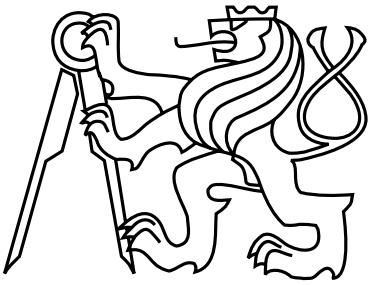
zde je zadani

České vysoké učení technické v Praze
Fakulta elektrotechnická



Diplomová práce

Řešení problému pokrytí

Bc. Filip Cakl

Vedoucí práce: Ing. Petr Fišer

Studijní program: Elektrotechnika a informatika, strukturovaný, Navazující magisterský

Obor: Výpočetní technika

květen 2010

Poděkování

Mé poděkování patří ing. Petru Fišerovi za odborné vedení, trpělivou pomoc a veškerý čas, který mi věnoval při zpracovávání této diplomové práce. Dále bych rád poděkoval rodině za podporu, bez které by pro mne psaní této práce bylo mnohem obtížnější.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady 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).

V Praze dne 1.12.2009 .....

Abstract

This thesis is focused on method of solving covering problem which uses Lagrangean relaxation. Covering problem occurs in several fields of computer science, logic synthesis and reliability analysis. This thesis contains design and implementation of algorithm, that solves covering problem by exploring the problem space with using branch and bound method. Heuristic used for branching is based on Lagrangean relaxation. Behavior of algorithm is studied and algorithm's efficiency is compared to AURAII. I have been given materials mentioned in the References. Final implementation of algorithm is integrated into program BOOM.

Abstrakt

Tato práce se zabývá řešením problému pokrytí pomocí Lagrangeovské relaxace. Problém pokrytí se objevuje v mnoha oblastech počítačového výzkumu, syntéze logických obvodů a analýze spolehlivosti. Tato práce obsahuje návrh a implementaci algoritmu, který řeší problém pokrytí prohledáváním stavového prostoru problému metodou větví a hranic. Pro ořezávání větví je využita heuristika založená na Lagrangeovské relaxaci. Studováno je chování této relaxace a rozdíly ve výkonnosti navrženého algoritmu proti algoritmu AURA II. Výsledný algoritmus je začleněn do programu BOOM.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

1.1 Obecný úvod

Problém pokrytí má použití v mnoha oblastech počítačového výzkumu. Je nutné jej řešit při syntéze logických obvodů, dvouúrovňové minimalizaci logických obvodů a plánování rozvržení zdrojů.

Řešení tohoto problému bývá obecně časově náročné – jedná se o NPO úplný problém. Byly navrženy mnohé algoritmy, které se liší v časových složitostech a kvalitě nalezených řešení.

Jedním ze způsobů, jak najít řešení, může být využití rekurzivního algoritmu, který zkouší smysluplné kombinace prvků řešení, dokud není nalezeno minimální pokrytí. Tento způsob řešení je ale časově velmi náročný. Snížení výpočetního času může být dosaženo například zamezením prohledávání prostoru v místech, kde se řešení nenachází [1].

Dále je možné využít genetických algoritmů nebo algoritmů simulovaného ochlazování. I tyto algoritmy jsou ale časově náročné [3].

Existují však i jiné možnosti - jednou z nich je použití heuristik, které dokážou najít řešení pokrytí v přípustném čase. Tento způsob je ovšem bývá vykoupen neoptimálností takto nalezeného řešení. Pro rozsáhlé problémy je to obvykle jediná možná varianta z důvodu časové náročnosti [6] [7].

Tato práce vychází z diplomové práce „Moderní metody řešení problému pokrytí z roku 2008 od Lukáš Krejčíka, v které byl zpracován algoritmus pro nalezení optimálního řešení AURA II [2].

Lagrangeovská relaxace spočívá v transformaci problému pokrytí na problém Lagrangeovský, kdy se omezující podmínky problému přemění do tvaru tzv. trestných koeficientů (multiplikátorů). Při hledání optimálního řešení se tyto multiplikátory projevují v závislosti na plnění nebo neplnění omezujících podmínek původního problému. Optimální řešení Lagrangeovského problému tvoří spodní mez původního problému, což je vlastnost, kterou algoritmus popsaný v této práci využívá.

1.2 Cíl práce

Cílem této práce je popsat Lagrangeovskou relaxaci a použít jí v algoritmu, který řeší problém pokrytí. Chování Lagrangeovské relaxace by mělo být prostudováno a popsáno.

Druhým cílem je zakomponování vytvořeného algoritmu do exaktního algoritmu založeném na metodě vetví a hranic, tak, aby sloužil pro určování spodní meze ceny řešení. Takto vytvořený řešič bude porovnán se stávajícími algoritmy řešící problém pokrytí (AURA II, heuristické metody).

Vytvořený řešič má být zakomponován do minimalizátoru BOOM.

1.3 Členění práce

V předchozích odstavcích byly představeny některé algoritmy pro řešení problému pokrytí a byly specifikovány cíle této práce.

V následující kapitole je přesně definován problém pokrytí a některé základní pojmy.

Třetí kapitola se zabývá existujícími způsoby řešení problému pokrytí - základní rekurzivní algoritmus s využitím horních a spodním mezí, heuristiky pro řešení problému pokrytí a nakonec principy Lagrangeovské relaxace.

V čtvrté kapitole jsou popsány vytvořené algoritmy. Nejdříve celkový řešič problému pokrytí. Poté samotný algoritmus založený na Lagrangeovské relaxaci.

Pátá kapitola obsahuje popis prováděného měření a jejich výsledky.

Šestá kapitola se zabývá začleněním vytvořených algoritmů do programu BOOM.

V poslední kapitole jsou shrnuty výsledky práce a nástin možností dalšího rozšíření této práce.

Kapitola 2

Problém pokrytí

V této kapitole jsou shrnuty základní pojmy, definice problému pokrytí a možnosti jeho řešení.

Definice 2.0.1 *Nechť $X = \{x_1, x_2, ..., x_m\}$ je množina řádků a $Y = \{y_1, y_2, ..., y_n\}$ je množina sloupců v matici A o rozměrech $m \times n$ a $CENA$ 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 $\langle X, Y, CENA \rangle$ spočívá v nalezení podmnožiny S množiny Y takové, že pro každý prvek x množiny X existuje prvek y množiny Y tak, že xRy, přičemž součet $CENA(S)$ je minimální.*

Jinými slovy: je dána matice A, jejíž všechny prvky jsou rovny 1 nebo 0. Řádek x_i je pokrytý sloupcem y_j pokud $A_{i,j} = 1$. Sloupce jsou ohodnoceny kladným reálným číslem, zvaným cena sloupce. Řešení problému pokrytí spočívá v nalezení podmnožiny sloupců matice A tak, že každý řádek matice A je pokrytý alespoň jedním sloupcem podmnožiny, přičemž souhrnná cena sloupců v dané podmnožině je minimální. Tuto souhrnná cena bude dále označována jako cena řešení.

2.1 Dominance

Při hledání řešení je možné pracovat pouze s *cyklickým jádrem* matice A, které se získá tzv. *odstraněním dominancí*. Tyto pojmy jsou objasněny na následujícím příkladu.

x_1			1	
x_2	1	1		1
x_3		1		1
x_4		1	1	1
	y_1	y_2	y_3	y_4

Tabulka 2.1: Příklad na dominance

2.1.1 Dominance řádků

Definice 2.1.1 *Nechť $\langle X, Y, CENA \rangle$ 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 sloupcích jedničky a jiný řádek x obsahuje jedničky ve stejných sloupcích, (x může obsahovat jedničky i na dalších sloupcích), tak x' dominuje x. Vždy, kdy je pokryt řádek x', bude pokryt i řádek x. Řádek x tedy může být vyřazen z matice.

V tabulce 2.1 je vidět, že řádek x_3 dominuje řádku x_4, protože řádek x_3 je pokryt sloupci y_2 a y_4 a oba tyto sloupce také pokrývají řádek x_4. Totéž platí i pro řádek x_2. To, že řádek x_3 dominuje řádkům x_2 a x_4, znamená, že výběrem sloupce pokrývající řádek x_3 dojde vždy k pokrytí řádků x_2 a x_4. Proto lze řádky x_2 a x_4 vyloučit z matice.

2.1.2 Dominance sloupců

Definice 2.1.2 *Nechť $\langle X, Y, CENA \rangle$ je problém pokrytí. Sloupec y' množiny všech sloupců Y dominuje sloupci y právě tehdy, když všechny prvky množiny řádků X, které jsou pokryty y jsou také pokryté y'.*

To znamená, že pokud nějaký sloupec y' obsahuje jedničky ve všech řádcích, ve kterých je obsahuje sloupec y (y' může obsahovat jedničky i na jiných řádcích), tak y' dominuje y.

V tabulce 2.1 je vidět, že sloupec y_4 dominuje sloupcům y_2 a y_1 a zároveň sloupec y_2 dominuje sloupcům y_4 a y_1. Nabízí se tedy možnost vyloučit z matice sloupce y_4 nebo y_2. Při vylučování sloupců se však musí dát pozor na to, aby se nepřišlo o minimální řešení. Vyloučení je možné jen tehdy, pokud cena dominujícího sloupce menší než cena sloupce, kterému sloupec dominuje. Nelze tedy vyloučit sloupec y_2, kterému dominuje sloupec y_4, protože jeho cena je menší než cena sloupce y_4.

2.1.3 Nezbytné sloupce

Sloupec y množiny sloupců Y je nezbytný, pokud je jediný, který pokrývá řádek x z množiny řádků X. Tyto sloupce musí vždy patřit do řešení, můžou být proto z matice vyřazeny a problém tak může být redukován. Minimální řešení původního problému lze získat přidáním těchto sloupců do minimálního řešení redukovaného problému.

Při řešení dominancí sloupců však v matici mohou vzniknout nové dominance řádků a naopak. Algoritmus řešící problém pokrytí tedy musí opakované řešit dominance, dokud se v matici nějaké vyskytují. Při nalezení nezbytných sloupců musí tyto zahrnout do řešení. Algoritmus na matici aplikujeme algoritmy popsané výše tak dlouho, dokud je matice redukována odstraňováním dominancí.

Kapitola 3

Řešení problému pokrytí

Pokud jsou na matici aplikovány výše uvedené techniky, matice se dostane do stavu, kdy ji tyto procesy již více nezmenší. Zbylá matice se nazývá *cyklické jádro matice*, $\langle X, Y, CENA \rangle$. Pokud je prázdné, množina všech nezbytných prvků byla nalezena během procesů popsaných výše.

3.1 Metoda větví a hranic

Algorithm 3.1.0.1 Branch and Bound

```
Branch_and_bound(A,Path) {
  /* odstraní dominantní řádky a sloupce,
  vybere do řešení nezbytné sloupce */
  A = solve_dominance(A);
  if ( A == ∅ ) {
    if ( cost(Path) < UB ) UB = cost(Path)
    bestSolution = Path
  } else {
    j = select_column(A);
    Path1= Path + j;
    A' = A - { j + rows_covered_ by(j) }
    Solution1 = Branch_And_Bound(A',Path1)
    A'' = A – {j}
    Solution2 = Branch_And_Bound(A'',Path2)
    bestSolution = min( Solution1 , Solution2 )
  }
  return bestSolution
}
```

Pokud je nalezeno cyklické jádro, je třeba v něm najít minimální řešení. Nalezení přesného řešení problému pokrytí lze dosáhnout pomocí rekurzivního algoritmu - metodou větví a hranic popsané v [1] spočívajícím ve výběru sloupců do řešení a hledáním smysluplné

kombinace sloupců tak, aby byly pokryty všechny řádky a zároveň cena řešení byla co nejmenší. Algoritmus je schopen vyloučit z řešení takové kombinace sloupců, jejichž cena je vyšší než cena zatím nejlepšího nalezeného řešení (cenu dosud nejlepšího nalezeného řešení označujeme jako horní mez – UB).

Stavový prostor algoritmu je reprezentován binárním stromem. Kořen stromu představuje počáteční problém, hrany znamenají rekurzivní volání algoritmu, cesty do jednotlivých uzlů reprezentují částečné nalezené řešení ($Path$). List je dosažen tehdy, když je nalezeno řešení, nebo cena nalezeného řešení překročila horní mez. Vstupem algoritmu je cyklické jádro problému (A), výstupem je minimální řešení pokrytí ($bestSolution$).

3.1.1 Vylepšení

Tento algoritmus lze vylepšit dvěma způsoby:

1. Snížením horní meze UB
2. Zavedením spodní meze LB

3.1.2 Snížení horní mezeí

Čím nižší bude horní mez, tím více bude algoritmus ořezávat větve které nevedou na optimální řešení a tím rychlejší prohledávání prostoru ve výsledku bude. Snížení horní meze lze docílit dvěma způsoby:

1. Použitím heuristiky. Tento krok je vhodný provést především před samotným spuštěním algoritmu větví a hranic. Docílí se tím to, že algoritmus omezí prohledávání stavového prostoru hlavně pro počáteční fáze prohledávání.
2. Vhodným výběrem sloupců. Pokud algoritmus již od počátku běhu vybírá sloupce tak, že první nalezené řešení bude mít nízkou (pokud možno optimální) cenu, omezí se tím prohledávaný prostor.

3.1.3 Spodní mez

Předpokládejme, že známe hodnotu $LB(A_N)$ reprezentující minimální cenu řešení potřebnou pro pokrytí matice A v daném stavu N. Tato hodnota poskytuje spodní mez řešení matice A. V takovém případě, pokud :

$$UB \leq LB(A_N) + cena(Path) \tag{3.1}$$

hledání lze zastavit a algoritmus se ve stavovém prostoru může vrátit zpět. Následuje popis algoritmu využívající znalost spodní meze (LB).

Algorithm 3.1.3.1 Branch and Bound s využitím spodní meze

```
Branch_and_bound_LB(A,Path) {
  /* odstraní dominantní řádky a sloupce,
  vybere do řešení nezbytné sloupce */
  A = solve_dominance(A);
  /*Nalezne spodní mez pro aktuální problém*/
  LB = find_lower_bound(A);
  lBoundNew = cost(Path) + LB
  if ( lBoundNew ≥ UB ) {
    /*pokud jsme již dosáhli spodní meze, nebudeme větvit dále*/
    bestSolution = ∅
    return bestSolution
  }
  if ( A == ∅ ) {
    if ( cost(Path) < UB ) UB = cost(Path)
    bestSolution = Path
  } else {
    j = select_column(A);
    Path1= Path + j;
    A' = A - { j + rows_covered_ by(j) }
    Solution1 = Branch_And_Bound(A',Path1)
    A'' = A – {j}
    Solution2 = Branch_And_Bound(A'',Path2)
    bestSolution = min( Solution1 , Solution2 )
  }
  return bestSolution
}
```

Způsob, jak zjistit hodnotu $LB(A_N)$, může být např. zjištěním množiny nezávislých řádků - MSIR (*maximum set of independent rows*).

3.1.4 MSIR

Definice 3.1.1 *Nechť $\langle X, Y, CENA\rangle$ je problém pokrytí a Y' je podmnožina Y taková, že pro dva různé libovolné prvky y_1 a y_2 z množiny Y', všechny prvky x, které pokrývají y_1 nepokrývají y_2 a naopak. Množina Y' je pak nezávislou podmnožinou Y a její cena poskytuje spodní mez řešení.*

Jinak řečeno: v matici A se nachází množiny řádků takové, že žádný sloupec pokrývající řádek z jedné množiny nepokrývá ani jeden řádek z jiné množiny. V tabulce 3.1 je nezávislou podmnožinou řádků podmnožina $X = \{x_1, x_2\}$, neboť x_1 je pokryt právě sloupci y_1 a y_2 a x_2 je pokryt právě sloupci y_3 a y_4. Z toho lze vyvodit, že pro pokrytí bude třeba vybrat jeden sloupec z podmnožiny $\{y_1, y_2\}$ a jeden sloupec z podmnožiny $\{y_3, y_4\}$. Vybereme-li sloupce s nižší cenou, dostaneme minimální cenu pokrytí $2 + 2 = 4$.

x_1	1	1		
x_2			1	1
x_3			1	1
x_4			1	1
	y_1	y_2	y_3	y_4

Tabulka 3.1: Příklad na MSIR

Hledání množiny MSIR však znamená hledání řešení dalšího NP-úplného problému, a čas pro hledání MSIR by jistě převýšil čas uspořený nalezením přesné spodní meze. Proto je pro nalezení MSIR zvolena rychlá heuristická funkce. Její použití vede k nižší hodnotě než MSIR a tedy větší části stavového prostoru, který je třeba prohledat. Řešení zůstává řešením exaktním.

3.2 AURA II

Algoritmus AURA II je blíže popsán v [3]. Jde o algoritmus, který pro určení spodní meze využívá heuristiku hledající MSIR. Při splnění určitých podmínek používá tzv. negativní přístup, kdy se při splnění určitých podmínek snaží tuto spodní mez zvýšit bližším zkoumáním množin MSIR.

3.2.1 Heuristika MSIR

V předešlé části je řečeno, že hledání $MSIR$ je časově náročné. Proto se v AURA II k jejímu nalezení používá následující heuristika:

Algorithm 3.2.1.1 Find MSIR

```
FindMSIR(A) {
  bestRow = first_row(A)
  for each row ∈ A {
    r = find_row_with_minimum_1(A)
    if ( RP(r) < RP(bestRow) ) bestRow = r
  }
  for each row r ∈ A {
    if ( rows_are_dependent(r,bestRow,A))
      A' = remover_row_from_matrix(r,A)
  }
  MSIR = minimal_cost_column_covering(bestRow)
  MSIR = MSIR + FindMSIR(A')
  return MSIR
}
```

kde RP znamená řádkový příspěvek:

$$RP(x_i) = \sum_{j=1}^{n} A_{i,j} \cdot SP(y_j)$$

kde SP znamená sloupcový příspěvek:

$$SP(y_j) = \sum_{i=1}^{m} A_{i,j}$$

Čím nižší bude $RP(x_i)$, tím vyšší bude šance na nezávislost řádků. Jinými slovy, algoritmus vybírá řádky, které jsou minimálně pokryty a toto pokrytí je tvořeno sloupci, které pokrývají nejméně řádků.

Nechť:

$$K(A_N) = UB - LB(A_N) + cena(path)$$

Pokud je $K(A_N) < 0$, je třeba zastavit další volání algoritmu. V opačném případě je cena částečného nalezeného řešení menší než cena nejlepšího nalezeného řešení a je možné dále prohledávat stavový prostor. Tehdy má algoritmus dvě možnosti jak pokračovat. První z nich je označena jako *"pozitivní"* - algoritmus bude pokračovat ve větvení, aby se přesvědčil, že jím doposud nalezené řešení je nejlepší. Druhý přístup se nazývá *"negativní"* - algoritmus se snaží dokázat, že lepší řešení již neexistuje. Pokud je nalezená hodnota $K(A_N)$ nízká, je přirozenější pro algoritmus být „skeptický“ ohledně nalezení lepšího řešení.

Algoritmus AURA II se při řešení problému pokrytí snaží zahrnout obě techniky. Pracuje s upraveným rekurzivní algoritmem - zpočátku řeší problém v režimu pozitivního přístupu. Později, když počet vzniklých podproblémů je dostatečně veliký, je každý podproblém řešen v režimu negativního přístupu. Čím menší je hodnota $K(A_N)$, tím přirozenější je řešení podproblému v režimu negativního přístupu.

Negativní přístup AURA II spočívá ve zvyšování spodní meze postupným přidáváním řádků do MSIR a zkoumáním možných řešení. Algoritmus je přesně popsán v [1].

3.3 Heuristické metody

Nalezení exaktního řešení pro větší matice trvá velmi dlouhou dobu. V některých případech (především pro rozsáhlé problémy) je použití heuristické funkce pro nalezení řešení UCP jediná možnost, jak zjistit alespoň přijatelné řešení. V této kapitole jsou stručně popsány dvě hladové heuristiky.

3.3.1 Heuristika podle Servíta

Tato heuristika je popsána v [6]. Heuristika se snaží vybírat sloupce, které jsou pokryty řádky, které jsou pokryty co nejméně sloupci, přičemž se berou v úvahy ceny sloupců pokrývající řádky. Velikost pokrytí sloupců je udávána jako síla pokrytí SPS.

$$SPS(y_j) = \frac{\sum_{i=1}^{m} A_{i,j} \cdot SPR(x_i)}{cost(j)}$$

kde SPR značí sílu pokrytí řádku:

$$SPR(x_i) = \left(\sum_{j=1}^{n} \frac{A_{i,j}}{c_j} \right)^{-1}$$

Při každém přidání sloupce se redukuje původní problém (matice). Heuristika tak musí zkoušet, jestli v matici nevznikly nové nezbytné sloupce.

Algorithm 3.3.1.1 Servít

```
Servít(A) {
  while (A is not covered) {
    c = find_column_with_maximum_SPS(A)
    add_to_solution(c,solution)
    remove_from_problem(c,A)
    e = find_essential_columns(A)
    add_to_solution(e,solution)
    remove_from_problem(e,A)
  }
  return solution
}
```

3.3.2 Heuristika ContribAdd

Jedná se o heuritstiku, která se v algoritmu AURA II používá k výběru sloupce při větvení. Lze ji ovšem použít samostatně za předpokladu, že se v každé iteraci kontroluje, jestli v matici nevznikly nezbytné sloupce. Heuristika se snaží vybírat sloupce, které jsou pokryty řádky, které jsou pokryty co nejméně sloupci. Síla pokrytí sloupců se vypočítá takto:

$$SPS(y_j) = \frac{\sum_{i=1}^{m} A_{i,j} \cdot SPR(x_i)}{cost(j)}$$

kde SPR značí sílu pokrytí řádku:

$$SPR(x_i) = \left(\sum_{j=1}^{n} A_{i,j} \right)^{-1}$$

Algorithm 3.3.2.1 ContribAdd

```
ContribAdd(A) {
  while (A is not covered) {
    c = find_column_with_maximum_SPS(A)
    add_to_solution(c,solution)
    e = find_essential_columns(A)
    add_to_solution(e,solution)
    remove_from_problem(e,A)
  }
  return solution
}
```

3.4 Lagrangeovská relaxace

3.4.1 Principy Lagrangeovské relaxace

Lagrangeovská relaxace optimalizačních problémů s omezujícími podmínkami je velmi často používanou technikou pro jejich řešení. Lagrangeovská relaxace se tvoří takto [**?**, 3]

1. Odstraní se množiny omezujících podmínek problému
2. Tyto podmínky se promítnou do optimalizační funkce ve formě tzv. trestných koeficientů, které se nazývají *Lagrangeovské multiplikátory.*

3.4.2 Relaxace problému pokrytí

Mějme jiný zápis pro problém pokrytí P definovaném v 2.0.1:

$$minz = cx$$

$$podm.: \ Ax > e \ , \ x \in \{0,1\}^n$$

Kde: c je vektor cen.

x je binární vektor sloupců. Prvek tohoto vektoru je jedna, pokud daný sloupec patří do řešení, jinak je roven nule.

e je vhodný jednotkový vektor ($e_i = 1; i = 1, 2, \ldots .m$). Nerovnice $Ax > e$ tedy reprezentuje podmínku říkající, že každý řádek musí být pokryt alespoň jedním sloupcem.

Tento problém může být relaxován na problém lineárního programování LP změnou omezující podmínky:

$$minz = cx$$

$$podm.: \ Ax > e \ , \ 0 \leq x \leq 0$$

Pokud je optimální řešení tohoto problému celočíselné, je potom také optimální řešení původního problému P. Tomu však většinou není a optimální výsledek tak slouží jako spodní mez původního problému ($z_P^* \leq z_{LP}^*$).

Jelikož i tento lineární problém není snadné řešit, lze jej převést na problém Lagrangeovské relaxace LRP. To se provede postupem popsaným v 3.4.1. Výsledkem je:

$$minz = \tilde{c}x + \lambda e$$

$$podm.: \ 0 \leq x \leq 0$$

Kde: $\tilde{c} = c - A\lambda$ je vektor *redukovaných cen* sloupců (nebo také vektor Lagrangeovských cen).

λ jsou Lagrangeovské multiplikátory definované pro každý řádek matice A. Tyto multiplikátory charakterizují *míru neplnění původních omezujících podmínek* aktuálního řešení x pro jednotlivé řádky.

Ze vzorce pro redukovanou cenu sloupce vyplývá to, že bude-li sloupec pokrývat řádky s větší mírou neplnění omezujících podmínek, jeho redukovaná cena bude nižší a jeho zařazení do řešení je tudíž žádoucí. Optimálního řešení LRP lze dosáhnout vhodnou volbou sloupců. Očividně se vyplatí vybírat do řešení ty sloupce, jejichž redukovaná cena je záporná ($\tilde{c}_j^* \leq 0$) - tak se pokryjí právě ty řádky, které mají velkou míru neplnění omezujících podmínek. Sloupec se do řešení zahrne tak, že se nastaví jemu příslušná proměnná ve vektoru x ($x_j = 1$).

Řešení LRP nemusí být přípustným řešením problému pokrytí P (některé řádky zůstanou nepokryty), avšak modifikace řešení LRP přidáváním nebo odebíráním sloupců může vést k dobrým řešením.

Cena řešení LRP pro všechny $\lambda \geq 0$ tvoří spodní mez lineární relaxace ($z_P^* \leq z_{LP}^* \leq z_{LRP}^*$). Cílem Lagrangeovské relaxace je tedy určit takové hodnoty multiplikátorů λ, pro něž je cena řešení LRP $z_{LRP}^*(\lambda)$ nejvyšší. Technika zde označovaná jako subgradientem řízený postup, která umožňuje vyjít z jakýchkoliv hodnot λ_0 a postupně je vylepšovat – potenciálně až na hodnoty, kdy $z_{LRP}^*(\lambda) = z_P^*$.

3.4.3 Subgradientem řízený postup

Hlavním cílem je iterativně vylepšovat hodnoty λ_k na základě ceny posledního řešení $z_{LRP}(\lambda_{k-1})$. Optimální řešení problému $x_k^* = x_{LRP}^*(\lambda_k)$ porušuje omezení na pokrytí o hodnotu $s_{\lambda_k} = e - Ax_k^*$ (rezerva) a cena tohoto řešení je $z_{\lambda_k} = z_{LRP}^*(\lambda_k)$. Tyto vztahy vedou k úpravě λ_k podle vztahu:

$$\lambda_{k+1} = max\left(\lambda_k + s_{\lambda_k}\frac{|z_P^* - z_{\lambda_k}|}{\|s_{\lambda_k}\|^2}, 0\right)$$

V dalším kroku budou porušená omezení (reprezentované kladnými hodnotami s_{λ_k} penalizována tak, že bude navýšen příslušný multiplikátor, zatímco při splněných podmínkách budou multiplikátory sníženy (nikdy pod 0).

Subgradient $\|s_{\lambda_k}\|^2$ je vektor, který udává směr dalšího postupu při řešení. Jak vyplývá ze vtahu $s_{\lambda_k} = e - Ax_k^*$, složky subgradientu pro dosud nepokryté řádky jsou jednotkové, pro ostatní jsou nulové nebo záporné. Rozdíl $|z_P^* - z_{\lambda_k}|$ určuje, jak moc se cena optimalizační funkce blíží k optimální hodnotě. Čím je dále, tím je větší rozdíl a tím více se budou multiplikátory měnit. Tímto procesem se nastavují multiplikátory λ_k a hodnota z_{λ_k} postupně osciluje. Její nejlepší známá hodnota LB postupně roste a blíží se k z_P^*. Zároveň s tím λ_k obsahuje více informací o problému.

Teoreticky je zajištěna konvergence tohoto procesu. Většinou je ale hodnota z_P^* neznámá a je nahrazena horní mezí UB. Použití horní meze ale může zamezit konvergenci, protože rozdíl $UB - z_{\lambda_k}$ může být větší jak 0. Proto se používá zmenšující se koeficient kroku t_k. Tento je zmenšen (např. půlen) vždy po určitém počtu iterací N_t, během kterých se nezmění hodnota LB. Pokud se totiž tato hodnota nemění, je velice pravděpodobné, že je to způsobeno příliš velkými změnami v multiplikátorech. Výsledkem je mírně upravený vztah:

$$\lambda_{k+1} = max\left(\lambda_k + t_k s_{\lambda_k} \frac{|UB - z_{\lambda_k}|}{\|s_{\lambda_k}\|^2}, 0\right)$$

Proces končí ve chvíli, kdy je z_{λ_k} velmi blízko UB (může být zadaná určitá mez) nebo t_k je natolik malé, že neumožňuje další změny v LB ($t_k < t_{min}$). Třetí podmínka může nastat, pokud předpokládáme, že vektor cen c je celočíselný. V tom případě je z_P^* také celočíselné. Pokud známé celočíselné řešení má cenu $\lceil z_{\lambda_k} \rceil$, je optimální, protože žádné jiné nemůže být lepší.

První hodnoty λ_0, kdy nejsou známy žádné informace o problému, můžou být jakékoliv (pokud nejsou negativní). K jejich nastavení je ale také možné použít heuristiku.

Určování hodnot λ_0 , t_{min} , t_0 a N_t se věnují další kapitoly. Pro jejich určení bylo provedeno několik experimentálních měření.

3.4.4 Nastavení horní meze

Hodnotu UB je možné získat dvěma způsoby:

1. Heuristikou. Tato může být jakákoliv (např. Contrib Add - v 3.3.2). V heuristice je ale možné použít řešení LRP – taková heuristika se potom nazývá *Lagrangeovská heuristika*.

2. Pomocí duálního problému. Ten je popsán v kapitole 3.6.

Lagrangeovské heuristiky pro problém pokrytí pracují tak, že začnou s řešením LRP, přidají do řešení sloupce tak, aby upravené řešení bylo přípustným řešením problému pokrytí a případně odeberou z řešení sloupce nadbytečné. Pravidla pro přidávání či odebírání sloupců mohou být určena různě, vznikne tak několik variant Lagrangeovské heuristiky. Pravidla jsou převzata z [2].

Řešení LRP sice není přijatelné jako řešení P, ale jako výchozí řešení může poslouží dobře [4]. Pokud jsou hodnoty Lagrangeovských multiplikátorů téměř optimální, je pravděpodobné, že příslušné řešení LRP je blízko optimálního a Lagrangeovské ceny mohou poskytnout dobré informace o tom, jak upravovat řešení tak, aby bylo přijatelné pro problém P.

3.4.5 Úprava řešení problému Lagrangeovské relaxace

V této části je popsáno pět heuristik, které v této práci byly použity pro vytvoření přípustného řešení problému P z řešení LRP. Všechny tyto heuristiky mají společné to, že se skládají ze dvou fází:

1. V první fázi jsou do řešení LRP přidávány sloupce tak, aby byly pokryty všechny řádky. Přidávání sloupců probíhá tak, že se vždy pro jeden řádek vybírá mezi sloupci, který daný řádek pokrývají. Řádky se vybírají podle počtu sloupců, které pro daný řádek obsahují jedničku (řádky s více jedničkami se vybírají pro pokrytí jako první).

2. V druhé části jsou naopak sloupce odebírány, nikdy však tak, aby došlo k porušení pokrytí.

Heuristiky se liší v kritériích, podle kterých jsou sloupce přidávány a odebírány:

Varianta 1

- Přidávány jsou sloupce s nejnižší cenou.
- Odebírány jsou sloupce s nejvyšší cenou.

Tato varianta představuje původní řešení navržené Beasleym [2] a nevyužívá informace z vypočtených Lagrangeovských multiplikátorů. Ostatní varianty tyto informace různým způsobem využívají.

Varianta 2

- Přidávány jsou sloupce s nejnižší redukovanou cenou.
- Odebírány jsou sloupce s nejvyšší cenou.

Varianta 3

- Přidávány jsou sloupce s nejnižší modifikovanou redukovanou cenou.
- Odebírány jsou sloupce s nejvyšší cenou.

Varianta 4

- Přidávány jsou sloupce s nejnižší redukovanou cenou.
- Odebírány jsou sloupce s nejvyšší redukovanou cenou.

Varianta 5

- Přidávány jsou sloupce s nejnižší modifikovanou redukovanou cenou.
- Odebírány jsou sloupce s nejvyšší modifikovanou redukovanou cenou.

Pro varianty 3 až 5 se používá modifikovaná redukovaná cena. Její výpočet probíhá tak, že pro již pokryté řádky se Lagrangeovské multiplikátory vynulují.

3.4.6 Výstupy Lagrangeovské relaxace

Lagrangeovská relaxace poskytuje tři výstupy:

1. LB - Dolní mez řešení problému.
2. Řešení (ne nutně optimální) problému pokrytí.
3. Sloupec, který by se měl v algoritmu B&B přidat do řešení. Jedná se o sloupec, který je součástí heuristického řešení a jeho redukovaná cena je nejnižší.

3.5 Duální problém

V lineárním programování lze ke každému problému najít sdružený – tzv. *duální* problém [**?**, 9] Tvoří se takto:

1. Pokud je původní (*primární*) problém maximalizační, duální bude minimalizační (a naopak).
2. Proměnné v primárním problému slouží jako omezení v duálním.
3. Omezení v primárním problému slouží jako proměnné v duálním problému.

Mějme např. problém:

$$max \sum_{j=1}^{n} c_j x_j \ , \ podm. : \sum_{n=1}^{n} a_{i,j} x_j \leq b_i \ , \ x_j \geq 0$$
$$i = 1, 2, 3, ..., m \ , \ j = 1, 2, 3, ..., n$$
$$x = (x_1, x_2, x_3, ..., x_n) \ , \ y = (y_1, y_2, y_3, ..., y_m)$$

Potom duální problém vypadá takto:

$$min \sum_{i=1}^{m} b_i y_i \ , \ podm. : \sum_{i=1}^{m} y_i a_{i,j} \geq c_j \ , \ y_i \geq 0$$
$$i = 1, 2, 3, ..., m \ , \ j = 1, 2, 3, ..., n$$
$$x = (x_1, x_2, x_3, ..., x_n) \ , \ y = (y_1, y_2, y_3, ..., y_m)$$

Duální problém k duálnímu problému je primární problém.

Slabá věta o dualitě říká, že pokud jsou řešení primárního x i duálního y problému přijatelná, potom platí:

$$\sum_j c_j x_j \leq \sum_i b_i y_i$$

důkaz:

$$\sum_j c_j x_j \leq$$

$$\leq \sum_j \left(\sum_i y_i a_{ij} \right) x_j = \sum_{ij} y_i a_{i,j} x_j = \sum_i \left(\sum_j a_{ij} x_i \right) y_i \leq$$

$$\leq \sum_i b_i y_i$$

Silná věta o dualitě říká, že optimální řešení primárního x^* i duálního y^* problému jsou přijatelná pouze pokud platí:

$$\sum_j c_j x_j^* = \sum_i b_i y_i^*$$

důkaz je konstruktivní za pomoci simplexové metody [10].

3.5.1 Duální problém problému pokrytí

K problému LP uvedeném v 3.4.2 existuje následující duální problém LD:

$$max w = ey$$

$$podm.: \ Ay < c \ , \ 0 \leq y \leq \bar{c}$$

Kde: y je vektor je vektor duálních proměnných.

$\bar{c} = min_{j:A_{ij}=1} c_j$ je nejmenší možná cena sloupce, který pokrývá daný řádek.

LD je úzce spjat s LRP: pokud je duální přípustné řešení y použit jako vektor Lagrangeovských multiplikátorů λ, příslušné řešení LRP s cenou $z^*_{LRP}(m)$ je rovné ceně řešení duálního problému $w(m)$.

$$c - Ay = \tilde{c}(m) > 0 \rightarrow x^*_{LRP}(m) = 0 \rightarrow$$

$$\rightarrow z^*_{LRP}(m) = ye = ey = w_{LD}(m)$$

Tedy každé dobré duální řešení je dobrý vektor Lagrangeovských multiplikátorů. Především každé optimální řešení duálního problému je optimální vektor multiplikátorů, protože $z^*_{LRP}(m^*) = w_{LD}(m^*)$. Podle silné věty o dualitě [4] navíc platí, že $w^*_{LD} = z^*_{LP}$.

Řešení duálního problému je stejně složité jako řešení primárního. Lze jej ale podobně jako primární problém relaxovat na Lagrangeovský duální lineární problém LDP:

$$max w = \tilde{e}y + \mu c$$

$$podm.: \ 0 \leq y \leq \bar{c}$$

Kde: $\tilde{e} = e - A\mu$ je vektor *duálních redukovaných cen sloupců.*

μ jsou *Duální Lagrangeovské multiplikátory.*

Cena optimálního řešení LD představuje horní mez problému LP $w_{LDP}(\mu^*) \geq w^*_{LD} = z^*_{LP}$. Tato mez může vylepšit hodnotu UB použitou ve vztahu pro úpravu Lagrangeovských multiplikátorů λ v části 3.4.2.

Obdobně jako pro řešení LRP i pro řešení LDP platí, že žádoucí je do řešení zahrnout pouze řádky, jejichž duální redukovaná cena je nezáporná ($\tilde{e}_i^* \geq 0$) [**?**, 5] Stejně tak platí, že cílem duální Lagrangeovské relaxace je najít takové hodnoty multiplikátorů μ, pro něž je cena řešení problému LDP $w_{LDP}^*(\mu)$ nejnižší. Řádek se do řešení zahrne tak, že se nastaví jemu příslušná proměnná $y_i = \bar{c}_i$, jinak $y_i = 0$.

Vztah pro úpravu μ je následující:

$$\mu_{k+1} = max\left(\mu_k + t_{Dk} s_{D\lambda_k} \frac{|w_{LDP\lambda_k} - LB|}{\|s_{D\lambda_k}\|^2}, 0\right)$$

Kde: $s_{D\lambda_k} = Ay_k^* - c_{jk}$ (duální rezerva) představuje velikost porušení omezení $A_y \leq c$ problému D.

Duální koeficient kroku t_{Dk} se obdobě jako ve vztahu v 3.4.2 požívá proto, aby bylo zajištěno vylepšování hodnoty UB. Tento koeficient je zmenšen (např. půlen) vždy po určitém počtu iterací N_t, během kterých se nezmění hodnota UB.

Počáteční hodnoty multiplikátorů μ_0 mohou být určeny náhodně (pokud nejsou negativní) nebo mohou být určeny heuristikou.

Metoda subgradientem řízeného postupu je aplikována na primární i duální Lagrangeovský problém. Nejlepší známá řešení jsou použita jako meze problémů, což vede k přesnějšímu výpočtu multiplikátorů, které se tak nastaví v menším počtu kroků.

Kapitola 4

Implementace

V této části je popsán celkový algoritmus, který využívá postupy popsané v kapitole 3. Základem je poupravený Branch and Bound algoritmus, který pro určení spodní meze využívá Lagrangeovskou relaxaci. Tato relaxace také může vracet řešení (ne nutně optimální), které může být použito k nastavení horní meze a nejlepšího známého řešení. Relaxace také vrací sloupec, který se zahrne do řešení při větvení.

4.1 Základní algoritmus

Algorithm 4.1.0.1 Branch and Bound s využitím Lagrangeovské relaxace

```
Branch_and_bound_Lagrange(A,Path) {
  /* odstraní dominantní řádky a sloupce,
  vybere do řešení nezbytné sloupce */
  A = solve_dominance(A);
  tempSolution = bestSolution
  if ( A == ∅ ) {
    if ( cost(Path) < UB ) UB = cost(Path)
    bestSolution = Path
  } else {
    A = sort_matrix(A)
    /*Nalezne spodní mez pro aktuální problém*/
    Lagrange_solution, LB_Lagrange,
    Lagrange_column = Solve_Lagrange_(A,Params);
    lBoundNew = cost(Path) + LB_Lagrange
    if ( lBoundNew ≥ UB ) {
      /*pokud jsme již dosáhli spodní meze, nebudeme větvit dále*/
      bestSolution = ∅
      return bestSolution
    } else {
      /*Horní mez nebyla dosažena*/
```

```
      if ( LB_Lagrange == Lagrange_soluton_cost ) {
        /*Bylo nalazeno heuristické řešení, které již nemůže být lepší*/
        bestSolution = Lagrange_soluton
        uBound = cost(Path) + Lagrange_soluton_cost
        return bestSolution
      } else {
        /*Bylo nalazeno heuristické řešení, může ale existovat lepší řešení*/
        tempSolution = Lagrange_soluton
        uBound = cost(Path) + Lagrange_soluton_cost
      }
    }
    Path1= Path + Lagrange_column;
    A' = A - { j + rows_covered_ by(Lagrange_column) }
    Solution1 = Branch_And_Bound(A',Path1)
    A'' = A – {Lagrange_column}
    Solution2 = Branch_And_Bound(A'',Path2)
    bestSolution = min( tempSolution , Solution1 , Solution2 )
  }
  return bestSolution
}
```

Parametry předávané metodě Solve_Lagrange jsou popsané v části věnující se dané metodě.

4.1.1 Reprezentace matice

Data matice jsou uložena v dvourozměrném poli obsahujícím jedničky a nuly. Dále je uložen vektor obsahující ceny sloupců. Nakonec se k reprezentaci matice používají dva spojové seznamy - jeden pro řádky a druhý pro sloupce. V každém prvku seznamu se vede informace o tom, kterému řádku(resp. sloupci) prvek odpovídá a dále počet jedniček obsažených v daném řádku(sloupci). Tento údaj se s výhodou používá při odstraňování dominancí a třídění matice. Navíc se v takovéto reprezentaci dají velmi snadno odebírat řádky(sloupce) jednoduchým odebíráním prvků ze seznamů.

4.1.2 Odstranění dominancí

Odstraňování dominancí v algoritmu probíhá ve dvou fázích:

1. řádková fáze: Algoritmus prochází všechny řádky. Vždy vezme jeden řádek x' z matice a porovná ho s ostatními řádky. Pokud nalezne řádek x obsahující stejný nebo vyšší počet jedniček jako řádek x', je šance, že řádek x' dominuje řádku x. Dále algoritmus zjistí, jestli x obsahuje jedničky ve stejných sloupcích jako x'. Pokud ano, řádek x je odstraněn z matice. Jelikož se porovnává každý řádek s každým, operační složitost algoritmu v nejhorším případě je $O(n \times m^2)$, kde m je počet řádků a n počet sloupců matice. V této části také algoritmus při procházení porovnávaného řádku x' zjišťuje,

jestli daný řádek není pokryt pouze jedním sloupcem. Pokud ano, je sloupec pokrývající daný řádek zařazen do řešení jakožto nezbytný.

2. sloupcová fáze: Algoritmus vezme sloupec y' z matice a porovná ho s ostatními sloupci. Pokud sloupec y obsahuje stejný nebo menší počet jedniček jako sloupec y', je možnost, že y' dominuje y. Algoritmus zjistí, zda y obsahuje jedničky ve všech řádcích, ve kterých je obsahuje y', a je splněna podmínka $cena(y') \leq cena(y)$. Pokud ano, sloupec y je vyjmut z matice. Pokud podmínka splněna není, sloupec se neodstraňuje – algoritmus by se tak mohl připravit o minimální řešení. Opět se porovnává každý sloupec s každým, takže operační složitost v nejhorším případě, kdy se nevyškrtne žádný sloupec, je $O(m \times n^2)$,kde m je počet řádků a n počet sloupců matice.

4.1.3 Třídění matice

Pro potřeby Lagrangeovské relaxace je nutné mít matici v odpovídajícím tvaru. Algoritmus matici setřídí následovně:

- Řádky podle počtu sloupců, které daný řádek pokrývají
- Sloupce podle ceny. Pokud nějaké sloupce mají stejnou cenu, jsou seřazeny podle počtu řádků, které sloupce pokrývají.

Třídění matice je fyzicky provedeno tak, že se mění pořadí referencí sloupců a řádků ve spojových seznamech (viz 4.1.1).

4.2 Lagrangeovská relaxace

Tato metoda může a nemusí brát v úvahu duální problém. Porovnání jejich výkonnosti a účinnosti se nachází v části 3.4.5. Zde následuje popis obou metod.

4.2.1 Algoritmus Lagrangeovské relaxace

Algorithm 4.2.1.1 Lagrangeovská relaxace

Solve_Lagrange (A,t_init,t_min,N,N_t,N_h) {

t=t_{init}
for ($i_t = 1$ to N) {
/*Řešení LRP:*/
/*výpočet redukovaných cen a přidání sloupce do řešení LRP*/
$z_{LRP} = 0$
for ($j = 1$ to n) {
$\tilde{c}_j = c_j - \sum_{i=1}^{m} A_{i,j}\lambda_j$
if ($\tilde{c}_j < 0$) then {
$x_j = 0$
$z_{LRP} = z_{LRP} + c_j$
} else $x_j = 1$
}
/*dopočítání ceny řešení LRP*/
for ($i = 1$ to m) $z_{LRP} = z_{LRP} + \lambda_i$
/*zjištění UB_{heur}*/
if (i_t mod $N_h == 0$) then UB_{heur} = complete_solution(x)
/*=uprava mezí*/
$LB = max(LB, z_{DUAL})$
$UB = min(UB, UB_heur)$
/*výpočet vektoru subgradientu*/
for ($i = 1$ to m) $s_i = 1 - \sum_{j=1}^{n} A_{i,j}x_j$
/*vypočtení subgradientu*/
$s = 0$
for ($i = 1$ to m) $s = s + s_i^2$
if ($s == 0$) then return
/*úprava koeficientu kroku*/
if (LB hasn't changed for N_t) then $t = t/2$
if ($t < t_{min}$) then return
/*výpočet kroku*/
$step = t \cdot (UB - z_{LRP})/s$
/*úprava hodnot multiplikátorů*/
for ($i = 1$ to m) do $\lambda_i = max(0, \lambda_i + step \cdot s_i)$
}
}

4.2.1.1 Parametry Lagrangeovské relaxace

A	Cyklické jádro problému velikosti $m \times n$
t_{init}	Počáteční koeficient kroku
t_{min}	Minimální koeficient kroku
N	Maximální počet iterací
N_t	Počet iterací, než dojde ke snížení koeficientu kroku t
N_h	Počet iterací, než dojde k zavolání heuristiky na vytvoření přípustného řešení (a potencionálnímu nastavení UB)

Tabulka 4.1: Parametry Lagrangeovské relaxace

Hodnoty parametrů byly zjištěny experimentálně. Tomu se blíže věnuje kapitola ref.

4.2.2 Algoritmus Duální Lagrangeovské relaxace

Algorithm 4.2.2.1 Duální Lagrangeovská relaxace

Solve_dual_Lagrange (A,t_{init},t_{Dinit},t_{min},t_{Dmin},N,N_t,N_{Dt},N_h) {

t=t_{init}

t_D=t_{Dinit}

/*výpočet primárních a duálních redukovaných cen*/

for ($i_t = 1$ to N) {

$z_{LDP} = 0$

for ($j = 1$ to n) {

$\tilde{c}_j = c_j - \sum_{i=1}^{m} A_{i,j}\lambda_j$

if ($\tilde{c}_j < 0$) then {

$x_j = 0$

$z_{LDP} = z_{LDP} + c_j$

} else $x_j = 1$

$w_{LDP} = 0$

for ($i = 1$ to m) {

$\tilde{e}_i = 1 - \sum_{j=1}^{n} A_{i,j}\mu_i$

if ($\tilde{e}_i > 0$) then {

$y_i = \bar{c}_i$

$w_{LDP} = w_{LDP} + y_i\tilde{e}_i$

} else $y_i = 0$

}

/* dopočet hodnot řešení*/
for ($i = 1$ to m) $z_{LDP} = z_{LDP} + \lambda_i$
for ($j = 1$ to n) $w_{LDP} = w_{LDP} + c_j \mu_j$

/*zjištění UB_{heur}*/
if (i_t mod $N_h == 0$) then UB_{heur} = complete_solution(x)

/*=uprava mezí*/
$LB = max(LB, z_{DUAL})$
$LB = min(UB, UB_{heur}, w_{DUAL})$

/*výpočet vektorů subgradientů*/
for ($i = 1$ to m) $s_i = 1 - \sum_{j=1}^{n} A_{i,j} x_j$
for ($j = 1$ to n) $s_{Dj} = 1 - \sum_{i=1}^{m} A_{i,j} y_i - c_j$

/*výpočet subgradientů*/
$s = 0$
$s_D = 0$
for ($i = 1$ to m) $s = s + s_i^2$
for ($j = 1$ to n) $s_D = s_D + s_{Dj}^2$
if ($s == 0$ and $s_D == 0$) then return

/*úprava koeficientů kroku*/
if (LB hasn't changed for N_t) then $t = t/2$
if (best value of w_{DUAL} hasn't changed for N_{Dt}) then $t_D = t_D/2$
if ($t < t_{min}$ and $t_D < t_{Dmin}$) then return

/*výpočet kroku*/
$step = t \cdot (UB - z_{LRP})/s$
$step_D = t_D \cdot (w_{LRP} - LB)/s_D$

/*úprava hodnot multiplikátorů*/
for ($i = 1$ to m) do $\lambda_i = max(0, \lambda_i + step \cdot s_i)$
for ($j =$ j to n) do $\mu_j = max(0, \mu_j + step_d \cdot s_{Dj})$

}
}

4.2.2.1 Parametry Duální Lagrangeovské relaxace

A	Cyklické jádro problému velikosti $m \times n$
t_{init}	Počáteční primární koeficient kroku
t_{Dinit}	Počáteční duální koeficient kroku
t_{min}	Minimální koeficient kroku
t_{Dmin}	Minimální duální koeficient kroku
N	Maximální počet iterací
N_t	Počet iterací, než dojde ke snížení koeficientu kroku t
N_{Dt}	Počet iterací, než dojde ke snížení duálního koeficientu kroku t
N_h	Počet iterací, než dojde k zavolání heuristiky na vytvoření přípustného řešení (a potencionálnímu nastavení UB)

Tabulka 4.2: Parametry duální Lagrangeovské relaxace

Hodnoty parametrů byly zjištěny experimentálně. Tomu se věnuje následující kapitola.

4.2.2.2 Optimalizace implementace

Přesná implementace se od algoritmu 4.2.2.1 liší v tom, že v každé iteraci nepřepočítává všechny relevantní parametry (redukované ceny, multiplikátory,rezervy). Používají se totiž vektory δ_λ a δ_μ, které označují změnu daných multiplikátorů v předešlé iteraci. Tyto vektory jsou řídké. V každé iteraci se tedy prozkoumávají tyto změnové vektory a nastavují se nové redukované ceny (primární i duální). Pokud dojde ke změnám v znaménkách cen, opraví se příslušné primární a duální rezervy s (a s_D). Nakonec se přepočítané rezervy použijí k přepočítání multiplikátorů a nastavení rozdílových vektorů δ_λ a δ_μ.

4.2.3 Reprezentace matice

Pro potřeby Lagrangeovských relaxací je matice reprezentována jako dvourozměrné pole. Matice je setříděna postupem popsaným v 4.1.3. Pro každý řádek (sloupec) je určeno, jaké sloupce (řádky) daný řádek pokrývá. Toho je využito především při převádění řešení LRP na přípustné řešení problému pokrytí (tedy při Lagrangeovské heuristice), kdy se do řešení vybírají sloupce podle daného kritéria (viz 3.4.5).

4.2.4 Jazyk implemetace

Pro implemetaci byl podle zadání použit jazyk C++. Především z důvodu rychlosti a také aby byla možná snadná integrace do programu BOOM.

Kapitola 5

Experimentální výsledky

5.1 Testovací data a prostředí

Testovací data byla vytvořena pomocí generátoru náhodných booleovských funkcí [8] a booleovského minimalizátoru BOOM [9]. Vstupem generátoru jsou parametry booleovských funkcí (počet vstupů, počet výstupů a termů) a výstupem jsou PLA tabulky. Z těchto tabulek BOOM generuje matice, na kterých řeší problém pokrytí - na základě jeho řešení BOOM generuje minimalizované tabulky PLA.

Pro účely měřený byla vygenerována sada padesáti PLA tabulek pro 25 vstupů, 25 výstupů a 25(26) termů. Z těchto tabulek byly pomocí BOOMu vygenerovány matice. Tyto matice poté byly pro účely měření upraveny tak, aby měly stejné parametry (zmenšena velikost, upravena hustota) - nejedná se tedy o reálná testovací data.

V části 5.5.3 jsou pro porovnávání řešičů použity neupravené matice vygenerované nástrojem BOOM. Část PLA tabulek, podle kterých jsou matice generovány, jsou MCNC benchmarky []. Zbytek byl vytvořen generátorem náhodných funkcí. Generování matic nástrojem BOOM proběhlo s nastavením iterací 10.

Z tabulky je jasně vidět, že náhodně vygenerované matice mají málokdy hustotu (poměr počtu jedniček k počtu prvků matice) větší jak 15%. Tento fakt byl zohledněn i při dalším měření, kdy se závislosti na velikosti matice měří při hustotě matice 7%.

Veškerá měření byla prováděna na sestavě: Intel Core Duo T2400 1.83 GHz, 2 GB RAM.

Pro všechny grafy uvedené v této kapitole platí, že spojnice mezi naměřenými body jsou uvedeny z důvodu přehlednosti grafů.

5.2 Chování Lagrangeovské relaxace

Před zkoumáním chování celého algoritmu Branch and Bound Lagrange (4.1.0.1) byla zkoumána samotná Lagrangeovská relaxace. První měření se zabývají chováním relaxace v závislosti na jejích parametrech (viz 4.2.1.1). Další měření se zabývají měření výkonnosti relaxace v závislosti na parametrech matice. Také byl změřen vliv odstranění dominancí z matice na výsledky relaxace.

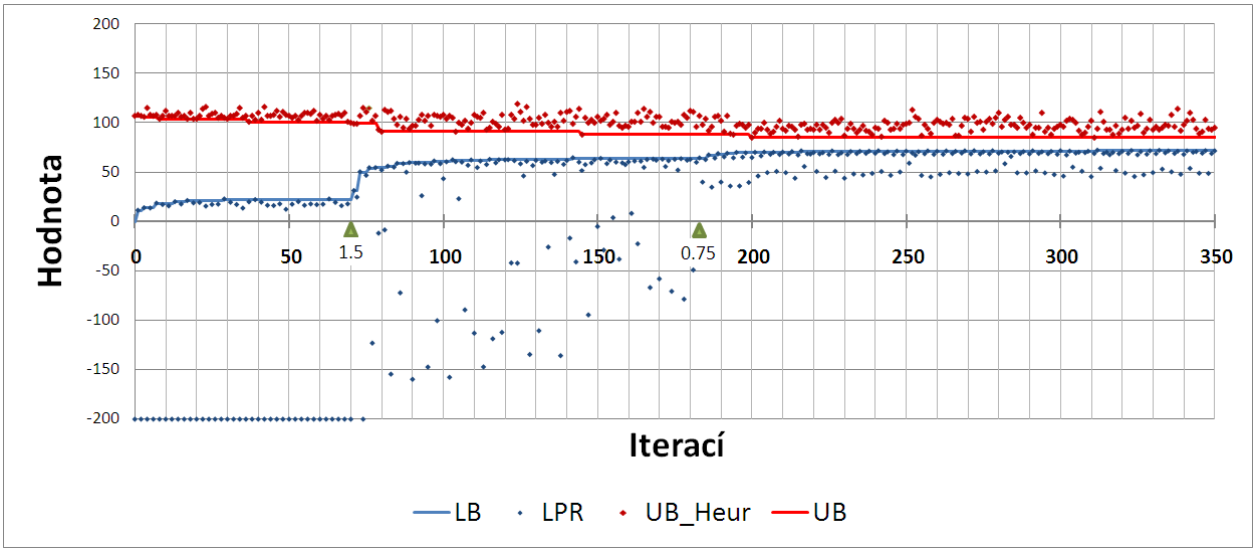
	PLA Tabulka					Matice		
název	vst.	výst.	termů	idc [%]	odc [%]	řádků	sloupců	hustota [%]
08x08x08	8	8	8	0	0	31	16	13.5
08x12x12	8	12	12	0	0	63	40	9.5
100x01x100	100	1	100	0	0	53	209	14.2
10x01x200	10	1	200	0	0	98	147	4.3
10x10x10	10	10	10	0	0	47	38	11.1
10x10x20	10	10	20	0	0	95	96	5.7
10x10x30	10	10	30	0	0	141	153	4.4
12x12x12	12	12	12	0	0	77	81	10.1
20x05x20	20	5	20	0	0	53	85	11.4
20x05x20_10	20	5	20	10	0	57	91	9.8
20x05x20_20	20	5	20	20	0	50	77	7.4
50x50x05_10	50	50	5	10	0	119	27	15.4
15x15x50_30_80	15	15	50	30	80	52	59	5.7
15x15x50_50_60	15	15	50	50	60	127	87	2.3
50x10x50_10_80	50	10	50	10	80	43	89	12.5
50x10x50_50_70	50	10	50	50	70	79	118	3.4
50x10x50_60_30	50	10	50	60	30	197	90	1.8
apex3	54	50	1036	89	93	1022	601	0.5
bw	5	28	93	27	81	247	79	8.1
cps	24	109	855	75	94	946	285	0.8
ex1010	10	10	1297	12	84	742	328	0.3
mainpla	27	54	2507	68	82	6416	549	1.2
pdc	16	40	822	34	79	520	249	1.1
spla	16	46	837	54	90	749	353	0.6
table3	14	14	1686	57	70	643	264	0.6
xparc	41	73	3226	84	85	2653	508	0.7

Tabulka 5.1: Vlastnosti matic reálných dat

5.2.1 Vývoj horní a spodní meze

První graf 5.1 zobrazuje vývoj základních hodnot při běhu Lagrangeovské relaxace. Z grafu je patrné, že při větších hodnotách koeficientu kroku dochází v každé iteraci k velkým změnám hodnot LRP (až k hodnotě -3300 - v grafu pro přehlednost není znázorněno). Z grafu je dále patrné, že změny v LB se většinou projeví brzy po změně koeficientu kroku. To vede k domněnce, že by bylo možné zkrátit výpočetní čaa nastavením menší hodnota iterací N_t (pro tento případ bylo $N_t = 40$), která udává, jak kolik iterací relaxace se nesmí změnit hodnota LB než dojde ke snížení (půlení) koeficientu kroku. Naopak by mohlo být výhodné umožnit nastavení minimálního koeficientu kroku t_{min} na menší hodnotu než je v tomto případě použitých 0,5. Oběma těmto úvahám se věnuje část kapitoly 5.2.3. Ze zdrojových dat lze dále zjistit, že k nalazení lepší hodnoty UB dochází nezávisle na hodnotě ceny řešení LRP. Toto vede k úvaze, že by výpočetní čas mohl být snížen, pokud by se Lagrangeovská heuristika nespouštěla v každé iteraci relaxace. Tomu se věnuje kapitola 5.2.5. Průběh byl

naměřen na matici o velikosti 150 řádků × 235 slupců o hustotě 5,5%.



Obrázek 5.1: Vývoj mezí

5.2.2 Konvergence multiplikátorů

Tato část kapitoly se zaměří na sledování vývoje hodnot Lagrangeovských multiplikátorů v čase. Ze zřejmých důvodů nelze do grafu zahrnout průběh všech multiplikátorů pro zadaný problém.

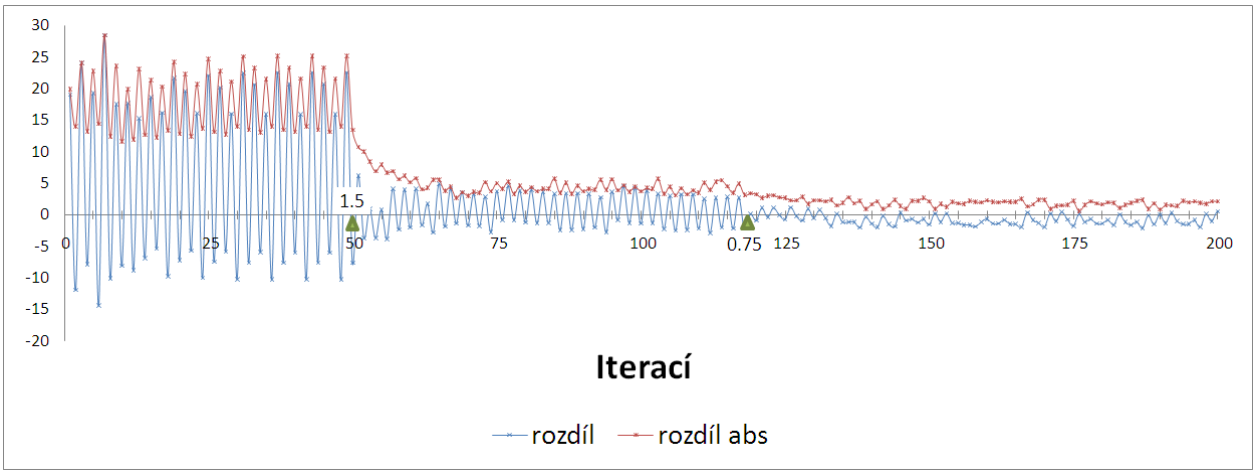
Následující tabulka zobrazuje výsledné hodnoty multiplikátorů pro stejný problém, ale s různými hodnotami nastavení iniciálních hodnot multiplikátorů λ_0 (Pro první řádek byly hodnoty nastaveny λ_0 tak, že odpovídají nejmenší ceně sloupců pokrývající daný řádek). Z tabulky je patrné, že výsledné hodnoty se od sebe příliš neliší a relaxace nastavuje multiplikátory v závislosti na struktuře problému a nezáleží na iniciálním nastavení multiplikátorů. Hodnoty byly naměřeny na matici o velikosti 25 řádků × 25 slupců o hustotě 15%.

$\lambda(1)$	$\lambda(2)$	$\lambda(3)$	$\lambda(4)$	$\lambda(5)$	$\lambda(6)$	$\lambda(7)$	$\lambda(8)$	$\lambda(9)$	$\lambda(10)$	$\lambda(11)$	$\lambda(12)$	$\lambda(13)$
0.43	2.84	3.43	3.55	0.52	1.16	0.00	1.58	2.35	0.32	0.60	1.17	1.08
0.00	3.17	3.14	3.79	0.52	0.84	0.00	1.86	1.77	0.05	0.85	0.91	1.46
0.32	2.80	3.29	3.53	0.64	0.86	0.00	1.35	2.11	0.02	0.52	1.10	0.86
0.32	2.98	3.26	3.53	0.64	0.86	0.00	1.35	2.11	0.02	0.52	1.10	0.86
0.74	2.92	3.18	3.46	0.64	1.94	0.00	1.13	1.90	0.00	0.21	0.62	0.75

Tabulka 5.2: Výsledné Lagrangeovské multiplikátory

Graf 5.2 zobrazuje vývoj hodnot multiplikátorů. Řada hodnot "rozdíl"označuje součet rozdílů hodnot multiplikátorů v dané iteraci k (λ_k) od konečných hodnot (λ_{final}). Řada hodnot "rozdíl abs"bere v úvahu absolutní rozdíly hodnot multiplikátorů. Podobně jako v grafu 5.1 je vidět změna v chování při změně koeficientu kroku - menší krok dovolí multiplikátorům, aby se nastavily na hodnoty které vedou na hodnoty redukovaných cen,

tak, že řešení LRP méně porušuje podmínky pokrytí. Pro měření byla použita stejná matice jako v předchozím měření.



Obrázek 5.2: Vývoj multiplikátorů

5.2.3 Multiplikátory kroku a iterace

Na základě úvahy v části 5.2.1 bylo změřeno chování Lagrangeovské relaxace s různými parametry počátečního koeficientu kroku t_{init}, minimálního koeficientu kroku t_{min} a počtu iterací změny kroku N_t - viz tabulka 4.1. Následující tabulka 5.3 ukazuje, jak se změnil výpočetní čas, počet iterací a výsledné hodnoty LB a UB vztažené k ceně optimálního řešení problému . Do časů je zahrnuto úvodní odstraňování dominancí.

Měření byla pro každý případ provedena na 50 maticích o velikosti 150×300 a hustotě 7% (uvedené hodnoty jsou průměrné).

případ	t_{init}	t_{min}	N_T	čas	LB(%)	UB(%)	iterací
1	10	0.1	40	0.25	88.62	103.78	709
2	5	0.1	20	0.19	88.34	104.14	356
3	3	0.1	10	0.17	87.71	105.26	187
4	3	0.01	10	0.19	88.66	104.08	310
5	3	0.01	5	0.17	87.96	104.42	164
6	2	0.1	5	0.15	87.14	106.06	88
7	2	0.01	20	0.16	87.93	105.02	137
8	2	0.001	40	0.32	89.19	102.80	1284
9	2	0.001	5	0.17	88.03	104.51	190

Tabulka 5.3: Multiplikátory kroku a počet iterací

Z naměřených hodnot je vidět, že úvaha je pravdivá - vyplatilo se snížit počet iterací a zvýšit počet změn koeficientu kroku. Viz. případy 2 (koeficient kroku byl půlen dvakrát) a 4 (koeficient kroku byl půlen pětkrát) , kde došlo ke zvýšení spodní meze při snížení počtu

iterací (obdobně případy 3 a 9). Pro další měření budou použity hodnoty z případu 7, tj. $t_{init} = 2$, $t_{min} = 0.01$ a $N_t = 5$.

5.2.4 Výběr heuristiky

Následující tabulka udává hodnoty naměřené pro různé varianty Lagrangeovských heuristik (popsaných v 3.4.5), které z nepřípustného řešení LRP tvoří přípustné řešení problému pokrytí. Do časů je zahrnuto úvodní odstraňování dominancí. Měření bylo pro každou variantu provedeno na padesáti maticích o velikosti 150×300 a hustotě 7% (uvedené hodnoty jsou průměrné).

varianta	čas	LB(%)	UB(%)	iterací
1	0.17	87.17	104.10	137
2	0.17	87.16	105.58	137
3	0.16	87.14	103.67	139
4	0.18	87.17	104.58	138
5	0.17	87.17	104.67	136

Tabulka 5.4: Výběr heuristiky

Z naměřených hodnot vyplývá, že na zvolené variantě heuristiky příliš nezáleží. Pro další měření byla zvolena varianta 3.

5.2.5 Četnost spouštění kompletující heuristiky

Na základě úvahy v části 5.2.1 bylo změřeno chování Lagrangeovské relaxace s různými prodlevami ve spouštění Lagrangeovské heuristiky (která vylepšuje hodnotu UB) N_h. Následující tabulka 5.5 ukazuje, jak se změnil výpočetní čas a výsledné hodnoty LB a UB vztažené k ceně optimálního řešení problému. Do časů je zahrnuto úvodní odstraňování dominancí.

případ	N_h	čas	LB(%)	UB(%)	iterací
1	1	0.16	87.17	104.10	137
2	2	0.16	87.15	104.41	137
3	5	0.16	87.19	106.33	139
4	10	0.16	87.21	108.18	138
5	15	0.16	87.23	109.50	139
6	20	0.16	87.18	108.92	137

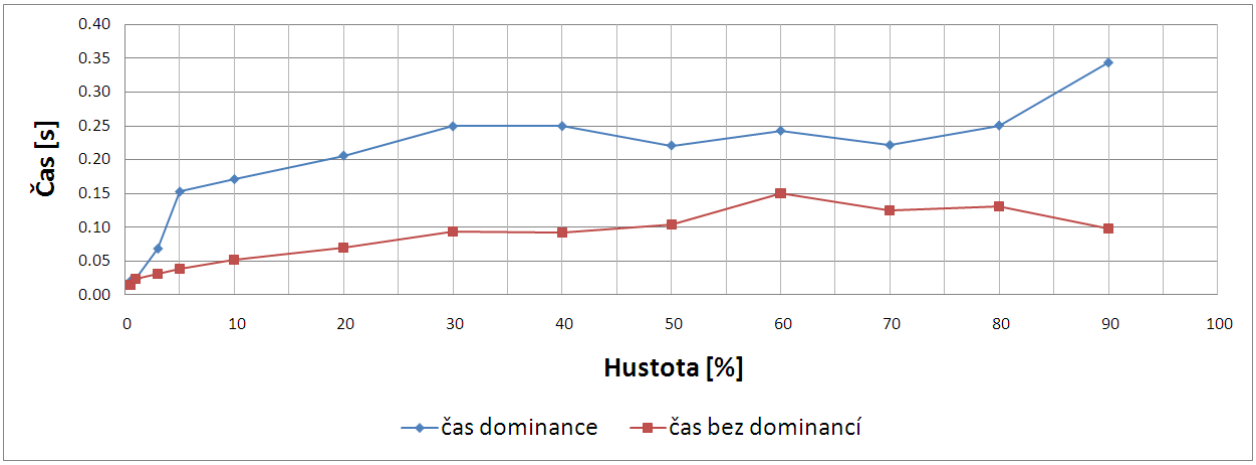
Tabulka 5.5: Četnost spouštění Lagrangeovské heuristiky

Měření bylo pro každou hodnotu četnosti provedeno na padesáti maticích o velikosti 150×300 a hustotě 7% (uvedené hodnoty jsou průměrné).

Z naměřených hodnot je patrné, že pokud se heuristika nespouští v každé iteraci relaxace, jsou nalezena horší přípustná řešení (nižší hodnoty UB). Navíc vliv na výpočetní čas je minimální. Proto pro další měření je heuristika spouštěna v každé iteraci relaxace, tedy $N_h = 1$.

5.2.6 Závislost běhu relaxace na hustotě matice

Grafy 5.3 a 5.4 zobrazují závislost trvání relaxace a kvality nalezeného heuristického řešení a spodní meze na hustotě matice. Cílem je zjistit, jestli má odstranění dominancí před začátkem relaxace vliv na celkový průběh relaxace. Pokud by vliv byl malý, mohlo by být výhodnější ve výsledném řešiči vynechat odstraňování dominancí pro každé volání rekurzivní funkce řešiče. Tím by došlo k ušetření výpočetního času. Měření bylo pro každou hodnotu hustoty provedeno na padesáti maticích o velikosti 150×300 (uvedené hodnoty jsou průměrné).



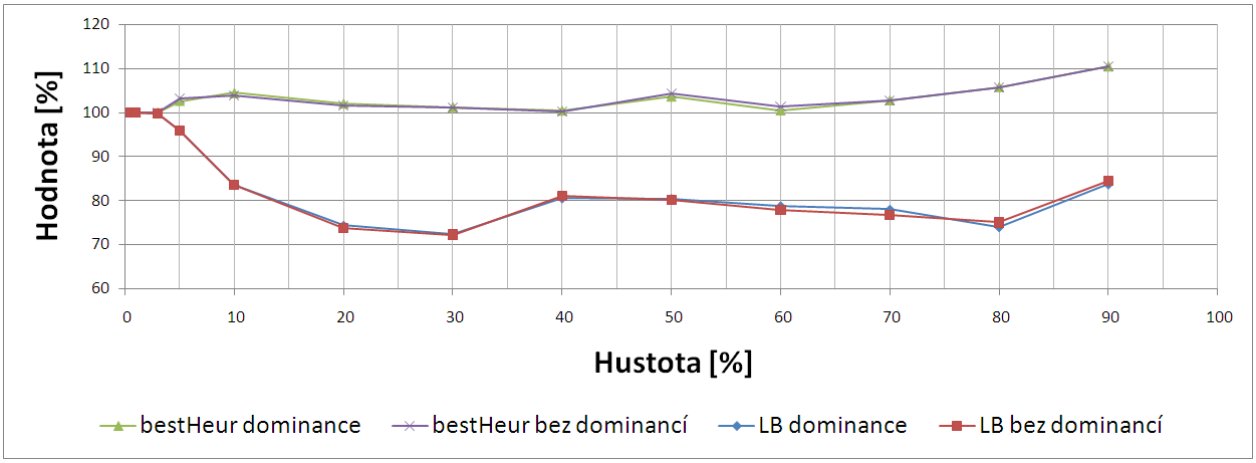
Obrázek 5.3: Trvání relaxace pro různé hustoty matice

Graf 5.3 zobrazuje, jak se s zvětšující hustotou matice roste výpočetní čas. Jelikož při vysoké hustotě jeden sloupec pokrývá mnohem více řádku, při jeho zařazení nebo nezařazení do LRP je ovlivněno mnoho multiplikátorů. To vede k tomu, že se daří často nacházet nové hodnoty LB (v některých případech docházelo ka nastavování nové hodnoty LB v každé iteraci relaxace. Těchto iterací bylo 10000, kdy byla relaxace zastavena.). Pro matice s větší hustotou jak 60% platí, že cena optimálního řešení je velmi nízká (pro pokrytí je potřeba málo sloupců). To vede k tomu, že k nalezení dostatečné hodnoty LB dochází dříve. Měření proběhlo na stejných maticích jako v měření pro graf 5.3.

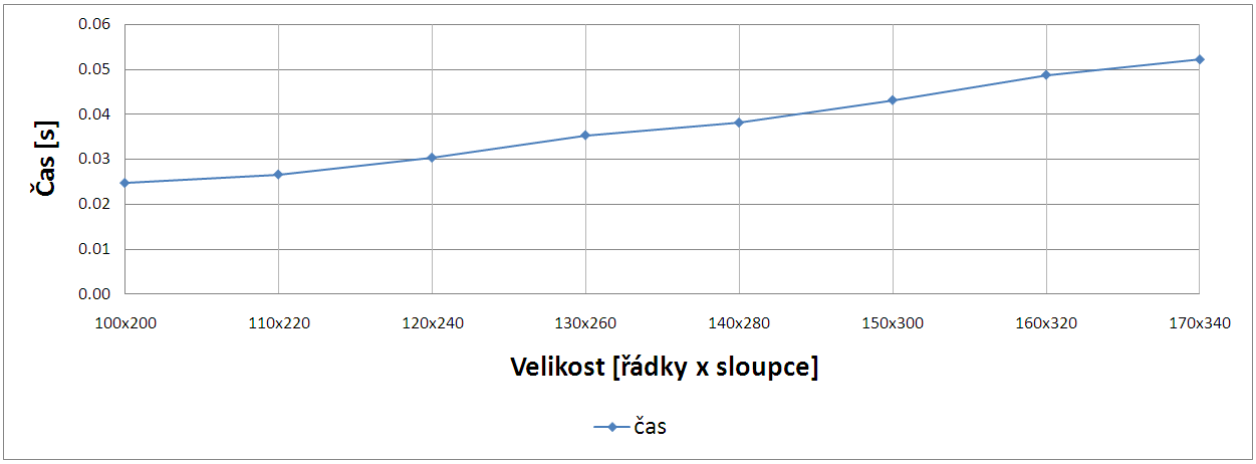
Graf 5.4 zobrazuje, jak se s zvětšující hustotou matice snižuje kvalita nalezené spodní meze. Jelikož pro husté matice je cena optimálního řešení nízká a hodnota LB není celočíselná, je v grafu uvedena hodnota LB zaokrouhlená nahoru na celé číslo. Tato hodnota se totiž v relaxaci využívá k uršní, jestli se má relaxace zastavit nebo ne. Z grafu je také vidět, že odstraňování dominancí má na kvalitu nalezeného heuristického řešení a spodní meze minimální vliv.

5.2.7 Závislost běhu relaxace na velikosti matice

Grafy 5.5 a 5.6 zobrazují závislost trvání relaxace a kvalitu nalezených mezí na velikosti matice. Měření byla provedena na maticích o hustotě 7% (pro každou velikost bylo vygenerováno 50 různých matic - uvedené hodnoty jsou průměrné).



Obrázek 5.4: Výsledky relaxace pro různé hustoty matice

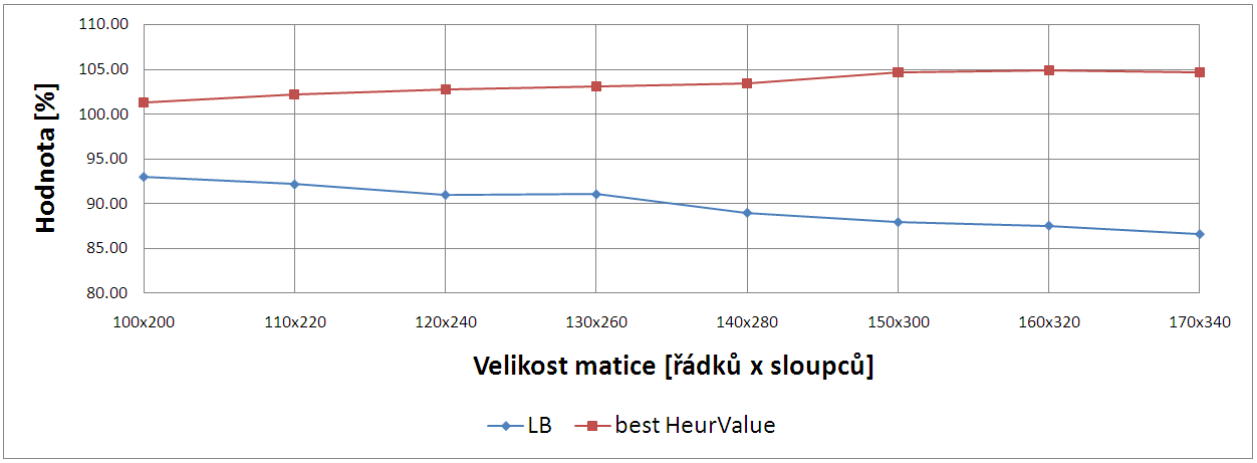


Obrázek 5.5: Trvání relaxace pro různé velikosti matice

Graf 5.6 zobrazuje, jak se s zvětšující velikostí matice snižuje kvalita nalezené spodní meze. Pro výsledný řešič to znamená, že pro veliké matice nebude relaxace dostatečná pro určení spodní meze a řešič tak bude muset prohledávat větší část stavového prostoru, čímž dojde ke zvýšení výpočetního času.

5.3 Duální relaxace

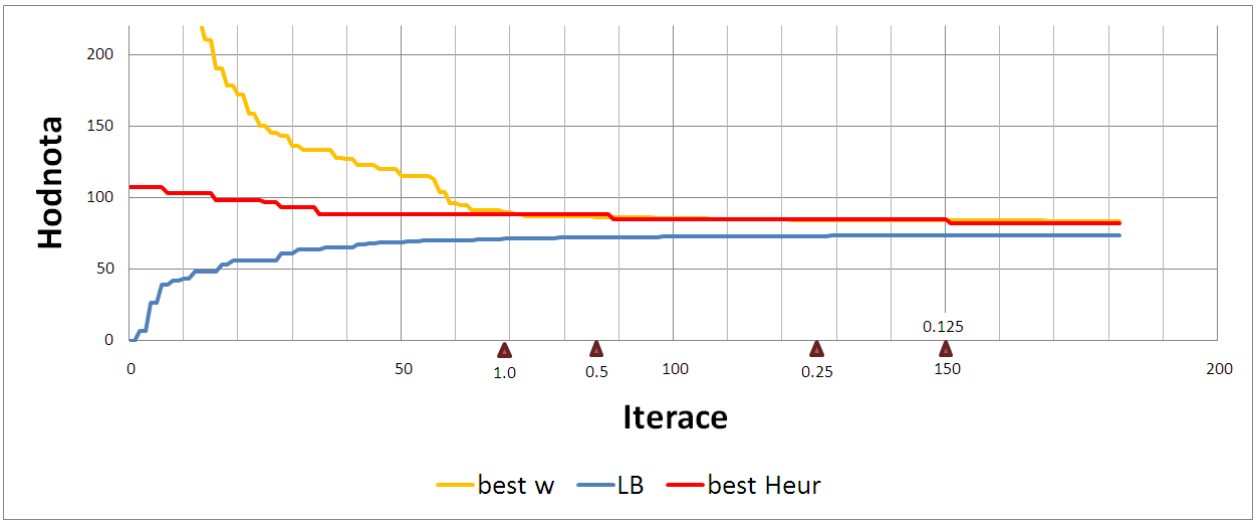
Tato část je věnována chování měření duální Lagrangeovské relaxace. Cílem je zjistit, o kolik je duální relaxace výpočetně náročnější než jednoduchá a jak moc se projeví použití ceny řešení duálního problému jakožto horní meze ve vztahu pro úpravu multiplikátorů na výsledné hodnotě LB.



Obrázek 5.6: Výsledky relaxace pro různé velikosti matice

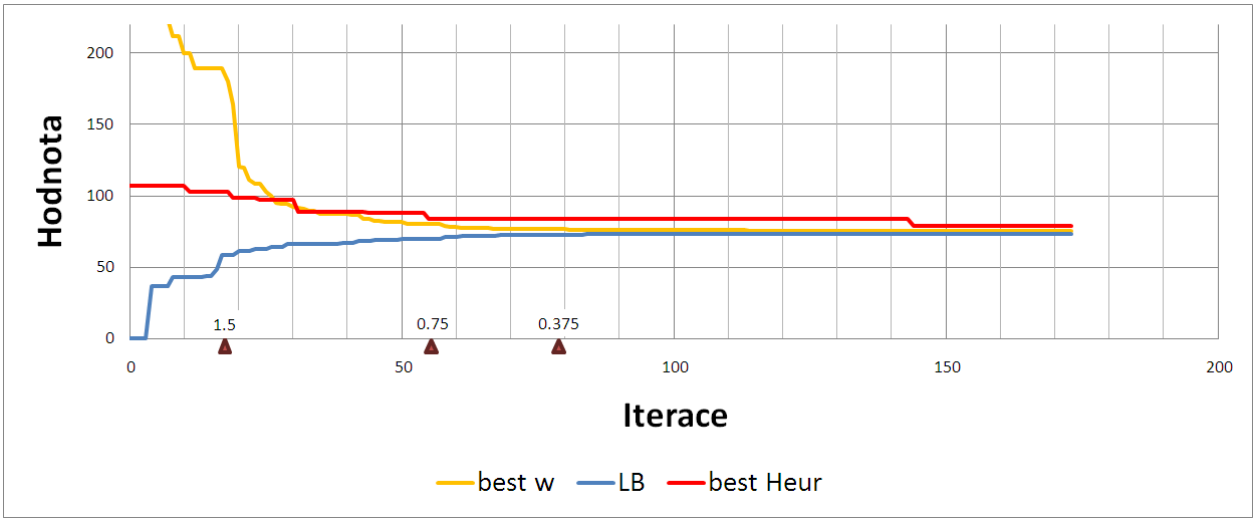
5.3.1 Vývoj spodní a horní meze

Z grafu 5.7 lze vyčíst, jakým způsobem je vylepšována cena duálního řešení w_{LDP} tak, aby se co nejvíce přiblížila hodnotě LB. Také je zřejmé, že pro iniciální duální koeficient kroku $t_{Dinit} = 2$ nedošlo k nalezení koeficientů takových, aby cena duálního řešení byla nižší, než cena nejlepšího řešení nalezeného Lagrangeovskými heuristikami spouštěnými v rámci relaxace. Pro měření byla použita stejná matice jako pro měření v 5.2.1.



Obrázek 5.7: Vývoj mezí duální relaxace

Graf 5.8 zobrazuje situaci, kdy byl zvolen vyšší duální koeficient kroku $t_{Dinit} = 3$. Nalezené duální řešení w_{LDP} poskytuje přesnější hodnotu horní meze UB pro Lagrangeovskou relaxaci a díky tomu je teoreticky možné nalézt řešení LRP s vyšší cenou a tak zvýšit LB. Hodnoty nejsou relativní vůči ceně optimálního řešení.



Obrázek 5.8: Vývoj mezí duální relaxace

5.3.2 Srovnání jednoduché a duální relaxace

Následující tabulka zobrazuje srovnání vybraných případů jednoduché a duální Lagrangeovské relaxace spouštěné s různými parametry t_{Dinit}, t_{Dmin} a N_{Dt}. Z hodnot lze vyčíst, že použití duální relaxace nevede k nalezení lepších hodnot LB. Navíc přepočítávání duálních koeficientů je výpočetně náročné a prodlužuje se tak celkový výpočetní čas. Pro další měření tedy bude použita jednoduchá Lagrandeovská relaxace bez počítání duálního problému. Měření byla pro každý případ provedena na 50 maticích o velikosti 150×300 a hustotě 7% (uvedené hodnoty jsou průměrné).

případ	t_{Dinit}	t_{Dmin}	N_{Dt}	čas	LB(%)	BHV(%)*	iterací
Simple	X	X	X	0.04	87.91	104.69	135
Dual1	10	0.1	40	0.38	87.95	102.45	1249
Dual2	10	0.1	5	0.09	87.74	104.06	209
Dual3	5	0.1	40	0.36	87.85	102.57	1199
Dual4	5	0.1	5	0.10	87.78	104.20	247
Dual5	3	0.1	40	0.52	88.02	102.11	1813
Dual5	3	0.1	5	0.13	87.94	102.23	353
Dual5	3	0.01	40	0.61	88.04	101.91	2134
Dual5	3	0.01	5	0.17	87.97	102.58	515

Tabulka 5.6: Srovnání jednoduché a duální relaxace

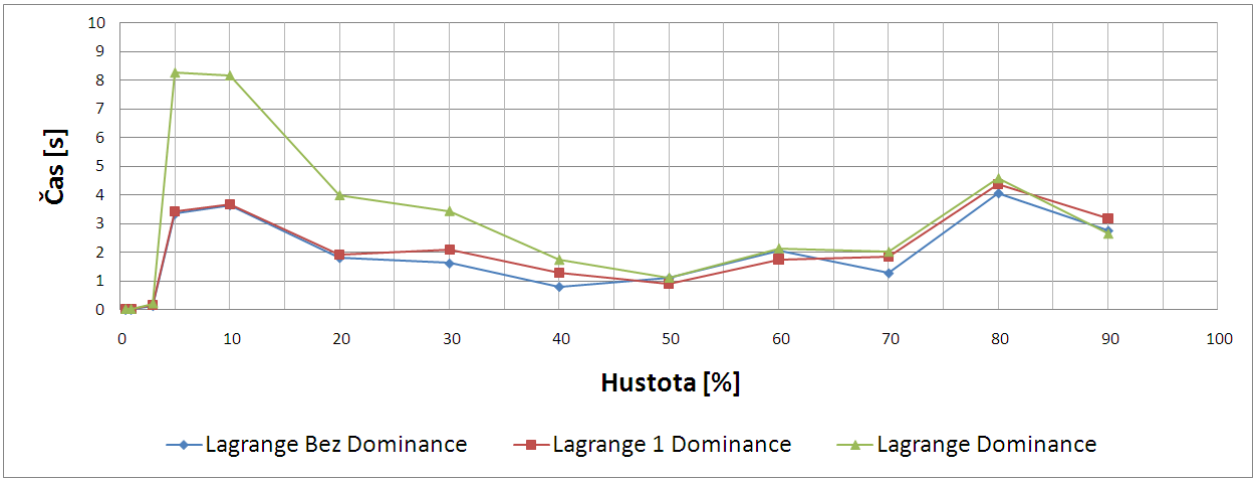
* - BHV je nejnižší cena řešení nalezeného Lagrangeovskými heuristikami.

5.4 Chování řešiče

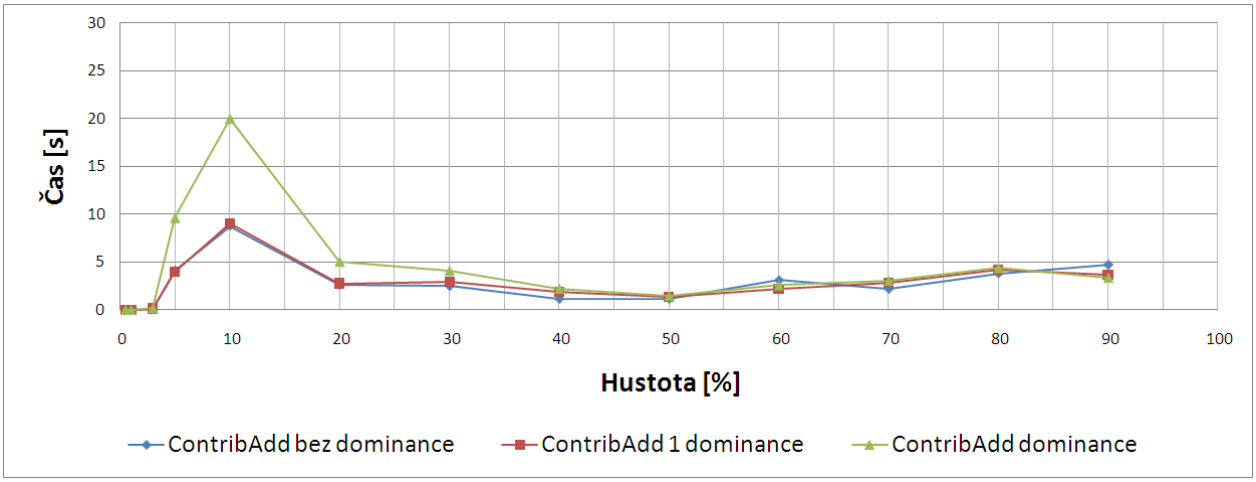
Tato část je věnována měření chování celkového řešiče, který využívá informace o spodní mezi problému z Lagrangeovské relaxace.

5.4.1 Závislost běhu řešiče na hustotě matice

Grafy 5.9 a 5.10 zobrazují srovnání běhu algoritmu v šesti variantách, které se liší v četnosti odstraňování dominancí pro různé hustoty matice a ve způsobu výběru sloupce. Varianty odstraňování dominancí jsou tři. V první variantě je odstraňování dominancí provádí při každém volání metody řešiče. V druhé variantě se odstraňování dominancí provede pouze při prvním volání, v ostatních se pouze kontroluje, zda podproblém neobsahuje esenciální sloupce. V poslední variantě k odstraňování dominancí nedochází nikdy, pouze se kontrolují esenciální sloupce. Pro měření byly použity stejné matice jako pro měření v 5.2.6. Tyto tři varianty jsou spouštěny se dvěma způsoby vybírání sloupců do řešení - pomocí heuristiky ContribAdd a pomocí informací získaných v Lagrangeovské relaxaci. Cílem tohoto měření je ověřit, která varianta je výhodnější a také potvrdit výsledky měření v části 5.2.6 - tedy ukázat, že se odstraňování dominancí při každém volání rekurzivní funkce nevyplatí.



Obrázek 5.9: Závislost doby běhu řešiče na hustotě matice (Lagrange)

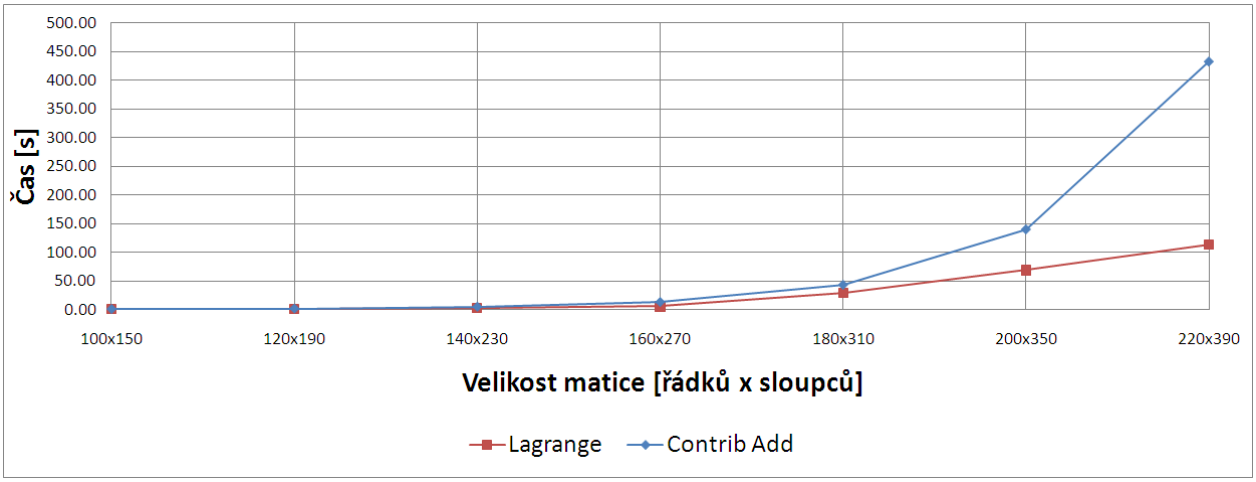


Obrázek 5.10: Závislost doby běhu řešiče na hustotě matice (ContribAdd)

Pro matice s hustotou do tří procent je čas řešení nízký - to je způsobeno značným množstvím esenciálních sloupců. S přibývající hustotou se cyklické jádro matic zvětšuje, a tak se zvyšuje i doba běhu. Okolo 10% hustoty trvá řešení nejdéle. Další zvyšování hustoty má za následek to, že každý sloupec pokrývá více řádků. Při jeho zařazení do řešení se tak problém zmenší více než pro řidší matice a je zapotřebí méně volání rekurzivní funkce. Tím se celková doba běhu algoritmu snižuje. Pro husté matice (hustota větší jak 50 %) příliš nezáleží na volbě způsobu odstraňování dominancí a způsobu výběru sloupce. Pro řidší matice se nejvíce vyplatí způsob vybírání sloupců založený na informacích Lagrangeovské relaxace bez odstraňování dominancí.

5.4.2 Závislost běhu řešiče na velikosti matice

Graf 5.11 zobrazuje závislost doby běhu řešiče na velikosti matice ve dvou variantách, které se stejně jako v předešlém měření liší ve způsobu výběru sloupců přidávaných do řešení. K odstraňování dominancí došlo pouze při prvním volání rekurzivní funkce řešiče. Měření byla provedena na maticích o hustotě 7% - pro každou velikost bylo vygenerováno padesát instancí (uvedené hodnoty jsou průměrné).



Obrázek 5.11: Porovnání závislosti doby běhu řešičů na velikosti matice

	Lagrange		ContribAdd	
velikost matice	čas [s]	stavů	čas [s]	stavů
100x150	0.9	124	1.7	220
120x190	1.7	173	2.5	254
140x230	3.5	304	5.7	484
160x270	6.1	447	13.9	925
180x310	29.8	1962	43.7	2547
200x350	69.2	3677	140.1	6917
220x390	114.6	5002	432.0	18668

Tabulka 5.7: Porovnání řešičů pro různé velikosti matic

Z grafu a tabulky je vidět, že při vybírání sloupců pomocí heuristiky ContribAdd doba běhu roste rychleji než při vybírání sloupců pomocí informací z Lagrangeovské relaxace. Také je vidět, že doba běhu řešiče B&B Lagrange se pro danou hustotu matic (7%) roste exponenciálně se zvětšující se maticí.

5.4.3 Závislost běhu řešiče na nastavení Lagrangeovské relaxace

Tabulka 5.8 zobrazuje, jak se mění výpočetní čas v závislosti na parametrech Lagrangeovské relaxace. Cílem je zjistit, jestli z hlediska celkové doby běhu řešiče není vyhodnější, pokud jsou paramtery relaxace nastaveny tak, aby proběhla v rychlejším čase za cenu nižší hodnty nalezené spodní meze. Měření byla provdena pro matice o velikosti 130×260 a hustotě 7% - pro každý případ bylo vygenerováno 50 instancí (uvedené hodnoty jsou průměrné).

případ	t_{init}	t_{min}	N_t	čas	stavů
1	10	0.1	40	6.54	122
2	10	0.1	20	4.37	142
3	10	0.1	10	2.97	151
4	10	0.1	5	2.70	216
5	5	0.1	20	4.05	142
6	5	0.1	10	2.85	151
7	5	0.1	5	2.59	213
8	3	0.1	20	3.91	141
9	3	0.1	10	2.93	173
10	3	0.1	5	2.27	197
11	2	0.1	20	3.22	128
12	2	0.1	10	2.36	143
13	2	0.1	5	1.96	171
14	2	0.01	20	4.39	131
15	2	0.01	10	3.08	151
16	2	0.01	5	2.49	193
17	1.5	0.1	20	2.96	136
18	1.5	0.1	10	2.20	156
19	1.5	0.1	5	2.07	206

Tabulka 5.8: Vliv nastavení Lagrangeovské relaxace na dobu běhu řešiče

Z naměřených časů lze vyčíst, že parametry relaxace byly v části 5.2.3 nejsou z hlediska výsledného řešiče optimální. Proto byly pro další měření zvoleny parametry jako pro případ 13. V jiných případech sice byla prozkoumána i menší část stavového prostoru, ale relaxace se provádí pro každý zkoumaný stav, proto se vyplatí takové nastavení relaxace, pro které tyto relaxace netrvají zbytečně dlouho.

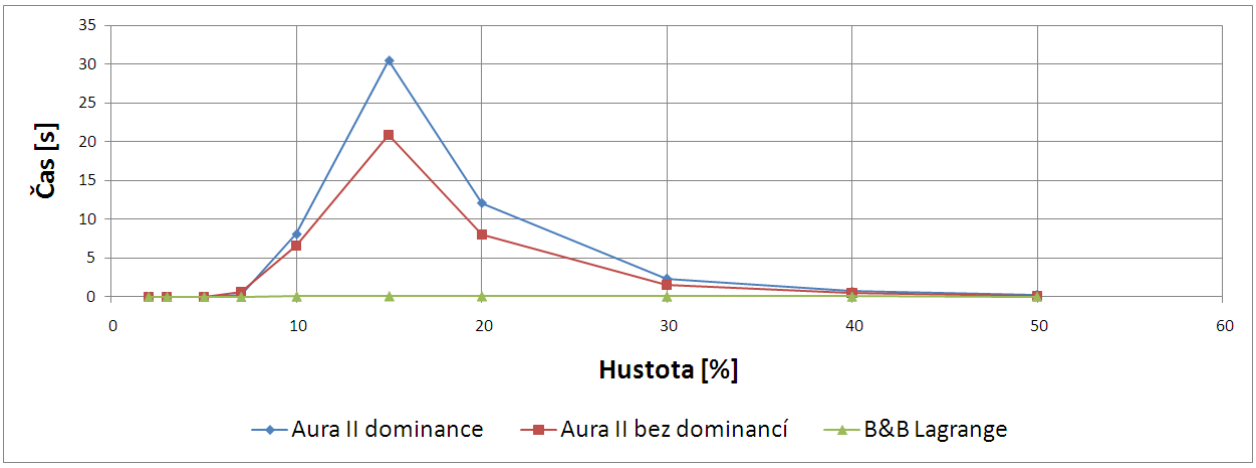
5.5 Srovnání řešičů problému prokrytí

Tato část se zabývá porováním vytvořeného řešiče a Lagrangeovské relaxace s již existujícím řešičem problému pokrytí AURA II a heuristikami ContribAdd a Servítovy heuristiky.

5.5.1 Porovnání B&B Lagrange s AURA II

Tato část se věnuje porovnání výsledného řešiče B&B Lagrange s AURA II.

Následující grafy 5.12 a 5.13 zobrazují doby běhu řešičů pro různě veliké matice a pro matice s různou hustotou. Závislosti na hustotě byly měřeny pro matice o velikosti 60×70. Závislosti na velikosti byly měřeny pro matice o hustotě 7%. Pro každé měření bylo vygenerováno 50 instancí matic - uvedené hodnoty jsou průměrné.



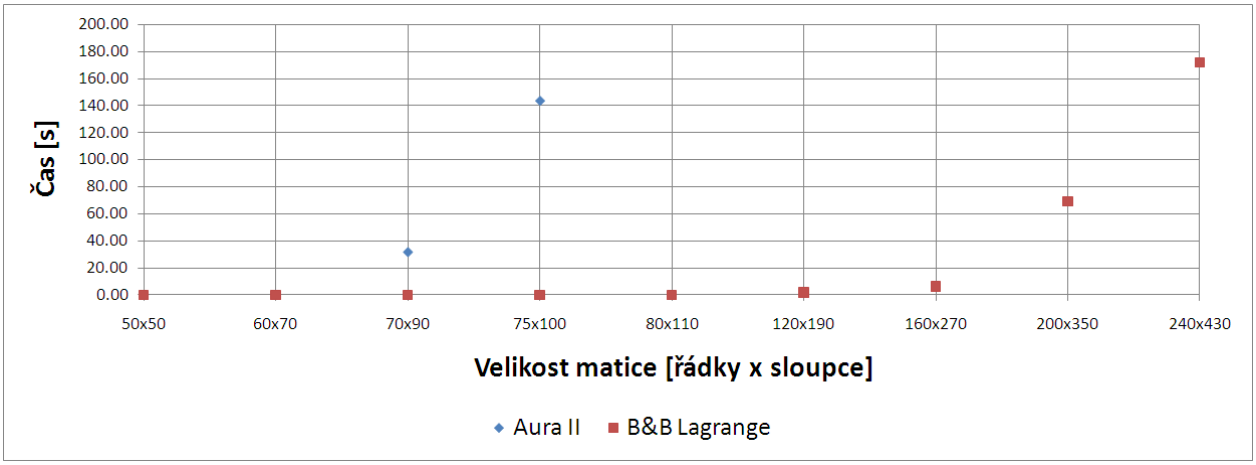
Obrázek 5.12: Porovnání závislostí doby běhu řešičů na hustotě matice

	AURA II dom.		AURA II bez dom.		B&B Lagrange	
hustota [%]	čas [s]	stavů	čas [s]	stavů	čas [s]	stavů
2	<0.01	1	<0.01	6	<0.01	1
3	<0.01	3	<0.01	16	<0.01	1
5	<0.01	38	0.58	162	<0.01	2
7	0.28	849	0.02	2267	0.01	6
10	8.10	15722	6.64	25516	0.06	21
15	30.49	57608	20.93	89941	0.09	27
20	12.11	25083	7.99	37554	0.08	21
30	2.37	5069	1.59	7634	0.06	13
40	0.70	1250	0.45	1918	0.05	9
50	0.27	368	0.15	539	0.04	6

Tabulka 5.9: Porovnání závislostí doby běhu řešičů na hustotě matice

Z tabulky 5.9 je vidět, jaký má hustota matice vliv na hledání minimálního pokrytí pomocí AURA II. Velikost matice je ale příliš malá na to, aby se hustota nějak projevila i na řešiči B&B Lagrange. Pokud by ale byla zvolena větší matice, doba běhu AURA II by byla nepřijatelně veliká - jak je vidět z grafu 5.13. Z tabulky je také vidět, že odstraňování dominancí sice vede na menší počet volání rekurzivnío algoritmu (méně prozkoumaných stavů ve stavovém prostou), nicméně celkově se odstraňování dominancí nevyplatí - trvá dlouho a algoritmus je tak celkově pomalejší. Z naměřených dat pro jednotlivé instance (veškerá

naměřená data jsou přiložena na CD) také vyplynulo, že AURA II v některých případech nenachází optimální řešení.



Obrázek 5.13: Porovnání závislostí doby běhu řešičů na velikosti matice

	AURA II bez dom.		B&B Lagrange	
velikost matice	čas [s]	stavů	čas [s]	stavů
50x50	<0.01	38	<0.01	1
60x70	0.32	1275	0.01	5
70x90	31.8	83 462	0.07	23
75x100	144	336 481	0.11	28
80x110	-	-	0.20	46
100x150	-	-	0.88	124
120x190	-	-	1.71	173
140x230	-	-	3.47	304
160x270	-	-	6.24	457
180x310	-	-	29.8	1 962
200x350	-	-	69.2	3 677
220x390	-	-	117	5 806
240x430	-	-	172	6 512

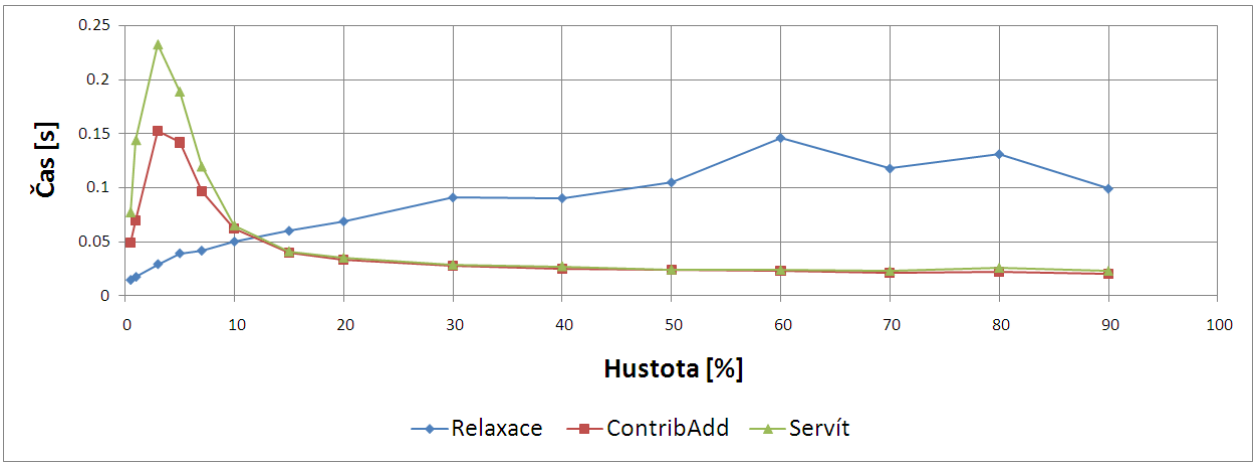
Tabulka 5.10: Porovnání závislostí doby běhu řešičů na velikosti matice

Z tabulky 5.10 a grafu 5.13 lze vyčíst, že doba běhu B&B Lagrange se pro danou hustotu matic (7%) roste exponenciálně se zvětšující se maticí. Pro B&B Lagrange je ale tento růst podstatně pomalejší.

5.5.2 Porovnání Lagrangeovské relaxace s heuristikami

Tato část se věnuje porovnání Lagrangeovské relaxace s heuristikami.

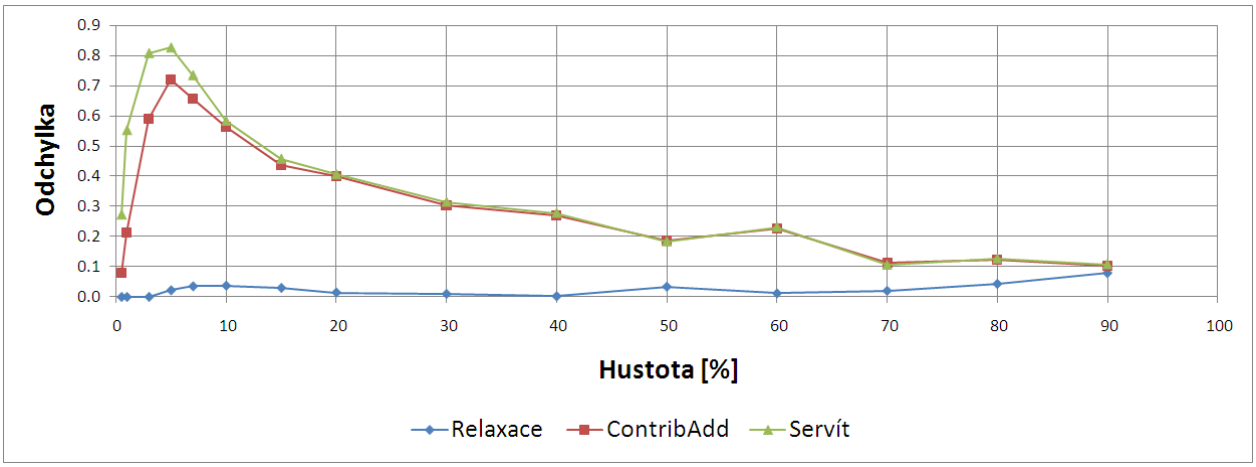
Graf 5.14 zobrazuje doby běhu heuristik a Lagrangeovské relaxace v závislosti na hustotě matice. Závislosti na hustotě byly měřeny pro matice o velikosti 150×300 - pro každý případ bylo vygenerováno 50 instancí (uvedené hodnoty jsou průměrné).



Obrázek 5.14: Porovnání závislostí doby běhu heuristik a relaxace na hustotě matice

Z grafu lze vyčíst, že Lagrangeovská relaxace je rychlejší než heuristiky především pro matice řidší jak 12%. U heuristik je klesání výpočetního času s rostoucí hustotou způsobené tím, že při vyšší hustotě se více zmenší matice problému s každým vybraným sloupcem. Chování relaxace v závislosti na hustotě matice je popsané v části 5.2.6.

Grafy 5.15 a tabulka 5.11 zobrazuje závislost odchylek nalezených řešení na hustotě matice. Závislosti byly měřeny pro matice o velikosti 150×300 - pro každý případ bylo vygenerováno 50 instancí (uvedené hodnoty jsou průměrné).



Obrázek 5.15: Porovnání závislostí odchylek řešení heuristik a relaxace na hustotě matice

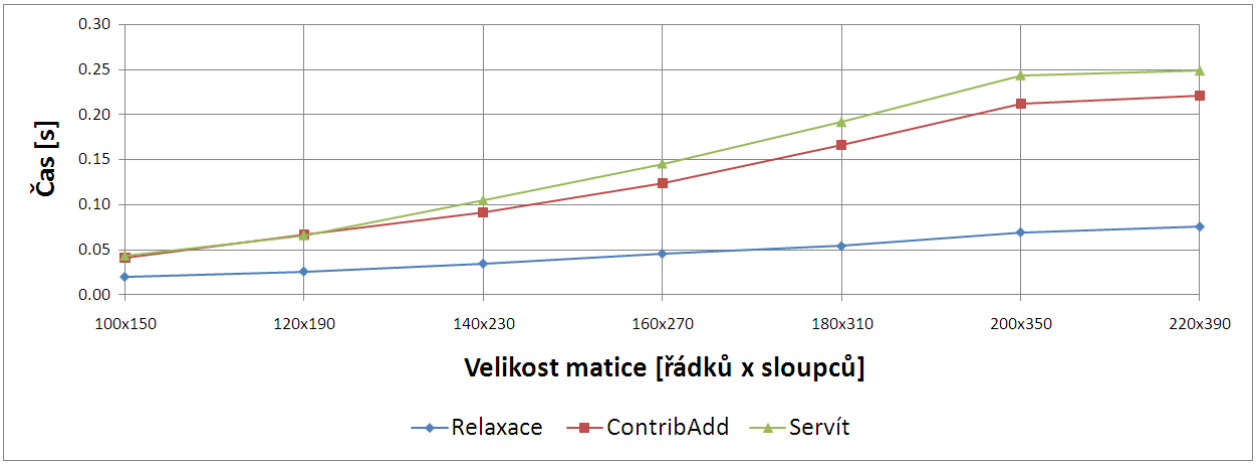
Z grafu a tabulky je patrné, že Lagrangeovská relaxace dává lepší výsledky než heuristiky především pro matice řidší jak 50% (ale za cenu delšího trvání - viz graf 5.15) . Zmenšování odchylek pro rostoucí hustotu je způsobeno tím, že v hustých maticích jeden sloupec pokrývá

	relaxace		ContribAdd		Servít	
hustota matice [%]	čas [s]	odchylka	čas [s]	odchylka	čas [s]	odchylka
0.5	0.01	0.00	0.05	0.08	0.08	0.27
1	0.02	0.00	0.07	0.21	0.14	0.55
3	0.03	0.00	0.15	0.59	0.23	0.81
5	0.04	0.02	0.14	0.72	0.19	0.83
7	0.04	0.03	0.10	0.66	0.12	0.73
10	0.04	0.04	0.06	0.56	0.07	0.58
15	0.04	0.03	0.04	0.44	0.04	0.46
20	0.04	0.01	0.03	0.40	0.04	0.41
30	0.05	0.01	0.03	0.30	0.03	0.31
40	0.05	0.00	0.02	0.27	0.03	0.28
50	0.05	0.03	0.02	0.19	0.02	0.18

Tabulka 5.11: Porovnání heuristik s relaxací pro matice s různou hustotou

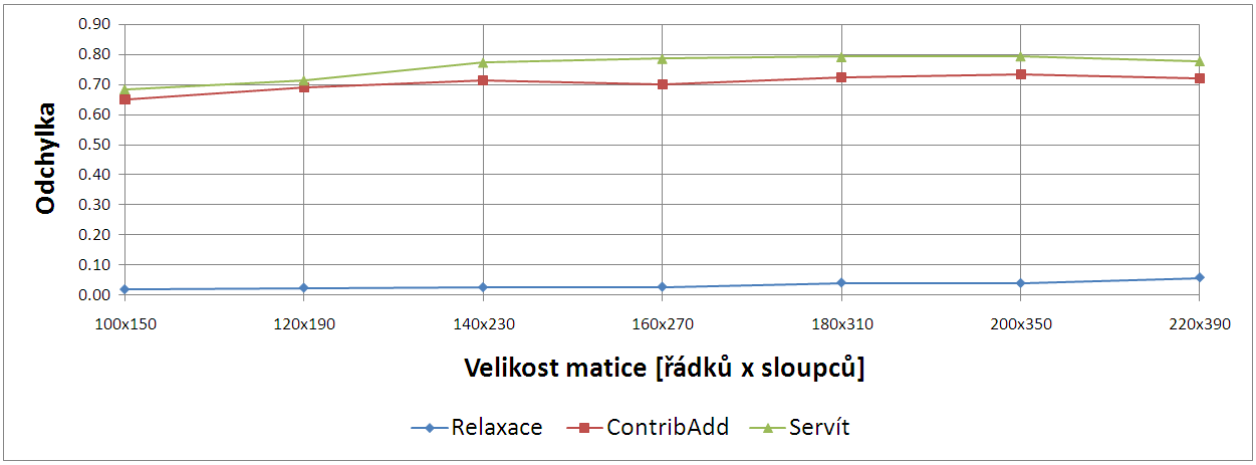
více řádků. Tím pádem je k pokrytí celé matice zapotřebí méně sloupců, a optimální cena je tak nižší než pro řídké matice. Chování relaxace v závislosti na hustotě matice je popsané v části 5.2.6.

Následující grafy a tabulky zobrazují závislosti doby běhu a odchylek nalezených řešení na velikosti matice. Měření se týkají heuristik a Lagrangeovské relaxacace. Závislosti byly měřeny pro matice o hustotě 7% a 20% - pro každý případ bylo vygenerováno 50 instancí (uvedené hodnoty jsou průměrné).



Obrázek 5.16: Porovnání závislostí doby běhu heuristik a relaxace na velikosti matice

Z grafu 5.16 lze vyčíst, že výsledná odchylka nalezeného řešení heuristik i relaxace pro danou hustotu matic není velikostí matice příliš ovlivněna. Z grafu 5.17 je vidět, že z hlediska výpočetního času jsou heuristiky citlivější na velikost matice (pro danou hustotu 7%). To je dáno tím, že pro větší počet sloupců trvá heuristikám déle vybírání sloupce, který bude zařazen do řešení.



Obrázek 5.17: Porovnání závislostí doby běhu heuristik a relaxace na hustotě matice

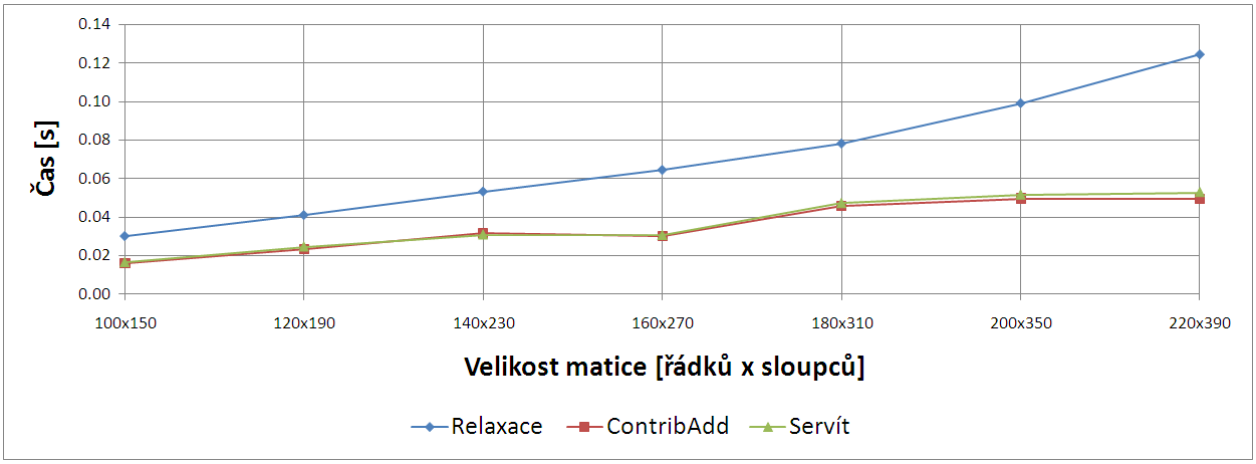
	Lag. relaxace		ContribAdd		Servít	
velikost matice	čas [s]	odchylka	čas [s]	odchylka	čas [s]	odchylka
100x150	0.02	0.02	0.04	0.65	0.04	0.68
120x190	0.03	0.02	0.07	0.69	0.07	0.71
140x230	0.03	0.03	0.09	0.71	0.11	0.77
160x270	0.05	0.03	0.12	0.70	0.15	0.79
180x310	0.05	0.04	0.17	0.72	0.19	0.79
200x350	0.07	0.04	0.21	0.73	0.24	0.79
220x390	0.05	0.06	0.22	0.72	0.25	0.78

Tabulka 5.12: Porovnání heuristik s relaxací pro matice s hustotou 7%

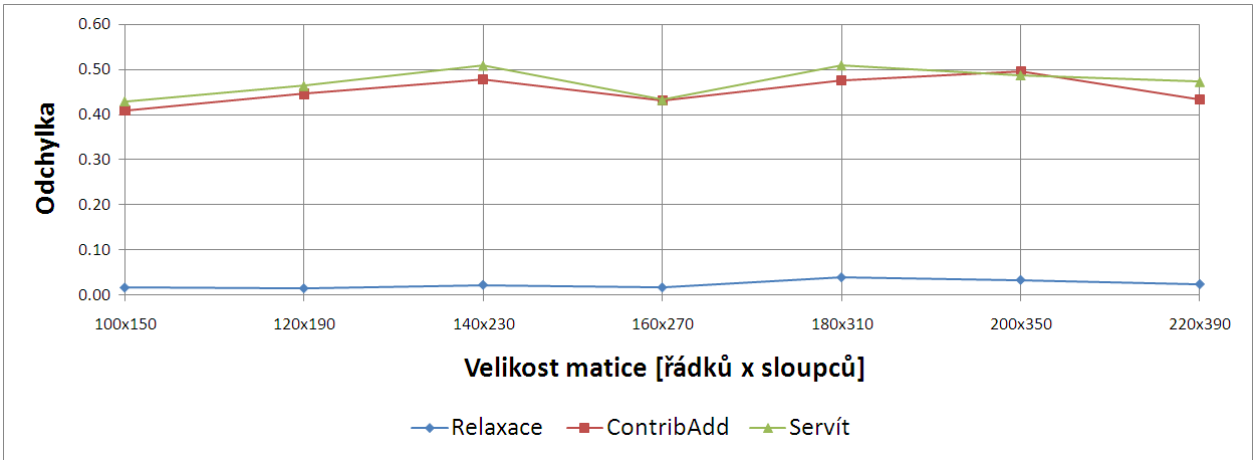
Následující grafy zobrazují závislost doby běhu heuristik a relaxace na velikosti matice pro hustotu matic 20%, kdy je podle grafu 5.14 relaxace pomalejší než heuristiky. Z grafu 5.19 je vidět, že pro danou hustotu je z hlediska doby běhu Lagrangeovská relaxace citlivější na změny velikosti matice.

	Lag. relaxace		ContribAdd		Servít	
velikost matice	čas [s]	odchylka [%]	čas [s]	odchylka [%]	čas [s]	odchylka [%]
100x150	0.03	0.02	0.02	0.41	0.02	0.43
120x190	0.04	0.02	0.02	0.45	0.02	0.46
140x230	0.05	0.02	0.03	0.48	0.03	0.51
160x270	0.06	0.02	0.03	0.43	0.03	0.43
180x310	0.08	0.04	0.05	0.48	0.05	0.51
200x350	0.10	0.03	0.05	0.50	0.05	0.49
220x390	0.12	0.02	0.05	0.43	0.05	0.47

Tabulka 5.13: Porovnání heuristik s relaxací pro matice s hustotou 20%



Obrázek 5.18: Porovnání závislostí odchylek řešení heuristik a relaxace na velikosti matice



Obrázek 5.19: Porovnání závislostí doby běhu heuristik a relaxace na velikosti matice

5.5.3 Porovnání řešičů na reálných maticích

Tabulka 5.14 zobrazuje porovnání běhů řešičů, Lagrangeovské relaxace a heuristik na maticích, které byly vygenerovány z PLA tabulek benchmarkových obvodů, které byly dodány vedoucím práce a PLA tabulek vygenerovaných nástrojem pro tvorbu tabulek logických funkcí. Parametry matic a příslušných tabulek jsou uvedeny v tabulce 5.1 na začátku této kapitoly. Z naměřených hodnot vyplývá, že pro matice generované přímo nástrojem BOOM chování řešičů odpovídá chování pro uměle vytvořené matice, které byly použity pro měření v předchozích částech této kapitoly. Při bližším zkoumání je ovšem možné najít výjimky:

název	AURA II	B&B Lag	Relaxace		ContribAdd	
-	čas [s]	čas [s]	čas [s]	odchylka	čas [s]	odchylka
08x08x08	<0.05	<0.05	<0.05	0.00	<0.05	0.32
08x12x12	<0.05	<0.05	<0.05	0.00	<0.05	0.59
10x01x100	0.48	<0.05	<0.05	0.00	0.05	0.69
10x01x200	140.4	0.83	<0.05	0.01	0.09	0.78
10x10x10	<0.05	<0.05	<0.05	0.03	<0.05	0.54
10x10x20	5.84	0.08	<0.05	0.01	<0.05	0.55
10x10x30	754.5	1.55	<0.05	0.02	0.09	0.74
12x12x12	2.48	0.06	<0.05	0.00	<0.05	0.77
20x05x20	1446.4	0.17	<0.05	0.00	<0.05	0.84
20x05x20_10	2.95	<0.05	<0.05	0.00	<0.05	0.66
20x05x20_20	0.25	0.06	<0.05	0.00	<0.05	0.64
50x50x05_10	<0.05	<0.05	<0.05	0.00	<0.05	0.47
15x15x50_30_80	<0.05	<0.05	<0.05	0.00	<0.05	0.38
15x15x50_50_60	<0.05	<0.05	<0.05	0.00	<0.05	0.26
50x10x50_10_80	7.42	<0.05	<0.05	0.00	<0.05	0.54
50x10x50_50_70	152.7	<0.05	<0.05	0.00	0.05	0.59
50x10x50_60_30	<0.05	<0.05	<0.05	0.00	<0.05	0.10
apex3	<0.05	0.19	0.30	0.00	0.47	0.4274
bw	0.48	<0.05	<0.05	0.00	<0.05	0.62
cps	140.4	0.06	0.09	0.00	0.14	0.29
ex1010	<0.05	0.06	0.11	0.00	0.11	0.04
mainpla	2.48	0.84	1.91	0.00	1.13	0.61
pdc	2.95	0.06	0.08	0.00	0.09	0.33
spla	0.25	0.08	0.09	0.00	0.14	0.18
table3	1346.4	0.06	0.08	0.00	0.08	0.16
xparc	<0.05	0.36	0.64	0.00	0.55	0.42

Tabulka 5.14: Porovnání řešičů na reálných datech

- Pro matice apex3, ex1010 a xparc trvá nalezení optimálního pokrytí řešiči B&B Lagrange déle než AURA II. Tyto 3 matice mají společné to, že jejich počet řádků je vyšší než počet sloupců a hustota je menší jak 1%. Lze usoudit, že nesrovnalost v časech je dána tím, že pro B&B Lagrange je do času započítáno tvoření přesné mapy pokrytí v matici (viz 4.2.3).
- Pro matice main a xparc je heuristika rychlejší než Lagrangeovská relaxace, což neodpovídá výsledkům měření v 5.5.2. Důvod je stejný jako v předchozím případě - do času je započítána i tvorba pomocných struktur, což se projeví v případě velikých matic jako tyto dvě.

Kapitola 6

Začlenění do programu BOOM

Algoritmus je implementován především ve dvou třídách:

- LagBaBCover - implementace algoritmu Branch and Bound Lagrange 3.3.2.1.
- LagrangeSolver - implementace algoritmů Lagrangeovských relaxací 4.1.0.1 a 4.2.1.1

Dále jsou využívány pomocné třídy pro třídění matic a pro reprezentaci matic.

Kód programu BOOM bylo potřeba poupravit tak, aby bylo možné parametry určit, jaký způsob řešení problému pokrytí se má použít. Seznam nových parametrů lze nalézt v příloze C.

Příklad použití v kódu:

```
CovMatrix *CM = new CovMatrix(rows,cols);
/*
* ...
* fill CM
*...
*/
LagBaBCover* lagCover = new LagBaBCover(CM);
list<unsigned int>  sol = lagCover->Run(options);
```

Kapitola 7

Závěr

V této práci byl popsán rekurzivní algoritmus pro řešení problému pokrytí. Tento algoritmus pracuje metodou větví a hranic a pro ořezávání větví využívá informace získané Lagrangeovskou heuristikou. Algoritmus byl implementován v jazyce C++ a začleněn do minimalizačního nástroje BOOM. Chování algoritmu bylo změřeno a v této práci popsáno. Také bylo provedeno měření pro nalezení optimálních parametrů nastavení Lagrangeovské relaxace. Dále byl výsledný řešič srovnán s řešičem založeným na algoritmu AURA II a Lagrangeovská relaxace byla porovnána s heuristikami ContribAdd a heuristikou dle Servíta, které jsou v této práci popsány.

Měřením bylo zjištěno několik faktů:

- Lagrangeovská relaxace nachází lepší hodnoty spodní meze než postupy používané v algoritmu AURA II, což vede k tomu, že výsledný řešič je rychlejší.
- Duální relaxace umožňuje najít lepší hodnotu horní meze, která se používá ve vztahu pro úpravu multiplikátorů v Lagrangeovské relaxi (3.4.3). Nicméně výpočet duálních multiplikátorů je časově náročný a z hlediska celkového času běhu řešiče se duální relaxace nevyplatí.
- Pro výběr sloupce, který se při volání rekurzivního algoritmu přidává do řešení, je oproti heuristice výhodnější využít informace dodané Lagrangeovskou relaxací.
- Měření prokázala, že se z časového hlediska nevyplatí odstraňovat dominance při každém volání rekurzivního algoritmu - to platí pro vytvořený řešič i pro AURA II.
- Bylo zjištěno, že Lagrangeovská heuristika oproti heuristikám nachází řešení s menší odchylkou v ceně, nicméně za cenu větší časové náročnosti pro určité matice.

Tuto práci by bylo možno rozšířit o negativní přístup, který by spočíval ve spouštění Lagrangeovské relaxace s parametry nastavenými tak, aby bylo nalezena větší hodnota spodní meze.

Literatura

[1] O.coudert. "Two-level logic minimization: an overview", *Integration*, 1994 17-2:97-140.

[2] Lukáš Krejčík. Moderní metody řešení problému pokrytí. 2008.

[3] Michael Kalouš. Moderní metody řešení problému pokrytí. 2009.

[4] CORDONE, Roberto et al. "An Efficient Heuristic Approach to Solve the Unate Covering Problem", *In Design, Automation, and Test in Europe (Paris France)*, 2000, ISBN 1-58113-244-1, s. 364 - 371.

[5] [CER95] CERIA, S.; NOBILI, P.; SASSANO, A. "A Lagrangian-based heuristic for large-scale set covering problems", *Mathematical Programming*, 1998, 81: 215-228.

[6] M.Servít. "A heuristic method for solving weighted set covering problems", *Digital Processes*, vol. X, 1975, pp. 177-182.

[7] Tomáš Měchura. Parametrizovaný generátor náhodných booleovských funkcí. 2006.

[8] Hlavička, J., Fišer, P. "BOOM, a Heuristic Boolean Minimizer", Proc. International Conference on Computer-Aided Design ICCAD 2001, San Jose, California (USA), 4.-8.11.2001, pp. 439-442.

[9] Robert Vanderbei Ph.D. - Linear Programming - přednáškové slajdy `http://www.princeton.edu/~rvdb/542/lectures/`.

[10] David C. Parkes. - Linear Programming - přednáškové slajdy `http://www.eecs.harvard.edu/~parkes/cs286r/spring04/sections/sec1.pdf`.

[11] Yang S. "Logic Synthesis and Optimization Benchmarks, Version 3.0", *Tech. Report, Microelectronics Center of North Carolina*, 1991.

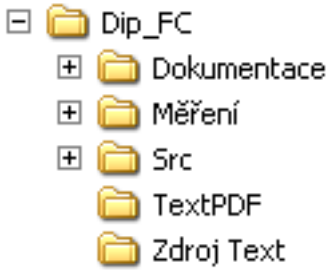
Příloha A

Seznam použitých zkratek

LB - Lower Bound (Dolní mez)
UB - Upper Bound (Horní mez)
CP - Covering Problem (Problém pokrytí)
P - Problem (Problém)
LP - Linear problem (Lineární problém)
LRP - Lagrangean relaxation problem (Problém Lagrangeovské relaxace)
DP - Dual problem (Duální problém)
LDP - Lagrangean dual problem (Lagrangeovský duální problém)

Příloha B

Obsah přiloženého CD



Složka `Měření` obsahuje výsledky měření, matice na kterých bylo prováěno měření a dávkové soubory spouštějící měření. Ve složce `Src` je umístěn kód programu a přeložený program spolu s připraveným příkladem ke spuštění. Dokumentace je uložena ve složce `Dokumentace` – generováno programem *Doxygen*. Zdrojový kód programu se nachází ve složce `src`. Text této práce v PDF je ve složce `TextPDF`. Zdrojový kód tohoto textu ve formátu TEXse nachází ve složce `Zdroj text`.

Příloha C

Ovládání programu BOOM

Program BOOM se ovládá parametry při spouštění v příkazové řádce. Stávající parametry jsou:
syntaxe:

```
BOOM [options] source [destination] [report]
```

-source - Vstupní PLA tabulka
-destination - Výstupní PLA tabulka
-report - Soubor, do kterého se vypisuje záznam průběhu běhu programu

Nově přidané parametry jsou:

-lagN n - Maximální počet iterací Lagrangevské relaxace
-lagNt n - Počet iterací Lagrangeovské relaxace před změnou koeficientu kroku
-lagNh n -Počet iterací Lagrangeovské relaxace mezi spouštěním Lagrangeovské heristiky
-lagTinit f - Počáteční koeficient kroku
-lagTmin f - Minimální koeficient kroku
-lagNtDual n - Počet iterací Lagrangeovské relaxace před změnou duálního koeficientu kroku
-lagTinitDual f - Počáteční duální koeficient kroku
-lagTminDual f - Minimální duální koeficient kroku

Použití kteréhokoliv z předchozích parametrů znamená, že se k řešení problému pokrytí bude používat B&B Lagrange.

-noDom - Řešič nebude odstraňovat dominance
-1Dom - Řešič bude odstraňovat dominance pouze při prvním volání rekurzivního algoritmu
-ContrAdd - Řešič bude k výběru sloupce využívat heuristiku ContribAdd (pokud se spouští algoritmus B&B Lagrange). Jinak tento parametr určuje, že se bude volat heuristika ContribAdd
-Servit - Bude se volat heuristika dle Servíta
-inputmatrix s - Vstupní matice problému

Příklad použití (k nalezení na přiloženém CD):

BOOM PLA\08x08x08.pla -i 10 vystup02.txt -lagNt 5 -lagNh 0 -lagTinit 2.0 -lagTmin 0.01 -noDom -inputMatrix matrixPLA\08x08x08.pla