Benchmarky

V této kapitole si definujeme benchmarky, které budou dále použity k otestování vlastností jednotlivých řešičů. Jedná se o pět různých problémů, které jsou dobře známy a zdokumentovány. My si na těchto benchmarcích ukážeme jak různý přístup řešičů k modelování problému tak budeme měřit kolik spotřebují strojového času a prostředků na vyřešení těchto problémů. Až na sebereferenční kvíz jsou všechny benchmarky škálovatelné a dají se použít také pro testy robustnosti řešiče. My budeme tuto robustnost testovat na magické sekvenci. Budeme měřit, jak velkou magickou sekvenci je ještě systém schopen spočítat do daného časového limitu. Poslední benchmark – umístění skladů – je příklad optimalizační úlohy. Řešič bude muset najít optimální řešení na základě dané objektivní funkce. Ke každému benchmarku uvedeme jeho stručný popis, model problému s omezujícími podmínkamu a ukázkovou implementaci. Pro ukázkovou implementaci použijeme programovací jazyk Essence. Základy tohoto jazyka popíšeme v první části této kapitoly.

# Essence

Essence je programovací jazyk pro modelování kombinatorických problémů. Umí popsat problém v omezujících podmínkách dostatečně srozumitelně a přehledně, aby i člověk, který tento jazyk nikdy neviděl, byl schopen určit, co daný program dělá. Pro překlad programů v Essence do jiných jazyků existuje volně dostupný překladač Taylor. Tento překladač umí přeložit zadaný program na zdrojový kód pro řešič Gecode případně na vstupní soubor pro řešič Minion. Samotný program v Essence sestává ze tří částí. První část definuje verzi jazyka, druhá část použité proměnné a konečně třetí omezující podmínky na těchto proměnných. Použité proměnné mohou být typu Integer, Boolean a vektor/matice.

# N-královen

Tento benchmark vychází z klasické šachistické úlohy, rozmístit na šachovnici 8 královen tak, aby se navzájem neohrožovaly. My si tuto úlohu zobecníme pro šachovnici o libovolném počtu sloupců. Cílem tedy je do tabulky o rozměrech n×n rozmístit n královen tak, aby se neohrožovaly. To znamená, že pro libovolné dvě královny platí, že nejsou ve stejném sloupci ani řádku a dokonce ani na stejné diagonále.

Pokud tento problém modelujeme, brzy zjistíme, že je výhodné ho modelovat pomocí pole proměnných délky n, kde každá proměnná nabývá hodnot od jedné do n. Dle zadání totiž nesmí být v jednom sloupci dvě královny a zároveň musí být v každém sloupci alespoň jedna královna. Stačí tedy pro každý sloupec určit, v jakém jeho řádku bude umístěná královna. Zároveň musí platit, že všechny hodnoty musí být různé, protože nelze umístit dvě královny na jeden řádek. Nakonec musíme vyřešit podmínku diagonál. Dvě královny jsou na stejné diagonále, pokud je mezi nimi stejný počet sloupců i řádků. V našem modelu to tedy znamená, že nesmí platit vztah: |Q(i) – Q(j)| = |i - j|.

## Model CSP

* Proměnné a domény: q1, ..., qn ∈{1, …, n}, qi odpovídá řádku ve kterém je umístěná královna ve sloupci i
* Podmínky:
  + Žádné dvě královny nejsou na stejném řádku: ∀ i,j ∈{1, …, n}: qi ≠ qj
  + Žádné dvě královny nejsou na stejné diagonále: ∀ i,j ∈{1, …, n}: |qi - qj| ≠ |i - j|
  + Pro omezení symetrie: q1 < qn

## Implementace v Essence

language ESSENCE' 1.b.a

find queens: matrix indexed by [int(1..n)] of int(1..n)

such that

forall i: int(1..n). forall j: int(i+1..n).

alldiff(queens),

| queens[i] - queens[j] | != | i - j |

# Magická sekvence

Magická sekvence je posloupnost čísel, pro kterou platí, že číslo na pozici k (číslujeme od nuly) určuje počet výskytů čísla k v posloupnosti. Například posloupnost (2 1 2 0 0) je magickou sekvencí délky pět, protože platí výše uvedená podmínka – číslo nula je v posloupnosti právě dvakrát a proto je na první pozici dvojka, jednička je právě jedna a proto má posloupnost na pozici jedna právě číslo jedna a tak dále.

## Model CSP

Model pro magickou sekvenci délky k:

* Proměnné a domény: Prvky magické sekvence: m0, ..., mk-1 ∈ {0, …, k}
* Podmínky:
  + Prvek na pozici i odpovídá počtu výskytů i v posloupnosti: ∀i ∈{0,…,k-1}:

V případě, že nelze v řešiči použít podmínku , dá se sestavit alternativní model

* Proměnné a domény:
  + Prvky magické sekvence: m0, ..., mk-1 ∈ {0, …, k}
  + Pomocné proměnné: ∀i,j ∈{0,…,k-1}: auxij ∈{0,1}
* Podmínky:
  + auxij má hodnotu 1 pokud mj má hodnotu i, jinak má auxij hodnotu 0: ∀i,j ∈{0,…,k-1}: (auxij = 1) ⇔ (mj = i).
  + Prvky magické sekvence odpovídají součtu příslušných pomocných proměnných: ∀i ∈{0,…,k-1}: .

## Implementace v Essence

language ESSENCE' 1.b.a

find s : matrix indexed by [int(0..n-1)] of int(0..n)

such that

forall i : int(0..n-1).

( s[i] = (sum j : int(0..n-1). (s[j] = i)))

# Sebereferenční kvíz

Sebereferenční kvíz je kvíz, kde odpovědi na otázky závisí na odpovědích na ostatní otázky v takovémto kvízu. Typickými otázkami v takovémto testu jsou například: „1. První otázkou na kterou je odpověď A je: **A:** 1, **B:** 2, **C:** 3, D: 4, **E:** na žádnou otázku není odpověď A“ a „2. Odpovědí na tuto otázku je: **A:** A, **B:** B, **C:** C, **D:** D, **E:** E“. Při řešení takovýchto kvízů se použijí zejména reifikované podmínky, tedy podmínky ve tvaru (C⇔x)&x∈{0,1}. Konstrukce takovýchto kvízů je popsaná v článku *[citace @article{jios:bubalo, author = {Bubalo, Maja and Čubrilo, Mirko}, title = {Modeling and solving self-referential puzzles}, journal = {JIOS}, volume = {29}, number = {1}, year = {2005}, pages = {268-306}}].* My použijeme kvíz, který byl použitý v článku *[citace @ARTICLE{fernandez00, title = {A Comparative Study of Eight Constraint Programming Languages Over the Boolean and Finite Domains}, author = {Fernández, Antonio~J. and Hill, Patricia~M.}, year = 2002, journal = {Constraints}}]*. Zadání testu je následující:

1. První otázka, na kterou je odpověď A je: (A) 4 (B) 3 (C) 2 (D) 1 (E) none of above
2. Jediné dvě otázky, které následují hned po sobě a které mají shodné odpovědi jsou: (A) 3 a 4 (B) 4 a 5 (C) 5 a 6 (D) 6 a 7 (E) 7 a 8
3. Příští otázka, na kterou je odpověď A je: (A) 4 (B) 5 (C) 6 (D) 7 (E) 8
4. První otázka se sudým číslem, na kterou je odpověď B je: (A) 2 (B) 4 (C) 6 (D) 8 (E) 10
5. Jediná otázka s lichým číslem, na kterou je odpověď C je: (A) 1 (B) 3 (C) 5 (D) 7 (E) 9
6. Otázka s odpovědí D: (A) se vyskytla před touto ale po ní se již nevyskytuje (B) se vyskytuje po této otázce, ale ne před ní (C) se vyskytuje před i po této otázce (D) se v testu nevyskytuje vůbec (E) žádná z uvedených možností
7. Poslední otázka s odpovědí E je: (A) 5 (B) 6 (C) 7 (D) 8 (E) 9
8. Počet otázek, jejichž odpovědí je souhláska je: (A) 7 (B) 6 (C) 5 (D) 4 (E) 3
9. Počet otázek, jejichž odpovědí je samohláska je: (A) 0 (B) 1 (C) 2 (D) 3 (E) 4
10. Odpověď na tuto otázku je: (A) A (B) B (C) C (D) D (E) E

Kvíz můžeme modelovat jako tabulku s pěti sloupci (A,B,C,D,E) a deseti řádky booleovských proměnných, pro každou odpověď v každé otázce jednu. Hodnota v řádku i a sloupci j je 1, pokud je odpověď na otázku i rovna j. Protože pro každou otázku platí, že je na ní pouze jedna přípustná odpověď, přidáme do modelu podmínku „v každém řádku je právě jedna hodnota 1“. Tento test má právě jedno řešení:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 2 | 1 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 | 0 |
| 4 | 0 | 1 | 0 | 0 | 0 |
| 5 | 1 | 0 | 0 | 0 | 0 |
| 6 | 0 | 1 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 1 |
| 8 | 0 | 1 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 1 |
| 10 | 0 | 0 | 0 | 1 | 0 |

## Model CSP

* Proměnné a domény: s1|A, s1|B, ..., s10|D, s10|E ∈ {0, 1}
* Podmínky:
  + V každém řádku je právě jedna hodnota rovna 1: ∀i∈{1,...,10}

## Implementation in Essence

language ESSENCE' 1.b.a

find s : matrix indexed by [int(1..10), int(1..5)] of bool

such that

$ the is only one answer to each question and there is not any unanswered question

forall row : int(1..10). ((sum col : int(1..5). s[row,col]) = 1),

$ Question 1

$ A to D

forall col : int(1..4). ( (s[1,col] = 1) <=> ( (s[(4-col+1),1] = 1) /\ ( forall row : int(1..(4-col)). (s[row,1] = 0) ) ) ),

$ E

(s[1,5] = 1) <=> (forall row : int(1..4). (s[row,1] = 0)),

$ Question 2

forall col : int(1..5). ( (s[2,col] = 1) <=> ( forall col2: int(1..5). (s[3+col-1,col2] = s[3+col,col2]) ) ),

$ Question 3

forall col : int(1..5). ( (s[3,col] = 1) <=> ( (s[(4+col-1),1] = 1) /\ ( forall row : int (4..2+col). s[row,1] = 0 ) ) ),

$ Question 4

forall col : int(1..5). ( (s[4,col] = 1) <=> ( (s[col\*2,2] = 1) /\ ( forall row : int(1..(col-1)). s[row\*2,2] = 0 ) ) ),

$ Question 5

forall col : int(1..5). ( (s[5,col] = 1) <=> (s[2\*col-1,3]=1) ),

$ Question 6

(s[6,1] = 1) <=> ( ( exists row : int(1..5). s[row,4] = 1 ) /\ ( forall row : int (7..10). s[row,4] = 0 ) ),

(s[6,2] = 1) <=> ( ( exists row : int(7..10). s[row,4] = 1 ) /\ ( forall row : int (1..5). s[row,4] = 0 ) ),

(s[6,3] = 1) <=> ( ( exists row : int(7..10). s[row,4] = 1 ) /\ ( exists row : int (1..5). s[row,4] = 1 ) ),

(s[6,4] = 1) <=> ( forall row : int (1..10). s[row,4] = 0 ),

(s[6,5] = 1) <=> (s[6,4] = 1),

$ Question 7

forall col : int(1..5). ( (s[7,col] = 1) <=> ( (s[col+4,5] = 1) /\ ( forall row : int (col+4+1..10). s[row,5] = 0 ) ) ),

$ Question 8

forall col: int(1..5). ( (s[8,col] = 1) <=> ( ( sum row: int(1..10). (s[row,2] + s[row,3] + s[row,4]) ) = (7-col+1) ) ),

$ Question 9

forall col: int(1..5). ( (s[9,col] = 1) <=> ( ( sum row: int(1..10). (s[row,1] + s[row,5]) ) = (col-1) ) )

$ Constraining the question 10 is not necessary

# Quasigroup with holes

Kvazigrupa neboli latinský čtverec je tabulka o velikosti n×n vyplněná čísly v rozsahu 0-n tak, že se žádné číslo v řádku ani ve sloupci neopakuje. Je možné zadat i dodatečnou podmínku na prvky kvazigrupy, například, že prvky na diagonále musí být sudé apod. Úkolem je pak pro danou částečně vyplněnou kvazigrupu najít zbylé prvky tak aby byla plně vyplněna a splňovala všechny podmínky na ní kladené. Takovéto zadání se nazývá problém doplnění kvazigrupy – quasigroup completion problem (QCP). Bohužel toto nemusí vést vždy ke stejným výsledkům. Některá zadání totiž není možné splnit, některá jsou překvapivě jednoduchá apod. Navíc je úloha zaplnění latinského čtverce NP-úplná, nemůžeme si tedy být nikdy jistí, zda je problém pouze moc náročný pro řešič a nebo zda řešič nedává odpověď proto, že řešení neexistuje. Proto se místo klasického QCP používá jeho modifikace – kvazigrupa s dírami (QWH). Nejprve si předem vygenerujeme kvazigrupu, která splňuje danou podmínku. Pak z této kvazigrupy odstraníme některé hodnoty, které pak necháme dopočítat solver. Tím máme zaručeno, že řešení existuje. Generováním vhodných zadání pro QWH se zabýval D. Achlioptas a kol., kteří ukázali, že těžkost problému souvisí s tzv. páteří *[citace @INPROCEEDINGS{Achlioptas00generatingsatisfiable, author = {Dimitris Achlioptas and Carla Gomes and Henry Kautz and Bart Selman}, title = {Generating satisfiable problem instances}, booktitle = {In AAAI/IAAI}, year = {2000}, pages = {256--261}, publisher = {AAAI Press}}]*. Páteř je množina prvků kvazigrupy, které se vyskytují ve všech řešeních. Pokud se podíl páteře blíží k 0 %, znamená to, že exiustuje mnoho různých řešení a tedy řešič může nějaké řešení najít „náhodou“. Pokud je oproti tomu podíl páteře blízký 100 %, existuje velmi málo řešení a vedou k němu všechny podmínky. Dá se tedy očekávat, že zajímavé kvazigrupy budou takové, jejichž páteř má podíl okolo 50 %. Experimenty pak prokázaly *[citace opět Achlioptas00generatingsatisfiable]*, že nejnáročnější jsou problémy s podílem páteře mezi 30 a 35 %.

My použijeme kvazigrupy bez dodatečné podmínky na prvky, které si vygenerujeme pomocí programů lsencode od Carly Gomezové [citace? odkaz?] a walksat od Henryho Kautze.

## Model CSP

Model pro kvazigrupu řádu n. Předvyplněné hodnoty jsou v poli dataij:

* Proměnné a domény: q11, …, qnn ∈ {1, …, n}
* Podmínky:
  + Prvky v jednom řádku jsou různé: ∀ i ∈ {1, …, n}: ∀ j,k ∈ {1, …, n}: qij ≠ qik
  + Prvky v jednom sloupci jsou různé: ∀ i ∈ {1, …, n}: ∀ j,k ∈ {1, …, n}: qji ≠ qki
  + Některé prvky mají hodnotu danou zadáním: dataij definováno ⇒ (qij = dataij)

## Implementation in Essence

language ESSENCE' 1.b.a

letting nDomain be domain int(1..n)

find qcp : matrix indexed by [nDomain,nDomain] of nDomain

such that

forall i : nDomain. alldiff(qcp[i, nDomain]),

forall i : nDomain. alldiff(qcp[nDomain,i])

# Výstavba skladů

Představme si, že máme hypotetické obchodní společnosti pomoci s rozhodnutím, jaké postavit sklady pro své prodejny. Hlavním kritériem je samozřejmě cena, jakou bude tato výstavba stát. Naším cílem je tedy tuto cenu minimalizovat. Na druhou stranu musíme vyhovět všem omezujícím podmínkám, na problém kladených. Společnost má již postavené prodejny a vytipované lokality pro své sklady. Každý možný sklad má definovanou svou maximální kapacitu – nejvyšší možný počet obsluhovaných obchodů. Každý obchod musí mít přidělený právě jeden sklad. Cena za výstavbu skladu je fixní ale náklady na zásobení prodejny z daného skladu je pro každou takovou dvojici různá. Řešič na začátku dostane jako zadání počet prodejen, počet možných skladů s jejich kapacitami a matici c, kde prvek cij určuje cenu za zásobování prodejny i ze skladu j.

## Model CSP

* Proměnné a domény:
* Podmínky:
* Objektivní funkce: