O więzach ciąg dalszy

Paweł Rychlikowski

Instytut Informatyki UWr

5 kwietnia 2023

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: $X + Y \geq Z, X \neq Y$

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 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 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)

Super słaby backtracking

- Zwróćmy uwagę, że zachłanny algorytm wybierający zmienną (trudną) i wartość (obiecującą) jest np. algorytmem układania planu (nawet bez backtrackingu).
- Z drugiej strony przestrzeń jest tak ogromna (kilkaset zajęć, każde w kilkudziesięciu terminach), że trudno spodziewać się, aby backtracking dał sobie z nią radę
(nawet backtracking na sterydach)

Super słaby backtracking

Limited Discrepancy Search

LDS (przeszukiwanie o ograniczonej rozbieżności) jest wariantem przeszukiwania, w którym jedynie d razy na całe przeszukanie, mamy prawo wziąć nie pierwszy najlepszy termin, lecz drugi! (d jest małe, rzędu 1,2,3)

Można myśleć o tym tak:

- Mamy ciąg decyzji (tyle ile zmiennych)
- Wybieramy d miejsc, w których podejmujemy „nieoptymalne” decyzje (z punktu widzenia heurystyki wyboru wartości)

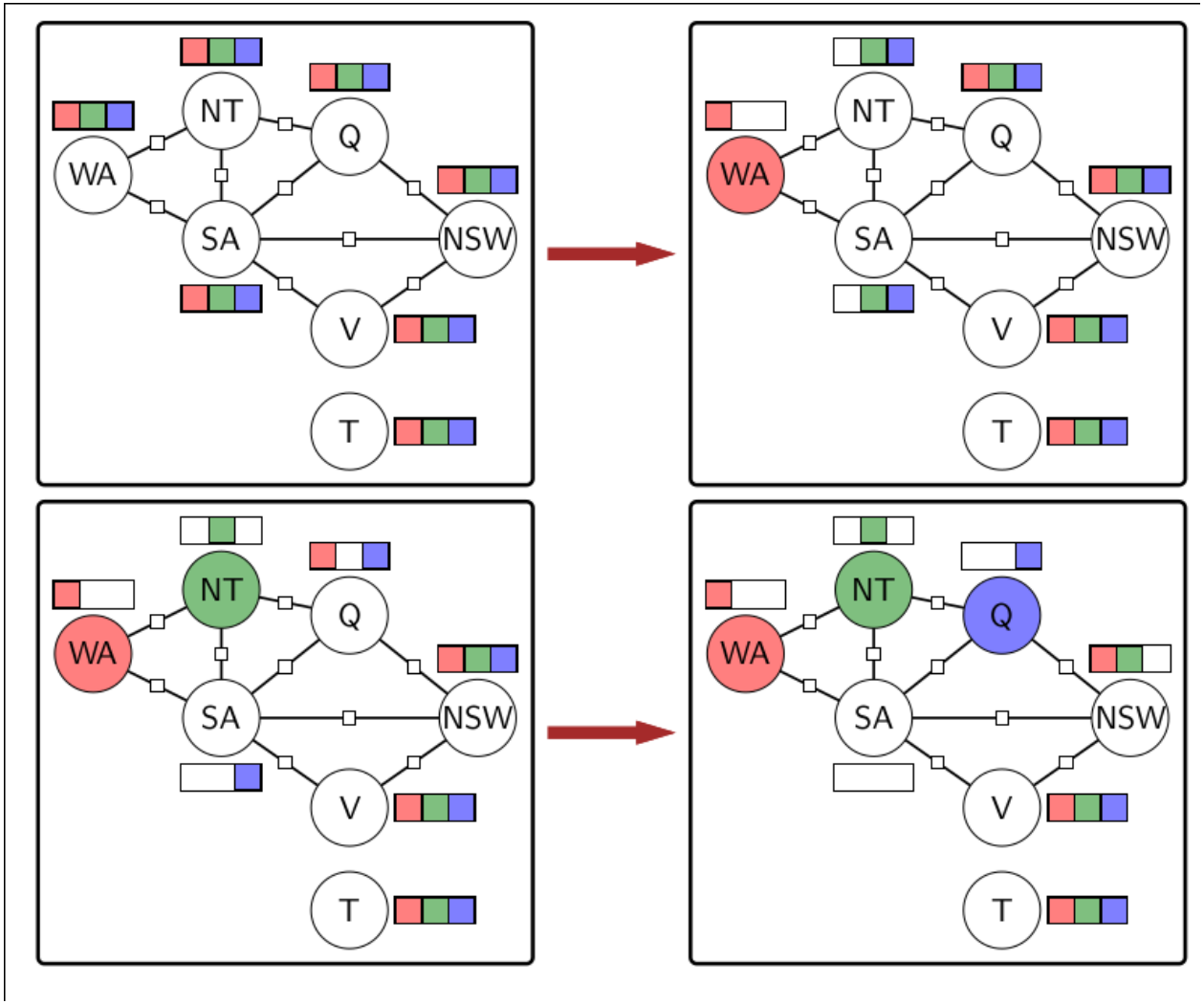
Przeplatanie poszukiwania i wnioskowania

- 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.

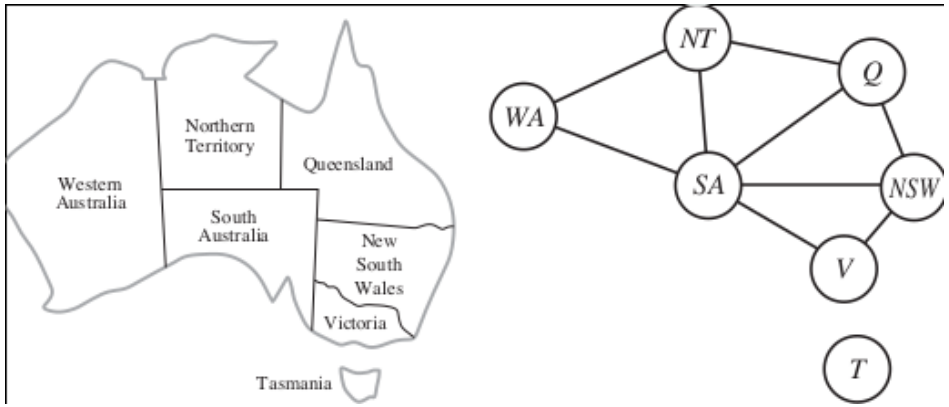
Forward Checking – przykład



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:

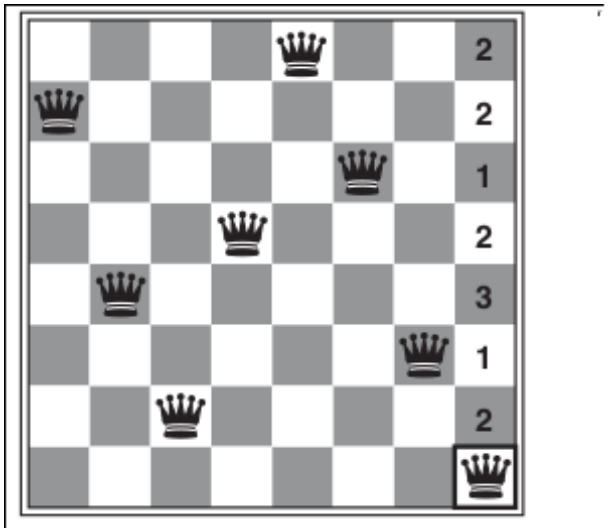
- Zmieniamy tę zmienną, która powoduje niespełnienie największej liczby więzów.
- Wybieramy dla niej wartość, która owocuje najmniejszą liczbą konfliktów.

Przykład: 8 hetmanów

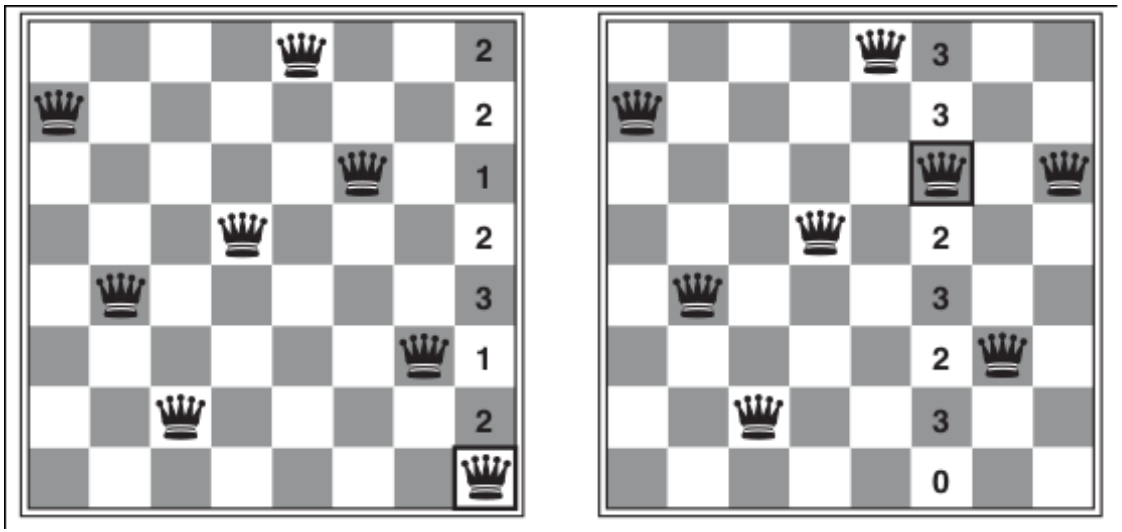
- Jak wybrać stan? (Wskazówka: powinniśmy umieć łatwo przejść ze stanu do stanu)
- Stan: w każdej kolumnie 1 hetman, Ruch: przesunięcie hetmana w górę lub w dół

Popatrzmy, jak działa min-conflicts dla hetmanów.

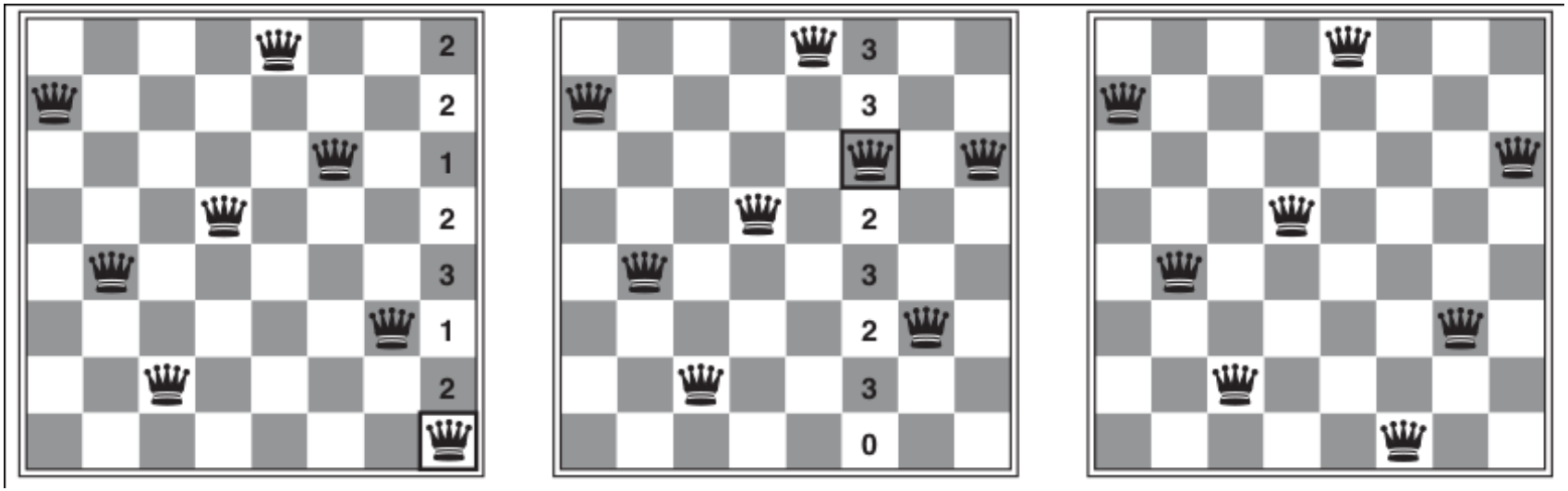
Min-conflicts dla hetmanów



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.

Constraint Logic Programming

- Spróbujemy powiedzieć o programowaniu logicznym z więzami mówiąc maksymalnie mało o samym programowaniu logicznym
- o którym z kolei trochę 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.

Constraint Solver

Deklaratywne podejście do programowania

- Wypisujemy więzy (w jakimś formalnym języku)
- (możemy się wspomóc programowaniem, więzów może być dużo)
- Rozwiązaniem zajmuje się Solver (nie musimy implementować propagacji więzów i backtrackingu)

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.
```

Wykorzystanie solwera więzowego

Postać programu CLP

```
name([V1, ..., Vn]) :-
      V1 in 1..K, ..., Vn in 1..K,
      V1 # >= V5, abs(V2+V6) # = abs(V7-V8),
      min(V3,V7) # > V3 + 2*V1,
      labeling([options], [V1,...,Vn]).
```

Wykorzystanie solwera więzowego

Postać programu CLP

```
name([V1, ..., Vn]) :-
      V1 in 1..K, ..., Vn in 1..K,
      V1 # >= V5, abs(V2+V6) # = abs(V7-V8),
      min(V3,V7) # > V3 + 2*V1,
      labeling([options], [V1,...,Vn]).
```

- Czyli mamy obsługę zmiennych i dziedzin, ustanowienie więzów oraz wywołanie przeszukiwania z nawrotami (labeling), a na końcu wywołanie głównego predykatu.

Wykorzystanie solwera więzowego

Postać programu CLP

```
name([V1, ..., Vn]) :-
      V1 in 1..K, ..., Vn in 1..K,
      V1 # >= V5, abs(V2+V6) # = abs(V7-V8),
      min(V3,V7) # > V3 + 2*V1,
      labeling([options], [V1,...,Vn]).
```

- Czyli mamy obsługę zmiennych i dziedzin, ustanowienie więzów oraz wywołanie przeszukiwania z nawrotami (labeling), a na końcu wywołanie głównego predykatu.
- Część czarna jest częścią *techniczną*, stanowiącą naszą *daninę* dla Prologa

Wykorzystanie solwera więzowego

Postać programu CLP

```
name([V1, ..., Vn]) :-
      V1 in 1..K, ..., Vn in 1..K,
      V1 # >= V5, abs(V2+V6) # = abs(V7-V8),
      min(V3,V7) # > V3 + 2*V1,
      labeling([options], [V1,...,Vn]).
```

- Czyli mamy obsługę zmiennych i dziedzin, ustanowienie więzów oraz wywołanie przeszukiwania z nawrotami (labeling), a na końcu wywołanie głównego predykatu.
- Część czarna jest częścią *techniczną*, stanowiącą naszą *daninę* dla Prologa
- Ten program możemy napisać, używając Ulubionego Języka Programowania – wystarczy, że ma `print`, `printf`, `puts`, ...

Testowanie hetmanów. Python + Prolog

- Zobaczmy, jak działa program queen_produce.py
- Jak wyglądają wynikowe programy
- Jak duże instancje jesteśmy w stanie rozwiązywać?