zu nah am Rand liegen. In Abb. 5 und 6 ist zu sehen, dass auch verschiedene nicht optimale Verteilungen untereinander verglichen werden können, und sinnvolle Ergebnisse liefern. Abb. 5 ist dabei schon gleichmäßiger als Abb. 6, und wird auch so bewertet.

1.3 Lösungsidee

Es ist schwer, in einem Schritt Punkte zu finden, die perfekt verteilt sind. Deshalb wird erst eine Basislösung ermittelt, bei der die Punkt bereits recht gut verteilt sind, und dann diese Lösung schrittweise verbessert.

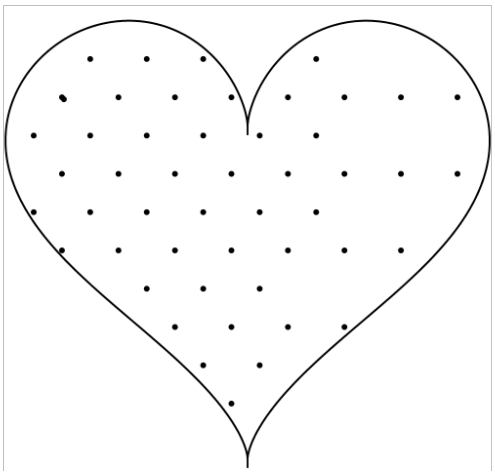


Abb. 5: Imperfekt 1, Score 4.3.

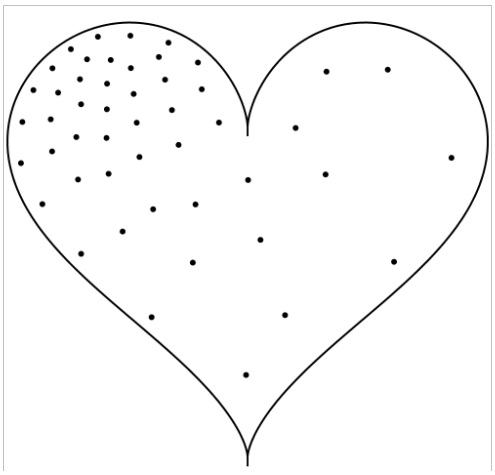


Abb. 6: Imperfekt 2, Score 4.57.

Finden der Basislösung

Die Basislösung muss nicht besonders gut sein, jedoch lässt sich durch eine gute Basislösung viel Zeit bei der Iteration sparen. Auch wenn die Punkte zufällig auf der Fläche platziert werden, findet die Iteration eine beste Lösung, und bei der hier verwendeten, schnellen Iteration stören die zusätzlichen Schritte kaum. Generell gibt es aber an dieser Stelle elegantere Lösungen.

Wir nutzen folgendes Verfahren: Gegeben sei die Anzahl der Kerzen n. Da auf einer unendlichen Ebene ein Wabengitter die beste Verteilung wäre, schätzen wir aus dem Verhältnis der Fläche zum umschließenden Rechteck einen Skalierungsfaktor, und legen dann das Wabengitter über die Fläche. Da die Schätzung nicht perfekt ist, müssen meist noch ein paar Kerzen ergänzt werden. Diese werden in den Mittelpunkten der gleichseitigen Dreiecke, welche durch die Mittelpunkte der Waben aufgespannt werden, platziert. Ein Beispiel findet sich bei dem bebilderten Lauf des Algorithmus in Abschnitt 1.5.

Iteration

Um die Basislösung schrittweise zu einer CVT zu verbessern, nutzen wir Lloyd's Algorithmus. Dessen Vorschrift ist denkbar einfach:

1. Ermittele die Voronoi-Zerlegung.
2. Berechne das Maß und terminiere, wenn das Ergebnis nahe genug an 0 ist.
3. Verschiebe jede Kerze in den Schwerpunkt ihrer zugehörigen Voronoi-Region. Gehe zu Schritt 1.

Auf einen exakten Beweis, dass dieser Algorithmus wirklich gegen die optimal Lösung konvergiert, soll an dieser Stelle verzichtet werden. Da jedoch in jedem Schritt die Kerzen in der Schwerpunkt verschoben werden, und der Algorithmus nur hält, wenn die Kerzen in ihrem Schwerpunkt liegen, ist es intuitiv klar.

Mit diesem Algorithmus ist es leider noch nicht getan, da auch das effiziente Ermitteln der Voronoi-Zerlegung nicht trivial ist. Wir nutzen dafür die Eigenschaft aus, dass das Voronoi-Diagramm der geometrisch duale Graph zur Delaunay-Triangulierung ist.

Delaunay-Triangulierung Die Delaunay-Triangulierung ist eine spezielle Triangulierung, bei der kein Punkt innerhalb des Umkreises einer der bei der Triangulierung entstandenen Dreiecke liegt.

Während der Berechnung des geometrisch dualen Graphen lässt sich auch noch ein weiteres Problem eliminieren: In der Theorie werden Voronoi-Regionen für Punkte auf einer unendlichen Ebene ermittelt, wir müssen diese noch auf die gegebene Fläche einschränken. Hierfür würden wir gerne einen der vielen bekannten Algorithmen für das Schneiden mit Polygonen nutzen, jedoch ist bei uns die Fläche (insbesondere beim Herz) nicht konvex, weshalb wir nur einen naiven Ansatz nutzen. Für jede Kante, die einen offenen Endpunkt hat, wird der Schnittpunkt mit dem Polygon ermittelt, und dann die entsprechende Region auf die begrenzende Fläche eingeschränkt.

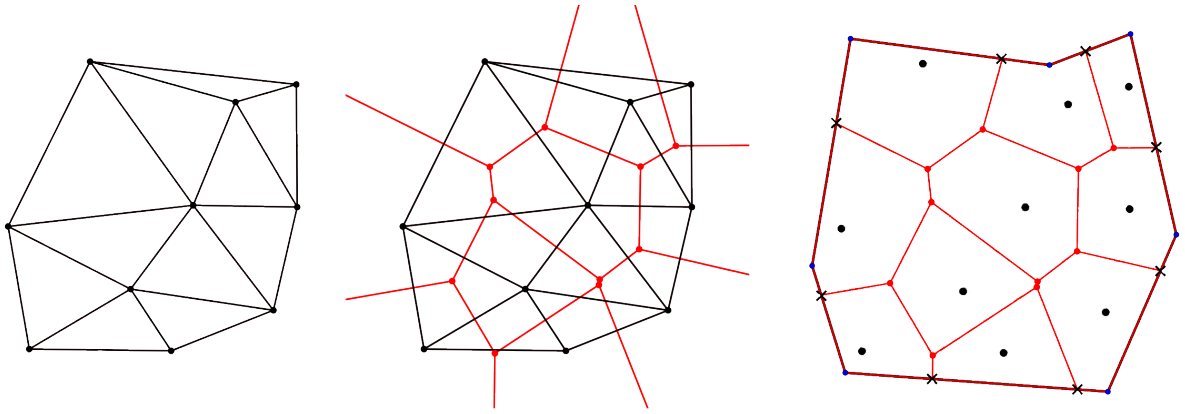


Abbildung 7: Delaunay-Triangulation zu Voronoi-Diagramm [1]

Jedoch sind wir an dieser Stelle noch nicht fertig: Die hier verwendete Delaunay-Triangulierung muss auch erst ermittelt werden. Um diese zu erhalten, ergänzen wir jeden Punkt um eine z-Koordinate $z = x^2 + y^2$. Von der entstehenden Menge an 3D-Punkten berechnen wir dann die konvexe Hülle mit Graham Scan. Um die Länge dieser Musterlösung zu beschränken, wird auf eine präzise Beschreibung dieses Algorithmus verzichtet. Bei weiterem Interesse: `http://www.iti.fh-flensburg.de/lang/algorithmen/geo/graham.htm`. Man hätte hier auch den einfacheren Quickhull-Algorithmus nutzen können, dadurch verliert man jedoch $\mathcal{O}(n \log n)$, und hat nurnoch $\mathcal{O}(n^2)$. Projiziert man nun alle Dreiecke der konvexen Hülle mit in der z-Koordinate negativem Normalvektor zurück in die Ebene, erhält man die Triangulierung. Weitere Details zur Ermittelung der Delaunay-Triangulierung mit Hilfe der konvexen Hülle lassen sich in der Literatur[2] und im Netz[3] finden.

[1] Mit Material aus `https://en.wikipedia.org/wiki/File:Delaunay_Voronoi.svg`
[2] Klein, Rolf: Algorithmische Geometrie. Springer, 2005, S. 304ff
[3] `https://de.wikipedia.org/wiki/Voronoi-Diagramm\#Algorithmus`

Implementierung

Auch wenn die Menge an verwendeten Algorithmen recht groß ist, sind alle gut dokumentiert im Internet und Werken zur algorithmischen Geometrie zu finden. Für diesen und für alle anderen Ansätze gilt: Die eigentliche Schwierigkeit der Implementierung ist es, alle Spezialfälle zu finden und Probleme, die aus der Rechenungenauigkeit entstehen, angemessen zu behandeln. Algorithmen zur analytischen Geometrie sehen leider nur als Pseudocode schön aus ...

1.4 Laufzeit

Sei n die Anzahl der Punkte, und sei k die konstante Anzahl der Ecken des herzförmigen Polygons. Damit lässt sich über die Laufzeiten der einzelnen Teile des Verfahrens Folgendes sagen:

Laufzeit zum Ermitteln der Basislösung: $\mathcal{O}(n)$

Laufzeit der Iteration pro Schritt:
Berechnung der konvexen Hülle: $\mathcal{O}(n \log n)$
Berechnung der Delaunay-Triangulierung aus der konvexen Hülle: $\mathcal{O}(n)$
Berechnung des Voronoi-Diagramms aus der Delaunay-Triangulierung: $\mathcal{O}(n)$
Schneiden des Voronoi-Diagramms mit der Fläche: $\mathcal{O}(kn) = \mathcal{O}(n)$

Sei i die Anzahl der Iterationsschritte. Damit ergibt sich als Gesamtlaufzeit: $\mathcal{O}(i \cdot n \log n)$

Reallaufzeit

Der Algorithmus ist ausreichend schnell, um für alle praktischen Anzahlen von Kerzen ($<$ 150) in unter einer Sekunde die beste Verteilung zu ermitteln. Für mehr als 1000 Kerzen werden dann schon einige Sekunden benötigt. Jedoch sieht man zu diesem Zeitpunkt unter den ganzen Kerzen den Geburtstagskuchen schon nicht mehr.

1.5 Resultat am Beispiel

Beispiellauf für 50 Kerzen

Die Punkte sind die aktuellen Positionen der Kerzen. Die kleinen Quadrate sind die aktuellen Positionen der Schwerpunkte der Zellen. Die Zellen sind die Voronoi-Regionen.

In Abb. 8 ist die Basislösung zu erkennen. Die Punkte sind grob auf die verschiedenen Teile der Fläche verteilt. In Abb. 9 hat sich eine gleichmäßige Schicht von Kerzen am Rand gebildet, jedoch ist der mittlere Teil noch recht chaotisch. In Abb. 10 beginnen die Regionen im Inneren, die Form gleichseitiger Hexagone anzunehmen. Dies ergibt Sinn, da dies auch die beste Verteilung auf einer unendlichen Fläche ist. In der letzten Abbildung ist das Ergebnis zu sehen, das der Algorithmus am Ende ausgibt. Das Maß bewertet die Verteilung mit fast 0,

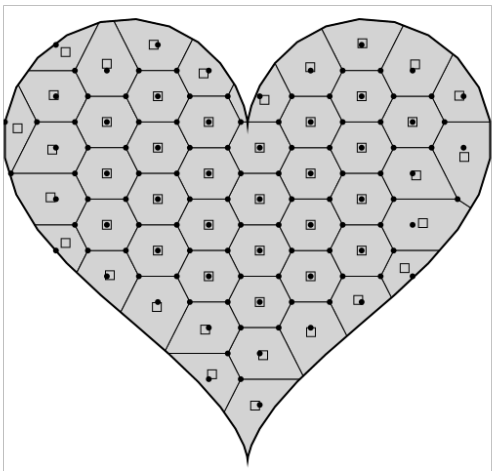


Abb. 8: Nach 0 Iterationen, Score 5.67.

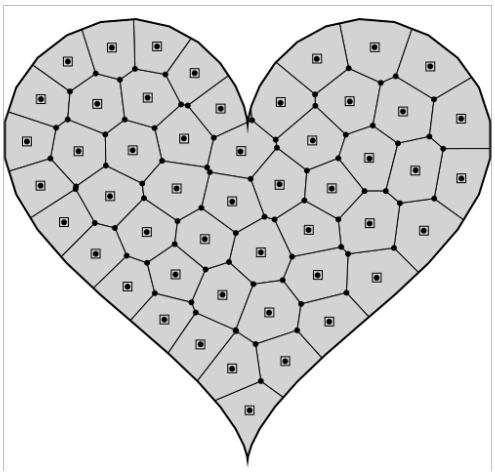


Abb. 9: Nach 25 Iterationen, Score 0.24.

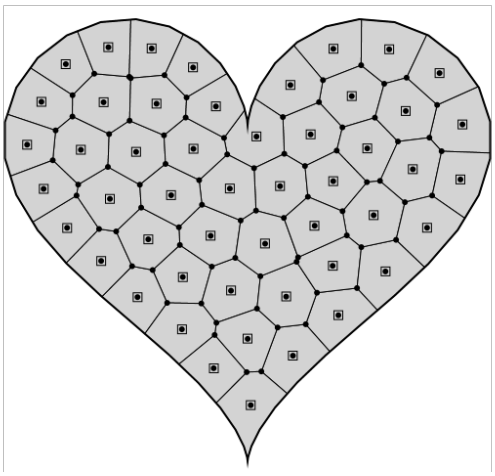


Abb. 10: Nach 65 Iterationen, Score 0.01.

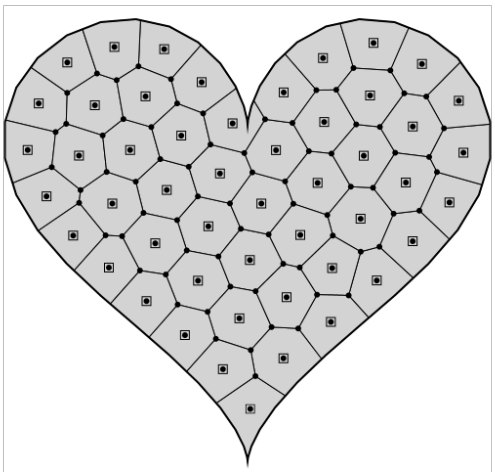


Abb. 11: Nach 500 Iterationen, Score 0.00001.

und die meisten Regionen haben eine Wabenform angenommen. Auch intuitiv würde man die Kerzen als gleichmäßig verteilt betrachten.

Beispiele für verschiedene Mengen an Kerzen

Der Score beträgt für jedes Bild etwa 0.00001.

In der ersten Abbildung wurden 12 Kerzen, genau wie auf dem Geburtstagskuchen für Lindas Schwester, verteilt. Diese Verteilung ist offensichtlich besser als die, die ihr kleiner Bruder produziert hat.

Es fällt auf, dass eine beste Verteilung im Allgemeinen nicht symmetrisch ist. Es existieren für jede Menge an Kerzen auch eine optimale symmetrische Verteilung, jedoch neigt der Algorithmus auf Grund von Floating-Point-Fehlern zu asymmetrischen Lösungen. Diese werden

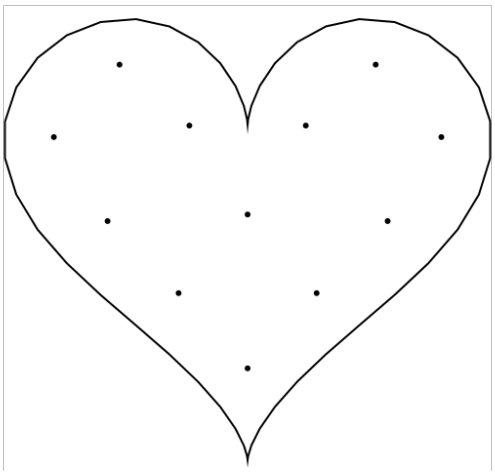


Abb. 12: 12 Kerzen, 224 Iterationen.

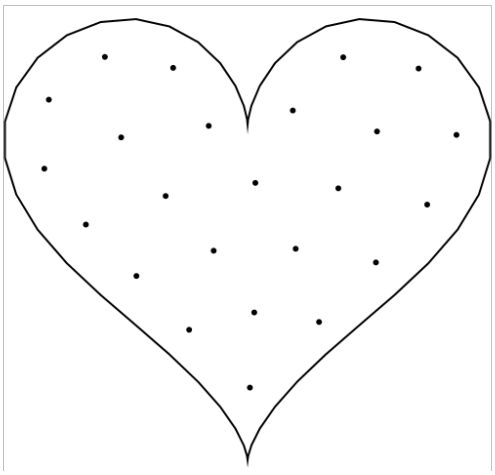


Abb. 13: 24 Kerzen, 287 Iterationen.

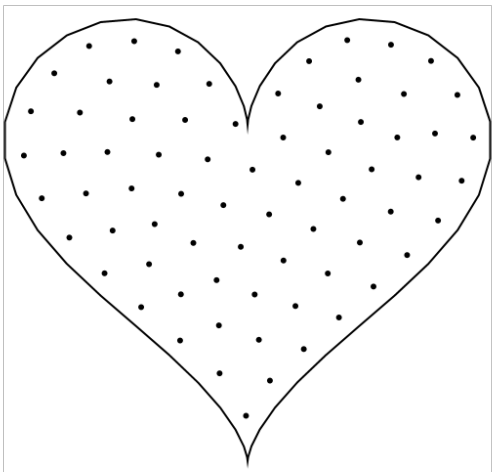


Abb. 14: 72 Kerzen, 569 Iterationen.

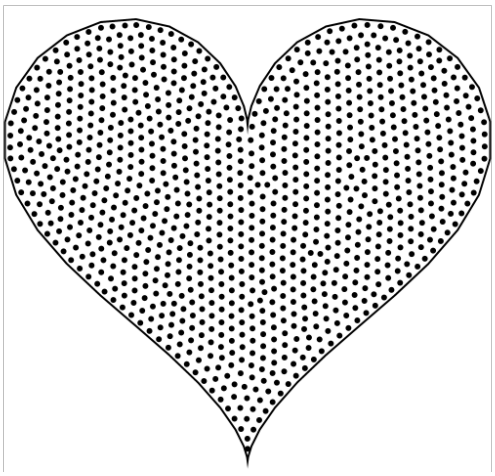


Abb. 15: 1024 Kerzen, 312 Iterationen.

genau so bewertet wie die symmetrischen, jedoch scheinen sie für den menschlichen Betrachter oft schlechter bei gleichem Score.

Auch für große Anzahlen an Kerzen wird noch ein gutes Ergebnis geliefert, bei dem die Regionen ein wabenförmiges Gitter bilden. Bei noch größeren Anzahlen wird es dann schwer, den Kuchen noch zu sehen.

1.6 Alternative Ansätze

Nicht nur beim Verteilungsmaß, sondern insbesondere bei den Verteilungsverfahren gibt es viele andere mögliche Ansätze.

Ein grundsätzlicher Unterschied ergibt sich, wenn die zu betrachtende Fläche nicht als geometrische Form angegeben ist, sondern als eine Menge diskreter Bildpunkte - also als Pixelgrafik. Die Verteilungsmaße und -verfahren können für Pixelgrafiken durchaus ähnlich denen

für geometrische Formen sein; man kann z. B. auch für Pixelgrafiken Schwerpunkt-bezogene Flächenzerlegungen definieren. Allerdings funktionieren die Berechnungen dann in der Regel deutlich anders. Bei einer hohen Auflösung der Grafik, also einer großen Menge an Pixeln kann die reale Laufzeit entsprechend hoch werden. Dem muss begegnet werden.

Das Verteilungsmaß „minimaler Abstand zweier Punkte“ kann annähernd maximiert werden, indem schrittweise (a) zunächst für jede Kerze der minimale Abstandsvektor zu einer anderen Kerze bestimmt und (b) dann die Kerze leicht in der Gegenrichtung dieses Vektors verschoben wird. Oben ist allerdings schon gesagt worden, dass bei Maximierung dieses Maßes Kerzen auf dem Rand platziert werden. Dagegen kann man einen (nicht ganz eleganten) Kunstgriff anwenden und von vornherein einen Randbereich festlegen, in dem keine Kerze platziert werden darf.

Eine gleichmäßige Verteilung lässt sich auch in Anlehnung an physikalische Modelle erreichen, z. B. die gegenseitige Abstoßung von Elektronenpaaren. Hier nimmt nicht nur die nächstliegende Kerze, sondern nehmen alle anderen Kerzen Einfluss auf die Position einer bestimmten Kerze. Auch bei einem solchen Ansatz wird die Verteilung der Kerzen schrittweise verbessert, unter Anwendung des gewählten Modells.

Ein allererster Gedanke könnte sein, eine regelmäßige Punkte-Struktur wie etwa ein Dreiecksgitter über die Fläche zu legen und die Kerzen auf die zugehörigen Punkte zu stellen (vgl. auch die oben beschriebene Basislösung). Gerade die Herzform mit ihrer Mischung aus Polygon und Kreisausschnitten zeigt die Schwäche eines solchen Ansatzes auf: Gute Verteilungen lassen sich zwar für einige, aber nicht für alle Kerzenzahlen finden.

1.7 Bewertungskriterien

- Auch wenn das in Teil 1 geforderte Maß und das in Teil 2 geforderte Verteilungsverfahren sich stark aufeinander beziehen (wie in dieser Beispiellösung), sollten doch die wichtigsten Elemente der Aufgabenteile einzeln erkennbar werden. Insbesondere muss für das Verteilungsmaß geklärt werden, für welche möglichen Flächen es berechnet werden kann. Eine Beschränkung z. B. auf Polygone ist akzeptabel; Punktabzug gibt es hingegen, wenn das Maß nur für die Herzform berechnet werden kann.
- Das Maß muss die drei oben genannten Kriterien (intuitiv, gut berechenbar, passender Algorithmus existiert) erfüllen. Offensichtlich nicht der Aufgabenstellung entsprechende Maße wie

  Der Kuchen wird in unterschiedliche Stücke geschnitten, je nach Wunsch der Geburtstagsgäste. Die Kerzen sind gleich verteilt, wenn auf jedem der Stücke eine Kerze ist.

  landen höchstens in den Perlen der Informatik, werden aber nicht mit Punkten belohnt. Achtung: Ob ein Maß „intuitiv“ ist, bezieht sich nur auf den Begriff der Gleichmäßigkeit, nicht auf andere ästhetische Kriterien. Führt das Maß zu einer Verteilung mit problematischen Eigenschaften (z. B. dass Kerzen bis zum Rand gesetzt werden), sollte das angesprochen werden.

Obwohl die Aufgabenstellung ein „formales Maß“ fordert, wird eine mathematische Formalisierung nicht zwingend erwartet. Wichtig ist, dass man aus der Beschreibung nachvollziehen kann, wie das Maß für eine Fläche und eine Punktmenge zu berechnen ist.

- Für Teil 2 muss zunächst die Herzform definiert werden oder gesagt werden, wie die verwendete Herzform ermittelt wurde. Die Form soll sinnvoll gewählt sein. Wie bereits gesagt: Solange die gewählte Form als Herz zu erkennen ist (wie etwa zwei Halbkreise auf einem Dreieck), ist sie gut genug. Nicht akzeptabel ist ein „Pseudo-Herz“, wenn z. B. mit Polygonen gerechnet wird, die die Herzform nur grob annähern.
- Das Verteilungsverfahren sollte in der Regel gute Ergebnisse liefern, und zwar bezüglich des gewählten Maßes (ansonsten implementiert das Verfahren nicht das beschriebene Maß) und für beliebige Kerzenzahlen. Falls der Algorithmus für eine kleinere Menge von Eingaben (z. B. geringe Anzahl von Kerzen) schlechte Ergebnisse liefert, ist das kein Problem, solange die Schwächen erkannt und besprochen werden.
- Schön ist natürlich, wenn nicht nur das Maß sondern auch das Verteilungsverfahren für andere Formen als die Herzform funktioniert. Für Verfahren, die sich aus der gewählten Darstellung der Herzform fest ableiten, kann es Punktabzug geben, während es für frei verallgemeinerbare Verfahren Pluspunkte geben kann.
- Pluspunkte kann es auch geben, wenn mehrere Varianten für das Verteilungsverfahren diskutiert und umgesetzt werden.
- Selbstverständlich hängt die (theoretische) Laufzeit direkt mit dem gewählten Maß zusammen, sollte wenn möglich aber nicht exponentiell sein, solange nicht zusätzlich optimiert wurde. Die meisten Ansätze können mit einer polynomialen Laufzeit (im günstigen Fall quadratisch oder geringer) auskommen. Die praktische Laufzeit sollte so sein, dass sich zweistellige, Anzahlen von Kerzen in wenigen Sekunden auf dem Herz verteilen lassen.
- Die Verfahren zur Ermittlung einer gleichmäßigen Verteilung und die Berechnung des Maßes müssen verständlich beschrieben werden. Es soll begründet sein, warum das Verfahren welche Ergebnisse liefert.
- Eine geeignete Anzahl an Beispielen und Bildern wird erwartet. Anhand der Beispiele sollte man Arbeitsweise und Qualität des Verfahrens einschätzen können; daraus ergeben sich Anforderungen an die Auswahl der Beispiele, aber auch an die (grafische) Darstellung der Verteilung. Insbesondere sind mehrere Beispiele für die Herzform Pflicht, und zwar mit verschiedenen Kerzenanzahlen. An dieser Stelle sollten – falls vorhanden – auch Eingaben gezeigt werden, für die der Algorithmus schlechte Arbeit leistet, und erwähnt werden, warum.
- Falls ein Algorithmus aus einem Buch oder Internet übernommen wird, müssen die Quellen angemessen referenziert werden. Außerdem muss erkenntlich sein, dass der Autor der Einsendung begriffen hat, warum der Algorithmus das tut, was er tut.

- *Erweiterungen*: Die Anwendbarkeit der Lösung auf verschiedene Formen ist noch keine echte Erweiterung. Neue Anforderungen ergeben sich aber z. B., wenn eine 3D-Verteilung (Stichwort: Rosinen im Kuchen) oder eine Gewichtung der Kerzen (dicke Kerzen benötigen mehr Raum um sich als dünne) realisiert wird.

Aufgabe 2: Missglückte Drohnenlieferung

2.1 Grundsätzliche Überlegungen

In der Aufgabe geht es darum, die Bewohner der Stadt Amacity bei der richtigen Zustellung von Paketen zu unterstützen. Die auf die Dächer der Häuser per Drohnenlieferung falsch abgeworfenen Pakete sollen durch Zuwerfen zwischen Nachbardächern an ihre jeweilige Bestimmungsadresse transportiert werden. Bei der Suche nach einem optimalen Wurf-Plan kann die Informatik eine ihrer großen Stärken ausspielen, nämlich die Simulation (realer) Sachverhalte mithilfe von Modellen. Im Zuge der Optimierung muss dann nämlich nicht mehr auf realen Objekten operiert werden (in unserem Beispiel würde dies bedeuten, dass die Bewohner von Amacity die Pakete in unzähligen Durchgängen hin- und her werfen müssten, um eine optimale Lösung zu finden), sondern es kann mit abstrahierten Modellen gearbeitet und schließlich die gefundene Optimallösung ausprobiert werden. Dies wollen wir im Folgenden versuchen. Dazu werden wir das Problem allgemein für eine $n \times n$-Stadt mit einer beliebigen Paketverteilung lösen und dann auf Amacity anwenden.

Ziel ist es, mit möglichst wenigen Schritten alle Pakete an ihre Zieladresse zu befördern. Wichtig: da Pakete parallel geworfen werden können, ist dies nicht zwingend damit gleichzusetzen, möglichst wenig *Würfe* zu verwenden. Die Bedingung, dass sich nach jedem Schritt auf jedem Haus genau ein Paket befinden muss, führt dazu, dass an einem Dach, von dem aus ein Paket weggeworfen wurde, auch eines ankommen muss. Dies wiederum führt zur Bildung von *Wurfkreisen*, bei denen der Weg von einem geworfenen Paket über alle nachfolgenden Dächer bis zum Ausgangsort zurückverfolgt werden kann. Die einfachste Art eines solchen Zyklus ist ein Pakettausch zwischen zwei Dächern. Es gibt aber auch größere Kreise oder komplexere Muster (*Schlangen*), die jedoch alle eine **gerade** Anzahl an Würfen umsetzen: *bewegt* man sich vom Ursprungsdach um n Würfe in x-Richtung weg, so muss die gleiche Anzahl an Paketen entgegen dieser Richtung geworfen werden, was zusammen $2n$ Würfe macht. Gleiches gilt für die y-Richtung. Es können innerhalb eines Schrittes auch mehrere Wurfkreise nebeneinander stattfinden, solange es keine Überschneidungen gibt, d. h. sie keine gleichen Dächer betreffen. Gerade dies wird von großer Bedeutung sein. Wir werden dazu die Durchführung eines Schrittes (eine Menge von $n > 0$ Wurfkreisen) als eine Runde betrachten und rundenbasiert nach einer optimalen Lösung suchen.

2.2 Erkenntnisse über die Existenz von Lösungen

Existiert immer eine Lösung?

Ja, denn mindestens die folgende (sicher nicht optimale) Strategie führt immer zu einer vollständigen Zustellung aller Pakete: Man befördere alle Pakete, die in die Zeile $n-1$ gehören, z. B. durch Vertauschungsoperationen genau in diese Zeile. Dass andere Pakete dabei eventuell weiter von ihrer Zieladresse entfernt werden, ist irrelevant. Anschließend sortiert man die Pakete innerhalb ihrer Zeile mittels Vertauschungsoperationen (Bubble-Sort) in die richtigen Spalten. Nun ist die letzte Zeile des Rasters sortiert, es verbleibt ein unsortiertes Raster der Form $(n-1) \times n$. Das Verfahren wird wiederholt, bis alle Zeilen sortiert sind.

Was ist die bestmögliche Lösung?

Um zu beurteilen, wie weit (d. h. wie viele Würfe) sich ein Paket von seinem Bestimmungsort entfernt befindet, verwenden wir die Manhattan-Metrik (auch Cityblock-Metrik). Bei diesem Distanzmaß werden die absoluten Strecken in x- und y-Richtung addiert. Das Paket mit dem Ziel $(5,3)$ befindet sich am Punkt $(2,4)$ deshalb mindestens $|5-2|+|3-4|=4$ Würfe von seinem Zuhause entfernt. Die minimale Anzahl an Schritten, die noch nötig sind, um alle Pakete an ihren Zielort zu transportieren, ist deshalb die Manhattan-Distanz des aktuell am weitesten entfernten Pakets. Diese maximale Manhattan-Distanz (d_{ManMax}) ist also die untere Schranke für eine Lösung. Benötigt eine Lösung genau diese Anzahl an Schritten, dann wissen wir bereits, dass sie optimal ist. In Amacity hat das Paket mit der Zieladresse $(0,9)$ mit einer Distanz von 15 den größten Abstand von seinem Bestimmungsort, also ist $d_{\text{ManMax}}=15$. Das Abstandsmaß gibt uns einen ersten Ansatz, um Pakete zu priorisieren, d. h. festzustellen, wie dringend es nötig ist, diese näher an ihren Bestimmungsort zu transportieren. Letzteres ist sehr wichtig, da wir nicht immer mit jedem Zug *alle* Pakete voranbringen können und manchmal Pakete sogar von ihrer Lieferadresse weiter entfernen müssen.

Es sollen folgende Begrifflichkeiten verwendet werden:

Zieldistanz eines Paketes ist die Manhattan-Distanz zwischen der aktuellen Position des Paketes und der Lieferadresse.

Alphapaket bezeichnet ein Paket, dessen Zieldistanz der höchsten vorkommenden Zieldistanz entspricht.

Betapaket bezeichnet ein Paket, dessen Zieldistanz der höchsten vorkommenden Zieldistanz weniger 1 entspricht.

Omegapaket bezeichnet ein Paket, das weder ein Alpha- noch ein Betapaket ist.

Kann die bestmögliche Lösung immer erreicht werden?

Um die bestmögliche Lösung zu erreichen, müssen in jedem Schritt alle Pakete, die am weitesten von ihrem Ziel entfernt sind, ein Dach näher an dieses geworfen werden. Dies ist aber nicht immer möglich, denn es gibt sogenannte Engstellen. Wie in Abb. 16 zu sehen ist, kann es z. B. passieren, dass zwei solcher Alphapakete auf ein und dasselbe Hausdach geworfen werden müssten, etwa wenn sie sich beide schon in der richtigen Spalte befinden, aber A an einen Ort unterhalb von B und B an einen Ort oberhalb von A gehört, diese aber nicht auf direkt angrenzenden Nachbardächern liegen, sodass keine Vertauschoperation möglich ist. Abb. 17 zeigt eine weitere Art von Engpass, bei der Alphapakete nur dann weitertransportiert werden können, wenn ein oder mehrere Betapakete in eine ihnen widerstrebende Richtung geworfen werden (dies führt dazu, dass sie nach dem Wurf eine Distanz haben, die der der aktuellen Alphas entspricht). In beiden Fällen kann nicht durch Anwendung von Würfen eine Verringerung der maximalen Manhattan-Distanz erreicht werden, und wir weichen von der "Optimallösung" ab. Aufgrund der oben genannten Regeln kann diese auch nicht wieder "eingeholt" werden!

Es muss also geklärt werden, wie mit solchen Engstellen umgegangen wird und wie sie bei der Suche nach einer Optimallösung möglichst vermieden werden können. Sicher vermeiden lassen sich Engstellen aber nicht, denn in einem gegebenen Szenario könnten sich die Pakete

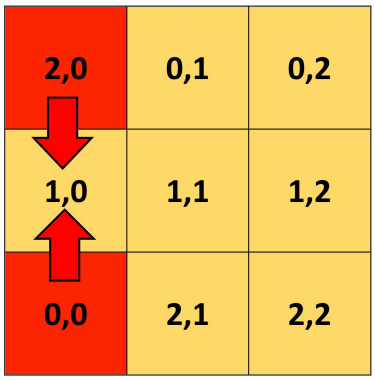


Abb. 16: Zum Erreichen der Optimallösung müsste sowohl $(0,0)$ als auch $(2,0)$ auf das Hausdach $(1,0)$ geworfen werden, was nicht möglich ist.

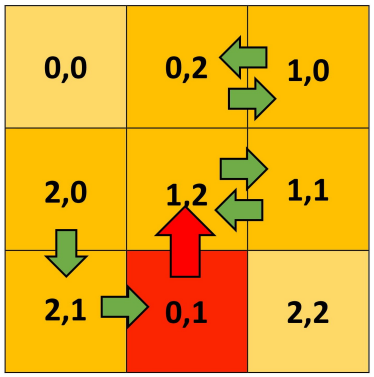


Abb. 17: Um Alphapaket $(0,1)$ zu verschieben, muss entweder Betapaket $(1,2)$ oder $(1,1)$ weiter von seinem Ziel entfernt werden, dieses wird dann zu einem Alphapaket.

schon in einer solchen Anordnung befinden. Das wiederum bedeutet, dass unsere "Optimallösung" für ein beliebiges Raster u. U. gar nicht existiert!

2.3 Verfahren zur Bestimmung von Lösungen

Der einfachste Ansatz zum Bestimmen eines optimalen Wurfplans ist ein Brute-Force-Verfahren, bei dem *alle* möglichen Pläne ermittelt werden und man dann den besten Plan auswählt. Eine derart ineffiziente Vorgehensweise ist jedoch allein aus Gründen der Laufzeit keine ernst zu nehmende Option. Um nämlich alle möglichen Kombinationen zu finden, müssen wir vom gegebenen Raster aus alle möglichen Schritte (d. h. Kombinationen aus Wurfkreisen) ermitteln und dann für jeden Schritt aus dem daraus folgenden Raster[4] die Nachfolgeschritte ermitteln.

Wollen wir den dazu benötigten Aufwand abschätzen, so müssen wir erstens wissen, wie viele mögliche Schritte für ein gegebenes Raster existieren und zweitens, wie viele verschiedene Kombinationen sich aus diesen Schritten ergeben. Diese beiden Angaben werden uns auch helfen, effizientere Algorithmen zu konstruieren, weshalb wir uns nun genauer mit ihnen beschäftigen werden.

Graphenmodell zur Zyklenfindung

Wir stellen die Situation in einem Graphen G dar. Jeder Knoten des Graphen steht für ein Hausdach, auf dem sich ein Paket befindet. Die (vier) Himmelsrichtungen, in die jedes Paket geworfen werden kann, werden durch gerichtete Kanten dargestellt (vgl. Abbildung 3). Eine Kante von *A* nach *B* heißt also, dass das Paket auf dem Dach *A* zum Dach *B* geworfen

[4] Das ist die durch die Würfe entstandene, neue Verteilung der Pakete.

wird. Es gibt auch eine Kante von B nach A, die den Wurf des Paketes, dass sich aktuell auf B befindet, zum Dach A repräsentiert. Diese beiden Kanten zusammen stellen schon einen gültigen Wurfkreis dar, nämlich den Sonderfall $n = 2$ Würfe, bei dem sich zwei benachbarte Einwohner gegenseitig ihr Paket zuwerfen (Tausch). Allgemeiner ist ein gültiger Wurfkreis ein Kreis des Graphen G - also ein Zyklus, bei dem *ausschließlich* Start- und Zielknoten übereinstimmen. Gesucht sind bei unserem Brute-Force-Ansatz sowohl alle Kreise von G als auch alle möglichen Kombinationen aus mehreren Kreisen, die keine gemeinsamen Knoten haben, denn es gilt ja z. B. als gültiger Schritt, wenn mehrere Bewohner gleichzeitig ihr Paket mit dem Nachbarn tauschen (sofern es sich nicht um ein und den selben Nachbarn handelt).

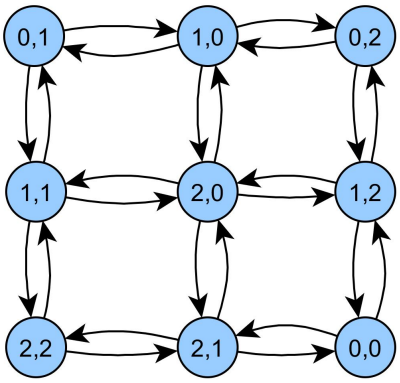


Abb. 18: Grundmuster einer 3×3-Siedlung.

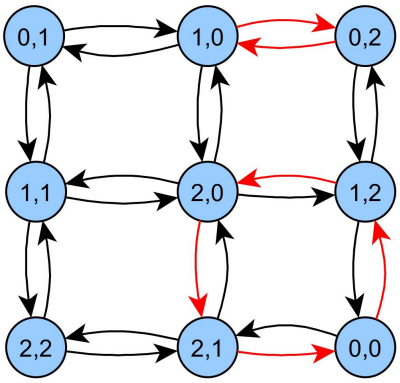


Abb. 19: Kanten von zwei angewendeten Kreisen.

In einem Raster der Größe $n \times n$ gibt es genau $k = 2 \cdot 2 \cdot n \cdot (n-1)$ Kanten. In jeder Zeile gibt es $n-1$ "Querverbindungen" zwischen den Häusern und es gibt n Zeilen. Das gleiche gilt für die Spalten ($\cdot 2$). Außerdem besteht jede Querverbindung aus zwei Kanten (Hin- und Rückkante, $\cdot 2$). Eine Abschätzung für die Anzahl an Kreisen der Länge $2i; i = 1,2,3\ldots$ liefert uns der Binomialkoeffizient $\binom{k}{2i}$, denn so viele Möglichkeiten gibt es, $2i$ Kanten zufällig aus der Menge von k Kanten auszuwählen. Die wirkliche Anzahl an Kreisen ist jedoch deutlich kleiner, denn eine zufällige Stichprobe kann auch ein ungültiger Kreis sein, wenn die Kanten gar nicht benachbart sind. Der Wert

$$\mathcal{O}\left(\sum_{i=1}^{2i<k/2} \binom{k}{2i}\right) \tag{2}$$

ist also eine obere Schranke für die Zahl aller Kreise und deren Kombinationen im Graphen. Nun ist folgende Verfeinerung möglich: wir untersuchen die Wahrscheinlichkeit, dass eine Stichprobe der Größe $2i$ ein gültiger Kreis ist. Wählen wir die erste (zufällige) Kante der Stichprobe, so ist es entweder eine Eckkante (dann gibt es nur eine Richtung, in die man von ihrem Zielknoten aus gehen kann), eine Randkante (zwei Richtungen) oder eine normale Kante (drei Richtungen). In einem Graphen für eine $n \times n$-Stadt gibt es 4 Eckhäuser, $4 \cdot (n-2)$ Randhäuser und $(n-2)^2$ normale Häuser. Ein zufälliges Haus hat deshalb nach dem Erwartungswert

$$1 \cdot \frac{4}{n^2} + 2 \cdot \frac{4n-8}{n^2} + 3 \cdot \frac{n^2-4n+4}{n^2} = 3 - \frac{4}{n} \tag{3}$$

Nachbarn. Die Wahrscheinlichkeit, dass einer dieser Nachbarn über eine Kante zu der Stichprobe gehört, beträgt $\frac{2i-1}{n^2-1}$, für alle Nachbarn insgesamt also $(3-\frac{4}{n})\cdot\frac{2i-1}{n^2-1}$. Wiederholt man dies für die verbleibenden $2i-2$ Kanten der Stichprobe[5], multiplizieren sich die Wahrscheinlichkeiten zu

$$\left(3-\frac{4}{n}\right)\cdot\frac{2i-1}{n^2-1}\cdot\left(3-\frac{4}{n}\right)\cdot\frac{2i-2}{n^2-2}\cdot\ldots=\left(3-\frac{4}{n}\right)^{2i-1}\cdot\frac{(n^2-2i)!}{(n^2-1)!}\cdot(2i-1)!. \qquad (4)$$

Diese Wahrscheinlichkeit ist deutlich kleiner als 1, und die Anzahl an Kreisen verringert sich zu

$$\mathcal{O}\left(\sum_{i=1}^{2i<k/2}\binom{k}{2i}\cdot\left(3-\frac{4}{n}\right)^{2i-1}\cdot\frac{(n^2-2i)!}{(n^2-1)!}\cdot(2i-1)!\right). \qquad (5)$$

Da hier aber keine Kombinationen aus Kreisen gezählt werden, handelt es sich bei diesem Wert um eine untere Grenze. Rechnet man für Amacity ($n=10$) nach, ergeben sich bei einer maximalen Kreislänge von 40 zwischen $7\cdot10^{41}$ (Untergrenze) und $2\cdot10^{53}$ (Obergrenze) Möglichkeiten.

Entscheidungsbaum mit *Branch and Bound*

Die Suche nach der besten Lösung wollen wir in einem Entscheidungsbaum umsetzen, bei dem ein Knoten für eine aktuelle Paketverteilung und eine Kante für einen möglichen Schritt steht (*Branch*). Ausgangspunkt (Startverteilung der Pakete) ist die Wurzel des Baumes, die Kinder dieses Knotens sind alle Verteilungen, die sich durch gültige Würfe von Paketen erreichen lassen. Ein gültiger Wurfplan ist dann ein Pfad von der Wurzel des Baumes bis zu einem Blatt (vollständig korrekte Paketverteilung). Aus allen möglichen Pfaden soll dann der optimale ausgewählt werden, was nach unser Zieldefinition der kürzeste Pfad ist. Da wir an dieser Stelle *alle* Schritte betrachten, auch solche, die völlig unsinnig sind und die Paketverteilung gar nicht sortieren, gibt es in unserem Baum diverse Pfade, die unendlich lang sind. Aus diesem Grund schneiden wir Pfade, die eine nach einer festgelegten Schranke bestimmte Länge überschreiten, einfach ab (*Bound*).[6]

Wir kennen bereits die Anzahl z aller möglichen Schritte für ein Raster (z gibt an, wie viele Kinder ein Knoten haben kann). Unser Baum hat in der Ebene $e\in\{1,2,3,\ldots\}$ deshalb exakt z^{e-1} Knoten. Nehmen wir für Amacity nun eine Tiefe von 20 als Schranke[7], dann gäbe es z^{20} mögliche Pfade, die wir untersuchen müssten, konkret also mindestens $(7\cdot10^{41})^{20}\approx8\cdot10^{836}$ Pfade. An dieser Stelle wird offensichtlich, wieso ein reiner Brute-Force-Ansatz nicht zum Ziel führen kann. Da wir nicht wissen, ob die Optimallösung überhaupt erreicht werden kann,

[5] Folgende Dinge wurden hierbei aus Gründen der Einfachheit außer Acht gelassen: 1. Beim Sonderfall $2i=2$ ist es möglich, sowohl die Hin- als auch die Rückkante zu nutzen. 2. Wurden bereits Kanten ausgewählt, so verändert sich der Erwartungswert der Anzahl an Nachbarn der verbleibenden Kanten. Da wir aber nicht unterscheiden können, welchen Typ (Ecke, Rand, Mitte) wir ausgewählt haben, lassen wir diese (eher marginale) Abweichung zu.

[6] Als guter Richtwert dient hierbei d_{ManMax} aus der Optimallösung.

[7] Die Optimallösung würde 15 Schritte benötigen (vgl. d_{ManMax}), aber wir lassen noch Platz für 5 mögliche Engstellen, auf die wir treffen könnten.

können wir auch keine Pfade vorzeitig abschneiden. Mitunter müsste sogar noch eine größere Schranke gewählt werden. Außerdem steigt die Anzahl der zu betrachtenden Pfade mit größerem Spielplan (sollte Amacity beispielsweise in Zukunft vergrößert werden) explosionsartig.

Verfeinerung und Heuristik

Wir wollen die Konzepte der Brute-Force-Methode jedoch nicht ganz verwerfen. Denn wir können die Anzahl an möglichen Pfaden erheblich reduzieren, wenn wir nur *sinnvolle* Pfade betrachten. Außerdem bieten uns Heuristiken die Möglichkeit, unter Umständen gar nicht alle Pfade betrachten (ergo auch nicht berechnen) zu müssen. Dazu werden wir die Auswirkungen von Würfen genauer untersuchen.

Sinnvolle Wurfkreise Es gibt maximal zwei Himmelsrichtungen, in die ein Paket geworfen werden sollte, wenn man seine Zieldistanz verbessern möchte. Befindet sich ein Paket bereits in der richtigen Zeile oder Spalte, dann gibt es nur eine Richtung. Ein Wurf in eine andere Richtung wirkt der Auslieferung an die richtige Adresse entgegen. Es können jedoch nicht immer nur Pakete näher an ihr Ziel gebracht werden; um Distanzverbesserungen bei bestimmten Paketen zu erreichen, müssen mitunter Verschlechterungen bei anderen in Kauf genommen werden. Wie schon angedeutet, sollte die Manhattan-Zieldistanz eines Paketes ausschlaggebend für die Priorisierung sein. Wir beginnen mit einer bestimmten maximalen Manhattan-Distanz und sollten versuchen, diese mit jedem Schritt um den Wert eins zu verkleinern, bis wir bei null angekommen sind, dann sind alle Pakete richtig zugeordnet. Daraus folgt:

- Rückschritt-Kriterium: Es sollte niemals ein Alphapaket weiter von seiner Zieladresse entfernt werden, da dies ein Rückschritt wäre. Die neue Maximaldistanz wäre um eins größer und wir würden somit auf das Niveau einer früheren Runde zurückfallen.
- Optimal-Alpha-Kriterium: Wenn möglich, so sollten alle Alphapakete näher an ihre Zieladresse gebracht werden, damit weiterhin die Optimallösung erreicht werden kann. Deshalb muss Alphapaketen vor allen anderen Pakettypen unbedingt Vorrang gewährt werden, wenn dadurch die Optimallösung realisiert wird.
- Optimal-Beta-Kriterium: Um die Optimallösung zu erreichen, muss außerdem unbedingt vermieden werden, dass Betas entgegen ihrer Zielrichtung geworfen werden, da sie sonst zu Alphas mit Distanz d_{ManMax} werden.

Die letzten beiden Kriterien sind also gleichwertig. Wenn ein Alphapaket in die richtige Richtung, dafür aber ein Betapaket in die falsche geworfen wird, so ist die maximale Distanz immer noch d_{ManMax} und es wurde keine Verbesserung erreicht.

Erfüllt ein Kreis oder eine Kombination aus Kreisen alle drei Kriterien, dann können wir weiterhin die Optimallösung erreichen. Gibt es hingegen nur Kreise, die eines oder mehrere Kriterien verletzen, dann liegt ein Engpass vor. Auch in solch einer Situation können diese Kriterien in leicht abgewandelter Form eingesetzt werden:

- Engstelle-Kriterium: Es soll der Kreis ausgewählt werden, in dem die Summe aller richtig geworfenen Alphas abzüglich der Summe aller falsch geworfenen Betas am größten ist.

Wie genau Engstellen zustandekommen und wie man sie vermeiden kann, soll im folgenden Abschnitt diskutiert werden.

Heuristik zur Vermeidung von Engstellen Wie schon angedeutet, sollte die Manhattan-Zieldistanz eines Paketes ausschlaggebend für die Priorisierung sein. Allerdings basieren die bisherigen Kriterien vor allem auf eine Greedy-Strategie, bei der sehr wenig auf die Auswirkungen der Schrittwahl auf die kommenden Runden geachtet wird, sodass sehr leicht Engpässe entstehen könnten. Wie besprochen liegt z. B. ein Engpass vor, wenn Paket *A* nach Süden geworfen werden muss, Paket *B* nach Norden, aber die beiden sich zudem in der selben Spalte befinden (Abb. 16). Das Problem rührt daher, dass es keine alternativen Wurfrichtungen für die beiden Alphapakete mehr gibt, der Handlungsspielraum also minimal ist. Gleiches gilt für die Engstelle in Abb. 17. Folgende Kriterien sollen den Handlungsspielraum im Laufe der Runden möglichst groß halten:

- Voraussicht-Kriterium: Auch wenn nicht zur Optimallösung erforderlich, ist es sinnvoll, aus zwei sonst gleichwertigen Kreisen denjenigen auszuwählen, der neben den obligatorischen Alphapaketen auch mehr sonstige Pakete in die richtige Richtung verschiebt, bzw. weniger in die falsche Richtung verschiebt. Auf diese Weise soll die Anzahl an Alphas in zukünftigen Runden klein gehalten werden.
- Bias-Kriterium: Das *Bias* sei die *Voreingenommenheit* eines Paketes, in eine bestimmte Richtung geworfen werden zu müssen. Das Bias errechnet sich aus der absoluten Differenz zwischen x- und y-Entfernung eines Paketes von der Zieladresse. Ein Paket, das 4 Spalten und 1 Zeile vom Zielort entfernt ist, hat ein Bias von 3. Je kleiner das Bias eines Paketes ist, desto größer ist der Handlungsspielraum beim Weiterwerfen (möglichst lange in zwei Richtungen).

  Auch wenn nicht zur Optimallösung erforderlich, ist es sinnvoll, aus zwei sonst gleichwertigen Kreisen denjenigen auszuwählen, durch den die Biaswerte der Pakete am kleinsten werden.

Die Anwendung dieser Kriterien macht das Entstehen von Engstellen nicht unmöglich, verkleinert aber die Wahrscheinlichkeit. Wir können nun folgende Effekte betrachten: Erstens werden die am weitesten entfernten Pakete vorrangig behandelt, Pakete die sehr nahe am Ziel sind, sind benachteiligt. Dies führt dazu, dass die Bewohner von Amacity ihre eigenen Pakete tendenziell später, aber dafür gleichzeitiger erhalten. Dies sollte jedoch sogar im Sinne der Gemeinschaft sein, da so die Bereitschaft, Pakete weiterzuwerfen lange groß bleibt (wohingegen es frustrierend ist, das Paket für das eigene Haus schon in den Händen zu halten, es aber dann doch wieder weiterwerfen zu müssen). Das Bias-Kriterium führt dazu, dass Pakete tendenziell eher einen Zickzack-Weg zu ihrem Ziel zurücklegen als erst entlang einer Spalte und dann entlang einer Zeile zu wandern.

Mit dem gefundenen Kriteriensatz kann die Zahl an zu betrachtenden Pfaden deutlich reduziert werden. Die Optimallösungskriterien/das Engstellenkriterium ermöglichen uns, nur noch

bestimmte Kanten des Paketgraphen für gültige Wurfkreise in Betracht zu ziehen, was die Menge an Kindern und somit die Breite des Entscheidungsbaums auf jeder Ebene verringert. Die Heuristik zur Vermeidung von Engstellen können wir dazu nutzen, aus einer Menge von Schritten denjenigen mit der besten Bewertung (gemäß unserer Kriterien) zu wählen und nur den zugehörigen Pfad im Entscheidungsbaum weiter zu berechnen.

Zur Sicherstellung der optimalen Lösung kann zusätzlich folgender Backtrack-Ansatz genutzt werden: Wird im aktuellen Pfad trotzdem eine Engstelle erreicht, so können wir den Baum wieder hinaufwandern und andere Pfade nachträglich ausprobieren. Sollte sich eine Engstelle in allen möglichen Pfaden ergeben, dann ist diese unausweichlich und wir wenden die Engstellen-Kriterien an, um fortzuschreiten. Auf diese Weise werden zwar mitunter doch *alle* Pfade ausprobiert, aber eben nur *bei Bedarf*, was viel Rechenaufwand einspart.

2.4 Grundlagen der Implementierung des Verfahrens

Wir abstrahieren Amacity wie besprochen in einem Graphen, wozu sich ein objektorientierter Ansatz empfiehlt. Dazu schreiben wir Klassen für die Bestandteile des Graphen: *Roof* als Knoten (Dach), *Parcel* als Inhalt der Knoten (Paket), *RoofEdge* als Kante zwischen zwei Dächern (Wurf). Eine Kante speichert neben dem Herkunftsdach und Zieldach des zu werfenden Paketes auch die Veränderung der Manhattan-Zieldistanz des Paketes sowie die Veränderung des Bias (beides binäre Werte, -1 oder 1). Wurfkreise werden durch die Klasse *Cycle* modelliert, die nicht nur alle Kanten speichert, die den Kreis bilden, sondern auch die akkumulierten Werte ihrer Distanz- und Biasveränderugen. Der Startzustand von Amacity wird aus einer Datei eingelesen und der zugehörige Graph erstellt. Kreise können ähnlich eines Wegfindungsalgorithmus gefunden werden. Von einem Startknoten aus werden alle gemäß unserer Optimallösungs-Kriterien gültigen Kanten ausgewählt und von deren Zielknoten aus rekursiv weiter gesucht. Wird ein bereits besuchter Knoten gefunden, der nicht der Startknoten ist, oder gibt es keine gültigen Kanten mehr vom aktuellen Knoten aus, dann ist der aktuelle Kreis ungültig. Wird jedoch der Startknoten wieder erreicht, so ist ein gültiger Kreis gefunden. Da wir vorrangig Alphapakete verschieben wollen, suchen wir alle Kreise von den Startknoten aus, auf denen momentan ein Alpha liegt. Sind alle Kreise im Graphen gefunden, so müssen noch ihre möglichen Kombinationen bestimmt werden. Dazu tragen wir alle Kreise in einen weiteren Graphen, den Optimierungsgraphen ein. Jeder Knoten steht für einen Kreis. Wir zeichnen eine Kante zwischen zwei Kreisen, wenn sie keine gemeinsamen Knoten haben. So ist eine gültige Kombination aus Wurfkreisen nämlich eine Clique des Optimierungsgraphen. Aus allen Cliquen des Graphen wählen wir dann diejenige aus, deren Wurfkreise nach unseren Kriterien die beste Wertung erreicht. Berücksichtigt wird also: Anzahl richtig verschobener Alphas, falsch verschobener Betas, richtig verschobener Betas und Gesamtveränderung der Distanzen aller restlichen Pakete, sowie Biasveränderungen der Alphas und Betas.

Anschließend werden die Wurfkreise der besten Clique in den Plan als Schritt aufgenommen und auf Amacity angewendet, d. h. die neue Paketverteilung errechnet. Von der neuen Verteilung aus werden die obigen Schritte solange wiederholt, bis alle Pakete richtig verteilt wurden (dann ist $d_{\text{ManMax}} = 0$). Um den Backtrack-Ansatz einzubauen, müssen die errechneten Cliquen entweder zwischengespeichert werden, damit nachher noch darauf zurückgegriffen werden kann, oder erneut errechnet werden - je nach dem, ob man mehr Speicherplatz oder

mehr Rechenkapazität und Zeit zu Verfügung hat. Die einzelnen Schritte der Lösung werden zu einem Wurfplan zusammengefasst und im vorgegebenen Format gespeichert. Zusätzlich sollte die Lösung grafisch ausgegeben werden, z. B. als Bildfolge oder Animation.

2.5 Bewertung des gefundenen Verfahrens

Wie bereits beschrieben, ist durch die Kriterien und die Heuristik unser Ansatz deutlich effizienter als eine Brute-Force-Lösung, und man könnte durch einen Backtrack-Ansatz dennoch die optimale Lösung garantieren. Es ist trotzdem immer noch ein großer Aufwand nötig, um das Verfahren anzuwenden. Ausschlaggebend sind hierbei die zu errechnenden Kreise und Cliquen. Die Menge an Kreisen ist vor allem von der Anzahl an Alphapaketen abhängig, die in der ersten Runde lediglich eins beträgt. Da aber in der Optimallösung d_{ManMax} jede Runde um eins kleiner wird, werden immer mehr Pakete den Alpha-Status erreichen. Ist ein Paket einmal ein Alpha, so wird es in der Optimallösung auch eines bleiben. Die Anzahl an Alphas steigt also kontinuierlich. Nehmen wir nun z. B. an, dass 90% aller Pakete gegen Ende Alphastatus erreichen werden (10% werden schon vorher richtig zugestellt) und dass die Menge an Alphas der Einfachheit halber linear zunimmt (in Wahrheit sollte sie eher exponentiell zunehmen und das Mittel deshalb kleiner sein), dann müssen wir durchschnittlich in jeder Runde die Kreise von 45 Startknoten aus berechnen. Von einem Startknoten aus gibt es geschätzt $(3-\frac{4}{n})^{i-1}$ Pfade der Länge $2i$, die wir mitunter alle betrachten müssten. Somit gäbe es $\mathcal{O}\left(45\cdot(3-\frac{4}{n})^{i-1}\right)$ zu betrachtende Pfade. Allerdings fallen alle Pfade weg, die Kanten haben, die unseren Kriterien widersprechen, die eigentliche Menge kann also deutlich kleiner sein. Dennoch muss die maximale Kreisgröße eingeschränkt werden, um realistische Laufzeiten zu erreichen. Die Cliquenfindung in beliebigen Graphen hat für die Größe k eine Laufzeit von $\binom{|V(G)|}{k} = \mathcal{O}(|V(G)|^k)$, auch hier muss also die Cliquengröße eingeschränkt werden. Die benötigten Berechnungen für jede Runde addieren sich durch die Nutzung der Heuristik aber lediglich. Das gefundene Verfahren hat für eine beliebige Siedlung der Größe $n \times n$ aufgrund der Kreis- und Cliquenfindung keine polynomielle Laufzeit. Jedoch kann durch geeignete Grenzen und für Amacity ($n = 10$) eine Lösung errechnet werden, für kleinere Siedlungen ist dieses Verfahren schneller. Wird beispielsweise eine maximale Kreislänge von 12 gewählt (dabei werden in manchen Runden bis zu 6000 ($\approx (3-\frac{4}{10})^{10-1}$) Zyklen untersucht), dann findet sich eine Lösung, die alle Pakete in 34 Schritten verteilt. Diese ist 7 Schritte schneller als die Beispiellösung. Durch effiziente Datenstrukturen und Berechnungsmodelle kann die Laufzeit weiter gesenkt werden.

2.6 Erweiterungsmöglichkeiten und fortgeschrittene Fragestellungen

Hier einige Ideen:

- Metrik: Je nach Wurfkraft der Einwohner von Amacity könnten auch Würfe zu diagonal benachbarten oder noch weiter entfernten Dächern erlaubt werden. Dazu müssten lediglich die Metrik angepasst und im Graphen neue Kanten gezeichnet werden. Zwar erhöht das den Rechenaufwand, aber tendenziell können so deutlich bessere Ergebnisse erzielt werden.

- Feldgröße und -maße: Unser Ansatz ist generell gehalten und kann leicht auf andere Städte der Größe $n \times n$, auf rechteckige Siedlungen der Größe $n \times m$ oder sogar auf ungewöhnliche konkave oder konvexe Formen angewendet werden, da wir lediglich den Graphen anpassen müssen.
- Gütekriterien: Es sind weitere Bewertungskriterien für die Qualität des Wurfplans denkbar, z. B. bzgl. der Anstrengung der Bewohner, etwa die Anzahl an benötigten Würfen oder auch die Verteilung der Pfade der Pakete (es werden eventuell manche Bewohner stärker belastet als andere). Da unser Lösungsraum jedoch sowieso sehr beschränkt ist, kommen diese Gütekriterien jedoch sowieso wenig zum Einsatz, oder diese müssten auf Kosten der Anzahl an Schritten optimiert werden.
- Paketpriorisierung: Vielleicht müssen manche Pakete doch dringender als andere ausgeliefert werden. Dann könnten diese z. B. immer wie Alphapakete bevorzugt behandelt werden. Wenn sie ihr Ziel erreicht haben, wird das Dach aus dem Graphen entfernt (Haus beteiligt sich nicht mehr an Paketverteilung, sondern öffnet das erhaltene Paket) und normal fortgefahren.

2.7 Auszüge aus der Lösung in 34 Schritten

Die Pfeile zeigen Wurfkreise und die kleinen Zahlen in jeder Zelle die Zieldistanz des sich dort befindenden Pakets an.

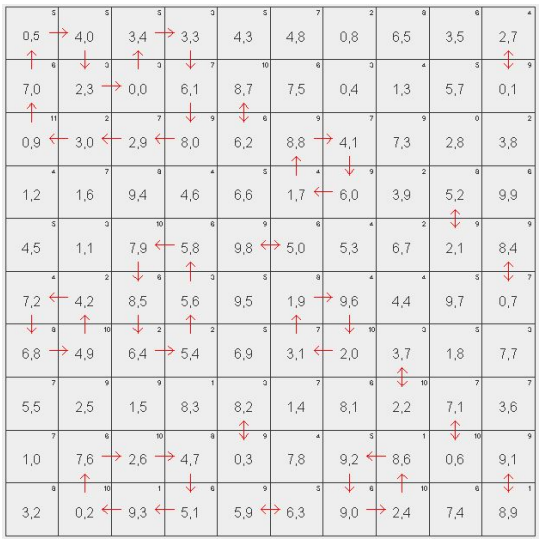


Abb. 20: Schritt 5, diverse Kreise.

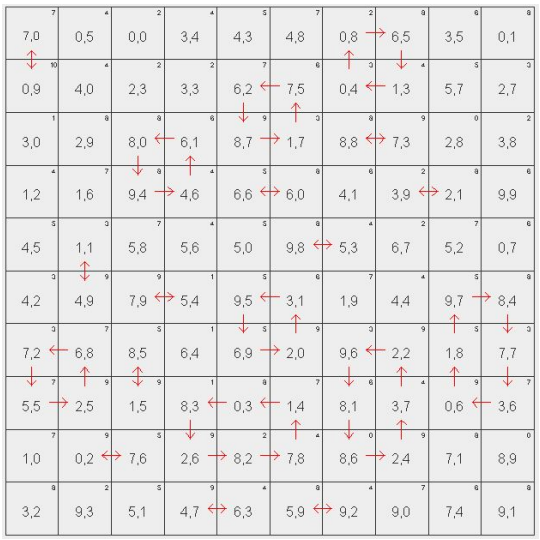


Abb. 21: Schritt 6, sehr nahe aneinander liegende Kreise.

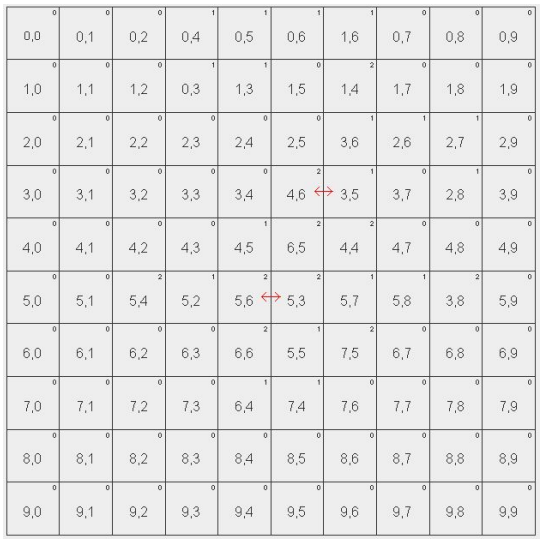


Abb. 22: Schritt 28, nur zwei Tauschoperationen.

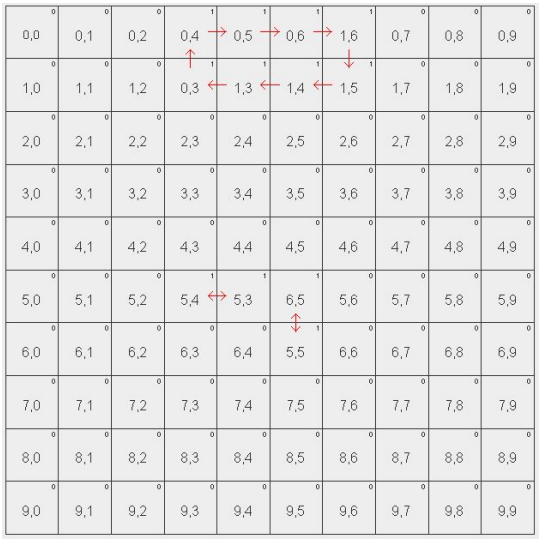


Abb. 23: Schritt 34, alle verbleibenden Pakete werden an ihre Zieladresse befördert.

2.8 Alternative Ansätze

Offensichtlich handelt es sich bei der Paketverteilung um ein schwieriges Problem. Verfahren, die garantiert optimale Lösungen bestimmen, sind in ihrer Laufzeit unbrauchbar. Der oben beschriebene Ansatz versucht, die Menge der möglichen Pläne durch heuristische Kriterien einzuschränken. Erstaunlich ist, dass sich zumindest für die hier konkret eingeforderte Verteilung für Amacity deutlich kompaktere Pläne bestimmen lassen, wenn noch stärker heuristisch vorgegangen wird. Auf durchaus unterschiedlichen Wegen lassen sich Pläne bestimmen, die in nur 17 Schritten alle Pakete in Amacity an den richtigen Platz bringen und damit nur zwei Schritte über der unteren Schranke von 15 Schritten liegen. Die vorliegenden Lösungen setzen vielfach heuristische Verfahren, die sich oberflächlich betrachtet oft deutlich unterscheiden. Im Wesentlichen werden aber folgende Ideen doch häufiger in ähnlicher Weise verwendet (vgl. den Abschnitt „Verfeinerung und Heuristik“):

Entfernte Pakete bevorzugt Pakete mit der größten Zieldistanz werden bevorzugt, evtl. sogar überproportional, z. B. indem die (als Manhattan-Distanz bestimmte) Zieldistanz potenziert wird.

Geringe Bewegungsfreiheit bevorzugt Interessant ist, dass die obige Idee verbessert werden kann, wenn die Zieldistanz *nicht* die klassische Manhattan-Distanz (Summe der Koordinatendifferenzen $x_\Delta + y_\Delta$) ist. Wenn man stattdessen etwa die Quadratsumme der Koordinatendifferenzen verwendet ($x_\Delta^2 + y_\Delta^2$), werden Pakete bevorzugt, deren optimaler Weg zum Ziel eher eine gerade Strecke bildet. Solche Pakete haben wenig Bewegungsfreiheit, während Pakete mit eher diagonalem Weg zum Ziel viele Möglichkeiten haben, diesen Weg zurückzulegen.

Tausch-Bewertung: Verbesserung bevorzugt Bei der Ermittlung möglicher (Ring-)Tausche werden diejenigen bevorzugt, die insgesamt die größte Auswirkung auf die Zieldistanzen haben. Auch hier können unkonventionelle Varianten sinnvoll sein: Wenn zur Bewertung eines Tausches nicht die Verbesserung der Zieldistanzen, sondern die Summe der Zieldistanzen der beteiligten Pakete summiert wird, werden Tausche, an denen Pakete mit großer Zieldistanz beteiligt sind, stärker bevorzugt.

Seiteneffekte berücksichtigen Jeder Wurf eines Paketes macht andere Würfe unmöglich. Deshalb sollte ein Wurf (oder auch ein kompletter Tausch) nicht nur nach seinen direkten Auswirkungen bewertet werden, sondern auch danach, welche Auswirkungen er verhindert. Hier gilt: Das Bessere ist des Guten Feind.

Starke Fokussierung der Lösungssuche Auch wenn sich nur auf direkte Paartausche und Ringtausche mit vier direkt benachbarten Feldern beschränkt wird, lassen sich gute Ergebnisse erzielen, zumindest für Amacity. Schon weniger eingeschränkt ist eine Art Tiefensuche nach möglichen Tauschen, bei denen nur die aktuell am besten bewertete Wurfkette um weitere Würfe erweitert wird – bis es nicht mehr geht, oder bis sie einen Ringtausch bildet. Wenn aus den im Wurfgraph aktuell vorhandenen Zyklen (also den möglichen Tauschen) der nächste Schritt bestimmt werden soll, ist es sinnvoll, nicht alle Mengen disjunkter (also parallel anwendbarer) Tausche zu zu betrachten, sondern auch hier Seiteneffekte zu berücksichtigen und schrittweise diejenigen Tausche mit dem größten „Verhinderungspotenzial“ auszuschließen, bis eine einzige Menge disjunkter Tausche gefunden ist.

Auflösung bzw. Vermeidung von Sackgassen Je stärker fokussiert die Lösungssuche ist, desto wahrscheinlicher ist es, dass die Verteilung der Pakete in eine Sackgasse läuft, etwa weil Pakete sich nicht mehr von ihrem Zielfeld weg bewegen lassen. Dem kann entgegnet werden, indem an geeigneter Stelle, etwa bei der Tausch-Bewertung, Zufallswerte verwendet werden. Wie allgemein bei randomisierten Verfahren wird es dann aber nötig, die Lösungssuche mehrfach durchzuführen und aus den gefundenen Lösungen die beste auszuwählen.

2.9 Bewertungskriterien

- Der Ablauf und die Regeln der Paketverteilung wurden korrekt erkannt und umgesetzt. Einzelne Würfe und Wurfkreise können in der Implementierung regelkonform angewandt werden, es befindet sich immer nur ein Paket auf jedem Dach.
- Das Verfahren bestimmt insgesamt korrekte Pläne zur Paketverteilung; insbesondere muss sich jedes Paket zuletzt auf dem Zieldach befinden.
- Es wurden sinnvolle Kriterien zur allgemeinen Bewertung von Schritten (einzelnen Würfen oder ganzen Wurfkreisen) begründet aufgestellt und angewandt. In der Regel sollte dabei die Zieldistanz mit der Manhattan-Distanz (die Nennung dieses Begriffs ist aber nicht notwendig) berechnet werden; begründete Ausnahmen sind aber möglich.
- Wie gezeigt gibt es immer eine Lösung; ein Verfahren sollte also für alle gültigen Konstellationen (jedes mögliche Ziel kommt genau einmal vor) auch eine Lösung finden.

Bei Ansätzen, die in Sackgassen führen könnten (z. B. durch starke Einschränkungen des Lösungsraums), sollten die potenziellen Schwächen erkannt und durch geeignete Gegenmaßnahmen behoben sein. Eine Lösung für Amacity muss in angemessener Zeit gefunden werden.

- Das Verfahren findet gute Lösungen für Amacity. Die untere Schranke $d_{\text{ManMax}} = 15$ für Amacity kann erreicht werden; es gibt also einen Lösungsplan für Amacity, der mit 15 Schritten auskommt. Das Verfahren sollte einen Plan liefern, der deutlich weniger als die 41 Schritte des von BwInf vorgegebenen Plans benötigt. Pluspunkte gibt es für Pläne mit 20 und weniger Schritten, Minuspunkte für Pläne mit mehr als den hier angegebenen 34 Schritten.
- Das Verfahren sollte verallgemeinerbar sein, d. h. es sollte prinzipiell auch auf andere Paketverteilungen oder Spielplangrößen als Amacity angewandt werden können.
- Es werden Überlegungen zur Laufzeit angestellt und ineffiziente Verfahren erkannt.
- Es sollte auch unabhängig vom Ergebnis für das Beispiel Amacity begründet werden, wieso die Lösung gut ist. Zur Abschätzung der Lösungsqualität ist die maximale Zieldistanz (Manhattan-Distanz) als untere Schranke für die Schrittzahl hilfreich.
- Ein Plan für Amacity sollte angegeben sein. Weitere eigene Beispiele sind nicht gefordert, aber lobenswert.
- Das Programm funktioniert mit dem von der Aufgabe spezifizierten Format; es kann Karten einlesen sowie die Lösung korrekt ausgeben.
- Schön ist, wenn das Programm die Lösung auch grafisch ausgeben kann, insbesondere durch eine Animation.

Aufgabe 3: Torkelnde Yamyams

3.1 Einleitung

Wir wollen herausfinden, von welchen Startfeldern aus ein torkelnder Yamyam mit Sicherheit irgendwann einen Ausgang erreicht. Dass wir etwas „mit Sicherheit irgendwann erreichen“ bedeutet, dass etwas im Grenzwert der Zeit gegen unendlich mit Wahrscheinlichkeit 1 erreicht wird, oder mathematisch ausgedrückt: $\lim_{t\to\infty} P_{\text{exit}}(t) = 1$, wobei $P_{\text{exit}}(t)$ die Wahrscheinlichkeit angibt, nach maximal t Zeitschritten einen Ausgang erreicht zu haben.

Wie wir sehen werden, gibt es verschiedene Lösungsansätze, von denen einige ein sehr gutes Laufzeitverhalten haben, wobei im Gegenzug aber weniger einsichtig ist, dass sie korrekt funktionieren. Andere Verfahren brauchen relativ lange für das Finden der Lösung, funktionieren aber offensichtlich korrekt. Wir werden daher zunächst ein langsames Verfahren erläutern, das auf dem Lösen eines Gleichungssystems beruht, und schließlich ein schnelles, das auf einem modifizierten Flood-Fill-Algorithmus beruht.

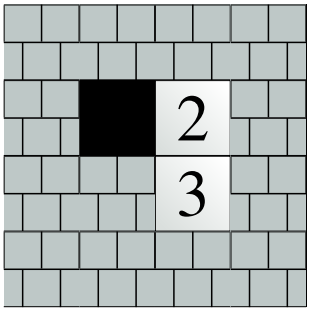
Die ansonsten klare Aufgabenstellung lässt Interpretationsspielraum in zwei Punkten: Ein Yamyam läuft stets so weit, bis er gegen eine Wand stößt. Gilt ein Ausgang nur dann als erreicht, wenn der Yamyam auf ihm zum Stehen kommt, oder genügt es, auf dem Weg über einen Ausgang zu stolpern? Und ist ein Ausgang auch dann schon erreicht, wenn es der Startpunkt ist; ist also jeder Ausgang notwendigerweise ein sicheres Feld?

In der Beispielwelt, die auf dem Aufgabenblatt abgedruckt ist, macht es keinen Unterschied wie die Regel interpretiert wird. Jedoch hat beispielsweise Beispiel 4 (siehe am Ende der Dokumentation) nur einen Ausgang, der in der Mitte des Raumes liegt, also nur erreicht werden kann, wenn das Darüberlaufen ausreicht. Wir gehen daher davon aus, dass es ausreicht, unterwegs an einem Ausgang vorbeizukommen. Das legt dann auch nahe, dass ein Yamyam, das auf einem Ausgang startet, diesen auch sofort erreicht.

3.2 Gleichungssystem und Gauß-Verfahren

Idee

Die Wahrscheinlichkeiten der Felder, den Ausgang zu erreichen, lassen sich insgesamt als lineares Gleichungssystem beschreiben. Wir betrachten zunächst folgendes kleines Beispiel:



Wir benennen die Felder wie im Bild von 1 bis 3, wobei der Ausgang Feld 1 sei, und die zugehörigen Wahrscheinlichkeiten, einen Ausgang zu erreichen, mit p_1 bis p_3. Da Feld 1 ein Ausgang ist, ist $p_1 = 1$. Von p_2 aus wählt ein Yamyam mit der Wahrscheinlichkeit $1/2$ den Weg nach links auf Feld 1, also den Ausgang, und ebenfalls mit Wahrscheinlichkeit $1/2$ den Weg nach unten auf Feld 3. Man kann also p_2 bestimmen zu

$$\begin{aligned} p_2 &= \frac{1}{2}p_1 + \frac{1}{2}p_3 \\ &= \frac{1}{2} + \frac{1}{2}p_3. \end{aligned} \tag{6}$$

Gleichzeitig gilt für das Feld 3, dass man mit Sicherheit nach oben auf Feld 2 läuft, also

$$p_3 = p_2. \tag{7}$$

Man kann Gleichung (6) in Gl. (7) einsetzen und p_2 und damit auch p_3 zu bestimmen: $p_2 = p_3 = 1$. In diesem Beispiel sind also alle Felder sicher. Dies mag einigen verwirrend vorkommen, schließlich ist es theoretisch möglich, dass man ständig zwischen Feld 2 und 3 hin und her läuft, ohne jemals den Ausgang zu erreichen. Tatsächlich ist die Wahrscheinlichkeit dafür aber verschwindend gering: Steht man auf Feld 2, so wählt man mit Wahrscheinlichkeit $1/2$ den Weg nach unten und wieder nach oben, erreicht also das Ziel nicht. Die Wahrscheinlichkeit, das gleiche anschließend erneut zu tun, beträgt $p = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$. Um n mal den unteren Weg zu wählen, bedarf es schon einer Wahrscheinlichkeit von $p = \frac{1}{2^n}$. Um das Ziel aber nie zu erreichen, muss dieser Weg unendlich oft gewählt werden, aber $\lim_{n\to\infty} \frac{1}{2^n} = 0$. Die Gegenwahrscheinlichkeit, das Ziel irgendwann zu erreichen, beträgt also $1 - p = 1$.

Das Verfahren im Beispiel lässt sich auch auf beliebig große Beispiele verallgemeinern. Jedem Feld i kann eine Wahrscheinlichkeit p_i, den Ausgang zu erreichen, zugeordnet werden. Kann man vom Feld i aus die Felder i_1 bis i_k erreichen (wobei k zwischen 1 und 4 liegt, je nachdem, in wie viele Richtungen man gehen kann), so gilt

$$p_i = \sum_{j=1}^{k} \frac{1}{k} p_{i_j}.$$

Eine solche Gleichung kann insbesondere für jedes „freie“ Feld aufgestellt werden, also für jedes Feld, das kein Ausgang (und keine Mauer) ist. Man muss dabei lediglich beachten, dass bei Richtungen, die über ein Ausgangsfeld führen, das erreichte Feld das Ausgangsfeld ist und nicht irgendeines dahinter. Die Wahrscheinlichkeiten der Ausgänge können wir fest auf 1 setzen. Anschließend brauchen wir nur noch eine Möglichkeit, dieses System von linearen Gleichungen zu lösen.

Ein solches Verfahren ist das sog. Gaußsche Eliminationsverfahren, das wir im Folgenden an einem Beispiel von drei Feldern illustrieren. Gegeben seien die Gleichungen

$$\alpha_{11}p_1 + \alpha_{12}p_2 + \alpha_{13}p_3 = b_1 \tag{8}$$
$$\alpha_{21}p_1 + \alpha_{22}p_2 + \alpha_{23}p_3 = b_2 \tag{9}$$
$$\alpha_{31}p_1 + \alpha_{32}p_2 + \alpha_{33}p_3 = b_3, \tag{10}$$

wobei die p_i die gesuchten Wahrscheinlichkeiten sind, die α_{ij} und b_i sind konstante und gegebene Zahlen. Wir bilden nun aus diesem Gleichungssystem ein äquivalentes, aber einfacheres,

indem wir das α_{21}/α_{11}-fache der Gleichung (8) von Gl. (9) und das α_{31}/α_{11}-fache der Gl. (8) von Gl. (10) abziehen. Dadurch erhalten wir

$$\begin{aligned} \alpha_{11}p_1 + \alpha_{12}p_2 + \alpha_{13}p_3 &= b_1 && (11)\\ \alpha'_{22}p_2 + \alpha'_{23}p_3 &= b'_2 && (12)\\ \alpha'_{32}p_2 + \alpha'_{33}p_3 &= b'_3 && (13) \end{aligned}$$

mit einigen anderen Koeffizienten α'_{ij} und b'_i. Anschließend kann das $\alpha'_{32}/\alpha'_{22}$-fache von Gl. (12) von Gl. (13) abgezogen werden. Das Ergebnis ist die Dreieckform

$$\begin{aligned} \alpha_{11}p_1 + \alpha_{12}p_2 + \alpha_{13}p_3 &= b_1 && (14)\\ \alpha'_{22}p_2 + \alpha'_{23}p_3 &= b'_2 && (15)\\ \alpha''_{33}p_3 &= b''_3. && (16) \end{aligned}$$

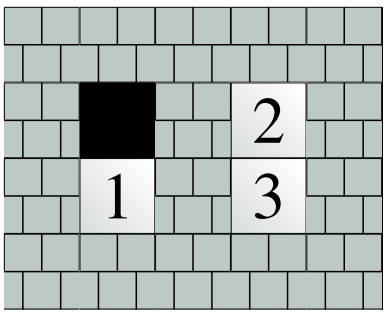
Jetzt sind wir so gut wie fertig: Aus Gl. (16) können wir direkt die Wahrscheinlichkeit p_3 ablesen. Diese können wir in Gl. (15) einsetzen, um p_2 zu bestimmen. Schließlich können p_2 und p_3 in Gl. (14) eingesetzt werden, um auch p_1 zu berechnen, und wir sind fertig.

Natürlich lässt sich das Verfahren auf offensichtliche Weise auf n Gleichungen mit n Unbekannten erweitern, die für n „freie“ Felder benötigt werden. Das Laufzeitverhalten ist jedoch nicht besonders gut. Zunächst muss ein Vielfaches der ersten Zeile zur zweiten addiert werden, was etwa n Operationen benötigt, anschließend zur zweiten, und so weiter, also n^2 Operationen. Dann muss die zweite Zeile alle darunterliegenden manipulieren, was $(n-1)^2$ Operationen benötigt, etc., insgesamt also

$$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

Operationen. Das anschließende Rückwärts-Einsetzen erfordert nochmal etwa n^2 Operationen, insgesamt ist die Laufzeit also durch $\mathcal{O}(n^3)$ beschränkt.

Allerdings birgt das Verfahren ein Problem in sich. Sei dazu folgendes Beispiel betrachtet:



Die Felder 2 und 3 können also keinen Ausgang erreichen, Feld 1 trifft aber mit Sicherheit den Ausgang. Als Gleichungssystem sieht das Beispiel wie folgt aus:

$$\begin{aligned} p_1 &= 1\\ p_2 - p_3 &= 0\\ -p_2 + p_3 &= 0. \end{aligned}$$

Man erhält es, indem man die Gleichungen aufstellt, die Wahrscheinlichkeiten der Ausgangsfelder auf 1 setzt und dann alle Variablen p_i auf die linke Seite bringt. Wendet man obiges Verfahren an, zieht also das (-1)-fache der zweiten Zeile von der dritten ab, erhält man

$$\begin{aligned} p_1 &= 1 \\ p_2 - p_3 &= 0 \\ 0 &= 0. \end{aligned}$$

Es existiert also keine eindeutige Lösung, da lediglich $p_2 = p_3$ festgelegt ist (mathematisch ausgedrückt, ist das Gleichungssystem linear abhängig). Dies stellt man während der Durchführung des Algorithmus gerade dadurch fest, dass man Zeilen erhält, deren Koeffizienten alle verschwinden. Durch unser Wissen über die Aufgabenstellung können wir solche Doppeldeutigkeiten zum Glück auflösen, denn in einem solchen Fall muss die entsprechende Wahrscheinlichkeit verschwinden, da offenbar kein Ausgang erreicht wird. Es reicht also, wenn man in der i-ten Zeile ist und feststellt, dass sie Null ist, den Eintrag $\alpha_{ii} = 1$ zu setzen, was nichts anderes bedeutet als $p_i = 0$.

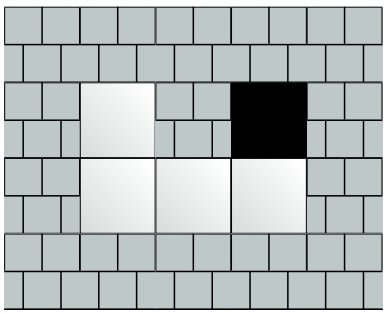
Implementierung

Angenommen, wir haben das Gleichungssystem bereits aufgestellt und die Matrix A enthält die Koeffizienten α_{ij} und der Vektor $\vec{b} = (b_1, \ldots, b_n)^{\mathrm{T}}$ die rechte Seite. Dann lässt sich der Gauß-Algorithmus wie im folgenden Python-Code implementieren.

```
1  # Löst das Gleichungssystem A*p = b nach p.
2  def solveSystem(A, b):
3    # Anzahl der Variablen
4    n = len(b)
5
6    # Lösungsvektor vorbereiten, mit 0 initialisieren
7    p = [0 for i in range(n)]
8
9    # Gehe alle Zeilen durch
10   for i in range(n):
11     # Test, ob aktuelle Zeile 0 ist:
12     isZero = True
13     for j in range(i, n):
14       if A[i][j] != 0
15         isZero = False
16         break
17
18     if b[i] != 0:
19       isZero = False
20
21     # Ist Zeile = 0? Dann setze Wkeit auf 0,
22     # indem aus der Zeile p_i = 0 gemacht wird.
23     if isZero:
```

```
24        A[i][i] = 1.0
25      # Ziehe Zeile von darunter liegenden Zeilen ab
26      for j in range(i+1, n):
27        coefficient = -(A[j][i]/A[i][i])
28        for col in range(i, n):
29          A[j][col] += coefficient*A[i][col]
30        b[j] += coefficient*A[i][col]
31
32    # Fertig! Nun fehlt noch die Röckeinsetzung
33    p[n-1] = b[n-1]/A[n-1][n-1];
34    for i in reversed(range(n-1)):
35      sum = 0
36      for j in range(i, n):
37        sum += p[j]*A[i][j]
38
39      p[i] = (b[i]-sum)/A[i][i]
40
41    return p
```

Nun da wir das Kernstück des Verfahrens implementiert haben, fehlt noch die Vor- und die Nachbereitung. Zunächst müssen wir die Welt einlesen und jedem freien Feld eine Nummer i und eine Wahrscheinlichkeit p_i zuordnen. Dazu wählen wir eine Darstellung der Welt als Matrix von Integer-Werten. Eine Mauer stellen wir als -1 dar, einen Ausgang als -2. Freie Felder erhalten aufsteigende Zahlen von 0 bis $n-1$, wobei n die Gesamtzahl der freien Felder ist. Die folgende Beispielwelt (links) wird also als die rechts stehende Matrix eingelesen:



$$\leftrightarrows \begin{pmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & 0 & -1 & -2 & -1 \\ -1 & 1 & 2 & 3 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{pmatrix}$$

Anschließend müssen wir die Gleichungen aufstellen, also die Einträge der Matrix A und des Vektors b bestimmen. Dazu iterieren wir über alle freien Felder i. Für jedes solches bestimmen wir für jede Himmelsrichtung das Endfeld in der Richtung, gehen also so weit, bis wir auf eine Wand oder einen Ausgang stoßen. Ist es ein Ausgang, erhöhen wir b_i um 1, ist es ein freies Feld j vor einer Mauer, so erniedrigen wir α_{ij} um 1. Anschließend teilen wir alle gesetzten Einträge durch die Anzahl von Himmelsrichtungen, die wir betrachten mussten (bei angrenzenden Mauern verringert sich diese entsprechend) und setzen $\alpha_{ii} = 1$.

Nachdem also die Welt als Integer-Matrix eingelesen und daraus wie oben beschrieben das Gleichungssystem mit Matrix A und Vektor b aufgestellt worden sind (die entsprechenden Code-Funktionen heißen hier *readWorld* und *prepareSystem*), müssen wir nur noch *solveSystem* aufrufen und das Feld ausgeben. Zusammen ergibt sich als Implementierung:

```
1 def main():
```

```
2
3   T, n, width, height = readWorld()
4   A, b = prepareSystem(T, n, width, height)
5   p = solveSystem(A, b)
6
7   printSystem(T, width, height, p)
8
9  def printSystem():
10   s = ""
11   for y in range(height):
12     for x in range(width):
13       t = T[y][x]
14
15       if t == _WALL:
16         s += "#"
17       elif t == _EXIT:
18         s += "E"
19       elif p[t] == 1: # sicher
20         s += "S"
21       else:
22         s += "␣"
23     s += "\n"
24
25   print(s)
```

3.3 Flood-Fill

Das Gauß-Verfahren liefert zwar das korrekte Ergebnis, hat dafür jedoch eine relativ hohe Laufzeit. Es geht allerdings sehr viel schneller, indem man einen modifizierten Flood-Fill-Algorithmus benutzt.

Dazu machen wir zunächst folgende Beobachtung: Man kann die Felder, die kein Ausgang sind, in zwei verschiedene Arten von Feldern unterteilen: Felder, von denen man einen Ausgang erreichen kann (blau) und Felder, von denen man den Ausgang nie erreichen kann (rot). Sei P_{exit} die Wahrscheinlichkeit, einen Ausgang zu erreichen, dann gilt:

Farbe	Wahrscheinlichkeit, einen Ausgang zu erreichen
rot	$P_{\text{exit}} = 0$
blau	$P_{\text{exit}} > 0$

Nun sollen die blauen Felder weiter in zwei verschiedene Arten unterteilt werden: Felder, von denen man rote Felder erreichen kann (gelb) und Felder von denen man rote Felder nie erreichen kann (grün). Sei also P_{rot} die Wahrscheinlichkeit, ein rotes Feld zu erreichen, dann gilt:

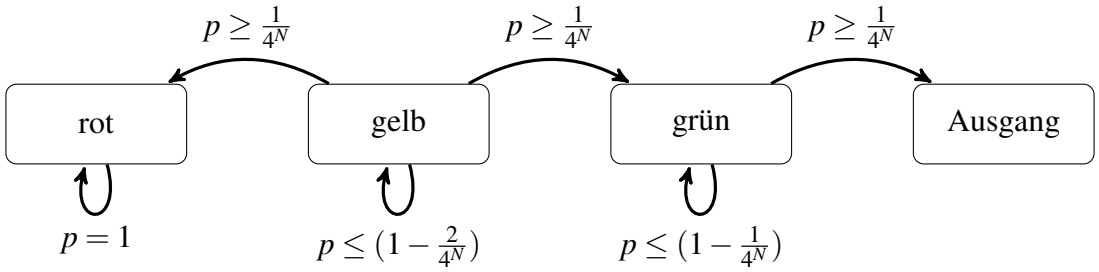
Farbe	Wahrscheinlichkeit, ein rotes Feld zu erreichen
grün	$P_{\text{rot}} = 0$
gelb	$P_{\text{rot}} > 0$

Jetzt fehlt nur noch folgende Beobachtung:

- Ein Yamyam auf einem gelben Feld kann auf gelbe, grüne und rote Felder gelangen.
- Ein Yamyam auf einem roten Feld kann nie wieder auf gelbe oder grüne Felder gelangen.
- Ein Yamyam auf einem grünen Feld kann nie wieder auf gelbe oder rote Felder gelangen.

Man kann sogar noch mehr sagen: Bei Betrachtung von genügend vielen (nämlich maximal $N = n \cdot m$ bei $m \times n$ Feldern) Schritten sind die Wahrscheinlichkeit $p_{\text{gelb}\to\text{rot}}$ von einem gelben Feld auf ein rotes Feld zu kommen, die Wahrscheinlichkeit $p_{\text{gelb}\to\text{grün}}$ von einem gelben Feld auf ein grünes Feld zu kommen und die Wahrscheinlichkeit $p_{\text{grün}\to\text{exit}}$ von einem grünen Feld auf einen Ausgang zu kommen größer als Null. Da es nur maximal N freie Felder gibt, sind sie sogar größer oder gleich $\frac{1}{4^N}$.

Wir können das in dem folgendem Diagramm ausdrücken:



Von diesem Diagramm können wir vier mögliche Endzustände ablesen:

1. Wir landen irgendwann auf einem roten Feld (und bleiben dann immer auf roten Feldern).
2. Wir bleiben unendlich lange auf den gelben Feldern.
3. Wir landen irgendwann auf einem grünen Feld und bleiben dann unendlich lange auf den grünen Feldern.
4. Wir landen irgendwann auf einem Ausgang.

Davon sind jedoch die Möglichkeiten 2 und 3 aber nur mit einer Wahrscheinlichkeit $p = 0$ tatsächlich erreichbar. Dies lässt sich durch eine einfach Abschätzung zeigen:

$$P_{\text{unendlich lange gelb}} = \lim_{t\to\infty} (p_{\text{gelb}\to\text{gelb}})^t \leq \lim_{t\to\infty} \left(1 - \frac{2}{4^N}\right)^t = 0$$

$$P_{\text{unendlich lange grün}} = \lim_{t\to\infty} (p_{\text{grün}\to\text{grün}})^t \leq \lim_{t\to\infty} \left(1 - \frac{1}{4^N}\right)^t = 0$$

Daran können wir sehen, dass es nur zwei Endzustände gibt, die mit möglicherweise nichtverschwindenden Wahrscheinlichkeiten erreicht werden:

- Entweder wir erreichen irgendwann einen Ausgang,
- oder wir landen irgendwann auf einem roten Feld.

Da es keine anderen Möglichkeiten mehr gibt, gilt also:

$$\text{für jedes Feld gilt:} \quad P_{\text{exit}} = 1 - P_{\text{rot}}$$

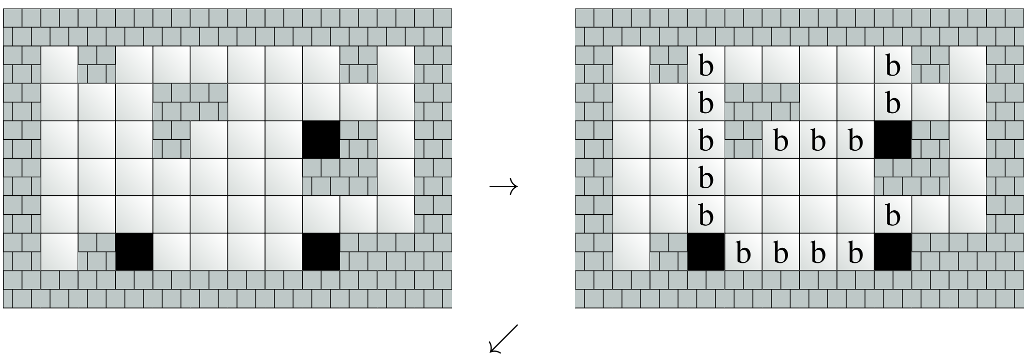
Damit können wir nun ablesen:

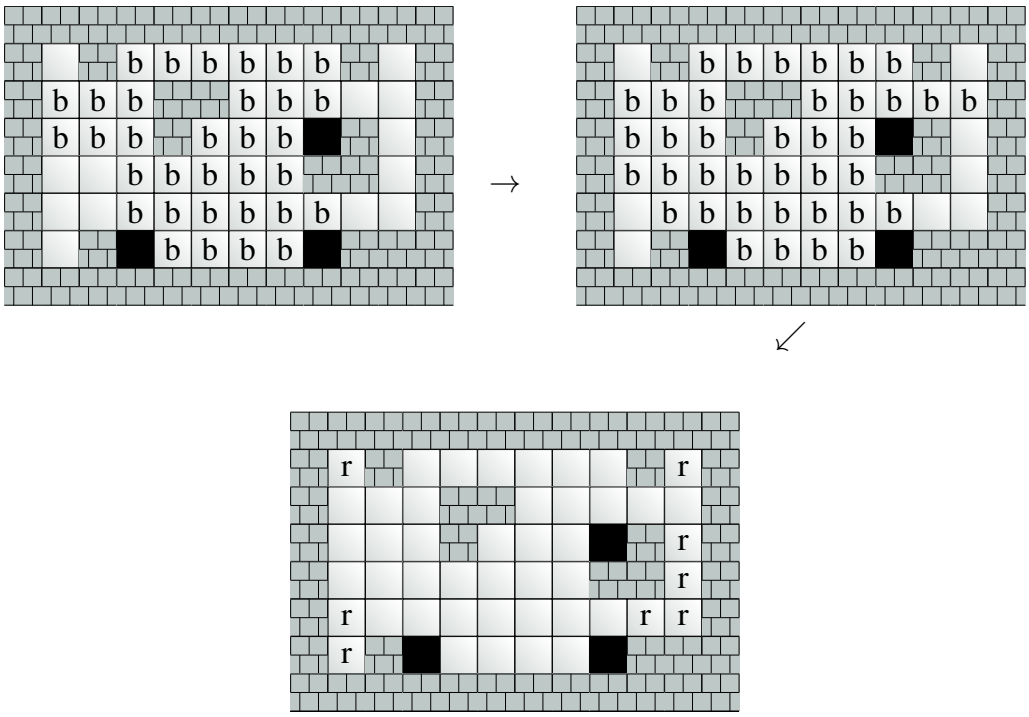
Farbe	Wahrscheinlichkeit, ein rotes Feld zu erreichen	Wahrscheinlichkeit, einen Ausgang zu erreichen
grün	$P_{\text{rot}} = 0$	$P_{\text{exit}} = 1$
gelb	$P_{\text{rot}} > 0$	$P_{\text{exit}} < 1$
rot	$P_{\text{rot}} = 1$	$P_{\text{exit}} = 0$

Damit haben wir gezeigt, dass genau die grünen Felder die Felder sind, von denen aus man mit Wahrscheinlichkeit $P = 1$ einen Ausgang erreicht.

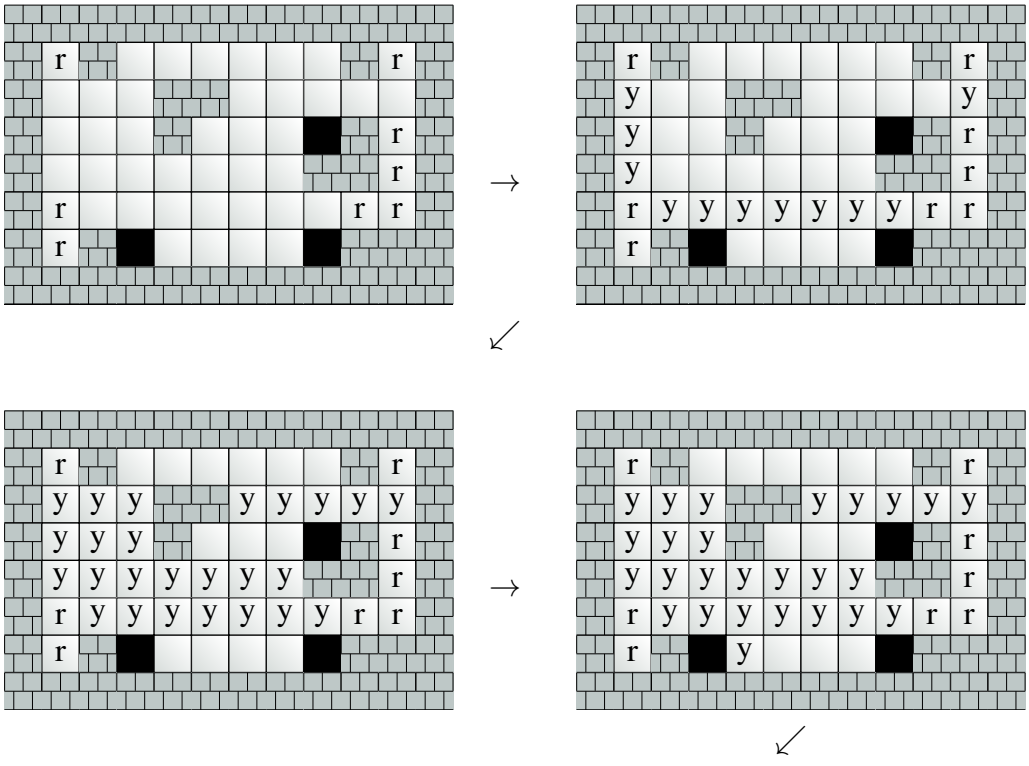
Schließlich müssen wir uns noch überlegen, wie wir das Einfärben vornehmen. Es sollen zunächst alle blauen Felder bestimmt werden, also solche, die einen Ausgang erreichen können. Dazu können wir eine Suche von den Ausgängen aus starten und alle Felder, die den Ausgang direkt erreichen können, blau markieren (das sind die Felder, die sich in die vier Himmelsrichtungen von dem Ausgang aus erstrecken). Anschließend färben wir alle Felder, die ein bereits blau markiertes Feld erreichen können, ebenfalls blau, usw., bis keine neuen Felder mehr markiert werden können. Algorithmisch lässt sich das über eine Warteschlange lösen. In diese tragen wir zunächst alle Felder ein, die direkt einen Ausgang erreichen können. Anschließend entfernen wir das erste Feld aus der Schlange und bestimmen alle anderen Felder, von denen aus es erreichbar ist. Es ist nur dann erreichbar, wenn es an mindestens eine Mauer grenzt und in der entgegen gelegenen Richtung begehbare Felder liegen. Jedes solche Feld, das noch nicht blau markiert ist oder sich in der Warteschlange befindet, fügen wir ans Ende der Warteschlange ein und markieren es blau.

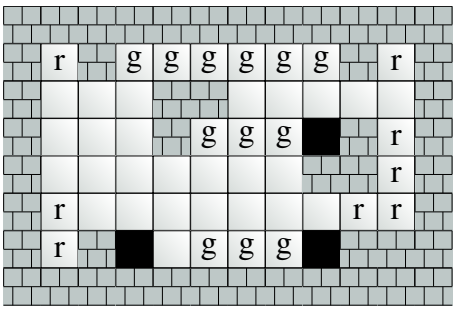
Folgende Grafik veranschaulicht den Vorgang für das Beispiel aus der Aufgabenstellung:





Nach dem Blaufärben können wir alle ungefärbten freien Felder rot markieren, denn von ihnen aus kann man offenbar niemals einen Ausgang erreichen. Schließlich müssen wir die blau gefärbten Felder in gelbe und grüne aufteilen, also solche, die rote Felder erreichen können, und solche, die es nicht können. Zum Gelbfärben können wir genau den gleichen Algorithmus wie zuvor benutzen, nur dass wir diesmal die roten Felder als Ausgänge betrachten. Der Färbevorgang wird in den folgenden Abbildungen verdeutlicht:





Die übrigen blauen Felder werden dann grün gefärbt und der Algorithmus ist komplett. Die nun grün gefärbten Felder sind tatsächlich die sicheren, wie erwartet.

Die Laufzeit des Verfahrens ist bei $m \times n$ Feldern durch $\mathcal{O}(m \cdot n \cdot \max(m,n))$ beschränkt, da jedes Feld höchstens zwei mal gefärbt werden muss und höchstens $\max(m,n)$ Nachbarn hat. Im Vergleich zur oberen Schranke von $\mathcal{O}(m^3 \cdot n^3)$ des Gauß-Verfahrens ist das eine deutliche Verbesserung.

3.4 Iterative Verfahren

Ein weiteres mögliches Verfahren ist ein iteratives Verfahren: Dazu setzt man zunächst die Wahrscheinlichkeiten für die Felder, die man erreichen will auf 100% und alle andere auf 0%. Nun aktualisiert man die Wahrscheinlichkeiten iterativ: In jedem Schritt ergibt sich die aktualisierte Wahrscheinlichkeit P_{neu} eines Feldes durch den Durchschnitt aller Wahrscheinlichkeiten der Felder, die von diesem Feld aus erreicht werden können:

$$P_{\text{neu}} = \frac{1}{4}(P_1 + P_2 + P_3 + P_4)\,.$$

Üblicherweise sind das vier Felder, deren Wahrscheinlichkeiten hier mit P_1 bis P_4 bezeichnet sind, in Randfällen kann es sich aber auch um weniger Felder handeln.

Mathematisch entspricht dies einer iterativen Lösung des Gleichungssystems aus dem ersten Abschnitt.

Dies ergibt in endlich vielen Schritten natürlich nur eine approximative Lösung des Gleichungssystems und damit und approximative Werte für die Wahrscheinlichkeiten. Dadurch kann in endlich vielen Schritten nicht festgestellt werden, ob eine bestimmte Klasse von Feldern sicher erreicht werden kann. Es kann aber festgestellt werden, ob eine bestimmte Klasse von Feldern sicher nicht erreicht werden kann. Dazu müssen aber im Allgemeinen mindestens $m \cdot n$ Iterationen durchgeführt werden.

Damit kann die Menge der sicheren Felder in der gleichen Weise bestimmt werden wie im vorherigen Abschnitt: indem man erst die roten Felder (mit mindestens $m \cdot n$ Iterationen) bestimmt, um dann damit die grünen Felder (ebenfalls mit mindestens $m \cdot n$ Iterationen) zu bestimmen.

3.5 Erweiterungen

Wie üblich ist es möglich, das zu lösende Problem etwas zu verallgemeinern und so interessante Erweiterungen möglich zu machen. Die erste vorgestellte Lösung, die das Gaußsche Eliminationsverfahren nutzt, ist zwar langsamer als die Lösung mit dem Flood-Fill-Algorithmus, gibt als Ergebnis aber nicht nur die sicheren Felder an, sondern sogar die exakte Wahrscheinlichkeit jedes Feldes, einen Ausgang zu erreichen. In der folgenden Liste der Beispiele wird diese jeweils mit angegeben.

Das Problem kann leicht ins Dreidimensionale erweitert werden. Allerdings lassen sich auch die vorgestellten Lösungen relativ leicht so anpassen, dass sie für 3D-Yamyam-Welten funktionieren.

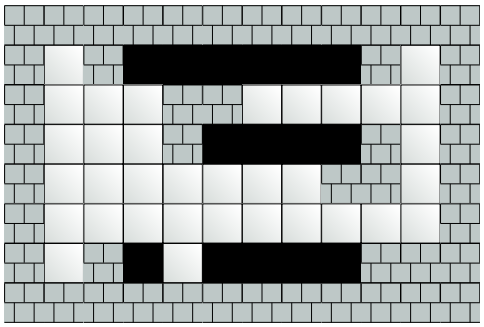
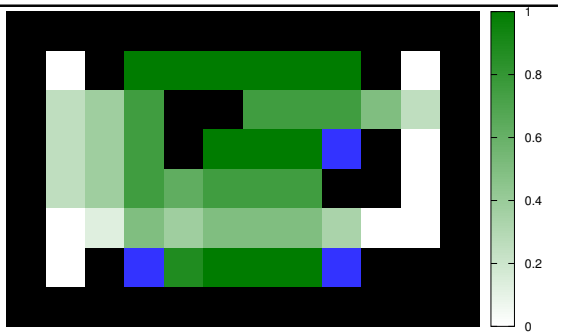
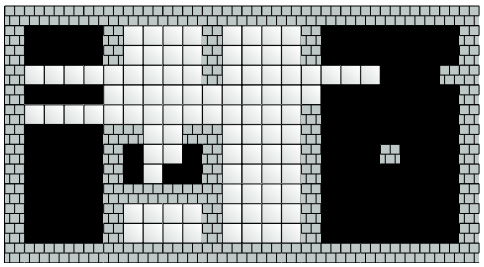
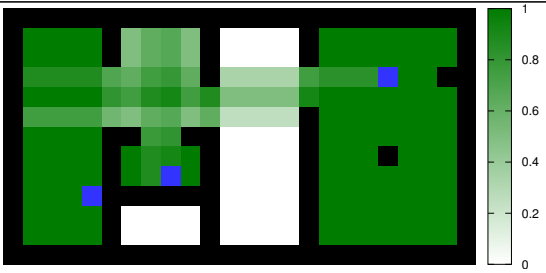
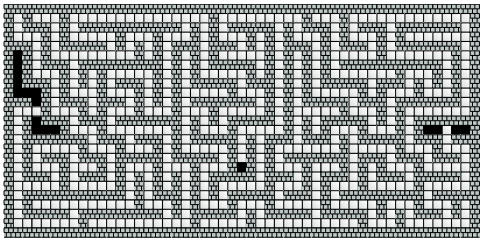
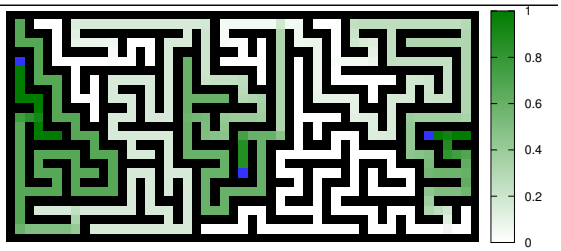
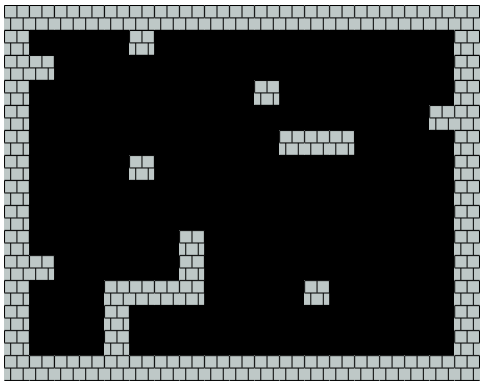
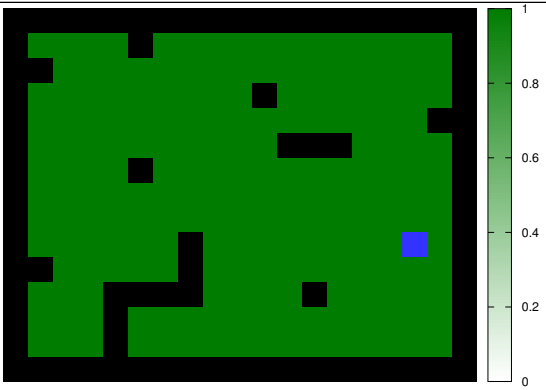
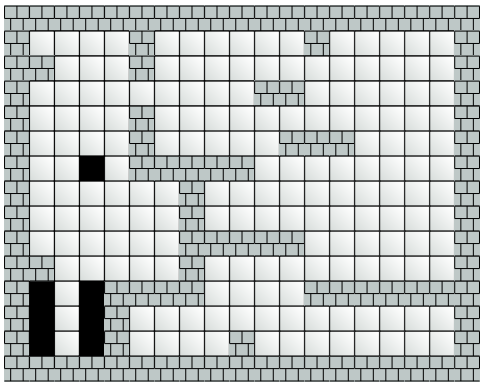
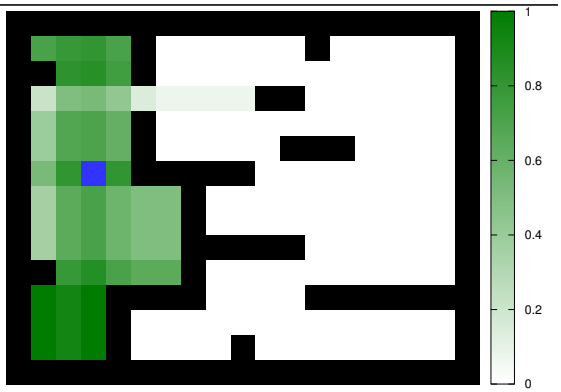
Ein gänzlich anderes Problem erhält man, indem man das nichtdeterministische zufällige Verhalten der Yamyams durch ein deterministisches ersetzt, beispielsweise durch das bekannte „immer links abbiegen“-Verhalten, das einen aus Labyrinthen befreit. In dem Fall könnte man die sicheren Felder beispielsweise durch eine Simulation der Bewegung bestimmen.

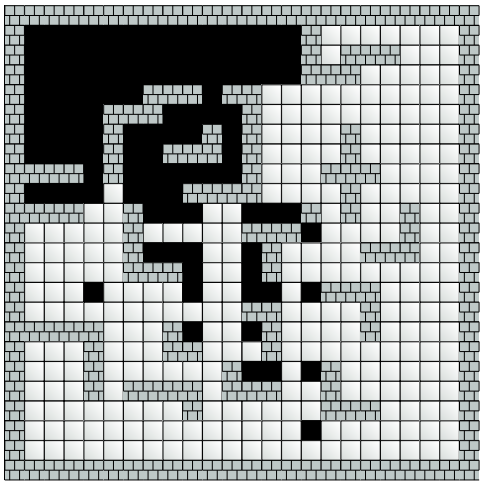
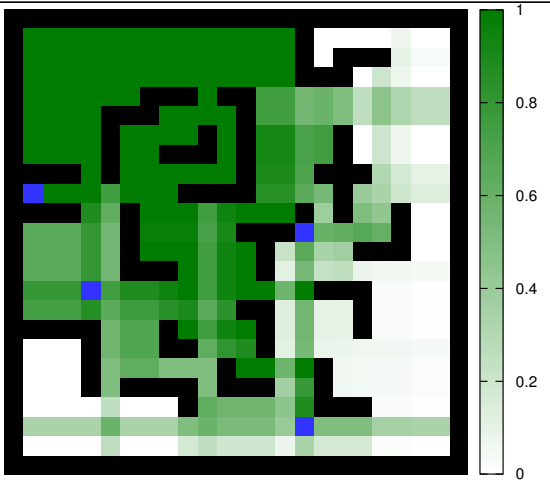
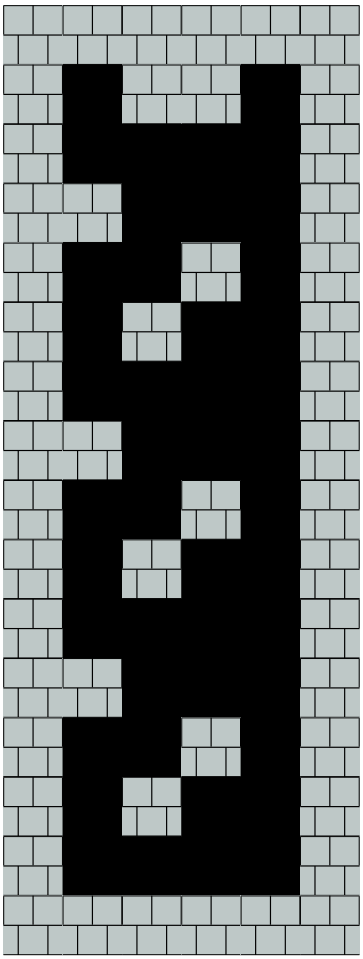
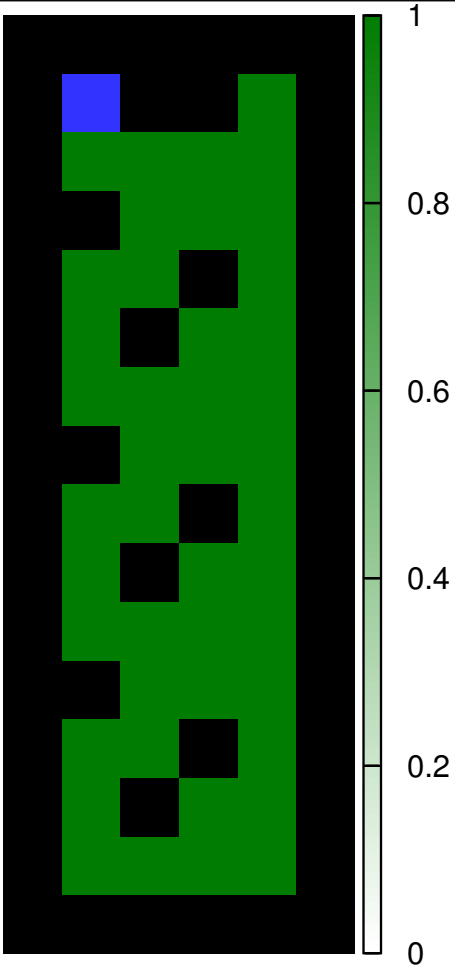
Besonders trickreich wird die Berechnung, wenn mehrere Yamyams im Labyrinth unterwegs sind und gegenseitig als Hindernisse gelten. Dann muss zusätzlich die Wahrscheinlichkeit berücksichtigt werden, dass ein Yamyam im Weg steht und ganz neue Bewegungsmuster werden plötzlich möglich. Dadurch werden einige Felder unsicherer, da man von ihnen aus nun doch unsichere Felder erreichen kann, diese können aber sicherer werden, da ein im Weg stehendes Yamyam einen aus ausweglosen Schleifen (wie im Beispiel des Aufgabenblatts die Felder in der oberen und unteren linken Ecke) befreien kann. Die Berechnung ist aber sehr kompliziert und erfordert eine genauere Beschreibung des Verhaltens der Yamyams.

Eine weitere interessante Ergänzung bestünde darin, das Spiel auf dem Turnierserver kompetitiv zu gestalten: Zwei von einer KI steuerbare Yamyams starten in einem Labyrinth und versuchen als erster einen Ausgang zu erreichen. Dabei sehen sie sich gegenseitig als stoppende Hindernisse an und müssen daher gleichzeitig einen möglichst kurzen Weg zum Ausgang finden und den Gegner durch geschicktes In-den-Weg-stellen vom gleichen Ziel abhalten.

3.6 Beispiele

Die folgenden Bilder zeigen die sicheren Felder der zur Verfügung gestellten Beispiele des BwInf in grün (auf der linken Seite). Außerdem zeigen sie jeweils die Wahrscheinlichkeiten der Felder für ein Yamyam den Ausgang zu erreichen (auf der rechten Seite). Dabei stehen dunkler gefärbtere Felder für Felder mit höherer Wahrscheinlichkeit einen Ausgang zu erreichen.

Nr.	Gelöste Welt	Wahrscheinlichkeitskarte
0		
1		
2		
3		
4		

Nr.	Gelöste Welt	Wahrscheinlichkeitskarte
5		
6		

3.7 Bewertungskriterien

- Die Berechnung der sicheren Felder soll korrekt sein. Abgesehen von Programmierfehlern gibt es wohl einen möglichen Verständnisfehler: Man könnte denken, dass alle Felder, die einen Ausgang erreichen können, auch sichere Felder seien; das ist aber falsch.

Ein weiteres Problem könnte bei isolierten Bereichen wie dem unten links im zweiten Beispiel (yamyams1.txt) auftreten.

- Ein Verfahren in der Art der Flood-Fill-Methode sollte „rückwärts“, also von den Ausgängen her arbeiten. Alles andere ist komplizierter (je nach Vorgehensweise sogar deutlich komplizierter) und auch weniger effizient.
- Werden die Wahrscheinlichkeiten iterativ bestimmt, müssen erst die *roten* Felder bestimmt werden. Dafür dürfen im Allgemeinen nicht weniger als $n \cdot m$ Iterationen (bei $m \times n$ Feldern) durchgeführt werden. Dann können (ebenfalls mit nicht weniger als $m \cdot n$ Iterationen) die sicheren Felder bestimmt werden.
- Das Verfahren sollte auch nicht aus anderen Gründen inkorrekt oder unnötig ineffizient sein.
- Das Laufzeitverhalten der verwendeten Methode soll nicht zu schlecht sein, sofern es nicht begründet ist. Gut ist eine Lösung in $\mathcal{O}(m \cdot n \cdot \max(m,n))$. $\mathcal{O}((mn)^2)$ ist schlecht und $\mathcal{O}((mn)^3)$ sehr schlecht, es sei denn, dass zusätzlicher Erkenntnisgewinn den zusätzlichen Bedarf rechtfertigt (wie z. B. bei der Berechnung der exakten Wahrscheinlichkeiten durch das Gauß-Verfahren).
- Die Laufzeit von Verfahren in der Art des oben beschriebenen Flood-Fill-Ansatzes kann leicht als linear bezeichnet werden. Für Flood-Fill-Lösungen zu dieser Aufgabe ist das aber in der Regel nicht korrekt.
- Das verwendete Verfahren soll ausführlich beschrieben und seine Korrektheit begründet werden. Dabei sollen zumindest andeutungsweise wahrscheinlichkeitstheoretische Überlegungen angestellt worden sein.
- Die gefundenen Lösungen aller sieben Beispiele sollen angegeben werden, möglichst auch in der Dokumentation. Eigene Beispiele sind lobenswert, aber nicht notwendig.
- Für mindestens ein Beispiel sollten einzelne Schritte auf dem Weg zur Bestimmung der sicheren Felder dargestellt sein.
- Die Ausgabe des Ergebnisses soll übersichtlich und nachvollziehbar sein. Die Ausgabe einer Liste von Koordinaten der sicheren Felder ist nicht ausreichend, eine Ausgabe der Welt ähnlich zur Eingabe in Textform schon.
- Die Beschreibung der Welt muss korrekt eingelesen werden. Im Vergleich zu der ähnlichen Kassiopeia-Aufgabe der ersten Runde müssen hier die Dimensionen des Feldes selbstständig herausgefunden werden. Eine besondere Fehlerbehandlung ist aber nicht notwendig. Es darf insbesondere davon ausgegangen werden, dass alle Zeilen die gleiche Länge haben und das Feld von Mauern umgeben ist.

Perlen der Informatik – aus den Einsendungen

Teilweise mit Kommentaren von der Bewertung

Allgemeines

Leider ist gar nicht mal so wenig an diesem Projekt bzw. der Umsetzung zu kritisieren.

Die Lösung ist meistens suboptimal, aber die Komplexität ist dafür eindeutig reduziert.

Leider ist die Technologie noch nicht da, Videos auf Papiere auszudrücken.

Bei einer falschen Eingabe kommt es bei meinem Programm zu falschen Ausgaben oder das Programm hängt sich auf. Das ist so gewollt um die Leute, die mein Programm nicht richtig benutzen zu bestrafen.

Es kann passieren, dass das Programm sehr lange braucht. Versuchen Sie es noch einmal.

Worte des Wettbewerbs:
Algortihmus *Wow, die Variante hatten wir noch nicht.*
Max OS X, Millipixel, Sachgasse, Varible
kompeilt *Wenn das Programm nicht kom-, hat der Programmierer es ver-.*

Aufgabe 1: Geburtstagskuchen

Die Empirie legt nahe, dass man zwei X-Chromosomen braucht, um Herzen zeichnen zu können.

Das einzige objektive „gleichmäßig“ wäre damit der absolute Zufall.

[. . .], wobei bei abnehmender Kerzenzahl die Laufzeit zunimmt.

Am Ende ist noch zu überlegen, ob auf dem Tortenrand eine Kerze weniger verteilt werden soll, da in der Mitte des Geburtstagskuchens eine Lebenskerze stehen muss.

Aufgabe 2: Missglückte Drohnenlieferung

Ein Paket kann nicht gleichzeitig in mehrere Richtungen geworfen werden (Schrödingers Paket?), . . .

Dieses System sollte daher Amacity retten können.

Tipp: Ich habe die Anzeige noch nicht so performant umgesetzt, deshalb empfehle ich für sehr große Felder die Anzeige zu deaktivieren, damit die Erstellung des Plans schneller geht und die Menschen eher wieder vom Dach können.

Dem A*-Algorithmus nacheifernd soll es ein sogenannter „gieriger“ Algorithmus werden. *Wer dem (A-)Stern wirklich folgen will, sollte die bisherigen Kosten nicht ignorieren.*

Das Ziel von Teil 1 ist es also, die Summe aller Wichtigkeiten zu erniedrigen.

Eine mögliche Erweiterung für das Szenario wäre, dass es in der Mitte der Karte ein schwarzes Loch gibt, welches alle Häuser und Pakete in sich hinein saugt. Dadurch wäre die Materie der Pakete und der Häuser in einer Singularität, also auch beieinander. Wenn nun also jedes Paket bei jedem Haus ist, wäre das Szenario gelöst.

Aufgabe 3: Torkelnde Yamyams

Todeskorridor

Im Folgenden soll durch Straßen die Bewegung der Yamyams verdeutlicht werden, [...] wobei eine Straße nur maximal eine horizontale und eine vertikale Straße haben kann.

Die roten Kreise wurden liebevoll mit MS Paint ergänzt. ... *und mit zittrigen Fingern! ;-)*

Mithilfe der Methode 'wololo' wird ein Feld zu einer anderen Gruppe konvertiert.

Die Laufzeit des Algorithmus ist in jedem Fall konstant [...] plus/minus ein bisschen was für Zyklenerkennung. *Da der Algorithmus exponentielle Laufzeit hat, stimmt leider nur das „plus“.*

Ist am Ende kein unrasierter Knoten vorhanden, so ist die Stelle im Labyrinth sicher.

Mein System zur Bestimmung der sicheren Felder baut grundsätzlich auf den Gedanken einer allgemeinen Baumstruktur auf. *Wie bitte kann man wissen, was eine allgemeine Baumstruktur so denkt?*