Sztuczna inteligencja. Przeszukiwanie z informacją

Paweł Rychlikowski

Instytut Informatyki UWr

8 marca 2018

DLS. Przypomnienie

DLS = Depth Limited Search

Opis

- Określamy maksymalną głębokość poszukiwania.
- Przeszukujemy w głąb, ale nie rozwijamy węzłów na głębokości większej niż L.
- Wygodnie implementuje się rekurencyjnie (proste ćwiczenie)

Iterative Deepening

Uwaga

Iteracyjne pogłębianie to po prostu wywoływanie DLS na coraz to większej głębokości (bez zapamiętywania żadnych pośrednich wyników)

Może wydawać się to stratą czasu, ale:

- działamy w pamięci $O(bd)$,
- na czas wpływa ostatnia wartstwa, czyli $O(b^d)$

Przykładowe Iiterative Deepening

W drugiej kolumnie czas do tej pory, w trzeciej – czas *i*-tego pogłębienia.

10	0.02	0.04
11	0.06	0.1
12	0.16	0.27
13	0.44	0.75
14	1.19	2.23
15	3.42	5.91
16	9.33	16.63
17	25.96	47.02

Przykładowo dla ostatniego poziomu mamy:

- Czas działania to $25.96 + 47.02 = 72.98$
- Czyli narzut to 35%.

UCS. Właściwości

- UCS = Uniform Costs Search
- Zamiast kolejki FIFO mamy kolejkę priorytetową, z priorytetem równym kosztowi dotarcia do węzła.

Uwaga

Oczywiście umożliwia to różnicowanie kosztów dotarcia z węzła do węzła.

Uwaga 2

UCS rozwiązuje ten sam problem co algorytm Dijkstry (i w bardzo podobny sposób). Ale jest różnica powiedzmy filozoficzna

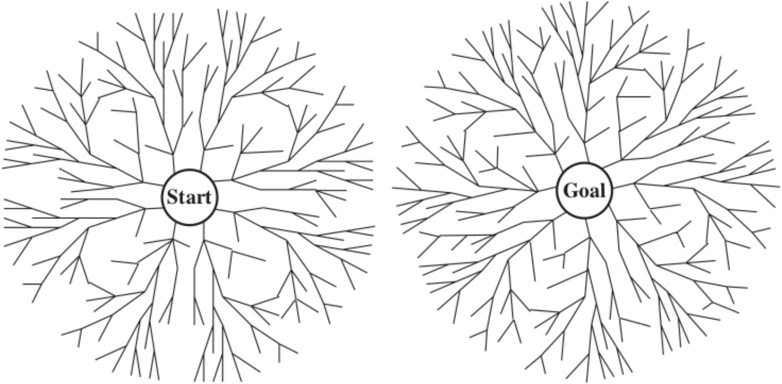
- UCS jest na sztucznej inteligencji, Dijkstra na algorytmach (to oczywiście nie jest poważna różnica).
- UCS jest przedstawiany najczęściej jako instancja algorytmu typu Best First Search
- Graf który przeszukujemy może być duży, nieznany w całości, nieskończony, itd.

Przeszukiwanie dwukierunkowe

Pomysł

Prowadźmy poszukiwania jednocześnie od przodu i od tyłu

Rysunek:



Przeszukiwanie dwukierunkowe. Problemy i korzyści

Problemy

Nie zawsze jest możliwe do zastosowania:

1. Musimy znać stan końcowy (vide hetmany czy obrazki logiczne)
2. Najlepiej jak jest jeden (albo niewiele i umiemy je wszystkie wymienić)
3. Musimy umieć odwrócić funkcję następnika (vide problem Knutha i funkcja `int ( )`)
4. Musimy pamiętać odwiedzone stany (przynajmniej z jednej strony)
   BFS + IDS (lub BFS + BFS) zamiast IDS+IDS

Korzyści

Podstawowa korzyść to czas działania. Dlaczego?
Odpowiedź: Zamiast jednego przeszukania na głębokości d mamy dwa przeszukania na głębokości $d/2$.

Przeszukiwanie bez wiedzy. Podsumowanie

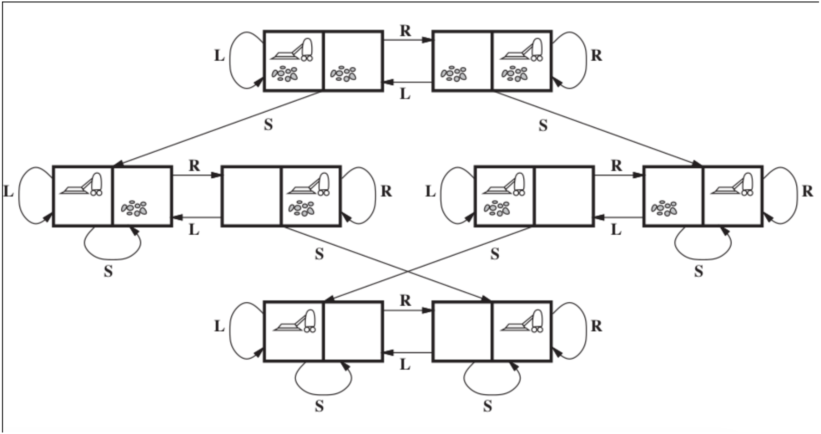
Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^a	Yes^{a,b}	No	No	Yes^a	Yes^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes^c	Yes	No	No	Yes^c	Yes^{c,d}

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.

Problemy bezczujnikowe (sensorless)

- Czujniki są drogie. Czasem wolimy na przykład znaleźć sekwencje akcji, która doprowadzi do celu niezależnie od stanu.
- Przykład 1 Szeroko działający antybiotyk
- Przykład 2 Robot w linii produkcyjnej, który składa jakieś części wykonując akcje niezależne od tego, jak te części się ułożyły.
- (czasem akcje są „puste” i generalnie robi ich się trochę za dużo)

Wszyscy wiemy o inteligentych odkurzaczach. Ten będzie trochę prostszy (rysunek z przestrzenią stanów):



Przestrzeń przekonań

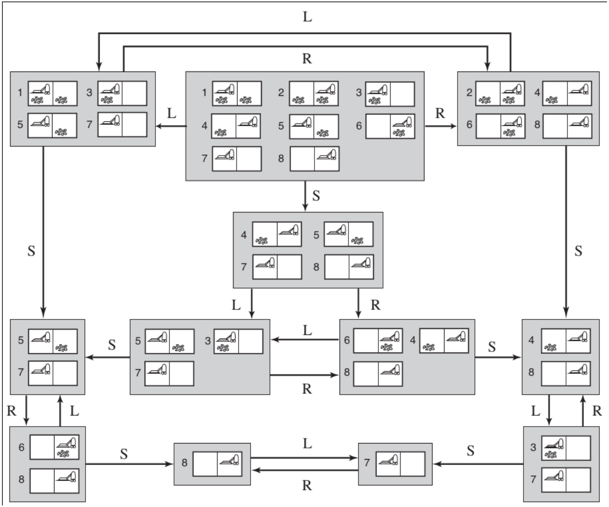
Definicja

Stanem przekonań jest zbiór stanów oryginalnego problemu, w których agent (być może) się znajduje.

Pytanie

Jak się poruszać w takiej przestrzeni?

Przestrzeń przekonań odkurzacza. Przykład



(pętle dla wszystkich stanów usunięte ze względu na czytelność.)

Graf przestrzeni przekonań

1. Przejścia w przestrzeni przekonań powstają przez zaaplikowanie funkcji przejścia do stanu (obliczenia obrazu funkcji)
2. Stan jest końcowy jeżeli wszystkie stany w nim zawarte są końcowe.
3. Koszt jednostkowy (spory problem w innym przypadku)
4. Stan startowy: zbiór wszytskich stanów.

Komandos z mapą. Mniej trywialny przykład

- Rozważmy zadanie, w którym do labiryntu wrzucony zostaje komandos z mapą...
- ale zrzut jest w nocy i nie wiadomo, gdzie trafił.
- Problem:

  *znajdź sekwencję akcji, która **na pewno** doprowadzi do jednego z celów (akcje niedozwolone nie przesuwają komandosa).*

Komandos. Jak go rozwiązać

- Zadanie z komandosem będzie na liście P2.
- Warto zatem poświęcić chwilę na „zbadanie” jak działa taka przestrzeń przekonań.

Zmniejszanie niepewności

Zobaczmy, jakie są możliwości zmniejszania niepewności w tym zadaniu (program `commando.py`).

Dodatkowa wiedza o problemie

- Opłaca się iść w kierunku rozwiązania.
- Co to oznacza?

Zakładamy, że umiemy szacować odległość od rozwiązania.

Przykłady

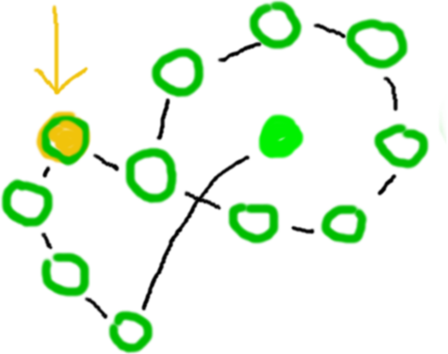
1. Odległość w linii prostej w zadaniu szukania drogi.
2. Odległość taksówkowa (Manhattan distance) w labiryncie.

Przeszukiwanie zachłanne

- Rozwijamy ten węzeł, który wydaje się najbliższu rozwiązania.
- Proste, intuicyjne, ale są problemy. Jakie?

Można ten algorytm „oszukiwać”, w skrajnym przypadku sprawić, żeby rozwiązanie w ogóle nie zostało znalezione.

Plansza nieprzyjazna dla algorytmu zachłannego



Algorytm A*

Definicje

- $g(n)$ – koszt dotarcia do węzła n
- $h(n)$ – szacowany koszt dotarcia od n do (najbliższego) punktu docelowego ($h(s) > 0$)
- $\mathrm{f(n)} = \mathrm{g(n)} + \mathrm{h(n)}$

Algorytm

Przeprowadź przeszukanie, wykorzystując $f(n)$ jako priorytet węzła (czyli rozwijamy węzły od tego, który ma najmniejszy f).

Wymagania dla heurystyki

Oczywiście od wyboru funkcji h zależą właściwości algorytmu A^*

Wymienimy najważniejsze właściwości funkcji h.

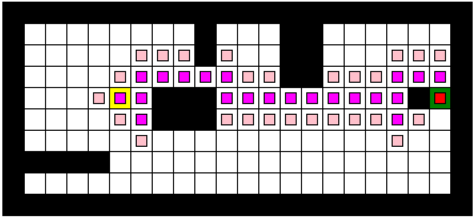
1. **Rozsądna**: $h(s_{\text{end}}) = 0$
2. **Dopuszczalna** (admissible):
   $h(s) <$ prawdziwy koszt dotarcia ze stanu s do stanu końcowego
   Inaczej: **optymistyczna**
3. **Spójna** (consistent), s_1, s_2 to sąsiednie stany:

$$h(s_2) + \text{cost}(s_1, s_2) \geq h(s_1)$$

Ostatnia własność przypomina własność trójkąta w definicji metryki.

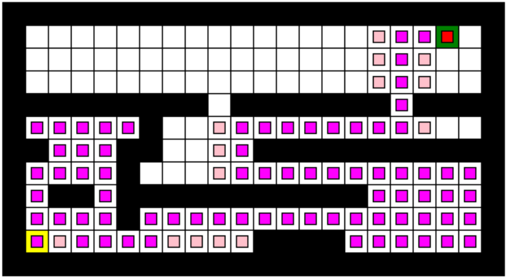
Kilka prostych konsekwencji

1. UCS to A^* z super-optymistyczną heurystyką (h(s) $= 0$)
2. Spójna heurystyka jest optymistyczna
   Dowód: Indukcja po węzłach (na ćwiczeniach)
3. Wyżej wymienione heurystyki (Manhattan, Euklidesowa) są optymistyczne, spójne i rozsądne.



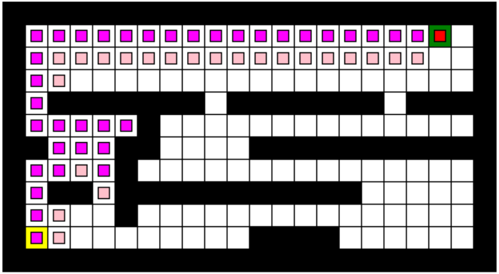
Jedynie dwa rozwinięte węzły poza optymalną ścieżka.

A^* w labiryncie (2)



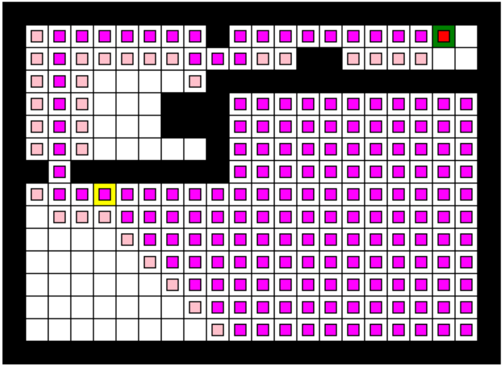
W dolnej części labiryntu heurystyka trochę prowadzi na manowce

A^* w labiryncie (3)



ale jeżeli w poprzednim labiryncie przebić drzwi, to wówczas znowu prawie idealnie.

A^* w labiryncie (4)



Heurystyka mocno „oszukana” przebiegiem labiryntu.

Właściwości A^*

Twierdzenie 1

A^* jest zupełny (warunki jak dla UCS).

Twierdzenie 2

Jeżeli h jest spójna, to A^* zwraca optymalną ścieżkę (wersja grafowa)

Twierdzenie 3

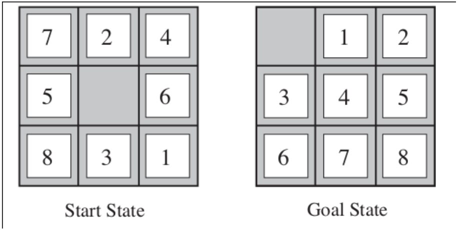
Jeżeli h jest optymistyczna, to A^* w drzewie zwraca optymalną ścieżkę.

Dowody: za chwilę, najpierw jeszcze trochę praktyki.

Heurystyki dla ósemki

Uwaga

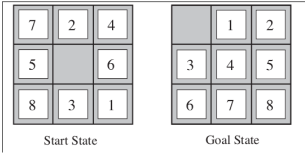
Pewne aspekty tworzenia heurystyk można dość dobrze prześledzić na przykładzie ósemki



Pytanie

Jak (optymistycznie) oszacować odległość tych dwóch stanów?

Heurystyki dla ósemki (2)



Pomysł 1

Jak coś jest nie na swoim miejscu, to musi się ruszyć o co najmniej 1 krok. Zliczajmy zatem, ile kafelków jest poza punktem docelowym ($h_1(s) = 8$)

Pomysł 2

Jak coś jest nie na swoim miejscu, to musi pokonać cały dystans do punktu docelowego. Zliczajmy zatem, ile kroków od celu jest każdy z kafelków ($h_2(s) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$)

Pytanie

Która intuicyjnie jest lepsza?

Relaksacja

Kiedy możliwy jest ruch w łamigłówce ósemka? Docelowe pole jest: (koniunkcja warunków):

a) sąsiadujące

b) wolne

Możemy rezygnować z części (lub wszystkich) warunków, otrzymując łatwiejsze łamigłówki.

Uwaga

Liczba ruchów w łatwiejszym zadaniu od startu do punktu docelowego jest często sensowną heurystyką w zadaniu orygialnym.

Heurystyka h_1

Ruch możliwy jest **zawsze**.

Heurystyka h_2

Ruch możliwy jest **gdy pole jest obok** (niekoniecznie puste).

Heurystyka h_3

Ruch możliwy jest **gdy pole jest puste** (niekoniecznie obok).

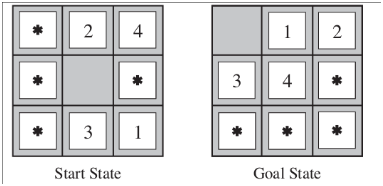
Efektywność w praktyce

- Dla h_2 efektywność A^* jest 50000 razy większa niż IDS.
- Istnieją heurystyki dające jeszcze 10000 krotne przyspieszenie dla 15-ki, a milionowe dla 24-ki (wobec h_2)

Bazy wzorców

Heurystyki możemy budować korzystając z baz wzorców, zapamiętujących koszty rozwiązań podproblemów danego zadania.

Przykład:



Działanie bazy wzorców

- Znajdujemy wszystkie podproblemy dla danego stanu (które mamy w bazie)
- A następnie bierzemy maksimum kosztów jako wartość heurystyki
- Możemy do tego maksimum dołożyć jakieś proste heurystyki (typu h_2).

Pytanie

A czy nie moglibyśmy użyć sumowania, zamiast maksimum?

Działanie bazy wzorców (2)

- Niestety suma daje niedopuszczalne heurystyki (bo pewne ruchy liczymy wielokrotnie, gwiazdki w jednym wzorcu są istotnymi kafelkami w innym)
- Da się temu zapobiec, stosując „rozłączne” wzorce (nic się nie powtarza) i w każdym wzorcu liczyć tylko ruchy kafelków z liczbami.
  To to są te najefektywniejsze heurystyki dla 8-ki

Uwaga

Niemniej warto wiedzieć, że czasem rezygnuje się z optymalności i stosuje niedopuszczalne heurystyki (które czasem przeszacowują odległość), ze względu na szybkość działania.

Własności A^*

Plan

Spróbujemy dowieść następujących rzeczy:

1) A^* zwraca najkrótszą drogę
2) A^* jest zupełny.

Dowód optymalności

Potrzebujemy dwóch faktów:

F1. Jeżeli h jest spójna, wówczas na każdej ścieżce wartości f są niemalejące.
D-d (n' jest następnikiem n):

$$f(n') = g(n') + h(n') = g(n) + c(n, n') + h(n') \geq g(n) + h(n) = f(n)$$

F2. Zawsze, gdy algorytm bierze węzeł do rozwinięcia, to koszt dotarcia do tego węzła jest optymalny (najmniejszy możliwy).

Dowód F2

- Bierzemy nie wprost węzeł n, do którego kolejne dotarcie daje mniejszy koszt niż dotarcie pierwsze.
- Wartość f dla tego węzła drugi raz jest mniejsza (h takie same, g mniejsze)
- Na ścieżce od początku do n-a drugiego mamy „różowy" węzeł n'
- Z własności F1 mamy: $f(n') < f(n_1)$

Zatem algorytm powinien wybrać n' a nie n_1. **Sprzeczność.**

Optymalność

Popatrzmy na pierwszy znaleziony węzeł docelowy (n_{end})

- $f(n_{\text{end}}) = g(n_{\text{end}}) + h(n_{\text{end}}) = g(n_{\text{end}})$ (bo h jest rozsądna)
- Każdy kolejny węzeł docelowy jest nielepszy, bo dla niego $g(n) \geq g(n_{\text{end}})$

Zupełność

Niech C^* będzie kosztem najtańszego rozwiązania ($g(n_{\text{end}})$)

- Algorytm ogląda wszystkie węzły, dla których $f(n) < C^*$
- Być może oglądnie również pewne węzły z konturu $f(n) = C^*$, przed wybraniem docelowego n, t.że $f(n) = g(n) + 0 = C^*$

Uwaga

Skończona liczba węzłów o $g(n) \leq C^*$ gwarantuje to, że algorytm się skończy. Do skończoności z kolei wystarczy założyć, że istnieje $\varepsilon > 0$, t.że wszystkie koszty są od niego większe bądź równe.

Lepsze heurystyki

Uwaga

A^* nie rozwija węzłów t.że $f(n) > C^*$. Zatem im większa h (przy założeniu spełniania warunków dobrej heurystyki), tym lepsza.

Konsekwencja

Mając dwie spójne (optymistyczne) heurystyki h_1 i h_2, możemy stworzyć $h_3(n) = \max(h_1(n), h_2(n))$, która będzie lepsza od swoich składników (szczegóły na ćwiczeniach).