



Suchalgorithmen zur Bahnplanung

Navigation Autonomer Mobiler Systeme

Prof. Dr. Bernd Krieg-Brückner
Dr. Bernd Gersdorf

Vortrag von:

Fatma Akin
Aneliya Maneva

14.12.2007



Gliederung

I. Inkrementelle Abtastung und Suche

- 1.1 Der allgemeine Rahmen
- 1.2 Anpassen von diskreten Suchalgorithmen
- 1.3 Zufällig generierte Potenzielle Felder

II. Schnell erkundende dichte Bäume

- 2.1 Erkundungsalgorithmus
- 2.2 Effizientes Finden von nahe stehenden Punkten/Knoten
- 2.3 Planung mit Hilfe von Bäumen



I. Inkrementelle Abtastung und Suche

1.1 Der allgemeine Rahmen

Die Abtastbasierte Algorithmen folgen dieser Vorlage:

1. **Initialization:** Let $\mathcal{G}(V, E)$ represent an undirected *search graph*, for which V contains at least one vertex and E contains no edges. Typically, V contains q_I , q_G , or both. In general, other points in \mathcal{C}_{free} may be included.
2. **Vertex Selection Method (VSM):** Choose a vertex $q_{cur} \in V$ for expansion.
3. **Local Planning Method (LPM):** For some $q_{new} \in \mathcal{C}_{free}$ that may or may not be represented by a vertex in V , attempt to construct a path $\tau_s : [0, 1] \rightarrow \mathcal{C}_{free}$ such that $\tau(0) = q_{cur}$ and $\tau(1) = q_{new}$. Using the methods of Section 5.3.4, τ_s must be checked to ensure that it does not cause a collision. If this step fails to produce a collision-free path segment, then go to step 2.
4. **Insert an Edge in the Graph:** Insert τ_s into E , as an edge from q_{cur} to q_{new} . If q_{new} is not already in V , then it is inserted.



1.1 Der allgemeine Rahmen

5. **Check for a Solution:** Determine whether \mathcal{G} encodes a solution path. As in the discrete case, if there is a single search tree, then this is trivial; otherwise, it can become complicated and expensive.
6. **Return to Step 2:** Iterate unless a solution has been found or some termination condition is satisfied, in which case the algorithm reports failure.



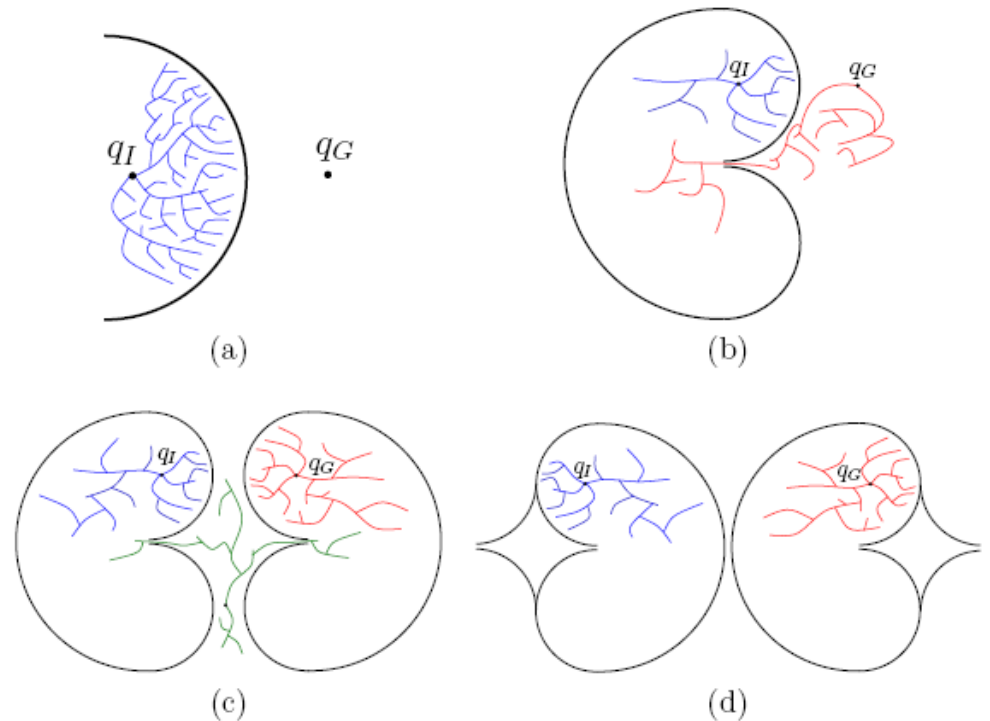
1.1 Der allgemeine Rahmen

LPM

- Kollisionsfreies Pfadsegment erzeugen und in den Graphen einfügen
- Lokal – Pfadsegment einfach, Entfernungen kurz
- Nicht global – ganzes Bahnproblem wird nicht von LPM gelöst

1.1 Der allgemeine Rahmen

- Meiste Suchalgorithmen fallen in einer Falle
- Abtastbasierte Algorithmen werden mit Suchalgorithmen kombiniert





1.1 Der allgemeine Rahmen

Algorithmen, die auf Suchbäume basieren

- Unidirectional method
 - Ähnlich wie Diskrete Vorwärtssuche
 - aber VSM und LPM – anders implementiert
 - Bug trap – Probleme bei Vorwärtssuche,
nicht bei Rückwärtssuche



1.1 Der allgemeine Rahmen

- Bidirectional method
 - bidirektionale Annäherung öfters besser
 - da sich 2 Flächen (der eine von $q(I)$ aus, der andere von $q(G)$ aus verbreiten)
 - werden zusammentreffen ohne viel Fläche zu verdecken



1.1 Der allgemeine Rahmen

- Multi-directional method
 - falls 2 bug traps existieren
 - noch 2 weitere Bäume von anderen Orten aus wachsen lassen
 - jedoch: erschwert das Verbinden von Bäumen



1.2 Anpassen von diskreten Suchalgorithmen

- Eine Art abtastbasierte Algorithmen durchführen
 - Ein Gitter über C definieren
 - Diskrete Suche durchführen
- diskrete Bahnplanungsproblem
 - Gitter und Nachbarkunkte Definieren
 - Für jeden Gitterpunkt q Nachbarkunkte definieren
 - q darf kein Grenzpunkt sein



1.2 Anpassen von diskreten Suchalgorithmen

Lösung von Gitterproblemen

- Die bisherige Methode wird Lösungen finden, wenn eine korrekte Auflösung gegeben ist
- ist die Anzahl der Punkte pro Achse hoch, dann ist die Suche sehr langsam
- Weniger Punkte = möglicherweise keine Lösung
- Verschachtelung von sampling und searching



1.2 Anpassen von diskreten Suchalgorithmen

- Iterative Verbesserung der Gitterauflösung
 - Gitter mit 2^n Punkten durchsucht mit Suchalgorithmen
 - Suche fehlgeschlagen= Auflösung wird verdoppelt
 - Suche wird wieder angewendet
 - Wiederholung bis eine Bahn gefunden ist
 - Nachteil = Suche kann lange dauern bis eine Auflösung gefunden wird



1.2 Anpassen von diskreten Suchalgorithmen

- Bevorzugbar:
 - Man erzeugt Gittern die i^n Punkte pro Achse haben
 - Es ergeben sich Größen von 2^n , 3^n usw.



1.2 Anpassen von diskreten Suchalgorithmen

- Viel besser:
 - Wenn man sampling und searching fester verschachtelt
 - Samples als dichte Sequenz α
 - Der Graph wird nach jeden 100 Punkten gesucht
 - Besser = wenn Sequenz α eine Kristallgitterstruktur hat
 - Wäre besser, wenn man lokale Minima vermeidet



1.3 Zufällig generierte potenzielle Felder

- Kleine Anzahl von Punkten
- oder niedrige Dimension
- => jeder Knoten in vernünftiger Zeit erreichbar
- falls die Dimension = 10,
Anzahl der Punkte = 50
- Unmöglich alle zu durchsuchen

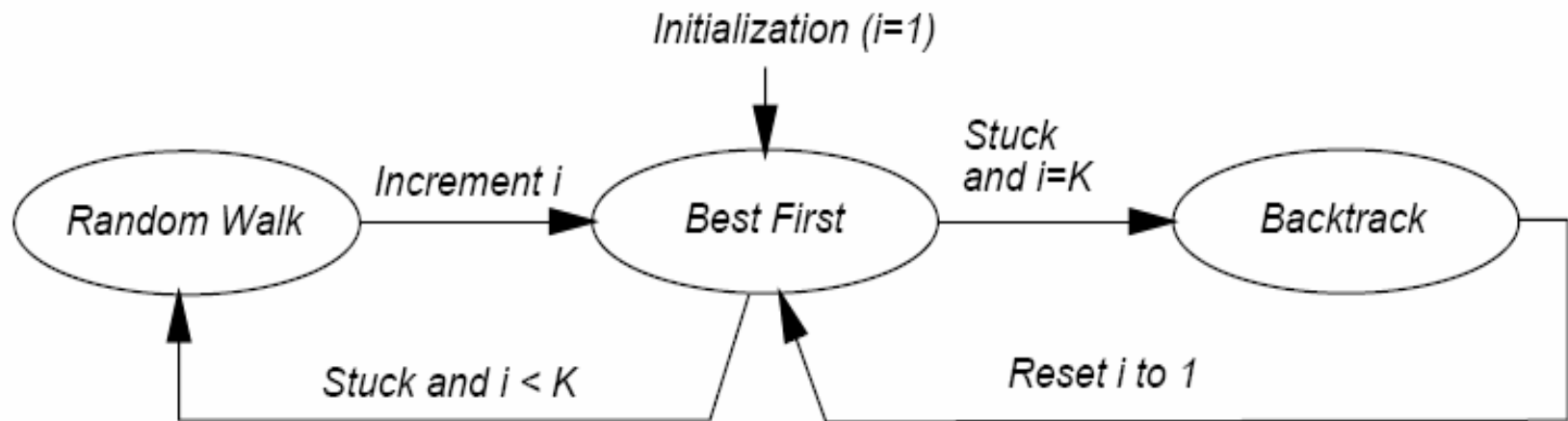


1.3 Zufällig generierte Potenzielle Felder

- Methoden mit Best First Mode würden die Lösung finden ohne alle samples zu suchen
- Aber nicht für das Problem zur Untersuchung von Knoten in einer Höhlung
- Random Walks
- Zur Vermeidung von lokaler Minima

1.3 Zufällig generierte Potenzielle Felder

Random Walk





II. Schnell erkundende dichte Bäume

- Rapidly Exploring Dense Trees (RDT)
- Baum – wegen Struktur des Weges
- Dichte – viele Blätter
- Erkundende – Algorithmus erkennt Hindernisse
- Schnell – Problem wird aufgeteilt



2.1 Erkundungsalgorithmus

Der Baum wird erweitert:

- Man setzt einen weiteren (neuen) Knoten (wo kein Hindernis ist)
- Nahestehenden Knoten finden
- Beide Knoten verbinden



2.1 Erkundungsalgorithmus

1. Graph G initialisieren
2. Schleife für neue Knoten
(erstellen und hinzufügen)
3. Knoten wird erstellt
4. Nahestehenden Knoten wird
gesucht
5. Kante zw. Knoten erzeugen

Textuelle Darstellung des Algorithmus

```
SIMPLE_RDT( $q_0$ )  
1   $\mathcal{G}$ .init( $q_0$ );  
2  for  $i = 1$  to  $k$  do  
3     $\mathcal{G}$ .add_vertex( $\alpha(i)$ );  
4     $q_n \leftarrow \text{NEAREST}(S(\mathcal{G}), \alpha(i))$ ;  
5     $\mathcal{G}$ .add_edge( $q_n, \alpha(i)$ );
```

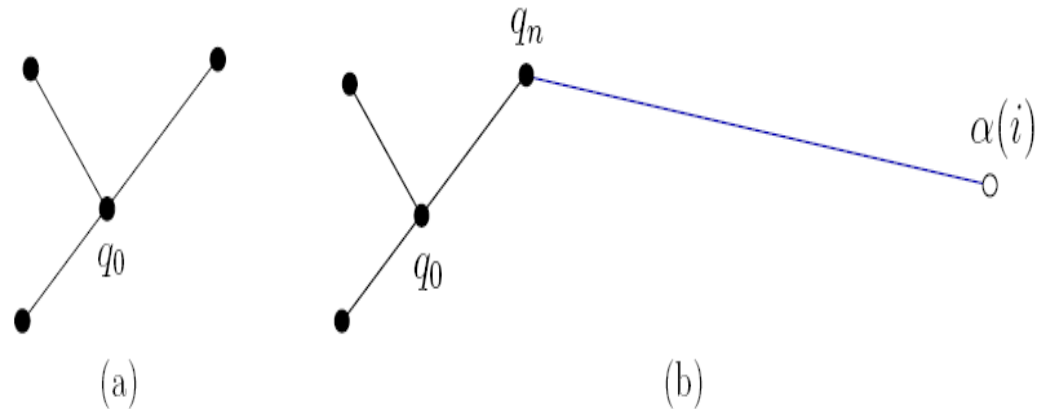
2.1 Erkundungsalgorithmus

1. Nahstehender Knoten
(vom Graphen) suchen

Hinzufügen von neuen Knoten (kein Hindernis)

2. Diesen mit dem
Zielknoten $\alpha(i)$
verbinden

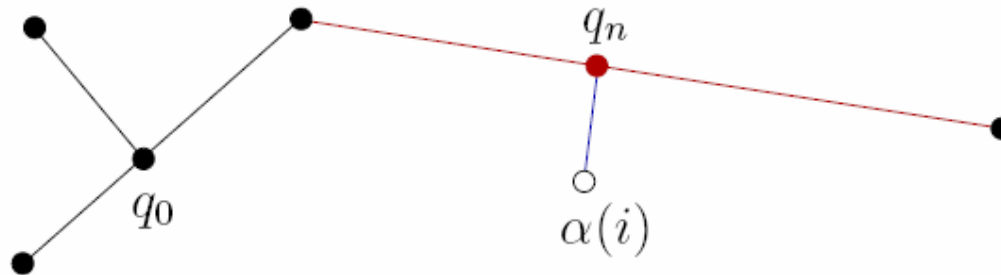
3. Neue Kante wird
erzeugt



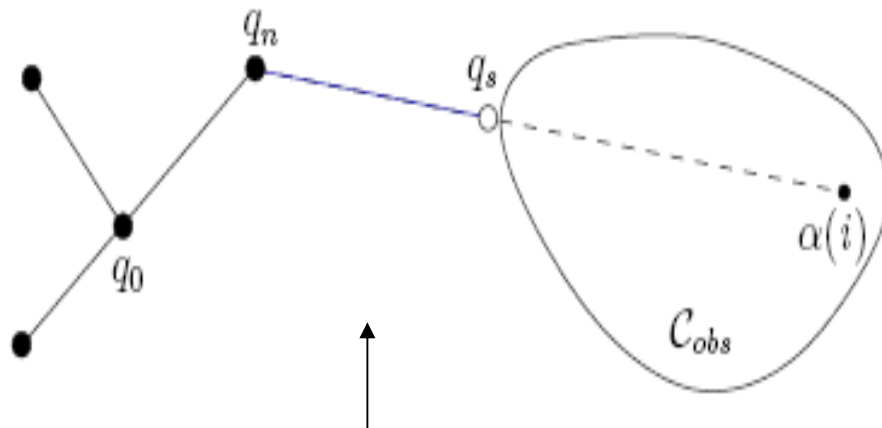
2.1 Erkundungsalgorithmus

Kanten Splitten um längere Strecken zu vermeiden:

- Ziel – $\alpha(i)$
- Nahester Punkt auf der Kante finden
- Splitten und Knoten erzeugen
- Kante erzeugen



2.1 Erkundungsalgorithmus



Hinzufügen von neuen Knoten
(mit Hindernissen)

Textuelle Darstellung des
erweiterten Algorithmus

- Ziel $\alpha(i)$ liegt im Hindernis
- Letzten Punkt vor dem Hindernis finden
- Ihn als Knoten in G einfügen

RDT(q_0)

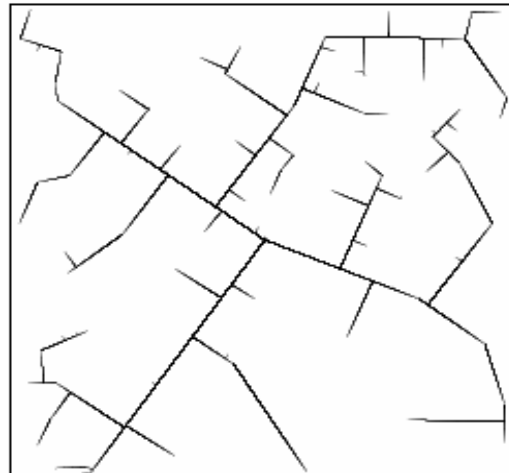
```
1   $\mathcal{G}.\text{init}(q_0)$ ;  
2  for  $i = 1$  to  $k$  do  
3     $q_n \leftarrow \text{NEAREST}(S, \alpha(i))$ ;  
4     $q_s \leftarrow \text{STOPPING-CONFIGURATION}(q_n, \alpha(i))$ ;  
5    if  $q_s \neq q_n$  then  
6       $\mathcal{G}.\text{add\_vertex}(q_s)$ ;  
7       $\mathcal{G}.\text{add\_edge}(q_n, q_s)$ ;
```

2.1 Erkundungsalgorithmus

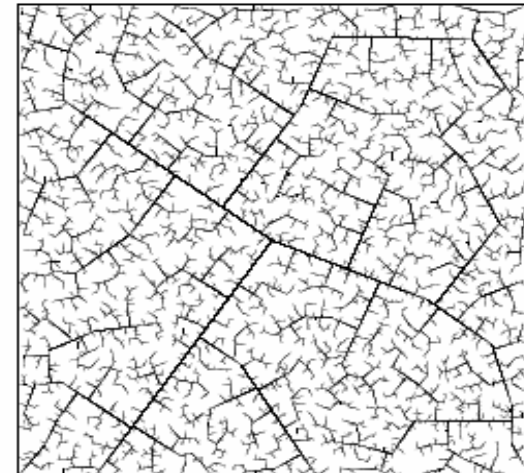
Zufällige weit entfernte Regionen:

- schnell erreicht
- Nach Anzahl der Iterationen -> Baum sehr dicht

**Rapidly Exploring
Random Trees (RRT)**



45 iterations



2345 iterations



2.2 Effizientes Finden von nahe stehenden Punkten/Knoten

Teilfunktion vom Erkundungsalgorithmus

- Exakte Methoden:
 - Knoten auf Entfernung zum neuen Knoten prüfen
 - derjenige mit geringster Entfernung – der Naheste



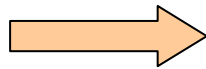
2.2 Effizientes Finden von nahe stehenden Punkten/Knoten

Teilfunktion vom Erkundungsalgorithmus

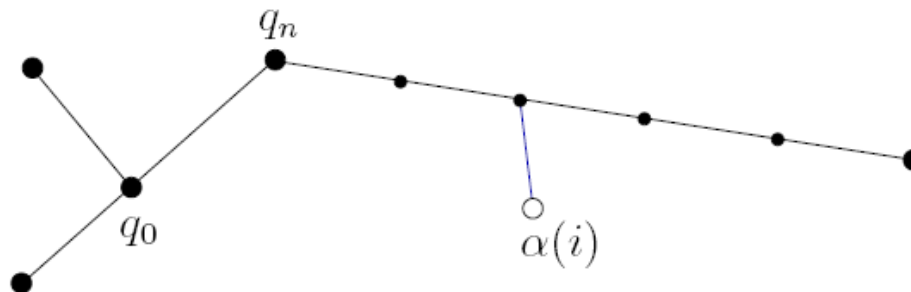
- Angenäherte Methoden:
 - Kante durch Punkte unterteilen
 - Punkte (\neq Knoten vom G) auf Entfernung zum neuen Knoten prüfen

2.2 Effizientes Finden von nahe stehenden Punkten/Knoten

- feineres Unterteilen von Kanten



- höhere Qualität des Ergebnisses
- längere Zeit fürs Algorithmus





2.3 Planung mit Hilfe von Bäumen

Single-tree search

Einzelbaumsuche – nur ein Baum wird erweitert

- Baum bewegt sich zufällig im Suchraum
- Größerer Suchraum => häufigere manuelle Festlegung des Zielknotens als Zwischenknoten



2.3 Planung mit Hilfe von Bäumen

Balanced, bidirectional search

Balancierte Suche in beide Richtungen

- Ein Baum vom Start zum Ziel
- Zweiter Baum vom Ziel zum Start

Man konzentriert sich auf den kleineren Baum
(zum Ausgleich)

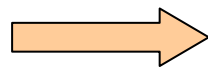


2.3 Planung mit Hilfe von Bäumen

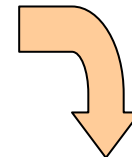
More than two trees

Mehr als zwei Bäume

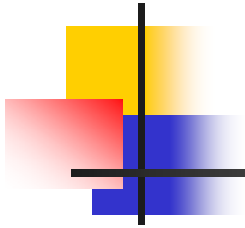
- Mehrere Bäume starten (in schlechte Regionen)
- Ein Knoten in 2 Bäume enthalten



Vereinigung der Bäume



Suche verkürzt



Vielen Dank für die Aufmerksamkeit !