Teil III
Erforschen von Graphen und Umgebungen

Martin Löhnertz, André Weilert

1 Einleitung

Der folgende Beitrag versucht, einen Einblick in graphentheoretische und geometrische Problemstellungen und Lösungsstrategien zu geben, die im Rahmen der Online-Problematik auftreten. Die Erkenntnisse, die in diesem Gebiet gewonnen wurden, sind, obwohl man annehmen sollte, daß die Probleme in modifizierter Form schon seit Jahrhunderten bestehen, überwiegend jüngeren Datums, was dazu führt, daß es weder eine einheitliche Theorie noch einen systematischen Ansatz zur Klassifizierung gibt. Einerseits inspiriert durch grundlegende Arbeiten wie z.B. [24] andererseits aus der Praxis, i.e. der Robotik oder sogar aus der Automatentheorie heraus ([6]) motiviert wurden zahlreiche Verfahren entwickelt, die aufgrund der Unterschiedlichkeit der Anforderungen, aber auch der analytischen Methoden bzw. der als solche deklarierten Variablen und Konstanten teilweise unvergleichbar sind. So finden sich in [21] Algorithmen zum Überwinden von Hindernissen, die in der Länge des Randes dieser Hindernisse kompetetiv sind, während z.B. [17] kompetetiv bezüglich der zurückgelegten Wegstrecke und [22] kompetetiv bezüglich der Zahl der Hindernisse bleibt. Bei den Graphen besteht immerhin noch ein Zusammenhang zu den theoretisch besser erschlossenen "Greedy"-Verfahren auf diesen (Matroide etc.); dieser wurde aber in keiner der von uns betrachteten Literaturstellen verwandt.
Dementsprechend beginnt der Beitrag mit einem kurzen Systematisierungsversuch, geht dann auf die Rahmenbedingungen ein und besteht im Hauptteil aus zwei getrennten Beiträgen zu Graphen[1] und Hindernisparcours [2], in denen der Versuch unternommen wird, einige der wichtigsten Ergebnisse und Strategien vorzustellen.

2 Definitionen

In der Literatur findet sich keine einheitliche Terminologie zur Beschreibung der Problematiken mit formalen Methoden. Um eine Klassifizierung der Problemstellungen vornehmen zu können, und allgemein eingängigere Formulierungen zu erzielen, ist es daher sinnvoll, zunächst einige Vereinbarungen vorzunehmen. Um eine unnötige Verkomplizierung der Situation, die sich durch eine allgemeine Definition häufig ergibt, zu vermeiden, unterscheiden wir allerdings zwischen der für dieses Referat notwendigen, und der tatsächlich zu verlangenden Notation. D.h. bei Verweisen auf Probleme,

[1] hauptsächlich André Weilert
[2] hauptsächlich Martin Löhnertz

die sich mit der nun gegebenen Definition nicht darstellen lassen, wird die entsprechende Erweiterung angegeben, wobei diese aber im allgemeinen nur die Schritte der Mengen-, bzw. Potenzmengenbildung verlangt.

2.1 Graphen

Definition 2.1.1 (Online Graphenproblem) *Ein Online Graphenproblem ist ein* 8*-Tupel* $\mathcal{GP} := (V, E, C, \mathfrak{O}, K_0, \mathcal{K}_{fin}, \kappa, \Upsilon)$,

wobei:

- $G = (V, E)$ der zu betrachtende (Multi-)Graph ist.
- C ist eine Menge von sonstigen Informationen, die mit Knoten oder Kanten assoziiert werden. Z. B. eine Kostenfunktion auf den Kanten.
- $\mathfrak{O} : 2^V \times 2^E \to 2^V \times 2^E \times 2^C$ wird im folgenden als Online-Begrenzungsfunktion bezeichnet. Sie ordnet den vollzogenen Aktionen die dadurch gewonnene Information zu.
- K_0 ist die Startkonfiguration
- $\mathcal{K}_{fin}$ die Menge der Endkonfigurationen
- $\kappa : \mathcal{K} \cup \mathcal{K}^2 \to \mathbb{R}$ die Kostenfunktion
- $\Upsilon : \mathcal{K}_{fin} \to \mathbb{R}$ die Zielfunktion

Definition 2.1.2 (Konfiguration) *Eine Konfiguration beschreibt einen Zustand während der Berechnung. Sie wird dargestellt durch ein* 3*-Tupel* $K_i = (\tilde{W}_i, \tilde{F}_i, p_i)$

- $\tilde{W}$:= "Menge der besuchten Ecken"
- $\tilde{F}$:= "Menge der besuchten Kanten"
- p ist der "Aufenthaltsort" nach Schritt i

Entsprechend ergibt sich die Menge $\mathcal{K}$ aller möglichen Konfigurationen, mit $K_0 \in \mathcal{K} \supset \mathcal{K}_{fin}$ Als weitere Abkürzungen seien vereinbart:

- $\hat{W} := \mathfrak{O}_W(\tilde{W}, \tilde{F}) := \pi_1 \circ \mathfrak{O}(\tilde{W}, \tilde{F})$ "Menge der bekannten Ecken"
- $\hat{F} := \mathfrak{O}_F(\tilde{W}, \tilde{F}) := \pi_2 \circ \mathfrak{O}(\tilde{W}, \tilde{F})$ "Menge der bekannten Kanten"
- $\hat{C} := \mathfrak{O}_C(\tilde{W}, \tilde{F}) := \pi_3 \circ \mathfrak{O}(\tilde{W}, \tilde{F})$ "Menge der bekannten sonstigen Informationen"

Diese Darstellung der Problematik schließt z.B. das Erhalten der Informationen aus anderen Quellen aus (s. vgl 5.4 auf Seite III-9).

2.2 Hindernisparcours

Definition 2.2.1 (Online Environment) *Ein Online Environment (Hindernisparcours) ist ein* 8*-Tupel* $\mathfrak{CE} = (A, H, M, \mathfrak{O}, K_0, \mathcal{K}_{fin}, \kappa, \Upsilon)$

mit

- Polygon $P : (p_j)_{j \in J} \;\; \forall j \in J : p_j \in \mathbb{R}^2$ (Kanten überschneiden einander nicht)[3]
- Umschließendes Polygon $A \subset \mathbb{R}^2$ oder $A = \mathbb{R}^2$
- Hindernisse (Polygone) $(H_i)_{i \in I}$ $\forall i : H_i \subset A$
- $M :=$ Menge von "interessanten" Punkten
- $\tilde{H} :=$ "Menge der besuchten Positionen"
- Online-Begrenzungsfunktion: $\mathfrak{O} : 2^{\mathbb{R}^2} \to 2^{\mathbb{R}^2}, \mathfrak{O}(\tilde{H}) = (\hat{W}, \hat{F})$
- Konfigurationen : $K_i = (\tilde{H}_i, p_i) \in 2^{\mathbb{R}^2} \times \mathbb{R}^2$, s.o.

Des weiteren scheinen folgende Vereinbarungen sinnvoll, auch wenn wir die Begriffe nur implizit benutzen:

- VisibilityGraph $G_{\mathfrak{CE}} = (V, E)$;

  $$V = \{p | \exists i : H_i = (\cdots, p, \cdots) \vee A = (\cdots, p, \cdots) \vee p \in M\}$$

  Kanten:

  $$(u, w) \in E \Leftrightarrow \forall \lambda \in (0, 1) : \lambda w + (1 - \lambda) u \in A \backslash \left(\bigcup_{i \in I} H_i \right)$$

- Identifikationsfähigkeit $\mathfrak{I} : 2^{\mathbb{R}^2} \to 2^{V_{\mathfrak{CE}}} \times 2^{E_{\mathfrak{CE}}}, \mathfrak{I} \circ \mathfrak{O}(\bar{X}) = (\hat{W}, \hat{F})$

2.2.1 Gegner und Begrenzungsfunktion

Bei der Konzipierung der Definitionen wurde in Anbetracht der Tatsache, daß in der Praxis vorkommende Gegner meist von Typ OBLIVIOUS sind eine eher statische Beschreibung der Probleme gewählt, was darin Ausdruck findet, daß die Mengen nicht von den Entscheidungen des Algorithmus abhängen. Es sei daher an dieser Stelle darauf hingewiesen, daß wir die obigen Vereinbarungen in erster Linie in analytischen Zusammenhängen einsetzen wollen. D.h. nachdem der Algorithmus abgelaufen ist, wird sein Verhalten anhand des hierbei erstellten Graphen analysiert. In diesem Zusammenhang erscheint eine ehemalige Entscheidung eines Gegners als Entdeckung einer bestehenden Tatsache. In den meisten Fällen ist es allerdings so, daß tatsächlich von festen Graphen oder Umgebungen ausgegangen wird, die der Gegner als Ganzes erzeugt.

[3] Dies läßt sich natürlich auf den $\mathbb{R}^n$ verallgemeinern.

Zudem läßt sich die Definition der Gegnertypen der Literatur zu diesem Thema hier nicht vollständig übertragen und ist auch nicht konsistent zur allgemeinen Definition der Online-Thematik, wie z.B. List-Update o.ä. . So z.B. wird im Environment-Kontext ein Gegner als OBLIVIOUS bezeichnet, dem zwar unbekannt ist, wie der Algorithmus zu einem speziellen Zeitpunkt entschieden hat, der aber über den jeweils aktuellen Standort informiert ist.

3 Komplexität

Vor dem Versuch geeignete Problemlösungen finden zu wollen, sollte man den Rahmen abstecken, in dem diese liegen können. Viele Graphenprobleme sind NP-hart, was die Existenz eines guten "Greedy"-Algorithmus von vornherein ausschließt. Da die Laufzeit bei der Bestimmung des optimalen Offline-Algorithmus aber keine Rolle spielt müssen die Online-Algorithmen sozusagen gegen die "vielleicht existente" polynomielle Lösung der NP-vollständigen Probleme[4] antreten, was sich am folgenden Beispiel am besten verdeutlichen läßt: Gesucht: ein kompetetiver Algorithmus für das Problem des Handlungsreisenden. D.h. die Lösung darf nur um einen konstanten Faktor c von der optimalen Lösung abweichen. Gewichtet man aber bei einem beliebigen einfachen Graphen alle existenten Kanten mit 1 und fügt die nichtexistenten hinzu und gewichtet diese mit $(|V| + 1) * c$, so würde obiger Algorithmus hamiltonsche Kreise in diesem Graphen aufspüren, da die Verwendung einer einzigen nicht existente Kante einen Weg liefern würde dessen Länge um mehr als das c-fache von einer optimalen Lösung (so existent) abweichen würde. Da aber die einzige Methode einen HC zu finden (Annahme, $\mathcal{P} \neq \mathcal{NP}$) im Durchtesten aller Varianten besteht, kann es keinen Greedy-Algorithmus und somit auch keinen Online-Algorithmus geben. Die folgende Abbildung zeigt z.B. einen Graphen, bei dem man bereits bei der ersten Kante die Entscheidung über das Gelingen trifft, ohne die notwenigen Informationen besitzen zu können.

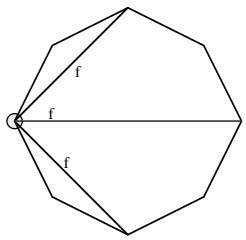

Abbildung 3.0.1 (OnlineHamiltongraph)

Konkret kann man für einige besonders einfache Probleme zeigen, daß das Finden einer Lösung in PSPACE (und somit in EXPTIME) ist[5]:

[4] d.h. zur Zeit gegen ein Backtracking Verfahren
[5] nach einer Beweisidee von [24]

Abbildung 3.0.2 (PSPACE)

Zu einer gegebenen Zielkompetetivität $r > 1$ seien m und M seien wie folgt gewählt: m sei so groß, daß $\frac{m+|V|}{m} < r$ und $\frac{M}{m+|V|} > r$. Die Kästchen entsprechen den Klauseln, und jedem der dick dargestellten Vierecke ist eine Variable zugeordnet. Die jeweils untere Ecke ist mit einer Klausel genau dann verbunden, wenn die entsprechende Variable in der Klausel negiert vorkommt, die obere, wenn sie nicht negiert vorkommt. Der Sucher kann von S aus nicht senkrecht nach unten gehen, da der Gegner die Kosten für diesen Weg auf M setzen und selber über den gleich beschriebenen Weg mit Kosten

$\leq m + |V|$ zum Ziel gelangen kann. Dieser Weg (A) steht also nur dem Gegner offen, und kostet diesen 1. Jede dick dargestellte Gabelung ohne Wertangabe entspricht einem Existenzquantor: Der Sucher kann sich den Weg aussuchen. Jede dick dargestellte Gabelung mit Wertangabe entspricht einem All-Quantor. Hier kann der Gegner durch geeignetes Zuweisen der Werte die Entscheidung des Suchers festlegen. Ist eine derartige Entscheidung getroffen, kann der Sucher von einem der mit 0 oder 1 markierten Knoten aus den "Ausgang" aller Klauseln (Kästchen) sehen, die die entsprechende Variable mit der entsprechenden Belegung beinhalten. Schließt der Gegner diese nicht, so hat der Sucher einen Weg der Länge $\leq m + |V|$ gefunden, der nach Wahl von m r-kompetetiv ist. Kann der Sucher auf diese Weise alle Klauseln erreichen, muß der Gegner alle Ausgänge schließen, wodurch der kürzeste Offline-Weg eine Länge $\geq M$ erhält, der Algorithmus also ebenfalls kompetetiv ist. Bleibt aber eine Klausel unerreicht und werden die anderen geschlossen, so kann der Gegner über Weg A mit Kosten $\leq m + |V|$ t erreichen, während der Sucher $\geq M$ benötigt. Es gibt offenbar genau dann eine allgemein kompetetive Strategie, wenn der Sucher an jeder Existenz-Entscheidung so entscheiden kann, daß egal, wie die $\forall$-Entscheidungen getroffen werden, alle Klauseln erreicht werden können, d.h. genau dann, wenn es eine Lösung des korrespondierenden Q-SAT-Ausdrucks gibt.

4 Erforschen von Graphen

Andre's Kram

5 Erforschen von Hindernisparcours

5.1 Definitionen und Vereinbarungen

Definition 5.1.1 (Aspect Ratio) *Unter der "Aspect Ratio" α eines allgemeinen Polygons versteht man das Verhältnis von Umkreis- zu Inkreisradius.*

Definition 5.1.2 (Aspect Ratio eines Rechtecks) *Bei einem Rechteck bezeichnet man als Aspect Ratio das Verhältnis der längeren Seite zur kürzeren.*

- **Sucher**: Die aktuelle Suchposition. z.B. ein Robot
- **Gegner**: Wenn nicht anders angegeben, handelt es sich um einen OBLIVIOUS Gegner, der den Algorithmus nicht kennt und dem die Aktionen seines Gegners bis zum jeweiligen Zeitpunkt seiner Entscheidung bekannt sind; dies ist kein OBLIVIOUS Gegner in dem Sinn, wie er im allgemeinen Online-Bereich verwandt wird, sondern eine eingeschränkte Variante eines adaptiven offline-Gegners, diese Art der Bezeichnung scheint sich im Umgebungskontext aber durchgesetzt zu haben [24].

5.2 Veränderungen gegenüber der Graphenproblematik

Betrachtet man den Visibility-Graphen eines Online-Environmentproblems (s. 2.2), so kann der Eindruck entstehen, daß durch Verwenden der für Graphen entwickelten Methoden sämtliche Online-Environment Probleme gelöst werden können. Dies ist aber nicht der Fall, da durch die Abstrahierung mit Graphen wesentliche Strukturinformationen verlorengehen[6]. Insbesondere erzeugen die Kenntnis des Zielpunktes und das Wissen, wann man einen Kreis geschlossen hat, so wie Beschränkungen die den Kanten durch ihre Lage im $\mathbb{R}^2$ auferlegt werden, eine völlig neue Gewichtung der Kanten. Diese läßt sich allerdings nicht so intuitiv, wie man vermuten möchte zur Modifikation der Strategien einsetzen. Hierzu möge folgendes Gegenbeispiel aus [23] als Anschauungshilfe dienen:

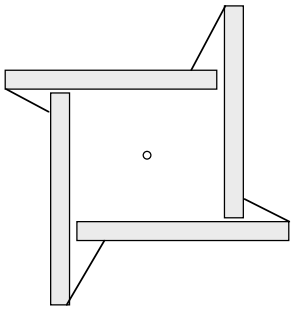

Abbildung 5.2.1 (Gegenbeispiel)

Offenbar würde ein Algorithmus, der der Strategie folgt, das Skalarprodukt zwischen dem Bewegungsvektor und dem zum Ziel zeigenden Vektor zu maximieren,in bestimmten Fällen nicht terminieren.

5.3 Einige grundlegende Ergebnisse

Das wohl grundlegenste Problem aus diesem Bereich besteht darin, auf einer Geraden einen konkreten Punkt in unbekannter Entfernung zu finden (nach [1]).

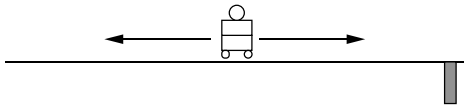

Abbildung 5.3.1 (Dimension:1)

Zur Lösung dieses Problemes verwendet man eine "Verdopplungsstrategie": Man gehe zunächst k Schritte nach links, von dort $3k$ Schritte nach rechts, dann 6 Schritte nach links usw. , so daß der Abstand von der Anfänglichen Startposition im $i$ten Schritt 2^i ist.

Behauptung 5.3.1 (Verdopplung) *Die obige Strategie ist* 9*-kompetetiv, und dies ist optimal.*

Es ergibt sich kanonisch das Problem, einen Punkt auf mehreren Geraden zu suchen, das sog. Cow-Path-Problem:

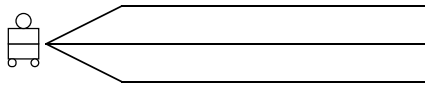

Abbildung 5.3.2 (Cow-Path)

Hier verdoppelt man bei jedem Wegwechsel (zyklisch) die zurückgelegte Strecke. Angewandt auf das Problem einen Zielpunkt in einem nichtkonvexen Polyeder zu finden (nach [5]), ergibt sich sofort, daß es hierfür keinen in der Anzahl der Hindernisse (hier 1) kompetetiven Algorithmus für dieses Problem geben kann.

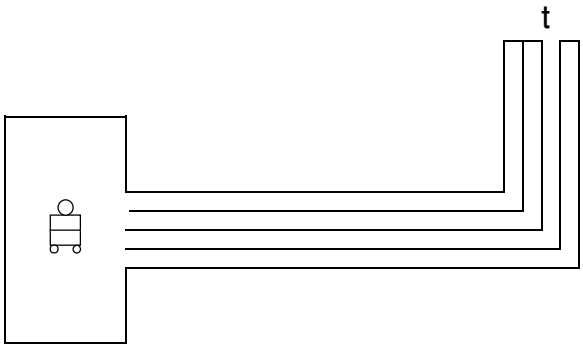

Abbildung 5.3.3 (Nichtkonvexe Hindernisse)

Hier geht man dann zur Anzahl der Ecken des jeweiligen Polyeders über, und erhält, daß für einen randomisierten Algorithmus eine Kompetetivität von $\frac{|V|}{8}$ nicht unterschritten werden kann ([5]).

5.4 Übersicht über die Problemstellungen und Begründung der Auswahl

Aufgrund der (wenn auch eher zukünftigen) Anwendungsrelevanz dieses Themas gibt es viele Ausprägungen der Problematik. Des weiteren führt die generelle oder praktische Unlösbarkeit allgemeiner Problemstellungen (s.vgl. 5.5.1 auf Seite III-11) dazu, daß man verschiedene Fälle mit verschiedenen Einschränkungen betrachten muß, um Lösungsstrategien für möglichst viele Situationen zu entwickeln. Wie auch bei den Graphen ist hier eine Klassifizierung nach Endzuständen und Online-Begrenzungsfunktionen möglich:
Man findet in der Literatur überwiegend zwei Fälle:

- FIND: $\mathcal{K}_{fin} = \{K = (\tilde{W}, \tilde{F}, p) | p = t_0 \in R^2\}$, d.h. das Ziel ist es, einen konkreten Zielpunkt (Target) zu erreichen.
- MAP: $\mathcal{K}_{fin} = \{K = (\tilde{W}, \tilde{F}, p | \tilde{W} = W$ [6]$\}$

Des weiteren ergibt sich die Online-Begrenzungsfunktion kanonisch aus den Eigenschaften des "Wahrnehmungsorgans" i.e. des Sensors der betreffenden Maschine: Hier gibt es im wesentlichen drei Fälle, die alle einer Definition genügen:

$$\mathfrak{O}(X) = \{y | \exists x \in X \wedge \overline{xy} \cap \bigcup_{i \in I} H_i = \emptyset \wedge \min_{x \in X} ||y - x|| \leq d\}$$

Wobei man die beiden Fälle $d = 0$ (BOUNCE) und $d = \infty$ (SEE) am ausführlichsten behandelt hat.
Man findet aber noch viele Spezialfälle:

- **Search Party Problem** [16] Statt einem Sucher gibt es hier k Stück. Dieses Problem wird durch die gegebenen Definitionen nicht gedeckt (man bilde das k-fache carteisische Produkt), da komplexe Kommunikationsstrukturen zwischen den Suchern gegeben sein können.
- **Mehrfaches Suchen** [3] Die selbe Suchaufgabe soll mehrfach ausgeführt werden. Da bei jedem Suchvorgang mehr Informationen gewonnen werden können, kann der Weg immer weiter verbessert werden. Amortisiert ist hier zunächst DFS auf dem Visibility-Graphen und dann $k-1$ faches Verwenden des optimalen (offline) Weges ideal. [3] zeigen aber eine Variante, diesen zu glätten. Hier treten die selben Endkonfigurationen wie bei FIND auf, aber man kann diese (zunächst) wieder verlassen, und zudem ist eine Zielfunktion Υ gegeben durch

$$\Upsilon(K) := \text{Länge des kürzesten Pfades von } s \text{ nach } t \text{ in } (\hat{W}, \hat{F})$$

Eine weitere Differenzierung ergibt sich nach der Form der Hindernisse. Hier unterscheidet man zwischen Objekten mit beschränkter bzw unbeschränkter Aspect Ratio, zwischen konvexen und nichtkonvexen Polyedern und betrachtet zumeist rechteckige Hindernisse. Bei der Auswahl der genauer betrachteten Probleme waren wir bemüht einerseits einen Querschnitt durch die Gesamtproblematik zu erzielen, und andererseits einige herausragende Ergebnisse zu präsentieren. Die folgende Tabelle gibt einen groben Überblick. Die eingerahmten Felder werden im weiteren genauer betrachtet werden. Die angegebenen Komplexitätsschranken wurden in dieser Form in der Literatur gefunden, nähere Angaben finden sich dort. (D: deterministisch,R: randomisiert).

Online-Erkunden einer unbekannten Umgebung

Fähigkeiten / Probleme		rein taktil		unbeschränktes Sehen		beschränktes Sehen
		allgemein	rechtwinklig	allgemein	rechtwinklig	
Erstellen einer Karte (mit Hindernissen)	D			133 (+O(k)) (+O($\sqrt{k\alpha}$))	(6)√2 (+O(k))	((17)?) ShortCut auf VisibilityGraph
	R				5/4 √2 (+O(k))	
Finden eines kurzen Weges	D		O($\sqrt{n/k}$)			
	R					
Durchdringen eines Hindernisparcours	D	Ω(√n)	Θ(√n)	Ω(√n)		
	R	O($n^{3/4}$)	O(√n)			
Erreichen eines Zielpunktes	D	O(√n)	(1.5) 1+3/5k $2^{c\sqrt{\lg n}}$		Θ(log n)	
	R			O(√n)		

DFS

Abbildung 5.4.1 (Uebersicht)

5.5 Bounce and Find

5.5.1 Untere Grenzen

Satz 5.5.1 (Lower Bound I) *Es gibt keinen kompetetiven Algorithmus für Bounce and Find, für einen offensichtlichen Gegner.*

Beweis: Man betrachte folgendes Beispiel: Gegeben ein Startpunkt s und eine Ziellinie t in einem ausreichend großen Gebiet A. Sei z.B. x der Nullpunkt in einem Koordinatensystem und t die Gerade $x = n$. Dann wartet der Gegner jeweils, bis der Sucher eine x-Koordinate zum ersten Mal erreicht und plaziert dann ein Hindernis H_i der Höhe n vor diesem, so daß dieser einen Umweg von $\frac{n}{2}$ machen muß, um das Hindernis zu umgehen. Zum erreichen der Ziellinie muß der Sucher also einen Weg der Länge $\Omega(n^2)$ zurücklegen. Nach dem Schubfachprinzip gibt es aber eine y-Koordinate k im Intervall $[0, n\sqrt{n}]$, die nur $\sqrt{n}$ Hindernisse trifft, da jedes der n Hindernisse n ganzzahlige Koordinaten überdecken kann. D.h. indem man zunächst zum Punkt$(0, k)$ geht, dann entlang der $(z, 0)$-Linie $\sqrt{n}$ Objekte mit einem Aufwand von $\frac{n}{2}$ umgeht, was zu einem Gesamtaufwand von $O(n\sqrt{n})$ führt, und somit einen kompetetiven Koeffizienten von $O(\sqrt{n})$ erzwingt.

Korollar 5.5.1 (Lower Bound I') *Das gleiche gilt natürlich für jede Art von schmalem, hohem Hindernis*

Stellt man also fest, daß der allgemeine Fall unlösbar ist, so betrachte man zunächst den einfachst möglichen: Quadratische Hindernisse (Aspect Ratio 1):

Korollar 5.5.2 (Lower Bound II) *Für quadratische Hindernisse kann man eine Competetive Ratio von $\frac{3}{2}$ nicht unterschreiten.*

Beweis: Es genügt dieselbe Konstruktion wie oben, es kann nun garantiert werden, daß der Sucher wenigstens die Strecke $n * 1.5$ zurücklegen muß, während der Optimale Weg die Länge $n + O(\sqrt{n})$ hat.

5.5.2 45-Grad-Heuristik

Lemma 5.5.1 (45-Grad) *Liegen Start und Zielpunkt relativ zur Ausrichtung der Hindernisse im Winkel von $45°$ zueinander, kann man einen kompetetiven Faktor von $\sqrt{2}$ erreichen.*

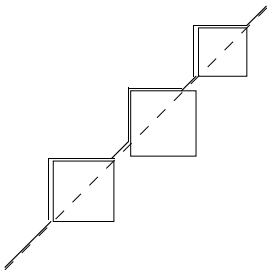

Abbildung 5.5.1 (45-Grad)

5.5.3 Bias-Heuristik

Als erstes einfaches Beispiel sei ein Verfahren angeführt, daß (im Grenzwert) für quadratische Hindernisse gleicher Seitenlänge (o.B.d.A. 1) einen kompetetiven Faktor von 1.5 garantiert.

`Algorithmus 5.5.1 (Bias-Heuristik)` *(Papadimitrou & Yanakis)*

Zu Begin sei $\beta := \epsilon := \frac{1}{\sqrt{n}}$
repeat
  Solange kein Hindernis berührt wird:
    Bewegung parallel zur Start-Ziel-Linie.
  SONST:
    Ist das näher zur Start-Ziel-Linie gelegene Ende
    des Hindernisses weniger als $\frac{1}{2} + \beta$ entfernt,
      gehe zu diesem Ende und verringere β um ϵ
    SONST
      Wähle das andere Ende und erhöhe β um ϵ
until Ziellinie ist erreicht.
Bewegung senkrecht zur Start-Ziel-Linie zum Zielpunkt.

Satz 5.5.2 (Bias-Heuristik) *Die untere Schranke von $\frac{3}{2}$ wird durch den "Bias-Algorithmus" asymptotisch erreicht.*

Beweis: Es sind folgende Bedingungen erfüllt:

- An jedem der n berührten Hindernisse wird maximal die Strecke $\frac{1}{2} + \beta$ zurückgelegt.
- Der Roboter entfernt sich maximal $\mathcal{O}(\sqrt{n})$ von der Start-Ziel-Linie, da für $\beta = \frac{1}{2}$ immer die näher an der Start-Ziel-Linie gelegene Richtung gewählt wird.
- Mit jedem Schritt, mit dem näher an die x-Achse herangehen und eine Strecke von mehr als $\frac{1}{2} + \epsilon$ zurücklegt wird, muß der Roboter sich zuvor entfernt haben, also eine Strecke von weniger als $\frac{1}{2} - \epsilon$ zurückgelegt haben. So daß bei jedem Hindernis maximal ein zusätzlicher Aufwand von ϵ verbleibt.

Als Schranke für die Gesamtstrecke erhalten wir somit:

$$n + \frac{1}{2}n + n * \frac{1}{\sqrt{n}} + \mathcal{O}(\sqrt{n})$$

Die Länge eines kürzesten Weges von s nach t muß aber mindestens die Länge n haben, wodurch man als asymptotische Grenze 1.5 erhält.

5.5.4 Beliebig orientierte rechtwinklige Hindernisse

Im folgenden wird ein Algorithmus von Karpinski [17] wiedergegeben, der für das Problem beliebig orientierter Rechtecke mit Breite mindestens 1 eine Schranke von $n^{\frac{3}{4}}$ garantiert.

`Algorithmus 5.5.2 (Karpinski-Grundform)` *(Karpinski, Berman)*

Alle im folgenden angegebenen Prozeduren werden von folgender Hauptroutine aufgerufen:

```
set(m,l);
repeat
        rand.set(r)
        for phase.no := 1 to m do
                for attack.no :=-(m-1) to m-1 do
                        lower := line {y = (i − 1)l}
                        upper := line {y = (i + 1)l}
                        central := line {y = il}
                        attack(attack.no);
        l := 2 ∗ l
forever
```

Ein Durchlauf der äußersten Schleife sei als STAGE bezeichnet, ein Durchlauf der "phase"-Schleife als PHASE
Zunächst sei ein deterministischer Algorithmus für orientierte Rechtecke betrachtet:
In set(m,l) seien gewählt: $m := \lceil \sqrt{n} \rceil$ und $l = m$.

`Algorithmus 5.5.3 (Karpinski I)`

```
Attack(i):repeat
            Gehe bis zum nächsten Hindernis.
            if es erstreckt sich von lower bis upper
              exit.
            else
              umgehe es um die nähere Kante zurück zu central
            forever
```

Satz 5.5.3 (Karpinski I) *Durch* `Karpinski I` *wird ein kompetetiver Faktor von* $\mathcal{O}(m) = \sqrt{n}$ *erreicht.*

der Satz folgt unmittelbar aus den folgenden Lemmata:

Lemma 5.5.2 (Obere Schranke) *Der Weg des Suchers bis einschließlich* STAGE l *ist beschränkt durch* $\mathcal{O}(m^2 l)$.

1. Die außerhalb der Attacken zurückgelegte Strecke ist beschränkt durch $\mathcal{O}(6m^2 l)$

2. Blockiert ein Objekt mehrere Phasen, so wird abgebrochen. Ansonsten erreicht man In jedem $2l$. Schritt einer Attacke (summiert über einen ganzen loop) einen Fortschritt von mindestens 1.Es können also in den jeweiligen Attacken also höchstens $nl = m^2 l$ Schritte auftreten.

Lemma 5.5.3 (Untere Schranke) *Wurde das Ziel in* STAGE l *nicht erreicht, so hat ein kürzester Weg mindestens die Länge* $\Omega(ml)$

Beweis:
Fall 1: Alle PHASEN dieses STAGE wurden vom selben Hindernis aufgehalten. Dann muß dieses Hindernis die Länge ml haben, jeder kürzeste Weg muß also wenigstens $\frac{1}{2}ml$ lang sein.
Fall 2: Da in m Phasen das Ziel nicht erreicht wurde, muß es wenigstens m disjunkte "Phasenbarrieren" (vgl. Abbildung) geben. Offenbar muß jeder kürzeste Weg entweder die Barrieren umgehen oder jeweils eine Distanz von l zurücklegen, um eine jeweilige Barriere zu überwinden. Da jede Barriere die Höhe ml hat, und m vorhanden sind, sind hierzu mindestens ml Schritte notwendig.

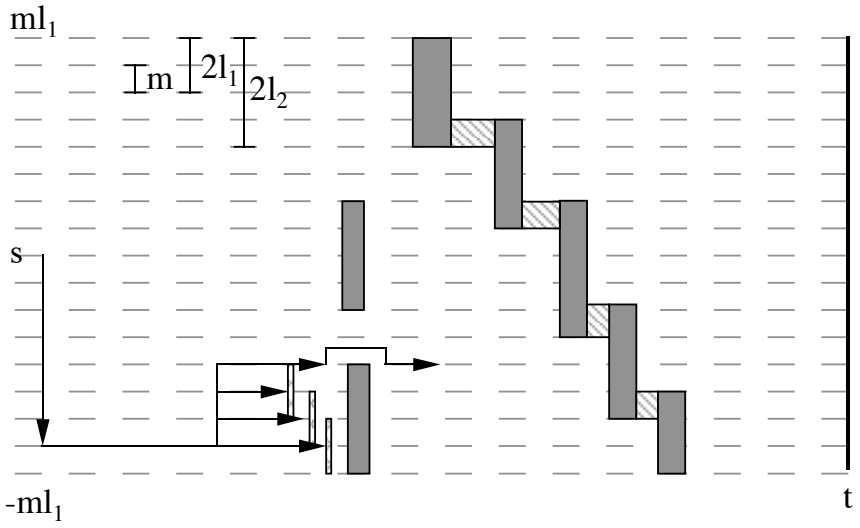

Abbildung 5.5.2 (Karpinski I)

Beim Übergang auf bliebig orientierte Rechtecke wird nun davon ausgegangen, daß der Roboter,wenn er auf ein Hindernis trifft, dessen Steigung spüren kann. Der Roboter kann sich entlang eines solchen Hindernisses dann entweder zum Ziel hin und von der momentanen "Angriffslinie" weg oder vom Ziel weg zur momentanen Angriffslinie hin bewegen.

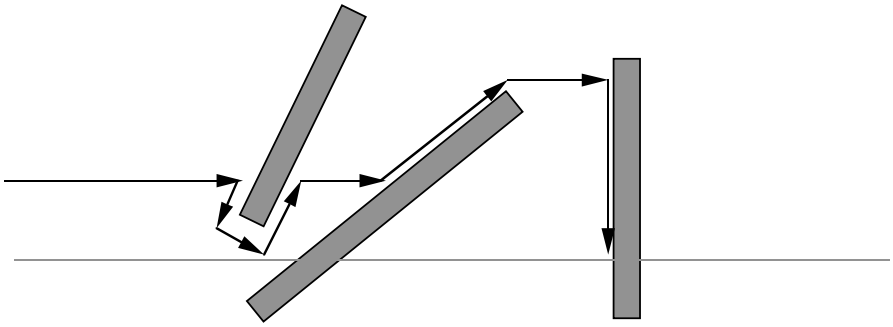

Abbildung 5.5.3 (Karpinski II)

Man kann nun den Wert, bei dem die Entscheidung getroffen werden soll, mit dem Zufallsgenerator so bestimmen, daß eine gute Schranke garantiert werden kann. Hierzu wähle man in set(m, l), bzw. rand.set(r):

$$m := \lceil n^{0.25} \rceil \quad l := m^3 \quad r \text{ aus } \{m^2 + 1, \cdots 2m^2\}$$

Das oben beschriebene Vorgehen wird durch folgende Subroutine realisiert:

`Algorithmus 5.5.4 (Stabilize(line))`

stabilize(line):
repeat
  if nicht an einem Hindernis
    then gehe Richtung Ziel
  else if feel $>$ r **then**
    folge dem Hindernis Richtung Ziel
  else folge dem Hindernis Richtung **line**
until line ist erreicht, ein Hindernis berührt und feel $\leq$ r

Unter Verwendung dieser Routine kann man nun die neue Angriffsvariante angeben:

Attack(i):
repeat
  if in diesem Stage zurückgelegter Weg $> cm^4 l$ **then exit**
  stabilize(central);
  if Hindernis sperrt von **lower** bis **upper**
    exit
  else
    folge dem Hindernis um die nähere Ecke zurück zu **central**
forever

Satz 5.5.4 (Karpinski II) *der Algorithmus* `Karpinski II` *hat eine Kompetetivität von* $\mathcal{O}(m^3) = \mathcal{O}(n^{\frac{3}{4}})$

Beweis: Es sei L die Länge des kürzesten Weges.

- in STAGE l wird maximal cm^4l zurückgelegt.
- Wird das Ziel in STAGE l nicht erreicht und der STAGE korrekt beendet, ist der erwartete Weg länger als ml.
- Es muß also die Länge der Wege abgeschätzt werden, die in STAGE mit $ml > L$ zurückgelegt werden, in denen dieses wegen eines Abbruchs nicht "bemerkt" wurde.
- Die Wahrscheinlichkeit, daß ein STAGE nicht korrekt beendet wird, ist kleiner als $\frac{1}{4}$.
- Für alle diese STAGES verdoppelt sich also die erlaubte Pfadlänge, während die Wahrscheinlichkeit, diesen STAGE überhaupt noch auszuführen um den Faktor 4 sinkt (s.u.). D.h jeder weitere (wegen eines Abbruchs aufgerufene) STAGE trägt im Erwartungswert höchstens die Hälfte des Vorangegangenen zur erwarteten Pfadlänge bei.
- Summiert über die möglichen STAGES ergibt sich also, daß die Länge des Weges, der in STAGES zurückgelegt wird, für die abgebrochen wird, obwohl der kürzeste Pfad kleiner als ml ist, als die doppelte ($\sum_{i=0}^{\infty} \frac{1}{2} = 2$) Länge des letzten STAGES, in dem diese Bedingung noch erfüllt war.

Im folgenden wird der Beweis, daß jeder Stage mit der Wahrscheinlichkeit $\frac{3}{4}$ beendet wird kurz skizziert. Für den exakten Beweis siehe [17].

- Die Bewegung des Roboters ist auf das Trapezoid $(-2m^3, \pm ml), (n, \pm(m+1)ml)$ beschränkt. Dies ergibt sich durch Nachrechnen: In Richtung **Central** kann man höchstens ml zurücklegen, und dabei um $rm < 2m^3$ zurückgehen. Angefangen von $(-rm, ml)$ kann man höchstens $(rm + m^4)$ nach rechts und somit $(l/r)(rm + m^4) \leq (1+m)ml$ nach oben gehen.
- Mit Wahrscheinlichkeit $3/4$ ist die Gesamtlänge der Hindernisse mit **feel** $> r$ durch $6m^4l$ begrenzt: Die Gesamtlänge aller Hindernisse ist durch die Fläche begrenzt, da man eine Dicke der Hindernisse von wenigstens 1 voraussetzt. Der Erwartungswert ist $1.5m^4l$, woraus mit der Chebyshevschen Ungleichung eine Abweichung um mehr als 4 mit einer Wahrscheinlichkeit $\leq 1/4$ auftritt.
- Wenn man zuläßt, daß der Robot Hindernissen mit $feel > r$ im Ausmaß von $6m^4l$ begegnet, so muß er höchstens $30m^4l$ Schritte zurücklegen, um das Ziel zu erreichen. Dies ergibt sich durch eine aufwendige Fallanalyse, die in der Originalquelle nachzulesen ist.
- Wählt man also $c := 30$ im Programm, so ist der Satz korrekt.

Karpinski und Berman verallgemeinern ihre Ergebnisse noch auf beliebige konvexe Polyeder (s. [17]).

5.6 Map and See

5.6.1 Allgemeines und Offline-Fall

In diesem Kapitel soll der "Map und See" Fall mit der zusätzlichen Annahme daß $I = \emptyset$ ist, d.h. das es keine Hindernisse gibt, betrachtet werden. Aufgabe ist es, ausgehend von einem gegebenen Startpunkt am Rand die gesamte Hülle des Polygons kennenzulernen, und dabei einen möglichst kurzen Weg zurückzulegen. Betrachtet man zunächst einmal den Offline-Fall, so ergibt sich intuitiv die in der Literatur fast überall konsistente Definition von "Line-Segments" und "Essential Segments":

Definition 5.6.1 (Line-Segment) *Verlängert man eine beliebige Kante des Polygons solange, bis sie das Polygon verläßt, so bezeichnet man die abgetrennten Teile des Poygons (so vorhanden) als Line-Segmente. Muß ein solches Segment betreten werden, um alle Punkte des Polygons betrachten zu können, so nennt man es ein notwendiges Line-Segment*

Definition 5.6.2 (Essential Segment) *Jedes notwendige Segment, daß nicht automatisch betreten werden muß, wenn man ein (bestimmtes) anderes notwendiges Segment besuchen will, wird als "Essential Segment" bezeichnet.*

In der folgenden Abbildung sind - unter der Annahme, daß s der Startpunkt ist - die notwendigen Segmente liniert, die essentiellen dunkel markiert.

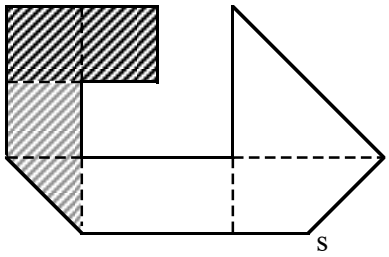

Abbildung 5.6.1 (Line Segments)

Im folgenden wird der rechtwinklige Offline-Fall (nach [10]) anhand eines Beispiels beschrieben. Der allgemeine Fall folgt dem gleichen Prinzip, ist aber mit erhöhtem technischem Aufwand verbunden.Offenbar ist eine Tour, die im Uhrzeigersinn die essentiellen Segmente besucht, optimal, da man eine Tour, die sich selbst schneidet, durch eine kürzere Tour ersetzen kann, bei der dies nicht der Fall ist.

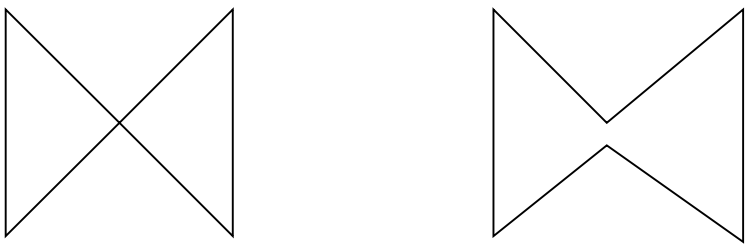

Abbildung 5.6.2 (Selfcut)

Es verbleibt also lediglich, den jeweiligen Punkt zu bestimmen, bei dem man das entsprechende Segment berührt. Man betrachte also das Problem, in diesem Polygon eine optimale Tour zu bestimmen.

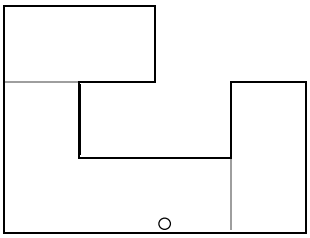

Abbildung 5.6.3 (Problemstellung)

Zunächst identifiziert man die essentiellen Segmente, und bestimmt das Polygon, das nach deren Entfernen entsteht. Das Problem kann nun so aufgefaßt werden, daß die grau schattierten Kanten des reduzierten Polygons betrachtet werden sollen.

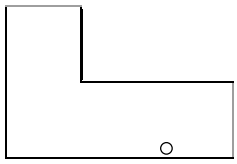

Abbildung 5.6.4 (Reduktion)

Definition 5.6.3 (Essentielle Kante) *Unter einer Essentiellen Kante versteht man eine Kante des Reduzierten Polygons, die an der Stelle neu entstanden ist, an der ein essentielles Segment entfernt wurde.*

Die verwendete Lösung entspricht der Vorstellung des "Lichtweges" in der Physik. Man spiegelt das Polygon an den zu besuchenden Kanten (im Uhrzeigersinn) und bestimmt dann einen kürzesten Weg vom Startpunkt zu seinem Bild im letzten erzeugten Spiegelbild. Es ergibt sich leicht aus der geometrischen Anschauung, daß dieser Weg die notwendigen Kanten genau dort schneidet, wo sie im Ursprungsbild berührt werden müßten.

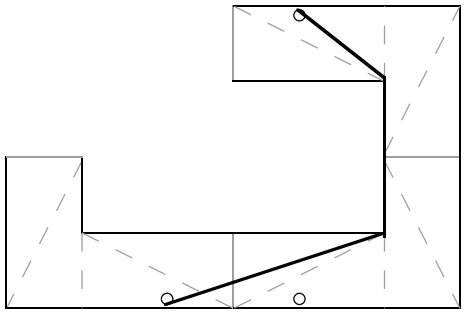

Abbildung 5.6.5 (Spiegelung)

Transformiert man dies wieder in das Ausgangsproblem zurück, so erhält man eine optimale Tour (auch Optimum-Watchman-Tour genannt).

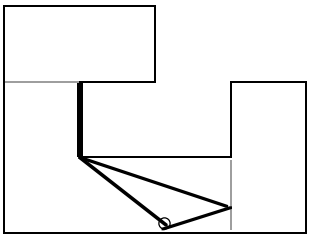

Abbildung 5.6.6 (Loesung)

Bei der Verallgemeinerung auf einfache Polygone tritt zusätzlich das Problem auf, daß man bei mehreren überlappenden essentiellen Segmenten entscheiden muß, welches man besucht. Siehe hierzu [11],[9].

5.6.2 Ein $\sqrt{2}$-kompetetiver Algorithmus für den rechtwinkligen Fall.

Schränkt man sich zunächst auf den Fall eines rechteckigen Polygons ein, so ist ein guter Algorithmus bekannt [12],[13]. Die wesentliche Idee besteht darin, von der euklidischen (L_2) auf die Betragsnorm ($L1$) zu wechseln, da es unter dieser irrelevant ist, an welcher Stelle man auf die essentielle Kante trifft.

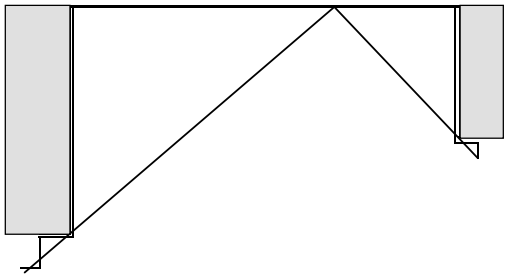

Abbildung 5.6.7 (L2-Norm)

Der Algorithmus verhält sich ansonsten intuitiv, d.h. er untersucht die jeweils nächste unbekannte Position.

Definition 5.6.4 ($l(f), r(f)$) *Für einen Punkt f auf dem Rand des Polygons ist $r(f)$ ($l(f)$) diejenige Kante, die f im (gegen den) Uhrzeigersinn folgt.*

`Algorithmus 5.6.1 (Rect)` *(Deng,Kameda,Papadimitriou)*

Sei f derjenige Punkt des Polygons, der das (in Uhrzeigerrichtung) Ende des bekannten Teils des Polygons darstellt, das den Startpunkt enthält. Der Algorithmus folgt nun diesen drei Schritten, wobei x der momentane Ort und x_0 der Startpunkt ist:

1. f kann von x aus gesehen werden, und ist eine 270-Grad Ecke: Gehe senkrecht auf $r(f)$ zu, bis es oder seine Verlängerung erreicht wird. Sollte dabei ein weiteres Hindernis getroffen werden, bewege man sich solange parallel zu $r(f)$ bis die Sicht wieder hergestellt ist.

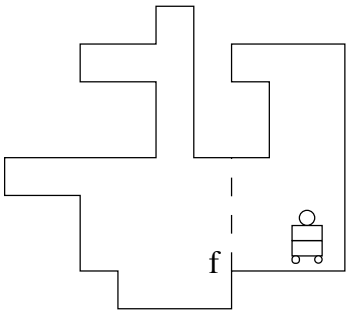

Abbildung 5.6.8 (Deng A1)

2. f ist innerer Punkt einer Kante $r(f) = l(f)$. Dann gibt es eine die Sicht zu f blockierende Ecke b (bei mehreren wähle die nächste). Man verhalte sich zu $l(b)$ wie zu $r(f)$ unter 1.. Sieht man dann f immer noch nicht, geht man entlang von $l(b)$.

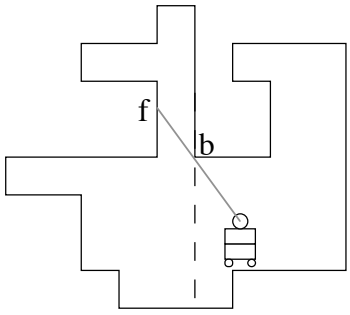

Abbildung 5.6.9 (Deng A2)

3. f kann von x aus nicht gesehen werden. Je nachdem ob f beim lezten Sehen gemäß 1. oder 2. wahrgenommen wurde, bewege man sich (wo schon bekannt auf dem kürzesten Weg) zu $r(f)$ bzw. $l(b)$

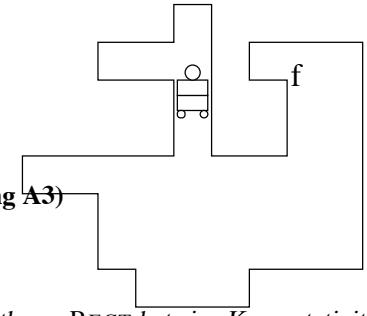

Abbildung 5.6.10 (Deng A3)

Satz 5.6.1 (DKP) *Der Algorithmus* RECT *hat eine Kompetetivität von* 1 *bezüglich der Betragsnorm*

Korollar 5.6.1 (DKP ') RECT *ist* $\sqrt{2}$*-kompetetiv bezueglich der euklidischen Norm.*

Beweis des Satzes: Es wird im folgenden nur eine Beweis-Skizze gegeben. Für den vollständigen und recht umfangreichen Beweis siehe: [13]
Es bleiben nach obigen Betrachtungen noch zwei Dinge zu zeigen:

1. Zwischen den Segmenten wird ein in der L_1-Metrik optimaler Weg gewählt. Zunächst ist nur klar, daß ein Weg gewählt wird, der minimal oft seine Richtung ändert. Man kann nun durch Induktion zeigen, daß in rechtwinkligen Polygonen ohne Hindernissen ein kürzester Weg in dieser "Link-Distanz" auch ein kürzester Weg in der L_1 Metrik ist.

2. Es werden die richtigen Segmente besucht.

   - jedes Segment, das besucht wird ist notwendig, da es an der entsprechenden Stelle etwas liegen muß, was noch nicht gesehen wurde
   - Alle essentiellen Segmente werden besucht, da der Algorithmus erst terminiert, wenn es keine unbekannten Stellen mehr gibt.
   - Der Besuch notwendiger Segmente vergrößert (in der L_1-Metrik) den zurückgelegten Weg nicht, da jedes dieser Segmente entweder selbst essentiell ist, oder ein essentielles Segment enthält.

3. Die essentiellen Segmente werden in der richtigen Reihenfolge besucht. Dies ist nicht generell der Fall, denn es kann wie man in Abbildung 3 erkennt vorkommen, daß sich zwei essentielle Segmente schneiden, und somit die im reduzierten Polynom gegebene Reihenfolge, von zwei entgegengesetzt liegenden Kanten erzeugt werden, die Kanten also nach Konstruktion in umgekehrter Reihenfolge besucht werden. (Die x_i und die y_i beschreiben die beiden verschiedenen Wege.)

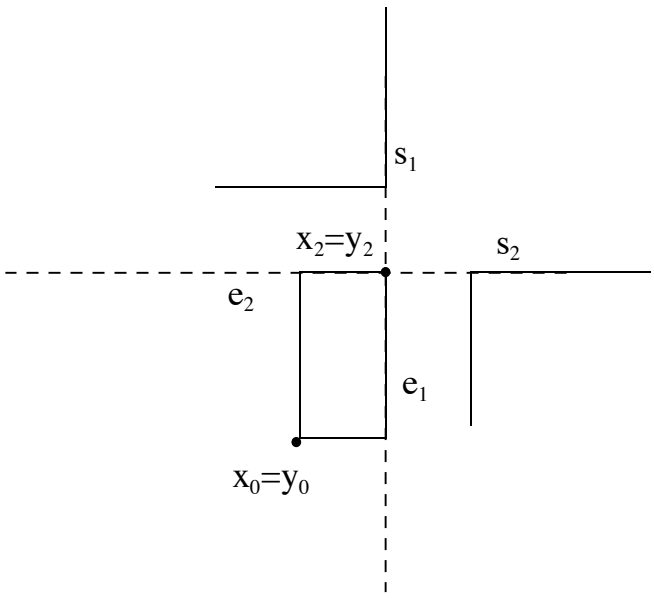

Abbildung 5.6.11 (Segmentschnitt)

Es ist aber der übernächste Punkt, der in beiden Reihenfolgen besucht wird wird, auf jeden Fall der Schnittpunkt der beiden Segmente, d.h.:

$$x_i \neq y_i \Rightarrow (x_{i+1} = y_{i+1} \wedge x_0 = y_0).$$

Es genügt also zu zeigen, daß die Wege, die zwischen x_{i-1}und x_{i+1} bzw. y_{i-1} und y_{i+1} zurückgelegt werden gleichlang sind (in der L_1- Metrik). Nun kann man aber leicht sehen, daß nach Konstruktion x_i auf dem kürzesten x_{i-1}, x_{i+1}-Weg liegt, und y_i auf dem kürzesten y_{i-1}, y_{i+1}-Weg liegt. Damit folgt die Gleichheit der Abstände. Konstruktionen wie in [20] lassen vermuten, daß $\sqrt{2}$ optimal ist.

5.6.3 Ein 133-kompetetiver Algorithmus für nicht rechtwinklige einfache Polygone

Der folgende Abschnitt gibt im wesentlichen den Algorithmus aus [15] wieder. Dieser Algorithmus verzichtet auf das Aufspüren von essentiellen Kanten und strebt danach, bestimmte Ecken des Polygons zu untersuchen (wobei die zugehörigen Segmente in einem untersucht werden.). Es ist klar, daß, wenn man auf einer engen Straße eine Links- und eine Rechtskurve einander folgen läßt, der kürzeste Weg über diese Strecke die beiden Ecken der Kurven miteinander verbinden muß. Will man diese Eigenschaft ausnutzen, so erweisen sich folgende Definitionen als sinnvoll:

Definition 5.6.5 (Links-Ecke) *Eine Links (rechts) Ecke ist eine Ecke mit einem Winkel größer 180 Grad die vom Baum der kürzesten Pfade berührt wird, und die von der Wurzel des Bauems aus (dem Startpunkt es Robots aus) nach links (rechts) abbiegt.*

Definition 5.6.6 (Door) *Eine "Tür" ist eine Kante im Kürzeste-Wege-Baum, die eine Ecke vom Links(Rechts)-Typ mit ihrem ersten Nachfolger des entgegengestezten Typs verbindet.*

Definition 5.6.7 (Hinge) *Eine "Hinge" ist die im Baum höher liegende Ecke einer Tür.*

Definition 5.6.8 (Right-Door) *Eine Rechts-Tür ist eine Tür, deren Hinge eine rechts-Ecke ist.*

Der folgende Algorithmus beruht darauf, das Polygon in einzelne Räume zu partitionieren, die jeweils durch Türen getrennt sind. Dabei "erbt" der jeweilige Raum den Typ der Tür, durch die er betreten wurde, wird aber ansonsten durch Türen des entgegengesetzten Typs begrenzt. Die Partitionierung wird online vorgenommen, indem die jeweiligen Türen erst dann passiert werden, wenn der Raum vollständig untersucht wurde. Nachdem ein jeweiliger Raum abgearbeitet ist, werden nacheinander alle Räume untersucht, die hinter Türen liegen, die nicht mit der Eingangstür identisch sind. Diese Aufgabe wird von der Procedure SAM wahrgenommen. Sie bedient sich hierbei der Scout-Funktionen, die dazu dienen, die Türen aufzuspüren. Da an jeder unbekannten Stelle eine Tür liegen könnte wird hierbei gleichzeitig der entsprechende Raum völlig kartographiert. Die Explore-Prozeduren dienen dazu, jeweils konkrete Wege zurückzulegen.

`Algorithmus 5.6.2 (SAM)` *(Hoffmann,Icking,Klein,Kriegel)*

```
var HingeList;
HingeList := Scout(x);
forall y of HingeList in counterclockwise order do
   MarchTo(y);
   SAM(y);
MarchTo(x);
```

Die folgende Abbildung veranschaulicht das Vorgehen von `SAM`. Die eingezeichneten Wege werden dabei von MarchTo zurückgelegt.

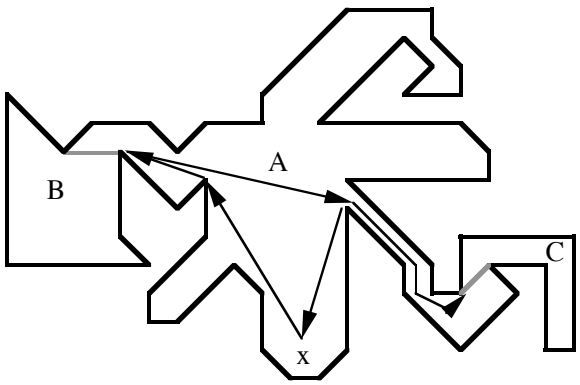

Abbildung 5.6.12 (SAM)

Scout dient dazu, die Hinges der jeweiligen Räume zu bestimmen. Im folgenden soll davon ausgegangen werden, daß es sich um einen Rechts-Raum handelt, d.h. es wird nach Links-Türen gesucht. Der andere Fall ist symmetrisch. Es geht also darum alle aufeinanderfolgenden Links-Rechts Eckenkombinationen zu bestimmen. Scout verwendet hierzu die Subroutinen Scout1 und Scout2. Die erste dient dazu, zunächst alle Links-Ecken, die hinter einer Rechts-Ecke verborgen sind, zu finden, so daß die zweite dann mit einer korrekten Liste aller Links-Ecken starten kann. Diejenigen Ecken, die von einer Prozedur zur anderen übergeben werden sind in 5.6.3 markiert.

`Algorithmus 5.6.3 (Scout)`

var LList, HList;
LList:=Scout1(x);
Hlist:=Scout2(x,LList);
return HList;

`Algorithmus 5.6.4 (Scout1(x))`

var RList,LList,Basepoint;
insert all right-vertices visible from x into RList;
whileRList $\neq \emptyset$ **do**
    Explore1(RList,BasePoint,LList);
    MarchTo(closest point to x_0 on Cut[7] of First(RList));
    BasePoint:=current position;
MarchTo(x);
return LList

Die Funktion `Scout2` ist analog aufgebaut, lediglich wird die von ihm verwendete LList (in `Scout1` *RList*) mit der Ausgabe von `Scout1` initialisiert.

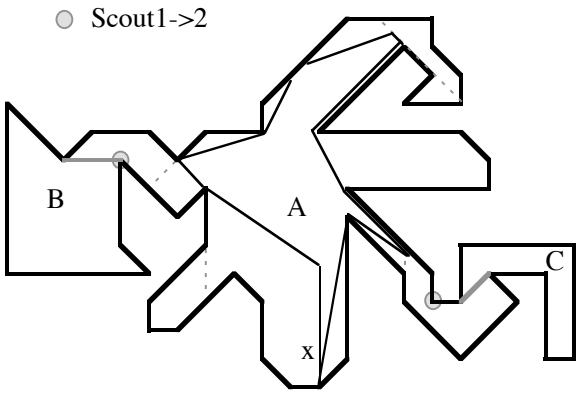

Abbildung 5.6.13 (Scout)

Die beiden Explore-Routinen übernehmen die Bewegung zwischen den einzelnen Base-Points (siehe Pseudocode). Dabei wird die nächste Ecke so angegangen, daß bei Berühren des zugehörigen Segments der Abstand zum Ausgangspunkt minimiert wird. Zu diesem Zweck folgt man einem Kreis, der seinen Mittelpunkt im Mittelpunkt der Strecke vom Ausgangspunkt zur Ecke hat. Mit dem Satz des Thales ergibt sich somit, daß in dem Punkt, in dem man das Segment berührt ein rechter Winkel zwischen Segment und gerader Verbindung zum Basepoint entsteht. Sollten dabei Hindernisse den Weg blockieren, gehe man auf geradem auf das Ziel zu, bis das Hindernis verschwindet. Sollte sich LList während des Vorgehens verändern, schwenkt man sofort auf das neue Ziel um.

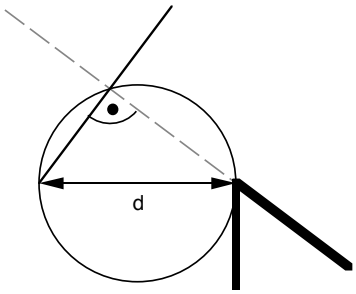

Abbildung 5.6.14 (Explore)

Hier wird nun die Prozedur `Explore2` beschrieben, die an der entsprechenden Stelle von `Scout2` aufgerufen wird, wie `explore1` in `Scout1`. `Explore1` ist analog aufgebaut. Es erzeugt lediglich andere Listen als `Explore2`

`Algorithmus 5.6.5 (Explore2)`

MarchTo(First(LList)), until visible;
repeat
    Follow Circle (oben beschrieben)
until First(LList) explored or right Vertex hidden by First(LList) discovered
if discovered **then** insert First(LList) into HList;
meanwhile do maintain LList

Satz 5.6.2 (HIKK) *Der Algorithmus SAM ist* 144*-kompetetiv.*

Wiederum soll nur eine Beweisskizze gegeben werden.

- Als Abschätzung des von Explore zurückgelegten Weges gegenüber der kürzestmöglichen Strecke zwischen BasePoint und Cut ergibt sich, wenn keine Hindernisse auftreten:

$$\frac{\text{Halbkreis}}{\text{Durchmesser}} = \frac{\pi}{2}$$

Durch Fallanalyse blockierender Hindernisse erhält man: $\frac{\pi}{2} + 1$.

- Abschätzung des zurückgelegten Weges zwischen zwei Base Points gegenüber dem kürzesten Weg: Der Weg des Robots setzt sich zusammen aus dem von `Explore` zurückgelegten Weg und dem Wandern auf dem Rand des Segments (Cut). Die auf dem Schnitt zurückgelegte Strecke läßt sich wie dargestellt auf den kürzesten Weg projizieren: Wegen

$$L(\text{BasePoint}, a) \leq K(\text{BasePoint}, c) \text{ und } |ac| < L(\text{BasePoint}, c)$$

gilt: $\frac{L(\text{Roboterweg})}{\text{kürzester Weg}} \leq \frac{pi}{2} + 2$.

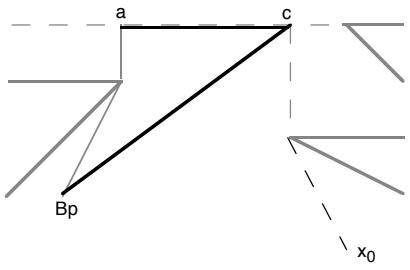

Abbildung 5.6.15 (ac)

- Abschätzung des Verhältnisses der Länge der relativen konvexen Hülle der Sequenz der BasePoints gegenüber der Weglänge:

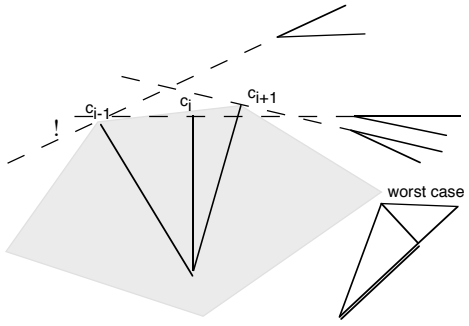

Abbildung 5.6.16 (Huelle)

Das kleine Bild am Rande zeigt, daß der "worst case" genau für rechtwinklige Dreiecke auftritt: Offenbar muß gelten: $|c_{i-1}c_i| + |c_i c_{i+1}| \leq \sqrt{2}|c_{i-1}c_{i+1}|$.

- Vergleich der relativen Konvexen Hülle mit dem kürzesten Weg, der den Raum kennenlernt: "One can show", dieses Verhältnis ist $\frac{\pi}{2} + 1$.
- Somit ist das Verhältnis der von Scout1 bzw. Scout2 zurückgelegten Strecke gegenüber dem kürzesten Weg, der den entsprechenden Raum kennenlernt beschränkt durch
$$\left(\frac{\pi}{2} + 2\right)\sqrt{2}\left(\frac{\pi}{2} + 1\right) \leq 13$$
- Durch eine weitere Fallanalyse erhält man: Das Verhältnis der Summe der Wege zur Erkundung der einzelnen Räume zur Optimum-Watchman Route (Wopt) ist 5. Es fließt z.B. ein, daß man beim Verlassen eines Raumes die Hinge nicht mehr berühren muß.
- Die Summe der von SAM zurückgelegten Wege ist beschränkt durch $3 * W_{\text{opt}}$

Somit ergibt sich insgesamt :

$$W_{\text{Icking}} \leq ((\underbrace{2}_{\text{Scout 1+2}} * 13 * (5)) + 3) W_{\text{opt}} = 133\, W_{\text{opt}}$$

Es ist allerdings zu vermuten, daß dieses Ergebnis noch verbessert werden kann, da der Robot z.B. im Rahmen von `SAM` manche Strecken mehrfach zurücklegt.

Behauptung 5.6.1 (Polynom mit Hindernissen) *Da sich das Verfahren analog auf das Äußere von Polyedern übertragen läßt, kann man mit diesem Verfahren einen $\mathcal{O}(k)$ kompetetiven Algorithmus zur Erkundung eines Polygons mit k Hindernissen erhalten.*

6 Schlußbemerkung und Ausblick

Die dargestellten Strategien vermitteln, so hoffen wir, einen exemplarischen Überblick über diesen Schnitt der Online-Thematik mit der Graphentheorie bzw. der algorithmischen Geometrie. Das einzige grundlegende Paradigma, daß sich bisher hier findet ist das der Verdopplung (5.3). Ansonsten hat jeder der Autoren seine eigenen Ideen verwandt. Hilfen für anstehende Probleme kann man in diesem Bereich also nur in Form von Impulsen nicht aber von Methoden erhalten. So erkennt man, daß Sichtwechsel, wie sie [13] mit ihrem Übergang zu einer anderen Metrik oder [15] mit dem vorübergehenden Außerachtlassen der Segmente vollziehen, häufig zu neuen Ergebnissen führen können, aber diese Vorgehensweisen sind wohl nicht klassifizierbar. Es bleibt zu hoffen, daß wie bei den meisten Problemgebieten mit zunehmender Erschließung eine Vereinheitlichung der Methoden erreicht werden kann, so daß Problemlösungen auseinander ableitbar werden. Ansonsten droht mit dem zunehmenden Einsatz autonomer Roboter auch dieses Gebiet dem in der Informatik üblichen Problem anheimzufallen:

Einer großen Mengen von Anwendungen auf einer unterentwickelten theoretischen Basis. Die hieraus resultierenden Schwierigkeiten im allgemeinen Engineering sowie insbesondere in der Verifikation dürften sich auf die Betriebssicherheit der Automaten nicht unbedingt positiv auswirken.
Ein weiteres anstehendes Problem ist die Integration der gefundenen Lösungen in die Planungsmodelle der künstlichen Intelligenz. Wie können gewonnene Erfahrungen über andere Gebiete zur Optimierung der Suchstrategien verwandt werden? Wie kann man dennoch eine gute Kompetetivität garantieren? Wird es vielleicht möglich sein online Online-Verfahren neu zu finden, oder ist es andererseits möglich, kognitiven Strukturen eine Bewertungsfunktion zuzuordnen, so daß man einen Kompetetiven Faktor zwischen einem online erlernten System und einem offline vorgegebenen bestimmen und somit vielleicht die Methoden der online-Thematik wiederum im KI-bereich anwenden kann?

7 Literaturverzeichnis

Literatur

[1] R. Baeza-Yates, J. Culberson, and G. Rawlins. Searching in the plane. *Information and Computation*, 106(2):234–252, 1993. Preliminary version in Proc. 1st Scandinavian Workshop on Algorithm Theory, Lecture Notes in Computer Science 318, Springer-Verlag, Berlin, 1988, 176–189. Also Tech. Report CS-87-68, University of Waterloo, Department of Computer Science, October, 1987.

[2] Piotr Berman, Avrim Blum, Amos Fiat, Howard Karloff, Adi Rosen, and Michael Saks. Randomized robot navigation algorithms. In *Proc. 7th ACM-SIAM Sympos. Discrete Algorithms*, pages 75–84, 1996.

[3] A. Blum and P. Chalasani. An on-line algorithm for improving performance in navigation. In *Proc. 34th Symp. of Foundations of Computer Science*, pages 2–11, 1993.

[4] A. Blum, P. Raghavan, and B. Schieber. Navigating in unfamiliar geometric terrain. In *Proc. 23rd Annu. ACM Sympos. Theory Comput.*, pages 494–503, 1991.

[5] Avrim Blum, Prabhakar Raghavan, and Baruch Schieber. Navigating in unfamiliar geometric terrain. IBM Research Report.

[6] M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *Proc. 19th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 132–142, 1978.

[7] H. L. Bodlaender. Complexity of path-forming games. *Theoret. Comput. Sci.*, 110:215–245, 1993.

[8] Prasad Chalasani. *Online Performance Improvement Algorithms*. PhD thesis, School of Computer Science Carnegie Mellon University, 1994.

[9] W. Chin and S. Ntafos. Optimum watchman routes. In *Proc. 2nd Annu. ACM Sympos. Comput. Geom.*, pages 24–33, 1986.

[10] W. Chin and S. Ntafos. Optimum watchman routes. *Inform. Process. Lett.*, 28:39–44, 1988.

[11] W.-P. Chin and S. Ntafos. Watchman routes in simple polygons. *Discrete Comput. Geom.*, 6(1):9–31, 1991.

[12] X. Deng, T. Kameda, and C. Papadimitriou. How to learn an unknown environment. In *Proc. 32nd IEEE Symp. Foundations of Computer Science*, pages 298–303, 1991.

[13] Xiaotie Deng, Tiko Kameda, and Christos H. Papadimitriou. How to learn an unknown environment i: The rectlinear case. Technical Report CS-93-05, York University, 1993.

[14] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear time algorithms for visibility and shortest path problems inside simple polygons. In *Proc. 2nd Annu. ACM Sympos. Comput. Geom.*, pages 1–13, 1986.

[15] Frank Hoffmann, Christian Icking, Rolf Klein, and Klaus Kriegel. A competetive strategie for learning a polygon. In *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms*, 1997.

[16] B. Kalyanasundaram and K. Pruhs. A competitive analysis of nearest neighbor based algorithms for searching unknown scenes. In *Proc. 9th Sympos. Theoret. Aspects Comput. Sci.*, volume 577 of *Lecture Notes in Computer Science*, pages 147–157. Springer-Verlag, 1992.

[17] Marek Karpinski and Piotr Berman. Randomized navigation to a wall through convex obstacles. CS-Report 85118-CS, University of Bonn, 1994.

[18] R. Klein. Walking an unknown street with bounded detour. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 304–313, 1991.

[19] J. M. Kleinberg. On-line search in a simple polygon. In *Proc. of the 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 8–15, 1994.

[20] Alejandro López-Ortiz and Sven Schuierer. Going home through an unknown street. In *Proc. 4th Workshop Algorithms Data Struct.*, volume 995 of *Lecture Notes in Computer Science*, pages 135–146. Springer-Verlag, 1995.

[21] V. J. Lumelsky and A. A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[22] Aohan Mei and Yoshihide Igarashi. Efficient strategies for robot navigation in unknown environment. In *Proceedings of ICALP*, volume 820 of *Lecture Notes in Computer Science*, pages 630–641. Springer-Verlag, 1994.

[23] B. John Oomen, S. Sitharama Iyengar, Nageswara S.V. Rao, and R. L. Kashyap. Robot navigation in unknown terrain using learned visibility graphs. part i: The disjoint convex obstacle case. *IEEE Journal on Robotics and Automation*, 3(RA-3,6), 1987.

[24] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. In *16th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science vol. 372*, pages 610–620. Springer-Verlag, 1989.

[25] D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.