Lösungshinweise und Bewertungskriterien

Bundeswettbewerb Informatik

Allgemeines

Grundsätzliches

Es ist immer wieder bewundernswert, wie viele Ideen, wie viel Wissen, Fleiß und Durchhaltevermögen in den Einsendungen zur 2. Runde eines Bundeswettbewerbs Informatik stecken. Um aber die Allerbesten für die Endrunde zu bestimmen, müssen wir die Arbeiten kritisch begutachten und hohe Anforderungen stellen. Deswegen sind Punktabzüge die Regel und Bewertungen mit Pluspunkten über die Erwartungen hinaus die Ausnahme.

Spannende bzw. schwierige Erweiterungen der Aufgabenstellung sind immer einige Extrapunkte wert, wenn sie auch praktisch realisiert wurden. Intensive theoretische Überlegungen der Informatik wie z. B. ein korrekter Beweis zur Komplexität des Problems werden ebenfalls mit zusätzlichen Punkten belohnt. Weitere Ideen ohne Implementierung und geringe Verbesserungen der bereits implementierten Lösung einer Aufgabe gelten allerdings nicht als geeignete Erweiterungen.

Wie auch immer Ihre Einsendung bewertet wurde, lassen Sie sich nicht entmutigen! Allein durch die Arbeit an den Aufgaben und ihren Lösungen hat jede Teilnehmerin und jeder Teilnehmer einiges gelernt; diesen Effekt sollten Sie nicht unterschätzen. Selbst wenn Sie zum Beispiel aus Zeitmangel nur die Lösung zu einer Aufgabe einreichten, so erhielten Sie eine Bewertung Ihrer Einsendung, die Ihnen bei der Anfertigung künftiger Lösungen hilfreich sein kann.

Bevor Sie sich in die Lösungshinweise vertiefen, lesen Sie bitte kurz die folgenden Anmerkungen zu den Einsendungen und 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 2. Runde geht die Bewertung von zwanzig Punkten aus (es gibt nur ganze Punkte, keine halben). Im Vergleich zur 1. Runde gibt es deutlich mehr Bewertungskriterien, bei denen Punkte abgezogen oder auch hinzuaddiert werden konnten.

Terminlage

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 Mathematikwettbewerbs, dem wir keine Konkurrenz machen wollen. Aber: die Bearbeitungszeit für die zweite BwInf-Runde beträgt etwa vier Monate. Frühzeitig mit der Bearbeitung der Aufgaben zu beginnen ist der beste Weg, zeitliche Engpässe am Ende der Bearbeitungszeit gerade mit der wichtigen, ausführlichen Dokumentation der Aufgabenlösungen zu vermeiden.

Einige Aufgaben in der zweiten Runde sind oft deutlich schwerer zu lösen, als sie auf den ersten Blick erscheinen mögen. Erst in der konkreten Umsetzung einer Lösungsidee stößt man manchmal auf weitere Besonderheiten bzw. noch zu lösende Schwierigkeiten, was dann zusätzlicher Zeit bedarf. Daher ist es sinnvoll, beide einzureichende Aufgaben nicht nacheinander, sondern relativ gleichzeitig zu bearbeiten, um nicht vom zeitlichen Aufwand der jeweiligen Aufgabe unangenehm kurz vor Ablauf der Bearbeitungszeit überrascht zu werden und keine vollständige Lösung mehr zu schaffen.

Dokumentation

Es ist sehr gut nachvollziehbar, dass Sie Ihre Energie bevorzugt in die Lösung der Aufgaben, die Entwicklung Ihrer Ideen und deren Umsetzung in Software fließen lassen. Doch ohne eine verständliche Beschreibung der Lösungsideen und ihrer jeweiligen Umsetzung, eine übersichtliche Dokumentation der wichtigsten Komponenten Ihrer Programme, eine geeignete Kommentierung der Quellcodes und eine ausreichende Zahl sinnvoller Beispiele (die die verschiedenen, bei der Lösung des Problems zu berücksichtigenden Fälle abdecken) ist eine Einsendung nur wenig wert.

Bewerterinnen und Bewerter können die Qualität Ihrer Aufgabenlösungen nur anhand dieser Informationen vernünftig einschätzen. Mängel in der Dokumentation der Einsendung können nur selten durch das konkrete Überprüfen der Ergebnisse der Programme durch die Bewerterin oder den Bewerter etwas ausgeglichen werden – wenn diese denn überhaupt ausgeführt werden können: Hier gibt es gelegentlich Probleme, die meist vermieden werden könnten, wenn Lösungsprogramme vor der Einsendung nicht nur auf dem eigenen, sondern auch einmal auf einem fremden Rechner hinsichtlich Lauffähigkeit 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 auch keine saubere Umsetzung hin, in welcher Programmiersprache auch immer. Einige unterhaltsame Formulierungsperlen der Informatik sind im Anhang wiedergegeben.

Bewertungskriterien

Bei den im Folgenden beschriebenen Lösungsideen handelt es sich nicht um perfekte Musterlösungen, sondern um sinnvolle Lösungsvorschläge, die nicht die einzigen Lösungswege darstellen, die wir gelten ließen. Wir akzeptieren vielmehr in der Regel alle Ansätze, auch ungewöhnliche, kreative Bearbeitungen, die die gestellte Aufgabe vernünftig lösen und entsprechend dokumentiert sind. Einige der Fußnoten in den folgenden Lösungsvorschlägen verweisen auf weiterführende Fachliteratur für wissenschaftlich besonders Interessierte; ihre Referenzierung und ihr tieferes Verständnis wurden natürlich nicht von den Teilnehmenden erwartet.

Unabhängig vom gewählten Lösungsweg gibt es einige Dinge, die auf jeden Fall in der Dokumentation erwartet wurden oder sogar Pluspunkte gaben. Zu jeder Aufgabe wird deshalb in einem eigenen Abschnitt jeweils am Ende des Lösungsvorschlags erläutert, worauf unter

anderem bei der Bewertung dieser Aufgabe besonders geachtet wurde. Außerdem gibt es aufgabenunabhängig einige Anforderungen an die Dokumentation (klare Beschreibung der Lösungsidee, genügend aussagekräftige Beispiele und wesentliche Auszüge aus dem Quellcode) einschließlich einer theoretischen Analyse (geeignete Laufzeitüberlegungen bzw. eine Diskussion der Komplexität des Problems) sowie an den Quellcode der implementierten Software (mit übersichtlicher Programmstruktur und verständlicher Kommentierung) und an das lauffähige Programm (ohne Implementierungsfehler). Wünschenswert sind unter anderem auch Hinweise auf die Grenzen des angewandten Verfahrens sowie sinnvolle Begründungen z. B. für Heuristiken, vorgenommene Vereinfachungen und Näherungen. Geeignete Abbildungen und eigene zusätzliche Beispieldaten können die Erläuterungen in der Dokumentation gut unterstützten. Die erhaltenen Rechenergebnisse für die Beispieldaten (ggf. mit Angaben zur Rechenzeit) sollten leicht nachvollziehbar dargestellt sein (z. B. durch die Ausgabe von Zwischenschritten oder geeigneten Visualisierungen). Eine Untersuchung der Skalierbarkeit des eingesetzten Algorithmus hinsichtlich des Umfangs der Eingabedaten ist oft ebenfalls nützlich.

Danksagung

Alle Aufgaben wurden vom Aufgabenausschuss des Bundeswettbewerbs Informatik entwickelt: Peter Rossmanith, *RWTH Aachen University* (Vorsitzender); Hanno Baehr, *RWE Supply & Trading GmbH, Essen*; Jens Gallenbacher, *TU Darmstadt, JGU Mainz*; Rainer Gemulla, *Universität Mannheim*; Torben Hagerup, *Universität Augsburg*; Christof Hanke, *Berufliches Gymnasium für Informatik, FLB Herford*; Thomas Kesselheim, *Universität Bonn*; Arno Pasternak, *Fritz-Steinhoff-Gesamtschule Hagen, TU Dortmund*; Holger Schlingloff, *Fraunhofer FOKUS, HU Berlin*; Melanie Schmidt, *Universität Bonn*; als Gäste im Ausschuss: Mario Albrecht und Wolfgang Pohl, *BWINF, Bonn*.

An der Erstellung der im Folgenden skizzierten Lösungsideen wirkten neben dem Aufgabenausschuss vor allem folgende Personen mit: Maximilian Azendorf (Aufgabe 1), Niccolò Rigi-Luperti (Aufgabe 2) und Dominik Meier (Aufgabe 3). Allen Beteiligten sei für Ihre Mitarbeit hiermit ganz herzlich gedankt.

Aufgabe 1: Lisa rennt

1.1 Lösungsidee

1.1.1 Lisas optimaler Pfad und der Sichtbarkeitsgraph

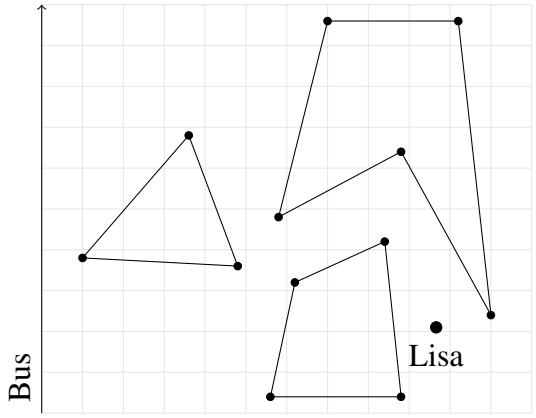


Abbildung 1.1: Eine beispielhafte Situation...

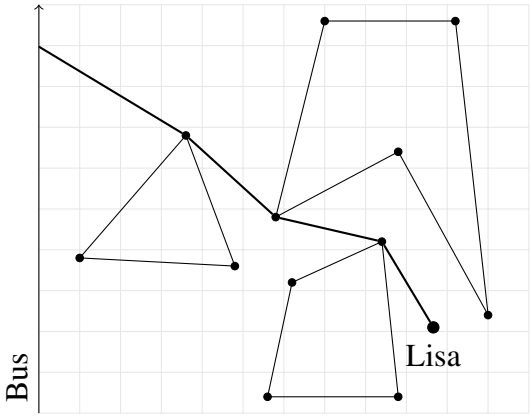


Abbildung 1.2: ...und die Lösung.

In dieser Aufgabe ist der Weg bzw. Pfad zu finden, der es Lisa erlaubt, so spät wie möglich loszulaufen und trotzdem noch den Bus zu erreichen.

Was wir uns zuerst ansehen müssen, ist, wie die optimalen Lösungen für dieses Problem aussehen. In Abbildung 1.2 sehen wir solch eine Lösung. Was uns zuerst auffällt, ist, dass die Lösung nur gerade Streckenabschnitte enthält, die an den Ecken der Polygone entlanglaufen. Der letzte Streckenabschnitt jedoch trifft stets mit einem bestimmten Winkel auf die Fahrtstrecke des Busses.

Dass die optimale Lösung tatsächlich immer aus geraden Strecken besteht und diese auch immer an den Ecken der Hindernisse entlanglaufen, kann man sich relativ einfach überlegen: Wenn eine Lösung eine Stelle enthält, die nicht gerade verläuft, kann man immer eine Abkürzung finden, indem wir einen Kreis um diese Stelle legen – ohne dass sich dieser mit den Hindernissen überschneidet – und den Pfad innerhalb des Kreises durch die Kreissehne, welche durch die Schnittpunkte des Kreises mit dem Pfad verläuft, ersetzen. Diese Konstruktion ist in Abbildung 1.3 exemplarisch illustriert.

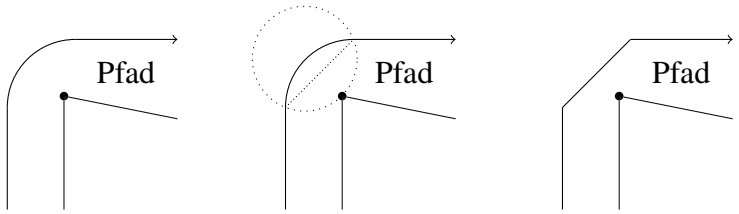


Abbildung 1.3: Exemplarische Konstruktion einer Abkürzung.

Danach haben wir natürlich immer noch nicht den kürzesten Pfad gefunden; man müsste diese Konstruktion oft anwenden, bis der optimale Pfad konstruiert wäre. Was uns diese Konstruktion aber deutlich macht: Solange wir sie durchführen können, kann unser Pfad nicht der kürzeste sein; und wir können sie nur dann nicht ausführen, wenn der Pfad dort, wo er kein Hindernis berührt, gerade verläuft.

Wir wissen jetzt also, dass der optimale Pfad immer durch die Ecken der Hindernisse verläuft. Um diesen Pfad ausrechnen zu können, benötigen wir den sogenannten Sichtbarkeitsgraph der Hindernisse, wie er aus der geometrischen Algorithmik bekannt ist.[1] Ein Sichtbarkeitsgraph ist ein Graph mit den Ecken der Hindernisse als Knotenmenge, bei dem zwei Knoten (also zwei Ecken) genau dann mit einer Kante verbunden sind, wenn diese beiden Ecken sich „sehen" können, also keine Hindernisse die direkte Verbindungslinie überschneiden. In Abbildung 1.4 sehen wir den Sichtbarkeitsgraphen des obigen Beispiels aus Abbildung 1.1.

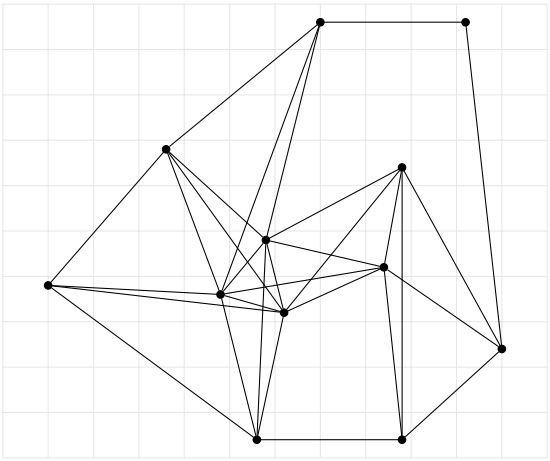


Abbildung 1.4: Der Sichtbarkeitsgraph des obigen Beispiels.

Allerdings müssen wir noch einige Modifikationen am Sichtbarkeitsgraphen vornehmen. Zum einen müssen wir einen Startknoten für Lisa einfügen, welcher eine Kante mit allen von ihr aus sichtbaren Ecken besitzt. Zum anderen müssen wir uns überlegen, wie wir das Ziel, also die Straße, auf der der Bus fährt, modellieren. Dazu fügen wir noch einen weiteren Knoten als Endknoten des Pfades ein, der die Straße repräsentiert. Im folgenden Abschnitt werden wir noch klären, welche Knoten wir mit diesem Endknoten verbinden und welche Gewichte diese Kanten haben.

1.1.2 Endkanten zum Endknoten der Straße und ihre Gewichte

Wir wissen, dass der optimale Weg immer aus geraden Segmenten besteht. Das letzte Segment verläuft damit immer direkt in einem bestimmten Winkel auf die Straße des Busses zu. Wir wollen nun den optimalen Winkel bestimmen, damit wir auf der „Zielgeraden" so viel Zeit wie möglich sparen.

Im Folgenden lösen wir das allgemeine Problem für beliebige Geschwindigkeiten von Bus und Lisa, auch wenn die konkreten Vorgaben beider Geschwindigkeiten in der Aufgabenstellung das Problem deutlich vereinfachen und sogar mit elementargeometrischen Mitteln lösbar machen.

Sei v_B die Geschwindigkeit des Busses und v_L die Geschwindigkeit von Lisa. Was wir zuerst erkennen, ist, dass der Bus auf jeden Fall schneller als Lisa sein muss, da Lisa sonst den Bus aus beliebiger Entfernung in endlicher Zeit einholen könnte. Im Folgenden gilt deshalb $v_L < v_B$. Wir betrachten jetzt die Situation, in der Lisa auf direktem Weg zur Straße laufen kann und dabei den orthogonalen Abstand d zur Straße hat.

Wir wollen nun den „optimalen" Winkel α finden, sodass Lisa so spät wie möglich losgehen kann und trotzdem noch den Bus erreicht. Wenn man das Problem anders herum betrachtet, wird es einfacher: Bei welchem Winkel α muss ein Bus am weitesten entfernt (am weitesten oben in der Skizze) starten, so dass Lisa ihn gerade nicht mehr erreichen kann? Wenn ein Bus nämlich bei diesem Winkel weiter oben losfahren muss, um Lisa zu entkommen, hat Lisa mehr Zeit, den Bus zu bekommen, der an einem festen Punkt (an der Haltestelle) losfährt, da der Bus ja eine konstante Geschwindigkeit hat. Wir wollen also den Fahrtweg des Busses s_B in Abhängigkeit

[1] Visibility Graph. Wikipedia, `https://en.wikipedia.org/wiki/Visibility_graph`

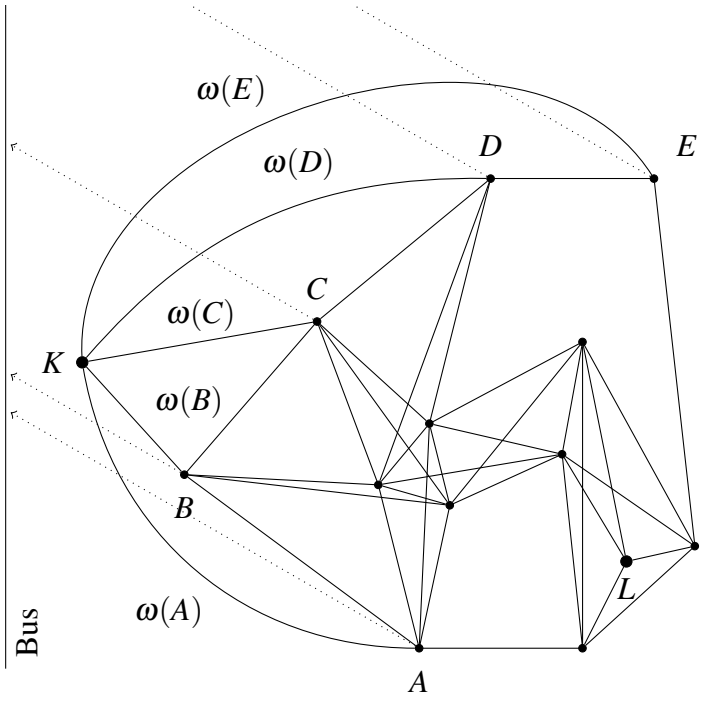


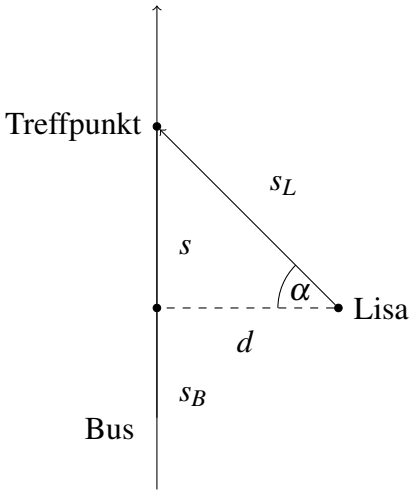
Abbildung 1.5: Der Sichtbarkeitsgraph mit Lisas Startknoten L und den möglichen Endkanten zum Endknoten K. Die gepunkteten Pfeile sollen die freien Blickrichtungen auf die Straße verdeutlichen. Die nicht angegebenen Gewichte der Kanten UV entsprechen der regulären Wegzeit $d(U,V)/v_L$ von Lisa.

von α minimieren; um α berechnen zu können, müssen wir s kennen. Dazu stellen wir uns zuerst zwei Gleichungen auf:

$$s_L = \sqrt{d^2 + s^2} \tag{1.1}$$

$$\frac{s_L}{v_L} = \frac{s_B + s}{v_B}\,. \tag{1.2}$$

Hierzu sehen wir uns die Situation einmal grafisch an:



Gleichung (1.1) beschreibt s_L in Abhängigkeit von d und s, und Gleichung (1.2) beschreibt die Tatsache, dass der Bus und Lisa für ihre jeweiligen Fahrt- bzw. Laufstrecken die selbe Zeit

benötigen. Die beiden Gleichungen ergeben dann folgende Berechnungsformel für s_B:

$$\frac{s_B + s}{v_B} = \frac{\sqrt{d^2 + s^2}}{v_L} \tag{1.3}$$

$$\Leftrightarrow \quad v_L(s_B + s) = v_B\sqrt{d^2 + s^2} \tag{1.4}$$

$$\Leftrightarrow \quad s_B = \frac{v_b}{v_L}\sqrt{d^2 + s^2} - s\,. \tag{1.5}$$

Nun leiten wir diese Formel nach s ein- und zweimal ab und erhalten:

$$s'_B(s) = \frac{\mathrm{d}s_B}{\mathrm{d}s} = \frac{v_B s}{v_L\sqrt{d^2 + s^2}} - 1 \tag{1.6}$$

$$s''_B(s) = \frac{\mathrm{d}^2 s_B}{\mathrm{d}s} = \frac{v_B d^2}{v_L(s^2 + d^2)^{\frac{3}{2}}}\,. \tag{1.7}$$

Um nun das Minimum von s_B zu finden, müssen wir die Gleichung $s'_B(s) = 0$ lösen (und überprüfen, dass bei diesem s auch $s''_B(s) > 0$ gilt):

$$s'_B(s) = 0 \tag{1.8}$$

$$\Leftrightarrow \quad 1 = \frac{v_B s}{v_L\sqrt{d^2 + s^2}} \tag{1.9}$$

$$s = \frac{v_L d}{\sqrt{v_B^2 - v_L^2}} \tag{1.10}$$

Setzt man dieses Ergebnis in $s''_B(s)$ (Gleichung (1.7)) ein, erhält man

$$s''_B(s) = \frac{d^2 v_B}{v_L\left(\frac{d^2 v_B^2}{v_B^2 - v_L^2}\right)^{\frac{3}{2}}} > 0\,, \tag{1.11}$$

womit wir das Minimum berechnet haben. Den Winkel α können wir somit durch

$$\alpha = \tan^{-1}\left(\frac{s}{d}\right) = \tan^{-1}\left(\frac{v_L}{\sqrt{v_B^2 - v_L^2}}\right) = \sin^{-1}\left(\frac{v_\mathrm{L}}{v_\mathrm{B}}\right) \tag{1.12}$$

berechnen, wobei die letzte Umformung für Winkel zwischen $-90°$ und $90°$ gültig ist, wie man sich an einem rechtwinkligen Dreieck mit der Hypothenuse v_B und der Gegenkathete v_L deutlich machen kann. Damit ergibt sich für die vorgegebenen Geschwindigkeiten in der Aufgabenstellung ein Winkel von

$$\alpha = 30° \tag{1.13}$$

Alternativ kann man auch direkt auf das physikalische Brechungsgesetz[2] zurückgreifen, von

[2] Snelliussches Brechungsgesetz. Wikipedia, `https://de.wikipedia.org/wiki/Snelliussches_Brechungsgesetz`

dem der hier auftretende Fall eines Spezialfall darstellt, bei dem ein Brechungswinkel 90° beträgt (die Strecke des Buses nachdem Lisa zugestiegen ist).

Der Zusammenhang aus der Minimierung der Wegzeit und dem Brechungsgesetz ergibt sich durch das Fermatsche Prinzip.[3] Im Ergebnis hat man die inversen Geschwindigkeiten als Brechungsindizes, und das Brechungsgesetz lautet:

$$\frac{1}{v_\mathrm{B}}\sin(90^\circ) = \frac{1}{v_\mathrm{L}}\sin(\alpha) \qquad \Rightarrow \qquad \alpha = \sin^{-1}\left(\frac{v_\mathrm{L}}{v_\mathrm{B}}\right) \tag{1.14}$$

Wenn wir jetzt die zeitlichen „Kosten“ (also die Zeit, die Lisa für den Laufweg zu Straße benötigt minus der Zeiteinsparung, die durch das „Abfangen“ des Busses nach der Haltestelle entsteht) für diesen Weg ausrechnen wollen, müssen wir zum einen die Zeit berücksichtigen, die Lisa für die Strecke vom jeweiligen Startpunkt aus bis zur Straße benötigt, und zum anderen die Zeit, die sie durch ihre Ankunft an dem bestimmten Punkt der Straße durch den Bus gewinnt, denn der Bus benötigt ja ebenso eine gewisse Zeit, um den Weg von der Haltestelle zum Treffpunkt mit Lisa zurückzulegen. Für einen Punkt P müssen wir uns also eine Kostenfunktion $\omega(P) = t_L(P) - t_B(P)$ überlegen, wobei $t_L(P)$ die Zeit, die Lisa für den Weg zur Straße benötigt und $t_B(P)$ die Zeit, die der Bus von der Haltestelle zum Treffpunkt mit Lisa benötigt, ist.

Die Zeit $t_L(P)$, die Lisa vom Punkt P mit den Koordinaten x und y für den optimalen Weg benötigt, lässt sich nach Gleichung 1.1 und dem Fakt, dass die Entfernung von Lisa zur Straße genau x beträgt (also $d = x$), mit

$$t_L(x) = \frac{\sqrt{d^2+s^2}}{v_L} = \frac{\sqrt{x^2+s^2}}{v_L} \tag{1.15}$$

$$\tag{1.16}$$

berechnen. Setzen wir jetzt noch die Formel für s aus Gleichung 1.10 ein, erhalten wir

$$t_L(x) = \frac{\sqrt{x^2 + \frac{v_L^2 x^2}{v_B^2+v_L^2}}}{v_L} \tag{1.17}$$

$$\Leftrightarrow \qquad t_L(x) = x \cdot \frac{v_B}{v_L\sqrt{v_B^2 - v_L^2}}\,. \tag{1.18}$$

Nun brauchen wir noch eine Formel für $t_B(P)$, also die Zeit, die der Bus von der Haltestelle zum Treffpunkt mit Lisa benötigt. Wenn man sich die oben abgebildete Grafische Darstellung noch einmal ansieht, sieht man, dass der Treffpunkt bei $(0, y+s)$ liegt; die Strecke, die der Bus also zurücklegen muss, ist einfach durch $y+s$ gegeben. Damit erhalten wir (wieder nach einsetzen der Formel aus Gleichung 1.10 für s) die Formel

$$t_B(P) = \frac{y+s}{v_B} \tag{1.19}$$

$$\Leftrightarrow \qquad t_B(P) = \frac{y + \frac{v_L x_1}{\sqrt{v_B^2+v_L^2}}}{v_B}\,. \tag{1.20}$$

[3] Fermatsches Prinzip. Wikipedia, `https://de.wikipedia.org/wiki/Fermatsches_Prinzip`

Nun können wir unser $\omega(P)$ noch etwas vereinfachen und erhalten damit

$$\omega(P) = t_L(P) - t_B(P) \tag{1.21}$$

$$\Leftrightarrow \quad \omega(P) = x\frac{v_B}{v_L\sqrt{v_B^2 - v_L^2}} - \frac{y + \frac{v_L x}{\sqrt{v_B^2+v_L^2}}}{v_B} \tag{1.22}$$

$$\Leftrightarrow \quad \omega(P) = \frac{x\sqrt{v_B^2 - v_L^2} - yv_L}{v_B v_L} \tag{1.23}$$

als Formel, um die zeitlichen Kosten für die Endkanten zu berechnen.

1.1.3 Der Algorithmus für den Sichtbarkeitsgraphen

Wir müssen zum einen den Sichtbarkeitsgraphen für alle Hindernisse und Lisa berechnen und dann einen Endknoten und die entsprechenden Kanten zu diesem hinzufügen. Wie dieser Graph dann aussieht, ist in Abbildung 1.5 gezeigt. Wir überlegen uns als Nächstes, wie wir den Sichtbarkeitsgraphen berechnen. Hierfür benötigen wir einen Algorithmus, der uns für eine Menge an polygonalen Hindernissen und einem Punkt P (der außerhalb der Hindernisse liegt) alle von P aus sichtbaren Ecken der Hindernisse ermittelt. Dafür gibt es mehrere algorithmische Möglichkeiten.

Der naive Algorithmus

Die offensichtlich einfachste Möglichkeit, alle von P aus sichtbaren Punkte zu finden, ist es, für jeden anderen Punkt Q zu überprüfen, ob sich die Verbindungsstrecke PQ mit irgendeiner Kante der Hindernisse schneidet.

Ob sich zwei Strecken PQ und ST mit $P = (x_1, y_1)$, $Q = (x_2, y_2)$, $S = (x_3, y_3)$ und $T = (x_4, y_4)$ schneiden, kann man mit folgenden Formeln feststellen:[4]

$$s = \frac{x_1(y_3 - y_4) + x_3(y_4 - y_1) + x_4(y_1 - y_3)}{(x_1 - x_2)(y_3 - y_4) - (x_3 - x_4)(y_1 - y_2)} \tag{1.24}$$

$$t = \frac{x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)}{(x_3 - x_4)(y_1 - y_2) - (x_1 - x_2)(y_3 - y_4)} \tag{1.25}$$

Die beiden Strecken schneiden sich, wenn $0 < s < 1$ und $0 < t < 1$ gelten, sonst nicht.

Um dann den Sichtbarkeitsgraphen zu berechnen, müssen wir diesen Algorithmus auf jeden der $n+1$ Ecken (inklusive dem Startpunkt von Lisa) anwenden. Um die letzte Wegstrecke von Lisa zur Straße zu berücksichtigen, berechnen wir für jeden Eckpunkt P mit den Koordinaten x und y zusätzlich seine Projektion P' auf die Straße im Winkel α (Gleichung 1.12) und prüfen ebenso die Sichtlinie PP'. Die Koordinaten x' und y' des Punktes P' berechnen sich mittels

$$x' = 0 \tag{1.26}$$

$$y' = y + x\tan\alpha\,. \tag{1.27}$$

[4] Schnittpunkt. Wikipedia, `https://de.wikipedia.org/wiki/Schnittpunkt#Schnittpunkt_zweier_Strecken`

Sollte P' von P aus sichtbar sein, kann Lisa vom Eckpunkt P aus auf direktem Wege zur Straße gehen; das Gewicht dieser Endkante PP' ist $\omega(P)$. Insgesamt ergibt sich wegen $(n+1)^2$ Eckenpaaren und n Kanten eine Laufzeit des naiven Algorithmus von $O(n^3)$.

Der Sweepline-Algorithmus

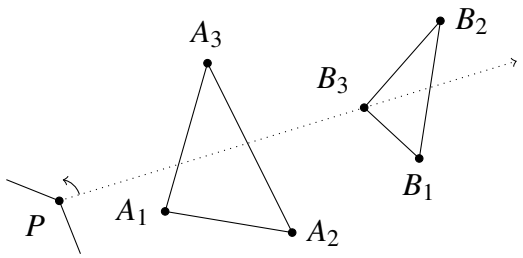


Abbildung 1.6: Während des Sweepline-Algorithmus wird der Suchstrahl, ausgehend von P, einmal an allen anderen Punkten vorbeigeführt. In diesem Beispiel wird gerade B_3 auf Sichtbarkeit von P aus überprüft.

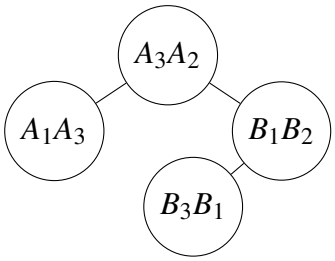


Abbildung 1.7: Der binäre Suchbaum für die Ausgangssituation in Abbildung 1.6. Nachdem B_3 besucht wurde, wird B_3B_1 durch B_2B_3 ersetzt, da B_3B_1 an B_3 endet und B_2B_3 an B_3 beginnt.

Eine etwas effizientere Methode als den eben beschriebenen Algorithmus bietet ein Sweepline-Algorithmus von Der-Tsai Lee.[5] Um alle von einem Punkt P aus sichtbaren Punkte zu berechnen, lässt man eine sogenannte „Sweepline“ einmal alle Punkte besuchen. Dabei geht diese Sweepline strahlenförmig von P aus und rotiert einmal um 360 Grad um P herum. Der Ablauf sieht wie folgt aus:

1. Wir berechnen für jeden Eckepunkt P' der Hindernisse einschließlich Lisas Startpunkt den Winkel zwischen der x-Achse und der Sichtlinie PP' (gegen den Uhrzeigersinn). Die Punkte sortieren wir nach diesen Winkeln.
2. Dann starten wir mit der Sweepline bei Winkel 0 und überprüfen für jede Kante aller Hindernisse, ob diese die (gegen den Uhrzeigersinn) rotierende Sweepline schneidet. Alle so gefundenen Kanten speichern wir in einem binären Suchbaum. Dabei sind die Kanten in diesem Suchbaum immer in der Reihenfolge geordnet, in der die Sweepline sie schneidet, d. h. die Kante, die die Sweepline zuerst schneidet (und somit von P sichbar ist), steht immer ganz links im Suchbaum.
3. Wir gehen nun jeden Punkt P' (in der vorher sortierten Reihenfolge) mit der Sweepline ab. Für jede Kante, die bei P' beginnt (der andere Endpunkt der Kante muss also erst noch von der Sweepline besucht werden), speichern wir diese neu gefundene Kante in unserem Suchbaum so ab, dass die Reihenfolge, in der die Sweepline die Kanten im Suchbaum schneidet, erhalten bleibt. Für jede Kante, die bei P' endet (der andere Kantenendpunkt wurde also schon besucht), löschen wir die Kante aus unserem Suchbaum.
4. Bei jedem Punkt P' können wir nun an unserem Suchbaum ablesen, ob die Kante, zu der P' gehört, ganz links im Suchbaum steht. Dann nämlich ist diese Kante (und damit auch

[5] Dave Coleman: Lee's $O(n^2 \log n)$ visibility graph algorithm: implementation and analysis, 2012, `http://dav.ee/papers/Visibility_Graph_Algorithm.pdf`

P') von P aus sichtbar, andernfalls nicht (da sonst eine oder mehrere Kanten vor P' bzw. zwischen P und P' liegen).

Beispielhaft ist in Abbildung 1.6 ein Schritt des Algorithmus sowie in Abbildung 1.7 der dazugehörige Binärbaum dargestellt. In diesem Beispiel sehen wir, dass B_3 nicht von P aus sichtbar ist, da B_3 nicht Teil der vordersten (d. h. der im Binärbaum am weitesten links stehenden) Kante ist.

Zu jedem Zeitpunkt des Algorithmus sind also alle Kanten, die die Sweepline schneiden, im Binärbaum nach ihrer Entfernung zu P gespeichert. Nun stellt sich noch die Frage, welche Entfernung von P zur Kante man jetzt als Schlüssel für den Binärbaum verwendet (da sich diese Entfernung mit der Bewegung der Sweepline stetig ändert). Würde man immer die Entfernungen nehmen, die man beim erstmaligen Besuchen der jeweiligen Kante berechnet (d. h. die Distanz von P zum ersten gefundenen Endpunkt der Kante), würde sich folgendes Problem ergeben, siehe Abbildung 1.8.

A_1 ist näher an P als B_1. Als A_1 von der Sweepline besucht wurde, wurde deshalb die Kante A_1A_2 mit der geringeren Distanz $d_1 = \overline{PA_1}$ in den binären Suchbaum gespeichert als die andere Kante B_1B_2 mit der Distanz $d_2 = \overline{PB_1}$. Nun würde, wenn die Sweepline B_1 als nächsten Punkt besucht, die beiden Kanten, die von B_1 ausgehen, mit d_2 als Schlüssel im Suchbaum gespeichert werden. Da aber die Kante A_1A_2 bereits mit dem kleinerem Schlüssel d_1 dort vorhanden ist (und damit im Suchbaum ganz links ist), würde man zum (falschen) Ergebnis gelangen, dass B_1 von der Kante A_1A_2 verdeckt wird und damit von P nicht sichbar ist.

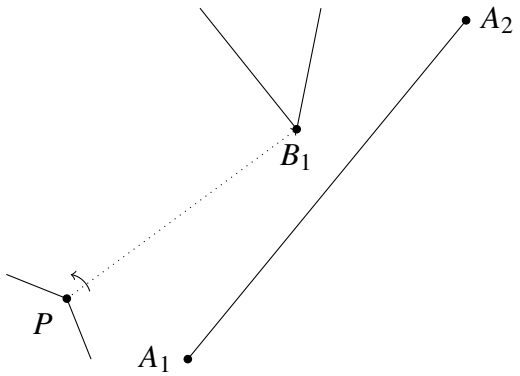


Abbildung 1.8: Der problematische Fall.

Wie stellen wir also sicher, dass die Kanten im Suchbaum immer in der richtigen Reihenfolge stehen, wenn sich diese Distanzen mit der Rotation der Sweepline immer wieder verändern? Um diesem Problem zu begegnen, speichern wir als Schlüssel für den binären Suchbaum keine konkreten Werte als Schlüssel. Stattdessen rechnen wir immer, wenn wir eine Distanz als Schlüssel für eine bestimmte Kante k im binären Suchbaum vergleichen wollen, diese Distanz einfach „on demand“ als Distanz zwischen der *aktuellen* Sweepline und der Kante k aus. Das funktioniert allerdings nur unter der (berechtigten) Annahme, dass sich die Kanten der polygonalen Hindernisse nicht überschneiden; dann nämlich ändert sich die Reihenfolge, in der die Kanten im Suchbaum von der Sweepline geschnitten werden, nicht und somit brauchen wir auch keine konkreten Schlüssel im Suchbaum. Ändert sich die Reihenfolge nicht, ändert sich auch der Suchbaum nicht und somit geht dieser auch nicht „kaputt“, wenn die in ihm gespeicherten Elemente variable Schlüssel haben.

In unserem Beispiel würden wir dann, wenn wir B_1 besuchen, wieder die beiden Kanten, die von B_1 ausgehen, im Suchbaum abspeichern. Wenn wir allerdings jetzt nach der passenden Stelle im Suchbaum suchen, rechnen wir die Distanz von P zum Schnittpunkt zwischen der aktuellen Sweepline und der Kante A_1A_2 neu aus und stellen dann fest, dass diese Distanz jetzt größer ist als die Distanz von P zu B_1; somit fügen wir diese beiden neuen Kanten vor (d. h. im Suchbaum weiter links) von der Kante A_1A_2 ein, so dass B_1 von P aus sichtbar ist.

Wenn wir dann noch nicht nur irgend einen binären Suchbaum, sondern einen selbst-balancierenden binären Suchbaum (z. B. einen AVL-Baum) verwenden, hat das Einfügen, Löschen und Suchen im binären Suchbaum jeweils eine Laufzeit von $O(\log n)$;[6] der Umstand, dass wir bei jedem Schlüsselzugriff diesen Schlüssel erst neu berechnen müssen, macht da keinen Unterschied, da diese Berechnung in konstanter Zeit geschieht. Da wir $n-1$ Punkte zu prüfen haben, hat dieses Verfahren dann insgesamt eine Laufzeit von $O(n \log n)$. Da wir dies wieder für jeden Punkt separat machen müssen, kommen wir insgesamt auf eine Laufzeit von $O(n^2 \log n)$, eine klare Verbesserung gegenüber dem naiven Algorithmus.

Auch hier müssen wir wieder einen Spezialfall für den direkten Weg zur Straße einbauen; dies funktioniert aber im Prinzip genau so wie beim naiven Algorithmus: Auch hier überprüfen wir beim Eckpunkt P noch seine Straßen-Projektion P' (in diesem Algorithmus dann als einzelner Punkt, von dem keine Hindernis-Kanten ausgehen). Sollte P' dann von P aus sichtbar sein, fügen wir eine Kante von P zum Endknoten mit Gewicht $\omega(P)$ ein.

Andere geometrische Algorithmen

Es gibt für das Berechnen des Sichtbarkeitsgraphen noch andere, noch etwas schnellere Algorithmen als den von Der-Tsai Lee.[7,8,9] So gibt es einen modifizierten Sweepline-Algorithmus, der eine Gesamtlaufzeit von $O(n^2)$ besitzt.[10] Außerdem existieren ausgabesensitive Algorithmen (d. h. die Laufzeit ist vom Ergebnis der Berechnung abhängig), die (zumindest in einigen Fällen) eine bessere asymptotische Laufzeit erreichen können (beispielsweise $O(E + n \log n)$, wobei E die Anzahl der Kanten der Ausgabe ist).[11] Da diese Verfahren aber recht aufwändig umzusetzen sind, werden sie an dieser Stelle nicht weiter besprochen.

1.1.4 Algorithmus für den kürzesten Weg

Nachdem wir den Sichtbarkeitsgraph berechnet haben, müssen wir noch den kürzesten Weg in ihm finden. Dies lässt sich mit Standardalgorithmen wie z. B. dem Dijkstra-Algorithmus lösen.[12] Die Kantengewichte in unserem Sichtbarkeitsgraphen entsprechen dabei der jeweiligen Zeit, die Lisa entsprechend der Länge der Wegkante benötigt, und $\omega(P)$ dem Zeitbedarf für eine Endkante, die am Straßenpunkt P' endet.

Bei allen bekannten Kürzester-Pfad-Algorithmen ist die Gesamtlaufzeit des Verfahrens durch das Berechnen der Sichtbarkeitsgraphen dominiert, da das Berechnen des Sichtbarkeitsgraphen im schlechtesten Fall („worst case“) nicht schneller als in $O(n^2)$ machbar ist (da der Sichtbarkeitsgraph im schlimmsten Fall aus n^2 Kanten besteht).

[6] Binärer Suchbaum. Wikipedia, `https://de.wikipedia.org/wiki/Binärer_Suchbaum`

[7] Der-Tsai Lee: Proximity and reachability in the plane. PhD thesis, Champaign, IL, USA, 1978.

[8] Christian Reksten-Monsen: Distance Tables Part 1: Defining the problem, 2016, `https://taipanrex.github.io/2016/09/17/Distance-Tables-Part-1-Defining-the-Problem.html`

[9] Christian Reksten-Monsen: Distance Tables Part 2: Lee's visibility graph algorithm, 2016, `https://taipanrex.github.io/2016/10/19/Distance-Tables-Part-2-Lees-Visibility-Graph-Algorithm.html`

[10] Welzl, Emo: Constructing the visibility graph for n-line segments in $O(n^2)$ time. Information Processing Letters, Vol. 20, 1985.

[11] Ein solcher Algorithmus findet sich in: Ghosh, S. K. und Mount, D. M.: An output sensitive algorithm for computing visibility graphs. 28th Annual Symposium on Foundations of Computer Science, 1987.

[12] Dijkstra-Algorithmus. Wikipedia, `https://de.wikipedia.org/wiki/Dijkstra-Algorithmus`

1.1.5 Innenliegende Kanten

Ein Problem, was uns bei der Berechung der Kanten des Sichtbarkeitsgraphen begegnet, ist, dass wir alle Kanten eliminieren müssen, die *innerhalb* eines Hindernisses verlaufen (welche aber trotzdem nicht durch andere Hinderniskanten unterbrochen werden). Wir müssen also überprüfen, ob eine Kante innerhalb oder außerhalb eines Hindernisses verläuft. Dies kann man – unter der Annahme, dass die Ecken des Hindernisses gegen Uhrzeigersinn sortiert sind – recht einfach überprüfen:

Betrachtet man gerade die Sichtlinie Z zwischen zwei Eckpunkten S und T, so gehört S zu einem bestimmen Hindernis P und hat dementsprechend auch zwei „Nachbarecken“ – ab jetzt $\overrightarrow{S}$ (gegen den Uhrzeigersinn vor S) und $\overleftarrow{S}$ (gegen den Uhrzeigersinn hinter S) – mit denen Z über eine Kante verbunden ist.

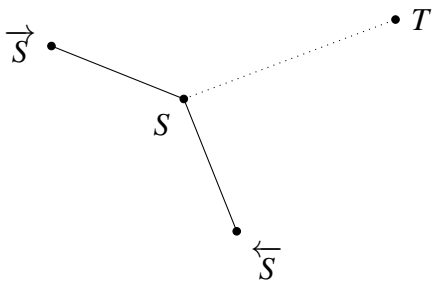


Abbildung 1.9: Die besprochene Situation.

Um nun zu überprüfen, ob die Sichtlinie Z *innerhalb* des Hindernisses verläuft, müssen wir allein überprüfen, ob der (mathematisch positive) Winkel[13] $\angle(\overleftarrow{S}ST)$ größer ist als $\angle(TS\overrightarrow{S})$; falls ja, verläuft Z innerhalb des Hindernisses P, sonst nicht.

Um allerdings sicherzustellen, dass alle Ecken eines Hindernisses auch tatsächlich gegen den Uhrzeigersinn sortiert sind, können wir einfach den vorzeichenbehafteten Flächeninhalt f des Hindernispolygons mit den Eckpunkten A, B, ... berechnen. Dies geht am einfachsten über folgende Formel:

$$f = \frac{1}{2}\left(\det\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} + \det\begin{bmatrix} b_1 & c_1 \\ b_2 & c_2 \end{bmatrix} + \ \ldots\ + \det\begin{bmatrix} z_1 & a_1 \\ z_2 & a_2 \end{bmatrix}\right) \tag{1.28}$$

Der Wert dieses Ausdrucks ist genau dann positiv, wenn die Ecken des Hindernispolygons gegen den Uhrzeigersinn sortiert sind. Falls f also negativ ist, müssen wir die Liste der Ecken von P also einfach nur invertieren.

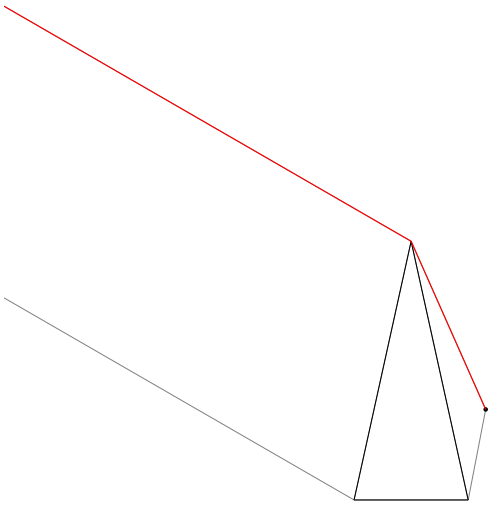
1.2 Beispiele

Für jedes der fünf vorgegebenen Beispiele ist im Folgenden nur jeweils eine Lösung als Ergebnis angegeben, insbesondere für Beispiel 5 gibt es mehrere Lösungen mit auf Sekunden gerundeter identischer Laufzeit für Lisa. In allen Abbildungen ist der rote Pfad die Lösung, d. h. Lisas optimaler Weg; die grauen Kanten stellen den Rest des Sichtbarkeitsgraphen dar.

[13] Dieser Winkel lässt sich in den allermeisten Programmiersprachen mit der `atan2`-Funktion berechnen.

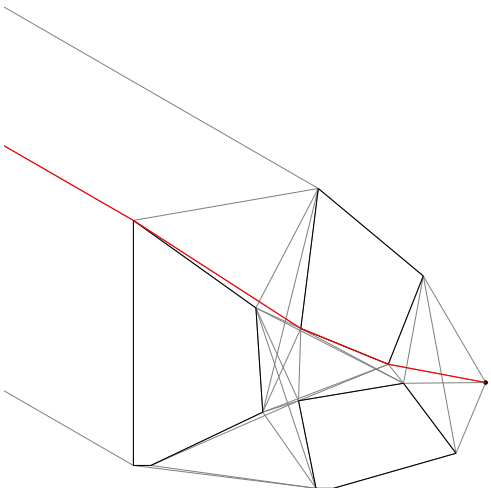
Beispiel 1: lisarennt1.txt

```
Startzeit: 07:27.59,981 (∼07:28.00)
Zielzeit: 07:31.26,266 (∼07:31.26)
y-Koordinate Treffpunkt: 718.8824 (∼719)
Länge von Lisas Route: 859.5188 (∼860)
Dauer von Lisas Route: 3.26,285 (∼3.26) / 206,285 (∼206)

Pfad: L (633, 189) -> P1 (535, 410) -> Straße (0, 718.8824)
```



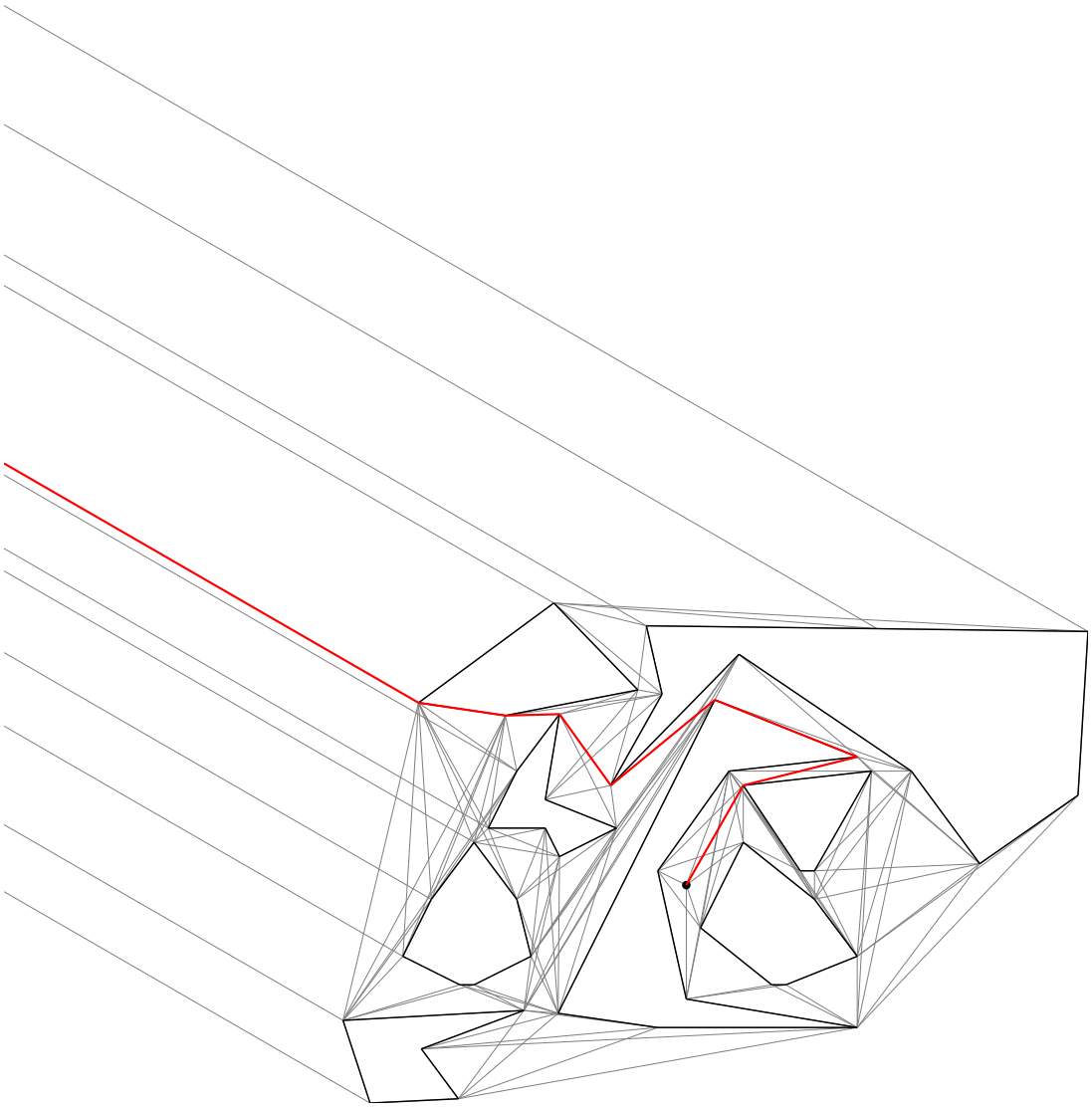
Beispiel 2: lisarennt2.txt

```
Startzeit: 07:28.08,991s (∼07:28.09)
Zielzeit: 07:31.00,018s (∼07:31.00)
y-Koordinate Treffpunkt: 500.1495 (∼500)
Länge von Lisas Route: 712.6106 (∼713)
Dauer von Lisas Route: 2.51,027 (∼2.51) / 171,027 (∼171)

Pfad: L (633, 189) -> P1 (505, 213) -> P1 (390, 260) -> P3 (170, 402) -> Straße
(0, 500.1495)
```



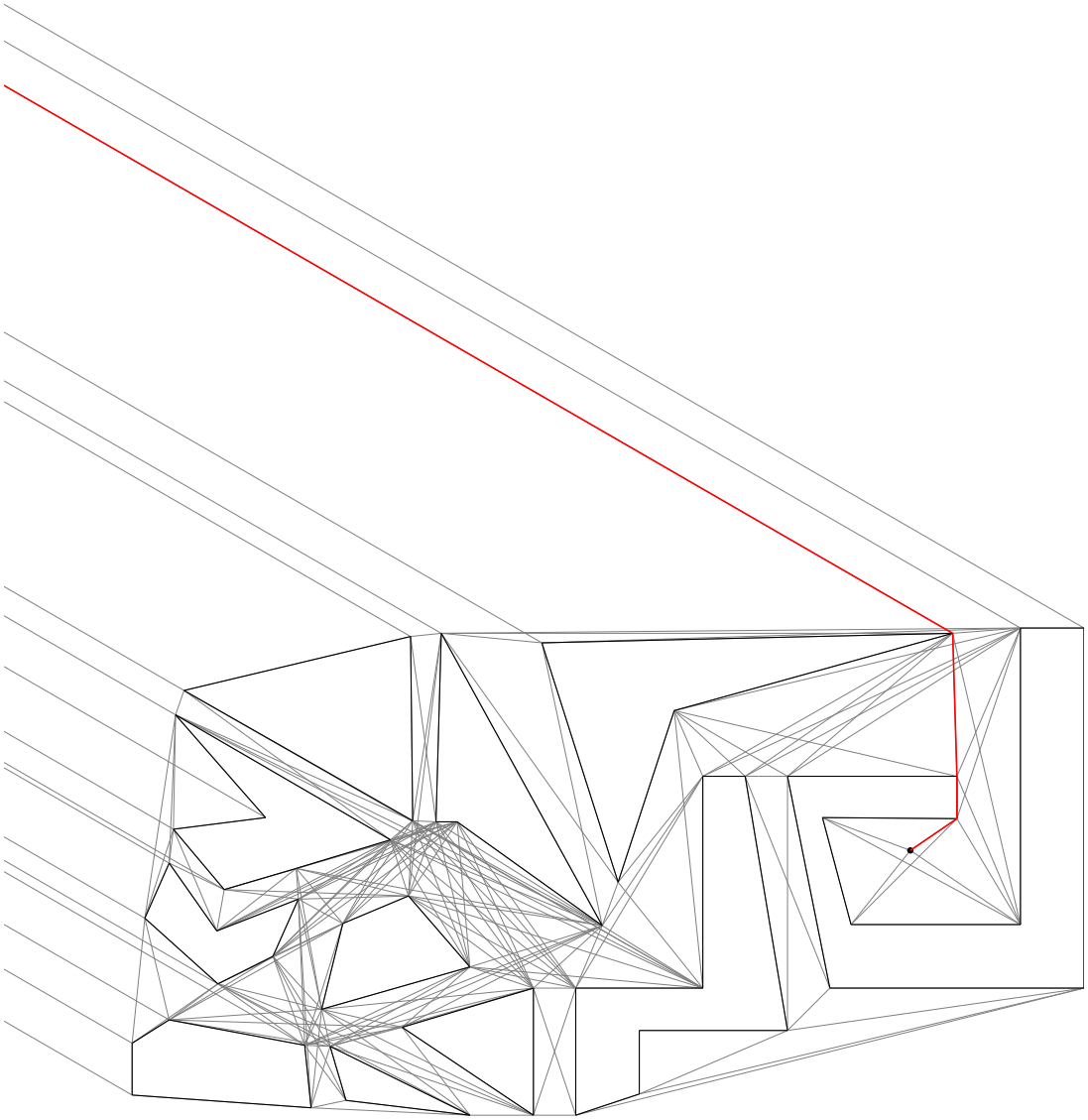
Beispiel 3: lisarennt3.txt

```
Startzeit: 07:27.28,661 (∼07:27.29)
Zielzeit: 07:30.55,681 (∼07:30.56)
y-Koordinate Treffpunkt: 464.0089 (∼464)
Länge von Lisas Route: 862.5844 (∼863)
Dauer von Lisas Route: 3.27,020 (∼3.27) / 207,020 (∼207)

Pfad: L (479, 168) -> P2 (519, 238) -> P3 (599, 258) -> P3 (499, 298) -> P8
(426, 238) -> P5 (390, 288) -> P6 (352, 287) -> P6 (291, 296) -> Straße (0,
464.0089)
```



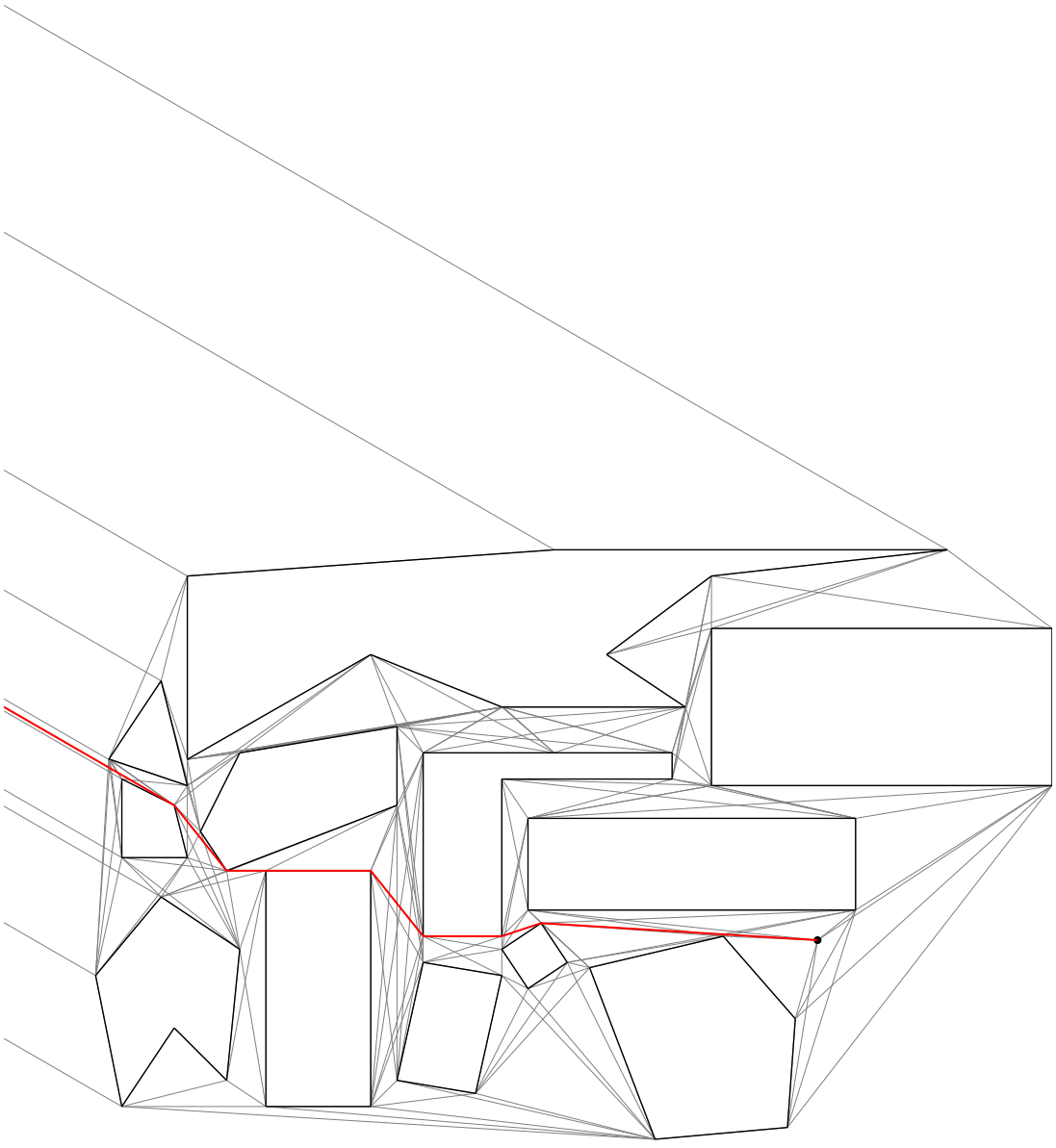
Beispiel 4: lisarennt4.txt

```
Startzeit: 07:26.55,975s (∼07:26.56)
Zielzeit: 07:31.59,077s (∼07:31.59)
y-Koordinate Treffpunkt: 992.3058 (∼992)
Länge von Lisas Route: 1262.925 (∼1263)
Dauer von Lisas Route: 5.03,102 (∼5.03) / 243,102 (∼243)

Pfad: L (856, 270) -> P11 (900, 300) -> P11 (900, 340) -> P10 (896, 475) ->
Straße (0, 992.3058)
```



Beispiel 5: lisarennt5.txt

```
Startzeit: 07:27.54,920s (∼07:27.55)
Zielzeit: 07:30.40,807s (∼07:30.41)
y-Koordinate Treffpunkt: 340.0555 (∼340)
Länge von Lisas Route: 691.1964 (∼691)
Dauer von Lisas Route: 2.45,887 (∼2.46) / 165,887 (∼166)

Pfad: L (621, 162) -> P8 (410, 175) -> P3 (380, 165) -> P3 (320, 165) -> P5
(280, 215) -> P5 (200, 215) -> P6 (170, 215) -> P9 (130, 265) -> Straße (0,
340.0555)
```



1.3 Mögliche Erweiterungen

Im Folgenden sind mögliche Erweiterungen der Aufgabenstellung aufgeführt.

- Beliebige Geschwindigkeiten für Bus und Lisa: Lisa und der Bus könnten beliebige Geschwindigkeiten haben statt fest vorgegebene; diese Erweiterung ist in der obigen Lösungsidee bereits berücksichtigt.
- Andere Formen für Lisa: Lisa ist in der Aufgabe punktförmig, aber man könnte ihr die Form eines Kreises oder eines Polygons geben.
- Mehrere Buslinien: Es könnte mehrere Buslinien geben, die unterschiedliche Straßen abfahren und verschieden lang brauchen.
- Mehr Dimensionen: Das Problem könnte man in die dritte Dimension übertragen. Der Weg verliefe dann nicht mehr nur über Ecken, sondern potentiell auch über die Flächen der Hindernisse. Die Schwierigkeit bestände hierbei vor allem darin, dass sich der Lösungsansatz mit dem Sichtbarkeitsgraphen nicht einfach auf höhere Dimensionen übertragen lässt.
- Parkour: Lisa könnte Parkour machen und Hindernisse mit entsprechendem Aufwand überqueren. Dazu könnte man jedem Hindernis einen Schwierigkeitsgrad zuweisen, mit dem man das Hindernis überqueren kann.

1.4 Bewertungskriterien

Einige der Bewertungskriterien werden hier näher erläutert:

- (Zu 1.1) Eine korrekte Modellierung des Problems erfordert geradlinige Wege zwischen Polygonecken sowie einen optimalen Winkel am Wegende zur Straße.
- (Zu 1.2) Das Verfahren gilt nur dann als komplett korrekt, wenn der Winkel am Wegende 30 Grad beträgt (bzw. bei beliebigen Geschwindigkeiten von Bus und Lisa korrekt berechnet wird).
- (Zu 1.3) Der einfache Algorithmus löst das Problem mit einer Laufzeit von $O(n^3)$. Daher sollte das Verfahren der Einsendung nicht schlechter sein.
- (Zu 1.4) Das Verfahren muss stets optimale (d.h. kürzeste) Ergebnisse liefern, die korrekt sind: Der Weg darf nicht durch Hindernisse verlaufen, muss bei Lisas Haus starten und an der Landstraße vor dem Einsteigen in den Bus enden.
- (Zu 1.5) Es genügt, wenn die Aufgabe für die vorgegebenen Geschwindigkeiten von Bus und Lisa gelöst wurde. Wurde die Aufgabe allgemein für beliebige Geschwindigkeiten von Bus bzw. Lisa gelöst, wird dies als eine mathematisch-physikalische Leistung mit wenigen Pluspunkten belohnt.
- (Zu 2.4) Ein vollständiger Beweis des 30-Grad-Winkels im mathematischen Sinne wird nicht erwartet, aber eine nachvollziehbare Begründung; das Fehlen einer verständlichen Begründung für den 30-Grad-Winkel führt daher zu Minuspunkten.
- (Zu 2.5) Ein vollständiger mathematischer Beweis des 30-Grad-Winkels am Wegende gibt Pluspunkte.

- (Zu 3.1) Bei konkaven Polygonen sollte beachtet werden, dass auch nichtbenachbarte Ecken zueinander sichtbar sein können. Nicht nur aus diesem Grund sollte die Erstellung des Sichtbarkeitsgraphen gut dokumentiert sein.
- (Zu 3.3) Bei den Rechenergebnissen sind Rundungen der Zeiten und Wegstrecken auf Sekunden bzw. Meter in Ordnung. Insbesondere beim fünften Beispiel gibt es bei Rundungen auf Sekundengenauigkeit mehrere richtige Lösungen. Die Angabe einer optimalen Lösung je Beispiel genügt.
- (Zu 3.5) Eine visuelle Darstellung der berechneten Wege ist für die vorgegebenen Beispiele ein Muss, sie kann jedoch auch nur manuell (nicht automatisch) generiert sein. Jedes Ergebnis sollte mindestens vier der folgenden Werte umfassen, wobei Lisas Startzeit gemäß der Aufgabenstellung angegeben werden muss: Start- und Zielzeit, y-Koordinate, Dauer, Länge, x-/y-Koordinaten aller Eckpunkte von Lisas Route.

Aufgabe 2: Dreiecksbeziehungen

2.1 Lösungsidee

2.1.1 Notwendigkeit einer Heuristik

Bei dieser Aufgabe ist es gerechtfertigt, eine Heuristik zu verwenden. Aufgrund sehr ähnlicher Probleme,[14,15] die NP-schwer sind, ist davon auszugehen, dass auch das in dieser Aufgabe zu lösende Problem NP-schwer ist. Es gibt $N!$ mögliche Anordnungen der N Dreiecke, und bei jeder Anordnung müssen zusätzlich noch $O(N)$ Winkel und Abstände fixiert werden. Das ist selbst für Supercomputer für eine größere Zahl N nicht in akzeptabler Zeit berechenbar.

Zur Lösung der Aufgabe nutzen wir daher die folgende Heuristik: Wir bauen die Siedlung schrittweise von links nach rechts auf (Abbildung 2.1). Den Vorgang, ein neues Dreieck an die Siedlung anzubauen, nennen wir einen Zyklus. Die Siedlung entsteht somit in aufeinanderfolgenden Zyklen. Dabei wählen wir in einem Zyklus als nächstes Dreieck immer das aus, dessen Schwerpunkt am weitesten nach links in die bestehende Siedlung hineingebracht werden kann. Die Siedlung wächst also, bis alle Dreiecke angelegt sind.

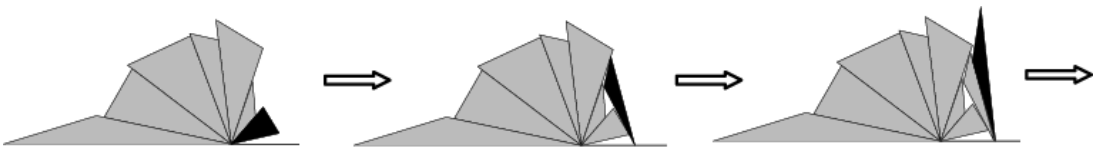


Abbildung 2.1: Die Siedlung wird schrittweise von links nach rechts aufgebaut.

Kurz erwähnen möchten wir eine weitere Heuristik, die wir hier nur am Rande benutzen, auch wenn sie von einigen Teilnehmenden intensiv eingesetzt wurde: die Anordnung der Dreiecke in einzelnen „Fächern“. Jeder Fächer hat einen zentralen Punkt, den alle seine Dreiecke gemeinsam berühren. Und je nach Optimierungsgrad könnte man dann zum Beispiel Dreiecke zwischen den Fächern austauschen, unvollständige Fächer zulassen oder die Fächer so anordnen, dass sie möglichst kompakt aneinander passen.

2.1.2 Durchprobieren der Dreiecke

Wir suchen bei unserer Heuristik in jedem Zyklus ein neues Dreieck mit einer bestimmten Rotation, das wir an die Siedlung anlegen. Dazu simulieren wir, wie jedes noch verfügbare Dreiecke passt, indem wir es einmal komplett rotierend an die Siedlung legen (Abbildung 2.2). Wir rotieren die Dreiecke dabei schrittweise um 1 Grad, und für jede neue Rotation berechnen wir die Distanz in x-Richtung vom Schwerpunkt des Dreiecks zur Siedlung; diesen Wert nennen wir Abstand. Am Ende wählen wir dasjenige Dreieck in demjenigen Rotationsgrad, bei dem der Abstand minimal ist.

[14] Amy Chou: NP-Hard Triangle Packing Problems, 2016.

[15] Ruimin Wang, Yuqiang Luo, Jianqiang Dong, Shuai Liu, and Xiaozhuo Qi: Heuristic Algorithm For Solving Triangle Packing Problem, 2013.

2.1.3 Wahl des Schwerpunkts

Die Idee unserer Heuristik ist, beim Anlegen eines Dreiecks möglichst viel „Masse“ nach links zu bringen. Da der Schwerpunkt der physikalische Massenschwerpunkt ist, war das die naheliegendste Wahl. Wir haben auch andere Möglichkeiten ausprobiert, z.B. den am weitesten links und/oder rechts liegenden Punkt des anzulegenden Dreiecks oder den Inkreismittelpunkt statt den Schwerpunkt zu verwenden. Diese anderen Ansätze ergaben aber immer Fälle, in denen sie sehr schlechte Ergebnisse lieferten, während die Heuristik mit dem Schwerpunkt fast immer verlässlich funktionierte. Das Problem beim am weitesten links und/oder rechts liegenden Punkt war unter anderem, dass sie Extremwerte ergeben, die nicht berücksichtigen, wie das Dreieck genau aussieht. Der Inkreismittelpunkt dagegen hat insbesondere bei gleichschenkligen, langen Dreiecken versagt, dort liegt er viel zu sehr am Rand. Mit anderen, ausgefeilteren Ansätzen könnte man den Algorithmus noch verbessern; für unsere Zwecke hat der Schwerpunkt genügt.

2.1.4 Modellierung

Wir benutzen Strecken und Ecken, um die Form einer Siedlung bzw. eines Dreiecks zu beschreiben. Die Strecken, die stets Dreiecksseiten sind, sind jeweils durch drei Parametern (Steigung a, Offset dx und maximale Höhe $ymax$) und die Ecken jeweils durch zwei Parametern (Höhe y und Offset dx) bestimmt (Abbildung 2.3).

Die Offsets beschreiben dabei die horizontale Distanz dx zu einem Referenzpunkt. Es gibt zwei Referenzpunkte, jeweils einen für die Siedlung und einen für das Dreieck. Bei der Siedlung ist der Referenzpunkt der am weitesten rechts liegende Kontakt mit der x-Achse, beim Dreieck sein Schwerpunkt.

2.1.5 Durchprobieren der Ecken

Um den jeweiligen Abstand zwischen Siedlung und einem weiteren, noch verfügbaren Dreieck zu berechnen, „schieben“ wir das Dreieck an die Siedlung heran, so dass sich Dreieck und Siedlung berühren (Abbildung 2.4). Der Kontakt der Siedlung zum Dreieck wird immer zwischen einer Ecke (bzw. Strecke) und einer Strecke entstehen.

Es wird von rechts nach links entlang der Außenform der Siedlung eine Liste der möglichen Ecken (bzw. Strecken) der Siedlung erstellt, die einen Kontakt mit dem Dreieck bilden könnten.

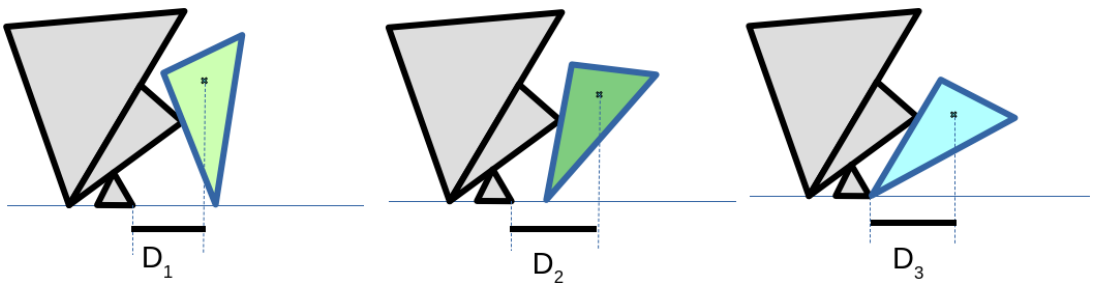


Abbildung 2.2: Verschiedene Rotationen jedes Dreiecks werden durchprobiert. Von allen Rotationen aller Dreiecke suchen wir diejenige mit minimalen Abstand D. Zur schnelleren Berechnung prüfen wir Rotationen nur in Schritten von 1 Grad.

Ecken
1: $[y_1, dx_1]$
2: $[y_2, dx_2]$
3: $[y_3, dx_3]$
4: $[y_4, dx_4]$
5: $[y_5, dx_5]$
6: $[y_6, dx_6]$
7: $[y_7, dx_7]$
8: $[y_8, dx_8]$

Strecken
A: $[a_A, dx_A, ymax_A]$
B: $[a_B, dx_B, ymax_B]$
C: $[a_C, dx_C, ymax_C]$
D: $[a_D, dx_D, ymax_D]$
E: $[a_E, dx_E, ymax_E]$
F: $[a_F, dx_F, ymax_F]$

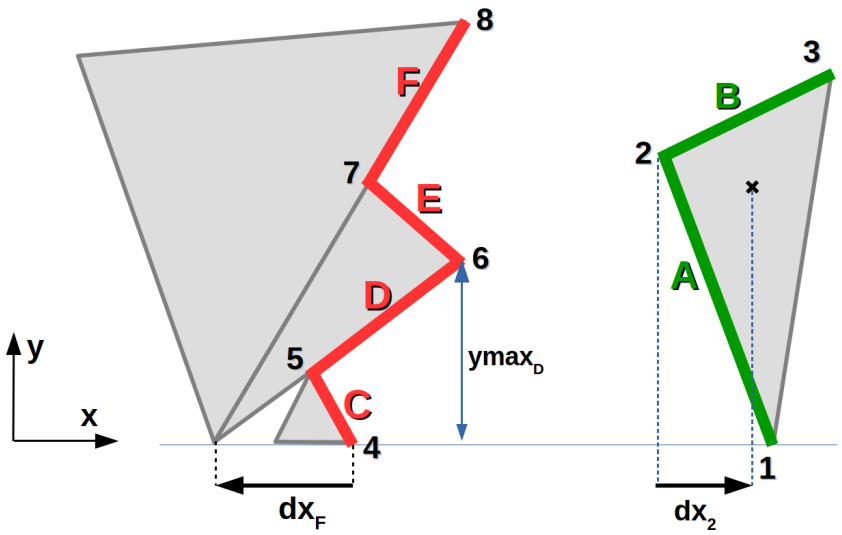


Abbildung 2.3: Strecken der Dreiecke bzw. ihre Ecken werden durch Steigungen (a), Höhen ($ymax$ bzw. y) und Offsets (dx) beschrieben.

Hierbei wird eine weitere Ecke (bzw. Strecke) der Siedlung in die Liste aufgenommen, wenn sie eine höhere y-Koordinate hat als alle bisher in die Liste aufgenommenen Ecken (bzw. Strecken). Die jeweilige Liste wird jedes Mal aktualisiert, wenn die Siedlung erweitert wird.

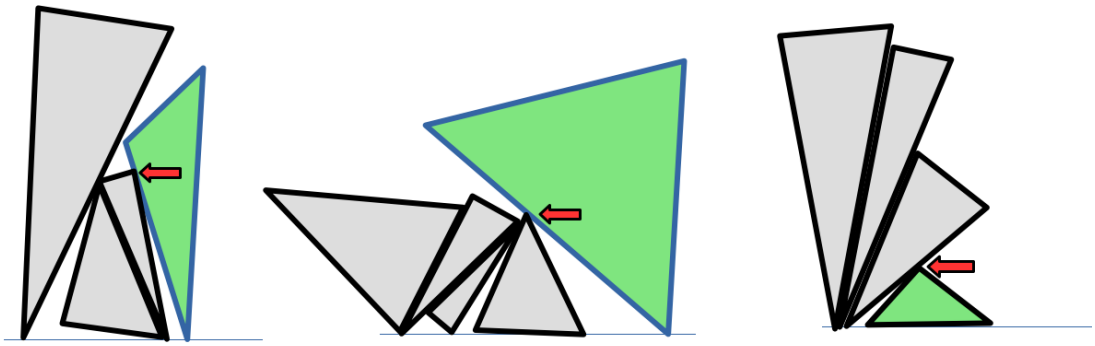


Abbildung 2.4: Der Kontakt von einem weiteren Dreieck zur Siedlung findet immer an einer Ecke statt. In der Abbildung sind drei verschiedene Szenarien beispielhaft skizziert.

Die Frage ist nun noch, an welcher Ecke der Erstkontakt stattfindet. Dies finden wir dadurch heraus, dass wir nacheinander alle Ecken durchprobieren, d.h. Dreieck und Siedlung so aneinander schieben, dass jeweils eine Ecke auf ihre gegenüberliegende Strecke trifft (Abbildung 2.5). Jede Ecke hat eine eindeutig gegenüberliegende Strecke, welche durch die Höhe der Ecke gegeben ist. Bei allen bis auf einen dieser simulierten Kontakte wird es zu Überlappungen zwischen der Siedlung und dem Dreieck kommen. Wir suchen also diejenige Ecke, bei der noch keine Überlappung, sondern nur ein Kontakt entsteht. Dies erkennen wir daran, dass der jeweils berechnete Abstand maximal ist.

Der Vorteil dieser Methode ist, dass sie wesentlich schneller berechenbar ist als übliche Verfahren zur Kollisionsberechnung wie z.B. solche, die auf Schnitten von Strecken bzw. Geraden

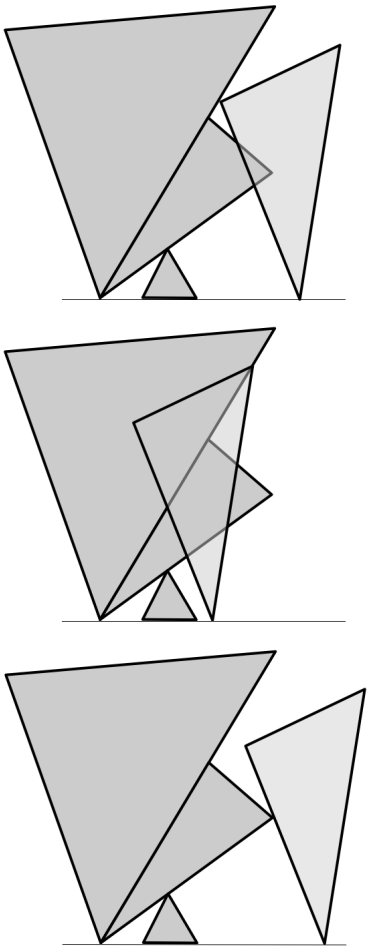


Abbildung 2.5: Der Kontakt an jeweils einer Ecke wird geprüft. Wie in der Skizze dargestellt gibt es hier drei verschiedene Ecken zu prüfen. Die korrekte Ecke ergibt sich aus dem maximalen Abstand des Dreiecks zur Siedlung.

basieren.

Zur Berechnung des Abstands für eine bestimmte Ecke benötigen wir fünf Werte: die Höhe y der Ecke, wodurch sich die Strecke auf der gegenüberliegenden Seite ergibt, die Steigung a von dieser Strecke, und von sowohl der Ecke als auch der Strecke die Offsets dx_e und dx_s zu den beiden Referenzpunkten. Damit berechnet sich der Abstand D wie folgt aus der Summer dreier Teildistanzen (Abbildung 2.6):

$D = y \cdot a + dx_e + dx_s$

Insbesondere der erste Distanzterm $y \cdot a$ repräsentiert den Kontakt zwischen der Siedlung und des weiteren Dreieck. Sein Wert stellt hier eine Teildistanz in der positiven Richtung der x-Achse dar.

Da zunächst unklar ist, welche Ecke die tatsächliche Kontaktecke ist, werden allen Ecken geprüft. Deshalb zeigen wir in Abbildung 2.7, wie zwei weitere Ecken geprüft werden. Bei diesen reihen sich die einzelnen Teildistanzen ebenfalls aneinander, aber sie können verschiedene Vorzeichen haben.

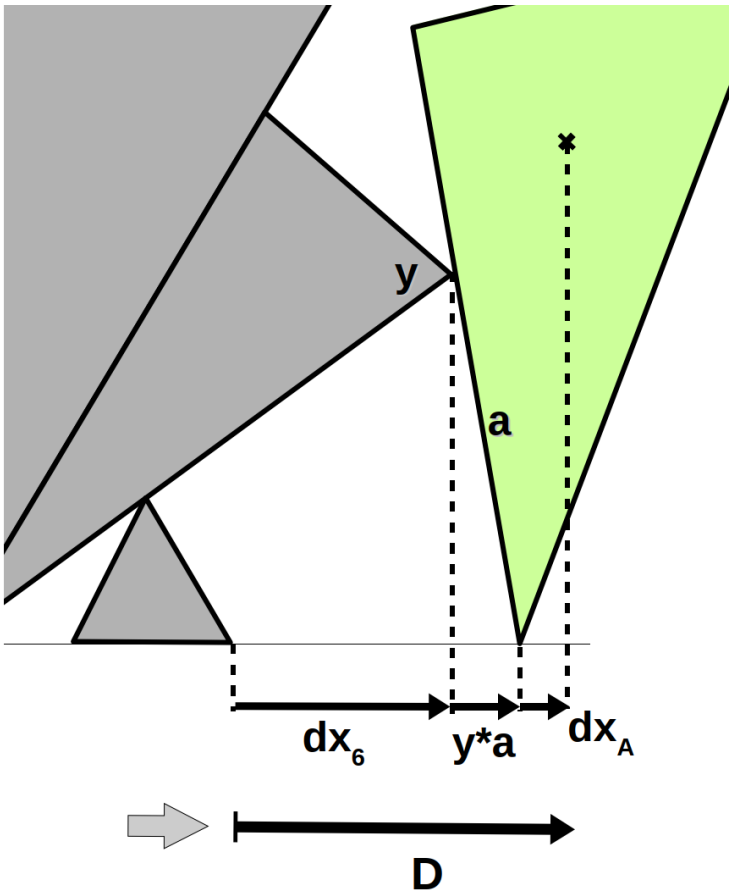


Abbildung 2.6: Der Abstand *D* für eine Ecke wird aus drei Teildistanzen berechnet.

2.1.6 Vorberechnen der Parameter

Die Parameter Höhe, Steigung und Offset berechnen wir bereits beim Einlesen der Dreiecke vor, um den Ablauf des Programms deutlich zu beschleunigen. Beim Siedlungsbau werden dieselben Parameter hundertfach bis tausendfach benötigt; also lohnt es sich, sie einmalig am Anfang alle zu berechnen und zu speichern. Jeder Parameter bezieht sich nur auf ein einzelnes Dreieck, unabhängig von den restlichen Dreiecken, so dass damit in linearer Zeit gerechnet werden kann.

Wir extrahieren zunächst 6 Längen und 9 Winkel aus jedem Dreieck; die Werte sind in Abbildung 2.8 gezeigt. Danach werden die Rotationen für jedes Dreieck durchgeführt. Hierbei wird ein Dreieck jeweils in Schritten von 1 Grad um eine seiner Ecken gedreht, und für jeden Schritt werden die Koordinaten des Schwerpunkts sowie der beiden anderen Ecken ermitteln (relativ zu der Ecke, um die gedreht wird), um daraus dann jeweils die Parameter Höhe, Steigung und Offset zu berechnen.

Die Rotationen eines Dreiecks geschehen in sechs Phasen. In jeder Phase wird jeweils um eine Ecke gedreht, d.h. um drei Ecken insgesamt und zusätzlich um drei Ecken in der gespiegelten Version des Dreiecks. Die erste Phase ist die Rotation um den kleinsten Winkel, beginnend mit der längsten Seite horizontal, siehe Abbildung 2.9. Den aktuellen Rotationswinkel, den wir schrittweise um 1 Grad erhöhen, nennen wir α.

Die Koordinaten der Ecken und des Schwerpunkts sind durch folgende Formeln gegeben, wobei die Indizes für (l)inks, (r)echts und (S)chwerpunkt stehen:

$$xl = -l_2 \cdot cos(\alpha)$$
$$yl = l_2 \cdot sin(\alpha)$$

$$xr = -l_3 \cdot cos(\theta_1 + \alpha)$$
$$yr = l_3 \cdot sin(\theta_1 + \alpha)$$

$$xs = -ls_1 \cdot cos(\theta_{12} + \alpha)$$
$$ys = ls_1 \cdot sin(\theta_{12} + \alpha)$$

Wenn um die erste Ecke fertig rotiert wurde, startet in der zweiten Phase die Rotation um die zweite Ecke. In Abbildung 2.10 sind alle 6 Rotationsphasen dargestellt. Insgesamt wird das Dreieck einmal um 360 Grad in seiner „normalen“ Version (wie in der Eingabedatei gegeben) und einmal um 360 Grad in der gespiegelten Version gedreht. Das macht 720 diskrete Rotationen bei unserer gewählten Schrittweite von 1 Grad. Für jedes Dreieck werden somit ca. 15.000 Double-Werte gespeichert, was ca. 0,12 MB je Dreieck sind.

Das Zwischenergebnis je Rotationsschritt sind die Koordinaten (xl, yl), (xr, yr) und (xs, ys) relativ zum Rotationspunkt. Mit diesen Koordinaten berechnen wir die Parameter Höhe, Steigung und Offset. Wir benutzen die Indizes L, R und T, um damit die Strecken Links, Rechts bzw. Top eines Dreiecks wie in Abbildung 2.11 zu bezeichnen. Ursprung des Koordinatensystems ist der Fußpunkt des Dreiecks, also diejenige Ecke, die die x-Achse berührt.

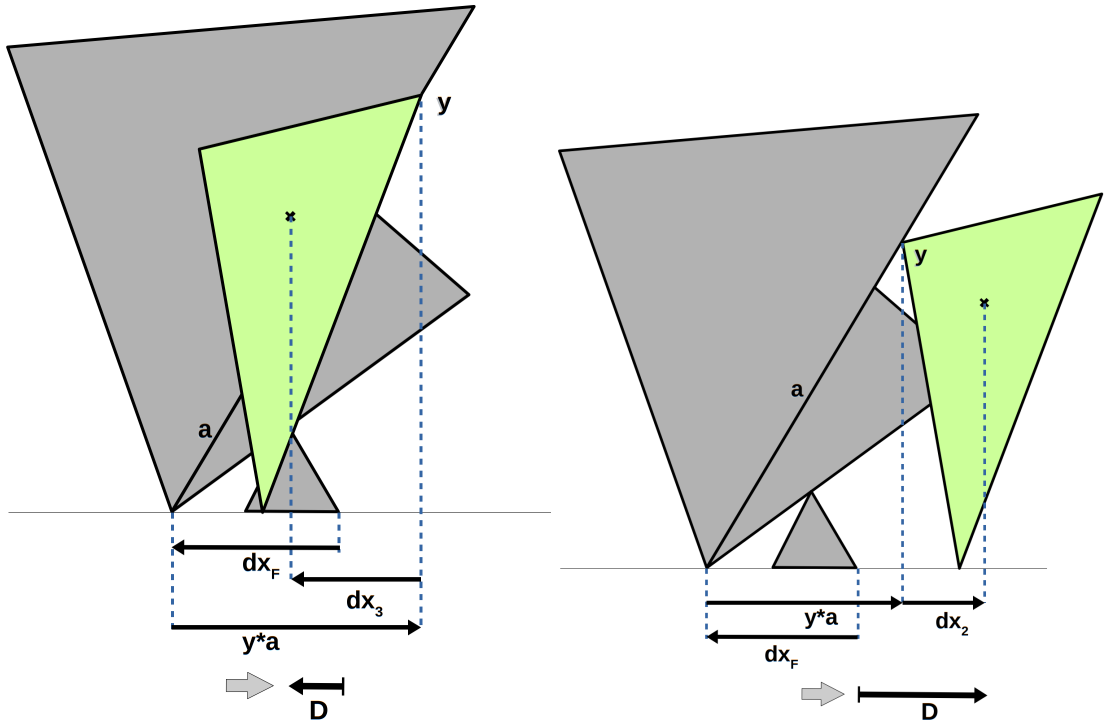


Abbildung 2.7: Der Abstand D wird für zwei weitere Ecken ausgerechnet. Die Pfeilrichtungen kennzeichnen die Vorzeichen der einzelnen Teildistanzen (nach rechts positiv, nach links negativ).

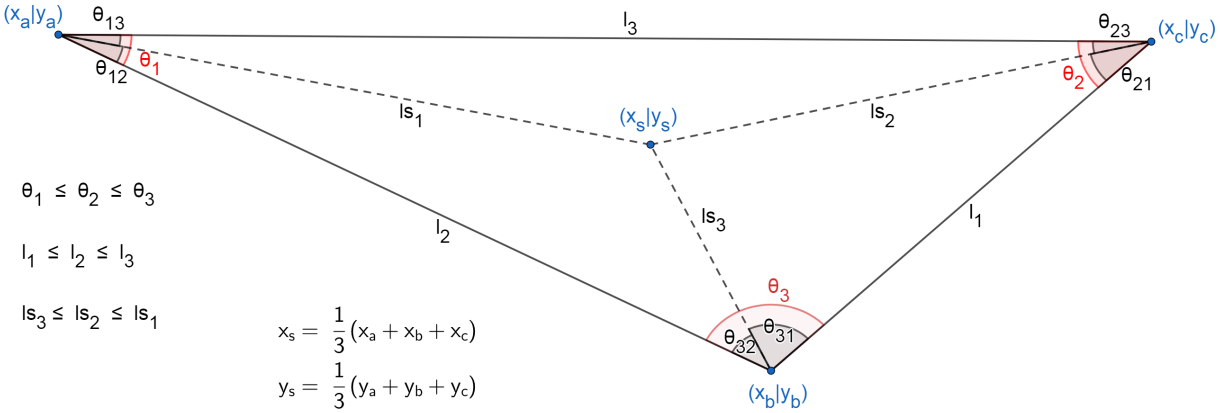


Abbildung 2.8: Eine Übersicht der 15 Werte, die aus jedem Dreieck extrahiert werden. Die berechneten Schwerpunktkoordinaten (xs, ys) werden nicht gespeichert.

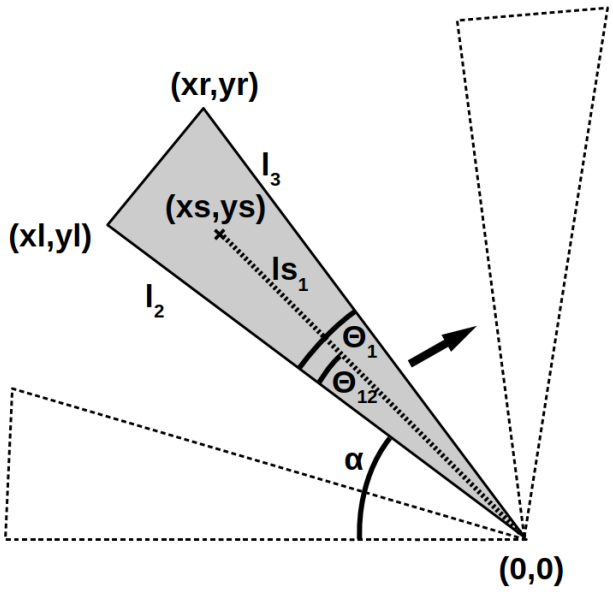


Abbildung 2.9: Ein Dreieck wird um Winkel α gedreht und die Koordinaten seiner Ecken neu berechnet.

Strecken

L: Steigung: $a_L = -xl/yl$
L: Offset: $dx_L = xs$

R: Steigung: $a_R = xr/yr$
R: Offset: $dx_R = 0$

T: Steigung: $a_T = (+/-)(xr - xl)/(yr - yl)$ (positiv, wenn $yl < yr$, sonst negativ)
T: Offset: $dx_T = -a_T \cdot yl + (xs - xl)$ (wenn $yl < yr$) oder $dx_T = -a_T \cdot yr + xr$ (wenn $yl > yr$)

Ecken

Rechte Ecke:
Offset zu Fuß: $dxf_r = xr$
Offset zu Schwerpunkt: $dxs_r = xs - xr$

Linke Ecke:
Offset zu Fuß: $dxf_l = xl$
Offset zu Schwerpunkt: $dxs_l = xs - xl$

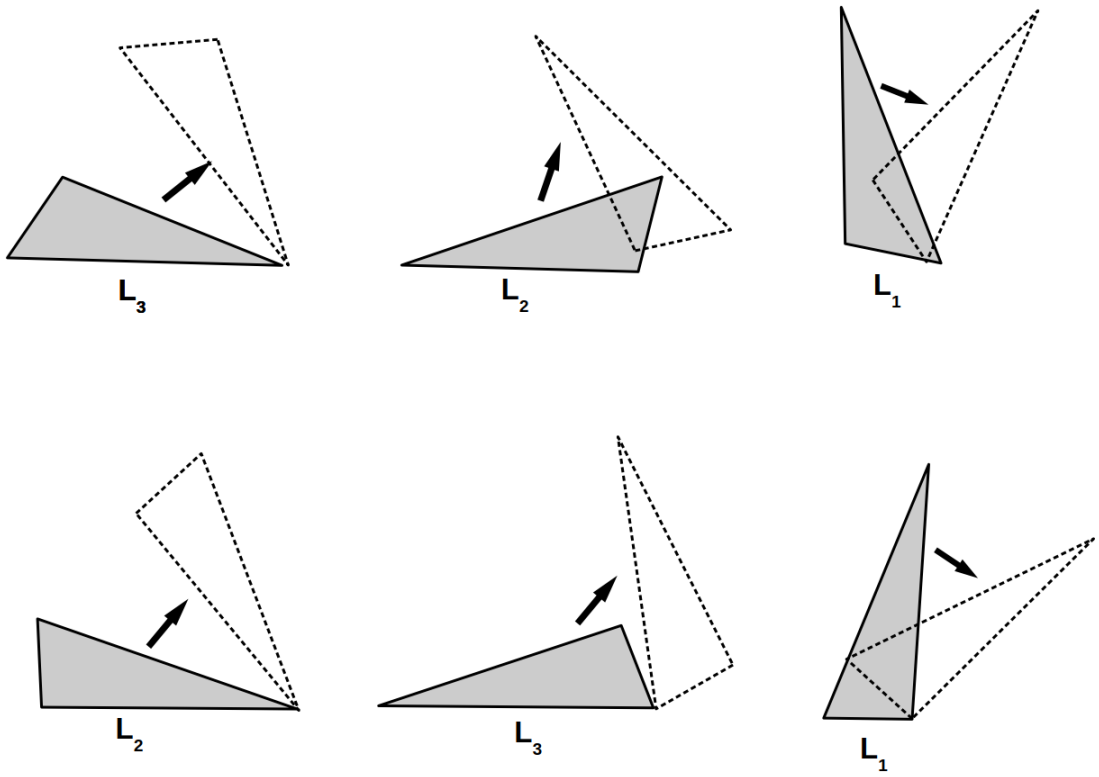


Abbildung 2.10: Die Rotation eines Dreiecks gliedert sich in sechs Phasen.

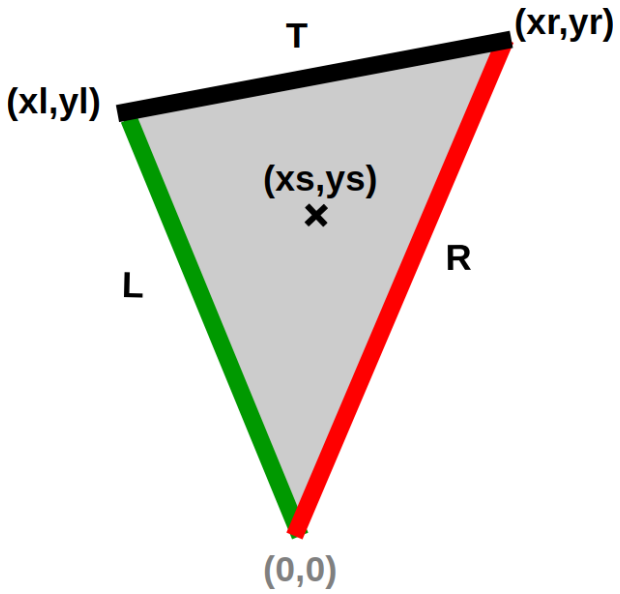


Abbildung 2.11: Parameter Höhe, Steigung und Offset je Dreieck.

Auf diese vorberechneten Strecken- und Eckparameter wird während des Verfahrens immer wieder zugegriffen. Dies ist eine Stärke des Verfahrens: Beim gesamten Siedlungsbau muss nur noch wenig neu berechnet werden, da hauptsächlich schon berechnete Parameter nur mehr in verschiedenen Kombinationen angewandt werden.

Die L-, R- und T-Strecken sind für unterschiedliche Zwecke relevant. Zum Beispiel wird beim Anlegen eines weiteren Dreiecks an die bereits bestehende Siedlung auf die L-Strecke und ggf. auf die T-Strecke zugegriffen, weil nur diese beiden Strecken für das Anlegen relevant sind. Die

R-Strecke des anzulegenden Dreiecks zeigt nämlich von der Siedlung weg und kann mit dieser nicht in Berührung kommen.

Analog wird zum Beschreiben der Außenform der Siedlung nur auf R-Strecken und ggf. T-Strecken zurückgegriffen. Meistens besteht die Außenform aus mehreren Strecken, so dass sie durch die verschiedenen R-Strecken der beteiligten Dreiecke beschrieben wird, jeweils eine R-Strecke für einen bestimmten Höhenabschnitt.

Die T-Strecke kann sowohl bei einer Siedlung als auch bei einem Dreieck relevant sein. Dies hängt davon ab, ob die T-Strecke nach rechts oder nach links geneigt ist. Deshalb wird dieser Unterschied berücksichtigt und die Parameter der T-Strecke werden entsprechend gesetzt. Ab dann erfüllt die T-Strecke dieselben Funktionen wie entweder die L-Strecken für das Dreieck oder die R-Strecken für die Siedlung. Wegen dieser Doppelrolle ist die zusätzliche Fallunterscheidung nötig, wie sie oben bei den Formeln bei der T-Strecke angegeben ist.

2.1.7 Starter und Stopper

Zum Starten und Beenden der Siedlung weichen wir von der Heuristik mit dem Schwerpunkt ab und benutzen eine andere Heuristik (Abbildung 2.12): Wir fächern die Dreiecke, jeweils mit ihrem spitzesten Winkel, um einen gemeinsamen Punkt. Grund dafür ist, dass am Rand der Siedlung Dreiecke auch sehr flach hingelegt werden können, wenn man weiß, dass keine weiteren Dreiecke mehr dazu kommen. Es geht darum, den freien Raum links vom Start und rechts vom Ende der Siedlung so effektiv wie möglich zu nutzen. Diese Situation am Rand ist so speziell, das die Heuristik mit dem Schwerpunkt dafür nicht geeignet ist.

Als Heuristik haben wir das Fächern gewählt, weil sie wenig Rechenaufwand benötigt und Dreiecke damit sehr eng aneinander gelegt werden können. Mit dem Fächern können wir so viel Masse wie möglich „weg von der Siedlung“ in den freien Raum verschieben und trotzdem den Berührungspunkt mit der x-Achse möglichst „weit rein in die Siedlung“ bekommen. Weiter haben wir diese spezielle Heuristik nicht optimiert, weil sie für eine steigende Anzahl von Dreiecken immer unbedeutender wird, gegenüber dem immer größer werdenden Mittelstück der Siedlung.

Abbildung 2.12: Heuristik zum Starten und Beenden der Siedlung durch Fächern.

Starter: Den Starter bauen wir als allerersten Fächer. Hierfür haben wir noch alle Dreiecke zur Verfügung. Wir wählen von allen Dreiecken diejenigen mit dem größten Wert „ls_1“ (Abstand des Schwerpunktes zur spitzesten Ecke) und fächern sie aneinander, bis die Summe ihrer spitzesten Winkel 90 Grad übersteigt. Die Idee ist hier, dass wir eine möglichst senkrechte Wand haben wollen, an der dann die Heuristik mit dem Schwerpunkt ansetzt. Beträgt die Summe der spitzesten Winkel weniger als 180 Grad, dann genügt insgesamt ein Fächer und die Siedlung hat Distanz 0 (vgl. Beispiel 2).

Stopper: Der Stopper wird erst am Ende des Siedlungsbaus gebildet, wenn die meisten Dreiecke schon verbaut sind und wir auf diese nicht mehr zurückgreifen wollen. Für den Stopper bleiben also nur noch die wenigen noch nicht verbauten Dreiecke übrig. Man könnte theoretisch einen Stopper anlegen, sobald die beteiligten Dreiecke einen gemeinsamen spitzesten Winkel kleiner 180 Grad haben. Aber je nach Geometrie kann es besser sein, ein paar Dreiecke erst noch normal einzeln anzulegen und dann erst den Stopper mit einer reduzierter Anzahl an Dreiecken zu bilden.

Für die letzten Zyklen wird jeweils geprüft, was die Gesamtdistanz der Siedlung wäre, wenn man alle verbleibenden Dreiecke als Stopper anlegen würde. Dieser Wert wird jeweils gespeichert, der nächste Zyklus durchgeführt und wieder geprüft, was nun die Gesamtdistanz der Siedlung wäre, wenn alle verbliebenen Dreiecke als Stopper angelegt würden.

Wenn alle Dreiecke mittels der Heuristik mit dem Schwerpunkt angelegt wurden, geht man im Anlegen zurück und wählt denjenigen Stopper fürs endgültige Anlegen aus, bei dem die kleinste Gesamtdistanz für die Siedlung gemessen wurde.

2.1.8 Laufzeitanalyse

Die Laufzeit unseres Algorithmus ist $O(N^2)$ mit Anzahl N der Dreiecke.

Der Algorithmus besteht aus zwei Teilen, dem Vorberechnen und dem Anlegen.

Das Vorberechnen geht in linearer Zeit $O(N)$. Jedes Dreieck wird einzeln eingelesen, der Einlese-Vorgang braucht eine konstante Zeit je Dreieck.

Das Anlegen geht in quadratischer Zeit $O(N^2)$. Es müssen N Positionen der Siedlung gefüllt werden, und für jede Position werden im Durchschnitt $N/2$ freie Dreiecke geprüft; das ergibt $N \cdot N/2 = O(N^2)$ Anlegeversuche, bis die Siedlung fertig ist. Die Laufzeit der Erstellung einer Liste von Ecken der Siedlung kann hierbei vernachlässigt werden, da sie linear ist.

2.1.9 Optimierung durch Randomisierung

Bislang wurde die Siedlung deterministisch erstellt. Aber wir können das Ergebnis verbessern, indem wir einen Zufallsfaktor einbauen (sogenannte Randomisierung des Algorithmus[16]), um ein Dreieck z.B. zu 50 Prozent beim Anlegen zu ignorieren und zu überspringen. Dies ermöglicht es, mehrmals eine Siedlung zu generieren (ein „Run“) mit stets verschiedener Reihenfolge der Dreiecke. Wegen des Zufallsfaktors entstehen verschiedene Siedlungen verschiedener Längen, wobei einige davon durch Zufall kürzer als andere sind. Aus diesem Pool wählen wir die kürzeste Siedlung aus; diese Siedlung hat sich bei den Berechnungen meistens als etwa 10-15 Prozent kürzer erwiesen als die deterministisch erzeugte Siedlung.

Der Grund für diese Verbesserung liegt darin, dass die Heuristik mit dem Schwerpunkt wie ein gieriger Algorithmus[17] arbeitet. Sie achtet immer nur auf einen Zyklus und sucht dort das lokal beste Dreieck. Diese Wahl kann aber global ungünstig sein. Durch den Zufallsfaktor durchsucht („sampelt“) das Programm einen größeren Raum an Lösungen. Das´Programm muss lokal schlechte Entscheidungen treffen, die sich dann aber zufällig später also global gute Entscheidungen erweisen können.

[16] Randomisierter Algorithmus. Wikipedia, `https://de.wikipedia.org/wiki/Randomisierter_Algorithmus`

[17] Greedy-Algorithmus. Wikipedia, `https://de.wikipedia.org/wiki/Greedy-Algorithmus`

Die Laufzeit eines Runs bei Beispiel 5 konnten wir von anfänglichen ca. 25 ms auf ca. 1 ms verringern. Dies erfolgte durch ein frühzeitiges Verlassen von Schleifen, wenn klar war, dass man keinen neuen besten Abstand findet und deswegen Rotationen nur noch in Abständen von 9 Grad statt 1 Grad geprüft werden, um dann nur um das gefundene Minimum herum noch einmal mit 1 Grad Genauigkeit zu prüfen. Damit können ca. 1.000 Siedlungen pro Sekunde erzeugt werden. Es hat sich gezeigt, dass ca. 5-20 s eine geeignete Laufzeit ist, in der das Programm zufällig immer wieder neue bessere Siedlungen findet. Danach wartet man teils mehrere Minuten auf eine neue Verbesserung der Distanz oft um nur mehr Bruchteilen von Prozentpunkten. Je nach Anspruch an Zeit und Lösungsqualität kann man das Programm also kurz oder lange laufen lassen.

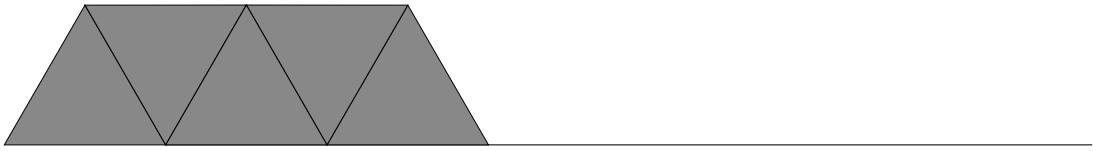
2.2 Beispiele

Für jedes der fünf vorgegebenen Beispiele sind im Folgenden eine oder mehrere Lösungen angegeben.

Beispiel	Distanz	Laufzeit
1	142,83 m	1,1 ms
2	0 m	0,1 ms
3	74,911 m	2,4 ms
4	224 m	0,050 s
	174 m	1 s
	173 m	2 s
	171,050 m	~2 h
5	724 m	0,150 s
	660 m	1 s
	650 m	5 s
	640 m	60 s
	630,741 m	~2 h

Bei den Beispielen 4 und 5 sind gemittelte Werte für Zeit und Distanz angegeben. Dort wird die Optimierung mit Zufallsfaktor benutzt, weshalb die Ergebnisse von Lauf zu Lauf schwanken. Deutlich zu erkennen ist eine drastische Zunahme der Laufzeit für die besten gefundenen Lösungen. Die letzten Prozent Optimierung dauern extrem lange und basieren zum Teil einfach auf Glück beim Zufallsfaktor. Dort stößt das Programm an seine Grenzen.

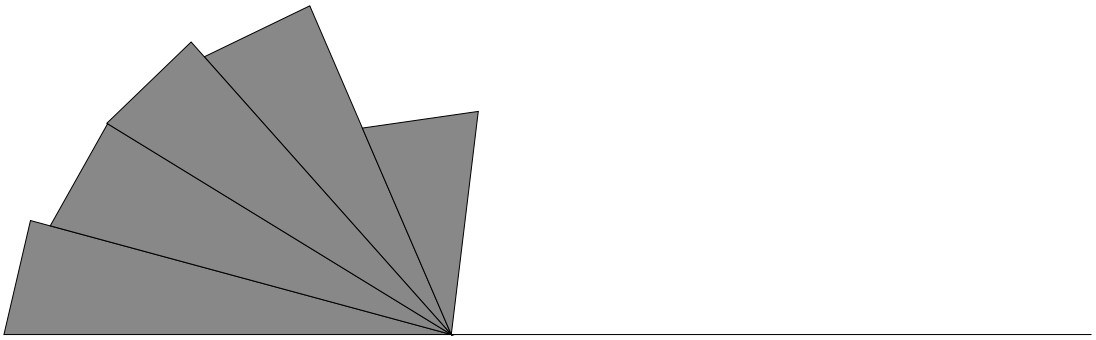
2.2.1 Beispiel 1



```
Länge: 142,83 m
Laufzeit: 1,1 ms
ID Reihenfolge: 0 1 2 3 4
Koordinaten (Format: ID x1 y1 x2 y2 x3 y3)
ID0 0.0 0 -71.47 123.71 -142.87 0.0
```

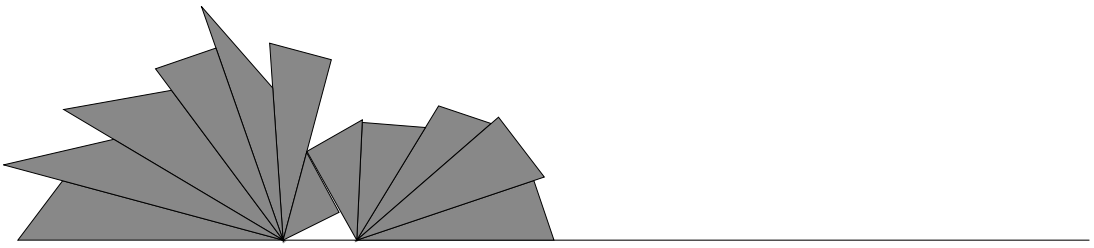
```
ID1 0.0 0 71.36 123.77 -71.47 123.71
ID2 142.83 0 285.7 0.0 214.31 123.71
ID3 142.83 0 214.31 123.71 71.47 123.77
ID4 142.83 0 71.41 123.74 0.0 0.0
```

2.2.2 Beispiel 2



```
Länge: 0 m
Laufzeit: 0,1 ms
ID Reihenfolge: 1 3 4 0 2
Koordinaten (Format: ID x1 y1 x2 y2 x3 y3)
ID0 0.0 0 -181.28 421.07 -316.33 355.8
ID1 0.0 0 -538.9 146.08 -572.94 0.0
ID2 0.0 0 34.6 285.85 -113.86 264.46
ID3 0.0 0 -440.16 270.02 -513.58 139.22
ID4 0.0 0 -333.15 374.71 -441.18 270.65
```

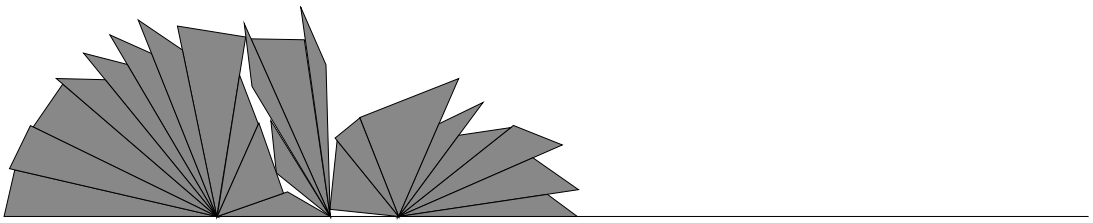
2.2.3 Beispiel 3



```
Länge: 74,911 m
Laufzeit: 2,4 ms
ID Reihenfolge: 3 11 1 10 8 4 7 9 6 2 5 0
Koordinaten (Format: ID x1 y1 x2 y2 x3 y3)
ID0 74.91 0 275.4 0.0 254.97 60.65
ID1 0.0 0 -113.53 152.43 -223.11 132.9
ID2 74.91 0 211.49 118.59 158.17 136.63
ID3 0.0 0 -224.22 60.44 -269.77 0.0
ID4 0.0 0 49.07 183.55 -13.71 200.34
ID5 74.91 0 265.6 64.23 219.0 125.1
ID6 74.91 0 144.76 114.63 80.47 119.75
```

```
ID7 0.0 0 56.87 28.38 23.91 89.26
ID8 0.0 0 -10.59 154.75 -83.08 237.63
ID9 74.91 0 80.59 122.36 24.13 90.27
ID10 0.0 0 -68.32 195.4 -129.78 174.24
ID11 0.0 0 -172.24 102.6 -284.23 76.62
```

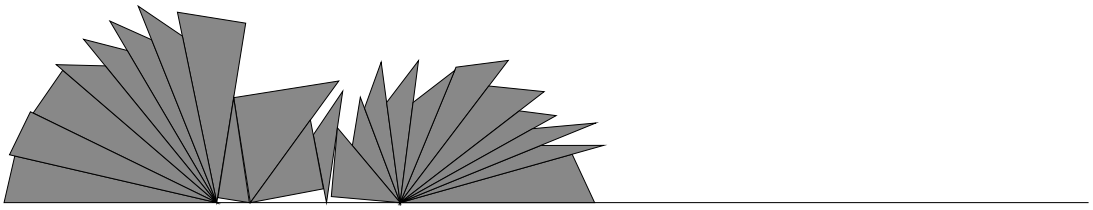
2.2.4 Beispiel 4 – Beste gefundene Siedlung (∼2 h Suche)



```
Länge: 171,050 m
Laufzeit: ∼2 h
ID Reihenfolge: 11 12 9 17 14 13 16 7 20 22 1 0 10 6 8 4 19 15 18 21 2 3 5
Koordinaten (Format: ID x1 y1 x2 y2 x3 y3)
ID0 106.57 0 50.88 90.13 56.06 40.9
ID1 106.41 0 66.53 23.12 0.46 0.0
ID2 171.05 0 324.98 67.3 279.0 85.62
ID3 171.05 0 340.18 25.19 297.29 55.19
ID4 171.05 0 113.18 71.42 106.4 6.78
ID5 171.05 0 339.05 0.0 310.92 20.83
ID6 106.59 0 82.38 166.25 32.18 167.12
ID7 0.0 0 27.17 168.82 -37.0 179.15
ID8 106.59 0 102.69 142.66 78.9 196.99
ID9 0.0 0 -144.89 124.52 -172.76 84.22
ID10 106.57 0 25.81 181.43 33.07 122.33
ID11 0.0 0 -190.0 43.84 -200.12 0.0
ID12 0.0 0 -175.28 85.44 -195.0 44.99
ID13 0.0 0 -61.26 153.2 -100.62 170.85
ID14 0.0 0 -84.24 143.03 -125.31 153.66
ID15 171.05 0 227.47 129.67 134.42 93.05
ID16 0.0 0 -32.36 156.69 -73.92 184.86
ID17 0.0 0 -104.91 128.64 -150.99 129.77
ID18 171.05 0 250.42 107.23 208.95 87.11
ID19 171.05 0 134.56 92.7 111.25 73.81
ID20 0.33 0 39.08 86.74 21.4 132.97
ID21 171.05 0 276.52 83.66 227.57 76.35
ID22 0.46 0 62.74 21.83 39.59 87.89
```

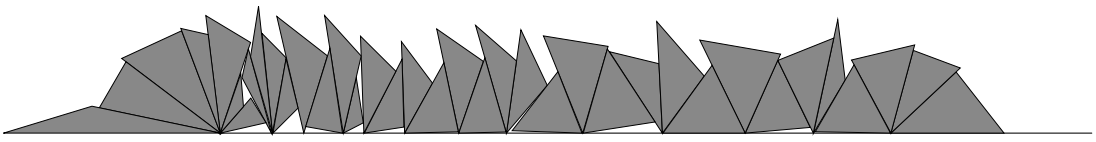
2.2.5 Beispiel 4 – Durchschnittliche Siedlung (2 s Suche)

```
Länge: 172,484 m
Laufzeit: 2 s
ID Reihenfolge: 11 12 9 17 14 13 16 7 19 15 22 0 4 1 18 20 21 2 3 5 10 8 6
```



```
Koordinaten (Format: ID x1 y1 x2 y2 x3 y3)
ID0 103.13 0 118.41 104.84 90.73 63.8
ID1 172.48 0 135.02 99.1 127.49 53.62
ID2 172.48 0 274.12 133.76 225.04 127.36
ID3 172.48 0 307.91 104.39 255.84 109.7
ID4 172.48 0 113.39 70.41 107.73 5.66
ID5 172.48 0 319.2 81.84 284.49 86.33
ID6 172.48 0 355.42 0.0 334.2 45.5
ID7 0.0 0 27.17 168.82 -37.0 179.15
ID8 172.48 0 363.97 53.88 304.67 53.81
ID9 0.0 0 -144.89 124.52 -172.76 84.22
ID10 172.48 0 356.42 74.88 297.12 69.52
ID11 0.0 0 -190.0 43.84 -200.12 0.0
ID12 0.0 0 -175.28 85.44 -195.0 44.99
ID13 0.0 0 -61.26 153.2 -100.62 170.85
ID14 0.0 0 -84.24 143.03 -125.31 153.66
ID15 31.53 0 114.66 114.41 15.89 98.76
ID16 0.0 0 -32.36 156.69 -73.92 184.86
ID17 0.0 0 -104.91 128.64 -150.99 129.77
ID18 172.48 0 154.57 132.2 138.89 88.86
ID19 30.62 0 15.85 98.52 0.99 4.69
ID20 172.48 0 189.71 133.52 159.73 94.14
ID21 172.48 0 223.83 124.44 184.64 94.21
ID22 31.53 0 100.32 12.97 88.08 77.83
```

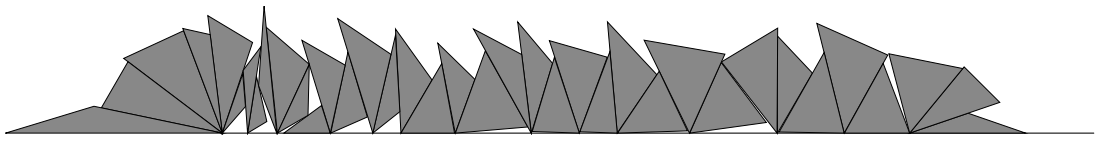
2.2.6 Beispiel 5 – Beste gefundene Siedlung (∼2 h Suche)



```
Länge: 630,741 m
Laufzeit: ∼2 h
ID Reihenfolge: 33 23 3 15 6 27 26 24 36 16 29 20 30 28 7 18 14 10 34 17 35 9 5
    32 19 21 4 22 31 25 8 11 0 13 12 2 1
Koordinaten (Format: ID x1 y1 x2 y2 x3 y3)
ID0 558.06 0 587.81 49.84 580.73 106.65
ID1 630.74 0 737.14 0.0 692.09 57.51
ID2 630.74 0 696.06 61.23 651.79 77.8
ID3 0.0 0 -36.4 96.3 -92.78 70.21
ID4 415.77 0 412.55 65.22 366.06 76.54
ID5 269.13 0 307.58 44.84 282.78 97.13
```

```
ID6 0.0 0 28.44 85.27 -13.65 110.52
ID7 115.75 0 134.52 9.62 127.84 52.38
ID8 493.88 0 554.94 5.34 524.38 68.52
ID9 269.13 0 278.55 67.82 240.08 101.3
ID10 173.38 0 199.59 46.93 170.37 86.06
ID11 557.8 0 577.06 89.39 524.46 68.36
ID12 630.74 0 653.18 82.92 593.71 68.77
ID13 630.38 0 596.14 64.24 559.68 1.23
ID14 135.49 0 172.7 5.66 170.95 69.59
ID15 0.0 0 -11.44 92.63 -37.2 98.42
ID16 49.1 0 62.01 70.69 39.24 93.82
ID17 224.37 0 247.43 67.47 203.56 97.94
ID18 135.06 0 165.11 58.13 131.87 91.18
ID19 340.89 0 364.77 81.6 303.91 91.52
ID20 78.47 0 100.73 72.69 53.14 109.71
ID21 341.35 0 408.96 10.48 364.05 79.13
ID22 415.97 0 455.21 53.56 410.47 104.93
ID23 0.0 0 -88.68 67.11 -113.78 23.76
ID24 48.13 0 26.46 79.29 20.4 52.16
ID25 493.63 0 526.97 74.27 450.97 87.46
ID26 0.08 0 42.95 9.74 28.68 32.9
ID27 0.0 0 21.76 24.94 19.23 55.85
ID28 115.66 0 132.53 72.53 98.14 110.62
ID29 49.54 0 73.04 23.5 61.88 69.98
ID30 114.93 0 103.03 79.73 80.34 6.09
ID31 493.61 0 462.55 63.71 416.45 0.0
ID32 340.21 0 317.58 57.7 274.51 2.29
ID33 0.0 0 -120.72 25.21 -203.79 0.0
ID34 173.57 0 223.85 1.15 210.29 66.24
ID35 224.37 0 268.75 1.02 248.79 70.92
ID36 49.1 0 35.91 119.27 28.44 72.06
```

2.2.7 Beispiel 5 – Durchschnittlich gefundene Lösung (20 s)



```
Länge: 646,525 m
Laufzeit: 20 s
ID Reihenfolge: 33 23 3 15 6 27 24 7 36 17 29 26 18 14 20 16 5 34 10 21 28 9 35
    11 30 22 32 25 13 19 4 12 1 8 31 2 0
Koordinaten (Format: ID x1 y1 x2 y2 x3 y3)
ID0 646.52 0 755.56 0.0 701.46 18.73
ID1 585.0 0 625.97 73.44 559.26 103.24
ID2 646.52 0 731.27 28.89 697.75 62.22
ID3 0.0 0 -36.4 96.3 -92.78 70.21
ID4 522.3 0 555.13 56.44 522.3 91.26
ID5 168.05 0 197.31 49.92 162.91 97.95
```

```
ID6 0.0 0 28.44 85.27 -13.65 110.52
ID7 23.77 0 41.83 10.9 32.18 53.09
ID8 646.52 0 621.59 65.6 585.23 0.0
ID9 290.71 0 310.63 65.51 277.87 104.59
ID10 219.06 0 237.61 50.45 202.63 84.53
ID11 335.9 0 362.11 71.4 307.64 86.96
ID12 584.88 0 566.1 75.81 523.76 1.06
ID13 439.63 0 511.74 9.99 469.24 66.5
ID14 101.94 0 136.42 15.1 118.18 76.4
ID15 0.0 0 -11.44 92.63 -37.2 98.42
ID16 141.57 0 166.99 20.17 162.79 91.92
ID17 51.78 0 81.77 64.69 41.32 99.58
ID18 101.55 0 115.53 63.92 74.87 87.25
ID19 521.81 0 522.34 98.71 469.46 66.99
ID20 141.54 0 158.67 74.06 108.61 107.68
ID21 219.24 0 287.41 5.74 247.4 77.36
ID22 371.46 0 408.8 54.9 362.3 104.67
ID23 0.0 0 -88.68 67.11 -113.78 23.76
ID24 23.77 0 36.23 81.25 19.65 58.93
ID25 439.39 0 472.73 74.27 396.73 87.46
ID26 101.46 0 93.34 25.96 57.86 0.0
ID27 0.0 0 21.76 24.94 19.23 55.85
ID28 290.59 0 281.63 73.92 236.3 97.95
ID29 51.78 0 80.57 16.61 81.82 64.4
ID30 371.46 0 364.57 76.74 336.35 1.22
ID31 646.52 0 696.51 60.71 627.01 74.64
ID32 371.77 0 437.47 2.12 411.01 58.18
ID33 0.0 0 -120.72 25.21 -203.79 0.0
ID34 218.4 0 206.35 65.38 168.11 0.0
ID35 335.9 0 312.33 71.2 291.54 1.54
ID36 51.25 0 39.34 119.41 34.12 44.62
```

2.2.8 Mögliche Erweiterungen

Im Folgenden sind mögliche Erweiterungen der Aufgabenstellung aufgeführt:

- Siedlungsbau auf beiden Seiten der Küstenstraße
- Verwendung von oberen und unteren Schranken (ggf. Auslassen einzelner Dreiecke)
- Nichtlinearer Verlauf der Küstenstraße
- Andere Formen (z.B. Polygone oder Ellipsen) statt Dreiecke
- Eingreifen des Benutzers zur interaktiven Verbesserung eines Ergebnisses

2.3 Bewertungskriterien

Einige der Bewertungskriterien werden hier näher erläutert:

- (Zu 1.1) Bei der Modellierung des Problems ist für die Anordnung der Dreiecke zu beachten, dass die Dreiecke verschoben, rotiert und gespiegelt werden können. Die Lösungen

müssen alle Dreiecke umfassen und dürfen außer Berührungen keine Überlappungen der Dreiecke enthalten.

- (Zu 1.2) Die Richtschnur ist, dass nur ein fortschrittliches, heuristisches Verfahren Null oder Pluspunkte erhält, wenn die Heuristik mit deutlichen Verbesserungen und sinnvollen Optimierungen ausgestattet ist, ansonsten gibt es:
  - 4 Minuspunkte, wenn nicht zumindest Beispiele 1 und 2 richtig gelöst wurden;
  - 3 Minuspunkte, wenn nur die beiden Beispiele 1 und 2 richtig gelöst wurden;
  - 2 Minuspunkte, wenn das Verfahren vornehmlich in einer simplen (z.B. fächerartigen) Anordnung der Dreiecke ohne weitere Verbesserungen besteht;
  - 1 Minuspunkt, wenn das Verfahren nicht nur in einer simplen Anordnung der Dreiecke besteht, sondern zumindest Verbesserungen unternimmt, die jedoch auch simpel sind bzw. nur eine geringe Verbesserung der Ergebnisqualität bewirken.
- (Zu 1.3) Die Laufzeit des Verfahrens sollte quadratisch oder besser sein.
- (Zu 1.4) Die in der Beispiellösung enthaltenen Ergebniswerte sind eine bedeutende Orientierungshilfe für die Qualität möglicher Ergebnisse für die vorgegebenen Beispiele. Abweichungen von den Orientierungswerten führen entsprechend zu Minus- oder Pluspunkten.
- (Zu 1.5) Die Grenzen der angewandten Heuristik sollten erkannt und diskutiert werden.
- (Zu 2.4) Da zur Lösung nur heuristische Verfahren sinnvoll sind, sollte dieser Sachverhalt näher erläutert werden.
- (Zu 3.3) Die Angabe einer Lösung für jedes Beispiel genügt. Jedes Ergebnis muss den Gesamtabstand angeben. Rundungen der Rechenergebnisse sind in Ordnung.
- (Zu 3.5) Eine visuelle Darstellung der Ergebnisse ist ein Muss, sie kann allerdings auch nur manuell generiert sein.

Aufgabe 3: Schach dem Wildschwein

Da die Aufgabenvarianten 3A und 3B algorithmisch sehr ähnlich sind, 3A aber kaum bearbeitet wurde, wird im Folgenden nur eine Lösung für die Variante 3B näher beschrieben.

3.1 Lösungsidee

3.1.1 Schach und Matt

Schach, auch das Spiel der Könige genannt, ist eines der populärsten und gleichzeitig komplexesten Brettspiele. Die Anzahl aller möglichen Stellungen liegt vermutlich im Bereich von 10^{43}. In dieser Aufgabe soll untersucht werden, ob mit drei Springern ein Schachmatt erzwungen werden kann. Schachmatt ist dabei die Situation, bei der das Feld des Königs von einer gegnerischen Figur bedroht wird und es keinen erlaubten Zug des Königs gibt, um die Bedrohung aufzuheben.

In der Schachwelt ist dies ein Problem der sogenannten elementaren Mattführung, die allerletzte Phase in Schachendspielen, wenn ein Spieler nicht mehr über genügend Figuren verfügt, um dem angestrebten Mattangriff des Gegners noch entscheidenden Widerstand entgegensetzen zu können.[18,19] Wenn auch etwas unwahrscheinlich, kann dieses Problem durch die Umwandlung eines weißen Bauern in einen dritten Springer real auftreten. Es ist bereits bekannt, dass Endspiele mit nur zwei weißen Springen nicht gewonnen werden können.[20,21]

Interessante Ressourcen zur Schachprogrammierung im Allgemeinen sind u.a. das Chess Programming Wiki[22] und, insbesondere für Szenarien wie dieser Aufgabe, die Endspieldatenbank von Shredder[23], in der für jede Stellung mit sechs Figuren oder weniger eine optimale Strategie eingetragen ist. Über die Notation von Schachstellungen informiert Wikipedia ausführlich.[24,25,26]

3.1.2 Grundlegende Lösungsstrategie

Eine mögliche Idee zur Lösung der Aufgabe ist, jede mögliche Ausgangsstellung zu prüfen, ob Weiß von dieser aus eine Siegstrategie gegen Schwarz hat. Dabei stößt man auf zwei Hauptprobleme, erstens, wie man mit der großen Menge an Brettstellungen umgeht, und zweitens, wie man Brettstellungen im Hinblick auf den Sieg von Weiß effizient evaluiert. Auf den allerersten Blick existieren immerhin $64 \cdot 63 \cdot 62 \cdot 61 \cdot 60 \approx 900$ Millionen Möglichkeiten, die fünf Figuren auf dem Schachbrett zu verteilen.

Jedoch ließ sich zunächst feststellen, dass ein erzwungenes Matt nur auf einem der 28 Randfelder statt auf allen 64 Felder des Bretts möglich ist: Auf jedem Feld außerhalb des Randes hat

[18] Elementare Mattführung. Wikipedia, https://de.wikipedia.org/wiki/Elementare_Mattführung
[19] Endspiel. Wikipedia, https://de.wikipedia.org/wiki/Endspiel_(Schach)
[20] Springerendspiel. Wikipedia, https://de.wikipedia.org/wiki/Springerendspiel
[21] Two knights endgame. Wikipedia, https://en.wikipedia.org/wiki/Two_knights_endgame
[22] Chess Programming Wiki, https://www.chessprogramming.org
[23] Shredder Endgame Database, https://www.shredderchess.com/online-chess/online-databases/endgame-database.html
[24] Schachnotation. Wikipedia, https://de.wikipedia.org/wiki/Schachnotation
[25] Forsyth-Edwards-Notation (FEN). Wikipedia, https://de.wikipedia.org/wiki/Forsyth-Edwards-Notation
[26] Portable Game Notation (PGN). Wikipedia, https://de.wikipedia.org/wiki/Portable_Game_Notation

der schwarze König 8 Nachbarfelder. Für ein Matt des Königs müsste Weiß diese 8 Felder und das eine Feld des Königs bedrohen, also 9 Felder insgesamt. Der weiße König kann davon maximal 3 Nachbarfelder bedrohen. Zudem befinden sich unter den von ihm unbedrohten Feldern stets zumindest drei weiße und drei schwarze Felder, da der weiße König, der keinesfalls neben dem schwarzen König stehen darf, nie ein 3x2-Rechteck von Feldern bedroht. Ein Springer aber kann von diesen 6 Feldern des Rechtecks nur zwei bedrohen. Auch sind alle Felder, die ein Springer bedroht, von der gleichen Feldfarbe, da ein Springer mit jedem seiner Züge die Farbe seines Feldes wechselt. Daher benötigt man für die drei schwarzen Felder mindestens zwei Springer und für die drei weißen Felder mindestens zwei andere Springer, insgesamt also vier Springer. Deswegen reichen die drei vorhandenen weißen Springer für ein Matt des schwarzen Königs auf einem Nichtrandfeld des Schachbretts nie aus.

Außerdem lohnt es sich, zunächst die Anzahl der zu betrachtenden Schachstellungen zu reduzieren. Die erste Beobachtung ist die Äquivalenz von Stellungen, wenn die drei Springer untereinander getauscht werden. Des Weiteren erkennt man, dass nur Stellungen, in denen der schwarze König in der oberen Hälfte des Bretts steht, betrachtet werden müssen, da das Brett ansonsten gespiegelt werden kann, um auf eine äquivalente Stellung zu kommen. Diese Überlegungen reduzieren die Anzahl der zu betrachtenden Stellungen um die Faktoren sechs bzw. zwei, also insgesamt auf ein Zwölftel.

Um zu ermitteln, welche Spielstellungen den Sieg für Weiß bedeuten, werden wir jedoch nicht vorwärts, sondern rückwärts vorgehen und eine retrograde Analyse (Retroanalyse)[27,28] durchführen (auch bekannt als „Backward Chaining/Induction/Reasoning/Search“[29,30]):

Wir markieren von allen Stellungen zunächst nur die, in denen Schwarz bereits im Schachmatt steht. Da laut Aufgabenstellung bereits bekannt ist, dass der schwarze König in den vier Eckfeldern des Schachbretts durch Weiß matt gesetzt werden kann, lassen wir außerdem die Stellungen bei der Markierung weg, in denen der schwarze König in einer Ecke steht.

Da Weiß seine eigenen Züge frei bestimmen kann, kann bei einem Rückwärtsschritt einer weißen Figur ihr Herkunftsfeld (das Feld, auf dem die Figur im normalen Spielverlauf vorher gestanden haben könnte) im Prinzip unter Beachtung der Schachregeln beliebig gewählt werden, solange der schwarze König in der Herkunftsstellung nicht im Schach steht. Bei einem Rückwärtsschritt des schwarzen Königs muss dagegen geprüft werden, ob von seiner potentiellen Herkunftsstellung wirklich alle seiner möglichen Züge zu Siegstellungen für Weiß führen; ist dies nicht so, kann die jeweilige Herkunftsstellung nicht als Siegstellung markiert werden.

Somit wird also durch alle verbliebenen Stellungen iteriert und jeweils geprüft:

- falls Weiß am Zug: Gibt es mindestens einen Zug, der zu einer markierten Stellung führt?
- falls Schwarz am Zug: Führen alle Züge zu einer markierten Stellung?

Wenn eine der beiden Fragen zu bejahen ist, wird auch diese weitere Stellung zusammen mit der Information, wer am Zug ist, markiert. Wenn für einen Zug keine neuen Stellungen markiert werden, kann die Suche abgebrochen werden. Alle bisher unmarkierten Stellungen sind dann Stellungen, in denen Weiß kein Matt erzwingen kann.

[27] Retrograde Analyse. Wikipedia, `https://de.wikipedia.org/wiki/Retrograde_Analyse`

[28] Retrograde Analysis. Chess Programming Wiki, `https://www.chessprogramming.org/Retrograde_Analysis`

[29] Backward chaining. Wikipedia, `https://en.wikipedia.org/wiki/Backward_chaining`

[30] Backward induction. Wikipedia, `https://en.wikipedia.org/wiki/Backward_induction`

3.1.3 Binäre Indizierung des Schachbretts

Aufgrund der sehr hohen Anzahl von möglichen Spielstellungen bietet es sich an, auf Ganzzahlen als Bitfelder zu operieren, um die Ergebnisse des obigen Markierungsalgorithmus effizient zu berechnen und platzsparend zu speichern. So werden nun logische Operationen auf Bits kurz erklärt, da sie im Folgenden genutzt werden.

Bekanntlich werden Ganzzahlen („Integers“) vom Computer als Binärzahlen gespeichert. Daher sind verschiedene Operationen auf dieser Repräsentation möglich, die aus technischen Gründen sehr schnell durchgeführt werden können. Insbesondere gilt dies für das logische UND („AND“), das logische ODER („OR“) und das exklusive Oder, auch XOR („eXclusive OR“) genannt. Dabei ist das Ergebnis von einem AND von zwei Bits genau dann 1, wenn beide Bits 1 sind, von einem OR 1, wenn mindestens ein Bit 1 ist, und von XOR genau dann 1, wenn nur eines der Bits 1 ist, aber nicht beide. Werden zum Beispiel zwei Integer-Zahlen verundet, wird die AND-Operation bitweise ausgeführt und ergibt eine Zahl, deren Bitdarstellung an jeder Stelle jeweils das Ergebnis der Verundung der zwei Bits an der selben Stelle der verundeten Zahlen ist. Beispiele mit Zahlen aus vier Bits:

- $1010 \text{ AND } 1001 = 1000$
- $1010 \text{ OR } 1001 = 1011$
- $1010 \text{ XOR } 1001 = 0011$

Es gibt viele Möglichkeiten, ein Schachbrett als Datenstruktur zu repräsentieren.[31,32] So gibt es brettzentrische Repräsentationen wie ein 8×8-Feld, wobei jede Zelle die Figur enthält, die das Feld besetzt, und figurenzentrische Repräsentationen, bei der für jede Figur der Spieler die jeweilige Position gespeichert wird. Insbesondere bietet sich ein sogenanntes Bitboard oder ein Bit-Array an, um eine Schachstellung zu speichern.

In dieser Aufgabe wählen wir aufgrund der geringen Figurenzahl eine figurenzentrische Darstellung und nutzen ein sogenanntes 0x88-Brett, benannt nach der hexadezimalen Konstante 0x88, die noch in der Zuggenerierung eine wichtige Rolle spielen wird.[33]Dabei speichern wir für jede Figur ihre Position zwischen 0 und 63 sowie ihre äquivalente Position auf dem 0x88-Brett. Letzteres ist praktisch ein 16×8-Brett, dargestellt durch ein eindimensionales Array, wobei die Hälfte der Felder ungültige Positionen umfassen. Die spezielle Darstellung sorgt dafür, dass man einfach und effizient prüfen kann, ob eine Position auf dem Brett liegt oder nicht. Dabei wird die entsprechende Position einfach mit 0x88 verundet, beim Ergebnis 0 ist es eine gültige Position.

In dem von uns genutzten Bit-Array wird eine Stellung wie folgt kodiert: Für jede der Figuren werden je sechs Bits benutzt, außer für den schwarzen König, da für ihn bereits fünf Bits ausreichen. Damit werden die Positionen der Figuren auf dem Brett indiziert, wobei der Index 0 in der unteren linken Ecke a1 des Schachbretts ist und der Index 63 in der oberen rechten Ecke h8. Für den schwarzen König genügt, wie oben erklärt, die obere Hälfte des Brettes. Somit werden $4 \cdot 6 + 5 = 29$ Bits benötigt, um eine Brettstellung eindeutig zu beschreiben; wenn man möchte, kann man noch ein Extra-Bit dafür verwenden, welcher Spieler am Zug ist. Zudem können verschiedene Bits des Array-Eintrages markiert werden, um zu bestimmen, ob Weiß gewinnt,

[31] Board Representation. Chess Programming Wiki, `https://www.chessprogramming.org/Board_Representation`

[32] Board Representation. Wikipedia, `https://en.wikipedia.org/wiki/Board_representation_(chess)`

[33] Schachprogramm – 0x88-Darstellung. Wikipedia, `https://de.wikipedia.org/wiki/Schachprogramm#0%C3%9788-Darstellung`

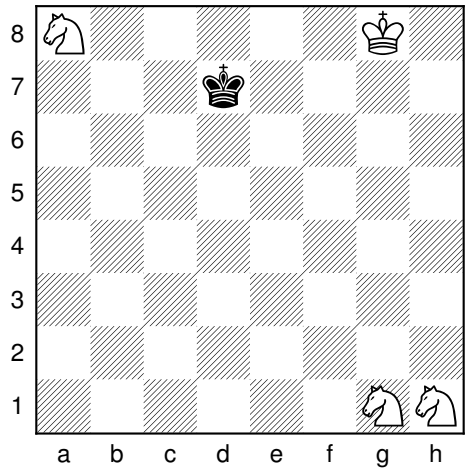


Abbildung 3.1: Beispielstellung für die Indizierung des Schachbretts.

wenn entweder Weiß oder Schwarz am Zug ist. (Den Inhalt dieser zusätzlichen Bits würde man durch eine Verundung mit einer entsprechenden Konstante ermitteln.) Auch wenn hier jeweils sechs Bit für die Springer genutzt werden, genügt es beim Iterieren über ihre Positionen, nur die zu betrachten, bei der sie aufsteigend sortiert sind.

Als Beispiel für einen Positionsindex wird hier der Index vom Schachbrett in Abbildung 3.1 angegeben. Der schwarze König steht auf d7, was Position 51 (binär: 110011) entspricht. Der weiße König ist auf Position 62 (binär: 111110), und die Springer stehen auf 6, 7 und 56 (binär: 000110, 000111, 111000). Ein möglicher Gesamtindex (die Reihenfolge der einzelnen Indizes ist frei wählbar, solange man konsistent bleibt) ist dann 000110000111111000111110110011 bzw. 102731699 als Dezimalzahl.

3.1.4 Zuggenerierung und Matt-Prüfung

Für die Zuggenerierung werden für jede Figur die möglichen Zugoffsets gespeichert, d.h. die Positionsdifferenzen, um die möglichen Zielfelder zu erreichen. Hierbei ist der Vorteil der Verwendung eines ein- statt zweidimensionalen Index, dass nur eine Addition erforderlich ist. So sind beispielsweise $\{-1,-9,-8,-7,1,7,8,9\}$ im Prinzip alle Zugmöglichkeiten eines Königs im Uhrzeigersinn: -1 ist dabei der Zug ein Feld nach links, -9 der Zug ein Feld diagonal nach links oben, …, 8 der Zug ein Feld nach unten und 9 der Zug ein Feld diagonal nach links unten.

Matt liegt vor, wenn Schwarz keine Zugmöglichkeiten mehr auf eine Position hat, wo der König nicht bedroht wird. Da Schwarz in dieser Aufgabe nur den König besitzt, muss nur für die aktuelle Position des Königs und jede seiner Zugmöglichkeiten geprüft werden, ob auch die neue Positionen bedroht werden. Dies wird bewerkstelligt, indem man den Index des schwarzen Königs vom Index jeder weißen Figur subtrahiert und prüft, ob die sich ergebende Differenz gleich einer der möglichen Zugoffsets des schwarzen Königs ist. Hierbei sind die Ränder des Schachbretts zu beachten.

Auch markiert man, ob der schwarze König einen Springer schlagen kann, und geht bei der Suche ähnlich wie bei der Matt-Suche vor, nur mit dem Unterschied, dass man die Stellungen markiert, bei der ein schwarzer Zug dazu führt, dass ein Springer geschlagen wird; wenn Weiß

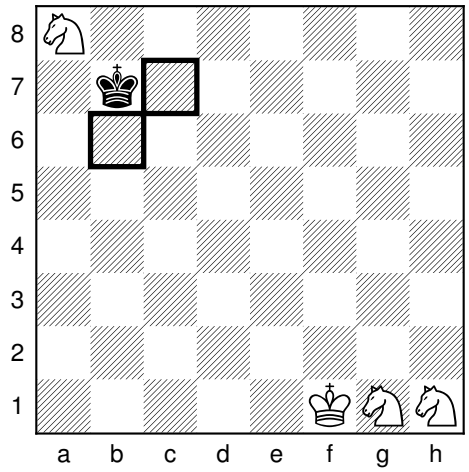


Abbildung 3.2: Weiß kann das Schlagen des Springers auf a8 nicht verhindern, da sowohl b6 als auch c7 vom schwarzen König bedroht werden.

am Zug ist, markiert man all die Züge, die dazu führen. Ein Beispiel dafür zeigt Abbildung 3.2. Weiß kann keinen Zug machen, der nicht zu einer Stellung führt, die bereits markiert wurde (Schwarz kann in einem Zug den weißen Springer schlagen).

3.2 Ergebnisse

Für jedes der fünf vorgegebenen Beispiele ist im Folgenden nur jeweils eine Lösung als Ergebnis angegeben, da es meist viele verschiedene Lösungen gibt. Die Anzahl der Züge ist in der jeweiligen Zugfolge enthalten.

3.2.1 Beispiel 1: schach1.txt

```
Ergebnis: Matt Schwarz
Anfangsstellung: 5N1k/3N4/8/5N2/8/8/4K3/8 w
Zugfolge (25 Halbzüge):
 1. f8-e6    2. h8-g8    3. d7-e5    4. g8-h8    5. f5-e7    6. h8-h7
 7. e5-g6    8. h7-h6    9. g6-f8   10. h6-h5   11. e7-f5   12. h5-g4
13. f5-g7   14. g4-g3   15. f8-d7   16. g3-g4   17. d7-e5   18. g4-g3
19. g7-f5   20. g3-g2   21. e6-f4   22. g2-h2   23. e5-f3   24. h2-h1
25. f5-g3
Endstellung: 8/8/8/8/5N2/5NN1/4K3/7k b
```

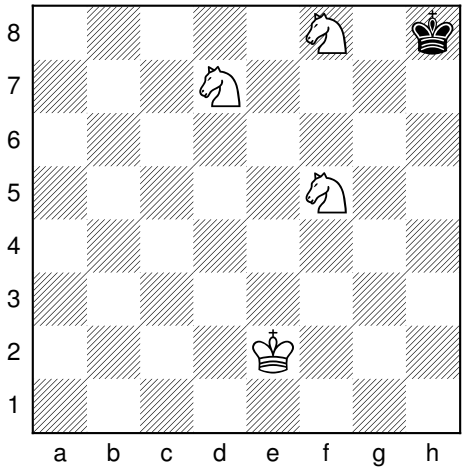


Abbildung 3.3: Beispiel 1 (Anfangsstellung)

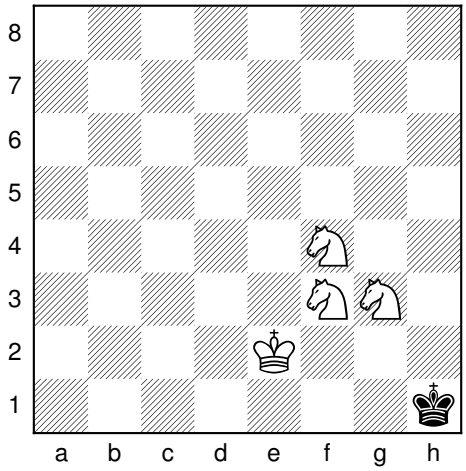


Abbildung 3.4: Beispiel 1 (Endstellung)

3.2.2 Beispiel 2: schach2.txt

```
Ergebnis: Matt Schwarz
Anfangsstellung: 8/2N5/8/8/k7/3N4/1K2N3/8 w
Zugfolge (9 Halbzüge):
 1. e2-c3   2. a4-a5   3. c3-d5   4. a5-a4   5. d3-b4   6. a4-a5
 7. b4-c6   8. a5-a4   9. d5-c3
Endstellung: 8/2N5/2N5/8/k7/2N5/1K6/8 b
```

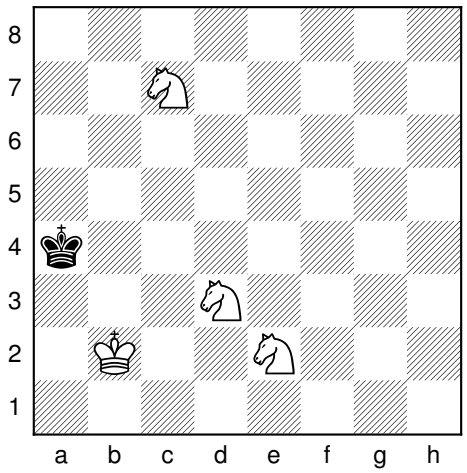


Abbildung 3.5: Beispiel 2 (Anfangsstellung)

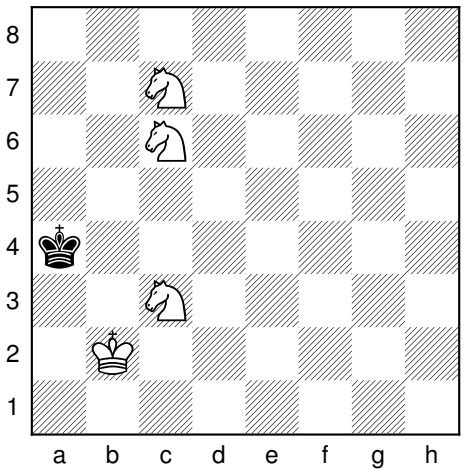


Abbildung 3.6: Beispiel 2 (Endstellung)

3.2.3 Beispiel 3: schach3.txt

```
Ergebnis: Matt Schwarz
Anfangsstellung: 1N6/7K/8/8/5k2/8/NN6/8 w
Zugfolge (47 Halbzüge):
 1. h7-g6    2. f4-f3    3. b2-d3    4. f3-g4    5. b8-c6    6. g4-f3
 7. c6-e5    8. f3-g2    9. d3-f4   10. g2-g3   11. g6-g5   12. g3-f2
13. g5-g4   14. f2-f1   15. e5-f3   16. f1-f2   17. a2-c3   18. f2-f1
19. c3-b1   20. f1-f2   21. b1-d2   22. f2-e3   23. f4-d5   24. e3-e2
25. g4-g3   26. e2-d3   27. g3-f2   28. d3-c2   29. f2-e2   30. c2-c1
31. f3-d4   32. c1-b2   33. d2-b3   34. b2-b1   35. d5-b4   36. b1-b2
37. e2-d3   38. b2-b1   39. b4-c6   40. b1-a2   41. d3-c2   42. a2-a3
43. c2-b1   44. a3-a4   45. b3-c5   46. a4-a3   47. d4-c2
Endstellung: 8/8/2N5/2N5/8/k7/2N5/1K6 b
```

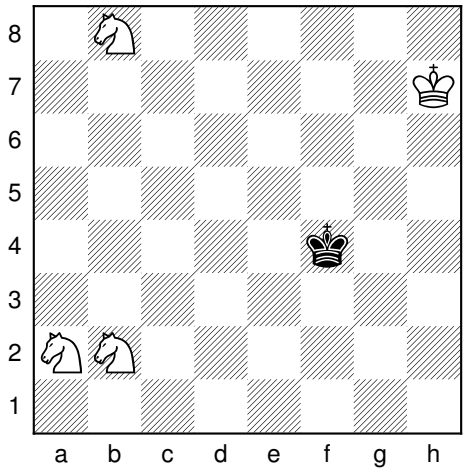


Abbildung 3.7: Beispiel 3 (Anfangsstellung)

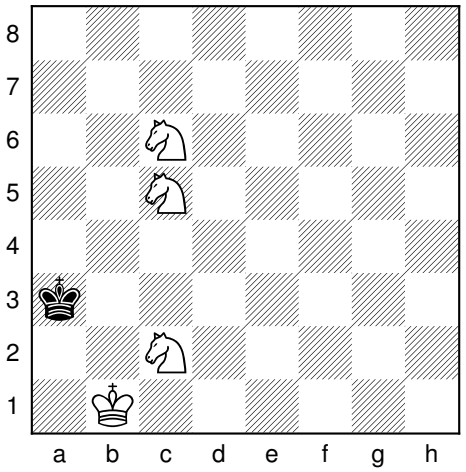


Abbildung 3.8: Beispiel 3 (Endstellung)

3.2.4 Beispiel 4: schach4.txt

```
Ergebnis: Matt Schwarz
Anfangsstellung: 8/1N6/6N1/8/8/8/6N1/1K3k2 w
Zugfolge (39 Halbzüge):
 1. g2-e3    2. f1-f2    3. e3-f5    4. f2-g2    5. g6-f4    6. g2-h2
 7. b7-d6    8. h2-h1    9. f5-e3   10. h1-h2   11. d6-e4   12. h2-h1
13. e3-g4   14. h1-g1   15. e4-f2   16. g1-f1   17. f2-h3   18. f1-e1
19. b1-a1   20. e1-d2   21. a1-b2   22. d2-e1   23. g4-e3   24. e1-d2
25. e3-g2   26. d2-d1   27. h3-f2   28. d1-d2   29. f2-e4   30. d2-d1
31. b2-a3   32. d1-c2   33. g2-e3   34. c2-c1   35. f4-d3   36. c1-b1
37. e4-c3   38. b1-a1   39. e3-c2
Endstellung: 8/8/8/8/8/K1NN4/2N5/k7 b
```

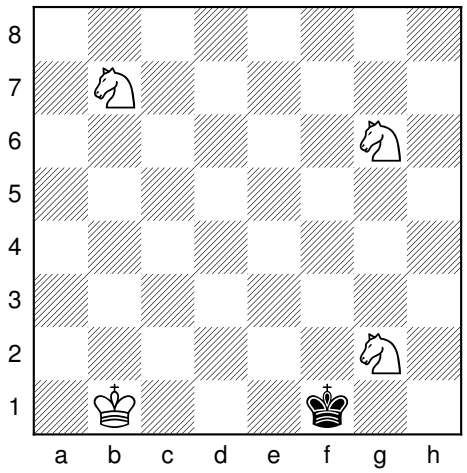


Abbildung 3.9: Beispiel 4 (Anfangsstellung)

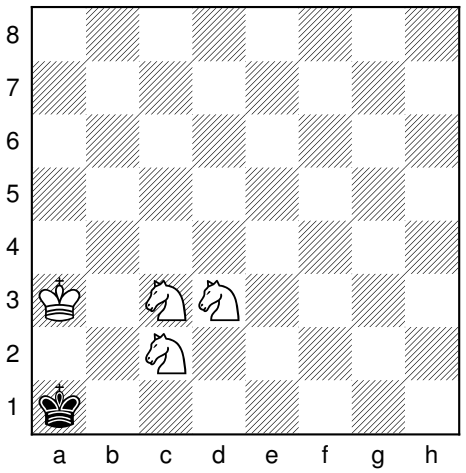


Abbildung 3.10: Beispiel 4 (Endstellung)

3.2.5 Beispiel 5: schach5.txt

```
1  Ergebnis: nichts gefunden
2  Anfangsstellung: 4N3/4k3/8/4N3/8/8/7N/3K4 w
```

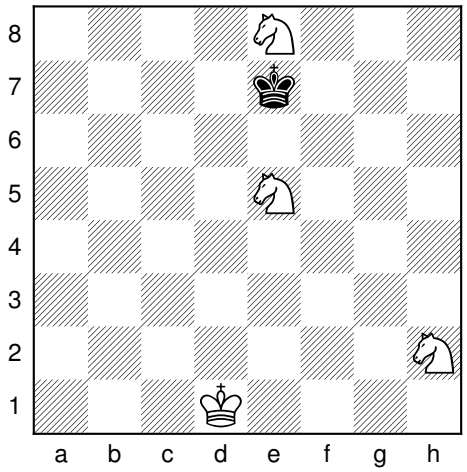


Abbildung 3.11: Beispiel 5 (Anfangsstellung)

Begründung für dieses Ergebnis: In diesem Beispiel kann Weiß keinen Sieg erzwingen, da der schwarze König bereits den weißen Springer auf e8 bedroht. Da Weiß am Zug ist, kann sein Springer nur auf c7 oder g7, aber nicht auf d6 oder f6 bewegt werden, ohne vom schwarzen König weiter bedroht und im nächsten Zug von ihm geschlagen zu werden. Nachdem aber Weiß seinen Springer auf c7 bzw. g7 zog, kann Schwarz auf d6 bzw. f6 ziehen und damit beide weiße Springer gleichzeitig bedrohen. Da Weiß im nächsten Zug nur einen der beiden Springer wegziehen und nicht beide Springer decken kann, wird der schwarze König einen der beiden weißen Springer schlagen können; aufgrund des Verlusts eines seiner drei Springer kann dann bekanntlich ein Sieg von Weiß nicht mehr erzwungen werden.

Lösung des Schachproblems

Es zeigt sich nach dem Ausrechnen aller Möglichkeiten, dass Weiß auf allen Randfeldern des Schachbretts Matt für Schwarz erzwingen kann.

Weitere Einsichten

Im Falle einer Stellung wie in Abbildung 3.12 hat Weiß immer eine Siegstrategie, außer dann, wenn Weiß ein Patt nicht vermeiden kann. Ein Beispiel für eine Stellung, in der Weiß in Aufgabenvariante 3A gewinnen kann, aber nicht in Variante 3B, ist in Abbildung 3.13 zu sehen. Stellungen dieser Art, die so direkt ins Patt münden, sind die einzigen, die sich zwischen den Aufgabenvarianten unterscheiden.

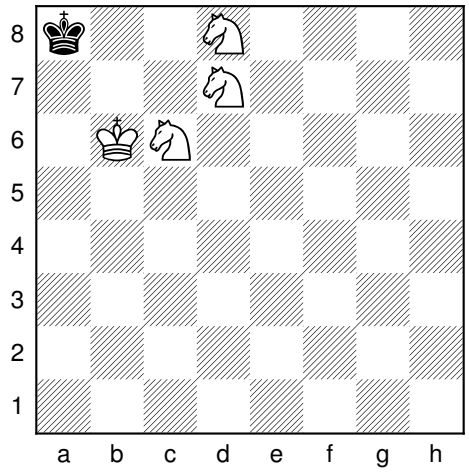


Abbildung 3.12: Kein Zug ist möglich, der das Patt bricht.

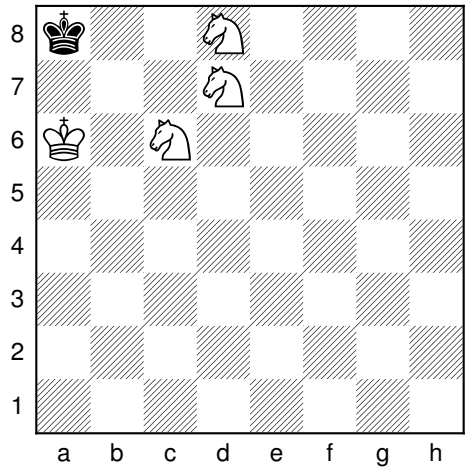


Abbildung 3.13: Normalerweise wäre Sd7-b6 ein Siegeszug für Weiß. Da aber Schwarz nicht in der Ecke mattgesetzt werden soll, bleiben Weiß nur Pattzüge übrig.

3.3 Alternative Verfahren und mögliche Erweiterungen

Grundsätzlich lassen sich die Stellungen entweder durch Vorwärtszüge oder Rückwärtszüge erreichen. Alternative Lösungsverfahren könnten auch andere Repräsentationen von Schachbrettern und Figurenstellungen involvieren, auch unterschiedliche Markierungsalgorithmen sind möglich.

Im Folgenden sind außerdem mögliche Erweiterungen der Aufgabenstellung aufgeführt:

- Berücksichtigung anderer Figuren.
- Erhöhung der Anzahl der Figuren.
- Änderung der Spielregeln.

3.4 Bewertungskriterien

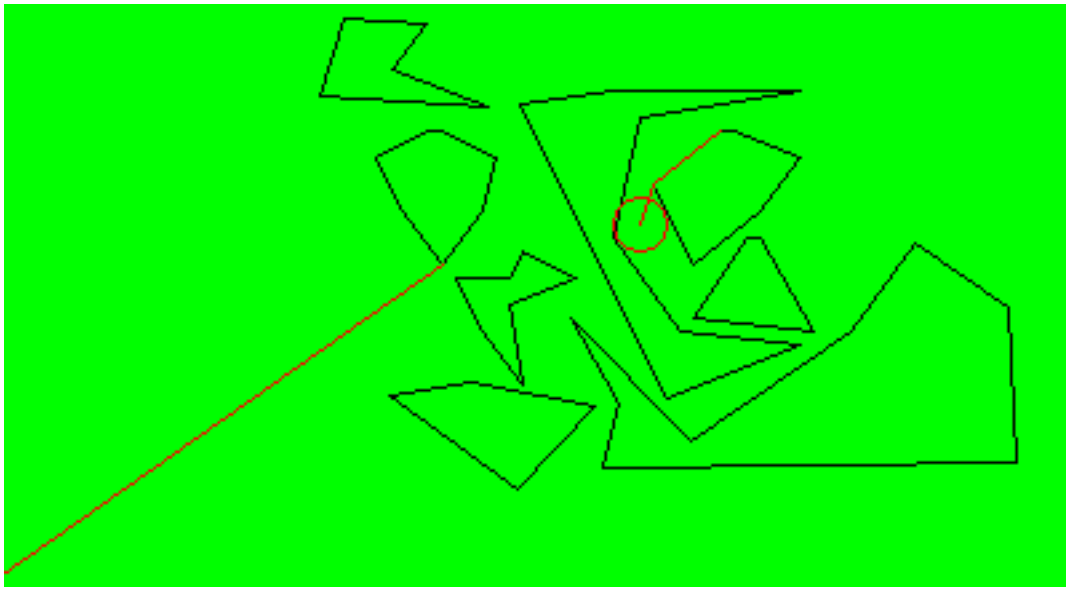
Einige der Bewertungskriterien werden hier näher erläutert:

- (Zu 1.1) Zur adäquaten Modellierung des Problems gehören die Beachtung der Schachregeln und auch der Möglichkeit, dass der schwarze König unter Umständen einen Springer schlägt (vgl. Beispiel 5).
- (Zu 1.2) Es ist damit zu rechnen, dass die Laufzeit eher lang ist; mehrere Stunden sind in Ordnung, aber nicht Tage.
- (Zu 1.3) Das angewandte Verfahren sollte stets den Sieg erzwingen können, wenn er möglich ist.
- (Zu 1.4) Nicht nur die vorgegebenen Beispiele sollten gelöst werden, sondern es sollte auch durch geeignete Überlegungen sowie insbesondere durch Ausrechnen ermittelt werden, dass nur auf den Randfeldern des Bretts ein Sieg erzwungen werden kann.
- (Zu 1.5) Die Stellungen sollten intern kompakt gespeichert werden.
- (Zu 1.6) Symmetrien zum Beispiel des Bretts sollten zur Reduktion des Rechenaufwands berücksichtigt werden.
- (Zu 2.4) Das Verfahren sollte effizient implementiert sein. Hohe RAM-Ansprüche des Programms sollten in der Dokumentation besprochen werden und sich in Grenzen halten (nicht mehr als 32 GB RAM), außer sie sind für sehr hohe Performanz (und nicht nur relativ schlechte Implementierung) gerechtfertigt.
- (Zu 3.5) Die Ergebnisse sollten Zwischenschritte enthalten.

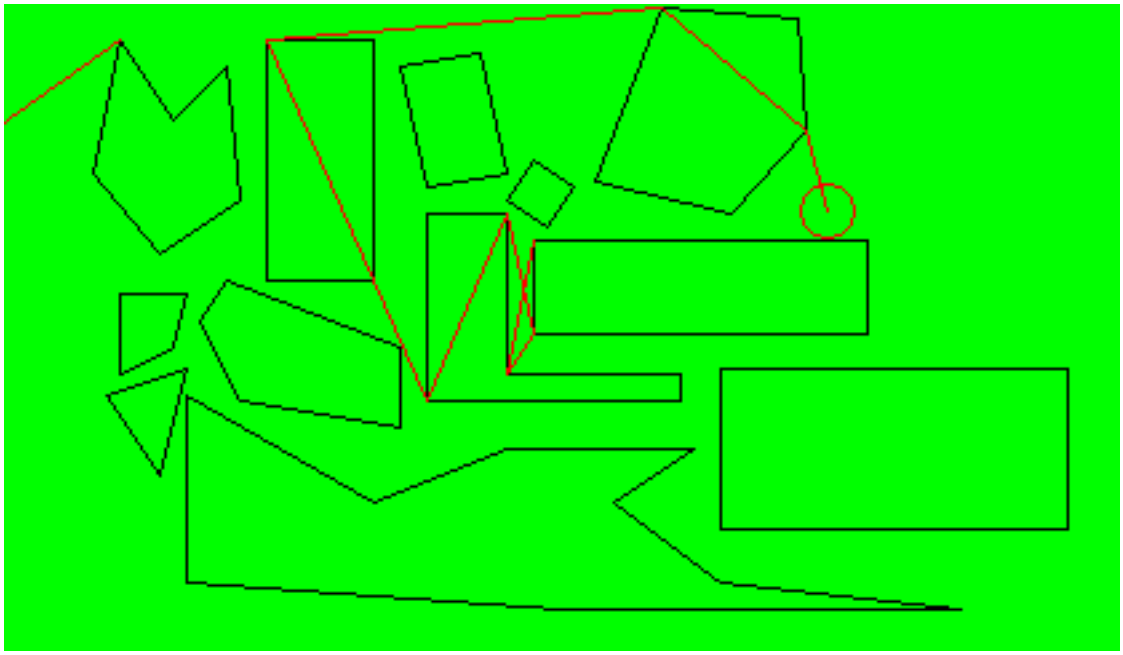
Anhang: Perlen der Informatik aus den Einsendungen

Aufgabe 1: Lisas Weg

- „Lisa rennt mittig durch die Polygone anstatt sie zu umgehen.“
- „Nun soll ein möglichst kurzer Weg [...] gefunden werden, wobei dieser nicht durch Hindernisse führen soll.“ Fußnote dazu: „Zumindest nicht, wenn Lisa als Zielposition nicht das Krankenhaus erreichen will.“
- „Eventuell ist Lisa gerade 18 geworden und möchte nun mit ihrem Auto zur Bushaltestelle fahren, auch wenn man über die ökologische Sinnhaftigkeit dieses Vorhabens streiten kann.“ Fußnote dazu: „Hier Fridays-for-Future-Demonstration einfügen.“
- „Die Angabe für die Dauer in den Ausgaben ist als Dezimalzahl in Minuten angegeben. Der Nachkommaanteil entspricht also nicht der Sekundenanzahl.“
- „Ich habe das Programm über die Nacht laufen lassen und es war immer noch am Ausprobieren. Das Maximum der Wegkombinationen ist bei 50 Punkten die Fakultät von 50 und die ist $3{,}04140932 \times 10^{64}$. [...] Ich musste Lösungsideen finden, wie das Programm schneller werden kann.“
- „Die Strecke, die Lisa zu laufen hat, beträgt 95 Meter. Sie braucht für diese Strecke bei 15 km/h 22 Sekunden. Indexfehler sorgt für Teleportationsfähigkeiten.“

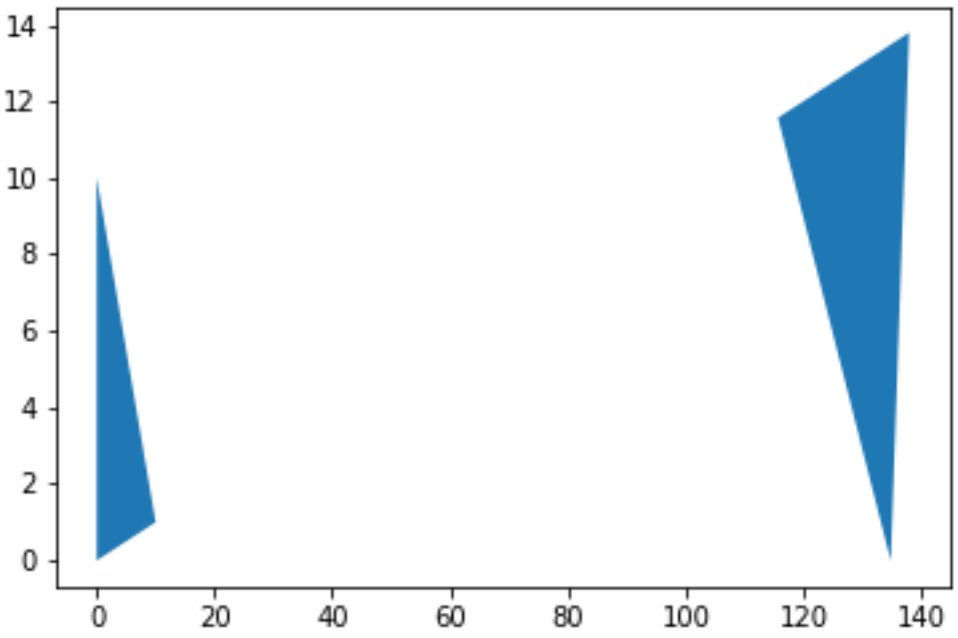


- „Physikalische Gesetze missachtet Lisa leider vehement. Gut, dass sie in den ersten beiden Stunden Physik hat. Schlecht, dass sie den Bus wohl nicht erreicht.“



Aufgabe 2: Dreiecksbeziehungen

- „Winkel, die einen Wert kleiner als -90° oder größer 90° haben, widersprechen der Aufgabenstellung und sind somit verboten.“
- „Nur wenn sie [die Konstante k] einen Wert zwischen 0 und 1 aufweist, besteht Handlungsbedarf.“
- „Das rechte Dreieck liegt nicht gut am linken an.“



Aufgabe 3: Schach dem Wildschwein

- „Da es pro Datenbank ca. [...] Gewinnstellungen gibt, beträgt die Geschwindigkeit des Programms somit [....] 36 Megastellungen pro Minute = 600 Kilostellungen pro Sekunde. Das bedeutet, dass die Überprüfung der Nachbarn einer einzelnen Stellung eine sechhunderttausendstel Sekunde, also ca. 1667 ns benötigt.“