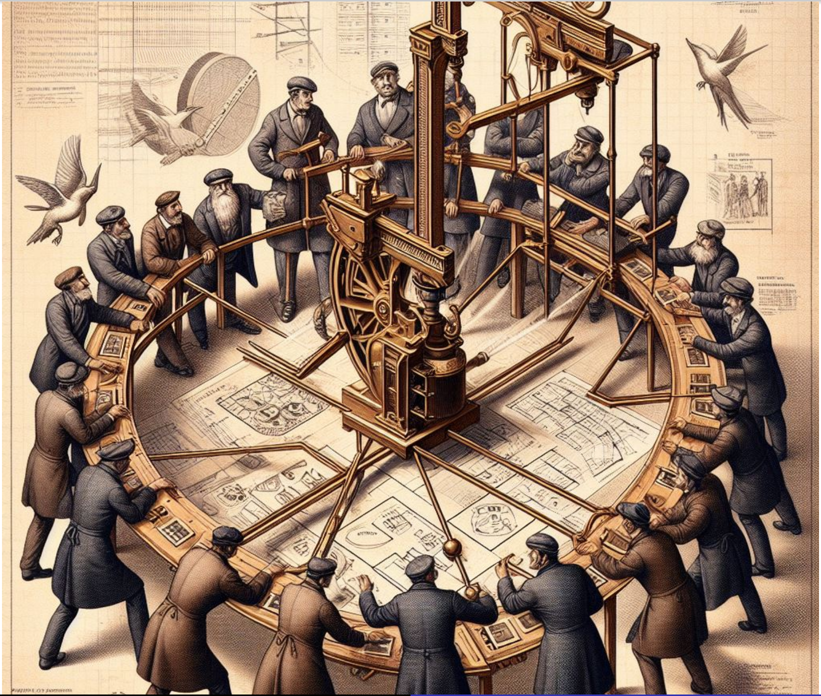
O więzach ciąg dalszy

Paweł Rychlikowski

Instytut Informatyki UWr

21 marca 2024



Problemy spełnialności więzów. Przypomnienie definicji

Definicja

Problem spełnialności więzów ma 3 komponenty:

1. Zbiór zmiennych $X_1, \ldots, X_n$
2. Zbiór dziedzin (przypisanych zmiennym)
3. Zbiór więzów, opisujących dozwolone kombinacje wartości jakie mogą przyjmować zmienne.

Przykład

Zmienne: X, Y, Z
Dziedziny: $X \in \{1, 2, 3, 4\}, Y \in \{1, 2\}, Z \in \{4, 5, 6, 7\}$
Więzy: $X + Y \geq Z, X \neq Y$

Spójność łukowa

Definicja

Więz C dla zmiennych X i Y z dziedzinami D_X i D_Y jest **spójny łukowo**, wtt:

- Dla każdego $x \in D_X$ istnieje takie $y \in D_Y$, że $C(x, y)$ jest spełnione
- Dla każdego $y \in D_Y$ istnieje takie $x \in D_X$, że $C(x, y)$ jest spełnione

Spójność łukowa

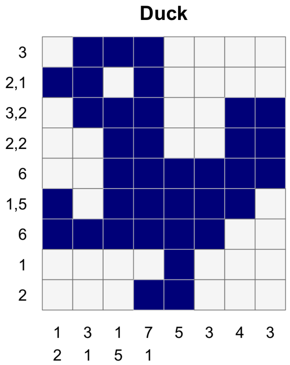
Definicja

Więz C dla zmiennych X i Y z dziedzinami D_X i D_Y jest **spójny łukowo**, wtt:

- Dla każdego $x \in D_X$ istnieje takie $y \in D_Y$, że $C(x, y)$ jest spełnione
- Dla każdego $y \in D_Y$ istnieje takie $x \in D_X$, że $C(x, y)$ jest spełnione

Sieć więzów jest spójna łukowo, jeżeli każdy więz jest spójny łukowo.

Spójność i obrazki logiczne



- Zmienne to wiersze i kolumny
- Spójność węzłowa (pojedynczy wiersz/kolumna zgodna ze specyfikacją)
- Spójność łukowa – zmiany w dziedzinie wierszu wpływają na kolumny i vice versa

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) ← 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 ← 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 ← 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)

Propagacja więzów

- Algorytm $AC - 3$ jest przykładowym algorytmem **propagacji więzów**
- Przeprowadzamy rozumowanie, które pozwala nam **bezpiecznie** usuwać zmienne z dziedziny.
- Zmniejszanie dziedziny zmiennej X może spowodować zmniejszenie dziedzin innych, związanych z nią zmiennych.

Proste rozwiązywanie więzów

Przypisuj wartości losowo i zarządzaj dziedzinami

Dla hetmanów na tablicy

Uwaga

Bardziej systematyczny sposób nawiązujący do tej metody nazywa się **backtrackingiem** (przeszukiwaniem z nawrotami)

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
```

(*assigment* zawiera również informacje o dziedzinach)

Backtracking. Uwagi

1. *remove* pewnie lepiej byłoby zastąpić po prostu zapamiętywaniem stanu poprzedniego i *undo*
2. Możliwy jest też taki wariant, że najpierw uruchamiamy AC-3, potem Backtracking z jakimś uproszczonym wnioskowaniem.
3. 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 bądź Pan Sekretarz?

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 X
  2. Musimy później przydzielić zajęcia A z klasą Y.
  3. Wcześniej przydzieliliśmy panią B z klasą Y w czwartek na 8.
  4. Jest to argument za tym, żeby (A,X) też była na ósmą w czwartek (bo nie stracimy żadnej możliwości dla (A,Y).

Wybór zmiennej vs wybór 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 ogranicza nam dalsze wybory)
- Wybieramy najlepszą wartość
  (ale często zależy nam na znalezieniu pierwszego rozwiązania, nie wszystkich)

Uwaga

Możliwe inne heurystyki preferujące najbardziej obiecujące wartości! (można myśleć, że w tu jest miejsce na dowolny sensowny zachłanny algorytm wybierający wartość)

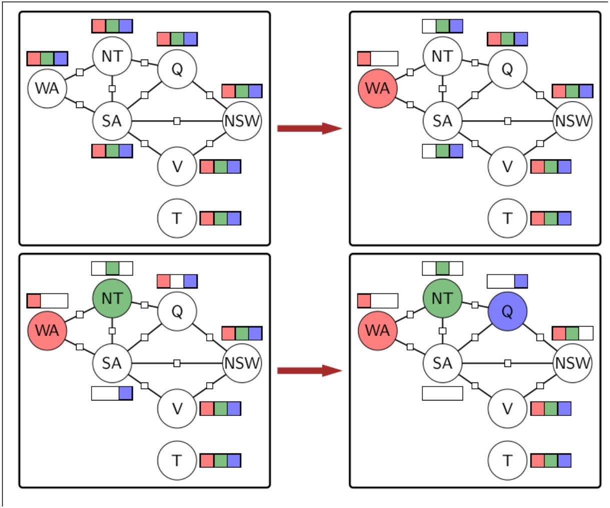
Przeplatanie poszukiwania i wnioskowania

- 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.

Uwaga

Coś takiego można wykorzystać jako pełnoprawny algorytm. Wystarczy dodać jakąś losowość i restarty.

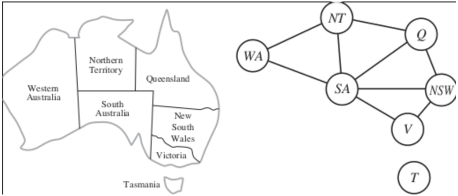
Forward Checking – przykład



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 (stosując Forward Checking).

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).

Przeszukiwanie lokalne dla CSP

- Przeszukiwanie lokalne nie próbuje systematycznie przeglądać przestrzeni rozwiązań (ogólniej: przestrzeni stanów)
- Zamiast tego pamięta jeden stan (lub niewielką, stałą liczbę stanów)
- Dla CSP stanem będzie kompletne przypisanie (niekoniecznie spełniające więzy).

Problemy optymalizacyjne

- W tych problemach szukamy stanu, który maksymalizuje wartość pewnej funkcji (jakość planu).
- Często problemy z twardymi warunkami da się zamienić na problemy optymalizacyjne. Jak?

Można policzyć liczbę złych wierszy (kolumn) w obrazkach logicznych, albo liczbę szachów w hetmanach, albo....

MinConflicts

Uwaga

Możemy myśleć o spełnianiu CSP jako o zadaniu maksymalizacji liczby spełnionych więzów.

Możemy zatem stworzyć algorytm, w którym:

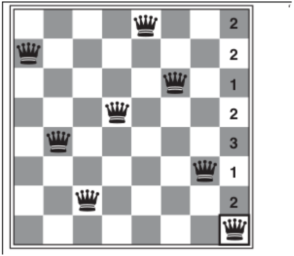
- Zmieniamy tę zmienną, która powoduje niespełnienie największej liczby więzów.
- Wybieramy dla niej wartość, która owocuje najmniejszą liczbą konfliktów.

Przykład: 8 hetmanów

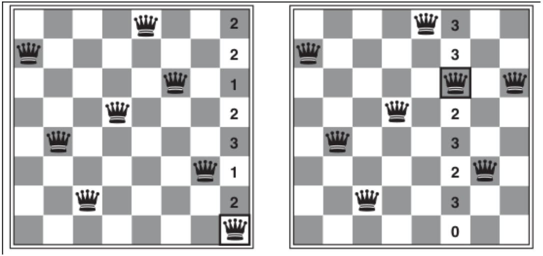
- Jak wybrać stan? (Wskazówka: powinniśmy umieć łatwo przejść ze stanu do stanu)
- Stan: w każdej kolumnie 1 hetman, Ruch: przesunięcie hetmana w górę lub w dół

Popatrzmy, jak działa min-conflicts dla hetmanów.

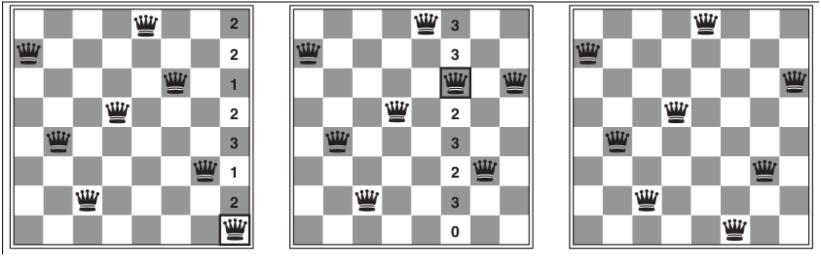
Min-conflicts dla hetmanów



Min-conflicts dla hetmanów



Min-conflicts dla hetmanów



Hetmany. Wyniki

- Dla planszy 8×8 osiąga sukces w 14% przypadków (tylko ruchy poprawiające).
- Niby niezbyt dużo, ale możemy go uruchomić na przykład 20 razy, wówczas p-stwo sukcesu to ponad 95%.
- Można dopuszczać pewną liczbę ruchów w bok (czyli, że nie musimy poprawić, ale wystarczy, że nie pogorszymy, jak na obrazkach).
- Jak dopuścimy ruchy w bok , to wówczas mamy sukces w 94%

Ważenie więzów

- Każdy więz ma wagę, początkowo wszystkie równe na przykład 1
- Waga więzów niespełnionych cały czas troszkę rośnie.
- Chcemy naprawiać nie zbiór więzów o liczności n, ale raczej zbiór więzów o największej sumarycznej wadze

Więzy trudne, rzadko spełniane będą miały coraz większy priorytet.

Więzy on-line

- Wyobraźmy sobie, że mamy problem, który się zmienia (ale w niewielkim stopniu)
- Przykład: obsługa linii lotniczych – bo zamykają się lotniska, pilot może złapać grypę, ...

On-line CSP

Min-conflicts umożliwia rozwiązywanie tego typu zadań: stan początkowy to ostatnie dobre przypisanie.