1 Problembeschreibung und -analyse

1.1 Problembeschreibung

Die Hauptschwierigkeit des Problems liegt daran, dass es sich eigentlich um eine Kombination aus (mindestens) zwei Problemen handelen. Jedes Problem für sich ist gut beschrieben und gelöst worden, doch für alles gleichzeitig nicht.

1. Graph coloring - Visit stuff
2. Evacuate

1.2 Annahmen über das Problem

- Roboter sind punktförmig
- Können in der Bewegung saugen
- Keine Beschränkung, wie viele Roboter sich in einem Punkt befinden
- Kommunikation ist instantan und ohne Berechnungszeit (Senden wie Empfangen)
- Kommunikation zwischen Robotern kann alles sein
- Roboter haben unbeschränkt viel Speicher

1.3 Disclaimer

Einschränkungen nicht simuliert, aber deren Auswirkungen

2 Vorgehensmodell

- Keep it simple
- Start with working prototype before you do srs stuff
- Getrennt marschieren, zusammen kämpfen (self contained subprograms, assemble at the end, bottom to top)
- ipython notebook
- using a lib before reinventing the wheel
- Optimize when needed, first correct and working, then fast
- 90 % thinking, 10 % coding
- Abstract away decisions not made or which might change (geometry, polygon, agent)

2.1 Design-Ziele

2.2 Nicht-Ziele

Schöne UI, nur usable

3 Problembetrachtung

Das Problem kann unter vielen verschiedenen betrachtet werden. Je nachdem, in welcher Domäne der Informatik es eingeordnet wird, gibt es unterschiedliche Lösungsansätze.

In diesem Abschnitt wird kurz beschrieben, welche Ansätze erdacht wurden, welches deren Vor- und Nachteile sind, und für welche schließlich ausgewählt wurden.

3.1 Graphenproblem

Suche

3.2 Optimierungsproblem

3.3 Maschinenlernen

3.4 Künstliche Intelligenz

4 Umgebung

Die Modellierung des Meeresbodens hat zum Ziel, die folgenden Fragen zu beantworten oder einen guten Kompromiss zu finden:

1. Wie lassen sich eine begrenzte Anzahl an Kreisen auf einer unendlichen Fläche anorden, sodass die nicht von Kreisen bedeckte Fläche minimal ist (vergleichbar mit dem Ausstechen von Kreisen aus Keksteig)?
2. Wie lässt sich der Meeresboden unterteilen, sodass eine Simulation möglichst einfach wird?

Geplant ist, dass die Missionsdauer in Zeitschritt von 1s aufgeteilt wird. In jedem Zeitschritt bewegen sich die Roboter einen geometrischen Schritt auf dem Meeresboden weiter und säubern ihn dabei von Manganknollen. Der Vorteil darin besteht, dass die Optimierung der Ausbeute darauf reduziert wird, den nächsten Schritt möglichst geeignet auszuwählen.

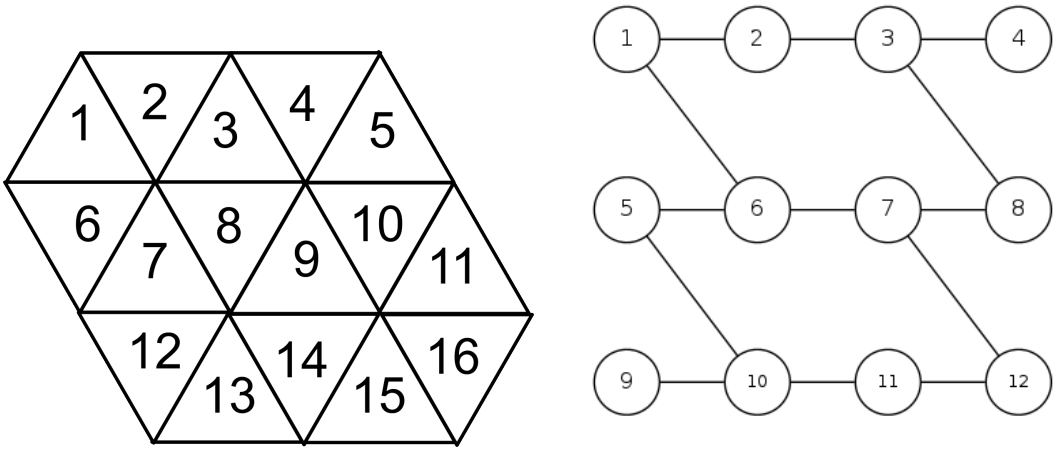
Dazu werden unendlichen Weiten des pazifischen Meeresgrundes in Abschnitte in Form von Polygonen eingeteilt (parkettiert), damit der Wertebereich der nächsten möglichen Schritte diskretisiert wird.

Die Wahl, mit welchem Polygon gearbeitet wird, hat direkte Auswirkungen auf Genauigkeit und Leistungsfähigkeit der Simulation, wie im Folgenden gezeigt wird.

4.1 Konkrete Modellierung des Meeresbodens

Der Meeresboden kann dann als Graph gesehen werden, bei denen Knoten als Zellen gesehen werden und geometrisch einem Polygon entsprechen. Angeordnet werden diese so, dass zwischen den Mittelpunkten der Innenkreise zweier benachbarter Zellen exakt 1m Abstand ist. Dies hat den Grund, dass ein Roboter in jedem Zeitschritt in die Mitte der nächste Zelle wechseln kann.

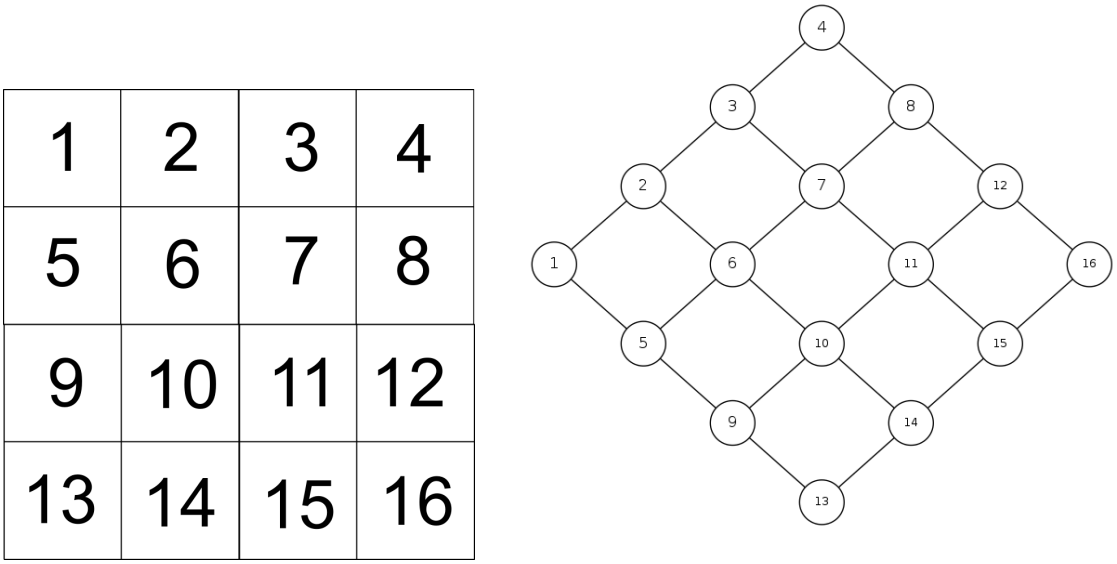
Der Zusammenhang zwischen realer Umgebung und Modellierung ist in Fig. 1 - 10 visualisiert.



(a) (b)

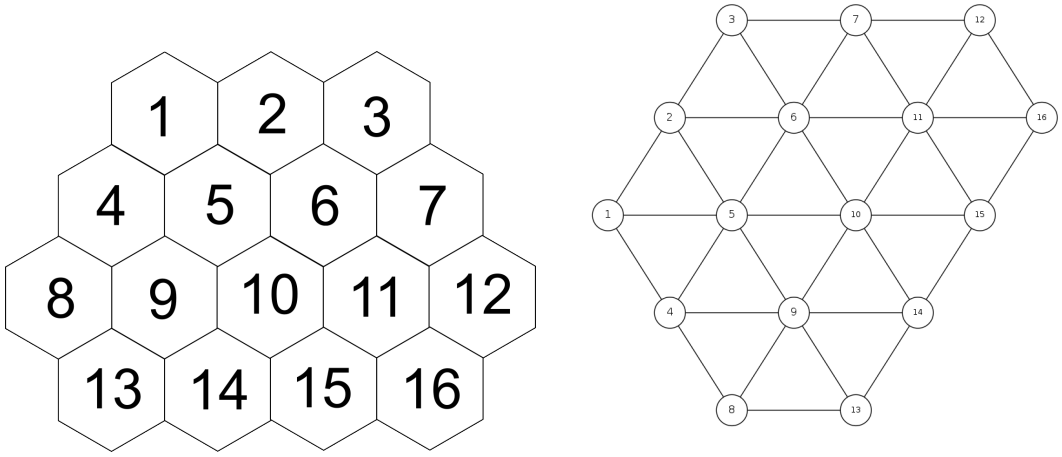
Abbildung 1: Parkettierung mit gleichseitigen Dreiecken

Roboter werden so simuliert, dass die Bewegungen nur über die Kanten einer Zelle möglich sind. Daher ist Anzahl der Ecken identisch mit den möglichen Bewegungsrichtungen. Somit gilt: je mehr Ecken,



(a) (b)

Abbildung 2: Parkettierung mit Quadraten



(a) (b)

Abbildung 3: Parkettierung mit regelmäßigen Sechsecken

desto mehr Freiheitsgrade. Die Kanten geben an, von welchem Knoten zu welchen Nachbarn gewechselt werden kann. Somit hat jeder Knoten auch so viele Kanten wie das gewählte Polygon Ecken hat.

Nach [HH02, S. 12] gilt:

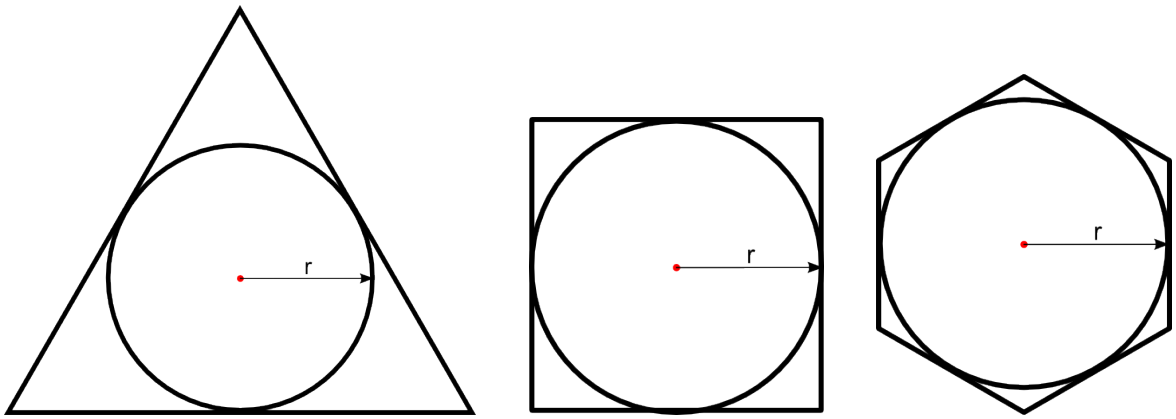
Satz 1. *Eine lückenlose Parkettierung* ***ohne*** *Überschneidungen ist nur mit den regulären n-Ecken für n = 3,4,6 möglich.*

Da eine Simulation mit unregelmäßiger Parkettierung unnötig komplex erscheint, entscheidet es sich zwischen gleichseitigem Dreieck, Quadrat und regelmäßigem Sechseck.

4.2 Polygone und Innenkreis

Der Arbeitsbereich eines Roboters ist kreisförmig, die Zellen, in denen er sich befindet, jedoch ein Polygon. Daher gibt es in den Ecken der Zelle Bereiche, die nicht (unmittelbar) gesaugt werden. Berechnen lässt sich der Verlust Q über das Verhältnis von der Fläche des Innenkreises des Polygons zum Flächeninhalt des Polygons selbst:

$$Q = \frac{A_{\text{Kreis}}}{A_{\text{Polygon}}} \tag{1}$$



(a) (b) (c)

Abbildung 4: Regelmäßiges 3,4,6-Eck mit eingezeichnetem Innenkreis und Radius

Die folgende Tabelle zeigt das Verhälntis von den drei Polygonen.

n	Q
3	60,46 %
4	78,50 %
6	90,69 %

Abbildung 5: Verhältnis Q der Flächeninhalte von Innenkreis eines n-Ecks und dessen gesamten Flächeninhalts

Somit würde zum Beispiel bei Modellierung durch Zellen in Form von regelmäßigen Dreiecken ein Roboter nur 60 % des Mangans darin einsammeln. Je größer n, desto besser wird die unmittelbare Ausbeute.

Die genaue Herleitung kann unter HIER[1] nachvollzogen werden.

4.3 Entscheidung

Die Auswahl fiel schließlich leicht, da die Simulation mit einem Quadrat als zugrundeliegendem Polygon folgende Vorteile bietet:

- Einfache Datenstruktur

[1] HIER

- Einfaches Berechnen der Nachbarn
- Visualisierung einfach, da eine Zelle einem Pixel entspricht, somit keine Umrechnung nötig
- Weniger Verlust als Dreieck
- Weniger Freiheitsgrade als Sechseck (weniger Auswahl bedeutet weniger Rechenaufwand)
- Einfache Berechnung von Entfernungen zwischen zwei Zellen 5.4

Die tatsächliche Modellierung ist in Fig. 6 dargestellt. Zu beachten ist, dass sich das Gitter unendlich weit in alle Richtungen erstreckt, aber in der Implementierung fast nur der erste Quadrant genutzt wird.

Bewegungen können nur in Nachbarzellen erfolgen. Lediglich die Zellen, welche eine Kante mit der Basisfläche gemeinsam haben, gelten als Nachbarn. Die wird auch Von-Neumann-Nachbarschaft genannt.

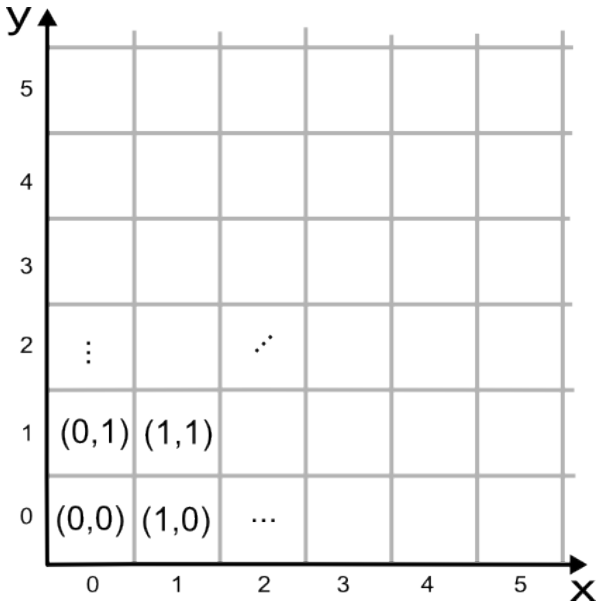


Abbildung 6: Diskretisierung der möglichen Positionen auf dem pazifischen Meeresboden durch quadratische Zellen. Jede Zelle ist eindeutig durch kartesische Koordinaten bestimmt. Das Gitter erstreckt sich unendlich weit in alle Richtungen.

Als wie gut sich diese Entscheidung schließlich herausgestellt hat, wird im Abschnitt HIER[2] evaluiert.

[2]HIER2

5 Theory Crafting

Der folgende Abschnitt beschreibt die zugrundeliegende Theorie hinter der Implementierung.

5.1 Platzierung

Die Platzierung der Roboter R kann wie folgt beschrieben werden: Platziere n Punkte so, dass folgende Bedingung gilt:

$$\forall r \in R \; \exists p \in R : p \neq r \wedge d(p, r) \leq 200 \tag{2}$$

$d(a, b)$ ist hier die euklidische Entfernung zwischen a und b.

Die hier gewählte Lösung ist iterativ. Beginnend mit einem ersten Punkt (gegeben durch `generateFirstPoint`), werden nach und nach Punkte durch `generatePointBoxed` erzeugt und überprüft, ob ein Punkt (2) einhält. Falls ja, wird der Punkt hinzugefügt. `generatePointBoxed` erzeugt einen zufälligen Punkt, der in einem Rechteck $\overline{PQRS}$ liegt, welches als Ecke links unten den Punkt P und als obere rechte Ecke den Punkt R hat. Es gilt:

$$P = (min_x - 200, min_y - 200)$$
$$R = (max_x + 200, max_y + 200)$$

$\texttt{min}_\texttt{x}$, $\texttt{min}_\texttt{y}$, $\texttt{max}_\texttt{x}$, $\texttt{max}_\texttt{y}$ sind hier die minimalen bzw. maximalen x- und y- Werte in R (**nicht** minimaler oder maximaler Punkt).

Dies schränkt die Menge an Punkten ein, die erzeugt werden kann und ist eine grobe Annäherung für (2). Wenn es nicht eingeschränkt würde, gäbe es zu viele Kandidaten zum testen, die eigentlich von vorneherein ausgeschlossen werden könnten.

Es wurde nicht darauf geachtet, wie die Wahrscheinlichkeitsverteilung der Platzierung aussieht, da keine Einschränkungen gegeben wurden, was zufällig genau bedeutet (uniform, normalverteilt, ...). Die zufällig erzeugten Elemente sind grundsätzlich gleichmäßig verteilt, aber die Trial-und-Error Methode könnte das Ergebnis eventuell verfälschen.

Algorithmus 1 : Platzierung von Robotern

Data : Die Anzahl von Robotern n, die platziert werden sollen
Result : Menge von Punkten, sodass (2) erfüllt wird. Punkte werden zufällig erzeugt.

```
R ←{ generateFirstPoint() };
for i ← 1 to n do
    min_x, min_y, max_x, max_y ←getExtrema(R);
    repeat
        p ←generatePointBoxed(min_x, min_y, max_x, max_y);
        nearest ←nearestNeighbour(R, p)
    until d(p, nearest) ≤ 200;
    R ← R ∪ {p};
end
return R
```

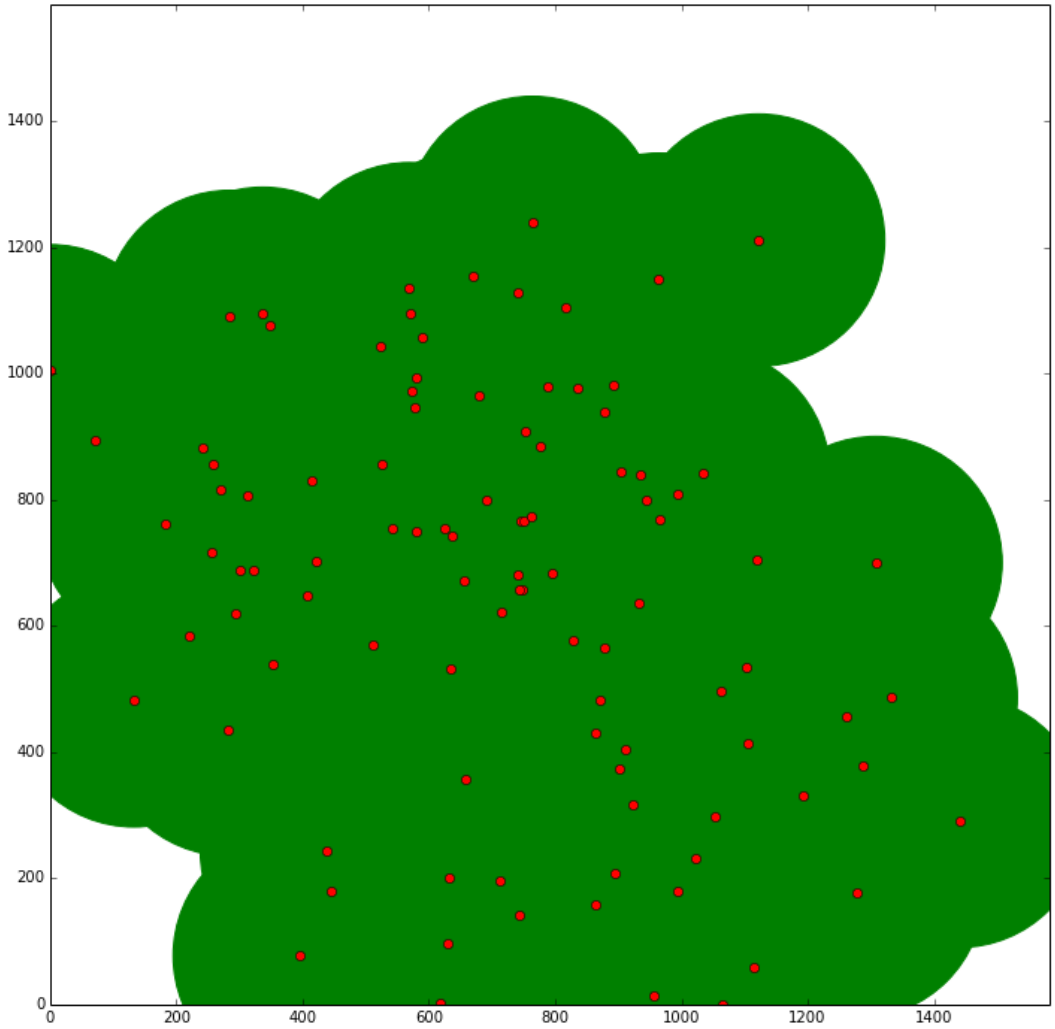


Abbildung 7: Platzierung von 100 Punkten. Die grüne Fläche entsteht aus Kreisen mit Radius 200 um jeden Punkt und bezeichnet die Positionen, für die (2) erfüllt ist. Da die Fläche verbunden ist und alle Punkte enthält, ist diese Anordnung valide.

5.2 Missionsdauer

Die Missionsdauer ist definiert als die minimal benötigte Zeit für ein Treffen aller Roboter an einem gemeinsamen Ort auf dem Meeresboden. Es kann auch als Sonderfall des 1-center problem angesehen werden, welches sagt: Gegeben sei eine Menge von Punkten R, platziere einen Punkt M so, dass die maximale Entfernung eines Punktes $p \in R$ zu M minimiert wird.

Dieses Problemes wird auf das Problem des kleinsten umschliessender Kreises reduziert. Dieser ist der Kreis, welcher alle Punkte enthält und dabei einen minimalen Radius hat. Dabei ist der Radius dessen die maximale Entfernung oder auch Missionsdauer. Der Mittelpunkt des Kreises ist ein potentieller Treffpunkt der Roboter.

Mit der Zeit sind viele verschiedene Algorithmen entwickelt worden. Der hier verwendete, im weiteren `minidisk` genannt, zeichnet sich dadurch aus, dass er äußerst einfach implementiert werden kann. Au-

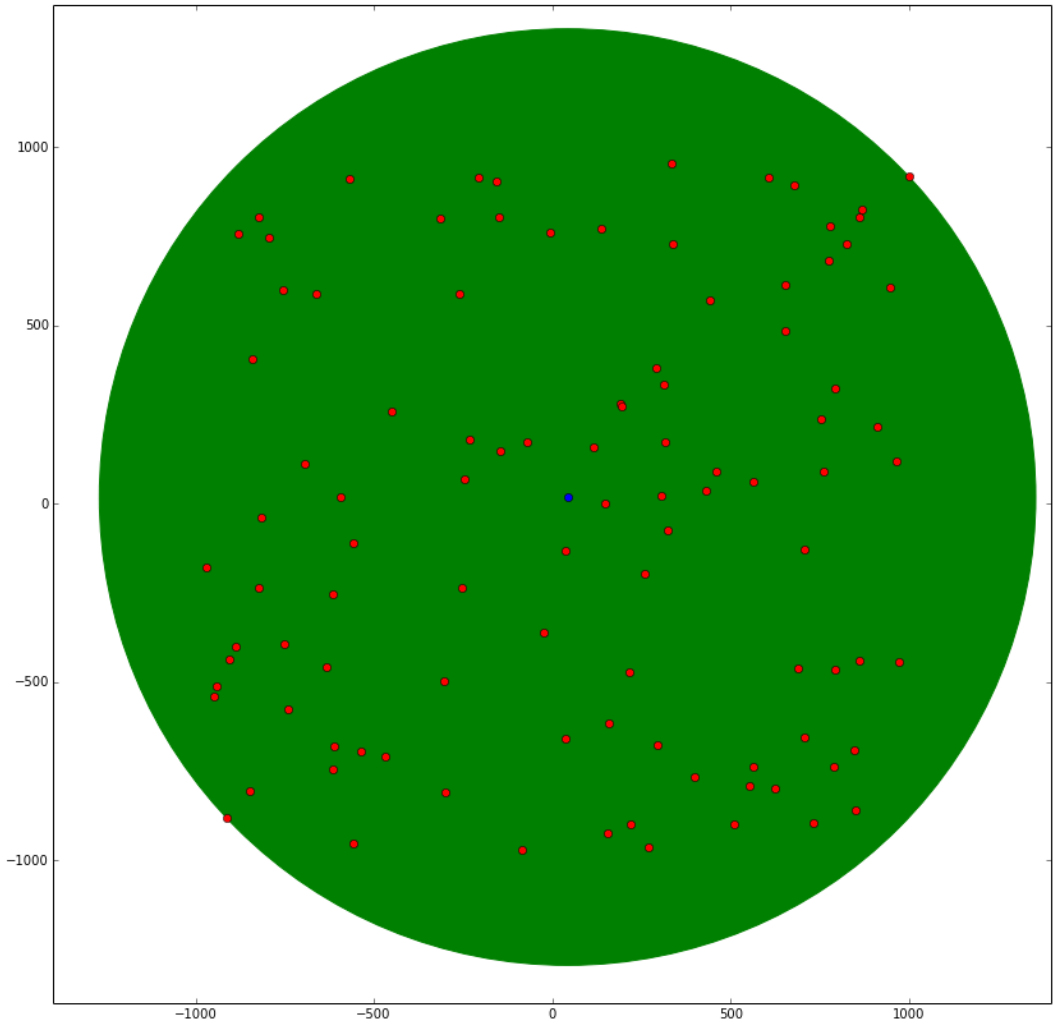


Abbildung 8: Kleinster umschliessender Kreis um eine Menge aus 100 zufällig generierten Punkten

ßerdem hat er eine Komplexität von erwarteten $\mathcal{O}(n)$. Interessant ist dies, da das Problem offensichtlich $\Omega(n)$ ist (jeder Punkt muss mindestens einmal gesichtet werden, da er prinzipiell auf der Kreislinie liegt und somit den Radius vergrößern kann).

Vorgestellt wurde er in [Wel91]. Es wird im Folgenden kurz beschrieben, wie er funktioniert. Für Einzelheiten, z.B. warum es funktioniert, sei auf dieses Paper verwiesen. Die Implementierung von `minidisk` ist wie folgt:

Algorithmus 2 : Kleinster umschließender Kreis

Data : Punktmenge P

Result : Radius und Mittelpunkt des Kreises, der alle Punkte p $\in P$ enthält und einen minimalen Radius hat

```
function minidisk(points)
    function bMinidisk(P, R)
        if P is ∅ then
            return empty disk;
        else if /R/ <= 3 then
            return disk(R);
        else
            return error;
        end
    end
    function bMinidisk(P, R)
        if P is ∅ or /R/ is 3 then
            D ← bMd(∅, R);
        else
            choose random p ∈ D;
            D ← bMinidisk(P \ p, R);
            if p ∉ P then
                D ← bMinidisk(P \ p, R ∪ p)
            end
        end
        return D
    end
    return bMinidisk(points, ∅);
end
```

Zu beachten ist, dass `disk(R)` die kleinste Scheibe berechnet, die alle Punkte in R enthält. Der Algorithmus funkioniert wie folgt: Es gibt zwei Mengen P, R. P sind die Punkte, die noch verarbeitet werden müssen, R die Punkte, die auf dem Rand der Scheibe liegen. In jedem Schritt überprüft, ob es schon eine eindeutige Lösung berechnet werden kann. Dies ist der Fall, wenn P leer ist (kein nächster Schritt möglich) oder R genau drei Elemente enthält (ein Kreis ist eindeutig durch drei Punkte definiert).

Falls keine dieser beiden Bedingungen zutrifft, wird ein Punkt p aus P zufällig ausgewählt und ohne ihn versucht, eine `minidisk` zu berechnen. Falls p dann nicht in der neuen Scheibe liegt, muss er zwangsläufig im nächsten Schritt auf dem Rand liegen.

5.3 Finden der Missionsdaten

Die einfache Missionszeit ist der Radius der `minidisk`, der Treffpunkt ist dessen Mittelpunkt. In Fig. 8 ist dargestellt, wie 100 Roboter (rot) positioniert sind und sich auf einen Treffpunkt (blau) geeinigt haben. Die Missionszeit hier ist minimal, da der Algorithmus versucht, die maximale Entfernung zu einem Punkt zu minimieren. Da nur mit Ganzzahlen gearbeitet wird, wird der Treffpunkt gerundet. Daher muss auch die Missionszeit angepasst werden, sie wird aufgerundet.

5.4 Manhattan-Metrik

Nachdem in 6 gezeigt wurde, wie genau die Umgebung modelliert wurde, fällt auf, dass die Distanz zwischen zwei Zellen nicht der euklidischen Entfernung entspricht. Dies ist in Fig. 9 gezeigt. Daher muss eine neue Entfernungsfunktion gefunden werden, bei der der Weg nur aus horizontalen und Vertikalen Wegstücken besteht.

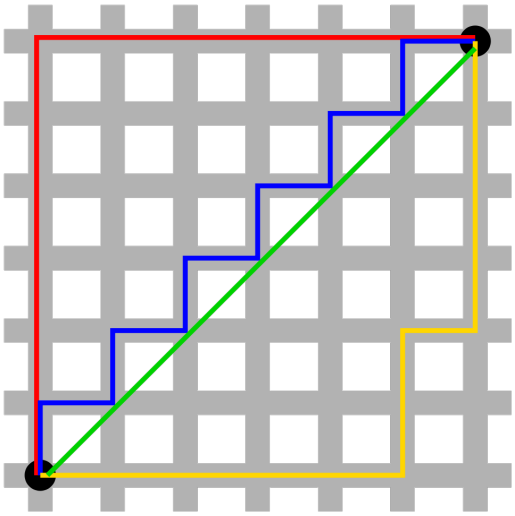


Abbildung 9: Verschiedene, gleichgroße Taxicab-Distanzen zwischen zwei Punkten (je 12 Einheiten lang). Zum Vergleich ist die grüne Linie die euklidische Distanz (etwa 8,5 Einheiten) eingetragen. [Wik]

Glücklicherweise wurde eine solche Geometrie, in der die Entfernung so definiert ist, bereits erforscht. Die Manhattan-Geometrie (oder Taxicab-Geometrie, Cityblock-Geometrie), ist eine Geometrie, in der die übliche, euklidische Entfernungsfunktion durch eine neue Metrik ersetzt ist. Eine Metrik ist hier eine Funktion, die zwei Elemente aus der Geometrie einen reele, nichtnegativen Abstand zuweist.

Ihr Name entstammt der Analogie mit dem Straßennetz von Manhattan (oder Mannheim, wo sich die Universität der beiden Autoren befindet). Es ist gitterförmig angelegt. Um ein Ziel zu erreichen, wird die Entfernung durch Aneinanderreihung von vertikalen und Horizontalen Stücken zurückgelegt.

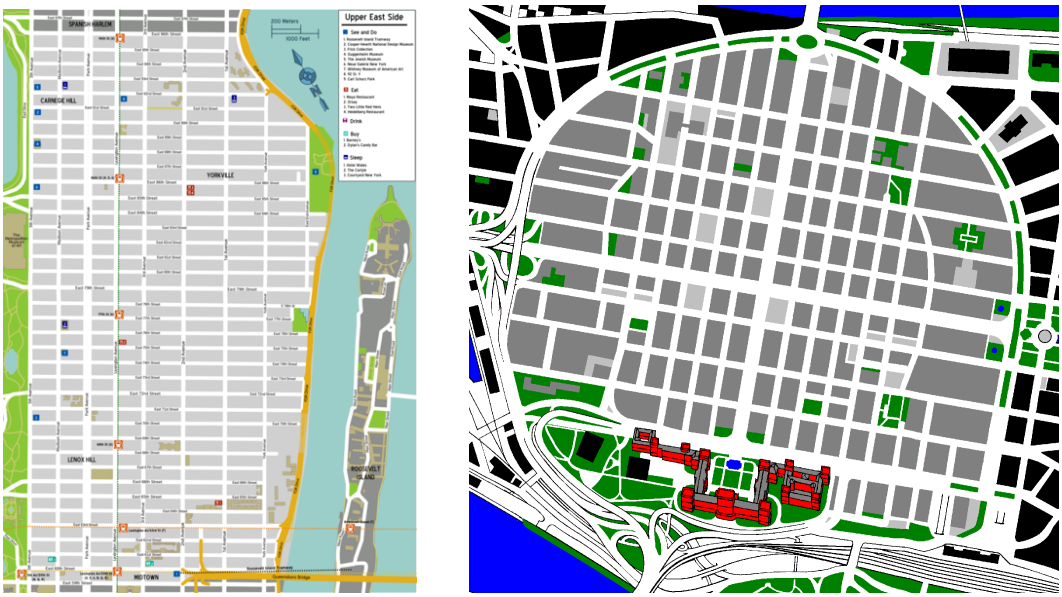
Ein Taxifahrer, welcher eine Route durch solch eine System plant, legt immer die gleiche Strecke zurück, wenn er Wege benutzt, die ihn näher zum Ziel bringen, egal welche Wahl er dabei an Abzweigungen trifft.

Da diese Geometrie der Modellierung des Meeresbodens durch Quadrate entspricht, werden im Folgenden notwendige und wichtige Eigenschaften derer beschrieben.

5.4.1 Entfernung

Die Entfernung zwischen zwei Punkten $a, b \in \mathbb{R}^n$ in einer Manhattan-Geometrie ist die Summe der absoluten Differenzen derer kartesischer Einzelkoordinaten:

$$d(a,b) = \sum_{i=1}^{n} |a_i - b_i|$$



(a) New York (b) Mannheim

Abbildung 10: Auszüge aus Stadplänen von New York und Mannheim.
Auffallend ist die gitterförmige Anordnung der Straßenzüge.

Für den hier relevanten Spezialfall von $\mathbb{R}^2$ gilt somit:

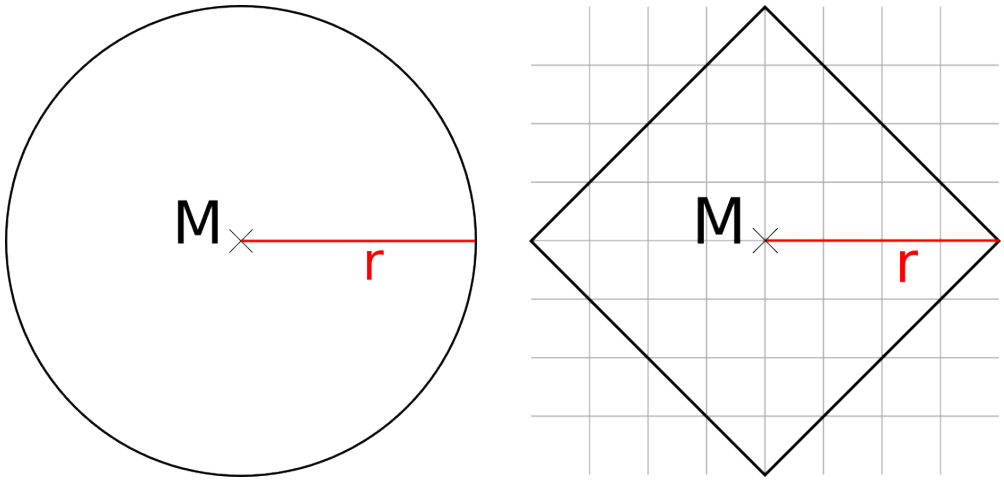
$$d(a, b) = |a_1 - b_1| + |a_2 - b_2|$$

5.4.2 Kreise

Da die Berechnung der Missionszeit auf der Berechnung von Kreisen basiert, müssen diese auch in der verwendeten Taxicab-Geometrie betrachtet werden. Bekannt ist, dass ein Kreis die Menge aller Punkte in einer Ebene ist, welche einen konstanten Abstand, dem Radius, zu einem bestimmten Punkt, dem Mittelpunkt, haben. Die gewählte Metrik, die in dem Wort *Abstand* versteckt ist, hat unmittelbar Einfluss auf die Form des Kreises.

Kreise in einer Taxicab-Geometrie haben die Form von Quadraten, die 45 ° um deren Mittelpunkt gedreht sind. Fig. 11 gibt eine Idee, wieso es so ist. Eine ausführliche Behandlung ist in [Jan07] zu finden.

Die Berechnung von Mittelpunkt und Radius eines Taxicab-Kreises ist schwieriger als in euklidischer Geometrie. Es wird hier keine Formel benutzt, sondern ein Algorithmus. Interessant hier ist das Berechnen von einem Kreis in Taxicab-Geometrie von zwei oder drei Punkten. Das Problem lässt sich auch wie folgt beschreiben: Finde das kleinste Quadrat, welches um 45 ° gedreht ist, welches zwei (drei) Punkte enthält. Da es nicht eindeutig definiert wird, reicht eine einzige Lösung für dieses Problem.



(a) Euklidischer Kreis (b) Taxicab-Kreis

Abbildung 11: Alle Punkte auf der Kreislinie haben den gleichen Abstand zum Mittelpunkt. Auffallend ist, das der Kreis in Taxicab-Metrik nicht rund, sondern quadratisch ist.

Algorithmus 3 : Kreis von zwei Punkten in Taxicab-Geometrie

Data : Zwei Punkte $X, Y \in \mathbb{R}^n$

Result : Radius und Mittelpunkt eines Quadrates, das X und Y enthält und eine minimale Seitenlänge hat. Radius hier ist die halbe Länge der Diagonale.

1. Um die Lösung des Problems zu vereinfachen, werden die Punkte je um 45 ° um den Ursprung herum gedreht, dann das Quadrat berechnet, die so enstandende Lösung schließlich zurückgedreht.
2. Rotiere X, Y 45 ° um den Ursprung und nenne die so entstehenden Punkte P, Q.
3. Offensichtlich ist nun die Seitenlänge s des minimalen Quadrates gegeben durch

$$s = \max\{|P_x - Q_x|, |P_y - Q_y|\} \tag{3}$$

4. Um eine Lösung zu finden, wähle die unterste linke Ecke als Ausgangspunkt und nenne sie A.

$$\begin{cases} A_x = \min\{P_x, Q_x\} \\ A_y = \min\{P_y, Q_y\} \end{cases} \tag{4}$$

5. Der rotierte Mittelpunkt M' ist somit gegeben durch

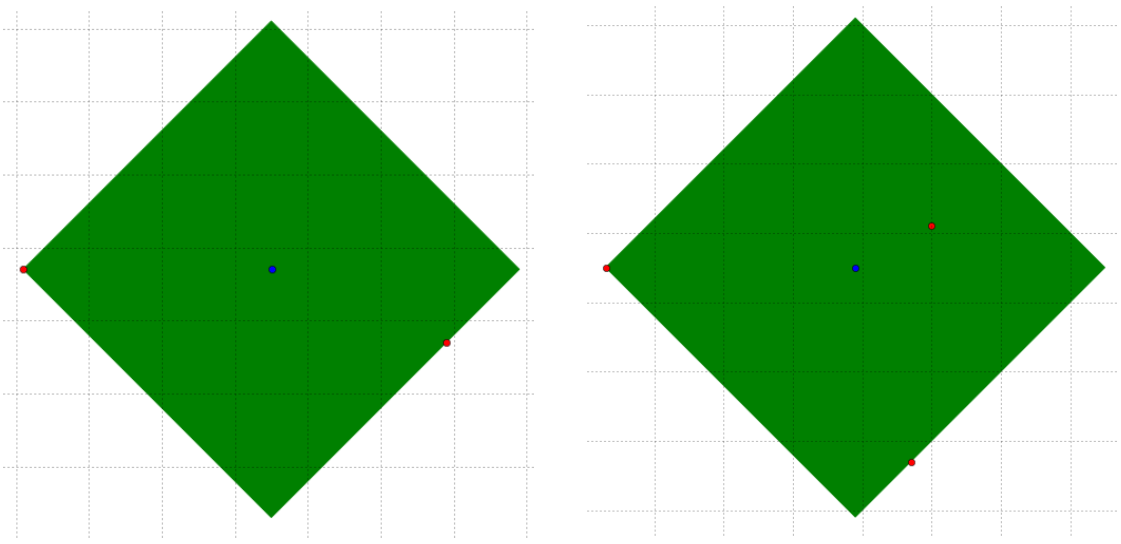
$$\begin{cases} M'_x = A_x + \frac{s}{2} \\ M'_y = A_y + \frac{s}{2} \end{cases} \tag{5}$$

6. Rotiere M' um 45 ° um den Ursprung und nenne den so entstehenden Punkt M.
7. Der Radius r ist gegeben durch

$$r = \sqrt{A_x - M'_x)^2 + (A_y - M'_y)^2} \tag{6}$$

return r, M

Interessanterweise liefert dieser Algorithmus durch geringfügige Anpassungen auch die Lösung für



(a) Zwei Punkte (b) Drei Punkte

Abbildung 12: Kreis in Taxicab-Geometrie von zwei respektiv drei Punkten. Auffallend ist, dass mindestens zwei Punkte auf der Kreislinie liegen.

folgendes Problem: Finde das kleinste Quadrat, welches um 45 ° gedreht ist, welches **drei** Punkte enthält:

Algorithmus 4 : Kreis von drei Punkten in Taxicab-Geometrie

Data : Drei Punkte $X, Y, Z \in \mathbb{R}^n$

Result : Radius und Mittelpunkt eines Quadrates, das X, Y, Z enthält und eine minimale Seitenlänge hat. Radius hier ist die halbe Länge der Diagonale.

1. Um die Lösung des Problems zu vereinfachen, werden die Punkte je um $45\,^\circ$ um den Ursprung herum gedreht, dann das Quadrat berechnet, die so enstandende Lösung schließlich zurückgedreht.
2. Rotiere X, Y, Z $45\,^\circ$ um den Ursprung und nenne die so entstehenden Punkte P, Q, R.
3. Offensichtlich ist nun die Seitenlänge s des minimalen Quadrates gegeben durch

$$s = \max\{|P_x - Q_x|, |P_x - R_x|, |Q_x - R_x|, |P_y - Q_y|, |P_y - R_y|, |Q_y - R_y|\} \tag{7}$$

4. Um eine Lösung zu finden, wähle die unterste linke Ecke als Ausgangspunkt und nenne sie A.

$$\begin{cases} A_x = \min\{P_x, Q_x, R_x\} \\ A_y = \min\{P_y, Q_y, R_y\} \end{cases} \tag{8}$$

5. Der rotierte Mittelpunkt M' ist somit gegeben durch

$$\begin{cases} M'_x = A_x + \frac{s}{2} \\ M'_y = A_y + \frac{s}{2} \end{cases} \tag{9}$$

6. Rotiere M' um $45\,^\circ$ um den Ursprung und nenne den so entstehenden Punkt M.
7. Der Radius r ist gegeben durch

$$r = \sqrt{A_x - M'_x)^2 + (A_y - M'_y)^2} \tag{10}$$

return r, M

Im Rahmen dieser Arbeit wurden diese Algorithmen auch mittels GeoGebra visualisiert. Eine interaktive Visualisierung für einen Kreis in Taxicab-Geometrie, der zwei (drei) Punkte enthält, ist unter folgenden Links zu sehen:

`http://www.geogebratube.org/student/m65241`
`http://www.geogebratube.org/student/m68655`

Interessanterweise funktioniert `minidisk` auch mit Taxicab-Kreisen, wenn man die `disk`-Funktion anpasst. Dies wurde hier nicht mathematisch bewiesen, aber die Ergebnisse sprechen für sich.

6 Zeitbeschränkung

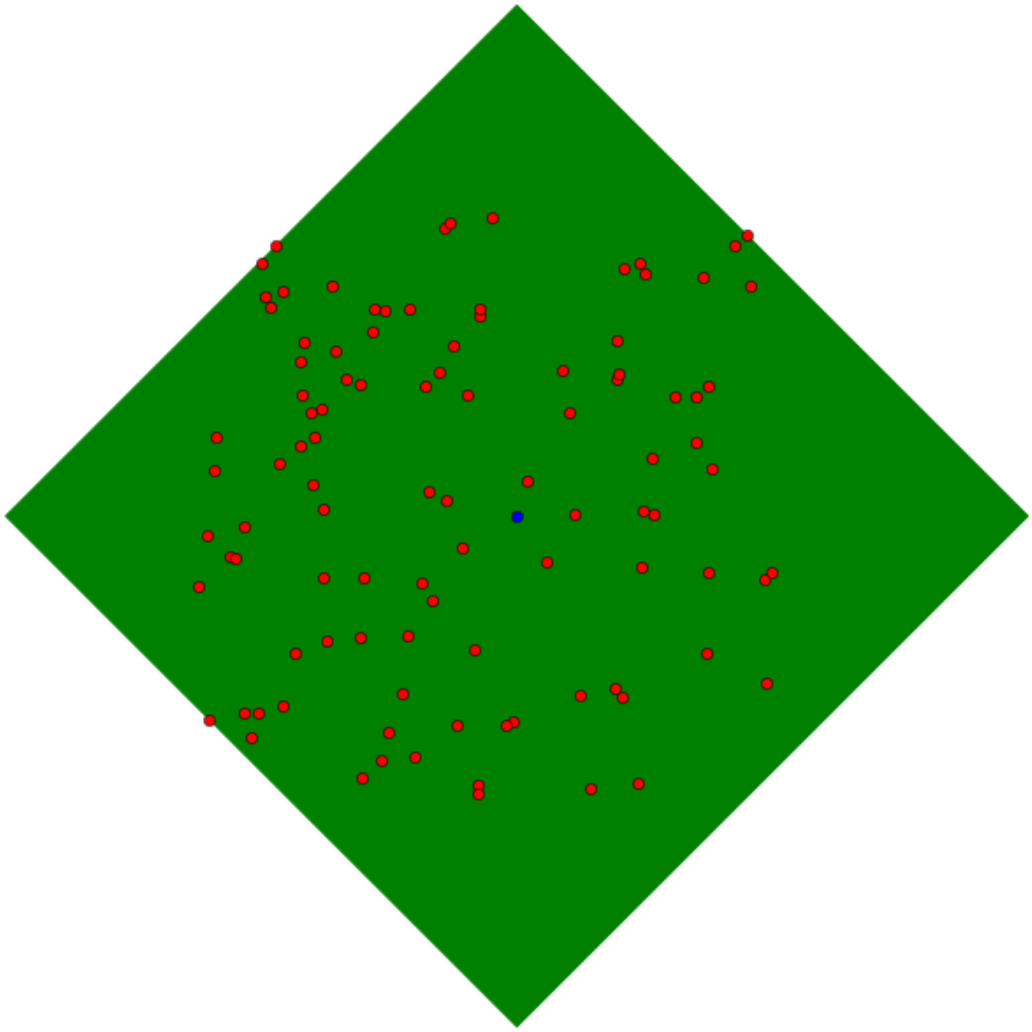


Abbildung 13: Platzierung von 100 Punkten und deren kleinster umschließender Kreis in Taxicab-Geometrie.

7 Implementierung

8 Diskussion

8.1 Performance

8.2 Fehlerrechnung

8.2.1 Abweichung vom Optimum

8.2.2 Fehler durch Modellierung

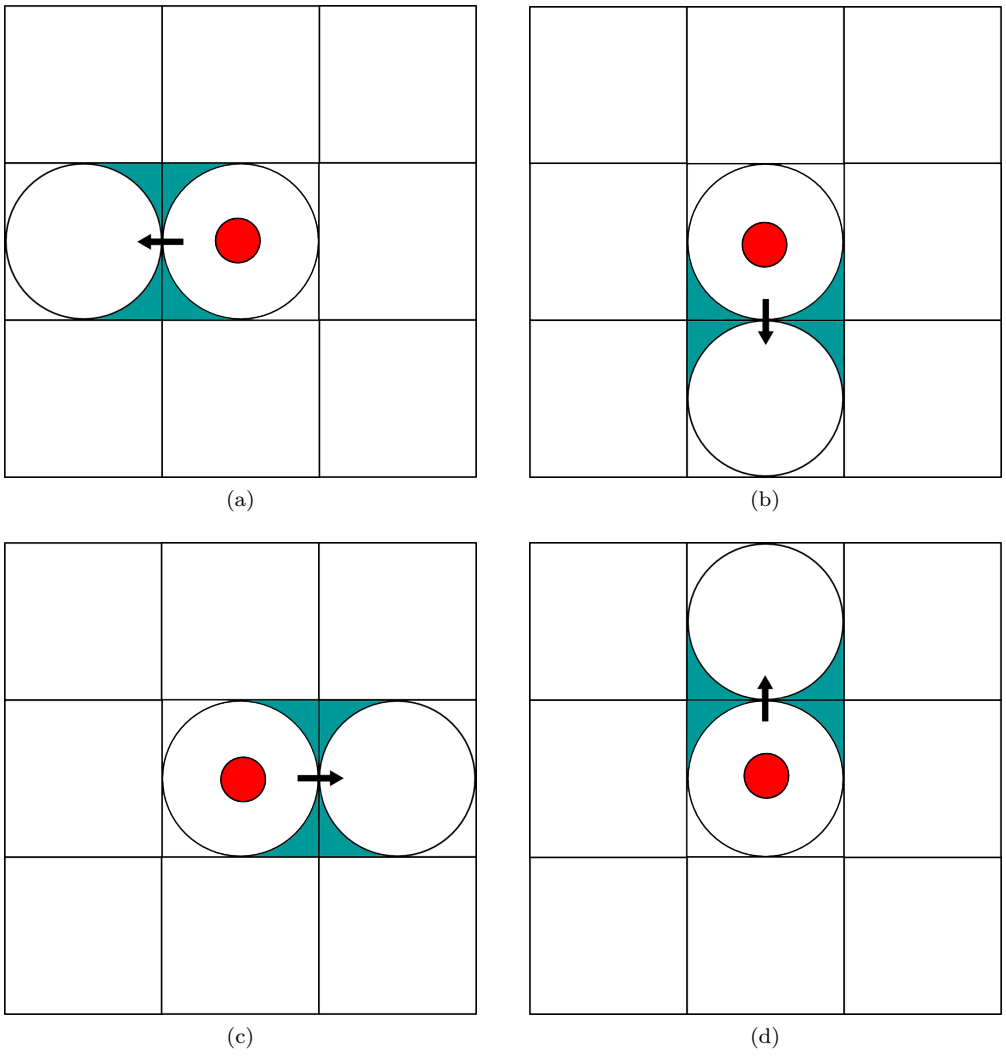


Abbildung 14: Überstreichen der Ecken beim Wechseln in eine benachbarte Zelle: Der Roboter (hier rot), kann nicht nur den ursprünglichen, kreisförmigen Teil in der Mitte einer Zelle abernten, sondern beim Wechseln auch die Ecken, da sein Arbeitsbereich diese in der Bewegung überschneidet.

8.3 Lessons learned/Fehlentscheidungen

- Zu viele Abstraktion, die sich später unnötig herausgetsellt hat (Geometry, Circle)

8.4 Ausblick

Literatur

[HH02] **Christian Hartfeldt** und **Prof. Dr. Herbert Henning**. *Muster, Flächen, Parkettierungen — Anregungen fur einen kreativen Mathematikunterricht*. Techn. Ber. Otto-von-Guericke-Universität Magdeburg, 2002. URL: http://www.math.uni-magdeburg.de/reports/2002/parkett.pdf.

[Jan07] **Christina Janssen**. *Taxicab Geometry: Not the Shortest Ride Across Town*. Techn. Ber. Iowa State University, 2007. URL: http://www.math.iastate.edu/thesisarchive/MSM/JanssenMSMSS07.pdf.

[Kli08] **Peter Kling**. *Facility Location*. Techn. Ber. Universität Paderborn, 2008.

[Wel91] **Emo Welzl**. "Smallest Enclosing Disks (balls and Ellipsoids)". In: *Results and New Trends in Computer Science*. Springer-Verlag, 1991, S. 359–370. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.46.1450&rep=rep1&type=pdf.

[Wik] **Wikipedia**. *Manhattan-Metrik*. URL: https://de.wikipedia.org/wiki/Manhattan-Metrik.