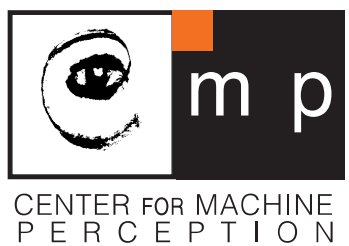
Constraint satisfaction problems

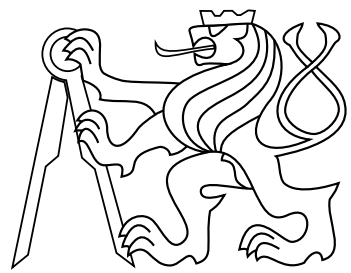
Problém splnitelnosti

Jan Kybic

http://cmp.felk.cvut.cz/~kybic
kybic@fel.cvut.cz

2016–2017





Constraint satisfaction problem (CSP)

Problém splnitelnosti

- Speciální případ prohledávání stavového prostoru — *struktura*
- Stav se skládá z *proměnných*.
- Cílové stavy musí splňovat *podmínky*.
- Cesta dosažení cíle není důležitá.

kniha *Russel, Norvig: Artificial Intelligence: A Modern Approach.*

Constraint satisfaction problem (CSP)

Problém splnitelnosti

- Speciální případ prohledávání stavového prostoru — *struktura*
- Stav se skládá z *proměnných*.
- Cílové stavy musí splňovat *podmínky*.
- Cesta dosažení cíle není důležitá.

$\rightarrow$ efektivnější algoritmy.

Obory proměnných nejčastěji diskrétní, konečné.

kniha *Russel, Norvig: Artificial Intelligence: A Modern Approach.*

Příklady CSP

- Logické hádanky
  - Problém 8 dam
  - Sudoku
  - Aritmogram *(TWO+TWO=FOUR)*
  - ‘Zebra’
- Barvení mapy
- Rozmístění objektů v pravoúhlé mřížce.
- Plánování/rozvrhování (např. školní rozvrh)

Definice

Problém splnitelnosti (CSP)

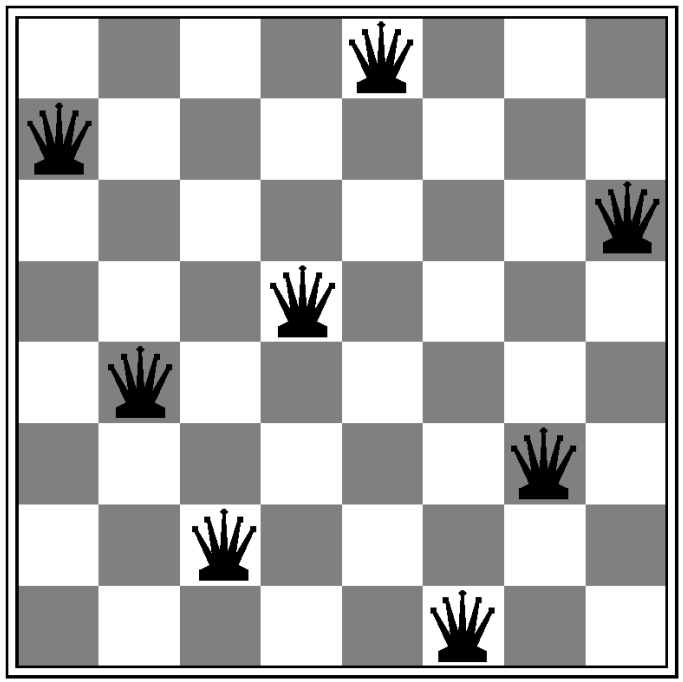
- Množina proměnných *(variables)* $x_1, x_2, \ldots, x_n$.
- Každá proměnná má svůj obor, $x_i \in D_i$,
- Množina podmínek *(constraints)* $C_1, C_2, \ldots, C_m$
  - Podmínka C_j se týká proměnných $x_{V_j(1)}, x_{V_j(2)}, \ldots, x_{V_j(N_j)}$
  - Podmínka $C_j \subseteq D_{V_j(1)} \times D_{V_j(2)} \times \cdot \times D_{V_j(N_j)}$ je splněna, pokud $x_{V_j(1)}, x_{V_j(2)}, \ldots, x_{V_j(N_j)} \in C_j$
  - Binární podmínky ($N_j = 2$) stačí.
- Hledáme takový hodnoty $x_1, x_2, \ldots, x_n$ aby všechny podmínky $C_1, C_2, \ldots, C_m$ byly splněny.

- Částečné/úplné přiřazení *(partial/complete assignment)*
- Platné přiřazení *(valid assignment)*

Problém n dam

Problem of n-queens

Na šachovnici o velikosti $n \times n$ umístěte n dam, aby se navzájem neohrožovaly. (Nesmí být ve stejné řadě, sloupci, ani diagonále.)



Algoritmy

- Zkoušení hrubou silou *(brute force search)*
- Využití struktury problému
- Postupné přiřazování *(progressive assignment)* + DFS
  - Stav je často velký — snažíme se nekopírovat *(backtracking)*
  - Včasná eliminace špatných větví *(pruning)*
  - Heuristika pro pořadí proměnných — nejmenší počet možností *(minimum remaining values, MRV)*
  - Heuristika pro pořadí hodnot — nejméně omezující *(least constraining value)*
  - Při selhání se vracíme ke *konfliktním proměnným*.
- Propagace omezujících podmínek *(constraint propagation)*

Algoritmy (2)

- Lokální hledání (local search)
  - Postupná optimalizace.
  - Umí využít předchozí řešení.
  - Minimalizujeme např. počet konfliktů — metoda největšího spádu, simulované žíhání. . . ...
  - Algoritmus *min-conflicts* — náhodný výběr konfliktní proměnné.
- Struktura problému — graf závislostí *(constraint graph)*
  - Nezávislé komponenty — rozklad na podůlohy.
  - Stromová struktura

Problém n-dam — reprezentace

Dámy jsou na pozicích (i, q_i) pro $i \in \{0, 1, \ldots, n-1\}$

- Hledáme $q_0, q_1, \ldots, q_{n-1}$ s hodnotami $\{0, 1, \ldots, n-1\}$
- Všechny q_i musí být různé.
- Všechny $i - q_i$ musí být různé.
- Všechny $i + q_i$ musí být různé.

Postupné přiřazování
(Backtracking)

```
def solve_queens(n):          # n je počet královen
  def solve_internal(i):      # teď hledáme pozici 'i'
    nonlocal queens
    if i>=n:
      return True             # řešení nalezeno
    for j in range(n):
      queens[i]=j
      if safe_position(queens,i):
        if solve_internal(i+1):
          return True
    return  False
  queens=list(range(n))
  if solve_internal(0):
    return queens
  return None                 # řešení nenalezeno
```

Soubor `nqueens.py`.

Ověření konzistence

Není dáma v pozici `(i,queens[i])` ohrožována dámami v pozicích `(j,queens[j])` pro $j < i$?

```
def safe_position(queens,i):
  for j in range(i):
    if (queens[i]==queens[j] or
        i-queens[i]==j-queens[j] or i+queens[i]==j+queens[j]):
      return False
  return True
```

Ověření konzistence

Není dáma v pozici `(i,queens[i])` ohrožována dámami v pozicích `(j,queens[j])` pro $j < i$?

```
def safe_position(queens,i):
  for j in range(i):
    if (queens[i]==queens[j] or
        i-queens[i]==j-queens[j] or i+queens[i]==j+queens[j]):
      return False
  return True
```

```
def print_queens(queens):
  n=len(queens)
  for i in range(n):
    for j in range(n):
      print('Q' if j==queens[i] else '-',end="")
    print()
```

Příklad

```
print_queens(solve_queens(8))
```

```
Q-------
----Q---
-------Q
-----Q--
--Q-----
------Q-
-Q------
---Q----
```

Příklad

```
print_queens(solve_queens(8))
```

```
Q-------
----Q---
-------Q
-----Q--
--Q-----
------Q-
-Q------
---Q----
```

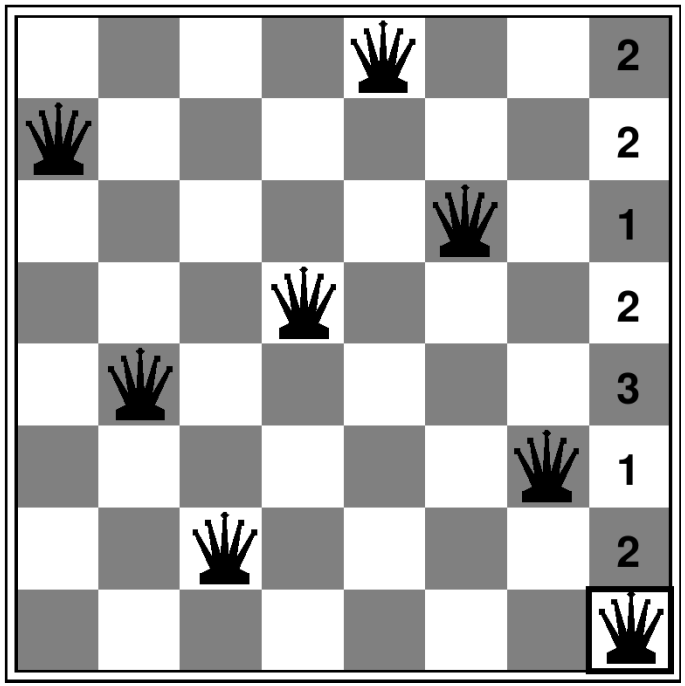
n	čas [s]
10	< 0.01
20	20
22	246

Závislost není monotónní.

Lokální hledání

Min-change

- Náhodně vyber proměnnou (q_i) v konfliktu.
- Vyber p minimalizující počet konfliktů po přiřazení $q_i \leftarrow p$ (vyber náhodně, je-li jich víc). Nepovol $q_i = p$.
- Zapiš $q_i \leftarrow p$ a iteruj, dokud jsou konflikty.



- Náhodně vyber proměnnou (q_i) v konfliktu.
- Vyber p minimalizující počet konfliktů po přiřazení $q_i \leftarrow p$ (vyber náhodně, je-li jich víc). Nepovol $q_i = p$.
- Zapiš $q_i \leftarrow p$ a iteruj, dokud jsou konflikty.

- Velmi úspěšný postup u řady úloh.
- Příklad randomizovaného algoritmu.

Lokální hledání — implementace

```
def solve_queens_minconflict(n,maxiter=100):
queens=permutation(n)  # generate a random permutation
for i in range(maxiter):
  nc=[ conflicts(queens,i) for i in range(n) ] # poč. konfl.
  ic=[ i for i in range(n) if nc[i]>0 ]  # konfl. dámy
  if len(ic)==0: # není konflikt, hotovo
    return queens
  j=ic[random.randrange(len(ic))] # náhodný výběr
  r=position_costs(queens,j)      # ceny pozic
  r[queens[j]]=n                  # chceme změnu
  minr=min(r)
  ps=[ i for i in range(n) if r[i]==minr ] # minima
  p=ps[random.randrange(len(ps))] # náhodný výběr
  queens[j]=p
return None
```

Soubor `nqueens.py`.

Lokální hledání — implementace (2)

Počet konfliktů královny `(i,queens[i])`

```
def conflicts(queens,i):
  c=0
  for j in range(len(queens)):
    if i!=j:
      if (queens[j]==queens[i] or queens[j]-j==queens[i]-i
            or queens[j]+j==queens[i]+i:
        c+=1
  return c
```

Lokální hledání — implementace (3)

Uloží do `r[j]` počet přímých konfliktů po přiřazení `queen[i]=j`

```
def position_costs(queens,i):
  n=len(queens)
  r=[ 0 for i in range(n) ]
  for j in range(n):
    if i!=j:              # for all other queens
      r[queens[j]]+=1 # conflicting column
      for col in [queens[j]-j+i,queens[j]+j-i]:
        if col>=0 and col<n:
          r[col]+=1
  return r
```

Lokální hledání — příklad

```
print_queens(solve_queens_mincoflict(8))
```

```
Solution found at iteration: 24
--Q-----
-------Q
---Q----
------Q-
Q-------
-----Q--
-Q------
----Q---
```

Lokální hledání — rychlost

n	čas [s]
10	< 0.01
20	0.07
100	1.2
200	7.7
500	83.8

Sudoku — příklad

```
display(grid_values(grid1))
```

```
. . 3 |. 2 . |6 . .
9 . . |3 . 5 |. . 1
. . 1 |8 . 6 |4 . .
------+------+------
. . 8 |1 . 2 |9 . .
7 . . |. . . |. . 8
. . 6 |7 . 8 |2 . .
------+------+------
. . 2 |6 . 9 |5 . .
8 . . |2 . 3 |. . 9
. . 5 |. 1 . |3 . .
```

Soubor `sudoku.py`. http://norvig.com/sudoku.html

Sudoku — propagace podmínek

Constraint satisfaction

- Propagace podmínek
  - Když má políčko už jen jednu možnou hodnotu, ostatní políčka v jednotce tuto hodnotu mít nemohou.
  - Pokud jednotka má jen jedno možné místo pro konkrétní hodnotu, určitě tam ta hodnota bude.
  - Uplatníme rekurzivně.
- Prohledávání do hloubky
- Velmi rychlé ($< 0.01\,\mathrm{s}$) pro velkou většinu úloh

Sudoku — příklad (1)

```
display(grid_values(grid1))
```

```
. . 3 |. 2 . |6 . .
9 . . |3 . 5 |. . 1
. . 1 |8 . 6 |4 . .
------+------+------
. . 8 |1 . 2 |9 . .
7 . . |. . . |. . 8
. . 6 |7 . 8 |2 . .
------+------+------
. . 2 |6 . 9 |5 . .
8 . . |2 . 3 |. . 9
. . 5 |. 1 . |3 . .
```

```
display(solve(grid1))
```

```
4 8 3 |9 2 1 |6 5 7
9 6 7 |3 4 5 |8 2 1
2 5 1 |8 7 6 |4 9 3
------+------+------
5 4 8 |1 3 2 |9 7 6
7 2 9 |5 6 4 |1 3 8
1 3 6 |7 9 8 |2 4 5
------+------+------
3 7 2 |6 8 9 |5 1 4
8 1 4 |2 5 3 |7 6 9
6 9 5 |4 1 7 |3 8 2
```

Sudoku — příklad (2)

```
display(grid_values(grid2))
```

```
4 . . |. . . |8 . 5
. 3 . |. . . |. . .
. . . |7 . . |. . .
------+------+------
. 2 . |. . . |. 6 .
. . . |. 8 . |4 . .
. . . |. 1 . |. . .
------+------+------
. . . |6 . 3 |. 7 .
5 . . |2 . . |. . .
1 . 4 |. . . |. . .
```

```
display(solve(grid2))
```

```
4 1 7 |3 6 9 |8 2 5
6 3 2 |1 5 8 |9 4 7
9 5 8 |7 2 4 |3 1 6
------+------+------
8 2 5 |4 3 7 |1 6 9
7 9 1 |5 8 6 |4 3 2
3 4 6 |9 1 2 |7 5 8
------+------+------
2 8 9 |6 4 3 |5 7 1
5 7 3 |2 9 1 |6 8 4
1 6 4 |8 7 5 |2 9 3
```

Náměty na domácí práci

- Řešte problém n dam pomocí vyzkoušení všech možností. Jak velké problémy tento přístup vyřeší během 10 s?
- Řešte problém n dam pomocí prohledávání do hloubky a propagace podmínek.
- Řešte Sudoku pouze pomocí zpětného prohledávání a porovnejte časovou náročnost s řešením P.Norviga.
- Vytvořte náhodný rovinný graf, kde hrany jsou neprotínající se úsečky. Obarvěte uzly tak, aby žádné dva uzly spojené hranou neměly stejnou barvu. Kolik barev na to potřebujete?
- Řešte aritmogramy a úlohy typu ‘zebra’.