Rozwiązywanie problemów więzowych

Paweł Rychlikowski

Instytut Informatyki UWr

29 marca 2019

Algorytm A*. Właściwości i wątpliwości

Definicje

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

Algorytm

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

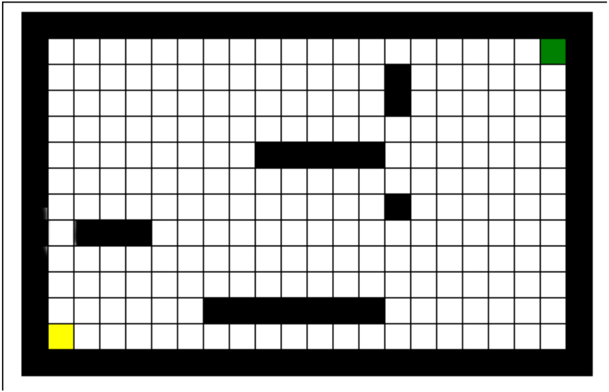
Kluczowa właściwość

1. A^* rozwija wszystkie węzły t.że $f(n) < C^*$.
2. A^* rozwija niektóre węzły t.że $f(n) = C$
3. A^* nie rozwija węzłów t.że $f(n) > C^*$.

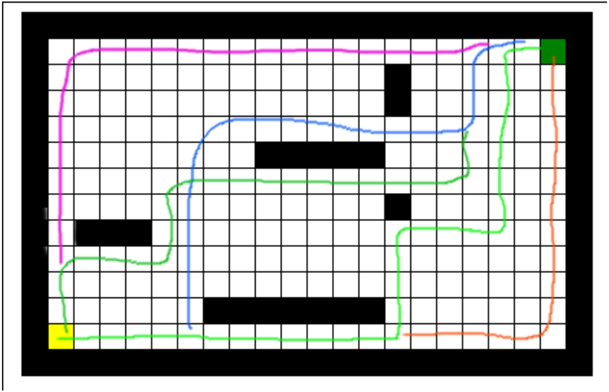
Pytanie

Co się będzie działo, jeżeli nasza funkcja h będzie liczyła dokładną odległość od celu?

Bierzemy heurystykę Manhatańską (przyjmijmy, że cel jest jeden), czyli $h(n) = |g_x - n_x| + |g_y - n_y|$



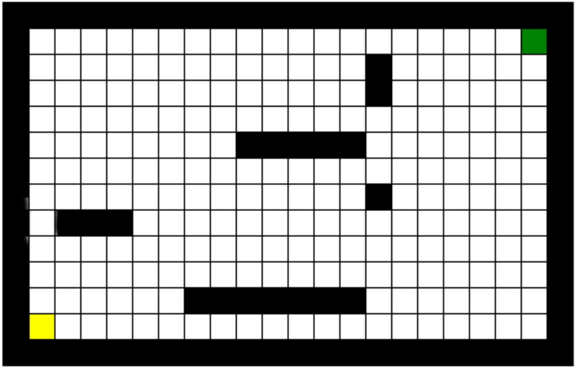
Płaska funkcja f



Wszystkie ścieżki idące w prawo i do góry są optymalne. Funkcja f jest stała.

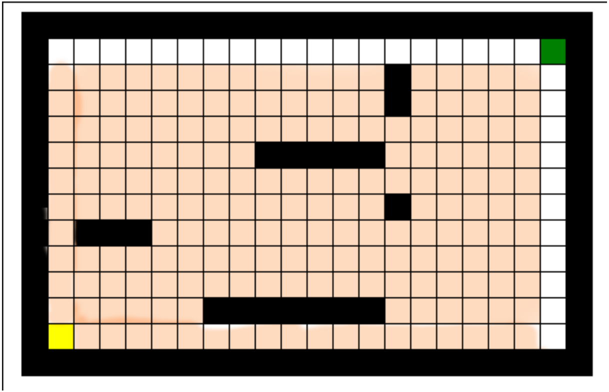
Porównanie heurystyk

Porównajmy na przykładzie heurystykę manhatańską i euklidesową. Która jest lepsza? **Poniżej zaznaczone węzły, które** **na pewno** **musi obejrzeć** A^* **z heurystyką Euklidesową**



Porównanie heurystyk

Porównajmy na przykładzie heurystykę manhatańską i euklidesową. Która jest lepsza? **Poniżej zaznaczone węzły, które na pewno musi obejrzeć A^* z heurystyką Euklidesową**



Heurystyki niedopuszczalne w praktyce

- **Pytanie**: Jaka jest najprostsza heurystyka niedopuszczalna
- **Odpowiedź**: $(1+\varepsilon)h(n)$, gdzie h jest dopuszczalna

Czy ma ona jakiś sens?

Dla małego ϵ będziemy rozstrzygać remisy oryginalnej funkcji f preferując węzły, które wydają się być bliższe celowi.

Spójność więzów. Przykład

Więz: $X < Y$
Dziedzina X: $\{4, 6, 7, 10, 20\}$
Dziedzina Y: $\{1, 2, 4, 6, 7, 10\}$
Brak spójności

- Jeżeli weźmiemy X, to możemy wykreślić wartośći 10, 20
- Jeżeli weźmiemy Y, to możemy wykreślić wartości 1,2, 4

Po wykreśleniu tych wartości warto przyjrzeć się innym więzom z X i Y.

Algorytm AC-3

Algorytm zapewnia spójność łukową sieci więzów.

Idea

1. Zarządzamy kolejką więzów,
2. Usuwamy niepasujące wartości z dziedzin, analizując kolejne więzy z kolejki,
3. Po usunięciu wartości z dziedziny B, sprawdzamy wszystkie zmienne X, które występują w jednym więzie z B

Algorytm AC-3

function AC-3(*csp*) **returns** false if an inconsistency is found and true otherwise
  inputs: *csp*, a binary CSP with components $(X,\ D,\ C)$
  local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

  while *queue* is not empty **do**
    $(X_i,\ X_j) \leftarrow$ REMOVE-FIRST(*queue*)
    if REVISE(*csp*, $X_i,\ X_j$) **then**
      if size of $D_i\ =\ 0$ **then return** *false*
      for each X_k **in** X_i.NEIGHBORS - $\{X_j\}$ **do**
        add $(X_k,\ X_i)$ to *queue*
  return *true*

---

function REVISE(*csp*, $X_i,\ X_j$) **returns** true iff we revise the domain of X_i
  revised $\leftarrow$ *false*
  for each x **in** D_i **do**
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j **then**
      delete x from D_i
      revised $\leftarrow$ *true*
  return *revised*

Algorytm AC-3. Uwagi

- Zwróćmy uwagę na niesymetryczność funkcji Revise (oznacza ona konieczność dodawania każdej pary zmiennych dwukrotnie)
- Zwróćmy uwagę, że istotny jest efekt uboczny tej funkcji: zmieniają się wartości dziedzin!

Złożoność algorytmu AC-3

- Mamy n zmiennych, dziedziny mają wielkość $O(d)$. Mamy c więzów.
- Obsługa więzu to $O(d^2)$
- Każdy więz może być włożony do kolejki co najwyżej $O(d)$ razy.

Złożoność

Złożoność wynosi zatem $O(cd^3)$ (raczej pesymistycznie)

AC-3 a obrazki logiczne

- Obrazki logiczne da się zapisać jako sieć więzów, z więzami typu:

$$\text{wiersz}_{[2,3,3]}(B_1, \ldots, B_n)$$

  dla wierszy i kolumn.
- Ale te więzy mają dużą arność, a my chcemy więzów binarnych.

Opcje

a) Utworzyć problem dualny (całkiem sensowna)

b) Połączyć zmienne z problemu dualnego i oryginalne.

AC-3 i obrazki logiczne

- Mamy zmienne B_i odpowiadające poszczególnym kratkom,
- Mamy zmienne K_i odpowiadające kolumnom i W_i odpowiadające wierszom
- Zwróćmy uwagę, że za definicję zadania w zasadzie odpowiadają więzy unarne na zmiennych K oraz W.
- Musimy powiązać kratki, wiersze i kolumny:

$$B_{ij} \text{ jest-elementem-j } W_i$$

oraz

$$B_{ij} \text{ jest-elementem-i } K_j$$

Uwaga

Binarna sieć więzów, zatem można stosować AC-3.

- Z wiersza (kolumny) do pola: pole B_{ij} musi mieć wartość b, bo wszystkie wartości W_i mają na j-tym polu b
- Z pola do wiersza (kolumny): skoro pole B_{ij} ma wartość b, to możemy wykreślić wszystkie układy z dziedziny K_j, które nie mają na i-tej pozycji b.

Dokładnie tak rozwiązują obrazki logiczne ludzie. Ale to nie zawsze doprowadzi do sukcesu...

Poszukiwanie z nawrotami dla problemów więzowych

przeszukiwanie z nawrotami = backtracking search

- Wariant przeszukiwania w głąb, w którym stanem jest niepełne podstawienie.
- Nie pamiętamy całej historii, ale potrafimy zrobić **undo**
- Po każdym przypisaniu wykonujemy jakąś formę wnioskowania, bo może da się zmniejszyć dziedziny...

Backtracking

```
function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment then
          add {var = value} to assignment
          inferences ← INFERENCE(csp, var, value)
          if inferences ≠ failure then
             add inferences to assignment
             result ← BACKTRACK(assignment, csp)
             if result ≠ failure then
                return result
      remove {var = value} and inferences from assignment
  return failure
```

1. Możliwy jest też taki wariant, że najpierw uruchamiamy AC-3, potem Backtracking z jakimś uproszczonym wnioskowaniem.
2. Wnioskowanie może nie tylko wykreślać elementy z dziedziny, może również dodawać inne więzy (implikacje)

W wielu sytuacjach, jak mechanizm wnioskowania jest silny, to wykonywane jest bardzo niewiele zgadnięć.

Parametry backtrackingu

1. Jak wybieramy zmienną do podstawienia (SelectUnassignedVariable)
2. W jakim porządku sprawdzamy dla niej wartości (OrderDomainValue)
3. Jak przeprowadzamy wnioskowanie (Inference)

Przykład. Plan lekcji

- Rozmieszczamy lekcje: zajęcia otrzymują termin
- Mamy naturalne więzy:
    - Jeżeli Z_1 i Z_2 mają tego samego nauczyciela (klasę, salę), wówczas $Z_1 \neq Z_2$
    - Nauczyciele nie mogą mieć zajęć o określonych porach (bo na przykład pracują w innych miejscach)
    - Wszystkie zajęcia klasy X danego dnia spełniają określone warunki: brak okienek, po jednej godzinie przedmiotu, itd.

Pytanie

W jakiej kolejności rozmieszcza zajęcia Pani Sekretarka?

Heurystyka: First Fail

Definicja

Wybieramy tę zmienną, która jest najtrudniejsza, co oznacza, że:

- ma najmniejszą dziedzinę,
- występuje w największej liczbie więzów.

Inne nazwy: Most Constrained First, Minimum Remaining Values (MRV)

Uzasadnienie

I tak będziemy musieli tę zmienną obsłużyć. Lepiej to zrobić, jak jeszcze inne zmienne są „wolne”

Wybór wartości

- Wybieramy tę wartość, która w najmniejszym stopniu ogranicza przyszłe wybory LCV, Least Contstraining Value.
- Przykład. W planie zajęć:
  1. Mamy teraz przydzielić termin zajęć panu A z klasą 1c
  2. Musimy później przydzielić zajęcia A z klasą 2a.
  3. Wcześniej przydzieliliśmy panią B z klasą 2a w czwartek na 8.
  4. Jest to argument za tym, żeby (A,1c) też była na ósmą w czwartek (bo nie stracimy żadnej możliwości dla (A,2a).

Wybór zmiennej vs wybór wartości

- W pierwszej chwili może dziwić przeciwne traktowanie wyboru zmiennych i wartości.
- Celem FirstFail jest agresywne ograniczanie przestrzeni poszukiwań.
- Celem LCV jest dążenie do jak najszybszego znalezienia pierwszego rozwiązania.

Musimy rozpatrzeć wszystkie zmienna, ale niekoniecznie wszystkie wartości!

Wybór zmiennej vs wybór wartości. Podsumowanie

- Wybieramy najgorszą zmienną
  (ale każdą kiedyś musimy wybrać, a ta najgorsza najbardziej utrudni nam dalsze wybory)
- Wybieramy najlepszą wartość
  (ale często zależy nam na znalezieniu pierwszego rozwiązania, nie wszystkich)

Więzy i maksymalizacja wartości

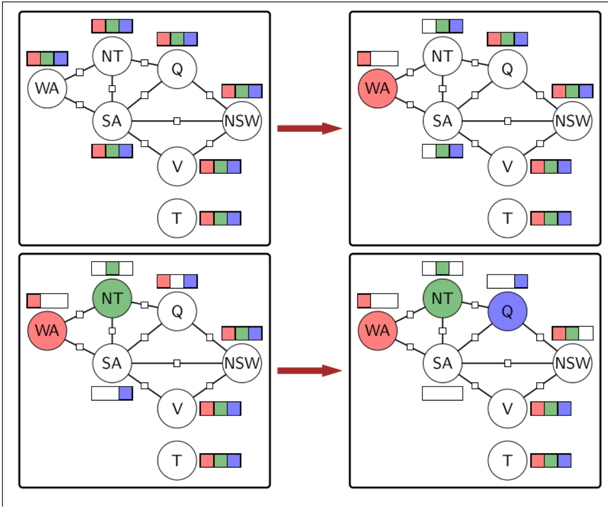
- Czasami do problemu więzowego dodajemy dodatkowo zadanie maksymalizacji wartości pewnej funkcji:
  - Przydział robotników do maszyn spełniający określone wymagania i maksymalizujący produktywność.
  - Poprawny plan lekcji, maksymalizujący liczbę spełnionych miękkich wymagań nauczycieli (np. wolałbym nie mieć zajęć w piątek po 12, ale ...)

Uwaga

W takich sytuacjach wybierając wartość bardzo często maksymalizujemy lokalne „zadowolenie” z rozwiązania.

- AC-3 może być kosztowne.
- Uproszczona forma: Forward Checking:
  - Zawsze, jak przypiszemy wartość, sprawdzamy, czy to przypisanie nie zmienia dziedzin innych zmiennych (które są w więzach z obsługiwaną zmienną)
  - I tu zatrzymujemy wnioskowanie.

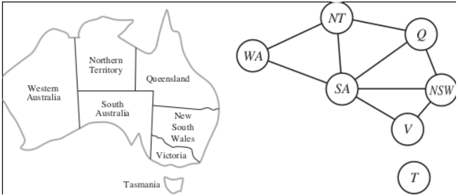
Forward Checking – przykład



Źródło: CS221: Artificial Intelligence: Principles and Techniques

First Fail w praktyce

- W kolorowaniu Australii wszystkie dziedziny na początku są równe...
- ale heurystyka First Fail w drugiej kolejności patrzy na liczbę więzów.



Wybór SA pozwala nam dalsze przeszukiwanie robić bez nawrotów.

Więzy globalne (1)

- Więzy globalne to takie, które opisują relacje dużej liczby zmiennych (np. klasa nie ma okienek)
- Dobrym przykładem jest więz alldifferent$(V_1, \ldots, V_n)$

Uwaga

Oczywiście da się wyrazić równoważny warunek za pomocą $O(n^2)$ więzów $V_i \neq V_j$.

Propagacja więzów globalnych

Przykład

Mamy taką sytuację: $X \in \{1, 2\}, Y \in \{1, 2\}, Z \in \{1, 2\}$,
Więzy: $X \neq Y, Y \neq Z, X \neq Z$

- Spójny łukowo (niemożliwa propagacja)
- Globalne spojrzenie umożliwia stwierdzenie, że wartości nie starczy

Daje to prosty algorytm wykrywania sprzeczności więzów (porównanie sumy mnogościowej dziedzin i liczby zmiennych).

Constraint Logic Programming

- Pewna część uczestników miała Prolog na Metodach programowania.
- Spróbujemy powiedzieć o programowaniu logicznym z więzami mówiąc maksymalnie mało o samym programowaniu logicznym
- o którym z kolei coś powiemy, jak będziemy zajmowali się logiką.

Uwaga

Możemy (na płytkim poziomie) potraktować CLP jako constraint solver, czyli system, w którym definiujemy zadanie więzowe i otrzymujemy rozwiązanie.

Przykładowe systemy CLP

- SWI-Prolog (ma moduł clpfd)
- GNU-Prolog (trochę stary i nierozwijany)
- Eclipse (http://eclipseclp.org/)

Więzy w SWI-Prolog.

- Zmienne FD (clpfd)
- Zmienne boolowskie (clpb)
- Zmienne rzeczywiste i wymierne (clpr)

Zajmiemy się tylko zmiennymi FD.

clp(X)

Rozważa się również inne X-y: napisy, zbiory, przedziały.

Składowe zadania w CLP

Przypominamy: musimy określić zmienne, ich dziedziny oraz więzy na nich.

Zmienne

Zmienne są zmiennymi prologowymi, piszemy je wielką literą.

Dziedziny

```
V in 1..10
[A,B,C,D] ins 1..10
```

Więzy

Języki CLP mają bardzo bogate możliwości wyrażania problemów za pomocą więzów.

Przyślijcie Więcej Pieniędzy


```
puzzle(Vars) :-
      Vars = [S,E,N,D,M,O,R,Y],
      Vars ins 0..9,
      all_different(Vars),
                S*1000 + E*100 + N*10 + D +
                M*1000 + O*100 + R*10 + E #=
      M*10000 + O*1000 + N*100 + E*10 + Y,
      M #\= 0, S #\= 0.
```

Więzy arytmetyczne

- Mają postać: `<wyrażenie> <operator-rel> <wyrażenie>`
- Operatory relacji to: #= #> #>= #< #<= #\=
  Uwaga na znaki # przy symbolach relacyjnych!
- Wyrażenia zbudowane standardowo z + - * abs min max mod // (i paru innych)

Uwaga

Dodanie znaku # mówi, że dany warunek jest więzem i należy go specjalnie traktować. W szczególności:

- `X > Y` sprawdzamy od razu,
- `X #> Y` odkładamy do magazynu więzów