Proseminar Diskrete Strukturen in der Praxis

Thema: **Navigation in unbekannter Umgebung**

Autor: Mathias Budil

Datum des Vortrags: 15. Juli 99

Zusammenfassung:

Die Ausarbeitung behandelt des Thema „Kompetitive Strategien“ sowie als spezielle Anwendungsbeispiele „Navigation in unbekannter Umgebung“ in verschiedenen Formen.
Es wird der Begriff der Kompetitiven Strategie eingeführt und einige Problemstellungen erläutert, die mit solchen Strategien effizient oder sogar optimal gelöst werden können.
Dabei wird die Kompetitivität der einzelnen Strategien auch meist mathematisch bewiesen und eine Abschätzung der Kosten durchgeführt. Anhand von graphischen Darstellungen wird versucht, alles möglichst anschaulich zu vermitteln.
Als Vorlage dienten Auszüge aus [Kle97].

1. Einleitung

Zunächst wird in Kapitel 2. der Begriff der „*kompetitiven Strategie*“ eingeführt.
Kompetitivität wird definiert als Eigenschaft der Strategie, daß ihre Kosten zu denen einer optimalen Strategie in einem linearen Zusammenhang stehen.
Im folgenden Kapitel werden nun Beispiele für Probleme erläutert, die mit derartigen Strategien gelöst werden können.
In Kapitel 3.1 wird eine Strategie für das „*bin packing*“-Problem angegeben und gezeigt, daß diese kompetitiv ist.
Ab Kapitel 3.2 wird dann das Problem „*Navigation in unbekannter Umgebung*“ eingehend behandelt. Darin werden verschiedene Strategien angesprochen, mit denen ein Roboter eine unbekannte Umgebung effizient absuchen kann. Der Suchraum, also die Umgebung des Roboters, wird dabei anhand mehrerer Beispiele immer mehr verallgemeinert.
Als Ausgangsbeispiel dient das Absuchen einer Wand vom Ausgangspunkt in zwei Richtungen, die schematisch als zwei Halbgeraden betrachtet werden (Kap 3.2)
Anschließend wird die Suche auf m Halbgeraden ausgedehnt, die sternförmig vom Startpunkt aus verlaufen (Kap. 3.3).
Im nächsten Kapitel (3.4) werden dann einfache Polygone als Umgebung betrachtet.
Um diese erkunden zu können, wird der Roboter mit einem Sichtsystem ausgestattet (3.4.1).
Zunächst wird eine Kostenabschätzung für allgemeine Polygone durchgeführt (3.4.2), anschließend wird eine Strategie eingehend behandelt, die solche Polygone anhand des sog. „shortest path tree“ absuchen (3.4.3).

2. „Kompetitive Strategie“

Häufig ist man in der Informatik mit Problemen konfrontiert, für die eine ideale Lösung zu finden nur schwer oder überhaupt nicht möglich ist. Dies ist meist der Fall, wenn nur unvollständige Informationen gegeben sind, also nicht alle zur vollständigen Berechnung notwendigen Größen bekannt sind.
Bsp.: Ein Roboter soll den kürzesten Weg zur nächsten Tür suchen, kennt aber seine Umgebung nicht.

Um das Problem zu lösen, müssen also Strategien gewählt werden, die einen höheren Aufwand oder höhere Kosten verursachen, als eine optimale Strategie. Man ist natürlich bemüht, eine möglichst „günstige“ Strategie zu finden, die sowenig Aufwand wie möglich erfordert. Bei obigem Beispiel wird das Ziel des Roboters sein, bei der Suche nach der Tür möglichst geringe Wege zurückzulegen.
Um den Aufwand von solchen nicht-optimalen Strategien „messen“ zu können und so mit anderen zu vergleichen wird dieser mit dem Aufwand einer optimalen Strategie verglichen. Die Eigenschaft der Kompetitivität ist dabei folgendermaßen definiert:

Definition: Eine Strategie *S* ist *kompetitiv* mit Faktor *C*, wenn es eine Zahl *A* gibt, so daß gilt:

$$K_S(P) \leq C \cdot K_{opt}(P) + A$$

Dabei ist *P* ein bestimmtes, frei wählbares Beispiel eines Problems Π, K_s die Kosten der Strategie *S* und K_{opt} die Kosten einer optimalen Lösung. Die Zahl *C* wird „*kompetitiver Faktor*“ genannt und ist ≥ 1. Die reellen Werte *A* und *C* dürfen dabei nur von Π und *S* abhängen, nicht aber von *P*. Es besteht also ein linearer Zusammenhang zwischen den Kosten der gewählten Strategie und der optimalen.

3. Beispiele für kompetitive Strategien

3.1 Ein einfaches Beispiel für kompetitive Strategien ist das *bin packing*:

Gegeben ist eine Folge von zweidimensionalen Objekten $Q_1,...,Q_n$, die alle dieselbe Breite, aber individuelle Höhen h_k haben.
Für diese stehen *n* leere Behälter (engl. *bins*) zur Verfügung, die so breit wie die Objekte sind und die Höhe *H* besitzen, wobei *H* mindestens so groß ist wie die Höhen h_k der Objekte.
Diese Objekte sollen nun in möglichst wenige Behälter verpackt werden.

Bsp.: Acht Objekte $Q_1 - Q_8$, die mit optimaler Platzausnutzung in vier Behälter gepackt wurden.

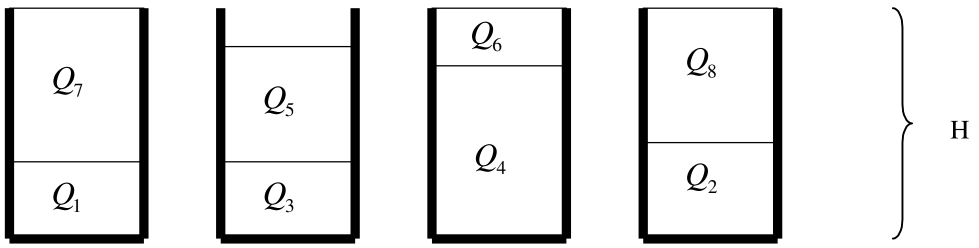

Abbildung 1: *bin packing* mit optimaler Verteilung

Ein möglicher Ansatz zum Verpacken der Objekte ist das *first fit* Verfahren:
Man nimmt die Objekte der Reihe nach in die Hand und legt jedes in den ersten Behälter von links, in dem zu diesem Zeitpunkt noch Platz ist.

Für das obige Beispiel ergibt sich dabei folgende Verteilung:

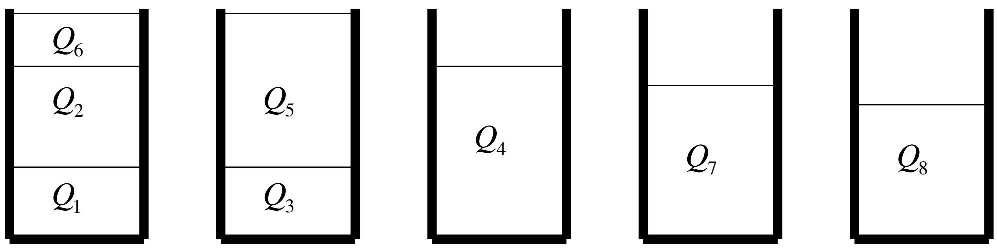

Abbildung 2: Mit dem *first-fit*-Verfahren erzielte Packung

Die *first fit*-Strategie benötigt fünf Behälter (Abb.2), während bei optimaler Verpackung vier Behälter ausreichen (Abb.4). Sie ist also wie zu Erwarten war schlechter, als eine optimale Strategie.

<u>Obere Schranke:</u>
Es gilt, daß die *first-fit*-Strategie höchstens doppelt so viele Behälter benötigt, wie eine optimale Packung der Objekte.

Beweis:
Angenommen, mit dem *first-fit*-Verfahren benötigt man m Behälter um die n Objekte zu verpacken. Alle Behälter bis auf höchstens einen sind dann mehr als zur Hälfte gefüllt. Wären nämlich zwei Behälter i und j nur halb voll, hätte das Verfahren die Objekte von Behälter j schon im i-ten Behälter abgelegt.

Es gilt daher:

$$(m-1)\cdot\frac{H}{2}<\sum_{k=1}^{n}h_k$$

$\Leftrightarrow$

$$m-1<\frac{2}{H}\sum_{k=1}^{n}h_k \le 2\left\lceil\frac{\sum_{k=1}^{n}h_k}{H}\right\rceil$$

Der Ausdruck $\left\lceil\frac{\sum_{k=1}^{n}h_k}{H}\right\rceil$ ist der reine Platzbedarf der n Objekte (in Behältereinheiten).

Die benötigte Anzahl m_{opt} von Behältern, die eine optimale Packung benötigt, ist immer größer oder gleich dieser Zahl. Daraus folgt, daß $m \le 2m_{opt}$.

Für das *bin-packing*-Problem ist die Strategie *first fit* also kompetitiv mit dem Faktor 2.
(Und damit auch jedem Faktor größer als 2)
Nach [Kle97] kann man zeigen, daß diese Strategie sogar kompetitiv mit Faktor 1,7 ist, aber für keinen kleineren Faktor.

3.2 Suche nach einer Tür in einer Wand

Die Navigation eines Roboters in einer unbekannten Umgebung liefert ein gutes Beispiel für kompetitive Strategien. Wir gehen dabei von einem Roboter aus, der sich frei bewegen kann und einen bestimmten Punkt suchen soll, z.B. eine Tür.
Angenommen, der Roboter steht vor einer sehr langen Wand, in der es irgendwo eine Tür gibt, die er finden soll.

3.2.1 Trivialer Ansatz

Ein Möglichkeit für den Roboter wäre, sich eine Richtung „auszusuchen“, und immer in dieser Richtung die Wand abzusuchen. Liegt die Tür in der Richtung in der er sich bewegt, erreicht der Roboter die Tür mit optimalem Aufwand, er legt genau den Weg vom Startpunkt zur Tür zurück. Die Wahrscheinlichkeit, den richtigen Weg einzuschlagen liegt bei 50%. Liegt die Tür aber in der anderen Richtung, findet er die Tür überhaupt nicht.

Da aber nur Strategien in Frage kommen, die in jedem Fall funktionieren, kann dieses Vorgehen nicht angewendet werden.
Nur in eine Richtung zu gehen führt nicht immer zu einer Lösung, der Roboter muß beide Richtungen der Wand absuchen.

3.2.2 Suche mit wechselnder Suchrichtung

Der Roboter muß also seine Bewegungsrichtung immer wieder ändern und abwechselnd den linken und den rechten Teil der Wand nach der Tür absuchen.
Eine mögliche Vorgehensweise ist folgende:
Vom Startpunkt s geht er zuerst einen Meter nach rechts, dreht dann um und geht wieder zu s zurück. Jetzt geht er zwei Meter nach links und dann wieder zurück zu s, anschließend drei Meter nach rechts und so weiter. Die Suchtiefe wird dabei jeweils um einen Meter vergrößert, die Suchrichtung ist immer abwechselnd, vgl. Abb.3

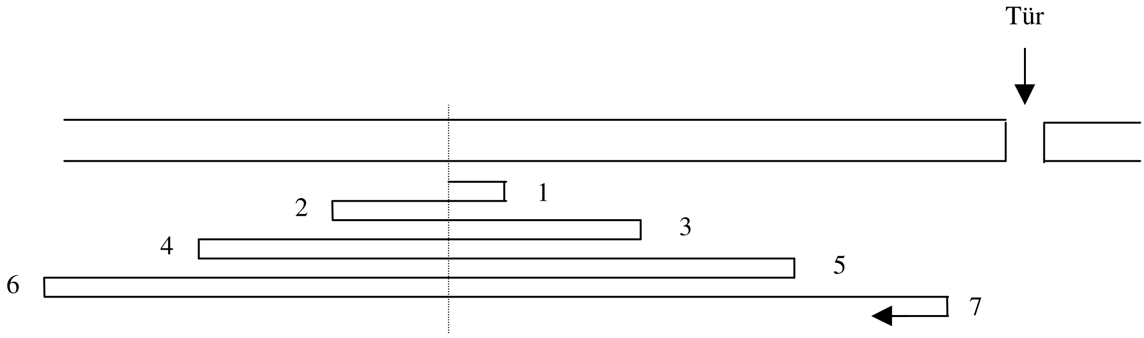

Abbildung 3: Suchtiefenerhöhung bei jeder Richtungsänderung um konstanten Faktor

Mit dieser Strategie wird der Roboter irgendwann die Tür finden.
Angenommen, die Tür befindet sich $d = l + \varepsilon$ Meter rechts vom Startpunkt *s*, dabei ist *l* eine ganze Zahl und $0 < \varepsilon < 1$. Wenn der Roboter *l* Meter nach rechts geht, verfehlt er die Tür gerade noch und kehrt um. Erst wenn er erneut umgedreht hat und wieder nach rechts geht trifft er die Tür.

Er legt folgende Wegstrecke zurück:

$$
\begin{aligned}
& 1 + 1 + 2 + 2 + \ldots + l + l + (l+1) + (l+1) + l + \varepsilon \\
= \quad & 2 \sum_{i=1}^{l+1} i + l + \varepsilon \\
= \quad & (l + 1)(l + 2) + l + \varepsilon \quad \in \Theta(d^2)
\end{aligned}
$$

Dieser Weg, der den Kosten der Strategie entspricht, hängt quadratisch vom Abstand der Tür zum Startpunkt ab. Diese Strategie ist also nicht kompetitiv.

3.2.3 Verdopplung der Suchtiefe

Ein besserer Ansatz ist, die Suchtiefe nicht um 1 Meter zu erhöhen sondern sie nach jedem Richtungswechsel zu verdoppeln. Die Suchtiefen nach rechts sind genau die geradzahligen Zweierpotenzen ($2^0, 2^2, 2^4, ...$), nach links die ungeradzahligen ($2^1, 2^3, 2^5, ...$), siehe Abb.4.

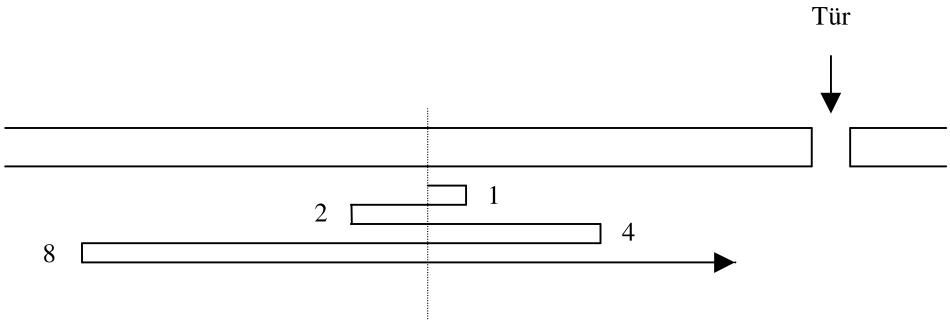

Abbildung 4: Hier wird die Suchtiefe bei jedem Richtungswechsel verdoppelt

Die Tür habe nun den Abstand $d = 2^{2j} + \varepsilon$ und liege rechts vom Startpunkt.
Der Roboter muß also folgenden Weg zurücklegen:

$$2^0 + 2^0 + 2^1 + 2^1 + ... + 2^{2j} + 2^{2j} + 2^{2j+1} + 2^{2j+1} + 2^{2j} + \varepsilon$$

$$= \quad 2 \sum_{i=0}^{2j+1} 2^i + 2^{2j} + \varepsilon$$

$$= \quad 2(2^{2j+2} - 1) + 2^{2j} + \varepsilon$$

$$= \quad 9 \cdot 2^{2j} - 2 + \varepsilon \ < \ 9d$$

Dieselbe Abschätzung gilt auch, wenn die Tür links vom Startpunkt mit Abstand $d = 2^{2j+1} + \varepsilon$ liegt.
Beträgt der Abstand nur $d = \varepsilon \leq 2^1$, kann der Weg folgendermaßen abgeschätzt werden:

$$1 + 1 + \varepsilon < 9\varepsilon + 2 = 9d + 2$$

In allen Fällen ist damit bewiesen, daß die Strategie mit abwechselnder Verdopplung der Suchtiefe kompetitiv mit dem Faktor 9 ist.
Ist die Tür z.B. 100 Meter vom Startpunkt des Roboters entfernt, muß dieser mit der Verdopplungsstrategie schlimmstenfalls 900 Meter zurücklegen.
Diese Strategie ist optimal, es gibt keine bessere, wie Baeza Yates u.a. in [BYCR93] bewiesen haben.

Der Grund, warum diese Verdopplungsstrategie kompetitiv ist, liegt an der Eigenschaft der geometrischen Reihe

$$2^0 + 2^1 + 2^2 + ... + 2^n < 2^{n+1}$$

3.3 Suche in m Halbgeraden

Erweitert man den Suchraum von 2 Halbgeraden (vom Startpunkt nach links bzw. rechts) auf *m* Halbgeraden, kann eine ähnliche Strategie angewendet werden:
Es sind *m* Halbgeraden gegeben, die sich in einem gemeinsamen Startpunkt s treffen.
In einer dieser „Wände“ ist wieder eine Tür, die der Roboter suchen soll (vgl. Abb.5).

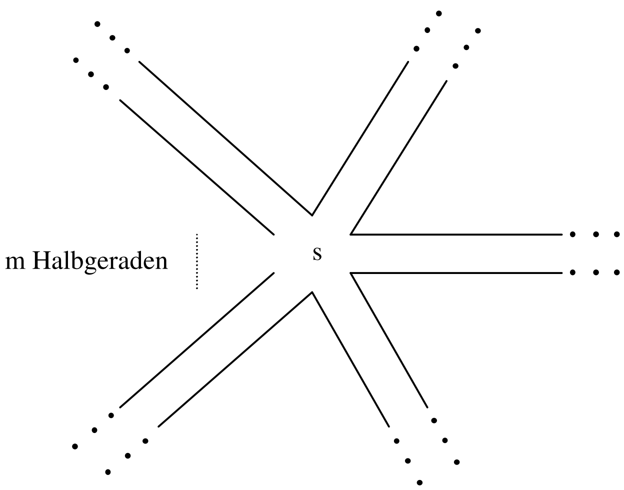

Abbildung 5: Erweiterung des Suchraums auf *m* Halbgeraden, sternförmig vom Startpunkt weg

<u>Strategie:</u>
Der Roboter legt eine bestimmte Reihenfolge fest und durchsucht die einzelnen Halbgeraden zyklisch. Die Suchtiefe wird dabei nicht verdoppelt, wie bisher, sondern wird folgendermaßen für den j-ten Suchvorgang berechnet:

$$f_j = \left(\frac{m}{m-1}\right)^j \qquad j = 1,2,3,...$$

Diese Suchstrategie ist kompetitiv mit dem Faktor

$$2\frac{m^m}{(m-1)^{m-1}} + 1 \leq 2em + 1$$

(mit $e = 2,718$; Eulersche Zahl)

Beweis:
Der schlimmste Fall tritt ein, wenn der j-te Suchvorgang die Tür knapp verpaßt, weil die Entfernung zum Startpunkt $f_j + \varepsilon$ beträgt. Dann wird eine ganze Runde umsonst gesucht mit den Tiefen $f_{j+1},..., f_{j+m-1}$, bevor der Roboter wieder in der richtigen Halbgerade sucht.

Der zurückgelegte Weg beträgt daher

$$2\sum_{i=1}^{j+m-1} f_i + f_j + \varepsilon \quad = \quad 2\sum_{i=1}^{j+m-1}\left(\frac{m}{m-1}\right)^i + f_j + \varepsilon$$

$$< \quad 2\frac{\left(\frac{m}{m-1}\right)^{j+m} - 1}{\frac{m}{m-1} - 1} + f_j + \varepsilon$$

$$\leq \quad 2(m-1)\left(\frac{m}{m-1}\right)^{j+m} + f_j + \varepsilon$$

Der kompetitive Faktor ist daher nicht größer als

$$\frac{2(m-1)\left(\frac{m}{m-1}\right)^{j+m} + f_j + \varepsilon}{f_j + \varepsilon} \quad \leq \quad 2(m-1)\left(\frac{m}{m-1}\right)^m + 1$$

$$= \quad 2m\left(1 + \frac{1}{m-1}\right)^{m-1} + 1$$

$$\leq \quad 2me + 1$$

Die letzte Abschätzung gilt, da $\left(1 + \frac{1}{m}\right)^m$ monoton gegen die Zahl e wächst.

Der kompetitive Faktor hängt also von der Anzahl m von Halbgeraden ab.
Dieser Faktor $2\frac{m^m}{(m-1)^{m-1}} + 1$ ist optimal. Für $m = 2$ ergibt sich der bekannte Faktor 9 aus Beispiel 3.2.3

3.4 Suche in Polygonen

Nun wird der Suchraum weiter verallgemeinert:
Der Roboter soll nicht mehr nur Halbgeraden absuchen, sondern selbständig Polygone auf effiziente Weise durchwandern.

Zunächst betrachten wir ein recht einfaches Polygon:

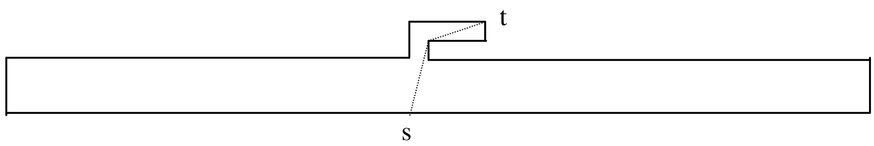

Abbildung 6: Einfaches Polygon

Der Roboter soll einen möglichst kurzen Weg vom Startpunkt *s* zum Zielpunkt *t* suchen. Würde der Roboter, wie bisher, entlang der Wand (hier der Polygonrand) in einer bestimmten Richtung suchen, würde er irgendwann den Punkt t erreichen.
Im Vergleich zum optimalen Weg, gestrichelt eingezeichnet, ist diese Strategie allerdings bedeutend schlechter und nicht kompetitiv, egal ob er nur in einer Richtung sucht oder abwechselnd. Der Roboter muß sich also von der Wand lösen und in den „Raum", also das Innere des Polygons gehen.

3.4.1 Einführung eines Sichtsystems

Dazu muß er mit einem Sichtsystem ausgestattet werden, welches ihm an jeder Stelle *p* im Polygon *P* das *Sichtbarkeitspolygon vis(p)* zur Verfügung stellt. Dieses Sichtbarkeitspolygon ist genau der Teil des Gesamtpolygons, den der Roboter gerade sieht. Das Sichtsystem zeichnet nun während der Roboter den Raum absucht eine partielle Karte, auf der alle Teile von *P* eingezeichnet sind, die schon einmal sichtbar waren.
Diese partielle Karte stellt immer ein Teilpolygon von *P*, sie enthält nach einem zurückgelegten Weg π die Punkte

$$\bigcup_{p \in \pi} vis(p)$$

Beispiel für eine partielle Karte:

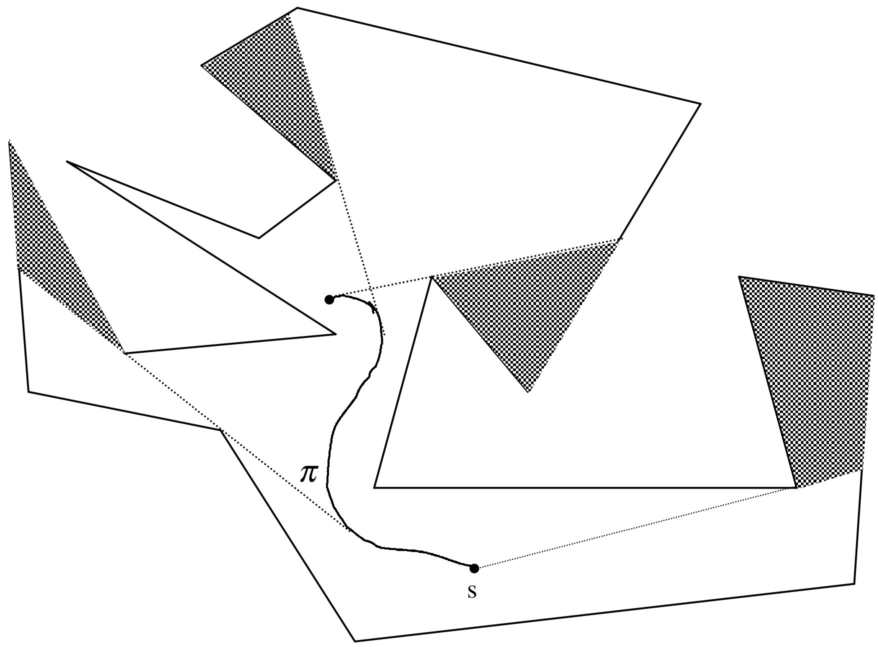

Abbildung 7: Die partielle Karte des Roboters, nachdem er den Weg π zurückgelegt hat

Diese Karte wird gezeichnet, wenn der Roboter sich von *s* auf dem Weg π bewegt.
Die grauen Flächen sind die Bereiche des Gesamtpolygons, die der Roboter nicht sehen konnte. Sie gehören deshalb nicht zur partiellen Karte.

3.4.2 Kostenabschätzung

Für die Kosten von möglichen Suchstrategien, nach denen der Roboter ein solches Polygon absuchen kann, gilt folgender Satz:

> *Keine Strategie für die Suche nach einem Zielpunkt kann in beliebigen Polygonen mit n Ecken stets einen kleineren Faktor als* $\frac{n}{2}-1$ *für das Weglängenverhältnis garantieren.*

Dies läßt sich leicht durch einen Widerspruchsbeweis zeigen:
Wir nehmen an, es gibt eine Strategie, die für jedes Polygon einen besseren Wegfaktor liefert. Wenn man ein Polygon findet, von dem gezeigt werden kann daß der Faktor mindestens $\frac{n}{2}-1$ beträgt für jede Strategie, ist das ein Widerspruch zu unserer Annahme.

Wir betrachten folgendes Polygon:

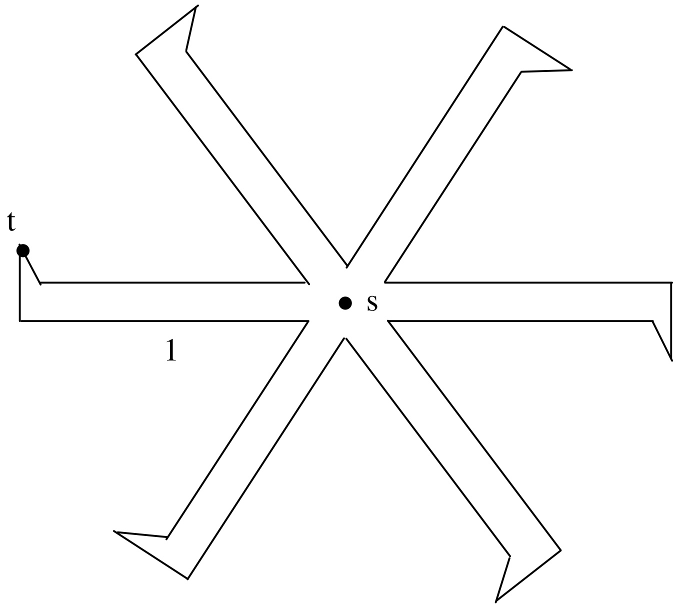

Abbildung 8: Im ungünstigsten Fall liegt der Zielpunkt *t* am Ende des zuletzt besuchten Korridors

Es hat *n* Ecken und besitzt $\frac{n}{4}$ viele Korridore, die beim Startpunkt *s* zusammenlaufen und an ihren Ecken abgeknickt sind. Um den Zielpunkt *t* zu finden, muß der Roboter die Korridore nacheinander absuchen und hinter die Ecken schauen. Im ungünstigsten Fall liegt der Zielpunkt am Ende des zuletzt besuchten Korridors. Die Korridore haben die Länge 1 und die Länge der abgeknickten Ecken sei vernachlässigbar klein.
Dann ist die Länge des Gesamtwegs

$$\left(\frac{n}{4}-1\right)\cdot 2+1=\frac{n}{2}-1$$

gegenüber der Länge 1 des kürzesten Wegs von *s* nach *t*.

q.e.d.

Falls es Strategie gibt, bei der das Verhältnis der Länge des zurückgelegten Weges zur Länge des kürzesten Weges stets beschränkt ist, kann diese Schranke auf jeden Fall keine Konstante sein. Schlimmstenfalls hängt der zurückgelegte Weg linear von der Anzahl *n* von Polygonecken ab.
Bei solchen Polygonen, die nur aus mehreren Korridoren bestehen, die am Startpunkt zusammenlaufen, kann die Strategie der exponentiellen Vergrößerung der Suchtiefe (s.3.3) verwendet werden. Dabei werden die Korridore genau wie Halbgeraden behandelt, auch wenn sie mehrere Knickstellen enthalten. Diese Strategie garantiert ein Weglängenverhältnis in $O(n)$.

3.4.3 Allgemeine Polygone: Verwendung des „shortest path tree“

Für allgemeine Polygone, die nicht nur aus Korridoren bestehen, benötigt man ein anderes Verfahren. Dazu nimmt man folgende Struktur zur Hilfe:
Man betrachtet die Gesamtheit aller kürzesten Wege im Polygon *P*, die vom Startpunkt *s* zu allen Eckpunkten von *P* führen.
Diese Wege bilden einen Baum, der *Baum der kürzesten Wege* von *s* in *P* genannt (engl. *shortest path tree*) und mit ***SPT*** bezeichnet wird. Man kann sich die Kanten des Baumes als gespannte Gummibänder vorstellen, die an spitzen Ecken von *P* abknicken können, vgl. Abb.9.

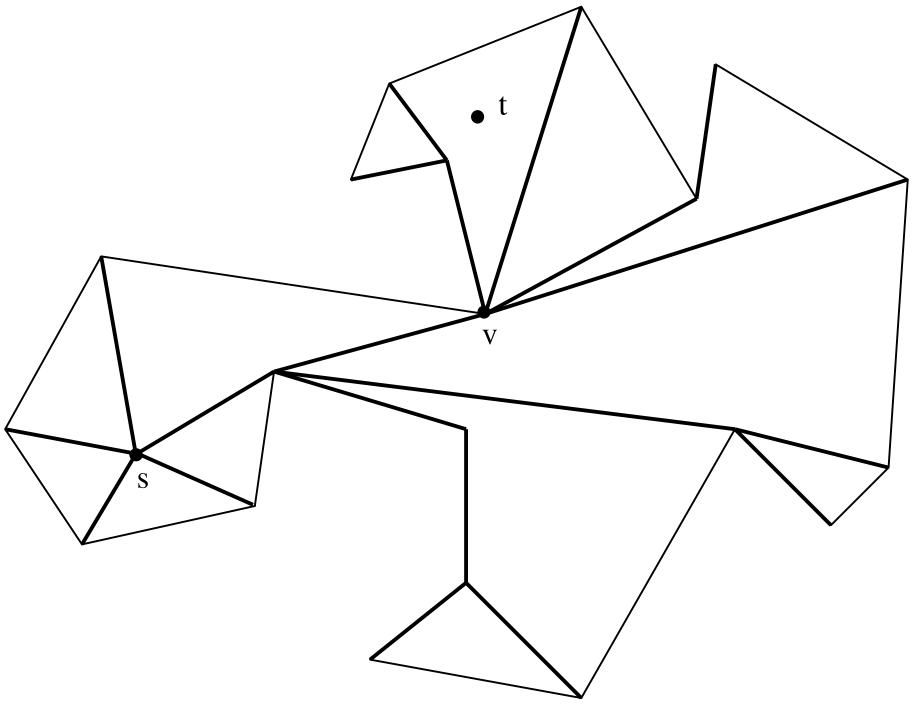

Abbildung 9: Der Baum *SPT* der kürzesten Wege von *s* zu allen Eckpunkten

Bei einem Polygon mit *n* Ecken und einem Startpunkt *s* im Innern des Polygons hat der Baum *SPT* genau $n+1$ Knoten, darunter $m \leq n$ viele Blätter.

Wir gehen davon aus, daß die betrachteten Polygone keine „Löcher“ besitzen, also keine Gebiete im Inneren, die nicht zum Polygon selbst gehören. Sie werden als einfach zusammenhängend bezeichnet. Bei Polygonen mit Löchern können verschiedene gleich lange Wege existieren. Wenn der Roboter auf dem Weg zu einem Zielpunkt beispielsweise links an einem Loch vorbei geht, könnte der rechte Weg u.U. kürzer oder genauso lang sein, vgl. Abb.10.

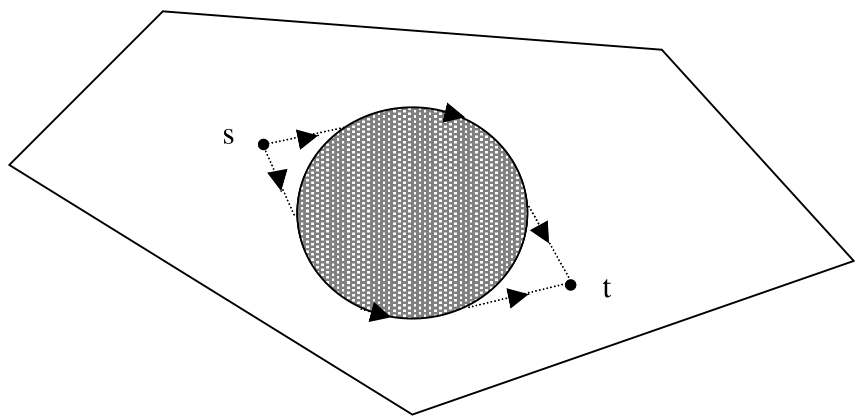

Abbildung 10: Polygon mit „Loch“, es gibt zwei gleich lange Wege von *s* nach *t*

Strategie für einfach zusammenhängende Polygone:
Der Roboter könnte das Polygon mit Hilfe des *SPT* folgendermaßen absuchen:
Der *SPT* hat *m* Blätter, somit gibt es *m* Wege vom Startpunkt zu diesen Blättern.
Diese Wege werden einfach als disjunkte Halbgeraden behandelt und mit exponentiell wachsender Suchtiefe abgesucht.
(Vergrößerung der Suchtiefe mit Faktor $\frac{m}{m-1}$, vgl. 3.3)

Der Roboter sucht dabei nach einem Punkt *v*, von dem aus der Zielpunkt *t* sichtbar ist (s. letzte Grafik). Dieser Punkt *v* sei der letzte Knickpunkt auf dem kürzesten Weg von *s* nach *t* und somit ein Knoten von *SPT*.
Es kann noch andere Knoten des *SPT* geben, von denen man den Zielpunkt *t* sehen. Falls der Roboter einen dieser Knoten vor dem Knoten *v* erreicht, würde er in der Praxis natürlich direkt von dort zu seinem Zielpunkt gehen. Zur Vereinfachung gehen wir aber davon aus, daß er solange sucht, bis er *v* findet und erst von dort *t* ansteuert.
Sei $d(s,p)$ die Länge des kürzesten Weges von *s* nach einem Punkt *p*, und $w(s,p)$ die Länge des tatsächlich zurückgelegten. Da der Knoten *v* auf dem kürzesten Weg von *s* nach *t* liegt, ist $d(s,t) = d(s,v) + |vt|$.
Wegen $d(s,v) \leq w(s,v)$ und der Abschätzung des kompetitiven Faktors bei der Suche in *m* Halbgeraden aus 3.3 ergibt sich folgendes Weglängenverhältnis:

$$\frac{w(s,t)}{d(s,t)} = \frac{w(s,v)+|vt|}{d(s,v)+|vt|} \leq \frac{w(s,v)}{d(s,v)} \leq 2em+1$$

Die Strategie erreicht also einen kompetitiven Faktor, der linear von der Anzahl *m* von Blättern im *SPT* abhängt.

In der Praxis ergibt sich aber die Schwierigkeit, daß der Roboter sich in einer unbekannten Umgebung befindet und somit weder den *SPT* noch die Anzahl m seiner Blätter kennt.
Um dies zu umgehen, erhöht der Roboter seine Suchtiefe nicht um den unbekannten Faktor $\frac{m}{m-1}$, sondern verdoppelt sie vor jedem Suchvorgang.
Das Weglängenverhältnis bleibt dabei kleiner als $8m$, was man leicht mit dem Beweis aus 3.3 zeigen kann, wobei man den Faktor $f_j = 2^j$ verwendet.
Das andere Problem, daß der Roboter nicht den gesamten *SPT* kennt, ist nicht sehr schwerwiegend. Er kennt nämlich zu jedem Zeitpunkt den Verlauf des SPT soweit er sehen kann, dies genügt für die Planung des nächsten Schritts.

Es gilt sogar folgende Aussage:

> *Für jeden Punkt $p \in P$, den der Roboter bereits gesehen hat, kennt er den kürzesten Weg von s nach p.*

Beweis:
Da der Roboter p schon gesehen hat, gehört p – genau wie auch s – zur partiellen Karte P'.
Er kennt somit den kürzesten Weg π' von s nach p in P'.
Gäbe es einen Weg π, der kürzer ist und nicht (komplett) in P' liegt, müßte dieser das Teilpolygon P' verlassen und auch wieder betreten. Diese Schlinge ist aber auf jeden Fall länger als der direkte Weg in P'. Somit wäre der Weg π länger als π', dies ist ein Widerspruch zur Annahme, der Weg π' ist auch der kürzeste Weg von s nach p in P.

q.e.d.

Beim Absuchen des Polygons speichert der Roboter die kürzesten Wege zu allen Eckpunkten von P , die er schon gesehen hat, in einer zyklischen Liste L. Diese Wege geht er nun rundenweise in einer bestimmten Reihenfolge nacheinander ab.

Graphisches Beispiel:

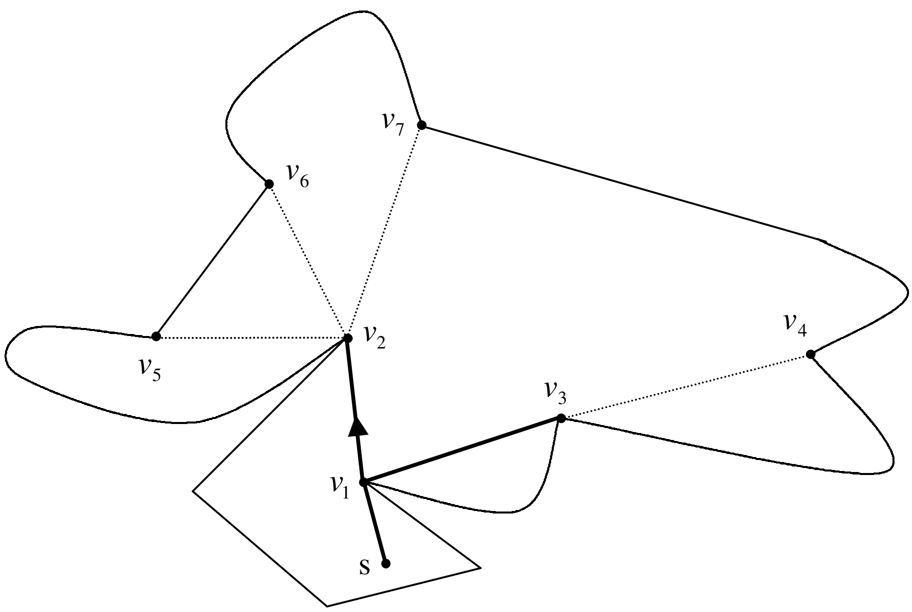

Abbildung 11: Wege des Roboters zu den Eckpunkten des Polygons

Der Roboter ist gerade auf dem Weg von *s* über v_1 nach v_2, siehe Abb.11.
Wenn er v_2 erreicht hat, sieht er die Eckpunkte v_5, v_6, v_7. Er weiß, daß der kürzeste Weg von *s* zu ihnen über den Punkt v_2 führt, deshalb ersetzt er in seiner Liste *L* den Weg (s, v_1, v_2) durch die drei Wege (s, v_1, v_2, v_5), (s, v_1, v_2, v_6) und (s, v_1, v_2, v_7).
Hat er in v_2 die Suchtiefe für die aktuelle Runde noch nicht erreicht, geht er weiter und erkundet den Weg (v_2, v_5). Anschließend untersucht er den Weg (s, v_1, v_2, v_6), und so weiter nacheinander jeden Weg aus seiner Liste.
Während der Roboter noch auf dem Weg von v_1 nach v_2 ist, sieht er den Eckpunkt v_4.
Da er weiß, daß der kürzeste Weg von *s* zu v_4 über v_3 führt, könnte er bereits jetzt in der Liste *L* den Weg (s, v_1, v_3) durch (s, v_1, v_3, v_4) ersetzen. Er kann dies aber auch erst tun, wenn beim Suchen der Weg (s, v_1, v_3) an der Reihe ist und er den Punkt v_4 „wiederentdeckt“.

Die beschriebene Strategie ist kompetitiv, sie garantiert in jedem einfachen Polygon mit *n* Ecken, wie schon erwähnt, den Faktor 8*n*.

Diese Abschätzung gilt für den *worst case*, also den schlechtesten Fall. In der Praxis läßt sich dieser Faktor häufig noch etwas verkleinern, indem man die Strategie etwas abändert und dadurch Weglänge einsparen kann. So muß der Roboter beim Absuchen des *SPT* nicht jedesmal bis zum Startpunkt *s* zurückgehen, bevor er den nächsten Weg absucht.
Es genügt, bis zu dem Eckpunkt zurückzugehen, an dem sich die beiden Wege teilen.
In manchen Situationen ist nicht einmal das erforderlich:
Im letzten Beispiel, vgl. Abb. 11, kann der Roboter, wenn er den Punkt v_7 erreicht, zur Erkundung des nächsten Weges (s, v_1, v_3, v_4) gleich direkt zu v_4 gehen.

Daneben kann der Roboter die Kanten, die zu Blättern des Baumes führen, also am Rand des Polygons liegen, vom Absuchen ausnehmen. Dies ist erlaubt, da jeder Punkt im Polygon bereits von irgendeinem inneren Knoten des *SPT* sichtbar ist.

Abschließend ist zu sagen, daß dieser Ansatz nicht nur für einfache Polygone funktioniert.
Laut [Kle97] läßt er sich auf beliebige Umgebungen verallgemeinern, was aber auf den ersten Blick nicht ersichtlich ist und auch nicht näher erläutert wird.
Ebenso ist das Verfahren der Suchtiefenverdopplung nicht nur auf eindimensionale Suchräume anwendbar.

Literaturverzeichnis

[Kle97] R.Klein: Algorithmische Geometrie, Addison-Wesley, 1997

[BYCR93] R. Baeza-Yates, J. Culberson, G. Rawlins: "Searching in the plane". Information and Computation, 106: 234-252, 1993