Moderní metody řešení problému pokrytí

Kamil Kopejtko

Prohlášení

Prohlašuji,že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu,projekty,SW atd.) uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

Mé poděkování patří ing. Petru Fišerovi za odborné vedení, trpělivou pomoc a veškerý čas, který mi věnoval při zpracovávání této bakalářské práce.

V Chomutově dne 10.1.2007 Kamil Kopejtko

Anotace
Cílem této bakalářské práce je naprogramovat nové metody zabývající se řešením minimálního pokrytí. Nejprve jsem v C++ naimplementoval AURU (exaktní metoda pro řešení problému pokrytí). Tuto jsem otestoval a porovnal s metodou limitní spodní hranice.
Dále jsem v C++ naimplementoval pokročilé heuristické metody, otestoval a porovnal s jinými dostupnými heuristikami.
Testy probíhali na náhodně generovaných maticích.

Abstract
The aim of this work is to program methods solving the covering problem. First I programmed in C++ method called AURA (exact method for solving the covering problem). This method I have tested and compared it with Limit Lower Bound method.
Next I have programmed in C++ advanced heurictic methods, tested them and compare them with other available methods.
The tests ran on the random generated matrixes.

Obsah

1. Úvod

Problém pokrytí má použití v mnoha oblastech počítačového výzkumu, syntéze logických obvodů, dvouúrovňové minimalizaci logických obvodů, analýze spolehlivosti a přesného kódování.

Tato práce má za cíl představit novou metodu řešení problému pokrytí, založenou na "negativním myšlení", a sice AURU.[2] Ta je zde porovnávána s dosud nejlepším napsaným algoritmem "pozitivního myšlení" - Metodou větví a hranic s limitní spodní hranicí[1], a dosahuje pozoruhodných výsledků.

Dále jsou zde představeny pokročilé heuristické metody pro řešení problému pokrytí [3] a opět porovnávány s heuristikami již dříve napsanými. [1].

Experimentální výsledky testování jsou představeny v Kapitole 7.

Celá práce je stavěna na dříve napsané bakalářské práci : Moderní metody řešení problému pokrytí od Lukáše Krejčíka [1]. Z níž je kompletně převzata 3. kapitola, podkapitoly 7.1 a 7.3 a některé popisy algoritmu větví a hranic a jeho vylepšování v podkapitole 4.1. (pro lepší pochopení AURY).

2. Problém pokrytí

Definice 2.1: *Nechť* $X=\{x_1,x_2,\ldots,x_N\}$ *je množina řádků a* $Y=\{y_1,y_2,\ldots,y_M\}$ *je množina sloupců v matici A o rozměrech MxN ,* <u>cena</u> *je funkce definovaná na* $Y \mid cena(y) \rightarrow R$, *která každému* $y \in Y$ *přiřadí kladné reálné číslo. Prvek* $y_j \in Y$ *pokrývá prvek* $x_i \in X$ *pokud* $A[i,j]=1$,*jinak* $A[i.j]=0$. *Problém pokrytí <X,Y,cena> spočívá v nalezení podmnožiny S množiny Y s nejmenší cenou, takové, že pro každý prvek x množiny X existuje prvek y množiny Y tak, že x R y.*

Nechť je dána matice *A* , všechny položky jsou 1 nebo 0. Řešení problému pokrytí spočívá v nalezení minimální podmnožiny sloupců matice *A* tak, že každý řádek matice *A* je pokrytý alespoň jedním sloupcem podmnožiny. Řádek *i* je pokrytý sloupcem *j* pokud *Aij = 1*. Sloupce jsou ohodnoceny kladným reálným číslem, zvaném cena sloupce. Hledání řešení spočívá ve výběru sloupců do řešení, dokud nejsou pokryty všechny řádky. Cena řešení je tvořena součtem cen sloupců vybraných do řešení. Pokrytí matice může být nalezeno buď přesně, s použitím rekurzivního algoritmu, nebo použitím heuristiky. Přesné řešení problému pokrytí je ve většině případů velmi časově náročné, v některých případech je použití heuristiky jedinou možností.

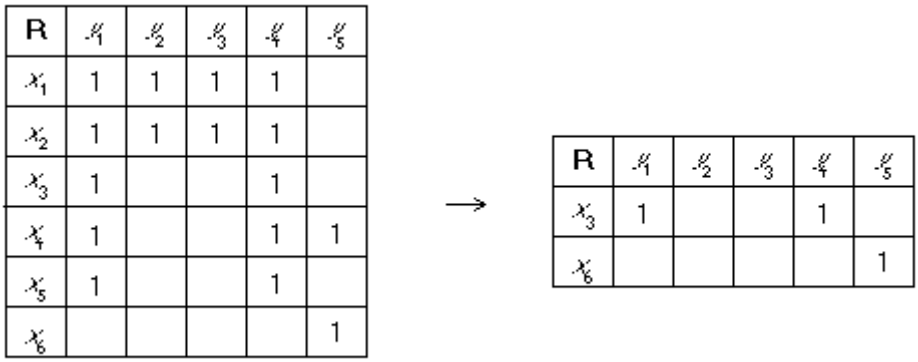
3. Dominance

Při řešení problému pokrytí můžeme zjistit, že řádek x_1 obsahuje jedničky všude, kde je obsahuje řádek x_2 a ještě na dalších místech. Při pokrývání to znamená, že pokud vybereme do řešení jakýkoliv sloupec pokrývající x_2, určitě pokryjeme i řádek x_1. Řádek x_1 může tedy být z matice odstraněn. Říkáme, že *x2* dominuje x_1. V následujícím odstavci si popíšeme, jak se odstraňují tyto dominance.

3.1 **Dominance řádků**

Definice 2.1.1 *Nechť <X,Y,cena> je problém pokrytí. Řádek x' množiny X všech řádků dominuje x právě tehdy, když všechny prvky množiny sloupců Y které pokrývají x' pokrývají také x .*

Jinými slovy,pokud nějaký řádek *x'* obsahuje ve sloupci jedničku stejně tak jako řádek *x* (*x* může obsahovat jedničky i na jiných místech), tak *x'* dominuje *x* , vždycky,když je pokryto *x'*, bude pokryto i *x*. Řádek *x* tedy může být vyškrtnut z matice. Funkce *R_Dominance (struct CoveringMatrix *)* vždy vezme jeden řádek *x'* z matice a porovná ho s ostatními. Pokud nalezne řádek x obsahující jedničky všude, kde je obsahuje *x'*, vyškrtne přepojením spojového seznamu řádků tento řádek z matice. Jelikož se porovnává každý řádek s každým, operační složitost algoritmu v nejhorším případě je $O(n^2)$, kde *n* je počet řádků matice. Na obrázku 2.1.1 je znázorněna redukce matice před a po odstranění dominance řádků.

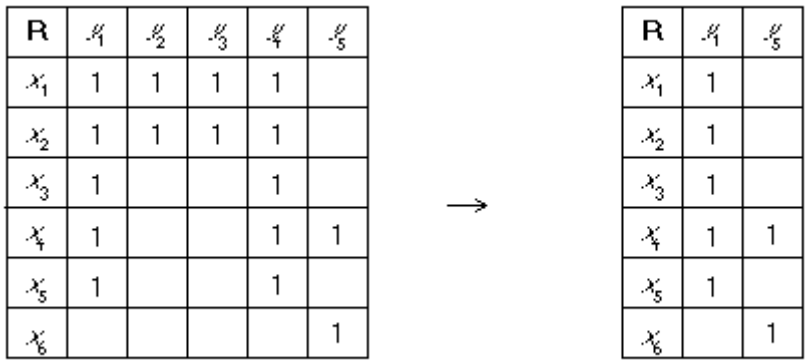


Obrázek 3.1 – Odstranění dominancí na množině řádků

3.2 **Dominance sloupců**

Definice 2.1.2 *Nechť <X,Y,cena> je problém pokrytí. Sloupec y' množiny všech sloupců Y dominuje sloupci y právě tehdy, když všechny prvky množiny řádků X pokryté y jsou také pokryté y'* .

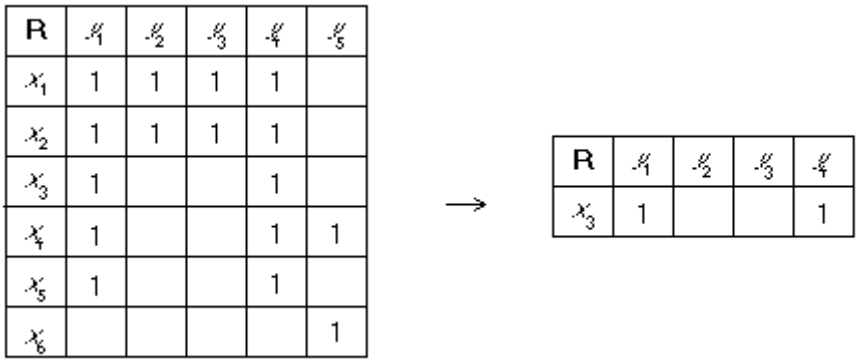
To znamená, že pokud nějaký sloupec *y'* obsahuje jedničky všude, kde je obsahuje sloupec *y* (*y'* může obsahovat jedničky i na více místech), tak *y'* dominuje *y*. Funkce *S_Dominance (CoveringMatrix *)* vezme sloupec *y'* z matice a porovná ho s ostatními sloupci. Pokud sloupec *y* obsahuje nuly všude,kde je obsahuje *y'*, a je splněna podmínka *cena*(*y'*) < *cena*(*y*), *y* bude vyjmut z matice. Pokud podmínka splněna není, mohli bychom ztratit minimální řešení. Opět se porovnává každý s každým, takže operační složitost v nejhorším případě, kdy se nevyškrtne žádný sloupec, je $O(n^2)$, kde *n* je počet sloupců matice. Na obrázku 2.1.2 je znázorněna redukce matice před a po odstranění dominance sloupců.



Obrázek 3.2 – Odstranění dominancí na množině sloupců

3.3 Nezbytné sloupce

Sloupec *y* množiny sloupců *Y* je nezbytný, pokud je jediný, který pokrývá řádek *x* množiny řádků *X*. Pokud *Y* je množina přímých implikantů, znamená to nesporný přímý implikant. Tyto sloupce vždycky musí patřit do řešení, můžeme je proto z matice vyřadit, a minimální řešení původního problému získáme přidáním těchto sloupců do minimálního řešení redukovaného problému. Při řešení dominancí, pokud narazíme na řádek obsahující pouze jednu jedničku,přidáme sloupec který ji obsahuje do řešení. Na obrázku 2.1.3 je znázorněna redukce matice před a po odstranění dominance řádků, s výběrem nezbytných sloupců. Sloupec y_5 byl vybrán do řešení, protože řádek x_6 byl pokrytý pouze tímto sloupcem.



Obrázek 3.3 – Odstranění dominancí na množině řádků s výběrem nezbytných sloupců

Při řešení dominancí sloupců můžou však v matici vzniknout nové dominance řádků a naopak. Musíme tedy zajistit opakované volání funkcí řešící dominance, dokud se v matici nějaké budou vyskytovat. Při nalezení nezbytných sloupců musíme tyto zahrnout také do řešení. Funkce *SolveDominance (CoveringMatrix *, Reseni *)* volá funkce popsané výše tak dlouho,dokud je matice měněna řešením dominancí. Druhý parametr má strukturu ukazující na spojový seznam množiny nezbytných sloupců, které byly funkcí vybrány do řešení.

4. Řešení problému pokrytí

Poté co matici zredukujeme pomocí procedur popsaných výše (dominance), zbude matice, kterou tyto procesy už více nezmění. Tato matice je nazývána *cyklické jádro <X,Y,cena>*. Pokud je prázdné, množina všech nezbytných prvků byla nalezena během procesů popsaných výše.

Pokud není prázdné, minimalizace může být zakončena pomocí rekurzivního algoritmu. Princip algoritmu spočívá ve výběru prvku *y* množiny *Y* a generování dvou podproblémů, jeden spočívá ve výběru prvku *y* do minimálního řešení, druhý ve vyškrtnutí *y* z minimálního řešení. Tento algoritmus se rekurzivně aplikuje na cyklická jádra vzniklých podproblémů, minimální řešení problému je minimum z obou řešení podproblémů. Jak již bylo řečeno, tento algoritmus je velice časově náročný, s operační složitostí O (2^n), kde *n* je počet sloupců matice.

V algoritmu můžeme provést několik úprav, popsané v podkapitole 4.1, které nám urychlí výpočet. Při nalezení minimálního pokrytí *Nejlepší* můžeme ořezat při prohledávání všechny větve, kde cena cesty do uzlu je vetší nebo rovna ceně již nalezeného pokrytí *Nejlepší*. Tím se urychlí výpočet, protože se nebude procházet prostor, kde se řešení určitě nenachází. Cena minimálního nalezeného řešení se nazývá horní hranice řešeni (*upper bound*).

4.1 Metoda větví a hranic a její vylepšení

Tato úprava rekurzivního algoritmu spočívá ve výše uvedeném ořezávání vyhledávacího prostoru o místa, kde cena částečného pokrytí převyšuje cenu nejlepšího nalezeného pokrytí.

Vyhledávací prostor je reprezentován binárním vyhledávacím stromem, kde kořen představuje vstupní problém, hrany znamenají volání rekurze, každá je ohodnocena 0 nebo 1, podle toho, zda sloupec uvažovaný v rodičovském uzlu je do řešení přidán či ne.Vnitřní uzel reprezentuje částečné řešení problému s *cenou* rovnou součtu cen sloupců na cestě z kořene do tohoto uzlu, listu je dosaženo pokud je nalezeno pokrytí.

Algoritmus se rekurzivně aplikuje na cyklická jádra vzniklých podproblémů, minimální řešení problému je minimum z obou řešení podproblémů. Procedura *Branch_And_Bound (CoveringMatrix *)* , jejíž parametrem je ukazatel na strukturu, vybere tento sloupec y_1 do řešení a vyškrtne tento sloupec společně s řádky, které pokrývá, z matice. Jelikož odebráním sloupce z matice mohou vzniknout dominantní řádky, opět se rekurzivně zavolají funkce řešící dominance a pokračuje se ve větvení. Pokud je matice prázdná, je nalezeno nové řešení, cena tohoto řešení je rovna součtu cen všech sloupců tvořících řešení. Toto řešení je označeno *Best* s cenou *cena(Best)*, jako zatím nejlepší nalezené řešení.

Možným vylepšením tohoto algoritmu je další omezení vyhledávacího prostoru, kde se řešení nenachází spodní hranicí.
Spodní hranice je vyjádřena součtem cen sloupců, jež pokrývají největší možnou množinu nezávislých řádků - MSIR- maximum set of independent rows (žádné dva řádky z množiny nemohou být pokryty jedním sloupcem a pro pokrytí každého řádku vybíráme sloupec s nejnižší cenou).

Známe-li spodní hranici minimálního řešení podproblému získanou v nějakém místě algoritmu, můžeme ukončit rekurzi jakmile je cena cesty do uzlu *cena(u)* a spodní hranice *lBound* větší nebo rovna ceně nejlepšího dosud nalezeného řešení *cena(Best)*. Tj. jakmile je splněna podmínka: *cena(u) + lBound >= cena(Best)*
Více o hledání spodní hranice v kapitole 5.

Dalším možným vylepšením je zavedení limitní spodní hranice. Principem je odebrání sloupců , jež nepokrývají žádný řádek z MSIR a přepočítání spodní hranice vždy, když rozdíl mezi (*cena(u) + lBound*) a *cena(Best)* dosáhne hodnoty *1*.

4.2 Zahrnutí "negativního" myšlení

Při procházení vyhledávacího prostoru často nalezneme dobré řešení poměrně rychle a to je poté pouze několikrát upraveno než je dosaženo nejlepší možné řešení. Tudíž většina vyhledávacího prostoru je procházena zbytečně.
Proto je pro algoritmus přirozené být "skeptickým" ohledně šance na zlepšení dosud nalezeného řešení.
To naznačuje, že při použití "větší" spodní hranice by došlo k dramatickému zrychlení celého algoritmu.

5. AURA

5.1 Úprava algoritmu větví a hranic

Naším cílem bude, pokusit se "vylepšit" spodní hranici ve chvíli, kdy bude "velmi blízko" hranici horní. Úspěšný pokus o zlepšení spodní hranice povede k zastavení prohledávání daného prostoru a celkovému urychlení. Neúspěšný pokus by měl vrátit nové nejlepší řešení (*uBound*) .

Stanovme si konstantu , kterou nazvěme *maxRaiser*. Pokud bude při běhu algoritmu spodní hranice vzdálena od horní o tuto konstantu, zavoláme proceduru *Raiser*, která se pokusí dostatečně "vylepšit" spodní hranici, aby bylo možné ukončit rekurzi, nebo vrátí nejlepší možné řešení (jež se nachází v onom vyhledávacím prostoru a je lepší než to, dosud nalezené).

Takový algoritmus by mohl vypadat následovně:

B&B (*CoveringMatrix *CM , Řešení * Sol*)

Vstup: Pokrývací matice
Výstup: Množina sloupců tvořící řešení, nebo ∅ pokud řešení nebylo nalezeno

```
1. Vyřeš_Dominance (CM , Sol);
2. if (CM) == ∅ {
                if  cena ( Sol ) < cena ( Best ) return newBest;
                        else return ∅
        }
        else {
                Vyber do řešení sloupec j;
                Najdi MSIR pro vzniklý podproblém
                Najdi rozdíl mezi spodní hranicí a horní hranicí - difference
                difference = uBound - lBound
                if difference <= 0, Solution Sol = ∅
                if difference <= maxRaiser ....zavolej Raiser
                else if cena ( cesta (j) + cena(j) + |MSIR| < cena ( Best )
                        Branch_and_Bound ( CM, Sol1 );

                Vškrtni sloupec j z řešení;
                Najdi MSIR pro vzniklý podproblém
                Najdi rozdíl mezi spodní hranicí a horní hranicí - difference
                difference = uBound - lBound
                if difference <= 0, Solution Sol = ∅
                if difference <= maxRaiser ....zavolej Raise
                else if cena ( cesta (j) + |MSIR| < cena ( Best )
                        Branch_and_Bound ( CM, Sol2 );
                return min (Sol1,Sol2);
        }
```

Diagram 5.1 Algoritmus Branch and Bound rozšířený o MSIR a volání Raiseru

5.2 Raiser algoritmus

K představení Raiseru budeme potřebovat následující názvosloví:

- *min(UCP(A))* je velikost minimálního řešení matice *A,* potažmo *UCP(A)* (United covering problem)
- nechť A´ je podmnožina matice A, přičemž *Sloupce(A´)=Sloupce(A)* a *Řádky(A´)* jsou podmnožinou *Řádků(A)*
- *MSIR* je maximální množina nezávislých řádků
- A´ + Ar označuje matici A´ k níž byl přidán řádek Ar ležící v *Řádky (A)-Řádky(A´)*.
- nechť *S* je řešení *UCP(A)*. A sloupec *j* ležící v řešení je nadbytečný, jestliže *S\{j}* je rovněž řešením. Řešení *UCP(A)*, které neobsahuje žádné nadbytečné sloupce je základní řešení.
- *Sol(A´,n)* označuje soubor základních řešení *UCP(A´)* skládající se z *n* nebo méně sloupců. *Sol (A´,m)*, kde *m=min(UCP(A´))*, je soubor všech minimálních řešení *UCP(A´)*.
- *O(Ai)* bude soubor sloupců pokrývajících řádek Ai neboli doména řádku
- řešení *UCP(A)* je představováno *krychlí C*. *Krychli C* definujeme jako soubor domén všech řádků matice *A´*.
- Cenu všech souborů sloupců v *krychli C* budeme označovat jako počet domén krychle C a budeme ji označovat *cost(C)*.
- řešení *S´*, které bude podmnožinou množiny *Sloupců(A)* je částečné řešení *UCP(A)* jestliže to není řešení *UCP(A)*.
- soubor *P* částečných řešení je úplný, jetliže pro každé řešení *S UCP(A)* existuje částečné řešení *S´* v *P* , přičemž *S´* je podmnožinou *S*.

Nyní si můžeme popsat podstatu Raiseru.

Předpokládejme, že pro matici *A´* matice *A* známe soubor řešení *Sol(A´,n)*. Spodní hranice získaná z *A´* je rovna *m=min(UCP(A´))*. Nyní přidáme řadek *Ai* z matice *A* do matice *A´*. Zřejmě *Sol(A´+ Ai, m)* je podmnožinou *Sol(A´,m)*, neboť obecně některá řešení z *Sol(A´,m)* nemusejí pokrývat řádek *Ai* a tak nejsou obsaženy v *Sol(A´+ Ai, m)*. Tedy po přidání řádků *Ai1,.....,Aik* matice A do matice A´, pokud se *Sol(A´+ Ai1 + ... + Aik,m) = 0*, pak jsme zlepšili spodní hranici pro *UCP(A)* o 1. Jestliže *Sol(A´+ Ai1 + ... + Aik,n)=0, n>=m,* pak jsme zlepšili spodní hranici o *n-m+1*.

Začneme od matice A´, která je souborem maximální množiny nezávislých řádků (MSIR) a poté přidáváme do A´ řádky s cílem zlepšit spodní hranici získanou právě díky MSIR. Je to proto, neboť když známe *Sol(A´,n)* není obtížné spočítat *Sol(A´+ Ai,n)*.

V každém uzlu N vyhledávacího stromu můžeme spočítat MSIR pro odpovídající matici *An*. Jestliže *MSIR+cesta(An)+n >= best*, kde *best* je nejlepší dosud nalezené řešení, zavoláme Raiser na matici *An*, jinak pokračujeme ve větvení.

Výsledkem volání Raiseru může být následující:
1) spodní hranice MSIR může být navýšena o *best-MSIR-cesta(An)* a rekurze v uzlu může být zastavena.
2) minimální řešení *S(An)* problému *UCP(An)* je nalezeno, takže *S(An) + cesta(An)*je nové nejlepší řešení *UCP(A)*.

Za povšimnutí stojí, že zlepšení spodní hranice byť jen o malý kousek povede k významné redukci výpočetního času. Jako příklad můžeme uvést metodu limitní spodní hranice, která umožňuje oříznout některé větve vyhledávacího prostoru ve snaze zkrátit

výpočetní dobu. Je zřejmé, že technika limitní spodní hranice neořízne více větví vyhledávacího prostoru než Raiser s *n* rovno 1.

Raiser (struct Cube *SolCube *[krychle obsahující sloupce pokrývající řádky MSIR]*, struct CoveringMatrix *CM *[matice jejíž řešení se hledá]*, unsigned long lowerBound *[spodní hranice řešení ~ MSIR]*, struct HeaderSol *SolUCP *[struktura do níž se uloží případné nalezené minimální řešení]* , long dosavadni *[cena dosud nalezeného částečného řešení]*)

Vstup: Pokrývací matice**,** krychle obsahující množinu sloupců pokrývajících MSIR**,** dolní hranice řešení ~ MSIR, cena částečného řešení
Výstup: Množina sloupců tvořící řešení, nebo ∅ pokud řešení nebylo nalezeno

if (spodní hranice byla navýšena natolik, že dosáhla horní hranice) return 1;
if (v matici již není žádný řádek) najdi minimální řešení , return 0;

vyber řádek z matice a najdi množinu sloupců , která ho pokrývá *SelectedRowDomain*;
odstraň řádek z matice;

while (krychle není prázdná) {
vytvoř kopii *cube -> cube`* ;
vyber první množinu sloupců krychle *cube`*1;

najdi průsečík množiny sloupců krychle a množiny sloupců vybraného řádku *inter*;
najdi množinu sloupců krychle, které nepokrývají vybraný řádek (~nejsou v množině sloupců řádku) *rest*;

if (existuje průnik *cube`*1 a *SelectedRowDomain ~ inter* obsahuje alespon 1 sloupec) {

if (*cube`1* obsahuje více sloupců než *inter*) {
*cube`*1 = *inter*;
Raiser (c*ube`, CM, lowerBound, SolUCP ,dosavadni*);

vyjmi ze souboru sloupců pokrývajících řádek sloupce obsažené v *inter ~ SelectedRowDomain = SelectedRowDomain - inter*;

cube`1 = rest;

}else ~ *cube`1== inter* ~ všechny sloupce obsažené v *cube`1* pokrývají vybraný řádek ~ tento řádek bude tedy pokryt vždy {

Raiser (c*ube`, CM, lowerBound, SolUCP ,dosavadni*);
break; // není třeba pokračovat, řádek bude pokryt vždy
}
}

vyber další množinu sloupců krychle *cube*;

```
            if ( v krychli již není žádná další množina sloupců ) {

                if ( SelectedRowDomain obsahuje ještě nějaké sloupce ) {

                    přidej ke krychli cube soubor sloupců obsažených v
                    SelectedRowDomain;
                    Raiser (cube, CM, lowerBound, SolUCP ,dosavadni);
                    break;
                }
            }

        }
```

5.3 Raiser v příkladu

Uvažujme 1-Raiser (hodnota n = 1) aplikovaný na následující matici A:

	1	2	3	4	5	6	7
1	0	0	0	1	1	0	1
2	1	0	1	0	0	0	0
3	0	1	1	1	0	1	0
4	0	0	0	0	0	1	1
5	1	1	0	0	0	0	0
6	0	0	1	0	1	0	0

Předpokládejme, že soubor řádků matice A´= {A1, A5, A6} byl vybrán jako MSIR(A). Soubor základních řešení UCP(A´) je:

C = {1,2} x {3,5} x {6,7}

pro které

$D = D1 \cup D2 \cup D3$
= {1,2} ∪ {3,5} ∪ {6,7}
= {1,2,3,5,6,7}.

C nám dává spodní hranici 3 (C má 3 domény). Kořenový uzel vyhledávacího stromu je určen krychlí C a maticemi A', A'' ,kde $Řádky\ (A'') = Řádky(A) \backslash Řádky(A')$.Úkolem 1-Raiseru je zlepšit spodní hranici ze tří na čtyři.
Vyberme řádek $A3$ z A'' a přidejme ho do A'. $O(A3)$ = {2,3,4,6} a tedy řádek $A3$ protíná všechny tři domény A'. Proto soubor všech možných základních řešení tvořených ne více jak čtyřmi sloupci matice $A' + A3$ je následující:

$C1$ = {2} x {3,5} x {6,7}
$C2$ = {1} x {3} x {6,7}
$C3$ = {1} x {5} x {6}
$C4$ = {1} x {5} x {7}

Část 1(C) = $C1 \cup C2 \cup C3$
Část 2(C) = $C4$

Sol(A´+ A3,C) = Část 1*(C)* $\cup$ *C4 x (O(A3)|D)*
tedy *Sol(A´+ A3,C) = ({2} x {3,5} x {6,7}* $\cup$ {1} x {3} x {6,7} $\cup$ {1} x {5} x {6})
$\cup$ {1} x {5} x {7} x {4}.

Krychle *C1* popisuje soubor řešení krychle *C* pokrývající *A´+ A3* v které je řádek *A3* pokryt sloupcem první domény krychle *C* (a možná i sloupci jiných domén). A tak *C1 = {1,2}* $\cap$ *O(A3) x {3,5} x {6,7}.*

Krychle C2 popisuje soubor řešení neobsažených v C1 ve kterém je řádek A3 pokryt sloupci druhé domény a tak *C2 = {1,2}\O(A3) x {3,5}* $\cap$ *O(A3) x {6,7} = {1} x {3} x {6,7}.*
Krychle C3 popisuje soubor řešení neobsažených v C1 a C2, ve kterém je řádek A3 pokryt sloupci třetí domény .

Konečně krychle *C4* popisuje soubor řešení *UCP(A´)* krychle *C*, které nepokrývají řádek *A3* a tedy žádné řešení *UCP (A´+ A3)*.

Tedy kořenový uzel má 4 potomky, každý určen jednou ze čtyř krychlí *C1, C2, C3, C4 x (O(A3) \ D)* a dvěma maticemi *A´+ A3, A´´ - A3*. Uvažujme nyní větev odpovídající C1 = {2} x {3,5} x {6,7}. Předpokládejme, že řádek *A2* je vybrán z *A´´ - A3* a je přidán do *A´ + A3*. *O(A2)* = {1,3} protíná pouze druhou doménu krychle *C1*, tedy krychle *C1* se rozpadne na :
C11 = část1(*C1*) = {2} x {3} x {6,7}, *C12* = část 2(*C1*) = {2} x {5} x {6,7}.

Tedy uzel odpovídající C1 má dvě větve , jejichž matice jsou *A´+ A3+ A2* a *A´´ - A3 - A2* a jejichž krychle jsou *C11* a *C12 x (O(A2) * D_{C1}*) =* {2} x {5} x {6,7} x {1}. Uvažujme větec odpovídající krychli *C11*. Pouze řádek A1 zbývá v *A´´ - A3 - A2 . O(A1) = {4,5,7}* , což protíná třetí doménu krychle *C11*. Krychle *C11* se tedy rozpadne na *C111 = část 1(C11) = {2} x {3} x {7}, C112 = část 2(C11) = {2} x {3} x {6}.*

Tedy uzel odpovídající C11 má dvě větve jejichž matice jsou *A´+ A3+ A2+ A1* a *A´´ - A3 - A2- A1* a jejichž krychle jsou *C111* a *C112 x (O(A1) * D_{C11}*) = {2} x {3} x {6} x {4,5}.*

Větev odpovídající uzlu *C111* vede k uzlu ve kterém první matice je rovna *A* a druhá matice je prázdná. Navíc krychle *C111* má tři domény. To znamená že krychle *C111* obsahuje řešení *A* sestávající ze třech sloupců (v tomto případě jediné řešení). Spodní hranice nemůže být zvýšena na čtyři a řešení *{2} x {3} x {7}* je vráceno jako právě nalezené nejlepší řešení.

6. Pokročilé heuristické metody

Řešení problému pokrytí exaktními metodami trvá často velmi dlouhou dobu. Zvláště pro velké matice, kdy je řešení pomocí heuristik často jediné možné řešení.
Na takové heuristiky jsou kladeny dva základní požadavky:
a) Výpočet heuristiky musí být tak rychlý, jak jen to bude možné
b) Výsledné řešení musí být tak dobré, jak jen to bude možné

Tyto heuristiky byli porovnávány s dříve vytvořenými [1]. Uvádím zde stručný popis oněch dvou předchozích:

NaturalHeuristic

while (CM == ∅) {
1. Z jisti počet jedniček ve všech sloupcích
2. Vyber sloupec s nejvíce jedničkami, přidej ho do řešení a vyjmi společně s řádky, které pokrývá, z matice

}

ContribAddHeuristic

while (CM == ∅) {
pro všechna $y_j \in Y$ Spočítej $SP(y_j)$
Vyber sloupec s největším SP, přidej ho do řešení a vyjmi společne s řádky, které pokrývá, z matice
}

přičemž: $SP(x_i) = \frac{1}{\sum_{j=1}^{M} A[i,j]}$ a $SP(y_j) = \sum_{i=1}^{N} A[i,j] \cdot SP(x_i)$.

6.1.1 Podstata pokročilých heuristik

názvosloví:
J - soubor sloupců, I - soubor řádků,

Podstatou těchto heuristik jsou dvě funkce:

Nezbytnost(Essenciality) - funkce e(k), $k \in J$ je heuristická funkce definovaná následovně:

Jestliže existuje alespoň jeden řádek $v \in I$ takový, že $a_{vk} = 1$ a $\sum_{j \in J} a_{vj} = 1$, pak funkce e(k) nabírá hodnoty TRUE, v opačném případě FALSE. (sloupec je/není esenciální).

Hodnota (score) - Hodnota p(k), $k \in J$ je heuristická funkce definovaná následovně:

Pro heuristiku dle Servita:

$$p(k) = \sum_{i \in I} \frac{a_{ik}}{c_k} \left(\sum_{j \in J} \frac{a_{ij}}{c_j} \right)^{-1}$$

Pro heuristiku dle Bowmana:

$$p(k) = \sum_{i \in I} a_{ik} (\sum_{j \in J} a_{ij})^{-1}$$

Pro heuristiku dle Neculy:

$$p(k) = \sum_{i \in I} \frac{a_{ik}}{c_k}$$

Pro heuristiku dle Michalského:

$$p(k) = \sum_{i \in I} a_{ik}$$

Pro heuristiku dle Rotha:

p(k) = $\frac{1}{c_k}$ jestliže $\sum_{i \in I} a_{ik} \neq 0$, p(k) = 0 jestliže $\sum_{i \in I} a_{ik} = 0$

6.1.2 Jednoduchý popis pokročilých heuristik

Tyto heuristiky pracují na principu nalezení esenciálních (nezbytných) sloupců a jejich přidání do řešení. Neexistuje-li již v matici žádný nezbytný sloupec (tj. v matici neexistuje řádek, který by byl pokryt právě jedním sloupcem (tento sloupec nazýváme esenciální)), nalezneme pomocí heuristické funkce (viz jejich popis výše) nejhorší sloupec (sloupec s nejnižší spočtenou hodnotou p(k)) a ten odebereme z matice. Vzhledem k tomu, že tento sloupec zcela jistě nebyl nezbytný, bude každý řádek jež tento sloupec pokrýval pokryt minimálně jedním dalším sloupcem.
Nyní můžeme opět zavolat fukci pro hledání nezbytných sloupců. Tento postup budeme opakovat do doby, dokud v matici budou zbývat nějaké řádky k pokrytí..

6.2 konstrukce algoritmu

Algoritmus:

Krok 1: (iniciace)
I = {1,2 ...m}
J = {1,2...n}

Krok 2: (výběr)
Pro každý sloupec k $\in$ J takový, že e(k) = TRUE nastavíme:
$x_k = 1$
J = J - k
I = I - i pro každé i $\in$ I takové že $a_{ik} = 1$.

Krok 3: (test)
Jestliže I = $\varnothing$ pak X je pokrytí - konec.

Jestliže $I \neq \emptyset$ a $J = \emptyset$ pak žádné pokrytí neexistuje - konec, jinak pokračuj

Krok 4:(vyloučení)
Najdi takový sloupec $k \in J$, pro který platí $p(k) <= p(j)$ pro každé $j \in J$ a nastav:
$J = J - k$
Opakuj Krok 2.

6.3 Další možná úprava těchto heuristik

Velmi jednoduchou úpravou algoritmu můžeme docílit významného snížení výpočetního času. A sice místo použití cenové funkce k nalezení nejhoršího sloupce, který bychom poté poté z matice odebrali, použijeme tuto funkci k nalezení nejlepšího sloupce (sloupec s nejvyšší spočtenou hodnotou p(k)). Tento sloupec přidáme do řešení, což nám umožňuje odebrat z mtaice všechny řádky, které tento sloupec pokrýval.
Takto upravené heuristiky budeme nazývat heuristikami pozitivními.

6.4 Shrnutí heuristik

Popsané heuristické metody pro řešení problému pokrytí zajišťují nalezení řešení v krátkém čase při relativně dobré ceně. Při použití pozitivního přístupu (přidávání nejlepších sloupců rovnou do řešení) lze razantně zredukovat výpočetní čas. Více viz kapitole 7.3 Shrnutí výsledků.

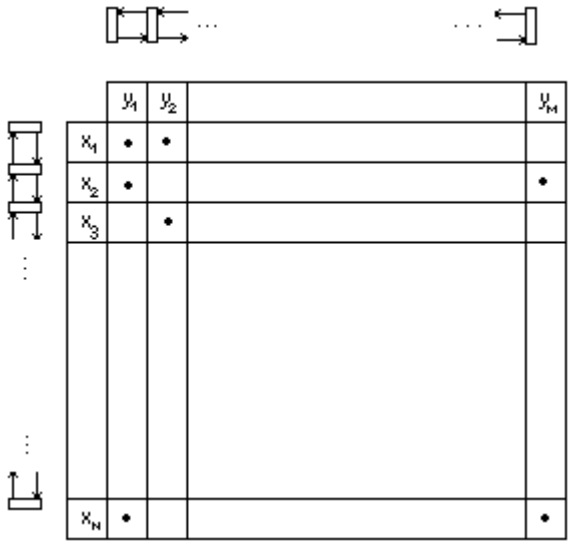
7. Implementace

7.1 Vnitřní paměťová reprezentace

Pro vnitřní reprezentaci potřebujeme strukturu, kde budeme mít uloženou matici *A* o rozměrech *MxN* .

Musíme počítat s tím, že budeme často vyškrtávat sloupce a řádky z matice, struktura by tedy mela umožňovat snadný přechod přes vynechané sloupce a řádky. Pokud bychom sloupci, resp. řádku přiřadili atribut zda je, či není v matici, museli bychom pokaždé testovat jeho hodnotu a to by bylo časově náročné. Potřebujeme znát offset, o kolik sloupců se máme v matici posunout od sloupce j ke sloupci k (přeskočit všechny vyškrtnuté sloupce).

Použijeme tedy následující strukturu. Matice *A* o rozměrech *MxN* je uložena v paměti jako jednorozměrné homogenní pole prvků typu *char*, vkládání prvků do pole se provádí po řádcích, tzn., adresa prvku V_{ij} = **(A+i*M+j)*. K této struktuře jsou vytvořeny dva obousměrně zřetězené spojové seznamy o *M*, resp., *N* prvcích,každý prvek seznamu nese informace o offsetu adresy, kde je uložen řádek, resp., sloupec matice, viz obrázek Tato reprezentace je vhodná, protože při redukování matice o řádek nebo sloupec jen přepíšeme ukazatele ve spojovém seznamu, toto má pozitivní vliv na rychlost vyhledávání prvků. Adresa prvku V_{ij} je tedy **(A + Seznam_Sloupcu->Offset + Seznam_Radku->Offset)*. Navíc při takovéto adresaci prvků pole pouze sčítáme, což má také značný vliv na urychlení.



Obrázek 7.1 – Vnitřní reprezentace matice pokrytí

7.2 Ovládání programu

Vstup programu je realizován vstupním souborem, kde na začátku je uveden rozměr matice, čísla oddělená mezerami, a pak následuje matice uložená po řádcích, první řádek obsahuje ceny sloupců. Čísla v řádcích jsou odděleny rovněž mezerami. Vstupní soubor se zadává z příkazové řádky.

Výstup se vypisuje na obrazovku a zároveň do souboru, který může být uvedený jako druhý parametr programu z příkazové řádky. Implicitní název výstupního souboru je „out.dat“.

Program vyřeší vstupní matici nejdříve pomocí heuristik a potom pomocí exaktních metod.

Názvy funkcí implementovaných v programu pro řešení problému pokrytí:

```
Funkce přirozená heuristika
```

void NaturalHeuristic (struct CoveringMatrix *CM,struct HeaderSol *SolUCP)

```
Heuristika,která vybírá sloupce podle vypočteného sloupcového příspěvku
```

void ContribAddHeuristic (struct CoveringMatrix *CM,struct HeaderSol *SolUCP)

```
Pokročilé heuristiky, které odebírají sloupce dle vypočítaného scóre
```

void RothHeuristic (struct CoveringMatrix *CM,struct HeaderSol *SolUCP)

void MichalskiHeuristik (struct CoveringMatrix *CM,struct HeaderSol *SolUCP)

void NeculaHeuristik (struct CoveringMatrix *CM,struct HeaderSol *SolUCP)

void BowmanHeuristik (struct CoveringMatrix *CM,struct HeaderSol *SolUCP)

void ServitHeuristik (struct CoveringMatrix *CM,struct HeaderSol *SolUCP)

```
Pokročilé heuristiky, které přidávají sloupce do řešení dle vypočítaného
scóre
```

void PositivRothHeuristic (struct CoveringMatrix *CM,struct HeaderSol *SolUCP)

void PositivMichalskiHeuristik (struct CoveringMatrix *CM,struct HeaderSol *SolUCP)

void PositivNeculaHeuristik (struct CoveringMatrix *CM,struct HeaderSol *SolUCP)

void PositivBowmanHeuristik (struct CoveringMatrix *CM,struct HeaderSol *SolUCP)

void PositivServitHeuristik (struct CoveringMatrix *CM,struct HeaderSol *SolUCP)

Vylepšená metoda větví a hranic

int BABLLB (struct CoveringMatrix *CM,struct HeaderSol *SolUCP)

Metoda větví a hranic s imlementovanou funkcí Raiser ~ AURA

int AURA (struct CoveringMatrix *CM,struct HeaderSol *SolUCP)

8. Experimentální výsledky

8.1 Heuristiky

Nejprve byly prováděny testy na zcela náhodných maticích vygenerovaných jednoduchým generátorem matic, jež jsem pro tento účel napsal. Měření byla prováděna na maticích při změně nějakého parametru matice (počet řádků, počet sloupců, hustota matice). Vzhledem k faktu, že při hledání řešení matice hraje roli zčásti i náhoda (pokud je nalezená hodnota "scóre" stejná pro více sloupců, vybírá se sloupec jež bude z matice odebrán / přidán do řešení náhodně.

Největší roli hraje náhodné vybrání sloupce asi u Rothovy heuristiky a její mírně pozměněné verzi (viz výše) pozitivní Rothovy heuristiky. U této heuristiky je totiž funkce pro výpočet "skóre" velmi jednoduchá a počet nalezených sloupců se stejným "scóre" je zde tedy nejvyšší.

Proto bylo měření každé jedné konkrétní matice opakováno 100x. Byla zaznamenána nejlepší nalezená cena (mincost), nejhorší takto nalezená cena (maxcost) a průměrná cena (averagecost) , tj. průměr ze všech 100 naměřených hodnot. A samozřejmě výpočetní čas (zaznamenaný ve vteřinách). Takto bylo pro každou konkrétní matici (pevný počet řádků, pevný počet sloupců, pevná hustota) vygenerováno 1000 instancí a na nich proběhlo měření. (zdrojový soubor naměřených hodnot viz cd příloha).

Při vyhodnocování odchylky jednotlivých cen byla tato vždy vztažena k nejnižší nalezené ceně pro danou matici. Odchylka byla počítána podle vzorce: ((nalezená cena - nejnižší cena) / nejnižší cena). Tyto odchylky spolu s výpočetním časem (ve vteřinách) uvedeny v tabulkách níže (každé číslo v tabulce je spočteno jako průměr z 1000 naměřených matic). Dále jsou uvedeny grafy odchylek a výpočetního času v závislosti na konkrétní matici (počet sloupců, počet řádků, hustota matice) a použité heuristice. V grafech jsou body každé konkrétní heuristiky proloženy spojnicí čistě z důvodu přehlednost. Vzhledem k poměrně časté situaci, kdy je více bodů různých heuristik soustředěno do jednoho místa. Při měření závislosti výpočetního času a velikosti odchylky jednotlivých heuristik na počtu řádků, byla provedena dvoje měření. Jedno pro matici o 100 sloupcích a druhé pro matici o 150 sloupcích.

Nakonec byla provedena měření na reálních instancích problému pokrytí. Tj. na maticích vygenerovaných programem Boom a následně zpracovány do grafů.

Při vytváření grafů byly použity údaje z barevně označených souhrnů výsledků měření. A sice žlutě byly označeny údaje sloužící k vytvoření grafů o závislosti zkoumaných údajů (výpočetní čas a odchylka) na počtu řádků pro matici se 100 sloupci. Bledě modře údaje sloužící k vytvoření grafů o závislosti zkoumaných údajů (výpočetní čas a odchylka) na počtu řádků pro matici se 150 sloupci. Červeně byly označeny údaje, které posloužily k vytvoření grafů v závislosti na počtu sloupců. A zeleně byly označeny údaje zkoumající údaje v závislosti na hustotě matice.

			natural heuristic				contrib add heuristic			
sloupce	řádky	hustota	time	mincost	maxcost	averagecost	time	mincost	maxcost	averagecost
100	25	5	0,027299	0,106833	0,159687	0,131493	0,033548	0,106833	0,159687	0,131468
100	50	5	0,046607	0,113186	0,185235	0,147048	0,058454	0,113182	0,185443	0,147009
100	75	5	0,077982	0,087265	0,138493	0,111276	0,098902	0,087456	0,138564	0,111171
100	100	25	0,068799	0,830736	0,954473	0,8902	0,120253	0,830736	0,954473	0,890014
100	150	5	0,22963	0,092	0,147057	0,117726	0,29908	0,091927	0,147234	0,117872
135	203	2	0,840469	0,043314	0,072654	0,056938	1,114773	0,04338	0,072465	0,056878
147	56	7	0,068148	0,305758	0,453713	0,372737	0,092343	0,30583	0,453609	0,373223
150	25	5	0,036723	0,161358	0,283052	0,215466	0,044053	0,161588	0,28257	0,214744
150	50	5	0,065294	0,189513	0,310367	0,244016	0,081167	0,18937	0,310279	0,243844
150	75	5	0,113513	0,198461	0,323906	0,255749	0,156245	0,198677	0,323863	0,255824
150	150	5	0,288194	0,212225	0,337775	0,270789	0,408437	0,212315	0,337643	0,27082
150	100	10	0,135395	0,449191	0,626869	0,53105	0,202932	0,448866	0,62665	0,530456
206	108	2	0,387147	0,079135	0,129489	0,103244	0,513158	0,079103	0,129337	0,103178
150	100	25	0,100655	0,912946	1,063136	0,987412	0,180099	0,912946	1,063136	0,987491
200	200	5	0,568297	0,307481	0,478658	0,387525	0,863081	0,316931	0,487821	0,397327
200	100	25	0,129987	0,951837	1,078585	1,013337	0,232905	0,951837	1,078728	1,013136
200	200	15	0,331306	0,867061	1,067782	0,963641	0,624107	0,867182	1,067613	0,963547
250	100	25	0,171056	0,920933	1,093881	1,005003	0,328222	0,920933	1,093756	1,00537
200	200	25	0,288205	1,021216	1,157748	1,088011	0,544974	1,021216	1,157748	1,087629
200	200	30	0,264731	1,057605	1,185218	1,117825	0,516623	1,05798	1,185218	1,119129

			roth heuristic				positive roth heuristic			
sloupce	řádky	hustota	time	mincost	maxcost	averagecost	time	mincost	maxcost	averagecost
100	25	5	0,215009	0,027845	0,120342	0,070457	0,055039	0,195341	0,383816	0,290897
100	50	5	0,258161	0,031409	0,11909	0,072185	0,096369	0,26231	0,442483	0,354249
100	75	5	0,246635	0,025229	0,090701	0,056246	0,119041	0,206402	0,335743	0,271552
100	100	25	0,833138	0,043245	0,323222	0,168181	0,098091	0,35472	0,794618	0,570774
100	150	5	0,294343	0,021059	0,076597	0,047385	0,197424	0,20091	0,311621	0,256978
135	203	2	0,403791	0,010151	0,035068	0,022103	0,256269	0,07395	0,119209	0,096738
147	56	7	0,678746	0,044283	0,185623	0,108099	0,164156	0,383872	0,638588	0,514149
150	25	5	0,460682	0,039952	0,168196	0,09758	0,084952	0,244791	0,484532	0,36752
150	50	5	0,609396	0,045622	0,152557	0,093938	0,172738	0,330617	0,5464	0,440995
150	75	5	0,740234	0,042752	0,140327	0,087642	0,242839	0,363203	0,556545	0,460504
150	150	5	1,069047	0,038434	0,125405	0,079571	0,428436	0,385771	0,555315	0,472011
150	100	10	1,094193	0,048514	0,198987	0,117597	0,232484	0,416748	0,670146	0,547255
206	108	2	1,364682	0,030535	0,07597	0,051773	0,6749	0,220366	0,309143	0,265712
150	100	25	1,781892	0,09483	0,41806	0,242994	0,147793	0,44803	0,941546	0,691326
200	200	5	2,637483	0,043549	0,134532	0,086564	0,878213	0,423556	0,585456	0,50527
200	100	25	3,12244	0,150446	0,497278	0,309437	0,193638	0,521389	1,045111	0,778299
200	200	15	4,437471	0,075529	0,288398	0,174202	0,452781	0,462118	0,741728	0,601155
250	100	25	4,923219	0,190474	0,549601	0,354376	0,251992	0,572625	1,103016	0,836553
200	200	25	5,827339	0,113269	0,399854	0,244864	0,350975	0,453371	0,866223	0,658885
200	200	30	6,47425	0,122308	0,438008	0,267869	0,321122	0,432912	0,902279	0,663994

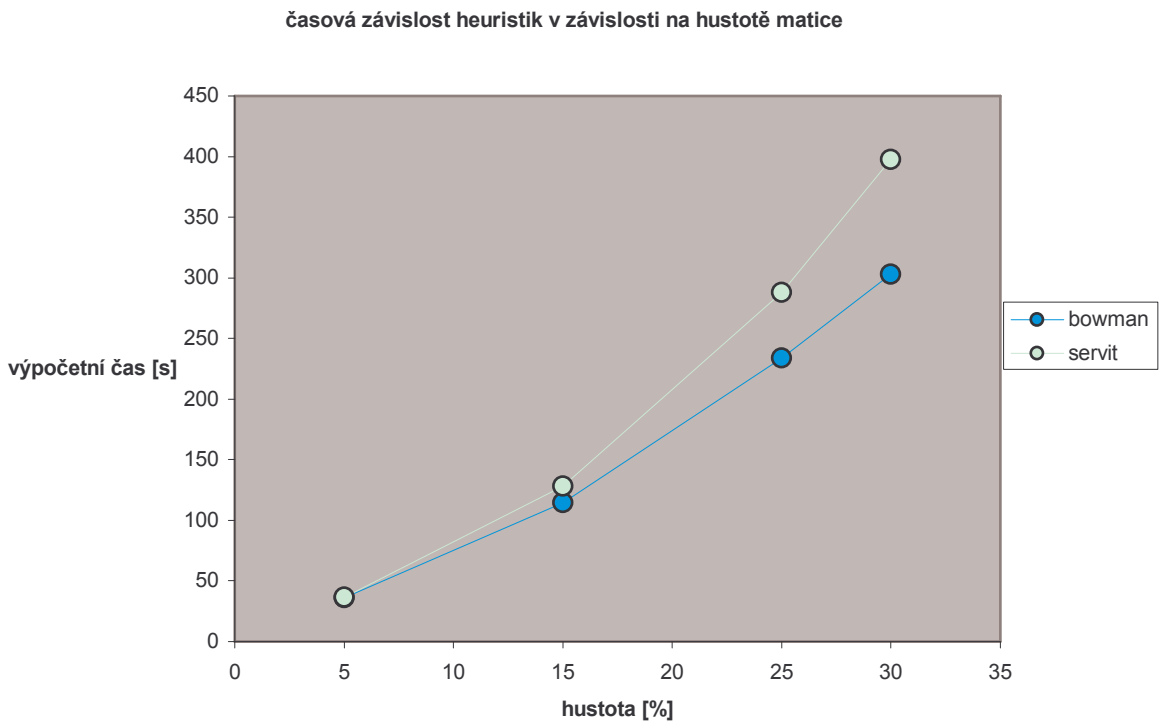
			michalski heuristic				positive michalski heuristic			
sloupce	řádky	hustota	time	mincost	maxcost	averagecost	time	mincost	maxcost	averagecost
100	25	5	0,213317	0,577065	1,143479	0,844632	0,063511	0,970885	1,745198	1,353321
100	50	5	0,269708	0,526472	0,926634	0,713885	0,113233	1,04495	1,584406	1,31437
100	75	5	0,253515	0,322007	0,581117	0,446345	0,135435	0,697988	1,030563	0,863049
100	100	25	0,857483	1,685172	3,154364	2,359384	0,109808	2,637934	4,457038	3,523617
100	150	5	0,301073	0,233853	0,420731	0,32331	0,217302	0,572242	0,796007	0,682461
135	203	2	0,393656	0,094832	0,186256	0,138544	0,269317	0,215445	0,32015	0,268009
147	56	7	0,742618	1,253137	1,931391	1,577577	0,184886	2,234362	3,152384	2,693386
150	25	5	0,476215	1,020653	1,753603	1,366263	0,09745	1,635428	2,672209	2,155087
150	50	5	0,659629	0,971595	1,495707	1,22391	0,189452	1,737705	2,452652	2,09313
150	75	5	0,794502	0,901856	1,334859	1,1064	0,274014	1,717143	2,289312	2,002835
150	150	5	1,122244	0,780738	1,110715	0,940839	0,48669	1,576383	1,96456	1,768409
150	100	10	1,164504	1,387885	2,083465	1,715169	0,273133	2,461941	3,321484	2,889206
206	108	2	1,398972	0,42854	0,604946	0,512901	0,73737	0,836127	1,05409	0,945124
150	100	25	1,846725	1,957419	3,595608	2,721197	0,162544	3,076382	5,07746	4,040875
200	200	5	2,67734	1,082715	1,44146	1,255095	0,938289	2,02236	2,433236	2,227181
200	100	25	3,238787	2,104359	3,815679	2,897679	0,21541	3,350188	5,460833	4,375991
200	200	15	4,517165	1,97216	2,988236	2,454877	0,484777	3,169841	4,321491	3,739056
250	100	25	5,037043	2,235058	4,038722	3,08943	0,277559	3,456944	5,627766	4,513296
200	200	25	5,963886	2,062747	3,484299	2,744811	0,367969	3,150629	4,816181	3,970271
200	200	30	6,597176	2,031874	3,648422	2,792955	0,334621	3,080053	5,019804	4,033676

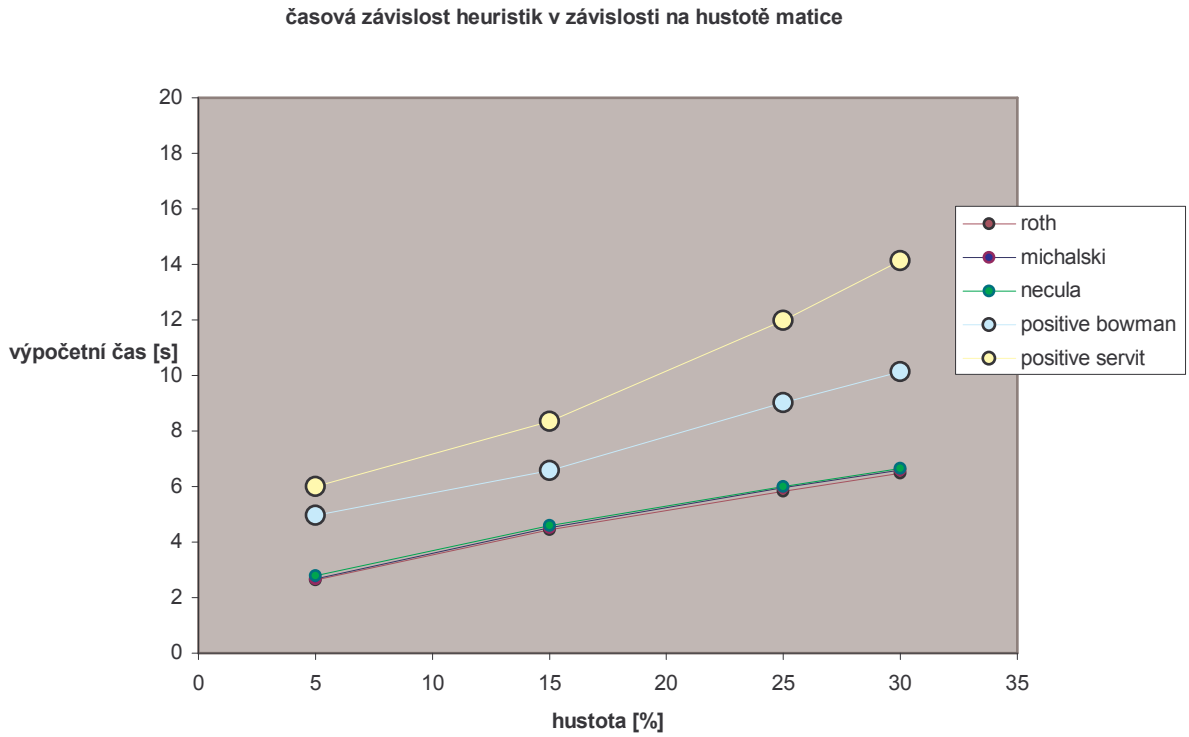
			necula heuristic				positive necula heuristic			
sloupce	řádky	hustota	time	mincost	maxcost	averagecost	time	mincost	maxcost	averagecost
100	25	5	0,24342	0,259655	0,430158	0,342863	0,143346	4,895105	6,636736	5,766843
100	50	5	0,283908	0,237187	0,410157	0,320861	0,18766	3,464723	4,545825	3,997059
100	75	5	0,259613	0,171567	0,293196	0,229722	0,209191	2,336125	2,987676	2,661574
100	100	25	0,865084	0,086725	0,418203	0,236814	0,10428	0,513286	1,69719	1,050871
100	150	5	0,302325	0,131323	0,237289	0,182056	0,276728	1,427998	1,845038	1,636185
135	203	2	0,373137	0,061525	0,102883	0,081606	0,363312	0,851252	1,060646	0,956043
147	56	7	0,774694	0,293424	0,518118	0,39582	0,26401	4,297646	6,148003	5,214915
150	25	5	0,552064	0,303199	0,48003	0,387681	0,220617	6,152741	8,367983	7,253225
150	50	5	0,698785	0,32735	0,511431	0,413708	0,317066	5,288063	6,733536	6,022091
150	75	5	0,818477	0,329146	0,500555	0,409806	0,390201	4,665958	5,799086	5,231231
150	150	5	1,155341	0,325796	0,49621	0,408498	0,571201	3,578495	4,371345	3,970335
150	100	10	1,185415	0,274258	0,513459	0,385329	0,310106	3,057421	4,804199	3,929301
206	108	2	1,412161	0,216118	0,284081	0,249539	0,984135	3,624881	4,052708	3,841827
150	100	25	1,874165	0,100647	0,428308	0,247383	0,150817	0,456345	1,039428	0,732003
200	200	5	2,782221	0,349882	0,513225	0,427551	0,969975	4,124996	4,95606	4,539212
200	100	25	3,287908	0,153167	0,498298	0,309604	0,200258	0,519347	1,050895	0,781682
200	200	15	4,597671	0,10292	0,331257	0,207558	0,467853	0,601217	1,342097	0,946543
250	100	25	5,130617	0,187341	0,553776	0,354317	0,260935	0,572448	1,101099	0,836806
200	200	25	5,993288	0,113305	0,404475	0,246582	0,352567	0,456677	0,887578	0,667784
200	200	30	6,657012	0,121365	0,439258	0,268466	0,325338	0,431582	0,903277	0,663686

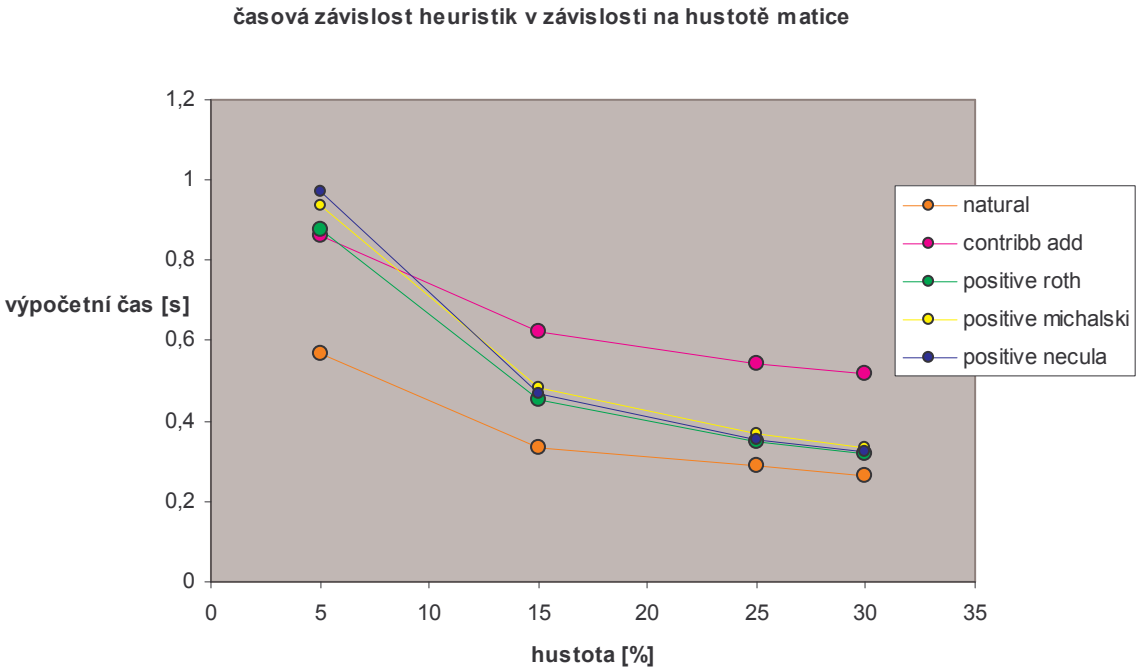
			bowman heuristic				positive bowman heuristic			
sloupce	řádky	hustota	time	mincost	maxcost	averagecost	time	mincost	maxcost	averagecost
100	25	5	0,669162	0,270966	0,501224	0,383807	0,084812	0,227913	0,601031	0,408902
100	50	5	1,060024	0,249056	0,415442	0,328525	0,17405	0,226739	0,478981	0,350242
100	75	5	0,934644	0,152909	0,26118	0,204411	0,186208	0,134307	0,318047	0,223363
100	100	25	15,3716	1,427755	1,535817	1,481643	0,98047	1,131002	1,490605	1,310031
100	150	5	1,192715	0,110164	0,183385	0,145105	0,342002	0,098074	0,224478	0,159556
135	203	2	0,988922	0,04909	0,083268	0,065949	0,311258	0,039002	0,1149	0,076219
147	56	7	5,166229	0,655677	0,84616	0,747598	0,580194	0,574967	0,831355	0,6992
150	25	5	1,971535	0,462658	0,803206	0,625215	0,197164	0,447301	0,883287	0,657978
150	50	5	3,540491	0,450179	0,667946	0,553219	0,481582	0,421334	0,700912	0,557085
150	75	5	5,089018	0,438749	0,604319	0,518241	0,766462	0,403424	0,628681	0,511205
150	150	5	9,843955	0,416237	0,510571	0,461709	1,899071	0,372487	0,528044	0,447828
150	100	10	12,50718	0,871301	0,969374	0,918159	1,339957	0,73033	0,927929	0,827453
206	108	2	7,456722	0,173442	0,268381	0,218717	1,558371	0,169811	0,334874	0,251238
150	100	25	49,11262	1,616218	1,742397	1,677368	2,15602	1,277528	1,667551	1,471747
200	200	5	36,29829	0,591717	0,67735	0,632875	4,952852	0,538836	0,679418	0,607282
200	100	25	114,3983	1,679272	1,795899	1,737559	3,97781	1,307012	1,671089	1,488056
200	200	15	114,6916	1,462548	1,540807	1,500938	6,57908	1,203373	1,421922	1,311741
250	100	25	226,9612	1,754077	1,86397	1,808339	6,308822	1,322909	1,69826	1,511418
200	200	25	233,6177	1,713854	1,802614	1,756932	9,016165	1,344333	1,657253	1,500825
200	200	30	303,1216	1,814475	1,911409	1,862032	10,12524	1,363695	1,764816	1,565159

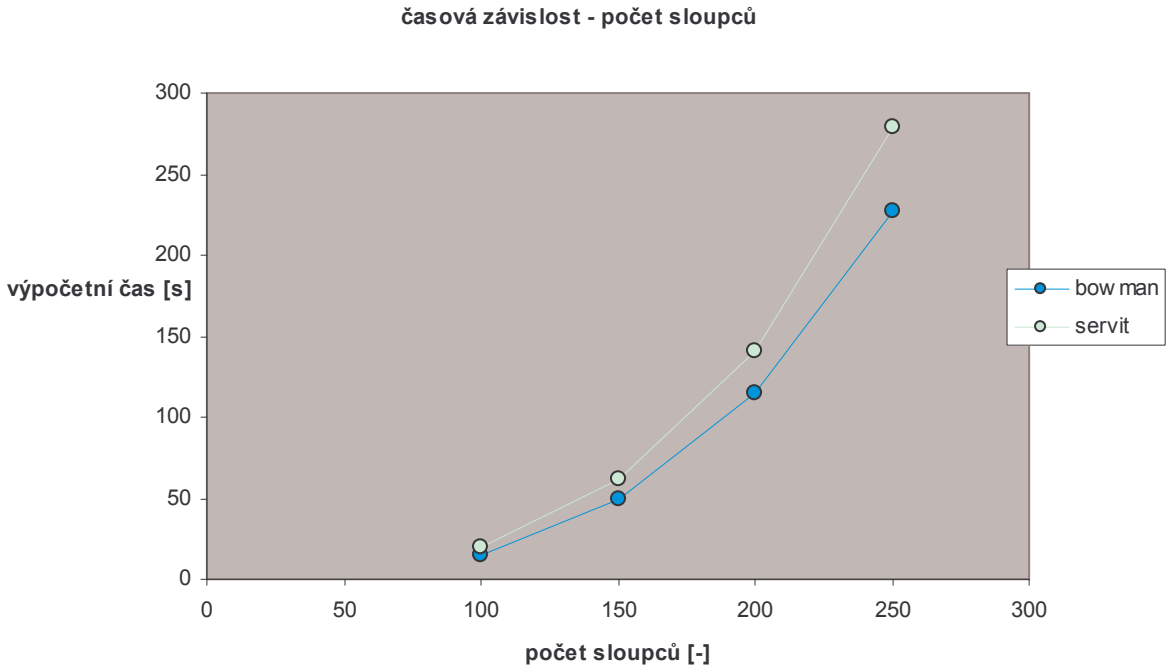
			servit heuristic				positive servit heuristic			
sloupce	řádky	hustota	time	mincost	maxcost	averagecost	time	mincost	maxcost	averagecost
100	25	5	0,756888	0,019303	0,028122	0,023375	0,106303	0,019807	0,020148	0,019978
100	50	5	1,178394	0,023087	0,029708	0,026262	0,211104	0,022531	0,022895	0,022739
100	75	5	1,002642	0,015211	0,01989	0,017529	0,229961	0,015736	0,016003	0,015883
100	100	25	19,49684	0,028356	0,030326	0,029265	1,435484	0,060314	0,060314	0,060314
100	150	5	1,255505	0,01318	0,016328	0,01473	0,402288	0,014965	0,015309	0,015136
135	203	2	0,986819	0,002554	0,003931	0,003195	0,34073	0,006185	0,006573	0,006382
147	56	7	5,711413	0,035284	0,041742	0,038141	0,720035	0,030101	0,030386	0,030258
150	25	5	2,203068	0,025531	0,042987	0,03374	0,248567	0,023486	0,023696	0,023593
150	50	5	3,913327	0,028797	0,038009	0,033175	0,593824	0,024035	0,02434	0,024178
150	75	5	5,54938	0,030006	0,035687	0,032702	0,955254	0,022293	0,022452	0,022365
150	150	5	10,30652	0,021693	0,025205	0,023357	2,275822	0,026375	0,026446	0,026406
150	100	10	13,85943	0,028064	0,031016	0,029452	1,706394	0,036213	0,036392	0,036295
206	108	2	7,8587	0,00917	0,016748	0,012732	1,770175	0,01174	0,012027	0,01188
150	100	25	61,7563	0,030429	0,033191	0,031731	3,091646	0,049448	0,049448	0,049448
200	200	5	36,76236	0,018485	0,022236	0,020194	5,995822	0,037695	0,037734	0,037715
200	100	25	140,6903	0,037736	0,040762	0,039255	5,441434	0,028714	0,028714	0,028714
200	200	15	128,3443	0,015222	0,017872	0,016592	8,347012	0,04552	0,04552	0,04552
250	100	25	278,8124	0,049542	0,053524	0,051477	8,461117	0,02004	0,02004	0,02004
200	200	25	288,1594	0,038044	0,040026	0,038966	11,98819	0,027809	0,027809	0,027809
200	200	30	397,6201	0,04478	0,047621	0,046027	14,13748	0,02471	0,02471	0,02471

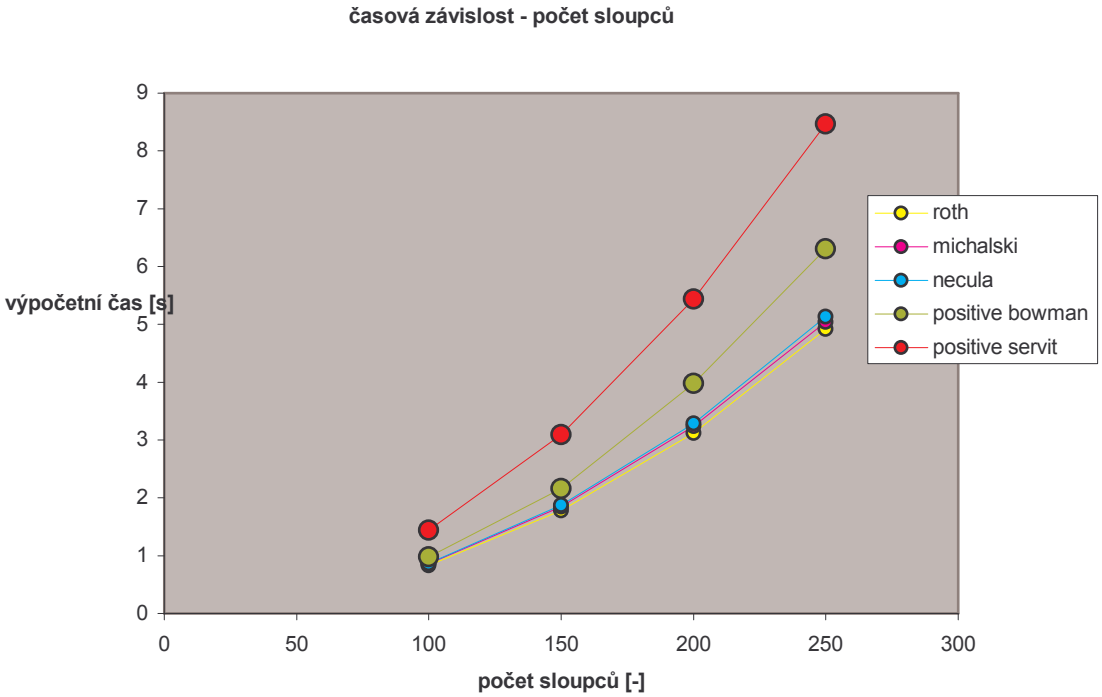
8.1.1 Zhodnocení heuristik

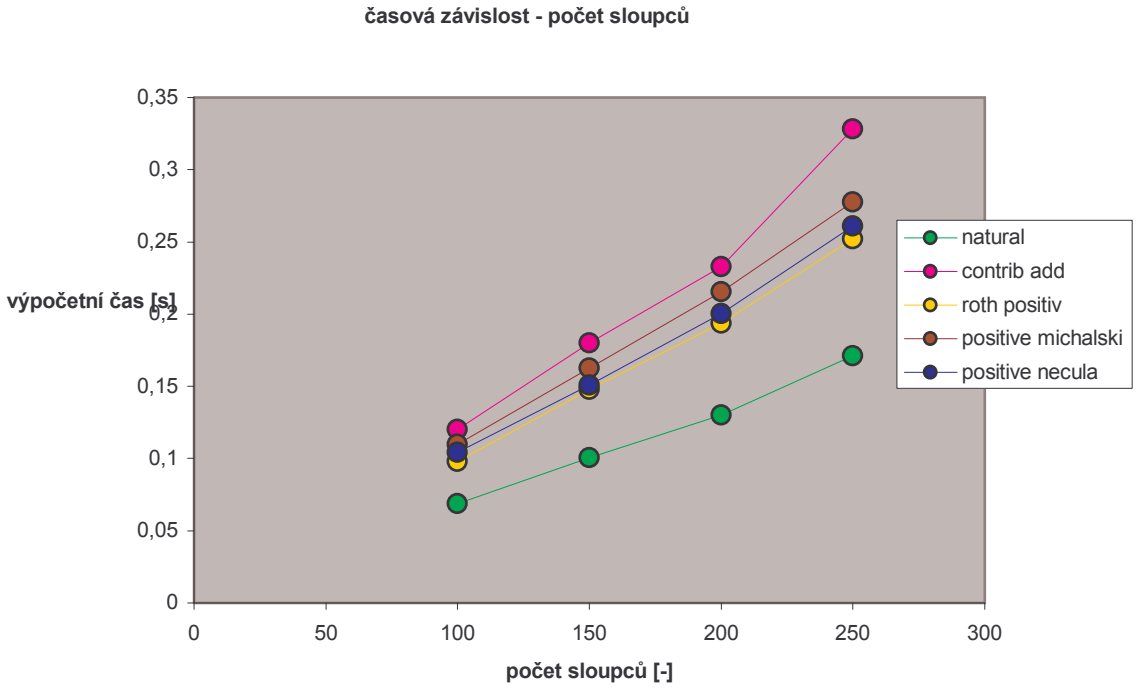


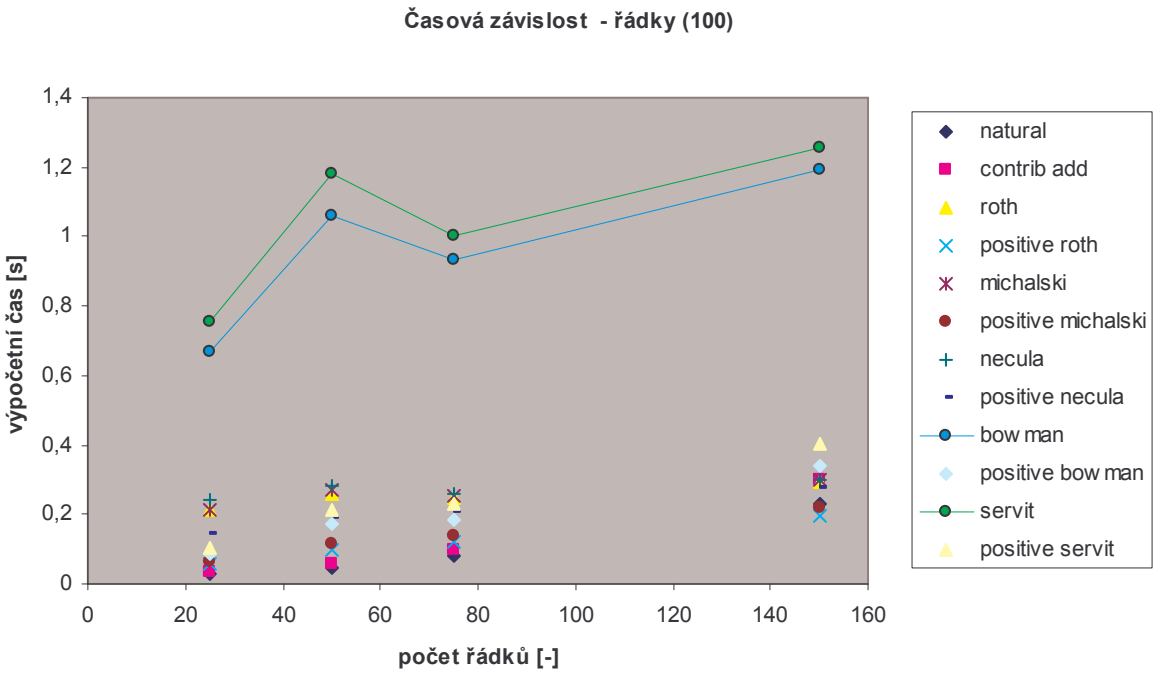


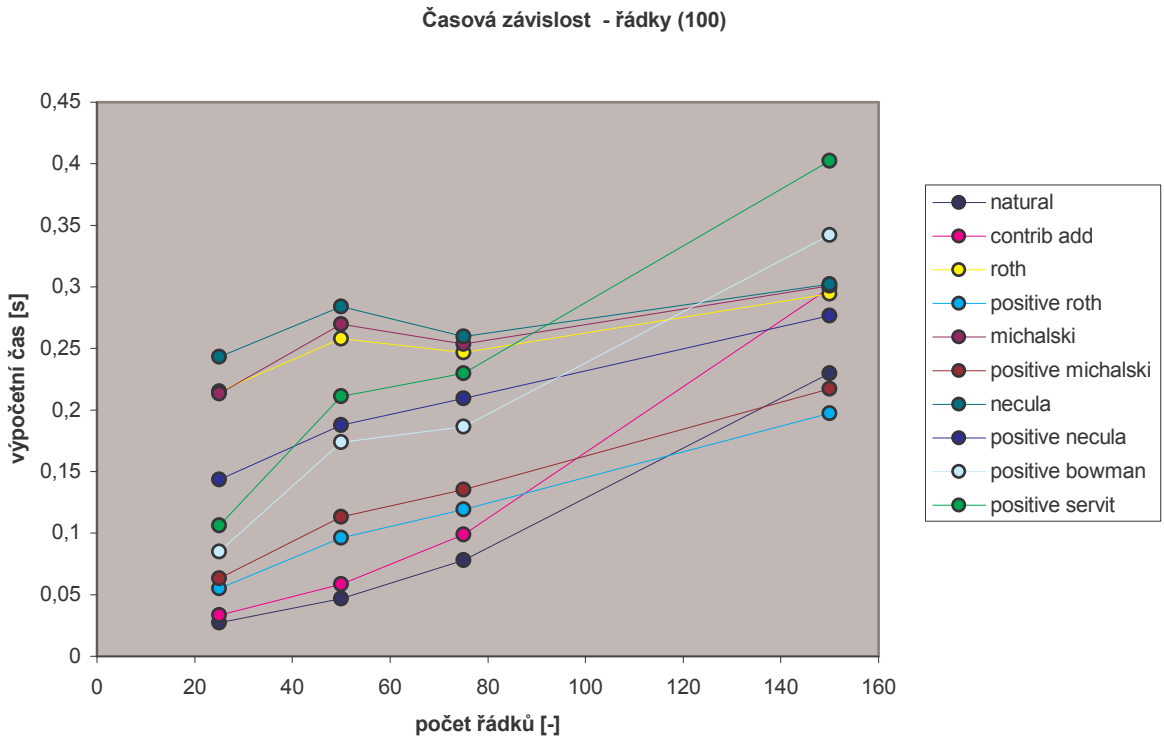

časová závislost heuristik v závislosti na hustotě matice
1,2
1
0,8
0,6
0,4
0,2
0
výpočetní čas [s]
0
5
10
15
20
25
30
35
hustota [%]
natural
contribb add
positive roth
positive michalski
positive necula

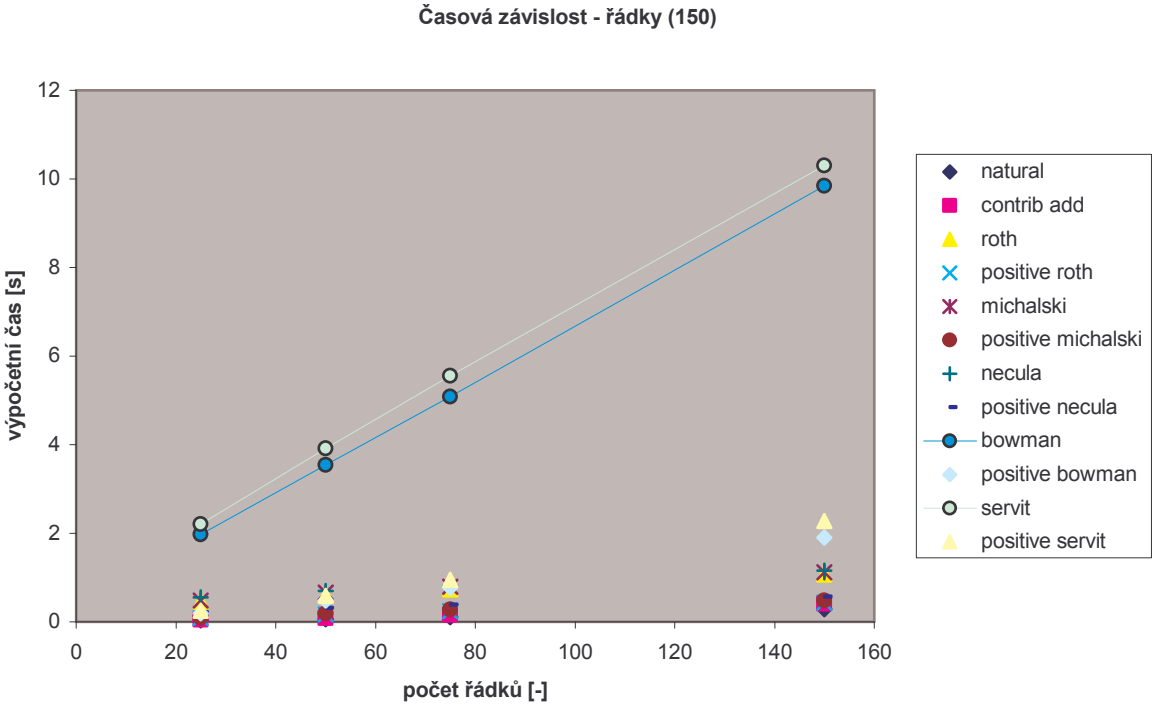

časová závislost - počet sloupců
300
250
200
150
100
50
0
výpočetní čas [s]
0
50
100
150
200
250
300
počet sloupců [-]
bow man
servit

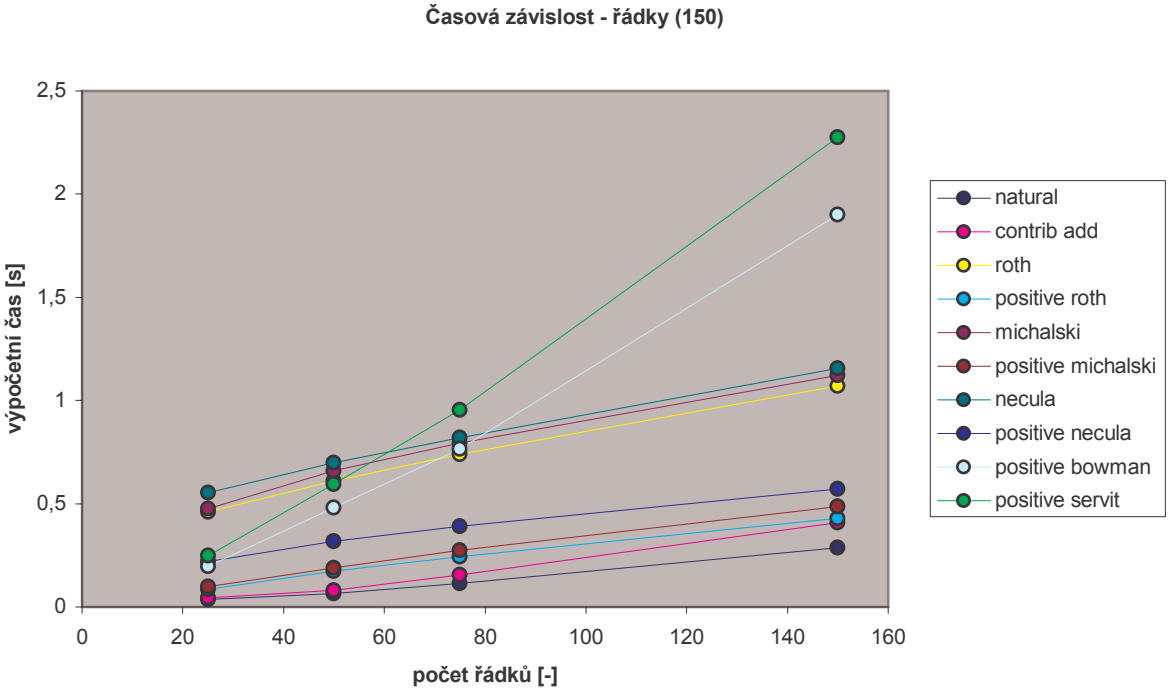

časová závislost - počet sloupců
výpočetní čas [s]
počet sloupců [-]
roth
michalski
necula
positive bowman
positive servit
0
1
2
3
4
5
6
7
8
9
0
50
100
150
200
250
300


časová závislost - počet sloupců
výpočetní čas [s]
počet sloupců [-]
natural
contrib add
roth positiv
positive michalski
positive necula
0
0,05
0,1
0,15
0,2
0,25
0,3
0,35
0
50
100
150
200
250
300


Časová závislost - řádky (100)
výpočetní čas [s]
počet řádků [-]
natural
contrib add
roth
positive roth
michalski
positive michalski
necula
positive necula
bow man
positive bow man
servit
positive servit


Časová závislost - řádky (100)
výpočetní čas [s]
počet řádků [-]
natural
contrib add
roth
positive roth
michalski
positive michalski
necula
positive necula
positive bowman
positive servit


Časová závislost - řádky (150)
výpočetní čas [s]
počet řádků [-]
0
2
4
6
8
10
12
0
20
40
60
80
100
120
140
160
natural
contrib add
roth
positive roth
michalski
positive michalski
necula
positive necula
bowman
positive bowman
servit
positive servit


Časová závislost - řádky (150)
výpočetní čas [s]
počet řádků [-]
0
0,5
1
1,5
2
2,5
0
20
40
60
80
100
120
140
160
natural
contrib add
roth
positive roth
michalski
positive michalski
necula
positive necula
positive bowman
positive servit

8.1.1.1 Zhodnocení heuristik - časová závislost

Z grafů je patrné, že nejvíce závislý je výpočetní čas pro použité heuristiky na počtu sloupců matice. Nejrychleji roste výpočetní čas s nárůstem počtu sloupců pro Servitovu a Bowmanovu heuristiku. Tyto dvě heuristiky jsou ostatně i časově nejnáročnější. Souvisí to se složitostí cenové funkce pro výpočet "skóre" jednotlivých sloupců.

Zajímavé bylo zjištění závisloti výpočetního času na hustotě matice. Jak v případě Bowmanovy tak Servitovy heuristiky roste výpočetní čas s vyšší hustotou matice prudce nahoru. U dalších dvou heuristik (pozitivní Bowmanova a pozitivní Servitova heuristika) se rovněž zvyšuje časová náročnost výpočtu s větší hustotou matice, avšak růst již není tak strmý.

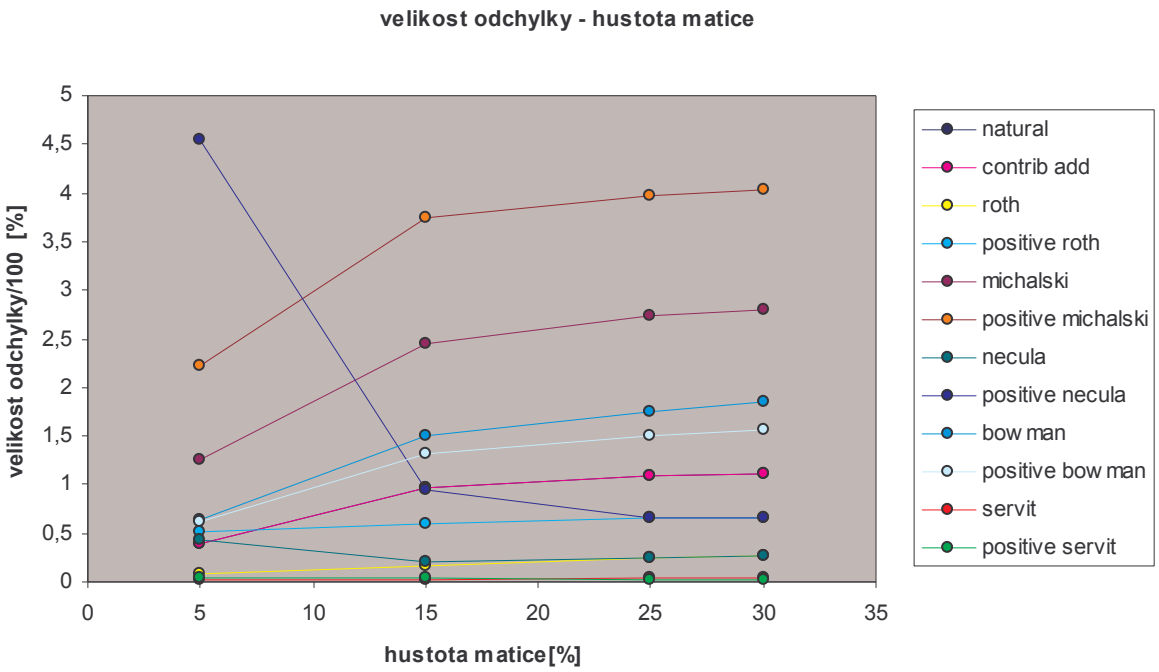
Heuristiky Roth, Michlaski a Necula jsou z hlediska časové složitosti výpočtu na hustotě matice víceméně stejné. Čas potřebný k výpočtu hustších matic roste velmi zvolna.

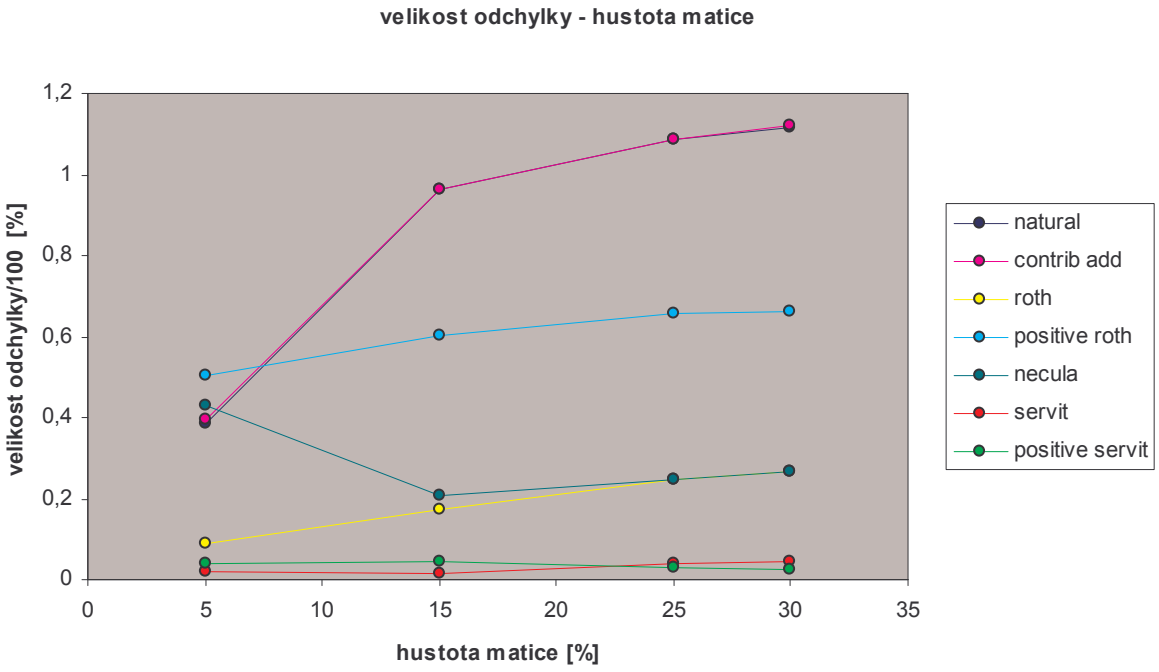
Obdoby těchto tří heuristik (pozitivní Rothova, pozitivní Michalskeho a pozitivní Neculova heuristika) mají opět téměř shodný průběh s jedním zásadním rozdílem. Výpočetní čas je pro ně tím nižší, čím větší hustotu má matice. Toto je tedy potvrzením předpokladu, který vychází ze vzorce pro výpočet "skóre" sloupce a tím zda je sloupec přidán do řešení nebo odebrán z matice. Čím má totiž matice vyšší hustotu, tím je pravděpodobnější, že jeden řádek bude pokryt mnoha sloupci (neboli jeden sloupec pokrývá mnoho více řádků) a tak při pozitivním přístupu (pozitivní heuristiky Roth, Michalski , Necula) odebereme z matice nejlepší sloupec, který přidáme do řešení a po odebrání všech řádků, které pokrýval, se nám matice významně zmenší. Oproti tomu, pokrýval-li by tento sloupec málo řádků, matice by se nám tak významně nezmenšila.

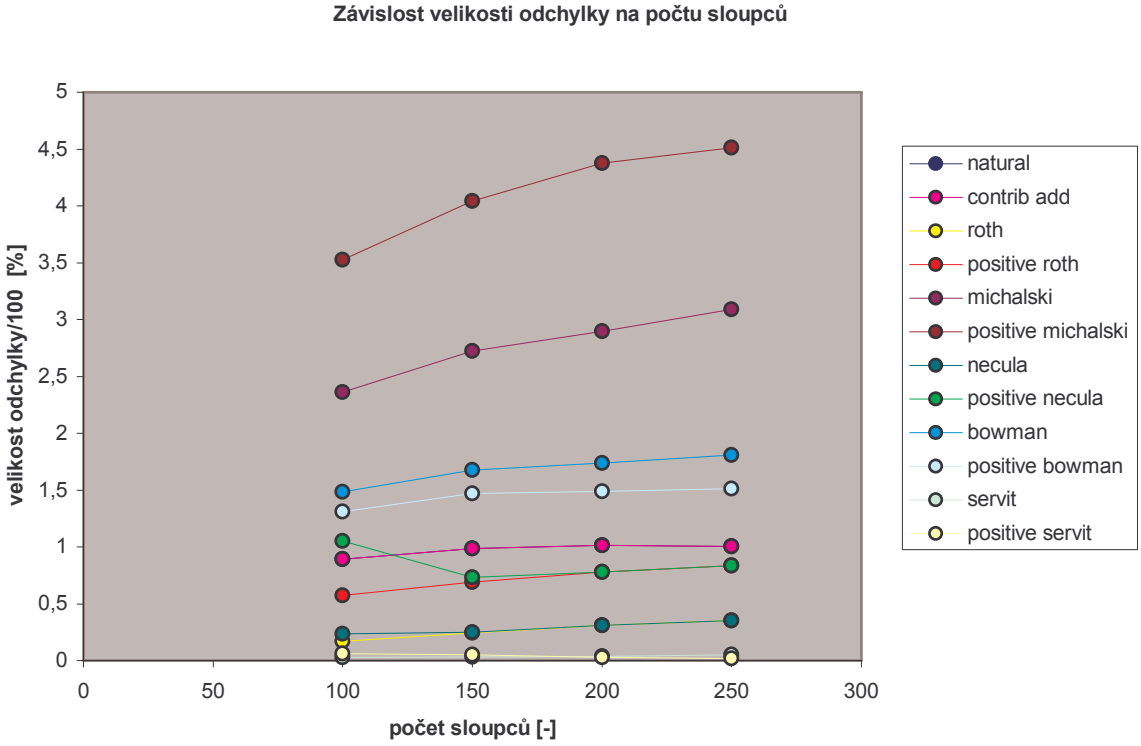
Pozitivní Servitova a Bowmanova heuristika se tomuto předpokladu vymyká . Je to dáno složitostí výpočtu "skóre" pro ohodnocení sloupce. Podíváme-li se totiž podrobně na cenové funkce ohodnocování sloupců u těchto heuristik , všimneme si, že pro každý řádek, který sloupec pokrývá, musíme provést další výpočet. Tedy čím více řádků sloupec pokrývá, tím časově náročnější bude spočtení "skóre" pro jeden sloupec. Časová složitost tohoto výpočtu je zřejmě tak vysoká , že převyšuje čas ušetřený odebráním více řádků (a tím zredukování matice) při větší hustotě matice.

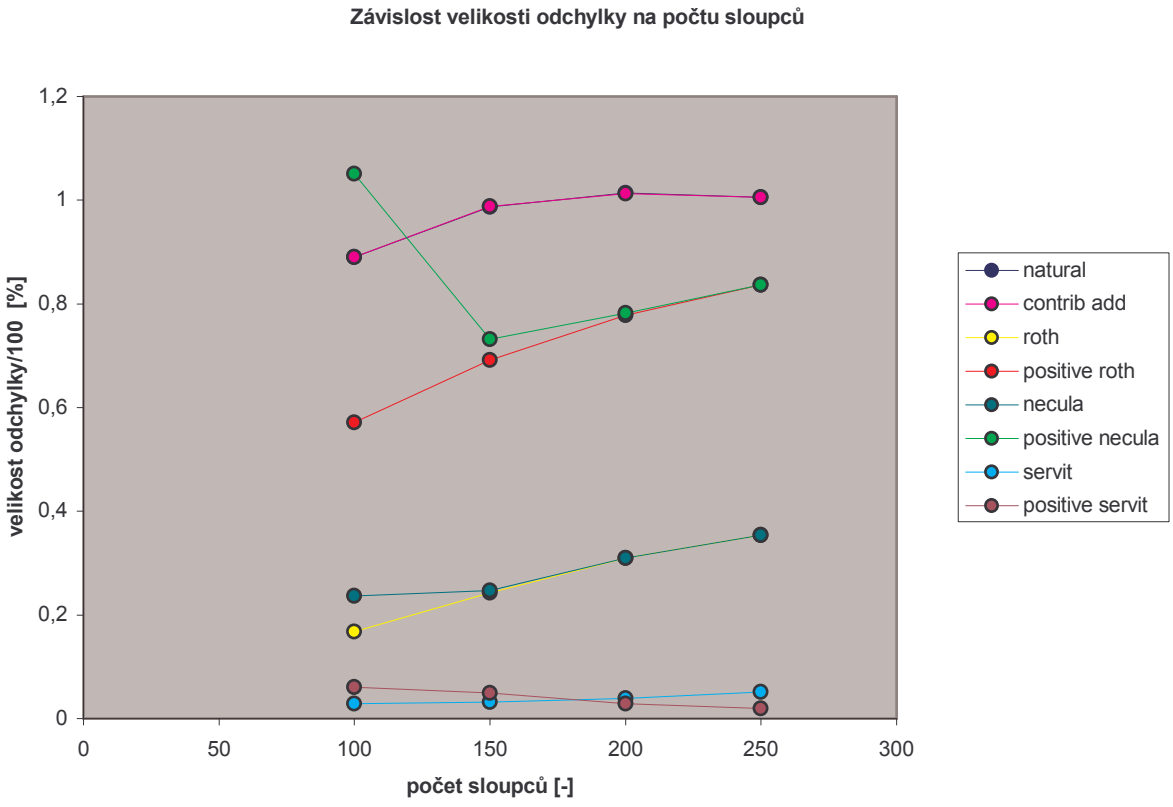
Heuristiky Natural a Contrib add mají rovněz při vyšší hustotě matice nižší čas potřebný pro nalezení řešení. Je to dáno opět způsobem výběru nejlepšího sloupce (vybere se sloupec pokrývající nejvíce řádků velmi jednoduchou cenovou funkcí , přidá se do řešení a z matice se odeberou řádky, které pokrýval) a lze je tedy zařadit do skupiny spolu s pozitivní heuristikou Roth, Necula a Michalski. Detailnější popis těchto dvou heuristik viz [1] na cd příloze.

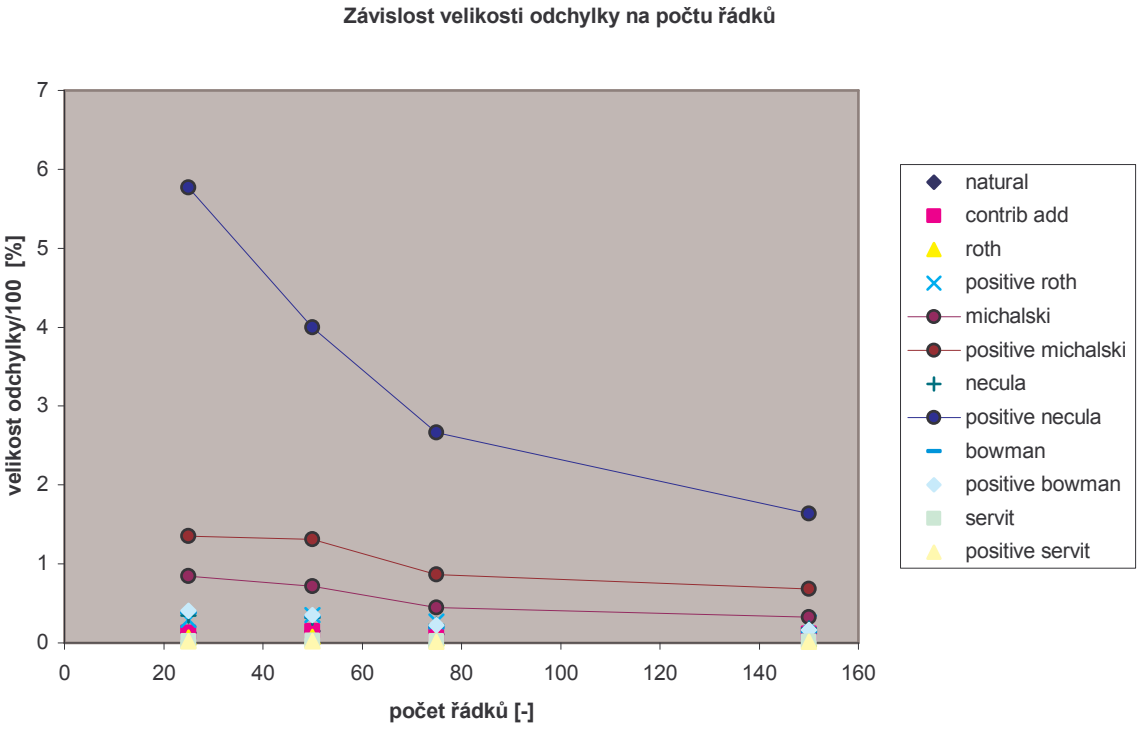
U časové závislosti na počtu řádků je složitost cenových funkcí heuristik Servita a Bowmana nejpatrnější. Jak u klasických (Servit, Bowman) heuristik tak u pozitivních (pozitivní Servitova a pozitivní Bowmanova) je vidět velmi průdký nárůst času nutného pro výpočet. U všech ostatních heuristik je nárůst času velmi zvolny. Pohlédneme-li na grafy, konkrétně na graf zobrazující časovou závislost na počtu řádků matice při počtu sta sloupců matice nemůžeme si nevšimnou oné nesrovnalosti, kdy je výpočet matice při vyšším počtu řádků náhle nižší a poté opět s rostoucím počtem řádků roste. Při přihlédnutí k počtu testů, které byly provedeny můžeme vyloučit např. příznivě vygenerovanou matici či jinou "náhodnou" událost, která by ovlivnila testování. Proto jsou naměřené výsledky charakteristické jen pro dané naměřené instance matice a jak již bylo zmíněno na začátku, spojnice nám pouze činí graf přehlednějším.

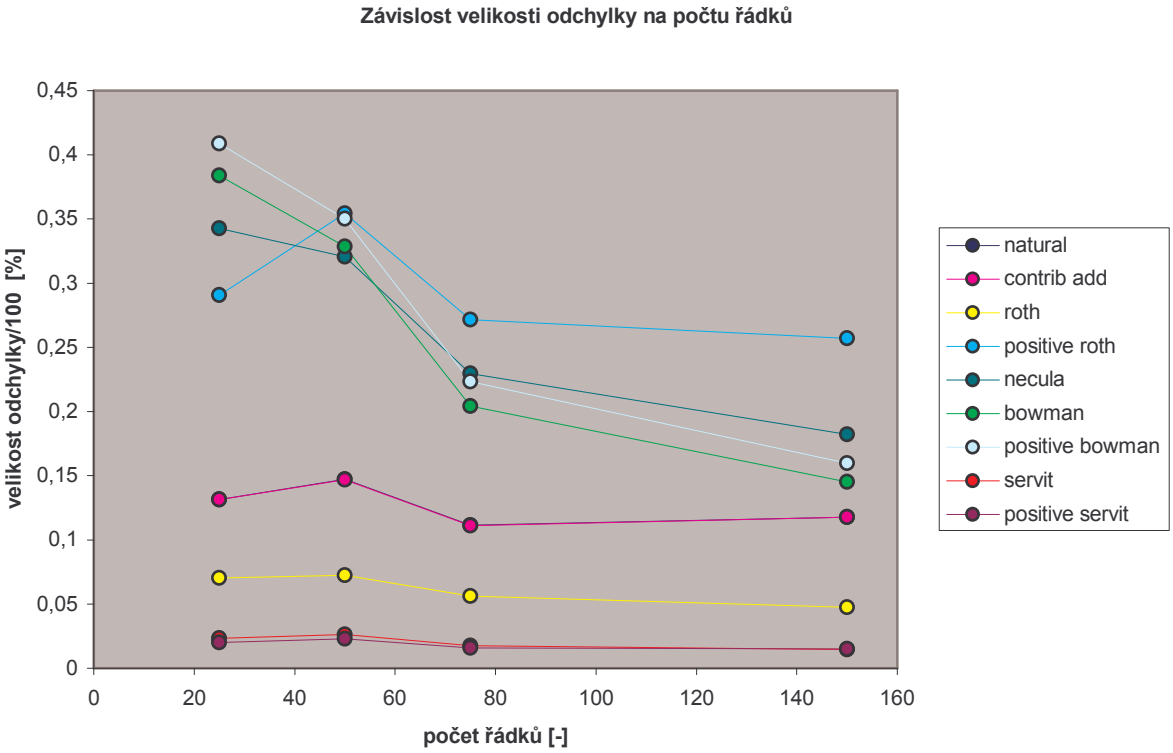

velikost odchylky - hustota matice
velikost odchylky/100 [%]
hustota matice[%]
natural
contrib add
roth
positive roth
michalski
positive michalski
necula
positive necula
bow man
positive bow man
servit
positive servit

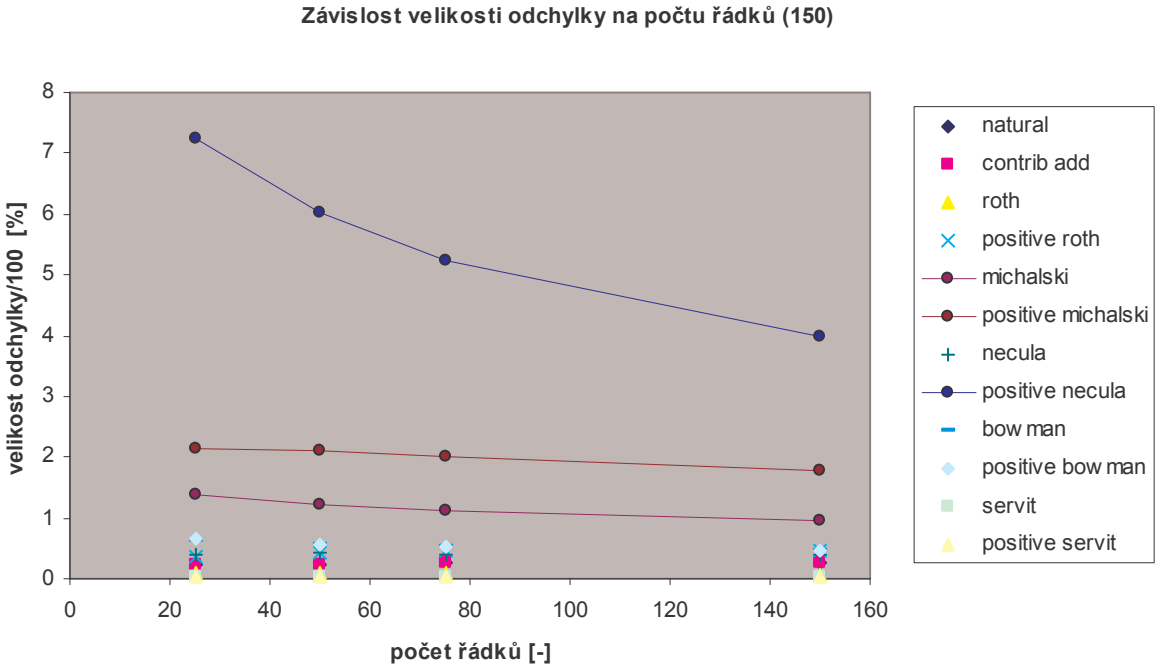

velikost odchylky - hustota matice
velikost odchylky/100 [%]
hustota matice [%]
natural
contrib add
roth
positive roth
necula
servit
positive servit

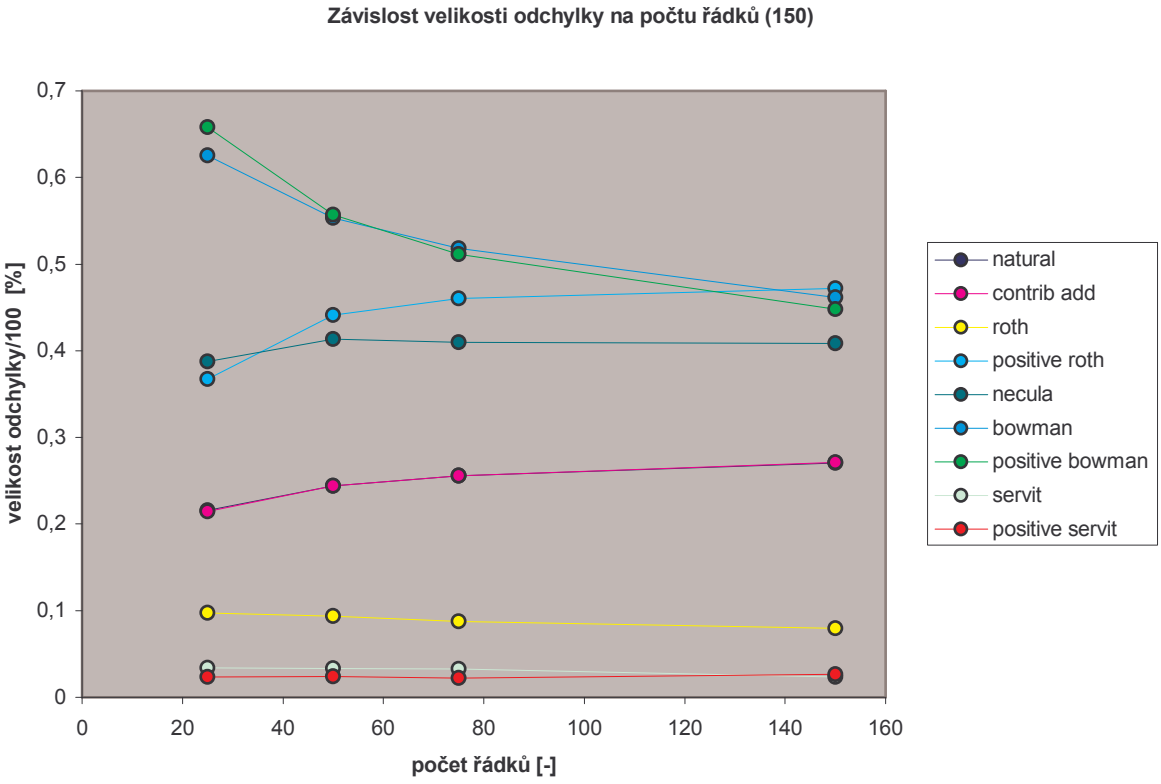

Závislost velikosti odchylky na počtu sloupců
velikost odchylky/100 [%]
počet sloupců [-]
0
0,5
1
1,5
2
2,5
3
3,5
4
4,5
5
0
50
100
150
200
250
300
natural
contrib add
roth
positive roth
michalski
positive michalski
necula
positive necula
bowman
positive bowman
servit
positive servit


Závislost velikosti odchylky na počtu sloupců
velikost odchylky/100 [%]
počet sloupců [-]
0
0,2
0,4
0,6
0,8
1
1,2
0
50
100
150
200
250
300
natural
contrib add
roth
positive roth
necula
positive necula
servit
positive servit


Závislost velikosti odchylky na počtu řádků
velikost odchylky/100 [%]
počet řádků [-]
0
1
2
3
4
5
6
7
0
20
40
60
80
100
120
140
160
natural
contrib add
roth
positive roth
michalski
positive michalski
necula
positive necula
bowman
positive bowman
servit
positive servit


Závislost velikosti odchylky na počtu řádků
velikost odchylky/100 [%]
počet řádků [-]
0
0,05
0,1
0,15
0,2
0,25
0,3
0,35
0,4
0,45
0
20
40
60
80
100
120
140
160
natural
contrib add
roth
positive roth
necula
bowman
positive bowman
servit
positive servit


Závislost velikosti odchylky na počtu řádků (150)
velikost odchylky/100 [%]
počet řádků [-]
0
1
2
3
4
5
6
7
8
0
20
40
60
80
100
120
140
160
natural
contrib add
roth
positive roth
michalski
positive michalski
necula
positive necula
bow man
positive bow man
servit
positive servit


Závislost velikosti odchylky na počtu řádků (150)
velikost odchylky/100 [%]
počet řádků [-]
0
0,1
0,2
0,3
0,4
0,5
0,6
0,7
0
20
40
60
80
100
120
140
160
natural
contrib add
roth
positive roth
necula
bowman
positive bowman
servit
positive servit

8.1.1.2 Zhodnocení heuristik - závislost velikosti odchylky nalezeného řešení

Při hodnocení odchylky v závislosti na hustotě matice stojí za povšimnutí pozitivní Neculova heuristika. A sice tato heuristika jako jediná s větší hustotou matice viditelně zmenšuje svou chybu. Pro matice o hustotě 5% se může odchýlit od minimální ceny až o neuvěřitelných 450%. Naopak s rostoucí hustotou se stává přesnější. Již při hustotě matice 25% dosahuje dobrých výsledků a její odchylka se pohybuje v celkem přijatelných mezích.

Vůbec nejhůře z celkového pohledu dopadla pozitivní Michalskeho heuristika a její normální verze. Odchylka těchto dvou heuristik se pohybuje vysoce nad 100% ve všech možných případech. Zklamáním se rozněž ukázala Bowmanovu heuristika a její pozitivní verze. Její odchylka od minimálního řešení není tak vysoká jako v případě heuristiky Michalského, avšak pořád je viditelně horší než zbylé zkoumané heuristiky. Když k tomu navíc připočteme její časovou náročnost, která se blíží k náročnosti heuristiky Servitovy, můžeme tuto heuristiku pro danou instanci problémů označit jako vyloženě špatnou.

Heuristiky Natural a Contrib Add mají odchylku téměř shodnou a v grafech se tyto heuristiky překrývají. Nejlépe je lze popsat jako průměrné. Ani špatné, ani dobré

Hlavním kritériem pro přesnost heuristik se ukázala právě hustota matice. Nejlépe dopadla ve všech ohledech (co se týče nalezení nejnižší ceny) Servitova heuristika a i její pozitivní verze. Jejich odchylka od minimální ceny nestoupne nad 6%. Průměrně se pohybuje okolo 2%. Pozitivní verze je o něco málo horší co se týče nalezení minimální ceny. Pokud ovšem přihlédneme k časové náročnosti výpočtu těchto dvou heuritik vychází nám pozitivní servitova heuristika jako ta nejlepší (pro dané instance problému).

Velkým překvapením se ukázala Rothova heuristika. Co do kvality nalezeného řešení se drží pod 20% odchylkou. Přihlédneme-li opět k časové náročnosti heuristiky, která patří k těm nejnižším z pokročilých heuristik, a k velmi jednoduché cenové funkci, ukazuje se, že i takto jednoduchá heuristika může být velmi efektivní.

8.1.2 Měření na reálních instancích problému pokrytí (matice vygenerované programem boom)

		natural heuristic				contrib add heuristic			
matrix	density	time	mincost	maxcost	average	time	mincost	maxcost	average
40x43	0,099419	0,02	0,1	0,2	0,13549	0,02	0,1	0,2	0,1329
46x34	0,053069	0,01	0,01785	0,0982	0,06607	0,02	0,01785	0,09821	0,0776
62x100	0,044032	0,01	0,0324	0,1298	0,09090	0,01	0,03246	0,1298	0,0714
81x149	0,031071	0,2303	0,00369	0,0073	0,00457	0,27	0,00369	0,00738	0,0047
83x98	0,040939	0,1102	0	0,0895	0,0325	0,13	0	0,09452	0,0256
99x156	0,038267	0,01	0,0560	0,1074	0,08691	0,01	0,05607	0,10747	0,0878
101x94	0,062987	0,1001	0,1481	0,2037	0,18018	0,14	0,14814	0,20370	0,1818
108x241	0,023244	0,03	0,0209	0,0524	0,0370	0,04	0,01837	0,05249	0,0398
114x160	0,030811	0,3104	0,0766	0,1653	0,1328	0,39	0,07258	0,16935	0,1285
127x203	0,023506	0,3305	0,0707	0,1356	0,09050	0,43	0,0707	0,13569	0,09194
139x241	0,019493	1,1717	0	0,0276	0,01592	1,46	0	0,02766	0,0165
157x256	0,017292	0,0501	0,0770	0,1001	0,08092	0,08	0,07707	0,08670	0,0813
160x56	0,075223	0,0801	0,0508	0,3220	0,20186	0,12	0,05084	0,32203	0,1949
201x108	0,01709	0,0401	0,0405	0,0598	0,04700	0,04	0,04059	0,07264	0,0532
238x153	0,023453	0,03	0,0202	0,0405	0,03310	0,04	0,0202	0,0405	0,0351
281x98	0,042814	0,4106	0,0205	0,1301	0,06787	0,57	0,02054	0,10273	0,0705
401x149	0,028235	1,1116	0,0560	0,168	0,12411	1,44	0,06542	0,18224	0,1280
606x203	0,022127	2,1731	0,0225	0,1616	0,09582	2,81	0,0338	0,17669	0,1023
621x204	0,047701	0,1102	0,0862	0,1551	0,11379	0,17	0,06896	0,15517	0,1086
2252x100	0,050941	0,1302	0,0422	0,1126	0,06338	0,26	0,01408	0,09859	0,0450
1096x1495	0,003316	303,49	0,0532	0,0618	0,05765	420	0,04726	0,06339	0,0554
2628x160	0,033164	16,524	0,0492	0,1901	0,09852	25,4	0,04929	0,16197	0,0964
4822x485	0,008495	58,644	0,0277	0,0711	0,05503	79,6	0,04861	0,08333	0,0598

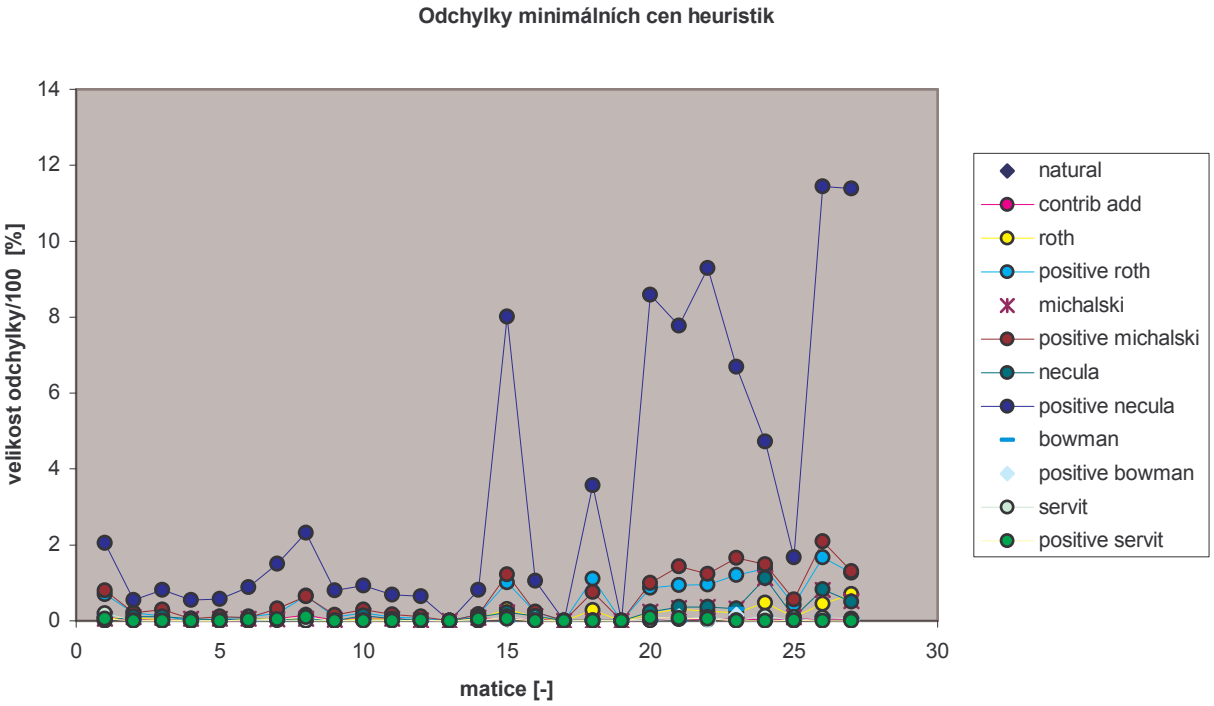
		roth heuristic				positive roth heuristic			
matrix	density	time	mincost	maxcost	averagecost	time	mincost	maxcost	average
40x43	0,099419	0,06	0,05	0,5	0,26450005	0,05	0,7	1,35	1,0680
46x34	0,053069	0,01	0,08035	0,16071429	0,11875003	0,01	0,205357143	0,29	0,2517
62x100	0,044032	0,01	0,03246	0,06493506	0,05844156	0,01	0,12987013	0,23	0,1720
81x149	0,031071	0,14	0	0,05535055	0,02734319	0,11	0,05904059	0,16	0,1067
83x98	0,040939	0,16	0,05970	0,08955224	0,07995029	0,1	0,109452736	0,18	0,1430
99x156	0,038267	0,01	0,04672	0,09345794	0,07009346	0,01	0,191588785	0,28	0,2238
101x94	0,062987	0,35	0,01851	0,07407407	0,03555552	0,19	0,666666667	0,94	0,8314
108x241	0,023244	0,01	0	0,01574803	0,00944883	0,02	0,097112861	0,16	0,1233
114x160	0,030811	0,39	0,05241	0,13709677	0,08374996	0,23	0,217741935	0,33	0,2785
127x203	0,023506	0,17	0,04424	0,12094395	0,07303838	0,1	0,100294985	0,18	0,1403
139x241	0,019493	0,57	0,00988	0,03754941	0,02023713	0,42	0,071146245	0,16	0,1098
157x256	0,017292	0,03	0,05394	0,0655106	0,06435457	0,04	0,175337187	0,21	0,194
160x56	0,075223	0,88	0,30508	0,44067797	0,37559328	0,27	1,016949153	1,51	1,2428
201x108	0,01709	0,05	0,1004	0,10042735	0,10042735	0,03	0,222222222	0,24	0,2324
238x153	0,023453	0,11	0,28378	0,31756757	0,29594597	0,08	1,108108108	1,17	1,1371
281x98	0,042814	3,53	0,19863	0,26712329	0,23520545	1,24	0,876712329	1,08	0,9914
401x149	0,028235	12,1	0,31308	0,35981308	0,33476642	3,17	0,943925234	1,11	1,023
606x203	0,022127	21,7	0,25187	0,28571429	0,25436092	4,7	0,958646617	1,11	1,0330
621x204	0,047701	2,53	0,20689	0,20689655	0,20689655	0,23	1,206896552	1,4	1,3103
2252x100	0,050941	21,2	0,47887	0,53521127	0,50704225	0,32	1,38028169	1,56	1,4619
1096x1495	0,003316	146	0,09602	0,11177794	0,10453862	173	0,419729932	0,44	0,430
2628x160	0,033164	959	0,45070	0,54225352	0,49338026	111	1,669014085	1,92	1,824
4822x485	0,008495	3286	0,70833	0,734375	0,72222222	276	1,265625	1,31	1,288

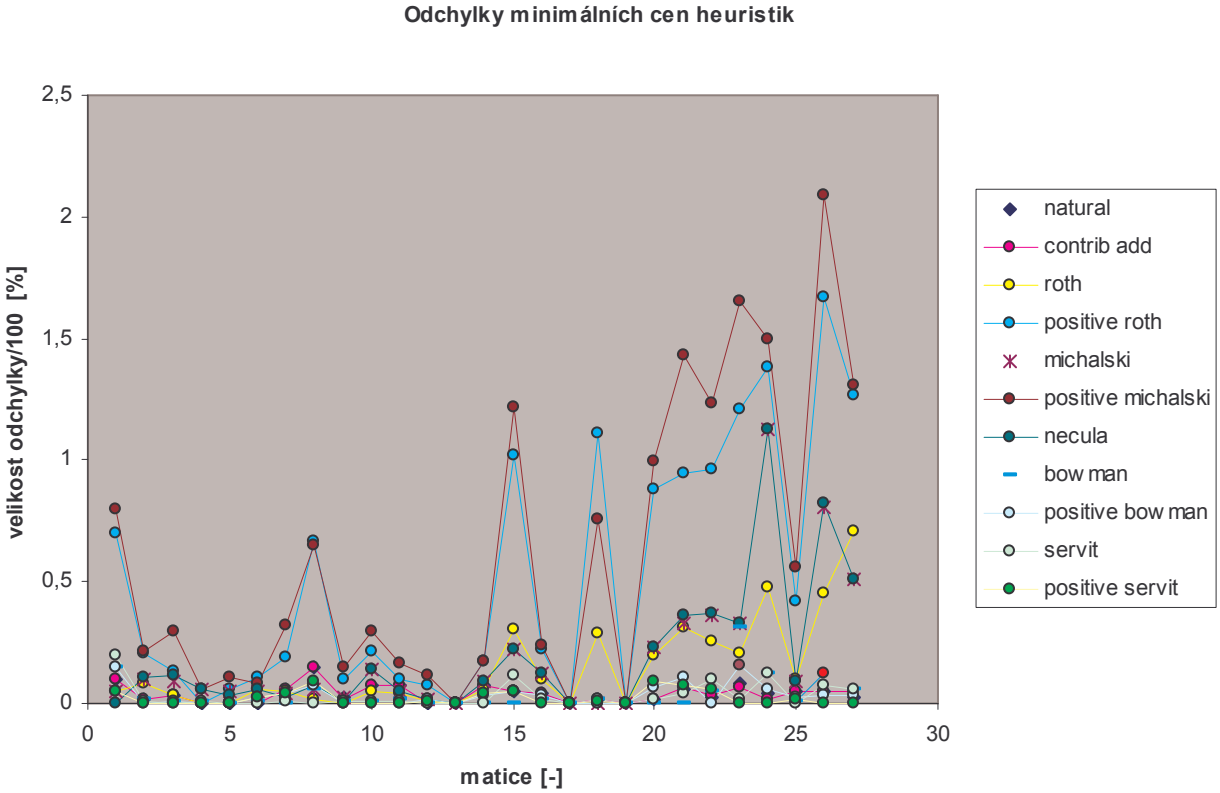
		michalski heuristic				positive michalski heuristic			
matrix	density	time	mincost	maxcost	averagecost	time	mincost	maxcost	averagecost
40x43	0,099419	0,06	0,05	0,7	0,35550003	0,05	0,8	1,5	1,06749992
46x34	0,053069	0,01	0,1	0,14	0,12321431	0,01	0,21	0,29	0,24910709
62x100	0,044032	0,01	0,09	0,16	0,11883119	0,01	0,29	0,38	0,3331169
81x149	0,031071	0,13	0,05	0,07	0,05594097	0,11	0,11	0,21	0,14966789
83x98	0,040939	0,15	0,05	0,1	0,08696515	0,11	0,08	0,26	0,14268653
99x156	0,038267	0,01	0,04	0,14	0,10140185	0,02	0,32	0,44	0,38738315
101x94	0,062987	0,43	0,06	0,2	0,11296293	0,18	0,65	0,85	0,73962967
108x241	0,023244	0,01	0,02	0,05	0,03648292	0,02	0,15	0,23	0,19317587
114x160	0,030811	0,37	0,14	0,21	0,16697582	0,28	0,3	0,41	0,34919357
127x203	0,023506	0,17	0,05	0,13	0,0878466	0,12	0,16	0,24	0,20917405
139x241	0,019493	0,5	0,02	0,06	0,03462454	0,44	0,11	0,23	0,16642297
157x256	0,017292	0,03	0,09	0,13	0,10308285	0,02	0,18	0,21	0,19865121
160x56	0,075223	0,9	0,22	0,29	0,22779665	0,25	1,22	1,73	1,49525426
201x108	0,01709	0,05	0,12	0,12	0,12222225	0,03	0,24	0,28	0,26794877
238x153	0,023453	0,13	0	0,01	0,0040541	0,07	0,76	0,84	0,79999996
281x98	0,042814	3,7	0,23	0,34	0,29020545	1,17	0,99	1,27	1,14369871
401x149	0,028235	11,6	0,33	0,41	0,36471957	3,79	1,43	1,61	1,53663564
606x203	0,022127	22,4	0,36	0,39	0,37864662	10,9	1,24	1,36	1,30992482
621x204	0,047701	2,74	0,33	0,45	0,41034488	0,21	1,66	1,76	1,69137942
2252x100	0,050941	18,8	1,13	1,21	1,18450713	0,39	1,49	1,65	1,56338028
1096x1495	0,003316	149	0,09	0,11	0,09917476	159	0,56	0,58	0,56616651
2628x160	0,033164	845	0,81	1,07	0,90640839	64,6	2,09	2,39	2,24338026
4822x485	0,008495	2377	0,51	0,53	0,52118058	148	1,31	1,38	1,35243056

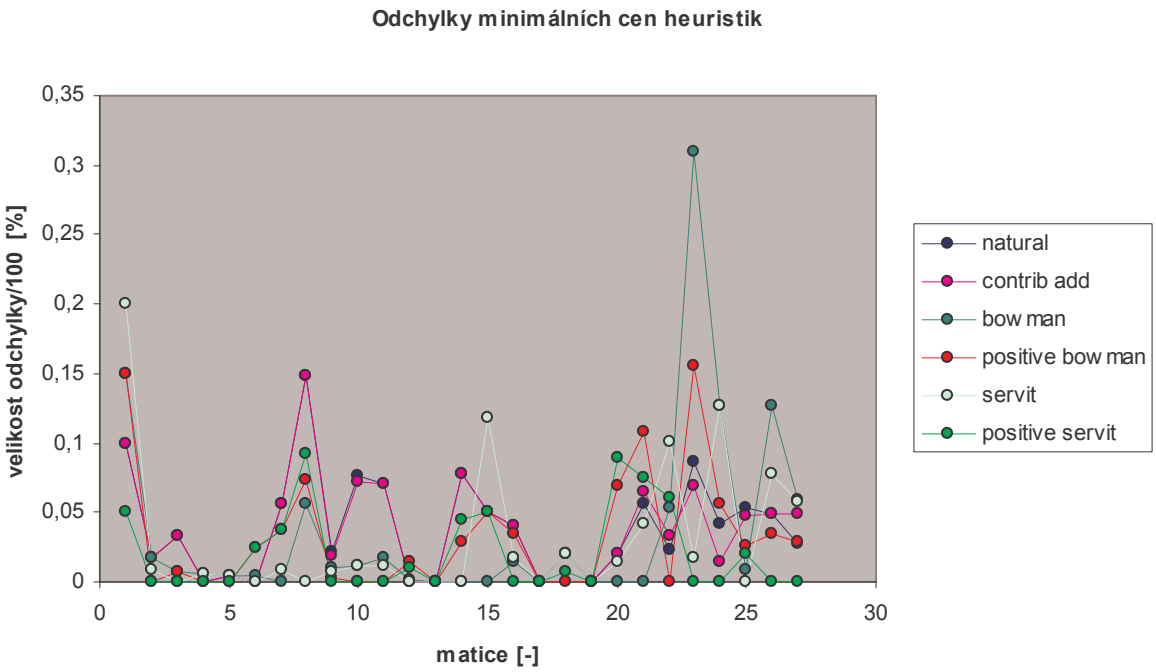
		necula heuristic				positive necula heuristic			
matrix	density	time	mincost	maxcost	averagecost	time	mincost	maxcost	averagecost
40x43	0,099419	0,06	0	0,5	0,30050001	0,06	2,05	3,35	2,65349998
46x34	0,053069	0,01	0,11	0,14	0,12857144	0,01	0,55	0,93	0,73035717
62x100	0,044032	0,01	0,12	0,16	0,12077926	0,01	0,82	0,92	0,87142865
81x149	0,031071	0,13	0,03	0,08	0,05531362	0,15	0,58	0,69	0,65867159
83x98	0,040939	0,15	0,05	0,09	0,08761194	0,16	0,89	1,2	1,07343288
99x156	0,038267	0,02	0,03	0,13	0,07943925	0,01	1,51	1,66	1,60700946
101x94	0,062987	0,47	0,07	0,15	0,09962965	0,3	2,31	2,81	2,58666653
108x241	0,023244	0,02	0,01	0,05	0,0346457	0,01	0,8	0,86	0,83595801
114x160	0,030811	0,33	0,14	0,2	0,16608872	0,37	0,93	1,15	1,03137096
127x203	0,023506	0,16	0,05	0,13	0,08749264	0,17	0,69	0,77	0,72474927
139x241	0,019493	0,48	0,02	0,06	0,03636368	0,56	0,64	0,76	0,70284588
157x256	0,017292	0,03	0,09	0,12	0,10385361	0,04	0,81	0,9	0,87263974
160x56	0,075223	0,96	0,22	0,31	0,22610163	0,54	8,02	8,97	8,5835592
201x108	0,01709	0,05	0,12	0,12	0,12200849	0,04	1,05	1,15	1,09871797
238x153	0,023453	0,13	0	0,01	0,00135133	0,09	3,57	3,78	3,67770262
281x98	0,042814	3,94	0,23	0,34	0,29424662	2,14	8,58	9,05	8,76390431
401x149	0,028235	12,2	0,36	0,44	0,36420562	6,21	7,77	8,21	8,02219626
606x203	0,022127	49,2	0,37	0,39	0,38116547	15,9	9,29	9,47	9,36725542
621x204	0,047701	2,72	0,33	0,43	0,38965515	0,32	6,69	6,93	6,85172403
2252x100	0,050941	32,4	1,13	1,21	1,16338037	0,97	4,72	4,94	4,80563397
1096x1495	0,003316	121	0,09	0,11	0,10116277	144	1,67	1,68	1,67273068
2628x160	0,033164	857	0,82	1,04	0,91922534	54,5	11,4	12	11,7162673
4822x485	0,008495	1962	0,51	0,53	0,51788192	148	11,4	11,4	11,4078123

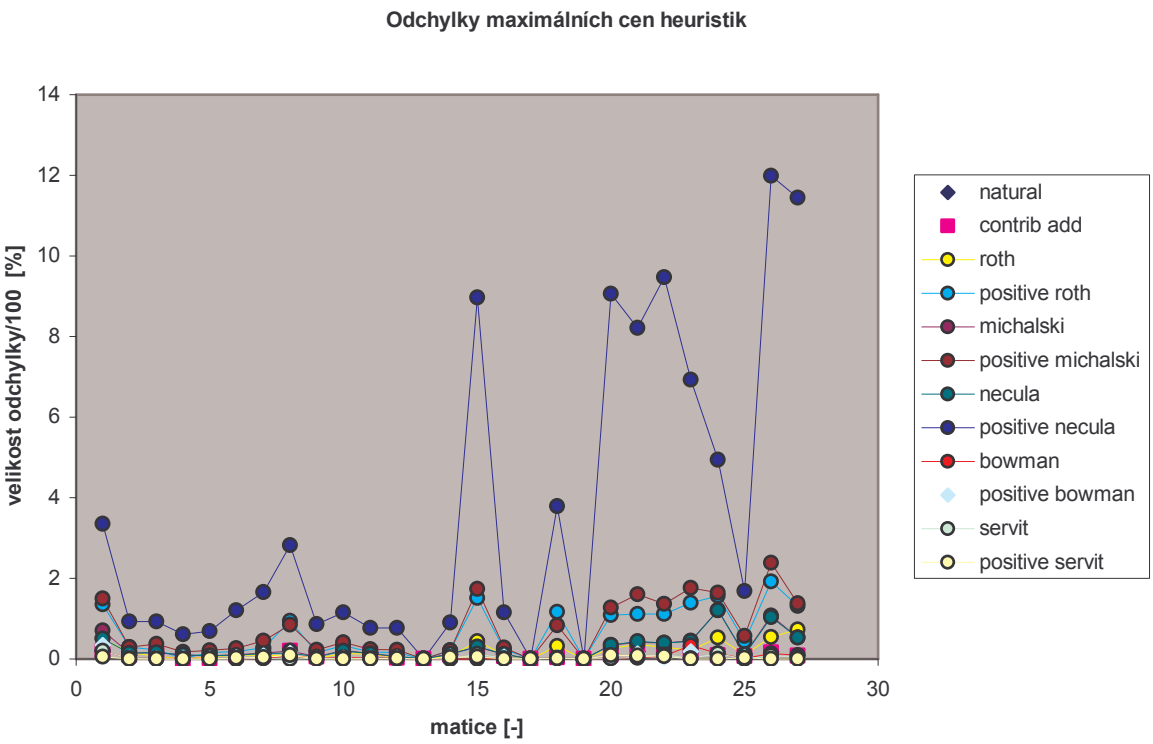
			bowman heuristic			positive bowman heuristic			
matrix	density	time	mincost	maxcost	averagecost	time	mincost	maxcost	averagecost
40x43	0,099419	0,200288	0,15	0,2	0,175	0,06	0,15	0,35	0,19700003
46x34	0,053069	0,0100144	0,02	0,02	0,01785714	0,01	0	0,03	0,01339286
62x100	0,044032	0,0100144	0,01	0,01	0,00974026	0,01	0,01	0,01	0,00844158
81x149	0,031071	0,30043125	0	0,01	0,00811813	0,13	0	0,01	0,00583021
83x98	0,040939	0,40057659	0	0	0,00497512	0,13	0,02	0,04	0,02741291
99x156	0,038267	0,100144	0	0,01	0,0046729	0,02	0,04	0,07	0,05747665
101x94	0,062987	2,20316887	0,06	0,07	0,05925927	0,44	0,07	0,15	0,11537036
108x241	0,023244	0	0,01	0,02	0,01312336	0,01	0	0,01	0,00708665
114x160	0,030811	1,30187273	0,01	0,04	0,01935485	0,38	0	0,02	0,00870969
127x203	0,023506	0,40057659	0,02	0,02	0,01887904	0,13	0	0,02	0,0118879
139x241	0,019493	1,30187035	0	0,01	0,00454543	0,63	0,01	0,03	0,02094857
157x256	0,017292	0,200288	0	0	0,00096339	0,04	0,03	0,05	0,03622348
160x56	0,075223	6,90993786	0	0	0	0,85	0,05	0,08	0,0594915
201x108	0,01709	0,50072014	0,01	0,02	0,01602564	0,08	0,03	0,05	0,04636755
238x153	0,023453	1,80259228	0,02	0,03	0,02364865	0,13	0	0,04	0,02027027
281x98	0,042814	41,15919113	0	0,01	0,0047945	4,82	0,07	0,1	0,07952055
401x149	0,028235	136,3961411	0	0,03	0,01869159	12	0,11	0,13	0,11742993
606x203	0,022127	1044,401855	0,05	0,08	0,06240604	47,3	0	0,04	0,02135336
621x204	0,047701	127,3831749	0,31	0,33	0,31896552	1,67	0,16	0,19	0,17931037
2252x100	0,050941	8992,330399	0,13	0,13	0,12676056	24,2	0,06	0,08	0,07746479
1096x1495	0,003316	2283,684082	0,01	0,01	0,01125281	280	0,03	0,03	0,02974495
2628x160	0,033164	78419,36279	0,13	0,13	0,12676056	1968	0,04	0,04	0,03591554
4822x485	0,008495	217413,418	0,06	0,09	0,07465278	2029	0,03	0,03	0,03229162

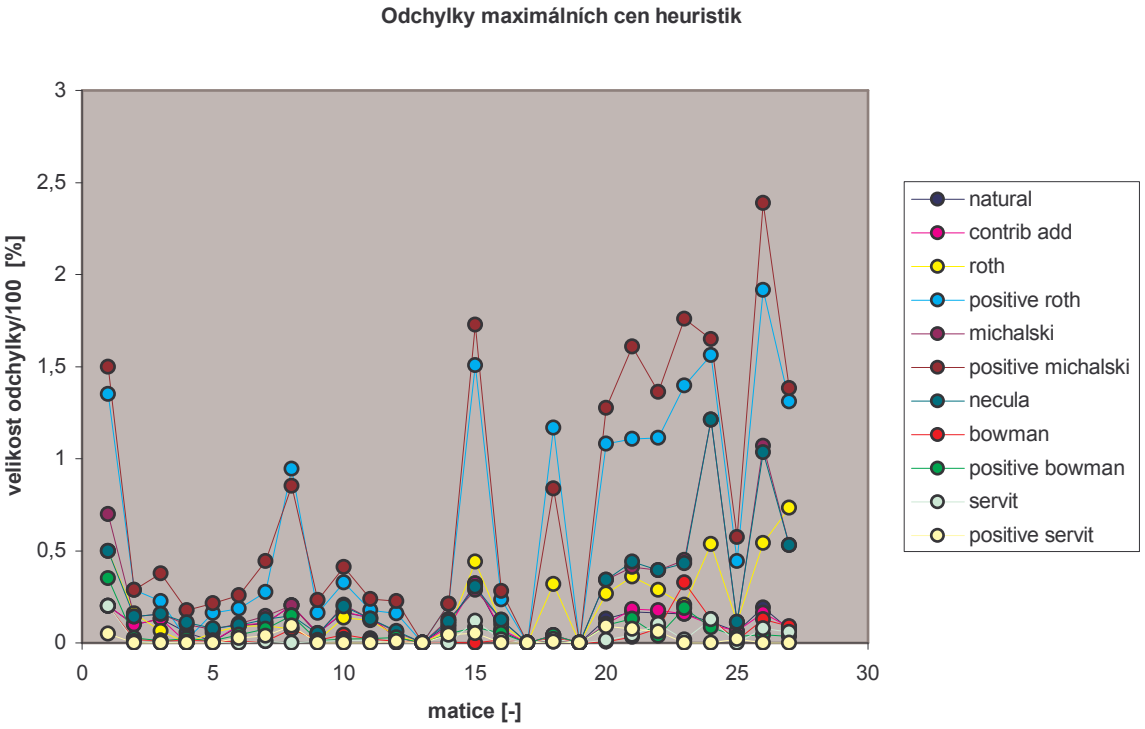
			servit heuristic			positive servit heuristic			
matrix	density	time	mincost	maxcost	everagecost	time	mincost	maxcost	everagecost
40x43	0,099419	0,30043185	0,2	0,2	0,2	0,06008643	0,05	0,05	0,05
46x34	0,053069	0,0100144	0,01	0,01	0,00892857	0,0100144	0	0	0
62x100	0,044032	0,0100144	0	0	0	0,0100144	0	0	0
81x149	0,031071	0,4005754	0	0	0,00369004	0,14020169	0	0	0
83x98	0,040939	0,50072074	0	0	0	0,13018715	0,02	0,02	0,024876
99x156	0,038267	0,100144	0,01	0,01	0,00934579	0,0300432	0,04	0,04	0,037383
101x94	0,062987	2,50360012	0	0	0	0,55079198	0,09	0,09	0,092593
108x241	0,023244	0,100144	0,01	0,01	0,00787402	0,0200288	0	0	0
114x160	0,030811	1,50216103	0,01	0,01	0,01209677	0,39056158	0	0	0
127x203	0,023506	0,50072074	0,01	0,01	0,01179941	0,14020157	0	0	0
139x241	0,019493	1,40201569	0	0	0	0,64092159	0,01	0,01	0,009881
157x256	0,017292	0,30043185	0	0	0	0,0400576	0,04	0,04	0,044316
160x56	0,075223	8,2118082	0,12	0,12	0,11864407	0,92132473	0,05	0,05	0,050847
201x108	0,01709	0,50072014	0,02	0,02	0,01709402	0,09012955	0	0	0
238x153	0,023453	2,20316887	0,02	0,02	0,02027027	0,14020157	0,01	0,01	0,006757
281x98	0,042814	43,06192398	0,01	0,01	0,01369863	5,03724289	0,09	0,09	0,089041
401x149	0,028235	145,3089905	0,04	0,04	0,04205607	11,65676117	0,07	0,07	0,074766
606x203	0,022127	656,043396	0,1	0,1	0,10150376	47,26794434	0,06	0,06	0,06015
621x204	0,047701	130,9883499	0,02	0,02	0,01724138	1,98285294	0	0	0
2252x100	0,050941	9375,181274	0,13	0,13	0,12676056	27,62976074	0	0	0
1096x1495	0,003316	2515,217285	0	0	0	285,8710938	0,02	0,02	0,020068
2628x160	0,033164	100702,6953	0,08	0,08	0,07746479	1639,958984	0	0	0
4822x485	0,008495	279239,3164	0,06	0,06	0,05729167	2199,652344	0	0	0

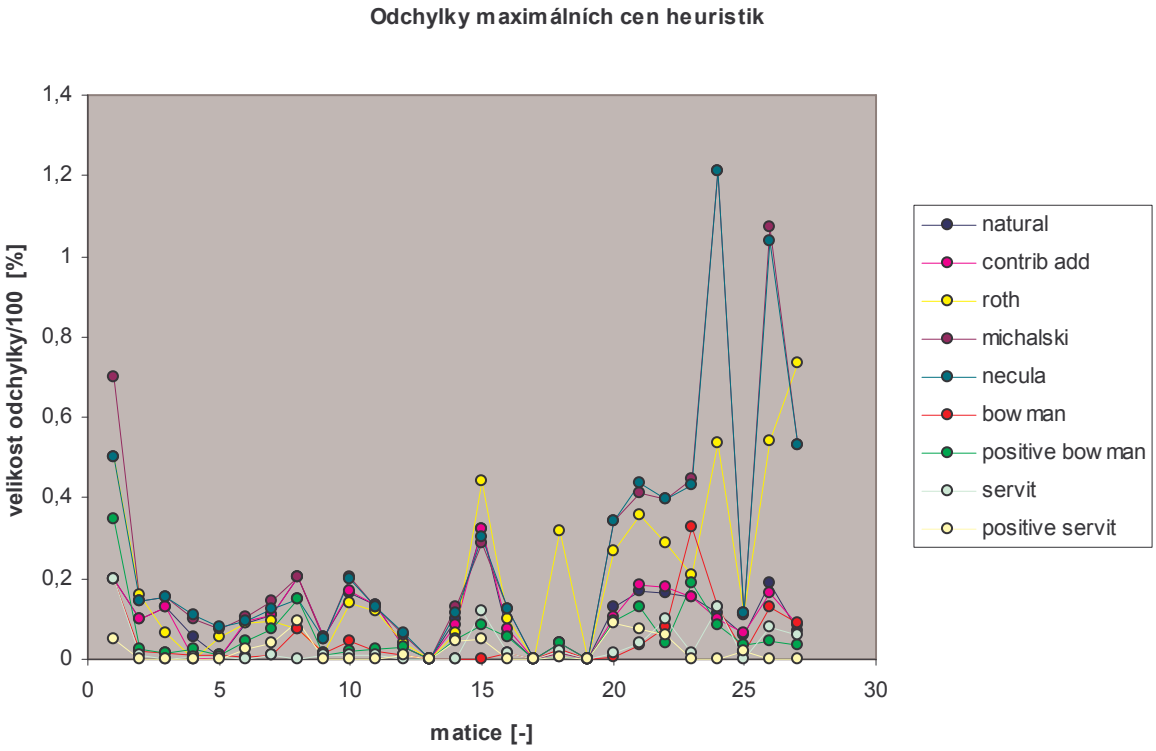

Odchylky minimálních cen heuristik
velikost odchylky/100 [%]
matice [-]
0
2
4
6
8
10
12
14
0
5
10
15
20
25
30
natural
contrib add
roth
positive roth
michalski
positive michalski
necula
positive necula
bowman
positive bowman
servit
positive servit

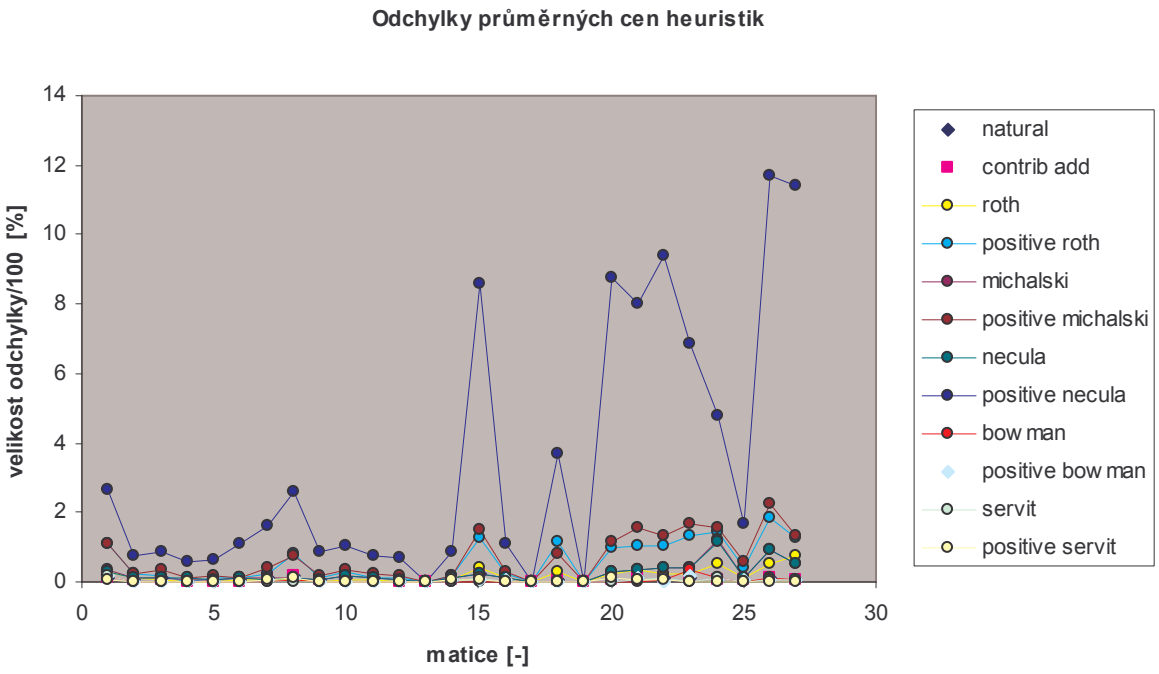

Odchylky minimálních cen heuristik
velikost odchylky/100 [%]
matice [-]
0
0,5
1
1,5
2
2,5
0
5
10
15
20
25
30
natural
contrib add
roth
positive roth
michalski
positive michalski
necula
bow man
positive bow man
servit
positive servit

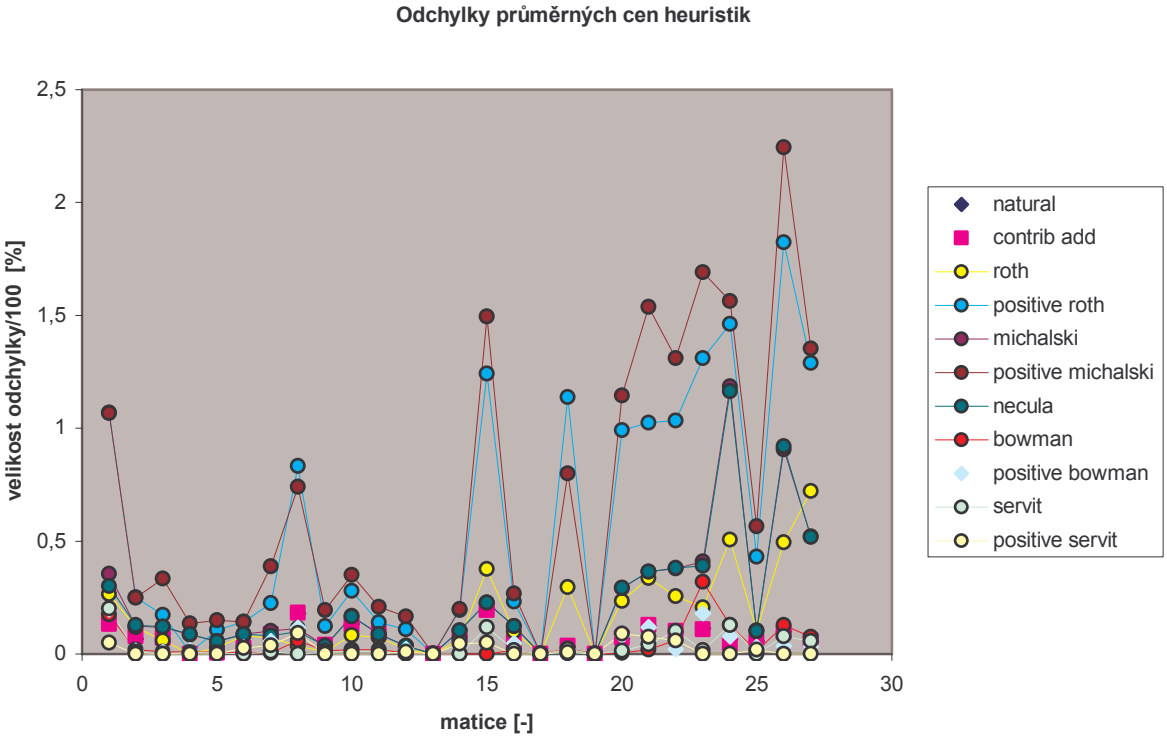

Odchylky minimálních cen heuristik
velikost odchylky/100 [%]
0,35
0,3
0,25
0,2
0,15
0,1
0,05
0
0
5
10
15
20
25
30
matice [-]
natural
contrib add
bow man
positive bow man
servit
positive servit

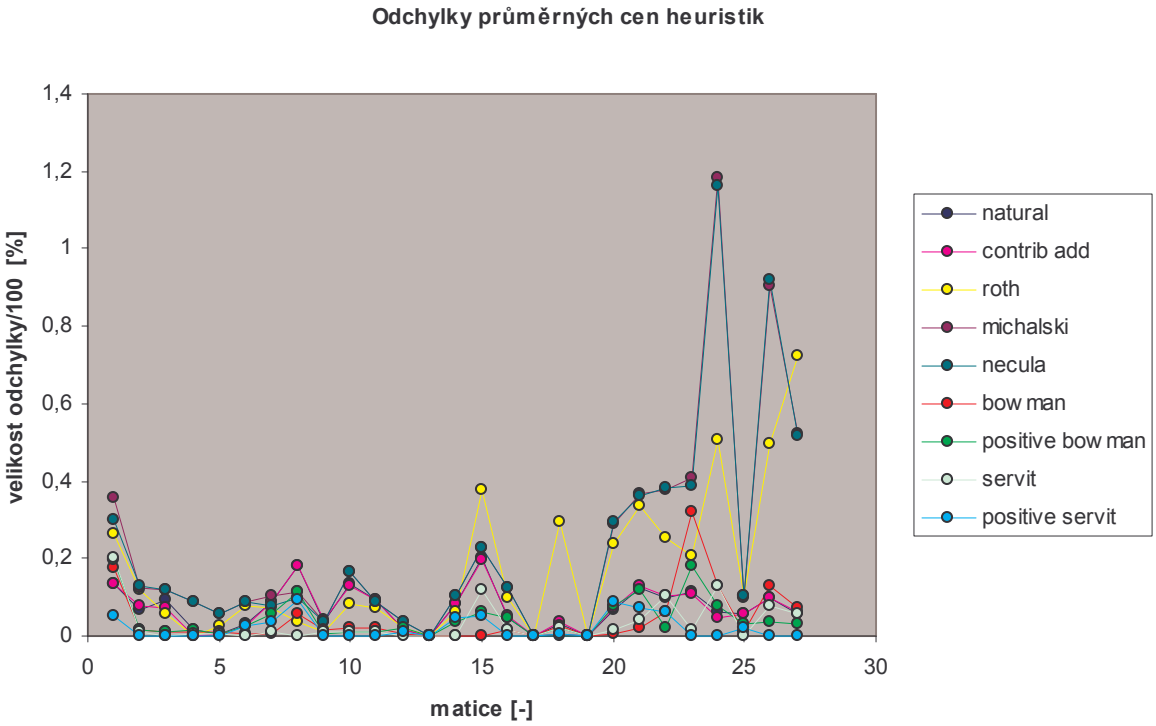

Odchylky maximálních cen heuristik
velikost odchylky/100 [%]
14
12
10
8
6
4
2
0
0
5
10
15
20
25
30
matice [-]
natural
contrib add
roth
positive roth
michalski
positive michalski
necula
positive necula
bowman
positive bowman
servit
positive servit


Odchylky maximálních cen heuristik
velikost odchylky/100 [%]
matice [-]
natural
contrib add
roth
positive roth
michalski
positive michalski
necula
bowman
positive bowman
servit
positive servit


Odchylky maximálních cen heuristik
velikost odchylky/100 [%]
matice [-]
natural
contrib add
roth
michalski
necula
bow man
positive bow man
servit
positive servit


Odchylky průměrných cen heuristik
velikost odchylky/100 [%]
matice [-]
natural
contrib add
roth
positive roth
michalski
positive michalski
necula
positive necula
bow man
positive bow man
servit
positive servit


Odchylky průměrných cen heuristik
velikost odchylky/100 [%]
matice [-]
natural
contrib add
roth
positive roth
michalski
positive michalski
necula
bowman
positive bowman
servit
positive servit



Z grafů je patrné, že pozitivní heuristiky při testování na reálných problémech nedopadly nejlépe vzhledem k ostatním heuristikám. Zejména pro velké matice se odchylka často pohybuje nad 50%. Nezklamala Servitova heuristika, ani její pozitivní verze. Odchylka obou těchto heuristik se drží pod 10%.

Rothova heuristika, která na náhodných maticích dosahovala dobrých výsledků, i při použití na velkých maticích často najde řešení s přijatelnou odchylkou. Přihlédneme-li k jednoduché cenové funkci podle níž vybírá sloupce a tím tedy nízkému času výpočtu, můžeme ji pro danou instanci problémů označit jako velmi efektivní.

Jako absolutně nejhorší můžeme označit pozitivní Neculovu heuristiku. Řešení jí nalezené může být až 10x větší než minimální.

8.1.3 Shrnutí heuristik

Z tabulek naměřených hodnot a grafů je patrné, že jak přirozená tak contrib add heuristika mají dobré výsledky , při velmi jednoduchém a rychlém algoritmu ohodnocování sloupců. Rychlost těchto dvou heuristik je dána vybíráním vždy nejlepšího sloupce do řešení. Pro další průchod algoritmu tak ubyde významné množství řádků. Toto je i důvod, proč mají pozitivní heuristiky o tolik lepší čas výpočtu.

Oproti tomu heuristiky, které odebírají vždy nejhorší sloupec jsou relativně pomalé, neboť při každém průchodu je odebrán vždy právě jeden sloupec (ten s nejhorší hodnotou) a žádný řádek. To se projeví zvláště pro velmi velké matice, kdy jak Servitova tak Bowmanova heuristika dosahují velmi špatného výpočetního času. Je to dáno právě přepočítáváním hodnot všech sloupců při každém průchodu a odebírání vždy jediného.

Vzhledem k tomu, že heuristiky jsou určeny především pro řešení velkých matic, neboť nalézt jejich řešení pomocí exaktních metod je časově velmi náročné, nám jako vítěz testování vychází pozitivní Servitova heuristika. Výpočetní čas této heuristiky je v porovnání s její normální variantou několikanásobně nižší a řešení jí nalezené se často blíží k tomu minimálnímu.

Ostatní pozitivní heuristiky mají sice čas výpočtu oproti svým normálním verzím velice nízký, avšak jejich odchylka od minimálního řešení je neúnosně vysoká.

8.2 Exaktní metody

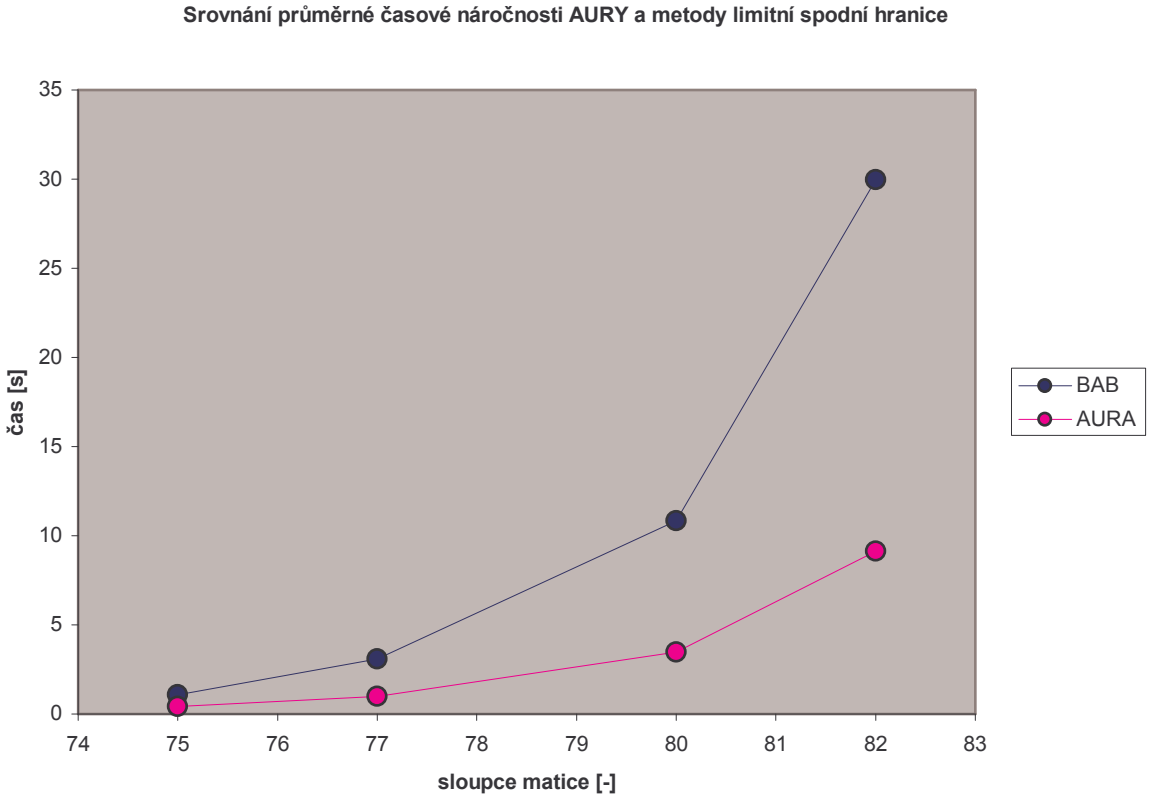
8.2.1 Srovnání AURY a algoritmu větví a hranic s limitní spodní hranicí

		BAB s limitní spodní hranicí			AURA ~ BAB s voláním Raiseru		
matice	hustota	čas CPU	volání	cena	čas CPU	volání	cena
75x75	0,050311	0,50072	1201	78	0,280403	96	78
75x75	0,0528	6,689619	9323	54	1,131627	546	54
75x75	0,052267	11,286228	66258	67	3,535084	2195	67
75x75	0,051022	1,772549	7648	65	1,742506	275	65
75x75	0,044978	0,030043	236	74	0,030043	16	74
75x75	0,050133	0,040058	367	70	0,060086	26	70
75x75	0,055467	0,130187	1386	49	0,070101	20	49
75x75	0,047822	1,101584	6359	69	0,290418	579	69
75x75	0,052622	0,050072	278	54	0,140202	25	54
75x75	0,051022	0,130187	1102	51	0,230331	25	51
75x75	0,053156	0,030043	358	60	0,040058	26	60
75x75	0,049778	0,050072	362	57	0,040058	24	57
75x75	0,058489	0,801152	2642	49	0,540778	222	49
75x75	0,052089	0,030043	332	59	0,010014	15	59
75x75	0,052089	0,030043	332	59	0,010014	15	59
75x75	0,046044	0,040058	228	58	0,060086	20	58
75x75	0,052622	0,871253	4279	57	0,450648	188	57
75x75	0,050667	0,260374	1511	51	0,270389	131	51
75x75	0,050667	0,260374	1511	51	0,270389	131	51
75x75	0,0528	0,420605	3182	57	0,100144	106	57
75x75	0,049778	0,030043	229	49	0,020029	15	49
75x75	0,049778	0,020029	229	49	0,020029	15	49
75x75	0,050667	1,271829	5927	59	0,490706	825	59
75x75	0,052267	1,392002	5988	55	0,480691	481	55

		BAB s limitní spodní hranicí			AURA ~ BAB s voláním Raiseru		
matice	hustota	čas CPU	volání	cena	čas CPU	volání	cena
77x77	0,051948	0,450648	1563	62	0,170245	287	62
77x77	0,055153	26,978793	135207	58	7,801216	10243	58
77x77	0,050093	0,34049	2497	58	0,200288	86	58
77x77	0,050936	2,773989	9479	64	2,143081	1326	64
77x77	0,048912	2,433499	6416	57	0,34049	409	57
77x77	0,051442	0,020029	167	55	0,010014	17	55
77x77	0,051105	1,75252	6184	64	0,570821	167	64
77x77	0,047226	0,100144	788	55	0,080115	35	55
77x77	0,047563	1,442074	7261	52	0,120173	85	52
77x77	0,049081	0,420605	2483	62	0,25036	101	62
77x77	0,050261	0,280403	964	59	0,130187	83	59
77x77	0,050261	0,280403	964	59	0,130187	83	59
77x77	0,052285	1,452088	4217	45	0,230331	183	45
77x77	0,046888	0,010014	20	65	0,010014	10	65
77x77	0,04672	7,430685	27463	70	2,824061	3500	70

		BAB s limitní spodní hranicí			AURA ~ BAB s voláním Raiseru		
matice	hustota	čas CPU	volání	cena	čas CPU	volání	cena
80x80	0,048594	0,460662	1992	66	0,120173	142	66
80x80	0,052344	149,434881	222134	61	38,214951	23744	61
80x80	0,050625	5,818367	9306	49	0,801152	287	49
80x80	0,047812	68,228105	78595	55	12,958633	5452	55
80x80	0,050469	4,216062	9143	61	2,573701	321	61
80x80	0,050469	2,804032	9696	55	1,402016	289	55
80x80	0,050625	2,042938	5488	56	1,221757	423	56
80x80	0,048281	0,741066	1508	69	0,59085	128	69
80x80	0,049531	5,638107	10243	62	2,193154	596	62
80x80	0,052656	19,728367	20905	57	7,28047	102	57
80x80	0,053438	2,673845	3804	48	1,09157	179	48
80x80	0,052344	0,240346	269	44	0,120173	16	44
80x80	0,052344	0,240346	269	44	0,110158	16	44
80x80	0,045	14,651067	31829	75	4,376293	1759	75
80x80	0,051875	1,932779	5114	58	1,34193	218	58
80x80	0,05125	1,081555	1874	50	0,460662	72	50
80x80	0,050937	2,683859	4282	57	1,772549	193	57
80x80	0,049063	6,819806	7167	62	2,834075	630	62
80x80	0,046875	0,550792	1513	59	0,821181	54	59
80x80	0,047031	5,798338	10994	63	1,782563	693	63
80x80	0,05	0,470677	1493	59	0,16023	62	59
80x80	0,047031	0,630907	1513	59	0,490706	191	59
80x80	0,048437	3,615198	6886	59	2,00288	342	59
80x80	0,052969	2,633787	3662	52	0,620893	22	52
80x80	0,05125	0,540778	1303	70	0,110158	41	70
80x80	0,046563	1,271829	3184	64	2,233211	230	64
80x80	0,052969	1,972837	4447	49	1,031483	99	49
80x80	0,048906	3,34481	6595	62	0,821181	42	62
80x80	0,050937	17,965833	26433	53	8,512241	1931	53
80x80	0,048906	12,958634	22972	68	5,047258	2144	68
80x80	0,05125	14,981542	21025	54	3,43494	866	54
80x80	0,050937	1,462102	4568	56	1,261814	283	56
80x80	0,048906	1,021469	1598	58	0,330475	129	58
80x80	0,048437	0,060086	113	60	0,060086	16	60
80x80	0,048437	0,060086	113	60	0,060086	16	60
80x80	0,050937	19,778441	28612	56	7,020094	984	56
80x80	0,052813	41,459615	78518	56	11,997253	6218	56
80x80	0,045625	0,220317	713	60	0,25036	61	60
80x80	0,045625	0,230331	713	60	0,25036	61	60
80x80	0,050781	0,110158	319	59	0,210302	32	59
80x80	0,050156	3,675285	5264	58	0,66095	242	58
80x80	0,053906	0,230331	476	39	0,400576	23	39
80x80	0,053906	0,230331	476	39	0,380547	23	39
80x80	0,052656	37,654146	58326	56	22,001637	9186	56
80x80	0,046563	23,60394	31372	61	4,546537	1820	61
80x80	0,054844	23,974474	20915	42	7,210367	361	42
80x80	0,053438	0,060086	201	53	0,040058	12	53

		BAB s limitní spodní hranicí			AURA ~ BAB s voláním Raiseru		
matice	hustota	čas CPU	volání	cena	čas CPU	volání	cena
82x82	0,047442	2,904176	9933	55	1,592289	121	55
82x82	0,049524	11,706833	30733	53	3,835516	1681	53
82x82	0,050119	7,000066	20188	54	3,124493	452	54
82x82	0,048929	14,891413	32402	55	6,198913	419	55
82x82	0,048929	1,902736	2805	58	1,09157	224	58
82x82	0,04878	50,392461	96458	64	13,699696	6022	64
82x82	0,051606	4,816926	13952	57	2,403456	288	57
82x82	0,049822	71,022122	69183	52	20,259132	3596	52
82x82	0,052201	32,306453	49642	62	17,264828	3747	62
82x82	0,053837	92,763386	126818	55	18,827072	13037	55
82x82	0,051904	6,839835	8973	47	1,672405	139	47
82x82	0,050119	8,311952	10613	56	2,273269	477	56
82x82	0,049375	130,667894	267875	74	25,386505	19935	74
82x82	0,051309	0,891282	1910	44	0,510734	82	44
82x82	0,050416	111,219926	141700	46	31,475266	2998	46
82x82	0,052796	7,931405	16283	46	3,204608	112	46
82x82	0,054283	66,495614	44810	46	23,784203	2426	46
82x82	0,04878	1,34193	3177	58	1,00144	198	58
82x82	0,050714	23,53384	26020	54	6,729677	1655	54
82x82	0,053688	4,59661	8679	53	1,301872	316	53
82x82	0,049524	0,16023	501	53	0,050072	30	53
82x82	0,049375	15,502291	24556	60	7,120238	2677	60
82x82	0,047739	0,25036	540	51	0,630907	31	51
82x82	0,048483	21,941551	27418	49	6,48933	1275	49
82x82	0,049375	22,231968	40613	70	5,227516	1711	70
82x82	0,049375	6,699634	21530	55	2,423485	506	55
82x82	0,048037	1,632347	2512	54	1,291858	93	54
82x82	0,053688	427,074113	509093	56	126,722198	43713	56
82x82	0,051011	9,253306	9723	50	2,072981	361	50
82x82	0,046698	0,150216	273	58	0,070101	31	58
82x82	0,046698	0,140202	273	58	0,080115	31	58
82x82	0,050268	7,17031	13068	58	3,985731	405	58
82x82	0,049673	29,532466	46197	60	13,429308	4628	60
82x82	0,050714	0,390562	806	49	0,040058	23	49
82x82	0,050714	0,390562	806	49	0,040058	23	49
82x82	0,048037	1,432059	2323	62	0,771109	80	62
82x82	0,050714	1,552232	2071	56	0,350504	177	56
82x82	0,05354	111,830804	95238	44	44,073372	4185	44
82x82	0,050119	3,474997	7830	55	0,430619	237	55
82x82	0,054729	16,934351	41488	51	8,812672	2027	51
82x82	0,049227	9,163176	10205	60	2,203168	634	60
82x82	0,049078	1,652376	1752	48	0,430619	58	48
82x82	0,050714	5,838395	7960	61	1,822621	405	61
82x82	0,046401	1,992866	1811	46	2,874133	93	46
82x82	0,053688	58,434022	86477	51	12,397827	2896	51
82x82	0,048929	26,818564	40001	70	6,489332	1764	70
82x82	0,049078	1,09157	2716	62	0,881267	150	62
82x82	0,051904	4,035803	6547	46	1,291858	362	46



Při testování Aury na náhodně generovaných maticích bylo možné pozorovat významné snížení času potřebného k nalezení minimálního řešení. Snížení počtu volání rekurzivního volání algoritmu je dosaženo právě lepším ořezáváním zkoumaného prostoru díky metodě Raiser.

8.2.2 Srovnání AURY a algoritmu větví a hranic s limitní spodní hranice na reálných maticích

		BAB s limitní spodní hranicí			AURA ~ BAB s voláním Raiseru		
matice	hustota	čas CPU	volání	cena	čas CPU	volání	cena
45 x 34	0,054248	0,01	112	71	0,01	112	71
200x203	0,023227	0,16	220	325	0,21	71	325
124 x 153	0,022085	26,05	136 339	192	4,59	2 748	192
245 x 282	0,011029	86,7	45 415	517	29,34	4 175	517
120 x 105	0,045079	3,45	11 782	72	1,05	262	72
122 x 160	0,031352	0,1	161	258	0,11	48	258
260 x 27	0,209972	5,84	9 029	12	3,5	18	12
113x149	0,032191	0,02	38	227	0,02	20	227
45 x 34	0,054248	0,1	71	112	0,1	20	112
39 x 100	0,067436	0,1	1	104	0,1	1	104
39 x 108	0,057455	0,1	1	147	0,1	1	147
34 x 34	0,020029	0,1		170	0,1	1	170
106 x 156	0,030539	0,08	136	241	0,08	45	241
63 x 98	0,019436	0,1	1	407	0,1	1	407
106 x 108	0,010395	0,1	1	682	0,1	1	682
82 x 204	0,048541	0,1	88	72	0,1	88	12
193x149	0,03081	394.91	89262	218	119,47	8 723	218
315x217	0,083658	427,0713	489063	164	126,722198	39713	164
132 x 149	0,027608	0,1	90	273	0,1	19	273
248 x 297	0,115944	0,1	1	50	0,1	1	50

Z údajů je patrné, že metody založené na "negativním" myšlení, jmenovitě AURA dosahují pozoruhodných výsledků oproti metodám založených na "pozitivním" myšlení.

Je vidět, že časový rozdíl mezi výsledky je znatelný. To je způsobeno exponenciální složitostí problému. V některých případech však došlo k vyřešení problému pomocí dominancí (byly do řešení vybrány nezbytné sloupce), cyklické jádro bylo prázdné. U takových matic je ve sloupci počet volání je uvedena 1. Některé matice nebylo možné otestovat exaktními metodami z důvodů časové náročnosti. Sloupec Limitní spodní hranice znázorňuje výsledky získané pomocí metody větví a hranic vylepšenou o vypočítávání limitní spodní hranice, sloupec AURA znázorňuje výsledky získané pomocí metody větví a hranic vylepšenou o proceduru Raiser.

V naprosté většině případů byl zaznamenán pokles výpočetního času významným způsobem. Tento pokles dosahuje až hodnoty 83% (v nejlepším případě) . V případě velmi malých matic můžeme pozorovat pro AURU mírně delší výpočetní dobu. Ta je způsobena inicializací všech struktur potřebných pro metodu Raiser a vytváření krychlí.

9. Závěr

Byly otestovány matice nejdříve heuristikami a poté exaktními metodami.

Základním stavebním kamenem exaktních metod je algoritmus větví a hranic, který je dále vylepšován. Původní algoritmus, který je skoro nepoužitelný pro hledání minimálního řešení z důvodu extrémní časové náročnosti se tak po aplikovaných úpravách stává velmi efektivním. Při implementaci procedury Raiser (založené na "negativním" myšlení) dosahuje o mnoho lepších výsledků než s použitím metody limitní spodní hranice.

Použití heuristik je žádoucí pro velké matice, pro něž trvají exaktní metody neúnosně dlouho. Výpočetní čas je zkrácen a často je dosažené řešení velmi blízko minimálnímu. Zejména heuristické metody Contrib add heuristic, Roth heuristic a Servitova heuristika zaznamenaly nejlepší výsledky. Contrib add heuristika proto, že v opravdu velmi krátkém čase nalézá poměrně dobré řešení. Pro větší matice již není ovšem tak přesná. Servitova heuristika zase nalézá stabilně stejně kvalitní řešení, ovšem pro velmi velké matice již spotřebuji více výpočetního času. Rothova heuristika nabízí kvalitní řešení v přijatelném čase.

Přesných výsledů tedy dosáhneme pouze s použitím exaktních metod, ovšem u hodně velkých problémů jsou časové nároky velmi velké. U velkých vstupních matic je tedy jediná přijatelná možnost řešení pomocí heuristik, za cenu ne vždy minimálního řešení.

Výsledkem této práce jsou tak naměřené hodnoty výpočetního času , které potřebují jednotlivé metody . Z těchto hodnot plynou výhody a nevýhody jednotlivých metod. U heuristik bylo navíc provedeno podrobné srovnání kvality nalezeného řešení spočtením odchylky od nejnižšího nalezeného řešení.

Celkem bylo provedeno přes 1 600 000 testů. Soubor naměřených hodnot je umístěn na cd příloze.

10. Použitá literatura

[1] Lukáš Krejčík - bakalářská práce Moderní metody řešení problému

[2] E.I. Goldberg, L.P.Carolni, T.Villa, R.K.Brayton, A.L.Sangiovanni-Vincenrelli. - Negative Thinking by Incremental Problem Solving: Application to Unate Covering

[3] E.I. Goldberg, L.P.Carolni, T.Villa, R.K.Brayton, A.L.Sangiovanni-Vincenrelli - Negative Thinking in Branch-and-Bound: The Case of Unate Covering

[4] M.Servit. - A heuristic method for solving weighted set covering problems

[5]L.P.Carloni.- Negative thinking in search problems

[6]L.P.Carloni - AURA : A New Search Strategy for Unate Covering Problem

[7]Olivier Coudert - Two-level minimization:an overview