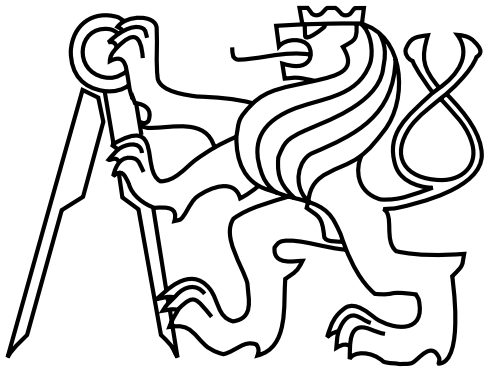
České vysoké učení technické v Praze
Fakulta elektrotechnická



Diplomová práce

Moderní metody řešení problému pokrytí

Michael Kalouš

Vedoucí práce: Ing. Petr Fišer

Studijní program: Elektrotechnika a informatika, dobíhající, magisterský
Obor: Výpočetní technika

leden 2009

Poděkování

Mé poděkování patří panu Ing. Petru Fišerovi za trpělivé vedení a veškerý čas, který mi věnoval při zpracování této diplomové práce.

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: Michael Kalouš

Declaration
I hereby declare that I have completed this master thesis independently and that I have listed all the literature and publications used. I have no objection to usage of this work in compliance with the act § 60 Zákona č.121/2000 Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

V Praze dne Michael Kalouš

Abstract

In this work is desribed and programmed one of exact algorithms for set covering problem, so called hybrid algorithm, that combines greedy algorithm, Lagrangian heuristic, non-binary genethic algorithm and L-class enumeration. Also alternative variant of the algorithm that replaces genethic algorithm by simulated annealing is programmed. Covering problem occurs in several fields of computer science, logic synthesis and reliability analysis. Solving the covering problem may be sometimes rather time-consuming. Performance of hybrid algorithm and its alternative variant is studied and compared to the AURA exact algorithm. Hybrid algorithm is incorporated to the program BOOM.

Anotace

V této diplomové práci je popsána a naprogramována jedna z exaktních metod pro řešení problému pokrytí - tzv. hybridní algoritmus, který kombinuje použití hladových algoritmů, Lagrangeovskou heuristiku, nebinární genetický algoritmus a výčet L-tříd. Naprogramována byla rovněž alternativní varianta hybridního algoritmu, kde je namísto nebinárního genetického algoritmu použito simulované ochlazování. Problém pokrytí se objevuje v množství oblastí počítačového výzkumu, syntéze logických obvodů a analýze spolehlivosti. Řešení problému pokrytí je výpočetně velmi náročné. Studována je výkonnost hybridního algoritmu a jeho alternativní varianty a výkonnost je srovnána s exaktním algoritmem AURA. Hybridní algoritmus je začleněn do programu BOOM.

Obsah

1 Úvod

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

Řešení problému pokrytí je obecně výpočetně velmi náročná záležitost. Pro řešení problému pokrytí byly navrženy algoritmy s různou složitostí a kvalitou řešení. Cílem algoritmů je nalézt co nejlepší řešení v přijatelném čase.

Řešení problému pokrytí může být dosaženo pomocí rekurzivního algoritmu, který zkouší přípustné kombinace prvků řešení, dokud není nalezeno minimální pokrytí. Řešení tímto způsobem je velmi časově náročné. Snížení výpočetního času může být dosaženo například ořezáváním stavového prostoru; neprohledávají se ty jeho části, o kterých je předem známo, že v nich optimální řešení není možné nalézt.

Při řešení rozsáhlých kombinatorických problémů a požadavku na optimalitu řešení se často používá technika kombinace aproximativních a exaktních algoritmů, kdy je nejprve aproximativním algoritmem nalezeno přípustné řešení, které slouží jako výchozí bod pro exaktní algoritmus.

Tato práce vychází z článku "A hybrid algorithm for set covering problem", jehož autory jsou od Anton V. Eremeev, Alexandr A. Kolokolov a Lidia A. Zaozerskaya, kde je popsán tzv. *hybridní algoritmus* pro řešení problému pokrytí, založený na technice zmíněné v předchozím odstavci. Hybridní algoritmus kombinuje tři aproximativní a jeden exaktní algoritmus. Výsledkem činnosti hybridního algoritmu je optimální řešení problému pokrytí.

Cílem této práce je popsat (s ohledem na možnost implementace) a implementovat hybridní algoritmus pro problém pokrytí, implementovat alternativní variantu hybridního algoritmu, prostudovat výkonnost algoritmu a porovnat ji s jinými algoritmy. Dalším cílem práce je začlenit algoritmus do programu BOOM, tak aby mohl být využit pro minimalizaci logických funkcí jako alternativa k dosud začleněným algoritmům.

Hybridní algoritmus pro problém pokrytí, včetně alternativní varianty byl naprogramován ve formě knihovny v ANSI dialektu jazyka C a začleněn do programu BOOM, prozkoumána byla jeho efektivita na různých problémech pokrytí a porovnána s algoritmem AURA.

2 Problém pokrytí

Problém pokrytí je kombinatorický optimalizační NP-těžký problém.

Neformálně lze problém pokrytí popsat takto: Na vstupu máme několik množin, které mají společné prvky. Naším úkolem je vybrat minimální sadu množin tak, aby tato sada obsahovala všechny prvky, které jsou obsaženy v kterékoliv množině ze vstupu.

Formálně lze popsat problém pokrytí několika způsoby, zde jsou uvedeny dva.

2.1 Problém pokrytí specifikovaný pojmy maticové algebry

2.1.1 Definice

Nechť $Y = \{y_1, y_2, ...y_m\}$ je množina řádků a $X = \{x_1, x_2, ...x_n\}$ je množina sloupců matice A o rozměrech $m \times n$ a CENA je funkce definovaná na množině X taková, že přiřazuje každému prvku $x_j \in X$ reálné číslo. Prvek $x_j \in X$ pokrývá prvek $y_i \in Y$, pokud $A[i, j] = 1$, jinak $A[i, j] = 0$. Problém pokrytí $< X, Y, CENA >$ spočívá v nalezení podmnožiny S množiny X, s nejmenší cenou, takové, že pro každý prvek y z množiny Y existuje prvek x z množiny X tak, že x pokrývá y.

2.1.2 Řešení

Řešením je nalezení minimální podmnožiny sloupců z matice A tak, že každý řádek matice A je pokrytý alespoň jedním sloupcem podmnožiny. Hledání řešení spočívá ve výběru sloupců do řešení, tak dlouho, dokud nejsou pokryty všechny řádky.

Při hledání řešení je možné pracovat pouze s *cyklickým jádrem* matice A, které se získá tzv. *odstraněním dominance;* tyto pojmy budou objasněny v příkladu.

2.1.3 Jednoduchý příklad

y_1	0	0	1	0
y_2	1	1	0	1
y_3	0	1	0	1
y_4	0	1	1	1
	x_1	x_2	x_3	x_4
c	2	1	3	5

Tabulka 1: Jednoduchý příklad

Hledaná podmnožina S je: $\{x_2, x_3\}$. Tato podmnožina sloupců pokrývá všechny řádky s cenou 4.

Dominance řádků

Říkáme, že řádek y' z množiny řádků Y dominuje řádku y právě tehdy, když všechny prvky množiny sloupců X, které pokrývají řádek y', pokrývají také řádek y.

V tabulce 1 snadno zjistíme, že řádek y_3 dominuje řádku y_4, protože řádek y_3 je pokryt sloupci x_2 a x_4 a oba tyto sloupce také pokrývají řádek y_4. Totéž platí i pro řádek y_2. To, že řádek y_3 dominuje řádkům y_2 a y_4, nám říká, že vybereme-li do řešení sloupec pokrývající řádek y_3, pak výběrem tohoto sloupce vždy pokryjeme i řádky y_2 a y_4. Proto můžeme řádky y_2 a y_4 vyloučit z matice. Snadno také nahlédneme, že řádek y_1 dominuje řádku y_4.

Dominance sloupců

Říkáme, že sloupec x' dominuje sloupci x právě tehdy, když všechny prvky množiny řádků Y pokryté x jsou také pokryté sloupcem x'.

V tabulce 1 vidíme, že sloupec x_4 dominuje sloupcům x_2 a x_1 a zároveň sloupec x_2 dominuje sloupcům x_4 a x_1. Nabízí se tedy možnost vyloučit z matice sloupce x_4 nebo x_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 je menší než cena sloupce, kterému sloupec dominuje. Nelze tedy vyloučit sloupec x_2, kterému dominuje sloupec x_4, protože jeho cena je menší než cena sloupce x_4.

Po odstranění dominancí sloupců mohou vzniknout nové dominance řádků a naopak, proto je potřeba postup odstraňování dominance opakovat. Případným opakovaným odstraňováním dominance získáme *cyklické jádro řešení.*

Nezbytné sloupce

Existuje-li řádek, který je pokryt jen jedním jediným sloupcem, je zřejmé, že tento sloupec bude vždy patřit do řešení, proto jej můžeme z matice ihned odstranit a přidat jej nakonec k řešení vzniklému postupem v odstavci 2.1.2. Sloupec s touto vlastností se označuje jako *nezbytný.* Příkladem takového sloupce je sloupec x_3, který jako jediný pokrývá řádek y_1.

2.2 Problém pokrytí specifikovaný pojmy teorie množin

2.2.1 Definice

Uvažujme množinu $M = \{1, ..., m\}$ a podmnožiny $M_j \subseteq M$, kde $J \in N = \{1, ..., n\}$. Podmnožina $J \in N$ je pokrytím množiny M když $\cup_{(j \in J)} M_j = M$. Pro každou podmnožinu M_j je stanovena kladná cena c_j. Problém pokrytí je pak nalezení pokrytí s minimální celkovou cenou.

2.2.2 Příklad

Uvažujme množinu $M = \{a, b, c, d\}$a podmnožiny z následující tabulky:

Podmnožina M_j	Cena c_j
$M_1 = \{b\}$	2
$M_2 = \{b, c, d\}$	1
$M_3 = \{a, d\}$	3
$M_4 = \{b, c, d\}$	5

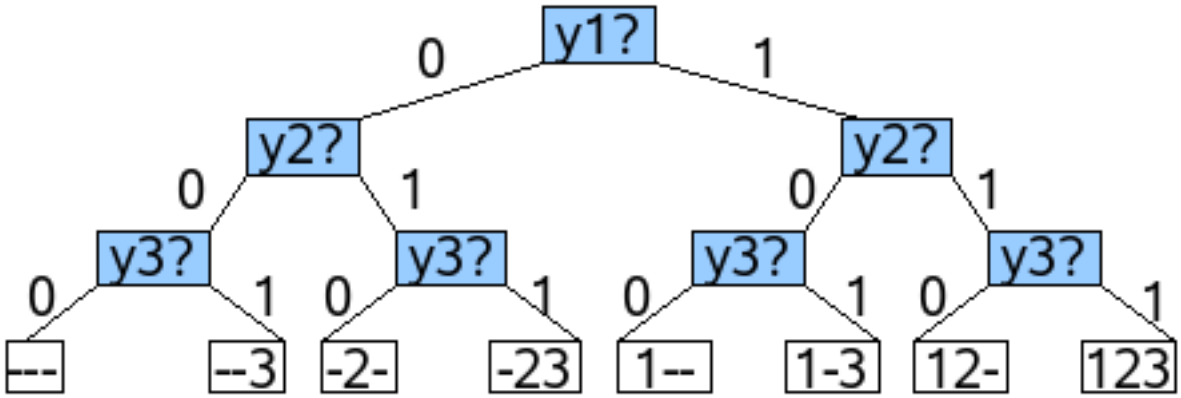
Řešením je pak podmnožina $J = \{2, 3\}$. Sjednocení podmnožin M_2 a M_3 je pokrytím s minimální celkovou cenou. Příklad je vlastně převedením výše uvedeného příkladu do pojmů teorie množin.

3 Exaktní algoritmy pro řešení problému pokrytí

3.1 Základní rekurzivní algoritmus

Rekurzivní algoritmus vybírá prvek x z množiny X a generuje dva podproblémy. Jeden podproblém spočívá v zařazení prvku x do minimálního řešení, druhý podproblém spočívá ve vyjmutí vyjmutí x z minimálního řešení. Algoritmus je pak aplikován na cyklická jádra těchto dvou podproblémů. Časová složitost algoritmu je exponenciální $O(2^n)$, kde n značí počet sloupců matice.

Na obrázku je znázorněn postup rekurzivních volání pro matici o velikosti 3 x 3 v nejhorším možném případě. Vyhledávací prostor je binární strom. Uzly představují vybraný sloupec x, ohodnocení hran pak označuje, zda byl sloupec x přidán do minimálního řešení (1), nebo odebrán z minimálního řešení (0). Listy pak představují pokrytí.



Obrázek 1: Rekurzivní volání pro matici 3 x 3

3.2 Metoda větví a hranic (Branch and Bound)

3.2.1 Základní varianta

Metoda větví a hranic představuje vylepšení základního rekurzivního algoritmu, vyhledávací prostor je ořezáván. Jsou ořezána místa, kde cena zatím nalezeného částečného pokrytí je větší než cena nejlepšího dosud nalezeného pokrytí. V těchto místech nemá smysl v rekurzi pokračovat, protože rekurze nemůže vést k lepšímu řešení.

Vyhledávací prostor je binárním stromem s kořenem, který představuje vstupní problém. Hrany reprezentují rekurzivní volání a jsou ohodnoceny 0 nebo 1 podle toho, zda je sloupec uvažovaný v rodičovském uzlu do řešení přidán nebo z něj odebrán. Vnitřní uzly stromu reprezentují částečné řešení. Listy stromu představují nalezená pokrytí.

Algoritmus se rekurzivně aplikuje na cyklická jádra podproblémů. Vždy se vybere sloupec x do řešení a vyškrtne se společně s řádky, které pokrývá. Odebráním sloupce mohou vzniknout dominantní řádky, proto se při větvení provede odstranění dominance. Je-li matice prázdná, pak bylo nalezeno řešení.

Algoritmus 1 Branch and Bound

```
Branch_and_bound(Matrix,Solution) {
  vyřeš_dominance(Matrix,Sol)
  if (matrix==∅) {
    if cena(Solution)<cena(best) return newBest
    else return ∅
  }
  else {
    vyber do řešení sloupec j
     if ( cena(cesta(j))+cena(j) < cena(best) )
       Branch_and_bound(Matrix,Sol1)
     vyškrtni sloupec j
     if ( cena(cesta(j)) < cena(best) )
       Branch_and_bound(Matrix,Sol2)
     return min (Sol1,Sol2)
  }
}
```

3.2.2 Vylepšená varianta

Metodu větví a hranic je možné ještě dále vylepšovat. Cílem vylepšení je stanovit takové podmínky pro ořezání, které vedou k zachování správnosti řešení a zároveň k ořezání více větví vyhledávacího prostoru.

Podmínka $cena(cesta(j)) + cena(j) < cena(best)$ slouží k ořezání těch větví stromu, kde cena řešení přesahuje cenu již nalezeného řešení. Hodnota na levé straně nerovnosti se označuje jako *spodní hranice řešení.* Je-li známa spodní hranice, může se při splnění podmínky rekurze ukončovat. Otázkou zůstává, zda není možné hodnotu na levé straně bezpečným způsobem zvýšit.

Předpokládejme, že řešení problému pokrytí není menší než nějaké číslo K. Pak cena částečného řešení, kterému odpovídá uzel u je rovna hodnotě $c = cesta(u) + K$. Známe-li cenu řešení *best*, pak lze rekurzi ukončit za podmínky, že $cena(cesta(u)) + K \geqq best$.

Existuje možnost, jak takové číslo K nalézt.

Nezávislá podmnožina řádků. Nechť $< X, Y, CENA >$ 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 poskytuje spodní hranici řešení.

Neformálně 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 2 je nezávislou podmnožinou řádků podmnožina $Y' = \{y_1, y_2\}$, neboť y_1 je pokryt právě sloupci x_1 a x_2 a y_2 je pokryt právě sloupci x_3a x_4. Z toho lze vyvodit, že pro pokrytí bude třeba vybrat jeden sloupec z podmnožiny $\{x_1, x_2\}$ a jeden sloupec z podmnožiny $\{x_3, x_4\}$. Vybereme-li "nejlevnější" sloupce, dostáváme minimální cenu pokrytí $2 + 2 = 4$.

y_1	1	1	0	0
y_2	0	0	1	1
y_3	0	0	1	1
y_4	0	0	1	1
	x_1	x_2	x_3	x_4
c	3	2	2	4

Tabulka 2: K nezávislé podmnožině řádků

Formálněji. Máme-li K nezávislých podmnožin řádků, pak je pro pokrytí třeba vybrat minimálně $|K|$ sloupců. Určíme-li maximální počet nezávislých podmnožin řádků (označme MSIR), dostaneme tak také mohutnost množiny Y'.

Chceme-li pokrýt řádek $y_1 \in Y'$, pak je nutné pro jeho pokrytí vybrat sloupec x s cenou $cena(x)$. Vybereme sloupec, který pokrývá řádek s nejnižší cenou a přičteme k MSIR. Takto upravená hodnota MSIR je spodní hranice řešení problému pokrytí.

Algoritmus 2 Metoda větví a hranic doplněná o MSIR

```
Branch_and_bound_MSIR(Matrix,Solution) {
  vyřeš_dominance(Matrix,Sol)
  if (matrix==∅) {
    if cena(Solution)<cena(best) return newBest
    else return ∅
  }
  else {
    vyber do řešení sloupec j
    najdi MSIR pro podproblém
     if ( cena(cesta(j)) + cena(j) + MSIR < cena(best) )
       Branch_and_bound(Matrix,Sol1)
     vyškrtni sloupec j
     najdi MSIR pro podproblém
     if ( cena(cesta(j)) + MSIR < cena(best) )
       Branch_and_bound(Matrix,Sol2)
     return min (Sol1,Sol2)
  }
}
```

Nalezení MSIR je však problém, který je stejně složitý jako problém pokrytí, proto se k nalezení MSIR používá heuristika. Její použití pouze sníží hodnotu MSIR a zvětší část stavového prostoru, který je třeba prohledat. Řešení zůstává řešením exaktním.

Algoritmus lze ještě dále zlepšit *zvýšením spodní hranice.*

Mějme Y' nezávislou podmnožinu Y. Nechť $lBound$ je spodní hranice nutná k pokrytí Y získaná z Y' a $uBound$ je horní hranice.
Označme $X' = \{x \in X | x \cap Y' = \varnothing, cena(cesta\, k\, uzlu) + lbound + cena(y) >= uBound\}$.
Pak problém může být zredukován na problém$< X - X', Y, CENA >$

Srozumitelněji řečeno, nalezneme-li při hledání řešení sloupce, které nepokrývají maximální množinu nezávislých řádků a zároveň splňují danou nerovnost, můžeme je z matice vyškrtnout a přepočítat MSIR (v důsledku vyškrtnutí sloupců se může změnit).

4 Základní aproximativní algoritmy

4.1 Přirozená heuristika

Princip přirozené heuristiky spočívá ve výběru prvku x, který pokrývá největší počet prvků Y. Pracuje se s funkcí NAT_HEURISTIC(*Matrix*,*Solution*), která z matice vybere sloupec pokrývající nejvíce řádků, společně s pokrytými řádky vybraný sloupec z matice vyjme a tak pokračuje dokud není matice prázdná. Současně se plní seznam sloupců *Solution* vybraných do řešení.

Algoritmus 3 Přirozená heuristika

```
1.Zjisti počty jedniček ve sloupcích
2.Vyber sloupec s největším počtem jedniček, vyjmi
sloupec a vyjmi řádky pokryté sloupcem
3.Není-li Matrix==∅,jdi na 1
```

Přirozená heuristika představuje velmi rychlý algoritmus, ale také algoritmus, který nedává příliš dobré výsledky; odchylky od exaktního řešení jsou příliš velké.

4.2 Contrib-Add heuristika

Tato heuristika je založena na výpočtu příspěvků jednotlivých sloupců k ceně. Heuristika pracuje se dvěma veličinami: Síla pokrytí řádku (CF) a sloupcový příspěvek (CC).

$$CF(y_i) = \frac{1}{\sum_{j=1}^{m} A[i,j]}$$

$$CC(x_i) = \sum_{i=1}^{n} A[i,j] \cdot CF(y_i)$$

Ze vzorců vyplývá, že síla pokrytí řádků se zmenšuje s počtem sloupců, které řádek pokrývají (je-li řádek pokryt mnoha sloupci, jeho síla pokrytí je malá). Sloupcový příspěvek se s počtem pokrytých řádek zvyšuje (pokrývá-li sloupec mnoho řádků, jeho sloupcový příspěvek je velký).

Heuristika upřednostňuje sloupce, které pokrývají mnoho řádků s velkou silou pokrytí.

Algoritmus 4 Contrib-Add heuristika

```
1.Když Matrix==∅, skonči
2.Spočítej pro všechna x_j hodnotu veličiny CC(x_j)
3.Vyber sloupec s největší hodnotou CC(x_j) a vyjmi ho spo-
lečně s řádky, které pokrývá
4.Jdi na 1
```

Contrib-Add heuristika dává výsledky, které se více blíží výsledkům exaktních algoritmů. Složitost algoritmu je $O(n^2)$, kde n je počet sloupců.

U obou heuristik může nastat situace, kdy na vyjmutí z matice budou existovat dva kandidáti. Při implementaci je třeba vždy vybrat pouze jednoho kandidáta (například náhodným výběrem).

4.3 Hladový algoritmus dle Chvátala

Algoritmus je popsán v článku [6].

1. Ze sloupců, které nejsou součástí pokrytí, vyber sloupec i takový, že poměr ceny sloupce c a počtu nově pokrytých řádků při zařazení sloupce c do pokrytí je minimální
2. Nejsou-li pokryty všechny řádky, pokračuj krokem 1

4.4 Hladový algoritmus dle Balase a Hoa

Algoritmus je popsán například v článku [9].

Algoritmus pracuje tak, že je náhodně vybrán řádek, který dosud není pokryt žádným ze sloupců. Pro každý sloupec, kterým je možné náhodně vybraný řádek pokrýt, je vypočten poměr počtu řádků, které by tímto sloupcem byly nově pokryty a ceny sloupce. Sloupec s maximálním poměrem se stane součástí řešení.

5 Hybridní algoritmus pro problém pokrytí

5.1 Úvod

Hybridní algoritmus pro problém pokrytí popsaný v článku [1] si klade za cíl nalézt optimální řešení problému pokrytí v přijatelném čase. Tohoto cíle se snaží dosáhnout vhodnou kombinací aproximativních algoritmů, které navazujíce jeden na druhý, postupně vylepšují přípustné řešení problému pokrytí.

Přípustné řešení nalezené aproximativními algoritmy je pak použito jako vstup pro finální exaktní algoritmus, kterým se získá řešení optimální. Exaktní algoritmus je urychlován vnitřními aproximativními kroky.

V hybridním algoritmu se používají následující techniky:

- Hladový algoritmus pro problém pokrytí dle Chvátala popsaný v sekci 4.3
- Lagrangeovská heuristika pro problém pokrytí, která je popsána v sekci 5.4.4
- Nebinární genetický algoritmus rozebraný v sekci 5.5
- Exaktní výčet L-tříd (LCE) urychlovaný použitím Lagrangeovské heuristiky

5.2 Specifikace problému pokrytí pro potřeby hybridního algoritmu

Pro potřeby hybridního algoritmu specifikujeme problém pokrytí jako problém *celočíselného lineárního programování (CLP problém):*

$$min\,\{c\mathbf{x} : A\mathbf{x} \geq \mathbf{e}, \mathbf{x} \in \{0,1\}^n\}$$

A je matice nul a jedniček o velikosti $m \times n$. Prvky matice nabývají hodnoty: $A_{ij} = 1$ právě tehdy když sloupec j pokrývá řádek i.

Dále $\mathbf{c}$ je vektor cen, který má n prvků a $\mathbf{e}$ je vektor jedniček o m prvcích. Předpokládáme také, že *sloupce matice A jsou seřazeny do posloupnosti s neklesající cenou.*

Řešení si můžeme představit jako bod v části n-rozměrného prostoru. Část prostoru je určena omezujícími podmínkami a má tvar polyhedronu, který vznikne jako průnik poloprostorů.

Pro ilustraci si převedeme zadání příkladu z odstavce 2.1.3.

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}, \mathbf{c} = (1,2,3,5), \mathbf{e} = (1,1,1,1)$$

Co vlastně zadání CLP problému říká ? Hledáme vektor $\mathbf{x} = (x_1, ..., x_n)$, $\mathbf{x} \in \{0,1\}^n$ takový, že jsou splněny následující podmínky:

Součet $x_1.c_1 + x_2.c_2 + x_3.c_3 + x_4.c_4$ je minimální

$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 \geq e_1$

$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 \geq e_2$

$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 \geq e_3$

$a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 \geq e_4$

5.3 Celkové schéma hybridního algoritmu

1. Algoritmus začne nalezením přibližného řešení problému pokrytí hladovým algoritmem. Cenu tohoto řešení označme jako $c(gd^0)$.

2. Dále je řešení problému nalezeno pomocí Lagrangeovské heuristiky. Tím se získá nejvyšší dolní mez ceny řešení LB a nějaké (ne nutně optimální, ale pokud možno kvalitní) řešení problému pokrytí, jehož cenu označíme$c(lg^0)$.

3. Hybridní algoritmus končí, jestliže $c(gd^0) = LB$, nebo $c(lg^0) = LB$. Při splnění jedné z těchto podmínek bylo nalezeno optimální řešení hladovým algoritmem nebo přímo Lagrangeovskou heuristikou.

4. Pro další kroky algoritmu máme k dispozici informaci, že cena řešení nepřekročí horní mez $UB = min\{c(lg^0), c(gd^0)\}$.

5. Horní mez se ještě může snížit nebinárním genetickým algoritmem GAH, jehož úkolem je simulovat evoluci přípustných řešení. Cenu řešení nalezeného genetickým algoritmem označme $c(ga^0)$.
   Hybridní algoritmus skončí, jestliže $c(ga^0) = LB$. Při splnění této podmínky bylo algoritmem GAH nalezeno optimální řešení.

6. Následuje algoritmus výčet L-tříd, který je jako jediný exaktní. Základem algoritmu LCE je hledání řešení *speciálně formulovaných LP problémů*, které jsou řešeny simplexovou metodou. Simplexová metoda je výpočetně velice náročná, proto se v algoritmu používají dva vnitřní aproximativní kroky, kterými jsou Lagrangeovská heuristika pro problém pokrytí a jednoduchá heuristika pro testování skupin L-tříd.

V dalším textu jsou jednotlivé části hybridního algoritmu podrobněji rozebrány a popsány.

5.4 Lagrangeovská heuristika pro problém pokrytí

5.4.1 Principy Lagrangeovské relaxace

Lagrangeovská relaxace optimalizačních problémů s omezujícími podmínkami je velmi často používanou technikou pro jejich řešení.

Lagrangeovská relaxace optimalizačních problémů je vytvořena:

- Odstraněním množiny omezujících podmínek

- Převedením těchto podmínek do optimalizační funkce ve formě tzv. trestných koeficientů, které se nazývají *Lagrangeovské multiplikátory.*

5.4.2 Relaxace problému pokrytí

Mějme problém pokrytí:

$min\,\{c\mathbf{x} : A\mathbf{x} \geq \mathbf{1}, \mathbf{x} \in \{0,1\}^n\}$

Omezující podmínku, kterou lze zapsat také jako: $\sum_{j=1}^{n} a_{ij}x_j \geq 1$ relaxujeme a definujeme Lagrangeovské multiplikátory $t_i \in (0, \infty)$ pro každý řádek matice A, která má m řádků a n sloupců. Po relaxaci dostaneme *problém lagrangeovské relaxace* (dále bude označován jako PLR) odpovídající následujícímu předpisu:

$$min \sum_{j=1}^{n}(c_j - \sum_{j=1}^{m}(t_i a_{ij})x_j + \sum_{j=1}^{m} t_i,\ x_j \in \{0,1\} \tag{1}$$

Dále zavedeme tzv. *redukovanou cenu sloupce*

$$\overline{c}_j = (c_j - \sum_{j=1}^{m} t_i a_{ij})$$

Lagrangeovské multiplikátory definované pro každý řádek matice A charakterizují *míru neplnění původních omezujících podmínek* aktuálním řešením 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.

Řešíme-li PLR, plnění předpisu (1) (minimalizace) můžeme dosáhnout pouze vhodnou volbou sloupců. Vybíráme do řešení sloupce, jejichž redukovaná cena je záporná - podle předchozího odstavce tak pokryjeme právě ty řádky, které mají velkou míru neplnění omezujících podmínek.

Řešení PLR nemusí být přípustným řešením problému pokrytí, avšak modifikace řešení PLR přidáváním nebo odebíráním sloupců může vést k dobrým řešením. Heuristický algoritmus, který používá řešení PLR a Lagrangeovské multiplikátory se nazývá *Lagrangeovskou heuristikou.*

Lagrangeovské heuristiky pro problém pokrytí pracují tak, že začnou s řešením PLR, přidají do řešení sloupce tak, aby upravené řešení bylo 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 [7].

5.4.3 Výstupy lagrangeovské heuristiky

Lagrangeovská heuristika poskytuje dva výstupy:

- Dolní mez řešení původního problému. Získaná dolní mez se používá v průběhu algoritmu LCE - viz sekce 5.6 k omezení počtu spuštění simplexové metody k řešení speciálních LP problémů - viz sekce 5.7.2.
- Řešení (ne nutně optimální) problému pokrytí. Cena tohoto řešení se rovněž využívá v algoritmu LCE, viz opět 5.7.2.

5.4.4 Postup při řešení problému pokrytí

Lagrangeovskou heuristiku budeme z úsporných důvodů označovat jako LH. LH zavádí proměnné, jejichž význam je uveden v tabulce 3.

Proměnná	Význam
ZLB	Optimální hodnota optimalizační funkce PLR
ZUB	Cena nejlepšího dosud nalezeného přípustného řešení problému pokrytí
LB	Nejvyšší dosud nalezená dolní mez
P_k	Dolní meze při zařazení sloupců k do řešení
t_i	Lagrangeovské multiplikátory

Tabulka 3: Proměnné LH

Krok 0. Předzpracování

1. Sloupce se seřadí podle vzrůstající ceny. Je-li cena sloupců stejná, pak preferován je ten sloupec, který pokryje více řádků.
2. Řádky se seřadí tak, aby první byly řádky pokryté nejméně sloupci

Krok 1 Inicializace

$LB = -\infty$, $ZUB = \infty$

$P_j = c_j \, (j = 1, ..., n)$, $t_i = min(c_j : a_{ij} = 1) \, (i = 1, ...m)$

Krok 2 Řešení PLR

Vyřeší se PLR se současnou množinou lagrangeovských multiplikátorů. Řešení označme $\mathbf{x}$ a cenu řešení ZLB.

$$ZLB = \sum_{j=1}^{m} \overline{C}_j x_j + \sum_{j=1}^{m} t_i$$

Aktualizuje se $LB = max\{LB, ZLB\}$. Sloupec je součástí řešení právě tehdy když jeho redukovaná cena je záporná. Řešení PLR nemusí být přípustným řešením problému pokrytí, avšak při konstrukci řešení problému pokrytí se z řešení PLR vychází.

Krok 3 Konstrukce řešení problému pokrytí

Varianta 1

1. Nechť $S = \{j | x_j = 1, j = 1, ..., n\}$ je řešení PLR. S je množina *indexů sloupců*.
2. Vybereme řádek, který není pokryt žádným sloupcem z množiny S, index tohoto řádku označme i.
3. Přidáme do množiny S sloupec s minimálním indexem, který pokrývá řádek s indexem i. Jdeme na 2.
4. Procházíme množinu S od nejvyššího indexu sloupce k nejnižšímu. Můžeme-li sloupec vyjmout z množiny při zachování pokrytí, vyjmeme jej z množiny S.

5. Aktualizujeme proměnnou ZUB. $ZUB = min(ZUB, \sum_{j \in S} c_j)$.

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

Varianta 2

1. Nechť $S = \{j | x_j = 1, j = 1, ..., n\}$ je řešení PLR. S je množina *indexů sloupců*.

2. Vybereme řádek, který není pokryt žádným sloupcem z množiny S, index tohoto řádku označme i.

3. Projdeme q nejlevnějších "kandidátských" sloupců, které pokrývají řádek s indexem i a vybereme sloupec, který má nejnižší *redukovanou* cenu. Konstanta q je předem dána. Menší hodnota konstanty q vede k úspoře výpočetního času za cenu horší kvality zkonstruovaného řešení.

4. Procházíme množinu S od nejvyššího indexu sloupce k nejnižšímu. Můžeme-li sloupec vyjmout z množiny při zachování pokrytí, vyjmeme jej z množiny S.

5. Aktualizujeme proměnnou ZUB. $ZUB = min(ZUB, \sum_{j \in S} c_j)$.

Varianta 3

1. Nechť $S = \{j | x_j = 1, j = 1, ..., n\}$ je řešení PLR. S je množina *indexů sloupců*.

2. Vybereme řádek, který není pokryt žádným sloupcem z množiny S, index tohoto řádku označme i.

3. Projdeme q nejlevnějších "kandidátských" sloupců, které pokrývají řádek s indexem i a vybereme sloupec, který má nejnižší *modifikovanou redukovanou* cenu. Konstanta q je předem dána. Menší hodnota konstanty q vede k ušetření výpočetního času za cenu horší kvality zkonstruovaného řešení.
   Při výpočtu modifikované redukované ceny se Lagrangeovské multiplikátory, příslušející již pokrytým řádkům, položí rovny nule.

4. Procházíme množinu S od nejvyššího indexu sloupce k nejnižšímu. Můžeme-li sloupec vyjmout z množiny při zachování pokrytí, vyjmeme jej z množiny S.

5. Aktualizujeme proměnnou ZUB. $ZUB = min(ZUB, \sum_{j \in S} c_j)$.

Varianta 4

1. Nechť $S = \{j | x_j = 1, j = 1, ..., n\}$ je řešení PLR. S je množina *indexů sloupců*.

2. Vybereme řádek, který není pokryt žádným sloupcem z množiny S, index tohoto řádku označme i.

3. Projdeme q nejlevnějších "kandidátských" sloupců, které pokrývají řádek s indexem i a vybereme sloupec, který má nejnižší *redukovanou* cenu. Konstanta q je předem dána. Menší hodnota konstanty q vede k úspoře výpočetního času za cenu horší kvality zkonstruovaného řešení

4. Seřadíme sloupce z množiny S podle redukované ceny

5. Procházíme množinu S od sloupce s nejvyšší redukovanou cenou ke sloupci s nejnižší redukovanou cenou. Můžeme-li sloupec vyjmout z množiny při zachování pokrytí, vyjmeme jej z množiny S.

6. Aktualizujeme proměnnou ZUB. $ZUB = min(ZUB, \sum_{j \in S} c_j)$.

Varianta 5

1. Nechť $S = \{j | x_j = 1, j = 1, ..., n\}$ je řešení PLR. S je množina *indexů sloupců*.

2. Vybereme řádek, který není pokryt žádným sloupcem z množiny S, index tohoto řádku označme i.

3. Projdeme q nejlevnějších "kandidátských" sloupců, které pokrývají řádek s indexem i a vybereme sloupec, který má nejnižší *modifikovanou redukovanou* cenu. Konstanta q je předem dána. Menší hodnota konstanty q vede k úspoře výpočetního času za cenu horší kvality zkonstruovaného řešení.
   Při výpočtu modifikované redukované ceny se Lagrangeovské multiplikátory, příslušející již pokrytým řádkům, položí rovny nule.

4. Seřadíme sloupce z množiny S podle modifikované redukované ceny.

5. Procházíme množinu S od sloupce s nejvyšší redukovanou cenou ke sloupci s nejnižší redukovanou cenou. Můžeme-li sloupec vyjmout z množiny při zachování pokrytí, vyjmeme jej z množiny S.

6. Aktualizujeme proměnnou ZUB. $ZUB = min(ZUB, \sum_{j \in S} c_j)$.

Krok 4 Možnost zastavení

Jestliže $\lceil LB \rceil = ZUB$, pak je LH ukončena. Operace nad proměnnou LB je nalezení nejbližší vyšší nebo rovné celočíselné hodnoty.

Krok 5 Aktualizace P_k, vyloučení sloupců

Uvážíme-li, že pokud k zadání problému pokrytí přidáme omezující podmínku spočívající ve vnucení určitého sloupce do řešení, dostaneme nový problém pokrytí, pro jehož dolní mez NLB platí:
$NLB = ZLB + \overline{c}_k$, pokud sloupec není v řešení PLR.
$NLB = ZLB$, pokud sloupec je v řešení PLR.

Aktualizace P_k

Jestliže $x_k = 0$, pak $P_k = max(P_k, ZLB + \overline{c_k})$

Jestliže $x_k = 1$, pak $P_k = max(P_k, ZLB)$

Z dalších úvah můžeme vyloučit sloupce, pro které platí, že $P_k \geq ZUB$ tak, že položíme $c_k = \infty$. Je-li dolní mez řešení při zařazení daného sloupce větší než nejlepší dosud nalezená dolní mez, nemůže zařazení sloupce vést ke zlepšení řešení.

Vzhledem k tomu, že práce s hodnotou nekonečno může při použití výpočetní techniky činit potíže, při implementaci algoritmu je vhodnější použít masku sloupců, nebo jejich spojový seznam.

Krok 6 Výpočet subgradientu

$$G_i = 1 - \sum_{j=1}^{n} a_{ij} x_j$$

Podobně jako gradient udává směr nejvyššího spádu funkce, subgradient je vektor, který udává směr dalšího postupu při řešení. Jak vyplývá ze vzorce, složky subgradientu pro dosud nepokryté řádky jsou kladné, pro ostatní jsou nulové nebo záporné. Podle vypočteného subgradientu se modifikují Lagrangeovské multiplikátory tak, aby indukovaly nižší redukované ceny sloupců pokrývajících řádky, jež dosud nebyly pokryty. Složky subgradientu náležející k těmto řádkům jsou rovny 1.

Podle [7] je užitečné subgradient upravit, tak že $G_i = 0$, jestliže $t_i = 0$ a zároveň $G_i < 0$.

Jestliže $\sum_{j=1}^{m} (G_i)^2 = 0$, nelze určit směr dalšího postupu a Lagrangeovská heuristika končí.

Krok 7 Velikost kroku

Modifikace lagrangeovských multiplikátorů probíhá v krocích. Velikost kroku je definována následujícím vzorcem.

$$T = \frac{f(1.05\, ZUB - ZLB)}{\sum_{j=1}^{m} (G_i)^2}$$

Velikost kroku se odvíjí od rozdílu mezi cenou nejlepšího nalezeného řešení a naposledy nalezenou dolní mezí. Je-li rozdíl malý, postupuje při modifikaci lagrangeovských multiplikátorů v jemných krocích. Je-li rozdíl velký, může se postupovat po větších krocích, ovšem pouze pokud je kvadratická norma subgradientu malá, tj. omezující podmínky jsou v plněny v dostatečné míře. Takto škálovatelná velikost kroku je motivována snahou o konvergenci proměnných ZUB a ZLB.

Počáteční hodnota proměnné f je 2. Jestliže se proměnná LB nezvýšila za posledních 30 iterací, hodnota proměnné f je vydělena dvěma a postupuje se v jemnějších krocích - viz [7].

Krok 8 Pokračování algoritmu, aktualizace lagrangeovských multiplikátorů

Jestliže bylo provedeno více než dané maximální množství iterací, Lagrangeovská heuristika končí. Jinou podmínkou pro ukončení může být pokles hodnoty proměnné f pod 0,005, nebo překročení nějaké předem dané horní meze optimalizační funkce.

Lagrangeovské multiplikátory jsou aktualizovány podle vzorce
$t_i = max(0, t_i + TG_i), i = 1, ..., m$. Subgradient tedy Lagrangeovské multiplikátory směruje tak, aby vedly k volbě sloupců pokrývajících řádky neplnící omezující podmínky.

Pokračuje se krokem 2.

5.5 Genetický algoritmus

Genetický algoritmus představuje jednu z variant evolučních algoritmů.

5.5.1 Obecně k evolučním algoritmům

Evoluční algoritmy jsou založeny na simulaci evoluce v přírodě a řídí se třemi základními principy, kterými se evoluce v přírodě vyznačuje.

Reprezentace řešení problému. Jednotlivá řešení problému (přípustná, ale ne nutně optimální) jsou reprezentována jako jedinci.

Princip přirozeného výběru. Přirozený výběr je proces, který vede k tomu, že někteří jedinci jsou schopni vyprodukovat více potomků než jiní jedinci. Pravděpodobnost předání genetického materiálu potomkovi je označována jako *zdatnost*. Zatímco v přírodě je obtížné určit, který jedinec je zdatnější, evoluční algoritmy často využívají deterministickou funkci zdatnosti, která přiřadí jedinci reálné číslo. Při určování zdatnosti nerozhodují jen vlastnosti jedince, ale i vlastnosti prostředí. Jedinec zdatný v jednom prostředí nemusí být zdatný v jiném prostředí.

Princip variace a iterativního vyhodnocování. V jedné iteraci (generaci jedinců) je na populaci aplikováno množství operací. Cílem těchto operací je zajistit drobné změny v populaci. Dvě nejpoužívanější operace jsou *křížení* a *mutace*.

5.5.2 Principy genetických algoritmů

Genetické algoritmy jsou nejpopulárnější variantou evolučních algoritmů. Genetické algoritmy pracují s obvykle ustálenou *populací*. Populace se skládá z *jedinců* představujících řešení optimalizačního problému. Každý jedinec je reprezentován dvojicí *genotypu g* (což je soubor genů daného jedince) a *fenotypu f(g)*. Fenotyp představuje kvalitativní vlastnosti jedince a funkce $x(g)$ mapuje genotyp na fenotyp.

Evoluční jevy (zdroje změn) genetické algoritmy reprezentují pomocí operátorů:

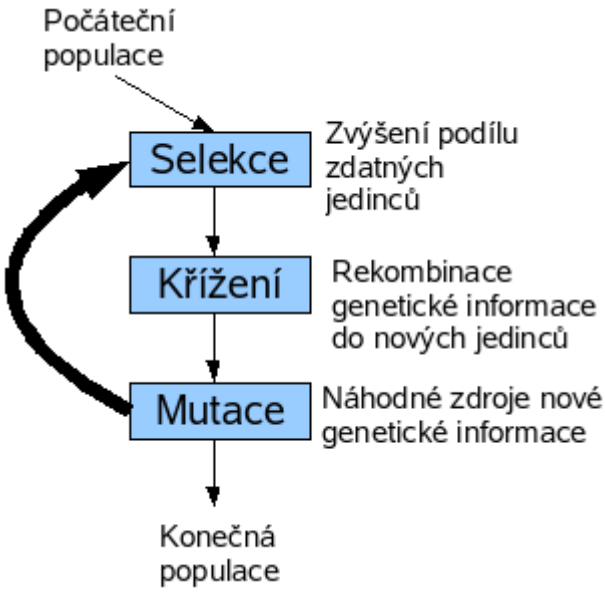
SELEKCE je výběr jedinců, kteří se stanou rodiči nového jedince.

KŘÍŽENÍ je proces, kdy část genů jednoho rodiče je nahrazena částí genů druhého rodiče

MUTACE reprezentuje náhodné jevy (např. působení radiace, změny podnebí trvalejšího charakteru aj.)

Selekce je řízena vyšetřením funkce zdatnosti $\phi(g)$ u každého genotypu. Funkce zdatnosti udává, nakolik je řešení reprezentované daným genotypem "dobré" (analogie "životaschopnosti" ve skutečné přírodě). V přírodě mívají zdatnější jedinci více potomků, toto se v genetických algoritmech zajistí tak, že zdatnější jedinci budou vybíráni jako rodiče potomků častěji.

Genetický algoritmus probíhá v mnoha iteracích, součásti každé iterace jsou znázorněny na obrázku 2.



Obrázek 2: Obecné schéma genetických algoritmů

5.5.3 Genetický algoritmus, který je součástí hybridního algoritmu (GAH)

GAH začíná vygenerováním *počáteční populace,* podle předem daného rozložení pravděpodobnosti.

Počet jedinců se v průběhu algoritmu nemění, GAH je tedy genetický algoritmus s *ustálenou populací.* V průběhu každé iterace je vygenerován jeden nový jedinec, který nahradí nejméně zdatného jedince.

Při generování nového jedince se nejprve vyberou dva jedinci, kteří jsou kříženi. Nový jedinec je pak podroben mutaci. V GAH je pravděpodobnost mutace konstantní p_m.

5.5.4 Reprezentace jedince

Nejprve připomeňme, jak vypadá matice A.

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

Označme množinu sloupců, která pokrývá daný řádek i jako $N_i = \{j \in N : a_{ij} = 1\}$. Genotyp se skládá z m genů $g^{(1)}, ..., g^{(m)}$ takových, že $g(i) \in N_i$. Každý gen tedy obsahuje index sloupce, který je přiřazen k pokrytí řádku.

Z matice nahlédneme, že:
$N_1 = \{3\}$
$N_2 = \{1, 2, 4\}$
$N_3 = \{1, 4\}$
$N_4 = \{1, 3, 4\}$

Například genotyp $< 3, 1, 1, 1 >$ znamená, že jedinec představuje řešení, které se snaží pokrýt první řádek třetím sloupcem a zbývající řádky prvním sloupcem. Dále se bude tato skutečnost popisovat tak, že daný jedinec používá k pokrytí určité sloupce. Při této reprezentaci $x(g)$ mapuje genotyp g na fenotyp $x(g) \in \{0, 1\}^n$, kde jedničky odpovídají přítomnosti sloupce v genech genotypu g. Například $x(< 3, 1, 1, 1 >) = (1, 0, 1, 0)$.

Protože geny mohou nabývat i jiných hodnot než jen 0 a 1 je tento genetický algoritmus *nebinárním genetickým algoritmem.*

Kvůli efektivní implementaci algoritmu je potřeba rozšířit množinu operátorů ještě o procedury, které eliminují redundantní sloupce. Podrobnosti jsou uvedeny níže. Musíme totiž vzít v úvahu, že různé genotypy může mapovat funkce $x(g)$ na stejné fenotypy. Například $x(< 1, 1, 1, 3 >) = x(< 1, 3, 3, 3 >) = (1, 0, 1, 0)$. V populaci však chceme mít pouze různé fenotypy.

5.5.5 Selekce a funkce zdatnosti

Funkce zdatnosti je pro jedince s genotypem g definována takto: $\phi_t(g) = c[x(g_{l(t)})] - c[x(g)] + c_n$, kde $l(t)$ je index jedince s nejvyšší cenou pokrytí v dané iteraci. Od ceny jedince s nejvyšší cenou pokrytí odečteme cenu pokrytí představovaného jedincem jehož zdatnost vyšetřujeme a přičteme cenu posledního (nejlevnějšího) sloupce.

5.5.6 Schéma GAH

Krok 1 Dokud není počáteční populace hotova:

- Krok 1.1 Generuj náhodný genotyp g
- Krok 1.2 Proveď proceduru Prime(g) - viz 5.5.8 a přidej genotyp do populace

Krok 2 V každé iteraci proveď následující:

- Krok 2.1 Operátorem SELEKCE vyber dva genotypy g_u a g_v
- Krok 2.2 Vytvoř nového jedince operátorem KŘÍŽENÍ
- Krok 2.3 Proveď MUTACI každého genu s pravděpodobností p_m
- Krok 2.4 Nechť g' je nejlepším výsledkem procedur Greedy(g) a DualGreedy(g) - viz 5.5.8
- Krok 2.5 Nejsou-li v populaci jedinci s fenotypem $x(g')$, pak g' nahradí nejméně zdatného jedince, jinak nejméně zdatného jedince nahradí g

5.5.7 Implementace operátorů

SELEKCE Operátor implementuje proporcionální schéma selekce, pravděpodobnost výběru k-tého jedince je dána následujícím vztahem:

$$P(k) = \frac{\phi_t(g_k)}{\sum_{l=1}^{s} \phi_t(g_{l(t)})}$$

Pravděpodobnost výběru zdatnějšího jedince je větší než pravděpodobnost výběru jedince méně zdatného. Tím dosáhneme toho, že zdatnější jedinci budou mít více potomků.

KŘÍŽENÍ V tomto případě se jedná o tzv. LP-křížení, tedy křížení při kterém je řešen LP problém. Operátor je implementován tak, aby našel nejlepší možnou kombinaci rodičovských genotypů g_u a g_v, pokud je to možné bez příliš náročných výpočtů.

Uvažujme problém optimálního křížení P_{oc}, což je vlastně redukovaná verze problému počátečního pokrytí množiny, kde pokrývající podmnožiny jsou omezeny množinou indexů $N' = \{j : x(g_u)_j = 1\} \cup \{j : x(g_v)_j = 1\}$.

Srozumitelněji řečeno, pro pokrytí množiny řádků máme k dispozici sloupce, které používají k pokrytí křížení jedinci.

Nejprve je na P_{oc} aplikována jednoduchá redukce. Označme $S = \{i \in M : |N_i \cap N'| = 1\}$. Každý řádek $i \in S$ může být pokryt právě jedním sloupcem $j(i)$. Pojmenujme $Q = \{j : j(i), i \in S\}$ množinu "upevněných" sloupců. Pak pro každý řádek $i \in \cup_{j \in Q} M_j$ je přiřazen jeden ze sloupců, který pokrývá gen $g^{(i)}$. Tyto geny považujme také za "upevněné".

Srozumitelněji řečeno, při dalším výpočtu nebudeme uvažovat řádky, které je možné pokrýt jen jedním sloupcem, ani sloupce, které tyto řádky pokrývají.

Jako výsledek této redukce dostaneme podproblém P_r, který se skládá ze sloupců a řádků, které nebyly "upevněny". K vyřešení lineární relaxace podproblému P_r se použije simplexová metoda - viz např. [10, 8]. Jestliže řešení $\mathbf{x}'$, které poskytne simplexová metoda je celočíselné, pak je toto řešení použito k doplnění genotypu g.

Aby se eliminovaly zdlouhavé výpočty, simplexovou metodu nespouštíme v případě, že počet řádků redukovaného problému přesahuje určitou předem danou hranici, nebo je překročen určitý počet iterací simplexové metody (implementace simplexové metody tedy musí podporovat možnost zastavení po určitém počtu iterací). V případě neceločíselného řešení $\mathbf{x}'$ operátor KŘÍŽENÍ vrátí nezměněný genotyp g_u.

Příklad křížení:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

Mějme genotypy $< 3, 1, 1, 1 >$ a $< 3, 4, 4, 4 >$. Pak $N' = \{3, 1\} \cup \{3, 4\} = \{1, 3, 4\}$.

Z matice nahlédneme, že:
$N_1 = \{3\}$
$N_2 = \{1, 2, 4\}$
$N_3 = \{1, 4\}$
$N_4 = \{1, 3, 4\}$

Určíme, že:
$|N_1 \cap N'| = 1$
$|N_{2,3} \cap N'| = 2$
$|N_4 \cap N'| = 3$

$S = \{1\}$

Řádek 1 může být pokryt jedině sloupcem 3. Dostaneme tedy podproblém s maticí A^1:

$$A^1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

Simplexová metoda pokryje zbývající řádky třetím sloupcem matice A^1 (připomeňme, že sloupce jsou seřazeny do posloupnosti s nerostoucí cenou). V původní matici A to byl sloupec 4. Výsledný fenotyp bude $x(g) = (0, 0, 1, 1)$. Fenotyp je nutné převést zpět na genotyp. Řádky samozřejmě pokrýváme pouze sloupci, které jsou přítomny ve výsledném fenotypu. Nový jedinec bude mít genotyp $< 3, 4, 4, 3 >$nebo $< 3, 4, 4, 4 >$.

MUTACE Pokud má být gen $g^{(i)}$ daného jedince mutován, pak pravděpodobnost, že mu bude přiřazen sloupec $j \in N_i$ je dána vztahem: $p_i(j) = \frac{\frac{1}{c_j}}{\sum_{k \in N_i} \frac{1}{c_k}}$

5.5.8 Implementace procedur pro odstranění redundantních sloupců

Greedy(Genotyp g). Vyřeší problém pokrytí hladovým algoritmem popsaným v sekci 4.3.

DualGreedy(Genotyp g). Kombinace postupného odstraňování sloupců a jejich adaptivního oceňování.

Označme množinu sloupců, která pokrývá daný řádek i jako $N_i = \{j \in N : a_{ij} = 1\}$a dále $N' = \{j \in N : x(g)_j = 1\}$, množinu indexů sloupců, které jedinec používá k pokrytí.

Krok 1 Nastav $N'_i = N_i \cap N', i = 1, ..., m; M' = M, J' = N', J = \emptyset$

Krok 2 Dokud $J' \neq \emptyset$:

- Krok 2.1 Existuje-li $i \in M'$ takové, že $|N'_i| = 1$, pak $J = J \cup \{j\}, M = M' \setminus M_j, j \in N'_i$, jinak vyber $j \in J'$ takové, že $j = argmax_{k \in J'} \frac{c_k}{|M_k \cap M'|}$
  Existuje-li dosud nepokrytý řádek, který je pokryt jen jedním sloupcem, pak tento sloupec použij k jeho pokrytí. Jinak vyber z dostupných sloupců takový, že poměr uvedený je maximální.
- Krok 2.2 Nastav $J' = J' \setminus \{j\}, N'_i = N'_i \setminus \{j\}$pro všechna $i \in M_j$
  Použitý sloupec odeber z množiny dostupných sloupců.

Prime(Genotyp g). Jednoduché postupné odstranění sloupců se vzrůstajícími indexy (připomeňme, že sloupce jsou seřazeny do posloupnosti s nerostoucí cenou). Sloupec je odstraněn, pokud zbývající řešení zůstává pokrytím. Výstupem je pokrytí bez redundantních sloupců.

5.6 Výčet L-tříd

Pro výčet L-tříd se bude používat zkratka LCE.

Principy

Jedním z možných přístupů k řešení LP problémů je použití zvláštních *dělení* prostoru R^n. Jedním z takových dělení je L-dělení, které rozdělí body v prostoru do L-tříd [1].

Uvažujme, že $\Omega = \{\mathbf{x} \in R^n : A\mathbf{x} \geq \mathbf{e}, 0 \leq \mathbf{x} \leq \mathbf{e}\}$ je relaxační množina problému pokrytí. Je to vlastně mnohostěn v prostoru R^n. Výčet L-tříd pracuje následujícím způsobem: Mnohostěn Ω může být rozdělen na *lexikograficky uspořádané* podmnožiny a optimální řešení může být nalezeno ověřením některých z těchto podmnožin.

Pro vyjádření lexikografického uspořádání budou v následujícím textu použity symboly $\prec \preceq \succeq \succ$. Například platí, že $(0,0,0) \prec (0,0,1) \prec (0,1,0) \prec (0,1,1)$.

Jak L-dělení rozděluje body do L-tříd? Dva body $\mathbf{x}, \mathbf{y} \in R^n$ náleží do stejné třídy L-dělení právě tehdy, když neexistuje žádný celočíselný vektor $\mathbf{z}$ takový, že $\mathbf{x} \succeq \mathbf{z} \succeq \mathbf{y}$.

Každý bod $\mathbf{z} \in Z^n$, tedy každý bod s celočíselnými souřadnicemi formuje oddělenou třídu L-dělení. Ostatní třídy, které obsahují pouze neceločíselné body jsou třídy *zlomkové*.

L-dělení množiny $X \in R^n$ indukuje množinu L-tříd, která bude dále označována jako X/L. Tato množina má zajímavé vlastnosti.

1. Je-li množina X ohraničená, pak je množina X/L konečná.
2. Pro X/L může být definováno lineární uspořádání: pro neprázdné $V^{'}, V^{''} \in X/L$ můžeme psát, že $V^{'} \succ V^{''}$, právě tehdy když pro libovolná $\mathbf{x}' \in V'$ a $\mathbf{x}'' \in V''$ platí, že $\mathbf{x}' \succ \mathbf{x}''$.

Necht' Ω_0 je množina optimálních řešení pro problém pokrytí. Algoritmus LCE generuje sekvenci aktuálních bodů $\sigma = \{\mathbf{x}(t)\}$ z množiny Ω a $t = 1, ..., \theta_L$. Sekvence bodů má následující vlastnosti:

1. $\mathbf{x}(t) \prec \mathbf{x}(t+1)$pro $t = 1, ..., \theta_L - 1$, tedy každý další bod v sekvenci je lexikograficky větší než bod předchozí.
2. Body v sekvenci patří do rozdílných L-tříd.
3. σ obsahuje celočíselnou pod-sekvenci $\sigma' = \{\mathbf{x}(t_k)\}, k = 1, ..., q$ takovou, že $z* \in \sigma'$ a když $t > t_k$, tak $(c, \mathbf{x}(t)) < (c, \mathbf{x}(t_k))$.

Z bodu jedna lze odvodit, že mohutnost sekvence σ je menší nebo stejná jako mohutnost X/L. Jednotlivé body sekvence $\mathbf{x}(t)$ se získají řešením *speciálně formulovaných LP problémů*. Tak je řešení CLP problému, který patří do skupiny NP těžkých problémů převedeno na řešení sekvence polynomiálních problémů.

Cílem algoritmu LCE je najít celočíselné optimum v množině $\Omega/L = \{V_1, ..., V_p\}$. Není nutné navštívit všechny L-třídy, dodatečná podmínka, jako podmínka
$(c, \mathbf{x}) \leq (c, \xi) - 1$,kde $\xi = \mathbf{x}(t_k)$ je dosud nejlepší nalezené *celočíselné* řešení, může vyloučit některé nepotřebné L-třídy. Hodnota optimalizační funkce $r = (c, \xi)$ se označuje jako *záznam* aktuálního kroku. Tato hodnota může být při inicializaci algoritmu nastavena na

∞, případně může být její počáteční hodnota určena nějakým aproximativním algoritmem (nebo algoritmy). V průběhu algoritmu je pak postupně snižována.

Aproximativní algoritmy spuštěné před algoritmem LCE mají za cíl snížit počáteční hodnotu záznamu co nejvíce je to možné, aby při výčtu L-tříd bylo co nejvíce těchto L-tříd vyloučeno podmínkou $(c, \mathbf{x}) \leq (c, \xi) - 1$.

Obecné schéma algoritmu LCE

Krok 0 Najdi nějaké přibližné řešení $\xi^0 \in Z^n$

Krok 0.1 Najdi $\mathbf{x}'$ jako $lexmin(\Omega)$

Krok 0.2 Nastav $r = min\{(c, \mathbf{x}'), (c, \xi^0)\}$
Záznam r nastav na na menší hodnotu z ceny přibližného řešení a ceny řešení představovaného lexikografickým minimem množiny Ω.

Krok 0.3 Nastav $p = max\left\{j : x'_j = 1, j = 1, ..., n-1\right\}$

Krok 1 Najdi $\varphi = max\left\{j : x'_j = 0, j = 1, ..., p-1\right\}$

Krok 1.1 Jestliže takové φ neexistuje, výčet L-tříd je ukončen.

Krok 2 Nastav $\mathbf{x}'' = \mathbf{x}'$. Najdi další L-třídu řešením následujícího problému:
$\mathbf{x}' = lexmin\left\{\mathbf{x} \in \Omega : (c, \mathbf{x}) \leq r - 1, x_1 = x''_1, ..., x_{\varphi-1} = x''_{\varphi-1}, x_\varphi = 1\right\}$
Podrobnosti k tomuto kroku jsou podrobně rozpracovány v sekci 5.7.

Krok 3 Krok 2 může poskytnout následující výsledky:
1. Nalezena celočíselná L-třída:
Nastav $p = max\{j : x'_j = 1, j \leq n-1\}$. Pokračuj krokem 1.
2. Nalezena zlomková L-třída.
Nastav $\varphi = min\{j : x'_j \neq \lfloor x'_j \rfloor, j = 1, ..., n\}$. Pokračuj krokem 2.
3. Nenalezena žádná L-třída:
Nastav $p = \varphi$. Pokračuj krokem 2.
4. Ukonči algoritmus.
Při tomto výsledku je algoritmus ukončen. Řešením problému pokrytí je naposledy nalezená celočíselná L-třída.

5.7 Ke krokům algoritmu LCE

Na úvod je třeba připomenout, že algoritmus LCE pracuje s *relaxací* původního CLP problému. To znamená, že původní podmínka pro vektor $\mathbf{x}$ je *oslabena* podle následujícího vzorce:

$$min\{\mathbf{cx} : A\mathbf{x} \geq \mathbf{e},\ x_{1...n} \in < 0, 1 >\} \tag{2}$$

Oslabení původní podmínky vede k přechodu od CLP problému k LP problému.

5.7.1 Lexikografické minimum relaxační množiny

$\mathbf{x} = lexmin(\Omega) :\ A\mathbf{x} \geq 1,\ x_n \in < 0, 1 >$

Procházejí se jednotlivé sloupce od prvního k poslednímu (připomeňme, že jsou seřazeny do posloupnosti s neklesající cenou). Sloupec je z pokrytí vyjmut, jestliže zbývající sloupce stále tvoří pokrytí. Lexikografické minimum je vždy celočíselné [1].

5.7.2 Krok 2 algoritmu LCE

V kroku 2 se řeší následující problém:

$$\mathbf{x}' = lexmin\left\{\mathbf{x} \in \Omega : (c, \mathbf{x}) \leq r-1, x_1 = x_1'', ..., x_{\varphi-1} = x_{\varphi-1}'', x_\varphi = 1\right\} \tag{3}$$

Hledání lexikografického minima představuje sekvenci řešení LP problémů s postupným zmrazováním jednotlivých proměnných. Zde jsou symbolicky naznačeny první tři LP problémy ze sekvence:

$k_1 = min\, x_1$	$k_2 = min\, x_2$	$k_3 = min\, x_3$
$A\mathbf{x} \geq 1$	$A\mathbf{x} \geq 1$	$A\mathbf{x} \geq 1$
$c_1x_1+c_2x_2+...+c_nx_n \leq r-1$	$c_1x_1+c_2x_2+...+c_nx_n \leq r-1$	$c_1x_1+c_2x_2+...+c_nx_n \leq r-1$
	$x_1 = k_1$	$x_1 = k_1$
		$x_2 = k_2$

Na vstupu máme vektor $\mathbf{x}$ a proměnné φ a p a záznam r. Vektor $\mathbf{x}$ můžeme rozdělit na upevněnou část a volnou část, tak jako je to naznačeno v následujícím předpisu. Volná část vektoru $\mathbf{x}$ je naznačena otazníky. V upevněné části se mohou vyskytovat pouze celá čísla, což lze snadno odvodit z kroku 3 algoritmu LCE.

$$\mathbf{x} = (x_1, x_2, x_3, ..., x_{\varphi-1}, \mathbf{1}, ?, ?, ?, .., ?)$$

Kontrola upevněné části. Nejprve zkontrolujeme upevněnou část - představuje sloupce, o kterých jistě víme, že budou či nebudou součástí pokrytí. Jestliže pro cenu $c(f) = (c_1x_1 + c_2x_2 + ... + c_\varphi x_\varphi)$ platí $c(f) > r - 1$, pak LP problém jistě nemá řešení a pokračuje se heuristikou pro testování skupin L-tříd popsanou v sekci 5.7.3.

Jestliže upevněné sloupce představují pokrytí (nerovnosti jsou již splněny), pak řešením problému je vektor
$\mathbf{x} = (x_1, x_2, x_3, ..., x_{\varphi-1}, \mathbf{1}, 0, 0, 0, .., 0)$, který je zřejmě celočíselný a lexikograficky minimální. Byla tedy nalezena celočíselná L-třída, která je zapamatována. Aktualizuje se záznam: $r = c(\mathbf{x})$. Krok 2 je ukončen a vrátí výsledek *Nalezena celočíselná L-třída.*

Při provádění dalších operací se pracuje pouze s *podproblémem*, který vznikne odstraněním sloupců upevněné části a řádků pokrytých těmito sloupci.

Lagrangeovská heuristika. Pokus o nalezení celočíselné L-třídy se provede aplikací lagrangeovské heuristiky popsané v sekci 5.4.4 na podproblém.

Jestliže aplikací lagrangeovské heuristiky obdržíme řešení $\mathbf{x}$, pro jehož cenu platí $c(\mathbf{x}) + c(f) <= r - 1$, pak se tato cena stane *horní mezí* pro simplexovou metodu. Rovněž obdržíme *dolní mez LB* ceny řešení, kterou porovnáme se záznamem. Simplexová metoda se nespustí, je-li $LB + c(f) > r - 1$. V tom případě se pokračuje heuristikou pro testování skupin L-tříd.

Simplexová metoda. Další pokus o nalezení L-třídy se provede řešením výše uvedené sekvence LP problémů.

Nalezne-li simplexová metoda celočíselné řešení $\mathbf{x}$, je nalezena nová celočíselná L-třída a aktualizován záznam: $r = c(f) + c(\mathbf{x})$. Krok 2 je ukončen a vrátí výsledek *Nalezena*

celočíselná L-třída. Není-li nalezené řešení celočíselné, krok 2 je rovněž ukončen a vrátí výsledek *Nalezena zlomková L-třída*.

Není-li nalezena žádná L-třída a je-li $\varphi = 1$, krok 2 vrátí výsledek *Ukonči algoritmus*. Jinak se pokračuje heuristikou pro testování skupin L-tříd.

5.7.3 Heuristika pro testování skupin L-tříd

To, že není nalezeno řešení problému se stává poměrně často. Úspory výpočetního výkonu se dosahuje heuristikou pro testování skupin L-tříd. Řešíme následující problém:

$$\mathbf{x}' = lexmin\left\{\mathbf{x} \in \Omega : (c, \mathbf{x}) \leq r - 1, x_1 = x''_1, ..., x_{\varphi-1} = x''_{\varphi-1}, x_\varphi = 1\right\}$$
$$\sum_{j=j_0}^{j_o+n_o+1} x_j \geq 1$$

Pro proměnné j_0 a n_0 platí: $j_0 = min\{j < \varphi : x_j = x_{j+1} = ... = x_{j_o+n_o-1} = 0\}$ a neexistuje k takové, že $k \in < j + n_o, \varphi - 1 >$

Na vstupu máme vektor $\mathbf{x}$ a proměnné j_0,n_0,φ. Jestliže hodnoty proměnných j_0 a n_0 odpovídající uvedeným podmínkám nelze najít, krok 2 vrátí výsledek *Nenalezena žádná L-třída*.

Vektor $\mathbf{x}$ lze rozdělit na tři části. Na upevněnou část$(x_1, ...x_{j_0-1})$, část kde musí být alespoň jedna jednička $(x_{j_0}, ..., x_{j_0+n_0-1})$ (označeno tučnými nulami) a volnou část (označeno otazníky).

$$\mathbf{x} = (x_1, x_2, x_3, ..., x_{j_0-1}, \mathbf{0}, \mathbf{0}, ..., \mathbf{0}, \mathbf{0}, ?, ?, ..., ?)$$

Kontrola upevněné části. Nejprve zkontrolujeme upevněnou část - představuje sloupce, o kterých jistě víme, že budou či nebudou součástí pokrytí. Jestliže pro cenu upevněné části $c(f) = (c_1x_1 + c_2x_2 + ... + c_{j_0-1}x_{j_0-1})$ platí $c(f) > r - 1$, pak problém jistě nemá řešení a krok 2 vrátí výsledek *Nenalezena žádná L-Třída*.

Jestliže upevněné sloupce představují pokrytí, pak řešením problému je vektor $\mathbf{x} = (x_1, x_2, x_3, ..., x_{j_0-1}, \mathbf{0}, \mathbf{0}, ..., \mathbf{0}, \mathbf{1}, 0, ..., 0)$, který je zřejmě celočíselný. Byla tedy nalezena celočíselná L-třída, která je zapamatována. Aktualizuje se záznam: $r = c(\mathbf{x}) + c_{j_0+n_0-1}$. Krok 2 je ukončen a vrátí výsledek *Nalezena celočíselná L-třída*.

Při provádění dalších operací se pracuje pouze s *podproblémem*, který vznikne odstraněním sloupců upevněné části a řádků pokrytých těmito sloupci.

Lagrangeovská heuristika. Pokus o nalezení celočíselné L-třídy se provede aplikací lagrangeovské heuristiky popsané v sekci 5.4.4 na podproblém.

Jestliže aplikací lagrangeovské heuristiky obdržíme řešení $\mathbf{x}$, pro jehož cenu platí $c(\mathbf{x}) + c(f) <= r - 1$, pak se tato cena stane *horní mezí* pro simplexovou metodu. Rovněž obdržíme *dolní mez LB* ceny řešení, kterou porovnáme se záznamem. Simplexová metoda se nespustí, je-li $LB > r - 1$.

Simplexová metoda. Další pokus o nalezení L-třídy se provede řešením výše uvedené sekvence LP problémů.

Nalezne-li simplexová metoda celočíselné řešení $\mathbf{x}$, je nalezena nová celočíselná L-třída a aktualizován záznam: $r = c(f) + c(\mathbf{x})$. Krok 2 je ukončen a vrátí výsledek *Nalezena celočíselná L-třída*. Není-li nalezené řešení celočíselné, krok 2 je rovněž ukončen a vrátí výsledek *Nalezena zlomková L-třída*.

Není-li nalezena žádná L-třída, krok 2 vrátí výsledek *Nenalezena žádná L-třída*.

6 Alternativní varianta hybridního algoritmu

Alternativní varianta hybridního algoritmu se liší od původní tím, že místo genetického algoritmu využívá algoritmus simulovaného ochlazování.

Použitý genetický algoritmus má v sobě zabudované použití simplexové metody pro řešení LP problému při operaci křížení. Simplexová metoda je obecně výpočetně náročná. Vzhledem k tomu, že je nutné spolehlivě ověřit, že nalezené řešení je celočíselné, musí se provádět alespoň část kroků simplexové metody v oboru přesně reprezentovaných racionálních čísel, která s sebou nesou další nároky na výpočetní výkon a paměťový prostor.

Simulované ochlazování je založeno na principech podobných genetickému algoritmu (lokální heuristika s možností překonávat lokální minima) a skládá se z velkého počtu opakování poměrně jednoduchých kroků (na rozdíl od genetického algoritmu).

Alternativní varianta hybridního algoritmu si klade za cíl dosáhnout stejných nebo podobných výsledků jako původní varianta prováděním většího množství jednodušších operací.

Algoritmy simulovaného ochlazování používají pro překonání lokálních minim stejně jako genetické algoritmy náhodné změny. Základnímu algoritmu simulovaného ochlazování chybí analogie ke křížení, proto byl pro alternativní variantu hybridního algoritmu vybrán algoritmus simulovaného ochlazování doplněný o určitou analogii křížení - morfing. Tento algoritmus je popsán v [9].

6.1 Principy simulovaného ochlazování

6.1.1 Fyzikální pohled na ochlazování

Proces simulovaného ochlazování lze ilustrovat na příkladu kovu, který je zahřátý na vysokou teplotu a po zahřátí je volně ponechán k pomalému vychladnutí.

V průběhu ochlazování se stav kovu mění. Je možné v něm pozorovat domény, jejichž atributem je energie atomů potřebná k vstupu do domény. Chladne-li kov pomalu, potřebná energie se v doménách příliš neliší a atomy mohou volně procházet mezi doménami a zaujímat pozice s nízkou energií. Chladne-li kov rychle, může se stát, že potřebná energie se v jednotlivých doménách bude lišit natolik, že rozdíl povede k vytvoření pro atomy nepřekonatelné hranice.

Nepřekonatelné hranice se pak mohou stát zlomovými čarami, podél kterých může kov praskat, pokud je ohýbán. Z tohoto důvodu (praskání je obvykle nežádoucí) se kov ochlazuje pomalu, tak aby se atomy mohly přemísťovat. Po celém kovu je pak rozložení minimální energie stejné. Celá masa kovu je pak tvořena jedinou pravidelně uspořádanou doménou.

Představme si atom, který má možnost ze stavu charakterizovaného energií $E1$ přejít do jiného stavu charakterizovaném energií $E2$. Pak pravděpodobnost přechodu do stavu $E2$ je dána vzorcem:

$$p(\rightarrow E2) = min\{1, e^{-\frac{E2-E1}{kT}}\}$$

T je teplota, k je daná konstanta. Nabízí se například Bolzmannova konstanta. Ze vzorce lze vyčíst, že pravděpodobnost přechodu do stavu s nižší energií je velká. Pravděpodobnost přechodu do stavu s vyšší energií je však nenulová. S klesající teplotou se pravděpodobnost přechodu do stavu s vyšší energií zmenšuje.

6.1.2 Pohled z hlediska kombinatorických optimalizačních problémů

Fyzikální pohled	KOP
Stav systému	Řešení
Změna stavu	Přechod k sousednímu řešení
Energie systému	Optimalizační kritérium
Kinetická energie částic	Ochota přejít od lepšího řešení k horšímu
Teplota	Parametr pro řízení algoritmu

6.1.3 Algoritmus simulovaného ochlazování

Simulované ochlazování (SA) je lokální heuristika, která simuluje proces ochlazování pomocí prostředků výpočetní techniky. Než může být tento proces simulován, je třeba rozhodnout následující otázky:

1. Jak budou kódovány jednotlivé stavy?
2. Jak je definováno optimalizační kritérium?
3. Jaký bude mechanismus náhodných změn systému?
4. Jak bude definován řídící parametr T analogický k teplotě?
5. Jaký bude *plán* ochlazování, tedy jak se bude snižovat řídící parametr$T$?

6.2 Algoritmus simulovaného ochlazování pro problém pokrytí (SASCP)

6.2.1 Kódování stavů a optimalizační kritérium

Algoritmus SASCP kóduje jednotlivé stavy jako vektory jedniček a nul (binární vektory). Optimalizačním kritériem je cena pokrytí.

6.2.2 Celkové schéma algoritmu SASCP

Nechť $\mathbf{x}_b$ je nejlepší dosud nalezené řešení a $(c, \mathbf{x}_b)$ jeho cena. Nechť $\mathbf{x}$ je řešení a $(c, \mathbf{x})$ jeho cena.

Krok 0 Získej počáteční řešení problému pokrytí $\mathbf{x}$. a dolní mez LB
Proveď počáteční nastavení: $\mathbf{x}_b = x$, $(c, \mathbf{x}_b) = (c, \mathbf{x})$

Krok 1 Vyhledej souseda $\mathbf{x}'$ řešení $\mathbf{x}$ pomocí procedury SEARCH
$\delta = (c, \mathbf{x}')$, $\omega = rand < 0, 1)$

Krok 2 Rozhodni o přijetí souseda
Jestliže $\delta \leq 0$ nebo $\omega < e^{-\frac{\delta}{T}}$, nastav $\mathbf{x} = \mathbf{x}'$ a jestliže $(c, \mathbf{x}) < (c, \mathbf{x}_b)$, nastav $(c, \mathbf{x}_b) = (c, \mathbf{x})$ a $(\mathbf{x}_b = \mathbf{x})$
Je-li splněna nerovnost $ZMAX \geq (c, \mathbf{x}_b)$, algoritmus je ukončen

Krok 3 Nebylo-li dosaženo daného počtu iterací, pokračuj krokem 1

Krok 4 Sniž teplotu. $T = T \cdot constant$
Byla-li dosažena koncová teplota, algoritmus je ukončen

6.2.3 Procedura SEARCH (hledání sousedního stavu)

Krok 1 Odstraň sloupec z řešení pomocí procedury DEFLECT

Krok 2 Proveď morfing pomocí procedury MORPH

Krok 3 Vytvoř řešení problému pokrytí pomocí procedury CONSTRUCT

Krok 4 Odstraň nadbytečné sloupce z řešení pomocí procedury REDUND

6.2.4 Procedura DEFLECT

Procedura DEFLECT simuluje náhodné změny v systému tím, že odstraňuje sloupce z řešení. Počet volání procedury DEFLECT se zaznamenává. Je-li počet volání dělitelný třemi, z řešení se odstraní sloupec, který poskytuje nejméně unikátních pokrytí řádků. Není-li počet volání dělitelný třemi, z řešení je odstraněn náhodně vybraný sloupec. Výsledkem procedury DEFLECT je částečné řešení problému pokrytí.

6.2.5 Procedura CONSTRUCT

Vzhledem k tomu, že procedura DEFLECT poskytuje částečné řešení problému pokrytí, je třeba k tomuto řešení přidat vhodně vybrané sloupce.

K doplnění se využívají dva hladové algoritmy popsané v sekcích 4.3 a 4.4. Po přidání každého čtvrtého sloupce do pokrytí je na částečné řešení aplikována procedura MORPH. Hladové algoritmy se střídají.

6.2.6 Procedura REDUND

Procedura slouží k odstranění nadbytečných sloupců z řešení vytvořeného procedurou CONSTRUCT. Jednotlivé sloupce se probírají od nejdražšího k nejlevnějšímu a testuje se zda je možné odstranit je z řešení.

6.2.7 Procedura MORPH

Procedura MORPH slouží k provádění *morfování*. Morfování, podobně jako křížení v genetických algoritmech, představuje prostředek, který má zajistit pokud možno rychlou konvergenci k optimálnímu řešení.

Tak jako se křížení v genetických algoritmech snaží přenést nejlepší vlastnosti rodičů k jejich potomkům, morfování si klade za cíl provádět to samé v algoritmu simulovaného ochlazování [9].

Pro každý sloupec, který se stal součástí řešení, je vytvořen seznam omezeného počtu "podobných sloupců", tzv. *morfů*.
Podobnost dvou sloupců p a q je definována vzorcem:

$$m(p,q) = \frac{V_p^T V_q}{V^1 V_q}$$

V^1 je řádkový vektor jedniček, V_p je sloupcový vektor matice problému pokrytí $(a_{1p}, a_{2p}, ..., a_{mp})$ a V_q je sloupcový vektor $(a_{1q}, a_{2q}, ..., a_{mq})$ matice problému pokrytí.

Do seznamu morfů jsou přidány pouze morfy, pro něž platí nerovnost $m(a, b) \geq 0,6$. Každý sloupec může mít v seznamu až 30 morfů.

Vlastní morfování probíhá následovně:

Krok 1 Vyber sloupec, který je součástí řešení

Krok 2 Pro všechny morfy (existují-li) vybraného sloupce vypočítej poměr:

$$\frac{cena\,částečného\,řešení}{počet\,pokrytých\,řádků}$$

Krok 3 Vyber morf s nejvyšším poměrem (existuje-li) a nahraď jím vybraný sloupec.

7 Implementace

Hybridní algoritmus pro problém pokrytí je implementován ve formě knihovny napsané v ANSI dialektu jazyka C s využitím techniky modulárního programování. Jazyk C byl zvolen zejména kvůli rychlosti provádění přeloženého programu.

Vzhledem k tomu, že překladače jazyka C jsou dostupné již od druhé poloviny sedmdesátých let 20. století a jsou dodnes velmi často používány, tvůrci překladačů měli dostatek času na vylepšování efektivity generovaného strojového kódu. Nezanedbatelnou výhodou je také množství platforem, pro které je překladač ANSI dialektu jazyka C k dispozici.

7.1 Moduly

Hlavní moduly

Vlastní knihovna se skládá z dále stručně popsaných modulů:

Modul hybridního algoritmu. Modul poskytující rozhraní pro aplikace využívající knihovnu. Umožňuje předat vstupní problém, nastavit parametry algoritmu a předat zpět výsledné řešení.

Modul hladových algoritmů. Implementace dvou druhů hladových algoritmů pro problém pokrytí. První algoritmus je hladový algoritmus pro problém pokrytí dle Chvátala, druhý je algoritmus podle Balase a Hoa - viz sekce 4.3 a 4.4. Modul je schopen spolupráce s modulem simulovaného ochlazování při morfování.

Modul lagrangeovské heuristiky. Implementace pěti variant Lagrangeovské heuristiky pro problém pokrytí.

Modul genetického algoritmu. Implementace nebinárního genetického algoritmu s nastavitelnými parametry. Modul spolupracuje s modulem hladových algoritmů a také s modulem pro spolupráci s externí implementací simplexové metody.

Modul simulovaného ochlazování. Implementace simulovaného ochlazování s morfováním. Modul spolupracuje s modulem hladových algoritmů.

Modul pro spolupráci s externí implementací simplexové metody. Modul zajišťuje spolupráci s knihovnou GLPK, která je určena pro řešení problémů lineárního programování simplexovou metodou.

Modul výčtu L-tříd. Implementace algoritmu "Výčet L-tříd", modul spolupracuje s modulem lagrangeovské heuristiky a s modulem pro spolupráci s externí implementací simplexové metody.

Modul pohledu na problém pokrytí. Implementace pohledu na problém pokrytí. Nad pohledy na problém pokrytí je definováno množství operací.

Pomocné moduly

Pomocné moduly jsou široce využívány v různých částech knihovny, jejich význam je zřejmý z jejich názvů.

- Modul pro reprezentaci problému pokrytí
- Pomocný modul pro řazení metodou QuickSort
- Pomocný modul pro načítání problémů pokrytí z textových souborů
- Modul poskytující implementaci spojového seznamu celých čísel

7.2 Datové struktury

V knihovně jsou většinou používány velmi jednoduché datové struktury.

Problém pokrytí

K reprezentaci problému pokrytí se používá dvourozměrné pole, k reprezentaci vektoru cen pole jednorozměrné.

Vzhledem k tomu, že matice problému pokrytí může obsahovat velké množství prvků, je nepřijatelné, aby bylo při běhu programu vytvářeno mnoho jejích kopií. Dále je nepřijatelné, aby se prováděly přesuny sloupců nebo řádků (například při řazení metodou QuickSort). Z tohoto důvody jsou zavedeny různé *pohledy na problém pokrytí*.

Masky

V modulech se často používá jednoduchá datová struktura - binární vektor či *maska*, která je vstupem nebo výstupem různých operací.

Pohledy na problém pokrytí

Hlavním cílem pohledů na problém pokrytí je šetření paměťového prostoru. Pohledy obsahují odkaz na reprezentaci problému pokrytí a zejména *mapovací pole*, která mapují souřadnice řádků a sloupců matice pohledu na souřadnice řádků a sloupců matice problému pokrytí. Navíc jsou uchovávána pole pro mapování na rodičovský pohled. Lze tedy vytvořit určitou hierarchii spolupracujících pohledů.

Při řazení a odstraňování sloupců či řádků jsou měněna pouze mapovací pole; přesunuje se tedy relativně malé množství dat. Tak se ušetří paměťový prostor, ale i výpočetní výkon. Pohled je datová struktura o velikosti:

$$V = 2m * sizeof(int) + 2n * sizeof(int) + 2 * sizeof(int) + PTR$$

kde m je počet řádků, n počet sloupců matice problému pokrytí a PTR velikost datového typu ukazatel.

Nad pohledy pokrytí je možné provádět množství operací, jako například odstranění dominancí, tvorba podpohledů definovaných maskami sloupců či řádků, odstranění nezbytných sloupců s vrácením jejich seznamu, detekce zda je nějaká maska řešením problému pokrytí, klonování pohledů.

7.3 Simplexová metoda

Pro řešení LP problémů v kroku 2 algoritmu LCE se používá simplexová metoda implementovaná v knihovně GLPK. Původní rozhodnutí pro vlastní implementaci bylo zavrženo, protože tvorba kvalitní implementace simplexové metody s sebou přináší množství úskalí - hlavními problémy jsou numerická stabilita a zejména degenerace - viz [10].

Vzhledem k tomu, že v algoritmu LCE je naprosto klíčové rozpoznávání celočíselných a neceločíselných řešení, je vyloučeno spoléhat se na nepřesně reprezentovaná reálná čísla odpovídající datovým typům *float* a *double* jazyka C.

Na druhou stranu práce s přesnými racionálními čísly je výpočetně a paměťově náročná, protože po každé aritmetické operaci definované nad těmito čísly se musí provádět jejich normalizace (například euklidovský algoritmus pro hledání společného jmenovatele). Při užití simplexové metody se provádí mnoho operací dělení, z čehož plyne, že jmenovatelé racionálních čísel mohou značně narůstat.

Z popsaných důvodů je použit kombinovaný postup. Nejprve se nalezne optimální báze řešení pomocí operací v prostoru nepřesných reálných čísel a dále se pak pokračuje operacemi v prostoru přesných racionálních čísel. Takový postup je přímo doporučen v dokumentaci knihovny GLPK.

Modul pro spolupráci s externí implementací simplexové metody lze snadno upravit tak, aby využíval i jiné externí implementace, například knihovnu lp_solve.

7.4 Kontrolní mechanismy

V případě potřeby lze knihovnu přeložit se začleněním mechanismu vynucených podmínek (assertions). Při ladění knihovny byla intenzivně používána sada nástrojů VALGRIND. Knihovna je prosta paměťových úniků a chyb vyplývajících ze značné volnosti přístupu k paměti v jazyce C.

7.5 Začlenění do programu BOOM

Při začleňování knihovny do program BOOM byl kladen důraz na to, aby nebylo nutné měnit kód knihovny samotné. Z tohoto důvodu byla podle pokynů vedoucího práce do programu BOOM přidána přizpůsobovací třída `HybridCover` s přesně definovaným rozhraním, která má dva úkoly - zajistit volání funkcí poskytovaných knihovnou a konvertovat reprezentace problému pokrytí a jeho řešení.

Vzhledem k tomu, že program BOOM používá odlišnou reprezentaci problému pokrytí (spojové seznamy proti matici), byl přidán krátký kód provádějící převod jedné reprezentace do druhé. Totéž platí i pro reprezentaci řešení problému pokrytí, které musí být převedeno z formy masky do formy spojového seznamu indexů sloupců.

8 Výpočetní experimenty

9 Závěr

Hybridní algoritmus pro problém pokrytí byl podrobně popsán pro potřeby implementace a společně s jeho alternativní variantou implementován ve formě knihovny psané v ANSI dialektu jazyka C a s pomocí jednoduché adaptační třídy začleněn do programu BOOM. Srovnáním výsledků s algoritmem AURA bylo ověřeno, že hybridní algoritmus pracuje *korektně* a jeho výsledkem je optimální řešení problému pokrytí. Algoritmus je schopen řešit problémy pokrytí i s nejednotkovými cenami sloupců. Ve srovnání s algoritmem AURA je hybridní algoritmus velmi pomalý.

Základní výpočetní experimenty ukázaly nepříjemnou skutečnost. Lagrangeovská heuristika určená pro urychlování výčtu L-tříd neplní dobře svoji funkci, protože jen výjimečně zabrání spuštění simplexové metody, nebo sníží hodnotu záznamu - viz 5.6. Z tohoto důvodu byla Lagrangeovská heuristika z výčtu L-tříd při dalších experimentech vyřazena.

Z provedených výpočetních experimentů vyplývá, že je třeba nejprve najít výkonnostní rezervy v mé implementaci hybridního algoritmu - zjištěné chování Lagrangeovské určené pro urychlování výčtu L-tříd heuristiky naznačuje, že by takové rezervy mohly existovat. Teprve v případě, že by tyto rezervy byly nedostatečné, hybridní algoritmus pro problém pokrytí nepočítat k dnes moderním metodám jeho řešení.

Literatura

[1] A. V. Eremeev, A. A. Kolokolov, L. A. Zaozerskaya: A hybrid algorithm for set covering problem. In Proc. of International Workshop "Discrete Optimization Methods in Scheduling and Computer-Aided Design, pp. 123—129

[2] http://www.wikipedia.org, leden 2008

[3] Demel: Grafy a jejich aplikace, Academia 2002

[4] Krejčík: Moderní metody řešení problému pokrytí, bakalářská práce, 2005.

[5] Schmidt: Problémy a algoritmy, přednáškové slidy

[6] Chvátal: V. A Greedy Heuristic for the Set Covering Problem, Mathematics of Operations Research, vol. 4, No 3, 1979

[7] Techapichetvanich, Bricker: INVESTIGATION OF LAGRANGIAN HEURISTICS FOR SET COVERING PROBLEMS, Department of Industrial Engineering, The University of Iowa,Iowa City, IA 52242,USA,May 1993

[8] Dudorkin: Operační výzkum, Praha 1991

[9] Brusco, Jacobs, Thompson: A morphing procedure to supplement a simulated annealing heuristic for cost and coverage correlated set-covering problems, Annals of Operations research 86, 1999

[10] Koberstein: The Dual simplex method, techniques for fast and stable implementation, disertační práce, 2005

[11] GNU Linear programming kit, Reference manual version 4.35, 2009

[12] Krejčík: Moderní metody řešení problému pokrytí, diplomová práce, 2007

[13] E. I. Goldberg, L. P. Carloni, T. Villa , R. K. Brayton: Negative thinking by incremental problem solving: application to unate covering. Proceedings of IEEE International Conference on Computer Aided Design (ICCAD, San Jose, CA, USA, 9-13 Nov. 1997). Los Alamitos, CA, USA:IEEE Comput. Soc, 1997. pp. 91—99

[14] L. P. Carloni, E. I. Goldberg, T. Villa, R. K. Brayton and A. L. Sangiovanni-Vincentelli: Aura II: Combining Negative Thinking and Branch-and-Bound in Unate Covering Problems. In "VLSI: Systems on a Chip" (L.M. Silveira, R. Reis, S. Devadas editors), Kluwer 1999

A Příklad použití knihovny hybridscp

Použití knihovny hybridscp v programech je přímočaré, spočívá v provedení následujících kroků:

1. Ke zdrojovým textům je třeba připojit hlavičkový soubor `hybrid.h`
2. Vytvoří se instance problému pokrytí a naplní se daty
3. Vytvoří se instance hybridního algoritmu, předat se jí problém pokrytí, a spustí výpočet
4. Po dokončení výpočtu se vyzvedne řešení ve formě masky

```
#include <hybrid.h>
#define REMOVE DOMINANCES 1
int main(int argc,char* argv) {

    SCProblem* problem=NULL;
    Hybrid* hybrid=NULL;
    unsigned char* solution;
    int result=0;
    problem=scproblem_new(20,10);
    scproblem_set(problem,2,4,1);
    ...
    scproblem_set_cost(problem,1,2);
    ...
    hybrid=hybrid_new(problem);
    hybrid_set_parameters(hybrid,...);
    hybrid_set_sa_parameters(hybrid,...);
    ...
    result=hybrid_solve(hybrid,REMOVE_DOMINANCES);
    switch (result)
        case HYBRID_OK : {
            solution=hybrid_get_solution(hybrid);
            break;
        }
        case HYBRID_NOTSOLVABLE: {
            printf("Problem not solvable");
        }
        case HYBRID_IMPLICIT: {
            printf("Not necessary to solve");
        }
    }
    return (EXIT_SUCCESS)

}
```