Universität Dortmund
Fachbereich Informatik
Wintersemester 1999/2000

Dortmund, 25. November 1999

Musterlösungen zur Vorlesung Einführung in die künstliche Intelligenz

Blatt 3

Aufgabe 3.1:

1. Wenn der Zustandsraum aus allen Positionen in der Ebene besteht ($\mathcal{R}^2$), dann gibt es unendlich viele Zustände und damit auch unendlich viele Pfade zum Ziel. Die Anzahl der kürzesten Pfade ist aber immer endlich, obgleich es mehrere optimale Pfade zum Ziel geben kann, z.B. 4 optimale Pfade in Abbildung 1.

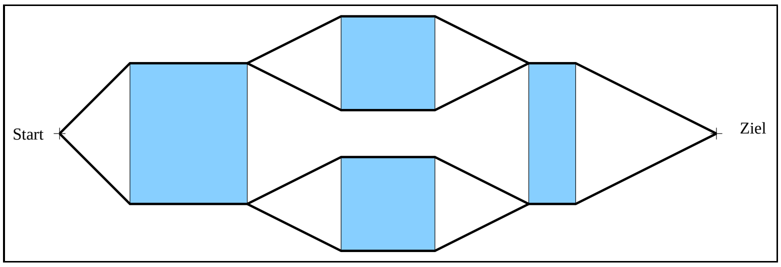

Abbildung 1: Eine Szene mit vier optimalen Pfaden

2. (a) Der kürzeste Pfad zwischen Start und Ziel ist eine Gerade. Wenn diese kein Polygon schneidet, ist dieses der kürzeste Pfad und wir sind fertig.

   (b) Schneidet diese Gerade genau ein Polygon, konstruieren wir zwei gültige Pfade (die Polygone sind konvex), indem wir die Schnittpunkte dergestalt verbinden, dass die Pfade über die Kanten laufen. Anschließend kann man unter Verwendung der Dreiecksungleichung diese Pfade immer dann verkürzen, wenn die Schnittpunkte keine Eckpunkte sind. Dazu werden die beiden zum Schnittpunkt benachbarten Ecken benutzt. Beide so verkürzten Pfade laufen nun über die Ecken der Polygone und der kürzeste Pfad ist per Konstruktion einer von ihnen, siehe Abbildung 2.

   (c) Nun werden alle Eckpunkte, die nicht zum selben Polygon gehören, mit allen anderen Ecken mit einer Gerade verbunden. Auf alle Geraden, die ein oder mehrere Polygone durchschneiden, wird wieder Punkt 2 angewendet.

   (d) Nun zeigt man, dass für zwei beliebige Punkte zwischen denen genau ein Polygon liegt, immer gilt, dass die kürzeste Strecke über die Ecken des Polygons führt und nicht über „freie“ Punkte im Raum. Siehe Abbildung 3.

   (e) Punkt 4 schliet Start und Ziel mit ein. Da nun zwischen zwei beliebigen Punkten immer eine direkte Gerade existiert, oder ein Weg, der über Ecken führt, gilt dieses auch für die optimale Lösung.

   Mit dieser Konstruktion ist gezeigt, dass der kürzeste Pfad über Ecken der Polygone führt, sofern keine Gerade direkt von Start zum Ziel verläuft. Das Ergebnis dieser Prozedur ist eine Menge von Pfaden, unter denen „der optimale Pfad“ enthalten ist.

   Ein guter Zustandsraum enthält folglich nur die Eckpunkte aller Polygone sowie Start– und Zielpunkt, sofern diese nicht mit einer Ecke zusammenfallen.

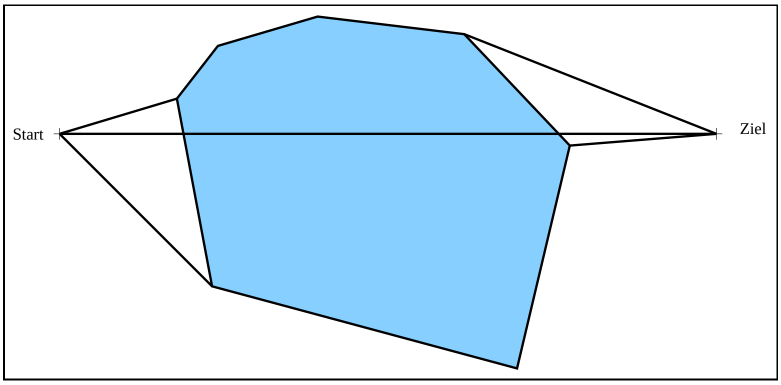

Abbildung 2:

3. Eine optimierte Version von A^* sieht folgendermaßen aus:

```
%%  Bibliotheken fuer Listenfunktionen und Negation:

:- ensure_loaded(library(basics)).
:- ensure_loaded(library(not)).
:- ensure_loaded(library(math)).

%%  Praedikat zum Starten des Suchprozesses, das den Start-Spielzustand
%%  auf den zugehoerigen Suchraumknoten abbildet, die Suche startet und
%%  die Anzahl der zum Erreichen des Ziels benoetigten Expansionen sowie
%%  den Suchraumknoten mit der Loesung liefert, der unter anderen den
%%  Pfad der Spielzustaende zum Ziel sowie die Kosten dieses Pfades umfasst:
%%      start( Startzustand+, Expansionen-, Loesung- )

start(Startzustand,Expansionen,Loesung) :-
    suche([pfad(0,0,0,Startzustand,[])],[],0,Expansionen,Loesung).

%%  A*-Suche (und Bergsteigen durch Aendern einer Zeile in 'sortiere'):
%%      suche( Offen+, Geschlossen+, AnzahlBisherigerExpansionen+,
%%              AnzahlExpansionenDerLoesung-, Loesung- )

suche([Aktuell|_],_Geschl,Expansionen,Expansionen,Aktuell) :- ziel(Aktuell).
    %%  Ende der Suche:  Ziel erreicht (aktueller gleich Ziel-Suchraumknoten)

suche([],_Geschl,Expansionen,Expansionen,[]).
    %%  Ende der Suche:  keine Loesung gefunden ('Offen' ist leer und Ziel
    %%                   nicht erreicht).

    %%  Expandierung des aktuell besten, noch offenen (Suchraum)Zustands
    %%  (= erstes Element in der 'Offen'-Liste):

suche([Aktuell|Offen],Geschl,Expansionen,ExpansionenErg,Loesung) :-
     write(Aktuell),nl,
```

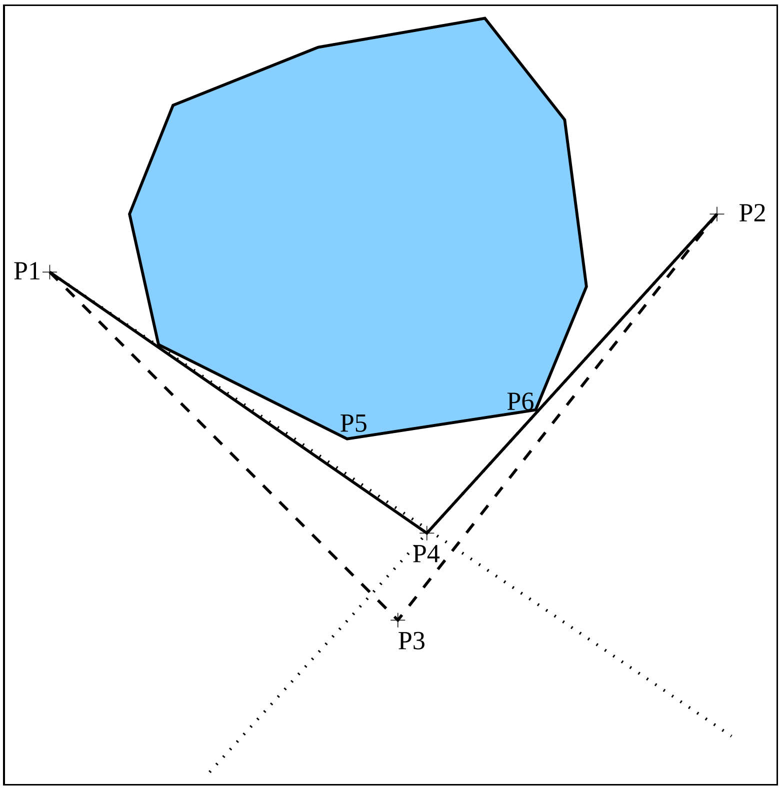

Abbildung 3:

```
    nachfolger(Aktuell,Geschl,Nachf),  %% Ermitteln aller noch zu
                                       %%   betrachtenden Nachfolger
    sortiere(Offen,Nachf,OffenNeu),  %% Aktualisiere & sortiere die Vereinigung
                                     %%   der Menge offener Knoten und der
                                     %%   Menge der aktuellen Nachfolger
    ExpansionenNeu is Expansionen+1,
    suche(OffenNeu,[Aktuell|Geschl],ExpansionenNeu,ExpansionenErg,Loesung).

    %% Bestimme alle noch zu betrachtenden Nachfolger-Suchraum-Zustaende des
    %% Suchraum-Zustands mit dem Spielzustand K:
nachfolger(pfad(_,G,_,K,P),Geschl,Nachf) :-
    findall(pfad(F1,G1,H1,K1,[K|P]), %% Ermittle alle (Spielzustand-)Nachfolger
        (nachf(K,K1,Kosten),         %%   K1 von K mit realen Kosten des
                                     %%   Wegs zu ihnen
         bewerte(K1,H1),             %% Schaetze Restkosten von K1 aus
                                     %%   entsprechend aktueller Heuristik
         G1 is G+Kosten,             %% Berechne neue reale Kosten
         F1 is G1+H1,                %% Berechne geschaetzte Gesamtkosten
         \+ (member(pfad(_,G2,_,K1,_),Geschl),  %% betrachte noch nicht
             G1 >= G2)),                        %%   geschlossene Nachfolger
        Nachf).                                 %%   sowie auch geschlossene,
```

```
                                                     %%   wenn sie jetzt mit
                                                     %%   kuerzerem realen Weg
                                                     %%   erreichbar sind

%%  Sortierte (Nachfolger)Liste:  eine Zeile fuer den A*-Algorithmus
%%  (Aufgabe 1.1) und eine Zeile fuer das Bergsteigen (Aufgabe 1.2),
%%  d.h. die jeweils andere ist auszukommentieren;
%%  die dritte Klausel ist beiden gemein:
%%      sortiere( Offen+, Nachfolger+, OffenNeu- )
%%% A*-Algorithmus

% best(Offen,Best,Restoffen):-
%      sort(Offen,[Best | Restoffen]).

%%% Bergsteigen
%%% (nur aktuell bestes Element behalten, den Rest verwerfen)
best(Offen,Best,[]):-
     sort(Offen,[Best|_]).

%%% Entferne alten Suchraumknoten mit groesseren realen Kosten fuer den
%%% aktuellen Spielraumknoten.
%%% sortiere(+Offen,+Nachfolger,-OffenNeu)

sortiere(Offen,[],[Best | Restoffen]) :- best(Offen,Best,Restoffen).


sortiere(Offen,[pfad(F,G,H,K,P)|Nachf],OffenNeu) :-   %% BEIDE Verfahren
    (  (member(pfad(FAlt,GAlt,HAlt,K,PAlt),Offen),
        G < GAlt) -> %% IF
           (append(Vor,[pfad(FAlt,GAlt,HAlt,K,PAlt)|Nach],Offen), %% THEN
            append(Vor,Nach,Offen2)
           )
        ;
        Offen2 = Offen  %% ELSE
    ),
    sortiere([pfad(F,G,H,K,P)|Offen2],Nachf,OffenNeu).


%     %% Abbildung Problemraum-Zielzustand <-> Spiel-Zielzustand

ziel(pfad(_,_,_,K,_)) :- zielzustand(K).


nachf((X1,Y1),(X2,Y2),Kosten):-
        nachf((X1,Y1),(X2,Y2)),
        XX is (X1-X2),
        YY is (Y1-Y2),
        hypot(XX,YY,Kosten).

%% benachbarte Knoten eines Polygons
```

```
nachf(K1,K2):-
        poly(Knotenliste),
        member(K1,Knotenliste),
        nachbar_poly(K1,K2,Knotenliste).

%% alle anderen erreichbaren Punkte auf Polygonen
nachf(K1,K2):-
        poly(Knotenliste),
        \+ member(K1,Knotenliste), % Knoten eines anderen Polygons
        member(K2,Knotenliste),
        kein_schnitt(K1,K2).

%% Ziel erreichbar?
nachf(K1,K2):-
        zielzustand(K2),
        K1 \== K2,
        kein_schnitt(K1,K2).

nachbar_poly(K1,K2,[K1, K2 |_]).

nachbar_poly(K1,K2,[_, K |Rest]):-
        nachbar_poly(K1,K2,[K| Rest]).

kein_schnitt(K1,K2):-
        findall((K3,K4),
                (poly(Knotenliste),
                 nachbar_poly(K3,K4,Knotenliste)),
                L),
        kein_schnitt(K1,K2,L).

kein_schnitt(_,_,[]):-!.

kein_schnitt(K1,K2,[(K3,K4) | Tail]):-
        kein_schnitt(K1,K2,K3,K4),
        kein_schnitt(K1,K2,Tail).

%% beide Strecken haben einen Punkt gemeinsam
kein_schnitt(K1,_,K1,_):-!.

kein_schnitt(K1,_,_,K1):-!.

kein_schnitt(_,K2,K2,_):-!.

kein_schnitt(_,K2,_,K2):-!.

%% boundingbox Test
kein_schnitt((X1,_),(X2,_),(X3,_),(X4,_)):-
        max(X1,X2,Max),
        min(X3,X4,Min),
        Max < Min,
```

```
        !.

kein_schnitt((X1,_),(X2,_),(X3,_),(X4,_)):-
        max(X1,X2,Max),
        min(X3,X4,Min),
        Min > Max,
        !.

kein_schnitt((_,Y1),(_,Y2),(_,Y3),(_,Y4)):-
        max(Y1,Y2,Max),
        min(Y3,Y4,Min),
        Max < Min,
        !.

kein_schnitt((_,Y1),(_,Y2),(_,Y3),(_,Y4)):-
        max(Y1,Y2,Max),
        min(Y3,Y4,Min),
        Min > Max,
        !.

kein_schnitt((X1,Y1),(X2,Y2),(X3,Y3),(X4,Y4)):-
        DetF is float((X2-X1)*(Y4-Y3) - (X4-X3)*(Y2-Y1)),
        DetF == 0.0,
        !.

kein_schnitt((X1,Y1),(X2,Y2),(X3,Y3),(X4,Y4)):-
        Det is (X2-X1)*(Y4-Y3) - (X4-X3)*(Y2-Y1),
        Det =\= 0.0,
        Lambda is ((X3-X1)*(Y4-Y3) - (X4-X3)*(Y3-Y1)) / Det,
        Mu is ((X3-X1)*(Y2-Y1) - (X2-X1)*(Y3-Y1)) / Det,
        (Lambda > 1; Lambda < 0.0; Mu > 1; Mu < 0.0),
        !.

%% ---------------------------------------------------------------

poly([(5,7),(7,2),(11,14),(5,7)]).
poly([(13,7),(15,5),(17,7),(15,9),(13,7)]).
poly([(8,12),(9,13),(8,14),(7,13),(8,12)]).

%% ---------------------------------------------------------------

zielzustand((25,8)).

%% ---------------------------------------------------------------

bewerte(K,Kosten) :- bewerte1(K,Kosten).        %% Heuristik 1

%bewerte(K,Kosten) :- bewerte2(K,Kosten).       %% Heuristik 2

%%  Heuristik 1:  h = 0  (Breitensuche bei A*,
```

```
%%                                          Tiefensuche bei Bergsteigen)

bewerte1(_,0).

%%  Heuristik 2:  Luftlinienentfernung

bewerte2((X1,Y1),Kosten):-
        zielzustand((X2,Y2)),
        XX is (X1-X2),
        YY is (Y1-Y2),
        hypot(XX,YY,Kosten).
```

4. Eine Beispielszene wird in Abbildung 4 gezeigt. A^* findet die optimale Lösung, Bergsteigen mit der Luftlinienentfernung als Heuristik findet eine Lösung und Bergsteigen mit $h = 0$ endet in einer Sackgasse.

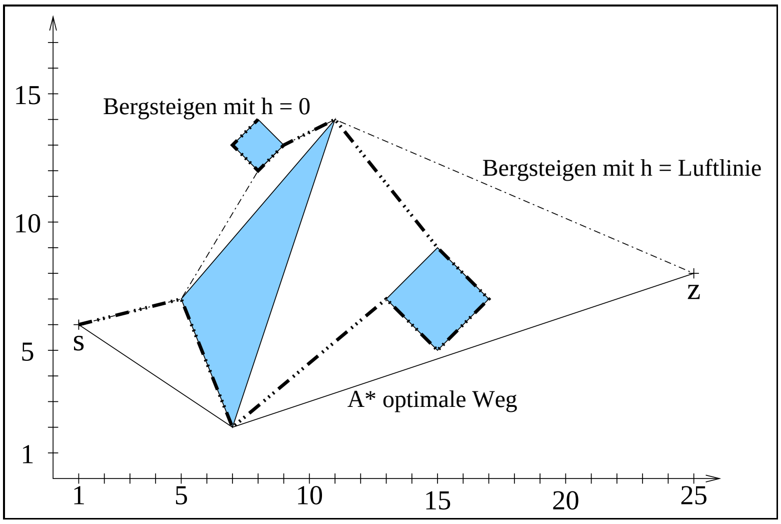

Abbildung 4:

Aufgabe 3.2:

1. Bei der Szene aus Abbildung 4 gibt es keine Lösung. Beachte aber, dass die Reihenfolge der Eckpunkte für das große Quadrat hierfür entscheidend ist! Die Reihenfolge hängt hier nur davon ab, in welcher Reihenfolge die Ecken im Prologprädikat `poly/1` definiert sind! Wenn das Polygon andersrum umlaufen würde, gäbe es eine Lösung!

2. Die Art der Hindernisse — konvex oder konkav — spielt keine Rolle, wenn es darum geht, ob Bergsteigen mit $h = 0$ erfolgreich oder wie hier erfolglos ist.

3. Siehe Abbildung 4.