Katedra informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Minimálne hranové rezy v hyperkockách

(Bakalárska práca)

Peter Glaus

Vedúci:
doc. RNDr. Rastislav Královič, PhD.

Bratislava, 2007

Čestne prehlasujem, že som túto bakalársku práca vypracoval samostatne s použitím citovaných zdrojov.

. . . . . .

Poďakovanie

V prvom rade by som chcel poďakovať môjmu školitelovi a zároveň zadávateľovi tejto práce, za inšpiratívne nápady a pomoc pri mojej práci.

Tiež chcem poďakovať rodičom za dobrú starostlivosť a svojej priateľke za trpezlivosť a podporu v najťažších chvíľach.

Abstrakt

V úlohe nájdenia minimálneho k-rezu grafu sa snažíme rozdeliť graf na k súvislých komponentov odobraním hrán najmenšej váhy. Pri neohodnotenom grafe sa snažíme minimalizovať počet odobraných hrán. V tejto práci sa zameriame na neohodnotené grafy a na konkrétnu podmnožinu grafov - hyperkocky. Ďalšie zúženie problému bude spočívať vo veľkosti komponentov. Snažíme sa nájsť minimálne rozdelenie n-rozmernej hyperkocky na k komponentov, ktorých veľkosť sa líši o konštantu. Čiastočne nadviažeme na výsledky z [Bez96], kde teóriou dokázali dolné ohraničenie a pre niektoré k aj zodpovedajúce horné ohraničenie.

V práci sú použité dva rôzne algoritmické prístupy k riešeniu úlohy. Jeden je heuristické simulované žíhanie prispôsobené na problém minimálneho delenia. Druhý problém využíva optimálne podgrafy, ktoré sa snaží ukladať do hyperkocky. Pre vhodné n tieto výsledky dopĺňajú teoretické výsledky z práce Sergeia Bezrukova [Bez96].

Kľúčové slová: delenie grafu na k častí, minimálny rez hyperkocky, heuristika, distribuované výpočty

Obsah

Kapitola 1

Úvod

1.1 Minimálny k-rez hyperkocky

Problém minimálneho rezu grafu je jeden zo základných optimalizačných problémov teórie grafov. Od neho je odvodený aj problém minimálneho rezu grafu na k častí. Pri riešení tohto problému sa snažíme nájsť množinu hrán, ktorej odobratím vytvoríme v grafe k súvislých komponentov. Okrem toho sa snažíme, aby táto množina bola minimálna. To znamená, že mohutnosť tejto množiny musí byť minimálna, resp. ak je graf ohodnotený, musí byť súčet váh hrán tejto množiny čo najmenší.

O tomto probléme bolo dokázané, že je NP-ťažký[GH94]. Bol nájdený deterministický algoritmus riešiaci túto úlohu s časovou zložitosťou $O(|V|^{k^2})$. Z toho dôvodu bolo veľa úsilia zameraného na nájdenie časovo efektívnejších aproximačných algoritmov, riešiacich tento problém [SV95][RS02][NR01]. Zatiaľ najlepšie riešenie priniesli Saran a Vazirani v [SV95], kde uvádzajú $(2 - 2/k)$ aproximačný algoritmus.

Jedna z motivácií hľadania minimálneho k-rezu prichádza z teórie distribuovaných systémov a paralelných výpočtov. Pri riešení fyzikálnych úloh pomocou metódy konečných prvkov, sa snažíme riešiť spojitý problém aproximáciou cez veľké množstvo konečných prvkov. Tieto prvky a ich spojenia tvoria graf s veľmi veľkým počtom vrcholov. Ak chceme takúto úlohu riešiť pomocou distribuovaného systému, ktorý má vo väčšine prípadov počet

procesorov rádovo menší ako počet vrcholov grafu, musíme graf rozdeliť na komponenty. Pri takomto rozdelení máme dve základné požiadavky. Prvá je minimálna komunikácia medzi procesormi, z toho vyplýva minimálnosť rezu. Tá druhá je približne rovnaké vyťaženie všetkých procesorov. Druhá požiadavka nám pridáva ďalšiu podmienku a tou je približne rovnaká veľkosť všetkých k komponentov.

Zameranie sa na hyperkocky má viacero dôvodov. Hyperkocka, ktorá má n rozmerov, je regulárny graf s 2^n vrcholmi, z čoho každý má n hrán. Okrem toho je symetrický, má malý polomer a každému vrcholu vieme priradiť číslo od 0 po $2^n - 1$, ktoré presne určí jeho susedov. Z tohto dôvodu sa dajú vytvoriť presnejšie odhady na minimálny počet prerezaných hrán. Výpočty na hyperkocke sú rovnako ľahšie zvládnuteľné, keďže hyperkocka má presnú organizáciu vrcholov a hrán, o ktorú sa dá oprieť.

Nájdenie minimálneho rezu hyperkocky na k rovnako veľkých komponentov môže mať aj ďalšie využitie. V distribuovaných systémoch boli hyperkocky a výpočtové modely usporiadané do tohto tvaru často využívané [Din88][BF88]. Ak by sme algoritmus navrhnutý pre hyperkocku chceli preniesť na iný distribuovaný systém s menším počtom procesorov, tak nás bude zaujímať hlavne prerozdelenie úloh. Jednotlivé procesory menšieho distribuovaného systému budú musieť nahradiť prácu viacerých preocesorov z hyperkocky. Opäť chceme, aby boli rovnako vyťažené a aby sme minimalizovali množstvo komunikácie medzi rôznymi procesormi.

V tejto práci riešim problém nájdenia minimálneho rezu hyperkocky algoritmicky. Navrhnuté algoritmy by sa dali zaradiť medzi optimalizačné heuristiky. To znamená, že nájdené riešenia nie sú optimálne, ale sú to iba lokálne optimá. Na rozdiel od aproximačných algoritmov, nie je možné ani zaručiť hornú hranicu optimálnosti nájdeného riešenia. Výhodou týchto algoritmov je ich rýchlosť, pretože sú značne rýchlejšie ako by bolo deterministické riešenie.

1.2 Členenie práce

V kapitole 2 sa budeme venovať teoretickým výsledkom. Prezentujem výsledky Sergeia Bezrukova [Bez96]. Ten sa zameral na teoretické odvodenie dolnej a hornej hranice pre minimálny k-rez hyperkocky. Podarilo sa mu odvodiť vzťah pre dolnú hranicu a pre niektoré k našiel aj hornú hranicu minimálneho rezu.

Samotné výsledky práce sa nachádzajú v kapitole 3 a 4. V tejto práci nadväzujem na výsledky z [Bez96]. Snaha je potvrdiť teoretické výsledky, výsledkami experimentálnymi. Tak isto sa pokúsim odhadnúť hornú hranicu aj pre také k, pre ktoré sa nepodarilo dokázať rozumné horné ohraničenie teoreticky.

V kapitole 3 je popísané priame hľadanie rezu hyperkocky na k rovnako veľkých komponentov. Táto metóda je presnejšia a je použiteľná pre hyperkocky menších rozmerov. Využívam heuristickú metódu, simulované žíhanie. Pri väčších hyperkockách je výpočtovo náročná a nájdené riešenia sú len málo kvalitné.

Kapitola 4 je zameraná na lepšie využitie vlastnosti hyperkocky. Algoritmus opísaný v tejto kapitole nehľadá minimálny rez priamo. Je založený na princípe uloženia optimálnych častí. Tieto časti sú rozdelené do hyperkociek menších rozmerov, ktoré algoritmus umiestňuje do hyperkocky. Postupuje od najväčších častí k menším a snaží sa umiestniť nasledujúcu podkocku tak, aby sa neprekrývala s inou. Okrem toho musí nadväzovať na ostatné časti z jej komponentu. Takýto prístup má výhodu uloženia veľa vrcholov naraz. Nevýhoda je prehľadávanie všetkých možností.

Rôzne prístupy k riešeniu tohto problému sú konkrétne porovnané v kapitole 5. Jednotlivé algoritmy sú porovnávané medzi sebou a takisto s dolnou hranicou.

Kapitola 2

Teoretické výsledky

2.1 Základné definície

Najprv si musíme zadefinovať pojem n rozmernej hyperkocky a problémy, ktoré budeme neskôr riešiť.

Definícia 1 *Hyperkocka $Q(n)$ je graf $G(V, E)$, ktorý má nasledovné vlastnosti. Počet vrcholov aj hrán je presne daný: $|V| = 2^n$, $|E| = n \times 2^{n-1}$. Každý vrchol má práve n hrán. Navyše, ak očíslujeme vrcholy grafu $K(n)$ číslami od 0 po $n-1$, tak medzi vrcholmi v_i a v_j je hrana len ak i_{bin} a j_{bin} sa líšia v práve jednom bite.*

Očíslovanie vrcholov a pohľad na očíslovanie v dvojkovej sústave sú veľmi dôležité. Je to tá vlastnosť, ktorá robí graf hyperkockou a zároveň nám zjednodušuje prácu s ňou. Podobne ako si rovinné grafy vieme nakresliť na rovinu, tak aby sa hrany nepretínali, a tým ich sprehľadniť, si vieme hyperkocku predstaviť v priestore. Konkrétne v n rozmernom priestore. Každý bit označenia vrcholu nám určuje pozíciu vrchola v jednom rozmere. Keď sa neskôr budeme baviť o rozmere v kocke bude tým myslený práve bit prislúchajúci tomu rozmeru.

Pre potreby tejto práce si zadefinujeme pojem delenie hyperkocky. Delenie hyperkocky bude v podstate vyjadrovať k-rez n-rozmernej hyperkocky. Na rozdiel od k-rezu určíme aj veľkosti vzniknutých komponentov pri hy-

perkocke. Vzhľadom na to, že sa celý čas bavíme a budeme baviť o delení hyperkociek, budeme uvažovať hyperkocky $Q(n), n \geq 1$.

Definícia 2 *Delenie hyperkocky $Q(n)$ $\mathscr{K} = \{K_1, \ldots, K_k\}$, je k-rez, pri ktorom sa veľkosť vzniknutých komponentov líši najviac o 1. Označením $\nabla\mathscr{K}$ budeme označovať mohutnosť tohto k-rezu.*

Z predchádzajúcej definície potom pre delenie hyperkocky $\mathscr{K}$ vyplýva nasledovný vzťah

$$\left\lfloor \frac{2^n}{k} \right\rfloor \leq K_i \leq \left\lceil \frac{2^n}{k} \right\rceil$$

Nasledujú definície dvoch problémov. Ako sa neskôr ukáže, problémy sú ekvivalentné. Prvý z nich je problém minimálneho delenia hyperkocky. Druhý je problém uloženia množín do hyperkocky.

Problém 1 *Zoberme hyperkocku $Q(n)$ a pevne zvolené k. Minimálne delenie hyperkocky je delenie hyperkocky $\mathscr{K}$ na k častí $K_1, \ldots, K_k$, také že mohutnosť rezu $\nabla\mathscr{K}$ je minimálna.*

Pre pevne zvolené k a n si označme $\nabla(n,k) = \min_{\mathscr{K}} \nabla\mathscr{K}$.

Problém 2 *Zoberme hyperkocku $Q(n)$ a pevne zvolené k, (n,k)-uloženie je uloženie k množín veľkosti $\left\lfloor \frac{2^n}{k} \right\rfloor$ do hyperkocky. Úlohou je nájsť minimálne uloženie $\mathscr{U}$ vzhľadom na počet hrán, ktoré spájajú dva vrcholy rôznych množín.*

Opäť si pre uloženie $\mathscr{U}$ označme počet hrán, ktoré spájajú dva vrcholy rôznych množín, ako $\nabla\mathscr{U}$. Pre n a k označme $\nabla'(n,k) = \min_{\mathscr{U}} \nabla\mathscr{U}$. Je zrejmé, že pre dosť veľké n sa $\nabla(n,k)$ a $\nabla'(n,k)$ sa líšia minimálne.

Táto druhá definícia nám dáva možnosť iného prístupu k riešeniu toho istého problému. Namiesto toho aby sme hľadali delenie, ktoré by bolo minimálne, budeme sa snažiť správne uložiť menšie množiny vrcholov. Ak sa nám podarí uložiť množiny veľkosti $\left\lfloor \frac{2^n}{k} \right\rfloor$, tak zostávajúce vrcholy, už neurobia výrazný rozdiel v mohutnosti rezu. Takýto prístup riešenia problému minimálneho delenia je použitý aj v algoritmoch popísaných v kapitole 4.

2.2 Dolné ohraničenie

Pred tým ako si bližšie určíme dolnú resp. hornú hranicu minimálneho delenia hyperkocky, skúsme sa najprv pozrieť na jednoduchšie konkrétne prípady resp. jednoduchšie problémy.

Zoberme konkrétne $k = 2$. Keď delíme hyperkocku na dve polovice rovnakej veľkosti, tak najmenší rez, ktorý môžeme spraviť, je ak kocku rozdelíme pozdĺž nejakého rozmeru. Tým je myslené to, že ak použijeme očíslovanie vrcholov v dvojkovej sústave, tak rozdelíme vrcholy na tie, ktoré majú i-ty bit rovný nule a tie, ktoré majú i-ty bit rovný jednej. V takomto prípade je $\nabla\mathscr{K} = \frac{1}{2}2^n$. Toto vcelku intuitívne tvrdenie si neskôr dokážeme vďaka vete z [Har64]. Pristúpme teda k zjednodušeniu problému.

Definícia 3 *Pre $A \subseteq Q(n)$ definujme $\partial A = \{(u, v) \in E_{Q(n)} | u \in A, v \in Q(n) \setminus A\}$.*

Podgraf $A \subseteq Q(n)$ je optimálnym ak platí $\forall B \subseteq Q(n), |B| = |A| \Rightarrow |\partial B| \leq |\partial A|$.

Inými slovami pre dané k je podgraf A veľkosti k optimálny ak je rez, ktorý ho oddeľuje najmenší možný. Treba si uvedomiť, že ak by sa nám podarilo rozdeliť hyperkocku na k optimálnych častí, tak by toto delenie bolo minimálne. Táto úvaha je zhrnutá v nasledujúcej leme. Tá sa ukáže byť veľmi dôležitá, pretože ju použijeme vo viacerých neskorších tvrdeniach.

Lema 1 *Ak pre všetky komponenty $K_1, \ldots, K_k$, vytvorené delením $\mathscr{K}$, platí, že sú optimálne, potom je $\mathscr{K}$ minimálne delenie.*

Dôkaz: Lema jasne vyplýva z definícií minimálneho delenia a optimálneho podgrafu.

Keby boli komponenty $K_1, \ldots, K_k$, optimálne a delenie $\mathscr{K}$ by nebolo minimálne, muselo by existovať delenie $\mathscr{K}', \nabla\mathscr{K}' < \nabla\mathscr{K}$. Keďže sa jedná o delenia hyperkocky podľa definície 2, musíme vedieť očíslovať komponenty delenia $\mathscr{K}'$ tak, aby platilo $|K_i| = |K'_i|, \forall i = \overline{1, n}$. Aby platil vzťah $\nabla\mathscr{K}' < \nabla\mathscr{K}$ musí existovať i také, že $\partial K_i > \partial K'_i$ čo je spor s minimalitou $K_1, \ldots, K_2$. □

Pre hyperkocku $Q(n)$ a jej vrchol u označme $l(u)$ ako číslo vrchola u pri konkrétnom očíslovaní vrcholov hyperkocky v zmysle definície 1. Následne definujme $L_m^n = \{u \in Q(n) | 0 \leq l(u) \leq m\}$. Teda podgraf L_m^n obsahuje prvých m vrcholov hyperkocky $Q(n)$.

Dva podgrafy A a B hyperkocky $Q(n)$ nazveme kongruentné a označíme $A \cong B$ ak pre nejaký automorfizmus $Q(n)$ je podgraf B obrazom podgrafu A. V hyperkocke vieme vďaka jej symetrickosti vytvárať automorfizmy veľmi jednoducho. Očíslujme vrcholy v binárnej sústave. Následne vieme vytvárať automorfizmy zmenou poradia bitov všetkých vrcholov. Okrem toho sa dajú automorfizmy vytvoriť aj negáciou i-teho bitu resp. viacerých bitov.

Veta 1 ([Har64]) *L_m^n je optimálny podgraf hyperkocky $Q(n)$ pre ľubovoľné $1 \leq m \leq 2^n$. Navyše ak A je optimálny podgraf $Q(n)$ a $|A| = m$, tak $A \cong L_m^n$.*

Dôsledok 1 *Dôsledok tejto vety a lemy 1 je už spomínané minimálne delenie pre $k = 2$*

$$\nabla(n, 2) = \frac{1}{2} 2^n$$

Dôkaz: Ak rozdelíme $Q(n)$ na $L_{2^{n-1}}^n$ a zvyšok, tak podgraf $Q(n) \setminus L_{2^{n-1}}^n$ je zjavne kongruentný s $L_{2^{n-1}}^n$. To znamená, že oba vzniknuté komponenty sú optimálne. Ďalej vieme povedať, že každý vrchol má práve jednu hranu súčasťou rezu teda počet prerezaných hrán je $\nabla\mathscr{K} = \frac{1}{2} 2^n$. □

Práve sme určili mohutnosť minimálneho delenia hyperkocky pre ľubovoľné n a $k = 2$. Z tohto dôvodu budeme v ďalšom texte uvažovať iba $k > 3$. Keď už vieme ako vyzerá optimálny podgraf hyperkocky, bolo by vhodné vedieť aj to koľko hrán obsahuje rez, ktorý tento podgraf oddelí od hyperkocky. Pri neskoršom použití lemy 1 a teda vzťahu minimálneho delenia a optimálneho podgrafu, sa nám táto informácia bude hodiť.

Veta 2 ([Kra02]) *Označme $g(n, m) = |\partial L_m^n|$. Potom platí nasledujúca rekurencia.*

$$g(n, m) = \begin{cases} g(n-1, m) + m & , m < 2^{n-1} \\ g(n-1, m - 2^{n-1}) + 2^n - m & , m > 2^{n-1} \\ 2^{n-1} & , m = 2^{n-1} \end{cases}$$

Dôkaz: Rekurencia ráta hrany postupným delením hyperkocky pozdĺž jednotlivých rozmerov. Hyperkocku rozdelíme na dve polovice a zrátame počet hrán, ktoré patria do rezu a ich vrcholy patria do tej istej polovice. K tomu prirátame počet hrán, ktoré patria do rezu a ich vrcholy patria do rôznych polovíc. Vďaka tomu, že L_m^n obsahuje prvých m vrcholov sa nikdy nestane, že by obe polovice obsahovali aj vrcholy z L_m^n aj vrcholy z $Q(n) \setminus L_m^n$.

Rozdeľme hyperkocku na polovicu podľa n-tého rozmeru inak povedané podľa najdôležitejšieho bitu.

- Ak $m = 2^{n-1}$, tak všetky vrcholy v jednej vzniknutej polovici patria do L_m^n, pretože to je prvých 2^{n-1} vrcholov. Takže v tejto polovici nie je rezom, ktorý oddeľuje L_m^n, "prerezaná" žiadna hrana. V druhej polovici naopak nie je žiadny vrchol, ktorý by patril do L_m^n, takže ani tam nie je žiadna hrana súčasťou rezu. Medzi jednotlivými polovicami prechádza 2^{n-1} hrán, ktoré sú zjavne všetky súčasťou rezu.

$$g(n, 2^{n-1}) = 2^{n-1}$$

- Ak $m < 2^{n-1}$, tak prvých m vrcholov patriacich do L_m^n je v jednej z polovíc a v tej druhej nie sú žiadne. Preto je v prvej polovici $g(n-1, m)$ hrán patriacich do rezu, a v druhej nie sú žiadne. Teraz treba prirátať hrany patriace rezu, ktoré prechádzajú medzi polovicami. Vďaka vlastnosti hyperkocky vieme, že každý vrchol z L_m^n má práve jedného suseda v druhej polovici a tento sused nepatrí do L_m^n teda hrana je súčasťou rezu. Preto v tomto prípade platí

$$g(n, m) = g(n-1, m) + m$$

- Posledná možnosť je ak $m > 2^{n-1}$. V tomto prípade v jednej polovici budú všetky vrcholy patriť do L_m^n a v druhej bude do L_m^n patriť presne $m - 2^{n-1}$ vrcholov. Z predchádzajúcich prípadov vieme, koľko hrán v jednotlivých poloviciach patrí do rezu. Okrem toho každý vrchol z každej polovice má práve jedného suseda v opačnej polovici. Zoberme polovicu, ktorá je celá podmnožinou L_m^n. V tejto polovici potom musí

mať presne $m - 2^{n-1}$ vrcholov svojho suseda v opačnej polovici, ktorý tiež patrí do L^n_m. Zvyšné vrcholy takého nemajú a teda ich hrany patria do rezu. Mohutnosť rezu v tomto prípade je

$$\begin{aligned} g(n,m) &= g(n-1, m-2^{n-1}) + 2^{n-1} - (m - 2^{n-1}) \\ &= g(n-1, m-2^{n-1}) + 2^n - m \end{aligned}$$

□

Dôsledok 2 $\nabla(n, 2^l) = \frac{l}{2}2^n$

Dôkaz: Keďže $k = 2^l$ je v tomto prípade mocninou čísla 2, tak každá z k častí bude obsahovať presne 2^{n-l} vrcholov. Navyše vytvorme časti tak, že kocku predelíme pozdĺž prvých l rozmerov. To znamená, že prvých l bitov označenia vrchola bude ovplyvňovať, do ktorej časti vrchol patrí. Takto sme vytvorili 2^l častí veľkosti 2^{n-l}. Teraz vieme spraviť pre každú časť taký automorfizmus hyperkocky $Q(n)$, aby jej obrazom bol $L^n_{2^{n-l}}$. To okrem iného znamená, že podľa vety 1 je každá časť optimálna. Potom mohutnosť minimálneho delenia je

$$\nabla(n, 2^l) = \frac{2^l \times g(n, 2^{n-l})}{2}$$

Po upravení rekurencie dostávame

$$\nabla(n, 2^l) = \frac{2^l \times (l \times 2^{n-l})}{2} = \frac{l}{2}2^n$$

□

Už sa nám podarilo určiť presnú hodnotu minimálneho delenia pre istú podmnožinu $k = 2^l$. Rovnako poznáme optimálny podgraf alebo komponent hyperkocky. Preto teraz nebude problém určiť rozumné dolné ohraničenie minimálneho delenia.

Veta 3 (Dolné ohraničenie) *Pre všetky n a k platí.*

$$\nabla(n,k) \geq \frac{k}{2} \times \min\left\{ g\left(n, \left\lfloor \frac{2^n}{k} \right\rfloor\right), g\left(n, \left\lceil \frac{2^n}{k} \right\rceil\right) \right\}$$

Dôkaz: Veta vyplýva z lemy 1 a vety 1. Pri delení hyperkocky $Q(n)$ na k častí, vzniknú časti veľkostí $\lfloor \frac{2^n}{k} \rfloor$ a $\lceil \frac{2^n}{k} \rceil$. Ak by sa nám podarilo rozdeliť hyperkocku na optimálne časti, tak toto delenie bude podľa lemy 1 minimálne. □

Pre niektoré m môže nastať situácia keď $g(n,m) > g(n,m+1)$. Kvôli takýmto prípadom je potrebné použiť zložitejší vzťah, ktorý vyberie menšiu hodnotu. Pri väčších hodnotách n je však tento rozdiel v porovnaní s 2^n úplne zanedbateľný. Existuje ešte presnejšie vyjadrenie dolného ohraničenia. Využíva fakt, že vieme koľko komponentov má mohutnosť $\lfloor \frac{2^n}{k} \rfloor$ a koľko nie. Takéto vyjadrenie bolo použité pri algoritme, ktorý počíta dolné ohraničenie pre konkrétne n a k.

$$\begin{aligned} z &= 2^n \bmod k \\ \nabla(n,k) &\geq (k-z) \times \frac{1}{2} g\left(n, \left\lfloor \frac{2^n}{k} \right\rfloor\right) + z \times \frac{1}{2} g\left(n, \left\lceil \frac{2^n}{k} \right\rceil\right) \end{aligned} \tag{2.1}$$

Lema 2 ([Kra02]) *Pre pevne zvolené k označme*

$$f_k(n) = \frac{k \times g\left(n, \lfloor \frac{2^n}{k} \rfloor\right)}{2^{n+1}}$$

Limita $\lim_{n\to\infty} f_k(n)$ existuje.

Dôkaz: Označme si $m = \lfloor \frac{2^n}{k} \rfloor$. L^n_m sa skladá z maximálne $n-1$ podkociek, ktorých rozmery sú navzájom rôzne. Ak je m párne, tak majú všetky tieto podkocky rozmer aspoň 1. Potom vieme vieme hyperkocku rozdeliť tak, aby vznikli dve optimálne $L^{n-1}_{m/2}$. Z čoho dostávame $g(n,m) \geq 2 \times g(n-1, m/2) \geq 2 \times g(n-1, m/2) - (n-1)$.

Pre nepárne m odoberme jeden vrchol, konkrétne ten, ktorý je súčasťou najmenšej podkocky rozmeru 0. Potom môžme celú hyperkocku rozdeliť na polovice tak, aby opäť z L^n_{m-1} vznikli dve menšie $L^{n-1}_{(m-1)/2}$. Hranový rez oddeľujúci L^n_m je dvojnásobkom $|\partial L^{n-1}_{(m-1)/2}|$, s nejakými zmenami kvôli vynechanému vrcholu. Ten však môže ovplyvniť nanajvýš $n-1$ hrán, preto $g(n,m) \geq 2 \times g(n-1,(m-1)/2) - (n-1)$. Takže pre párne aj nepárne $\lfloor \frac{2^n}{k} \rfloor$ platí $g\left(n, \lfloor \frac{2^n}{k} \rfloor\right) \leq 2 \times g\left(n-1, \left\lfloor \frac{2^{n-1}}{k} \right\rfloor\right) - (n-1)$.

Uvažujme teraz dve hyperkocky rozmeru $n-1$ a v každej z nich optimálne časti $L^{n-1}_{\lfloor 2^{n-1}/k \rfloor}$. Ak tieto hyperkocky chceme spojiť do hyperkocky rozmeru n, tak musíme každému vrcholu pridať hranu, ktorá ho bude spájať s vrcholom z opačnej hyperkocky rozmeru $n-1$. Vytvorme tieto hrany tak, aby boli vrcholy dvoch spomínaných optimálnych častí pospájané navzájom. Potom spojenie týchto optimálnych častí s možným pridaním jedného vrcholu, vytvorí časť veľkosti $\lfloor \frac{2^n}{k} \rfloor$, ktorú oddeľuje rez mohutnosti $2 \times g\left(n-1, \left\lfloor \frac{2^{n-1}}{k} \right\rfloor\right) + (n-1)$. Z toho dostávame $g\left(n, \lfloor \frac{2^n}{k} \rfloor\right) \leq 2 \times g\left(n-1, \left\lfloor \frac{2^{n-1}}{k} \right\rfloor\right) + (n-1)$.

Preto

$$f_k(n-1) - \frac{k \times (n-1)}{2^{n+1}} \leq f_k(n) \leq f_k(n-1) + \frac{k \times (n-1)}{2^{n+1}}$$

$$-\frac{k \times (n-1)}{2^{n+1}} \leq f_k(n) - f_k(n-1) \leq \frac{k \times (n-1)}{2^{n+1}}$$

Z rozdielu $f_k(n)$ a $f_k(n-1)$ dostávame $|f_k(n+1) - f_k(n)| \leq \frac{k \times n}{2^{n+2}}$, teda

$$|f_k(n+\delta) - f_k(n)| \leq \sum_{i=n}^{\infty} \frac{k \times i}{2^{i+2}} = \frac{k}{4} \times \sum_{i=n}^{\infty} \frac{i}{2^i}$$

Preto limita $\lim_{n\to\infty} f_k(n)$ existuje. □

Existencia danej limity, znamená, že pre dané k exituje konštanta, označme ju $c(k)$, ktorá spĺňa $c(k) = \lim_{n\to\infty} f_k(n)$. Teda mohutnosť, nazvime to ideálneho delenia, je daná parametrom k. Ideálneho preto, lebo stále sa bavíme o dolnom ohraničení. Už vieme, že pre $k = 2^l$ je to zároveň minimálne delenie, pretože ho vieme zostrojiť. Pre ostatné k je zatiaľ otázne, či minimálne delenie tiež spĺňa takúto vlastnosť.

2.3 Horné ohraničenie

Nájsť horné ohraničenie minimálneho delenia je v tomto prípade o niečo zložitejšie ako určenie dolnej hranice. Je možné určiť triviálne horné odhady minimálneho delenia. Hodnoty takýchto ohraničení sú však vzhľadom na dolné ohraničenie pomerne veľké. Snaha je nájsť horné ohraničenie, ktoré by spolu

s dolným ohraničením presne určilo mohutnosť minimálneho delenia.

Nasledujúce tvrdenia hovoria o takýchto presnejších horných odhadoch. Určujú horné ohraničenia minimálneho delenia hyperkocky pre niektoré k. Sami o sebe nie sú kľúčové, pre nás budú slúžiť ako vodítka v ďalšej práci. Z toho dôvodu a aj preto, že tieto tvrdenia sú prevzaté z [Bez96], sú vety uvedené bez dôkazu. Dôležité je, že dôkazy týchto tvrdení sú konštrukčné, takže môžu slúžiť čiastočne ako inšpirácia pri tvorbe niektorých algoritmov riešiacich túto úlohu.

Všetky vety sú uvedené v nasledovnom zmysle. Pre zvolené k vieme konštruovať delenia hyperkocky také, že ak označíme $f_k(n) = \frac{\nabla \mathscr{K}}{2^n}$ potom vieme určiť hodnotu $c(k)$ pre limitu $\lim_{n\to\infty} f_k(n) = c(k)$.

Veta 4 *Zoberme ľubovoľné $a \geq 1$ a $k = 2^a + 1$, potom*

$$c(k) = \frac{a(k-1)}{2(k-2)}$$

Veta 5 *Všeobecnejší prípad predchádzajúceho tvrdenia. Pre zvolené $a > b \geq 0$ a $k = 2^a + 2^b$)*

$$c(k) = \frac{a2^{a-1} - b2^{b-1}}{2^a - 2^b}$$

Veta 6 *Pre zvolené $a \geq 3$ a $k = 2^a - 1$*

$$c(k) = \frac{a(k+1) - 2}{2k}$$

Veta 7 *Opäť všeobecnejšia verzia predchádzajúcej vety. Pre zvolené $a, b \geq 0$, pre ktoré platí $a - b \geq 2$ a $k = 2^a - 2^b$*

$$c(k) = \frac{a2^{a-1} - b2^{b-1} - 2^b}{2^a - 2^b}$$

Kapitola 3

Priame hľadanie minimálneho delenia

3.1 Myšlienka riešenia

Najjednoduchší prístup k riešeniu problému minimálneho delenia hyperkocky je konštrukčný. Pozostáva z vytvorenia modelu hyperkocky, rozdelenia všetkých vrcholov medzi k komponentov a spočítania hrán, ktoré spájajú vrcholy z rôznych komponentov. Takto vytvoríme jedno riešenie delenia hyperkocky na k častí. To však s najväčšou pravdepodobnosťou nebude minimálne. Preto treba hľadať iné riešenie a porovnať či sa nám podarilo nájsť lepšie riešenie. Problém s hyperkockou je ten, že počet vrcholov rastie exponenciálne s rozmerom hyperkocky. A samozrejme, že počet všetkých rozdelení vrcholov hyperkocky do k množín závisí exponenciálne od počtu vrcholov. To znamená, že v závislosti od rozmeru hyperkocky n rastie počet všetkých riešení delenia hyperkocky veľmi rýchlo. Preto, už pre malé n je skúšať všetky riešenia a vybrať z nich to minimálne takmer nemožné.

Ďalšia možnosť sa naskytuje v použití nejakého efektívnejšieho algoritmu. Dala by sa využiť metóda *greedy*. Tá však nezaručuje v tomto prípade minimálne riešenie a časová a pamäťová náročnosť by bola podobne nezvládnuteľná pre väčšie n. Netreba zabúdať na to, že problém minimálneho k-rezu pre všeobecné grafy je NP-ťažký[GH94]. Rovnako iné techniky efektívnych

algoritmov sa len veľmi ťažko prispôsobujú problému minimálneho delenia hyperkocky.

Z vyššie uvedených dôvodov sa v takýchto prípadoch používajú znáhodnené algoritmy[APMS+99]. Tento prístup nie vždy zaručuje kvalitu nájdeného riešenia, rovnako ako nezaručuje efektivitu algoritmu. Znáhodnené algoritmy môžme rozdeliť do dvoch základných skupín, aproximačné algoritmy a heuristiky. Aproximačné algoritmy majú výhodu, že nám zaručujú istú kvalitu nájdeného riešenia. Ich nevýhoda spočíva v tom, že sú ešte stále časovo pomerne náročné. Heuristiky na druhej strane, nám neponúkajú žiadnu garanciu kvality riešenia. Zato však sú heuristiky jednoducho prispôsobiteľné na rôzne problémy a rádovo efektívnejšie ako aproximačné algoritmy. V mnohých prípadoch sa ukazuje, že pri tejto väčšej efektivite, ich riešenia dosahujú kvalitu riešení aproximačných algoritmov[Hro01]. Pri výbere algoritmu pre problém minimálneho delenia hyperkocky bolo zohľadnené najmä kritérium efektivity a vhodnosti algoritmu. Z týchto dôvodov bolo zvolené simulované žíhanie, spadajúce do množiny heuristických algoritmov.

3.2 Simulované žíhanie

Ako už bolo spomenuté, simulované žíhanie, ďalej len SA[1],je heuristický algoritmus. Pri použití SA nemáme zaručenú kvalitu riešenia, resp. jeho minimalitu, ale aj napriek tomu je často používané. Tento algoritmus je odvodený od prístupu zvaného lokálne hľadanie a bol inšpirovaný fyzikálnym procesom.

3.2.1 Fyzikálne žíhanie

Ilustračný popis fyzikálneho žíhania je nasledovný. Žíhanie je proces, pri ktorom sa postupným zahrievaním a chladením snažíme docieliť napr. lepšiu kryštalickosť látky. Snažíme sa priblížiť optimu. Látku zahrejeme, vtedy majú molekuly lepšie možnosti pohybu. Potom látku postupne chladíme. Molekuly sa ustaľujú a tým sa tvoria kryštály. Keď je chladenie dostatočne pomalé, je

[1] z anglického simulated annealing

vznik kryštálov pravdepodobnejší. Preto sa pri správnom žíhaní vytvorí viac kryštálov.

3.2.2 Lokálne hľadanie

Lokálne hľadanie je celkom intuitívny prístup k riešeniu optimalizačných problémov. Spočíva v nájdení nejakého riešenia a jeho zlepšovaním. Dôležité je, že musíme pre množinu všetkých platných riešení zadefinovať susednosť riešení. Najlepšie je definovať susednosť ako čo najmenšiu zmenu platného riešenia, ktorá vytvorí iné platné riešenie. Lokálne hľadanie na začiatku vyberie jedno platné riešenie. Potom prehľadá susedné riešenia tohto riešenia, vyberie z nich to najlepšie a presunie sa na toto riešenie. Samozrejme, že ak je nájdené riešenie horšie ako aktuálne riešenie, tak algoritmus skončí. Takýmto spôsobom je lokálne hľadanie schopné nájsť lokálne optimá. Problém je, že lokálne hľadanie len ťažko nájde globálne optimum.

3.2.3 Simulované žíhanie

Simulované žíhanie je teda vylepšením lokálneho hľadania a to práve v tom, že mu dáva možnosť presunúť sa z lokálneho optima do iného lokálneho optima. Tým sa zvýši pravdepodobnosť nájdenia globálneho optima resp. kvalitnejšieho optima. Toto vylepšenie je zabezpečené zavedením teploty. Teplota bude parameter SA, ktorý sa postupne počas behu programu zmenšuje. Zároveň bude teplota umožňovať lepší pohyb. V našom prípade je to pohyb medzi jednotlivými riešeniami a nie pohyb častíc. Pri SA budeme na rozdiel od lokálneho hľadania umožňovať aj prechod na nevýhodnejšie platné riešenie. Práve teplota spolu s mierou nevhodnosti riešenia budú určovať pravdepodobnosť prechodu na nevýhodné riešenie. Simulované žíhanie je dobre známy a často uvádzaný príklad heuristiky, bližší popis sa dá nájsť napríklad v [Hro01][APMS+99]. Teraz sa zameriame na konkrétne prispôsobenie SA na problém minimálneho delenia hyperkocky.

3.3 Popis použitého algoritmu

3.3.1 Susednosť platných riešení

Ako prvé si treba pri SA rovnako ako pri lokálnom hľadaní zadefinovať platné riešenie a susednosť riešení. V tejto práci som použil dve alternatívy. Platné riešenie je množina 2^n vrcholov, z ktorých má každý priradený jeden z k komponentov, do ktorého patrí. Zároveň je $2^n \bmod k$ komponentov, do ktorých patrí $\lceil \frac{2^n}{k} \rceil$ vrcholov a $k - (2^n \bmod k$ komponentov s $\lfloor \frac{2^n}{k} \rfloor$ vrcholmi.

Prvá použitá verzia susednosti je taká, že dve riešenia sú susedné ak z jedného riešenia vieme dostať druhé zámenou dvoch vrcholov z rôznych komponentov. V tomto prípade je množina susedov jedného riešenia príliš veľká. Z toho dôvodu nie je možné hľadať najvýhodnejšieho suseda z okolia platného riešenia. Tým, že upravíme pravdepodobnosť prejdenia na nevýhodnejšie riešenie a zväčšíme počet iterácií, vieme tento nedostatok kompenzovať.

Druhá definícia susednosti bola motivovaná zmenšením okolia susedov platného riešenia. Nevýhodou je, že povoľuje aj neplatné riešenia. Konkrétne dve riešenia sú susedné ak z jedného riešenia vieme dostať druhé riešenie presunutím jedného vrcholu do iného komponentu. Aby sme mohli presunúť vrchol do iného komponentu, zvolíme konštantu, o ktorú sa budú môcť líšiť počty vrcholov prislúchajúcich ku komponentu. Takto budeme prehľadávať riešenia, ktoré sa môžu minimálne líšiť od platných riešení problému. Pre potreby porovnania kvality nájdeného riešenia s predchádzajúcim prístupom bolo nutné po nejakom čase aktuálne riešenie upraviť tak, aby spĺňalo podmienky platnosti. Zvolený *greedy* prístup vyberie najvýhodnejšie vrcholy na presun z komponentu resp. do komponentu, ktorý má nesprávny počet vrcholov. Napriek tomu sa kvalita riešenia vždy o niečo zhoršila.

V tomto prípade susednosti som použil aj vyberanie spomedzi viacerých vrcholov. Podľa predom nastaveného parametru programu sa vyberie nejaký počet vrcholov. Každému vrcholu sa vyberie jemu najvhodnejší nový komponent. Potom sa z vrcholov vyberie ten najvýhodnejší.

3.3.2 Počiatočné riešenie

Prvé platné riešenie môže ovplyvniť výsledok SA. Na druhej strane, ak je počiatočná teplota dosť veľká, tak umožňuje veľký pohyb medzi riešeniami, ktorý po čase vytvorí náhodné riešenie. Odskúšané boli tri možné prístupy k vytvoreniu počiatočného riešenia.

Najjednoduchšie je postupne priradiť prvých $\lceil \frac{2^n}{k} \rceil$ vrcholov prvému komponentu, druhých $\lceil \frac{2^n}{k} \rceil$ druhému, ... Takto vznikne riešenie, ktoré je kvalitnejšie ako náhodne generované riešenie. Pre $k = 2^l$ dokonca takto vytvorené riešenie je aj optimálnym riešením. Vo väčšine prípadov sa však ukázalo, že takéto riešenie je blízko lokálneho optima, z ktorého sa algoritmus už nemusí vedieť dostať.

Ďalší možný prístup je vytvoriť náhodné platné riešenie. Posledný variant voľby počiatočného riešenia je popísaná v časti 4.2.3.

3.3.3 Teplota a zmena teploty

Voľba počiatočnej teploty závisela od rozmeru hyperkocky. Pri väčšom n je možné zhoršenie riešenia väčšie ako pri menšom rozmere hyperkocky. Vzorec, ktorý určuje pravdepodobnosť prechodu na nevýhodnejšie riešenie je štandardný $P = \exp(-\frac{\Delta|\partial\mathscr{K}|}{T})$. Zmenšenie teploty bolo rovnako štandardné $T_i = T_0 \times dec^i$, kde parameter dec bol volený rôzne medzi 0.99999 a 0.99999999. SA spočíva v zohriati, následnom chladení a opätovnom zohriati. Program mal dopredu určenú sadu teplôt. Začal na teplote T^0, keď sa ochladila a riešenia sa prestali zlepšovať prešiel na teplotu T^1. Takto sa riešenie niekoľko krát zohrialo a následne postupne chladilo.

3.3.4 Efektívnosť riešenia

Pre menšie rozmery hyperkocky dáva simulované žíhanie dobré výsledky[2]. Zväčšovaním rozmerov nastávajú dva základné problémy.

Prvý problém je triviálny a to výpočtová zložitosť ako taká. Pri rozmeroch $n > 25$ je aj zrátanie mohutnosti konkrétneho rezu časovo náročné a priame

[2] viac v kapitole 5

riešenie má aj vysoké nároky na pamäť.

Druhý problém je spôsobený vzdialenosťou dvoch susedných riešení. Susedné riešenia sú veľmi blízko čo sa týka kvality riešenia. Výmenou dvoch vrcholov resp. premiestnením jedného vrchola spôsobíme veľmi malú zmenu v kvalite riešenia. Počiatočné riešenie či už je volené náhodne alebo postupným priradením je veľmi vzdialené od minimálneho riešenia. Jedným prechodom na susedné hoci aj lepšie riešenie sa kvalita mení minimálne. Z tohto dôvodu je vhodný malý parameter zmeny teploty *dec*. To však nič nemení na tom, že riešenie konverguje v neskoršej fáze chladenia len veľmi pomaly.

Kapitola 4

Maximálne uloženie

4.1 Myšlienka riešenia

Ako už nadpis kapitoly napovedá, v tomto prípade sa budeme snažiť nájsť minimálne delenie pomocou maximálneho uloženia. Pomôžeme si zadaním problému 2. Ten hovorí o minimálnom uložení k množín veľkosti $\left\lfloor \frac{2^n}{k} \right\rfloor$ do hyperkocky $Q(n)$. Z vety 1 a lemy 1 vieme, že ak by sa nám podarilo uložiť k množín typu L^n_m, $m = \left\lfloor \frac{2^n}{k} \right\rfloor$ do hyperkocky, tak dosiahneme minimálne uloženie. Vhodným pridaním a preusporiadaním maximálne $k-1$ vrcholov by sme dosiahli riešenie, ktoré by bolo veľmi blízke minimálnemu ak nie minimálne. Z tejto idei sa bude odvíjať riešenie problému minimálneho delenia hyperkocky, ktoré je popísané v tejto kapitole.

Opäť sa naskytuje viacero možných prístupov ukladania L^n_m do hyperkocky. Jedna možnosť je ukladať celé takéto bloky. Tu však nastane problém ako ich tam ukladať tak, aby sa ich zmestilo čo najviac. Keďže nie je zaručené pre hoci aké k, že minimálne delenie existuje, nie je isté či sme schopní uložiť všetkých k L^n_m podgrafov. Čo potom spraviť s posledným? Alebo s predposledným?

Prístup zvolený v tejto práci ukladá tieto podgrafy po častiach. Podgraf L^n_m je zložený z viacerých menších hyperkociek – podkociek. Tie sú medzi sebou spojené tak, aby čo najmenej hrán išlo mimo tohto podgrafu. Okrem toho rozmery týchto podkociek tvoria klesajúcu postupnosť, teda sú všetky

rôzne. Algoritmus sa najprv pokúsi uložiť najväčšie podkocky všetkých komponentov. Potom druhé najväčšie podkocky všetkých komponentov.

Ukladanie podkociek do hyperkocky tiež nie je jednoduché. Prvé čo treba zaručiť je, aby sa žiadne neprekrývali. Druhá rovnako dôležitá a zložitá podmienka je dosiahnuť to, aby menšie podkocky boli "vedľa" väčších podkociek rovnakého komponentu. Chceme aby nakoniec tvorili útvar, ktorý je automorfizmom hyperkocky transformovateľný na L_m^n.

Kvôli týmto komplikáciám sa možnosť nejakého efektívnejšieho algoritmu, ktorý by rozumne ukladal jednotlivé časti, stáva málo pravdepodobnou. Výhoda je však v tom, že keď sa nám podarí uložiť prvých x najväčších podkociek jednotlivých komponentov, tak väčšina vrcholov je uložená ideálne. Treba už len preusporiadať malú časť vrcholov.

4.2 Popis použitého algoritmu

4.2.1 Ukladanie podkociek

Aj napriek tomu, že ukladáme podkocky do hyperkocky, môžme rozlišovať orientáciu a následne polohu uloženia. Podkocku veľkosti r, teda r rozmernú hyperkocku, vieme vrámci celej hyperkocky zadefinovať ako množinu všetkých vrcholov, ktoré majú istých $n-r$ bitov svojho označenia pevne daných. Zvyšných r bitov je ľubovoľných. Potom, to ktoré bity sú pevné nám udáva orientáciu podkocky. Konkrétne hodnoty týchto pevných bitov nám udávajú polohu.

Na ukladanie podkociek sa ťažko hľadá nejaký efektívny spôsob. Pokiaľ nechceme s každou podkockou kontrolovať aj polohy všetkých ostatných a zohľadňovať všetky nasledujúce podkocky, môžme len ťažko vedieť kam je dobré podkocku umiestniť. Preto algoritmus ukladá podkocky náhodne a následne skontroluje vhodnosť polohy. V čom sa ešte dá urobiť rozdiel je, či chceme najväčších k podkociek uložiť všetky rovnako orientované a len zmeniť polohu alebo hľadať pre ne aj orientáciu. Keď ich uložíme rovnako orientované, tak môžme prísť o niektoré riešenia, o ktorých nevieme či sú naozaj nevýhodnejšie. Naopak máme isté, že aspoň podkocky najväčšieho

rozmeru sa niekam uložili. V tomto algoritme používame náhodné priradenie orientácie aj polohy.

4.2.2 Kontrola pozície podkocky

Vyššie boli uvedené dve základné podmienky správneho uloženia podkocky, ktoré budeme dodržiavať. Podkocka nemôže prekrývať inú podkocku a podkocka musí správne nadväzovať na svoj komponent tak, aby bolo možné vytvoriť optimálnu časť. Preto si musíme pamätať pozíciu každej uloženej podkocky. Pozíciu podkocky si zapamätám vo forme pevných bitov, ktoré určujú jej orientáciu a polohu.

Každá podkocka je reprezentovaná dvomi 32 bitovými číslami, na ktorých robím bitové operácie. Okrem iného to znamená, že program pracuje len s hyperkockami, ktoré majú maximálne $n = 32$. Tieto dve čísla si označme ako *maska* a *bity*. *Maska* mi hovorí, ktoré bity čísla *bity* sú platné. To znamená, že bity ktoré sú v *maske* jednotkové určujú orientáciu podkocky, a teda ich je presne $n - r$ keď r je rozmer podkocky. Bity v *bity*, ktoré sú na rovnakých miestach ako sú jednotkové bity *masky* mi určujú polohu podkocky.

Pre dve podkocky z rôznych komponentov musí platiť nasledovné[1]

$$
\begin{aligned}
k_1 &\neq k_2 \\
0 &\neq maska_{k_1,p_1} \wedge maska_{k_2,p_2} && (4.1)\\
0 &\neq ((maska_{k_1,p_1} \wedge maska_{k_2,p_2}) \wedge bity_{k_1,p_1}) \oplus \\
&\quad\ ((maska_{k_1,p_1} \wedge maska_{k_2,p_2}) \wedge bity_{k_2,p_2}) && (4.2)
\end{aligned}
$$

Výraz (4.1) znamená, že dve podkocky musia mať aspoň jeden bit *masky* rovnaký. Inak by mali kocky určite nejaký prekryv. Výraz (4.2) následne porovná tie bity podkocky, na ktorých sa obe *masky* zhodujú. Ak sa xor týchto bitov rovná nule, znamená to, že majú tieto bity rovnaké a tým pádom sa podkocky prekrývajú.

Keď ukladám podkocky jedného komponentu od prvej najväčšej po poslednú, tak prvá voči ostatným nemusí spĺňať žiadne podmienky. Nasledujúce

[1] v rovniciach sú použité bitové operátory $\wedge$ and, $\oplus$ xor

podkocky musia voči predchádzajúcim spĺňať nasledovné.

$$\begin{aligned} p_1, p &\geq 2 \\ maska_{k,p_2} \wedge maska_{k,p_1} &= maska_{k,p_2}\ ,\ p_2 < p_1 && (4.3) \\ maska_{k,p-2} \wedge bity_{k,p} &= maska_{k,p-2} \wedge bity_{k,p-1}\ ,\ p \geq 3 && (4.4) \\ 2^x &= |maska_{k,p-1} \wedge bity_{k,p} - \\ &\quad maska_{k,p-1} \wedge bity_{k,p-1}| && (4.5) \end{aligned}$$

Prvý vzťah vyjadruje skutočnosť, že *maska* väčšej podkocky je podmnožinou *masky*. Alebo naopak, že *maska* menšej podkocky vznikne pridaním jednotkových bitov do *masky* väčšej podkocky. Posledný výraz (4.5) znamená, že podkocka p má na miestach platných bitov podkocky $p-1$ všetky bity až na jeden rovnaké ako podkocka $p-1$. Vzťah (4.4) potom bližšie špecifikuje, že bit v ktorom sa p a $p-1$ nezhodujú nie je jeden z tých ktoré má podkocka $p-2$ určené.

Tieto *bitové* a *maskové* podobnosti podkociek jedného komponentu sú práve na to, aby boli podkocky vedľa seba. Okrem toho musia byť vedľa seba tak, aby sa z nich prípadnou transformáciou dala vytvoriť L_m^n.

4.2.3 Implementácia a efektívnosť algoritmu

Bolo spomenuté, že algoritmus hľadá nové pozície podkociek náhodne s prihliadnutím na potrebné podmienky. Prvé ukladané podkocky majú veľmi veľa možností, kam môžu byť uložené. Zároveň dokážu výrazne obmedziť priestor pre nasledujúce podkocky, bolo použitie *backtrackingu* nevýhodné. Ak by bola prvá ukladaná časť uložená nevhodne, trvalo by strašne dlho, kým by sa *backtrack* k takejto časti vrátil. Zle uložená podkocka by obmedzovala vznik kvalitných riešení. Preto som použil inú metódu.

Každá kocka má nejaký počet pokusov, počas ktorých sa pre ňu musí nájsť umiestnenie. Počet pokusov závisí od počtu možností jej položenia. Ak sa nepodarí podkocku uložiť, nepokračuje sa zmenou polohy predchádzajúcej podkocky. Namiesto toho sa začnú ukladať podkocky odznova. Týmto sa odskúša menší počet možností ako pri *backtrackingu*, ale na druhej strane

sú tieto možnosti oveľa variabilnejšie ako možnosti, ktoré stihne odskúšať *backtrack*.

Efektívnosť algoritmu výrazne závisí od počtu komponentov k. Pre malé k má algoritmus veľmi dobré výsledky. Keď je počet komponentov veľký, tak vznikne situácia keď prvé najväčšie časti sú voči hyperkocke malé, preto sa výrazne rozšíri počet ich možných uložení. Okrem toho samotný algoritmus je iba v niektorých prípadoch schopný nájsť uloženie pre všetky podkocky všetkých komponentov.

Väčšina výsledkov tohto algoritmu je zo spomínaných dôvodov dobrý základný kameň pre presnejšie metódy – simulované žíhanie. Z výsledkov tejto metódy vytváram *greedy* princípom počiatočné riešenia pre simulované žíhanie, ktoré ich vylepšuje. Uložené podkocky tvoria množiny vrcholov, ktoré majú určený komponent. Zvyšným vrcholom algoritmus priraďuje komponenty podľa toho kde majú najviac vrcholov, s ktorými sú spojené. Čím viac podkociek sa podarí uložiť tým je *greedy* presnejšie a výhodnejšie a tým menej treba preusporiadavať pomocou SA. Ďalšie výsledky sú popísané v nasledujúcej kapitole 5.

Tento algoritmus by sa hodilo vylepšiť práve simulovaným žíhaním. Problém je s definíciou susednosti dvoch riešení rovnako ako prechod medzi nimi. Okrem toho sa až príliš neoplatí vyberať už uložený väčší komponent. Preusporiadavanie jednotlivých riešení tiež neprichádza do úvahy, keďže je väčšinou problém uložiť menší komponent a nie to ešte hýbať väčším.

Kapitola 5

Výsledky

Keď už vieme v čom vzpočívajú jednotlivé prístupy riešenia problému minimálneho delenia môžme pristúpiť k ich porovnaniu. V kapitolách 3 a 4 boli spomenuté aj niektoré výhody a nevýhody jednolivých algoritmov. Avšak ani jeden z algoritmov nemá garantovanú nejakú kvalitu vyprodukovaného riešenia. Preto je zaujímavé porovnanie jednotlivých algoritmov medzi sebou ako aj porovnanie algoritmov voči želanému dolnému odhadu.

5.1 Voľba parametrov

Prvý základný parameter pri testovaní kvality riešení je veľkosť hyperkocky v podaní jej rozmeru n. Ten druhý je samozrejme k, počet častí na ktoré budeme hyperkocku $Q(n)$ deliť. Tieto dve hodnoty som volil v rozumných medziach. Nemá zmysel deliť kocku s 512 vrcholmi na 200 častí. Rovnako pre všetky $k = \overline{2, 10}$ je dokázaný horný odhad, ktorý v tomto prípade znamená aj minimálne delenie [Bez96]. Rozhodol som sa použiť štyri hodnoty $k = 5, 7, 16, 24$, pre ktoré vieme, že existuje minimálne delenie na úrovni dolného ohraničenia. Pri týchto k vieme porovnať ako sa algoritmy priblížia k hodnote minimálneho delenia, ktoré sa skutočne dá skonštruovať. Ďalšia štvorica k je $k = 11, 13, 19, 21$. Pri týchto k nevieme určiť horné ohraničenie za pomoci žiadneho zo vzorcov v časti 2.3. Práve v tomto prípade sa budeme snažiť aproximovať horné ohraničenie epxerimentálne. Pri všetkých k bolo použité

$n = \overline{10, 20}$.

Ďalšie nastavenia sa týkajú konkrétne simulovaného žíhania. Ako parameter klesania teploty *dec* bola pri každom použití SA zvolená hodnota $dec = 0.9999999$. Rovnako bola pri SA použitá vždy tá istá sekvencia teplôt: $2.0, 1.6, 5.5, 1.6, 1.2, 4.0, 1.5, 1.0, 2.5, 1.6, 1.2, 3.5, 1.5, 1.1$. Pri "zohriatí riešenia" na teplotu okolo 2.5 a viac sa riešenia nejaký čas menia skoro úplne náhodne. Z tohto dôvodu nie je nutné púšťať algoritmus SA pre pevne zvolené k a n viac krát. Sekvencia teplôt s niektorými teplotami prehnane vysokými toto zabezpečí.

Algoritmus pracujúci na princípe ukladania podkociek do hyperkocky má konštantu, ktorá určuje koľko krát sa snaží uložiť jednu podkocku. Ak sa podkocku nepodarí na toľko pokusov uložiť, začína sa ukladanie odznova. Táto konštanta bola po celý čas nastavená na hodnotu 100. Zaujímavé je, že pri testovaní rôznych variánt, ktoré sa snažili nahradiť túto konštantu sofistikovanjším ohodocovaním, sa nepodarilo nájsť lepšie riešenie. Z toho dôvodu je táto konštanta ponechaná aj do finálnej verzie algoritmu. Okrem toho bol beh algoritmu obmedzený časovo.

Vzhľadom na to že aj tento druhý algoritmus je znáhodnený takým spôsobom, že sa hľadá riešenie vždy úplne od začiatku, nemá význam skúšať spúšťať tento algoritmus viac krát. Dve spustenia sú ekvivalentné jednému dlhšiemu spusteniu.

5.2 Porovnávané algoritmy

V nasledujúcich tabulkách sú porovnávané tieto hodnoty. Jednak pre k, pre ktoré je teoreticky dokázané presné horné ohraničenie je táto hodnota uvedená. Ďalej je vždy uvedená hodnota dolnej hranice, vypočítaná podľa vzťahu (2.1), označíme ju symbolom $\nabla(n, k)\nabla$.

Prvý prístup je spomínané simulované žíhanie, ktoré má susednosť riešení definovanú pomocou zámeny dvoch vrcholov. Ako druhé je rovnako použité simulované žíhanie, tentoraz s druhou spomínanou variantou susednosti riešení. Uvedené hodnoty boli získané po tom ako bolo konkrétne riešenie preusporiadné, tak aby spĺňalo podmienku platnosti riešenia. Viac v časti 3.3.1.

Nasleduje algoritmus postupného ukladania podkociek jednotlivých komponentov, označme ho "Pack". Aby sa dal porovnať s ostatnými boli neuložené vrcholy priradené k voľným komponentov jednoduchou *greedy* metódou. Posledné sú uvedené výsledky krátkeho simulovaného žíhania, ktoré má ako počiatočné riešenie výstup z algoritmu postupného ukladania podkociek, označenie "SA Pack".

5.3 Výsledky testov

V tabulkách sú uvedené jednak hodnoty $|\partial\mathscr{K}|$, ale kvôli lepšiemu porovnaniu sú uvedené aj asymptotické hodnoty $\frac{|\partial\mathscr{K}|}{2^n}$.

V drvivej väčšine prípadov sa ukázal najvýhodnejší prístup SA Pack. To je prístup ukladania podkociek komponentov do hyperkocky s neskorším použitím simulovaného žíhania. Dokonca dosť často už nie sme schopní vylepšiť riešenie žíhaním.

Hodnoty delení sa pri menších k pobybujú naozaj v okolí dolnej hranice minimálneho rezu. Napríklad pre $k = 11$ sme sa asymptoticky dostali takmer na úroveň minimálneho delenia. Tento výsledok naznačuje, že aj pre tie k, pre ktoré nie je dokázané horné ohraničenie teoreticky, existujú riešenia, ktoré sa od optimálneho riešenia líšia minimálne.

Nasledujú štyri tabulky pre hodnoty k, pre ktoré vieme určiť horné ohraničenie. Po nich sú tabulky pre hodnoty k, ktorých hornú hranicu minimálneho delenia nepoznáme.

k	n	c(k)	$\nabla(n,k)\nabla$	SA 1	SA 2	Pack	SA Pack
5	10	1.333	1.330 1362	1.357 1390	1.400 1434	1.343 1376	1.332 1364
	11		1.332 2728	1.357 2780	1.390 2848	1.334 2733	1.332 2729
	12		1.332 5458	1.360 5573	1.433 5870	1.333 5461	1.333 5461
	13		1.333 10920	1.361 11153	1.599 13103	1.333 10923	1.333 10922
	14		1.333 21842	1.394 22846	1.599 26212	1.334 21857	1.333 21847
	15		1.333 43688	1.599 52425	1.599 52425	1.333 43689	1.333 43689
	16		1.333 87378	1.599 104854	1.599 104854	1.333 87389	1.333 87388
	17		1.333 174760	1.599 209711	1.599 209711	1.333 174781	1.333 174781
	18		1.333 349522	1.599 419428	1.599 419428	1.334 349752	1.334 349752
	19		1.333 699048	1.599 838857	1.599 838857	1.447 759108	1.447 759089
	20		1.333 1398098	1.6 1677718	1.6 1677718	1.333 1398307	1.333 1398307

Tabuľka 5.1: výsledky testov pre $k = 3 = 2^2 + 1$

k	n	c(k)	$\nabla(n,k)\nabla$	SA 1	SA 2	Pack	SA Pack
7	10	1.571	1.568	1.580	1.639	1.573	1.570
			1606	1618	1679	1611	1608
	11		1.569	1.579	1.641	1.574	1.571
			3214	3234	3361	3224	3218
	12		1.570	1.580	1.642	1.573	1.572
			6432	6472	6729	6443	6439
	13		1.571	1.626	1.642	1.571	1.571
			12870	13325	13455	12872	12872
	14		1.571	1.646	1.642	1.573	1.573
			25742	26984	26913	25776	25776
	15		1.571	1.688	1.657	1.573	1.573
			51488	55327	54327	51576	51576
	16		1.571	1.845	1.803	1.576	1.576
			102982	120975	118195	103308	103308
	17		1.571	1.857	1.815	1.581	1.581
			205966	243412	237937	207314	207313
	18		1.571	1.857	1.837	1.579	1.579
			411936	486840	481600	413997	413996
	19		1.571	1.857	1.855	1.590	1.590
			823878	973677	972727	833952	833952
	20		1.571	1.857	1.857	1.589	1.589
			1647758	1947348	1947348	1666777	1666777

Tabuľka 5.2: výsledky testov pre $k = 7 = 2^3 - 1$

k	n	c(k)	$\nabla(n,k)\nabla$	SA 1	SA 2	Pack	SA Pack
16	10	2	2	2.031	2	2.031	2.031
			2048	2080	2048	2080	2080
	11		2	2.062	2	2.031	2.031
			4096	4224	4096	4160	4160
	12		2	2.259	2.031	2	2
			8192	9254	8320	8192	8192
	13		2	2.211	2.062	2.031	2.031
			16384	18119	16896	16640	16640
	14		2	2.296	2.269	2.031	2.031
			32768	37633	37184	33280	33280
	15		2	2.628	2.275	2.031	2.031
			65536	86143	74560	66560	66560
	16		2	2.478	2.314	2.031	2.031
			131072	162421	151680	133120	133120
	17		2	2.95	2.351	2.085	2.085
			262144	386663	308224	273408	273408
	18		2	–[1]	2.520	2.054	2.054
			524288	–	660792	538624	538624
	19		2	–	2.671	2.093	2.093
			1048576	–	1400624	1097728	1097728
	20		2	–	2.680	2.156	2.156
			2097152	–	2810192	2260992	2260992

Tabuľka 5.3: výsledky testov pre $k = 16 = 2^4$

k	n	c(k)	$\nabla(n,k)\nabla$	SA 1	SA 2	Pack	SA Pack
24	10	2.5	2.484	2.499	2.492	2.550	2.503
			2544	2559	2552	2612	2564
	11		2.492	2.555	2.496	2.547	2.519
			5104	5234	5112	5217	5159
	12		2.496	2.556	2.498	2.569	2.540
			10224	10471	10232	10524	10406
	13		2.498	2.652	2.654	2.558	2.547
			20464	21729	21747	20960	20873
	14		2.499	2.668	2.666	2.587	2.568
			40944	43723	43680	42397	42077
	15		2.499	2.669	2.666	2.632	2.561
			81904	87478	87368	86271	83949
	16		2.499	2.668	2.666	2.693	2.675
			163824	174892	174752	176496	175316
	17		2.499	2.667	2.666	2.781	2.781
			327664	349681	349512	364592	364592
	18		2.499	2.667	2.666	2.824	2.824
			655344	699238	699040	740318	740318
	19		2.499	2.667	2.666	2.884	2.884
			1310704	1398330	1398088	1512232	1512232
	20		2.499	2.666	2.666	2.900	2.900
			2621424	2796459	2796192	3041722	3041719

Tabuľka 5.4: výsledky testov pre $k = 24 = 2^4 + 2^3$

k	n	$\nabla(n,k)\nabla$	SA 1	SA 2	Pack	SA Pack
11	10	1.896	1.933	1.942	1.999	1.931
		1942	1980	1989	2047	1978
	11	1.901	1.940	1.942	1.958	1.933
		3894	3974	3978	4011	3960
	12	1.901	1.951	1.944	2	1.983
		7790	7993	7965	8192	8124
	13	1.902	1.957	2.105	1.987	1.970
		15584	16036	17245	16279	16146
	14	1.902	2.121	2.102	2.012	2.000
		31176	34753	34451	32972	32781
	15	1.903	2.021	2.114	1.996	1.988
		62358	66235	69280	65412	65175
	16	1.903	2.174	2.134	1.966	1.965
		124726	142489	139854	128898	128811
	17	1.903	2.181	2.176	1.980	1.980
		249454	285960	285295	259581	259579
	18	1.903	2.181	2.181	1.987	1.987
		498912	571936	571936	520902	520902
	19	1.903	2.181	2.181	2.028	2.028
		997832	1143892	1143892	1063484	1063484
	20	1.903	2.181	2.181	2.006	2.006
		1995670	2287793	2287793	2103476	2103476

Tabuľka 5.5: výsledky testov pre $k = 11$

k	n	$\nabla(n,k)\nabla$	SA 1	SA 2	Pack	SA Pack
13	10	2.023	2.044	2.059	2.074	2.051
		2072	2094	2109	2124	2101
	11	2.028	2.054	2.049	2.077	2.056
		4154	4208	4197	4255	4211
	12	2.029	2.100	2.079	2.085	2.061
		8314	8605	8519	8542	8443
	13	2.031	2.141	2.110	2.087	2.067
		16640	17540	17293	17098	16935
	14	2.031	2.260	2.162	2.081	2.064
		33282	37037	35429	34102	33831
	15	2.031	2.307	2.197	2.073	2.067
		66568	75596	71994	67946	67762
	16	2.031	2.289	2.211	2.092	2.092
		133144	150048	144933	137120	137120
	17	2.031	2.307	2.213	2.100	2.100
		266298	302464	290076	275335	275335
	18	2.031	2.307	2.270	2.100	2.100
		532602	604940	595083	550644	550644
	19	2.031	2.307	2.286	2.157	2.157
		1065216	1209882	1198973	1131308	1131308
	20	2.031	2.307	2.304	2.148	2.148
		2130434	2419770	2416288	2253125	2253125

Tabuľka 5.6: výsledky testov pre $k = 13$

k	n	$\nabla(n,k)\nabla$	SA 1	SA 2	Pack	SA Pack
19	10	2.279	2.311	2.324	2.388	2.321
		2334	2367	2380	2446	2377
	11	2.280	2.342	2.316	2.384	2.328
		4670	4797	4745	4884	4768
	12	2.281	2.384	2.354	2.415	2.357
		9344	9768	9644	9892	9657
	13	2.282	2.507	2.369	2.420	2.357
		18696	20539	19409	19826	19311
	14	2.283	2.586	2.388	2.434	2.402
		37408	42375	39126	39885	39366
	15	2.283	2.591	2.455	2.411	2.393
		74822	84920	80464	79011	78446
	16	2.283	2.562	2.464	2.431	2.419
		149656	167921	161522	159323	158572
	17	2.283	2.591	2.582	2.420	2.420
		299326	339733	338558	317242	317242
	18	2.283	2.592	2.584	2.450	2.450
		598662	679488	677589	642276	642276
	19	2.283	2.592	2.586	2.431	2.431
		1197342	1358981	1355864	1274928	1274928
	20	2.283	2.592	2.589	2.618	2.618
		2394686	2717976	2715537	2746086	2746086

Tabuľka 5.7: výsledky testov pre $k = 19$

k	n	$\nabla(n,k)\nabla$	SA 1	SA 2	Pack	SA Pack
21	10	2.335	2.360	2.361	2.411	2.374
		2392	2417	2418	2469	2431
	11	2.343	2.368	2.368	2.400	2.392
		4800	4850	4850	4916	4899
	12	2.346	2.458	2.375	2.434	2.410
		9610	10072	9729	9972	9873
	13	2.348	2.610	2.407	2.435	2.424
		19240	21384	19724	19953	19865
	14	2.348	2.640	2.523	2.451	2.426
		38482	43268	41344	40169	39756
	15	2.348	2.653	2.547	2.435	2.429
		76968	86942	83469	79818	79616
	16	2.349	2.590	2.555	2.575	2.549
		153944	169772	167482	168797	167096
	17	2.349	2.654	2.562	2.576	2.576
		307904	347870	335817	337660	337660
	18	2.349	2.654	2.565	2.744	2.744
		615818	695930	695930	719528	719528
	19	2.349	2.654	2.617	2.750	2.750
		1231656	1391859	1372460	1442148	1442148
	20	2.349	2.654	2.641	2.712	2.712
		2463314	2783708	2769369	2844201	2844201

Tabuľka 5.8: výsledky testov pre $k = 21$

Kapitola 6

Záver

V tejto práci sme sa venovali hľadaniu minimálneho delenia n rozmernej hyperkocky na k rovnako veľkých častí. Ukázali sme teoretické poznatky o tomto probléme, ktoré boli dokázané v [Bez96]. Tieto poznatky bližšie určujú hodnoty minimálneho delenia zavedením dolného ohraničenia. Okrem toho zavádzajú horné ohraničenie minimálneho delenia pre niektoré hodnoty k.

Mojou snahou bolo potvrdiť tieto výsledky experimentálne - algoritmicky. Následne potom môžme hľadať hodnoty minimálneho delenia pre k, pre ktoré nepoznáme horné ohraničenie. V tejto práci som prezentoval dva rozdielne prístupy k riešeniu daného problému. Oba prístupy sú znáhodnené algoritmy bez záruky kvality nájdeného riešenia. Napriek tomu sa pri testovaní a porovnaní s dolným ohraničením ukázalo, že riešenia produkované týmito algoritmami sa pohybujú veľmi blízko optimálneho riešenia. Pre hodnoty k a n, s ktorými sme schopní narábať v reálnom čase boli algoritmy viac menej úspešné. Najlepší algoritmus na hľadanie minimálneho delenia bol nakoniec vhodnou kombináciou oboch uvedených prístupov.

Literatúra

[APMS+99] Giorgio Ausiello, M. Protasi, A. Marchetti-Spaccamela, G. Gambosi, P. Crescenzi, and V. Kann. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[Bez96] Sergei L. Bezrukov. On k-partitioning the n-cube. In *WG*, pages 44–55, 1996.

[BF88] A. H. Bond and D. Fashena. Parallel vision techniques on the hypercube computer. In *Proceedings of the third conference on Hypercube concurrent computers and applications*, pages 1007–1010, New York, NY, USA, 1988. ACM Press.

[Din88] H-Q. Ding. Polymer simulation on the hypercube. In *Proceedings of the third conference on Hypercube concurrent computers and applications*, pages 1044–1050, New York, NY, USA, 1988. ACM Press.

[GH94] Olivier Goldschmidt and Dorit S. Hochbaum. A polynomial algorithm for the k-cut problem for fixed k. *Math. Oper. Res.*, 19(1):24–37, 1994.

[Har64] L.H. Harper. Optimal assignment of numbers to vertices. *Journal of the Society for Industrial Applied Mathematics*, 12(1):131–135, 1964.

[Hro01] Juraj Hromkovic. *Algorithmics for Hard Problems*. Springer-Verlag Berlin Heidelberg, Secaucus, NJ, USA, 2001.

[Kra02] Rastislav Kralovic. Partitioning of hypercubes. Unpublished research notes, 2002.

[NR01] Joseph Naor and Yuval Rabani. Tree packing and approximating k-cuts. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 26–27, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[RS02] R. Ravi and Amitabh Sinha. Approximating k-cuts via network strength. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 621–622, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[SV95] Huzur Saran and Vijay V. Vazirani. Finding k cuts within twice the optimal. *SIAM J. Comput.*, 24(1):101–108, 1995.