Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Számítástudományi és Információelméleti Tanszék

Módszerek közel optimális vágások keresésére

Diplomaterv

Készítette
Zalavári Márton

Konzulens
dr. Friedl Katalin

2021. október 10.

Tartalomjegyzék

Kivonat

Jelen dokumentum egy diplomaterv sablon, amely formai keretet ad a BME Villamosmérnöki és Informatikai Karán végző hallgatók által elkészítendő szakdolgozatnak és diplomatervnek. A sablon használata opcionális. Ez a sablon LaTeX alapú, a *TeXLive* TeX-implementációval és a PDF-LaTeX fordítóval működőképes.

Abstract

This document is a LaTeX-based skeleton for BSc/MSc theses of students at the Electrical Engineering and Informatics Faculty, Budapest University of Technology and Economics. The usage of this skeleton is optional. It has been tested with the *TeXLive* TeX implementation, and it requires the PDF-LaTeX compiler.

1. fejezet

Bevezetés

Gráfokon vágások keresése a (számítógép)hálózatok megjelenése óta egy sokat kutatott tématerület.

A polinom időben megoldható problémáktól, mint az egyszerű minimális vágás, a még közelítéssel is nehezen kezelhető problémákig, mint a maximális vágás, széles a paletta. Ezáltal nem csak az elméleti eredmények érdekesek, hanem az alkalmazási területek is rendkívül sokrétűek. Például egy klaszterezési probléma is megfogalmazható vágás kereséseként, ha az adatokat gráffal reprezentáljuk, úgy hogy az adatpontok a gráf csúcsai, az élek súlyát pedig valamilyen hasonlósági metrikából kapjuk. Ekkor a gráf k részre való vágása egy k klasztert előállító algoritmus.

A munkában először bemutatok általános optimalizációs technikákat, ezután több különböző vágási problémát vizsgálok meg, és elemzem, hogy milyen módon lehet az optimalizációs technikákat alkalmazni. A munka végén pedig a gyakorlati eredményeket és tapasztalatokat gyűjtöm össze.

2. fejezet

Optimalizációs technikák

Ebben a fejezetben általános optimalizációs technikákat tekintünk át.

2.1. Lineáris programozás

2.2. Kvadratikus programozás

A kvadratikus programozás a lineáris programozásnál egy általánosabb technika, hiszen megengedi négyzetes tagok jelenlétét a célfüggvényben és a korlátokban is. Ezzel az alkalmazások körét jóval kibővíti, ugyanakkor az általános feladat megoldása sokkal nehezebbé válik.

Az általános n változós, m korláttal rendelkező feladatot a következő mátrixos alakban írhatjuk le tömören.

Paraméterek:

Q	$\in \mathbb{R}^{n \times n}$	$n \times n$-es valós, szimmetrikus* mátrix, a négyzetes tagok együtthatói
c	$\in \mathbb{R}^n$	A lineáris tagok együtthatói
A	$\in \mathbb{R}^{m \times n}$	$m \times n$-es valós mátrix, a korlátokban szereplő együtthatók
b	$\in \mathbb{R}^m$	A korlátokban szereplő konstans tagok

Változók:

x	$x \in \mathbb{R}^n$	változók

Célfüggvény:

$$\min_x \frac{1}{2} x^T Q x + c^T x \tag{2.1}$$

Korlátok:

$$Ax \leq b \tag{2.2}$$

A felírásnál Q jellemzően egy szimmetrikus mátrix, ekkor a $q_{i,j}$ jelentése, a $x_i \cdot x_j$ változószorzat eggyütthatója, azonban mivel a pár kétszer is meg fog jelenni, így normálni

kell $frac12$-vel. Másik szokásos felírás, hogy Q egy felső háromszög mátrix. Ekkor ha $i \leq j$, akkor $q_{i,j}$ az az együtthatóval egyezik meg, különben nullával.

Említésre méltó megfigyelés még, hogy ha a kvadratikus polinomok helyett tetszőlegesen nagy fokszámú polinomok szerepelnek, a probléma mindig átírható klasszikus kvadratikus alakra, egyszerűen csak új változókat kell bevezetni, úgy, hogy a fokszámok csökkenjenek. Ezzel persze mind a változók száma, mind a kifejezések hossza rendkívüli módon megnőhet. Ha csak egyszerű mohó módszerrel próbálkozunk, akkor akár exponenciálisan is. Nem ismert, hogy van-e jó stratégia a polinomok fokszámának ilyen módon történő csökkentésének.

A kvadratikus optimalizálásnak több speciális esete is kutatott tématerület. Egyik ilyen eset, ha a Q mátrix szimmetrikus pozitív definit. Ekkor a probléma ekvivalens a legkisebb négyzetek megkeresésének problémájával.

Ebben a dolgozatban most egy másik speciális esetet fogok elemezni, méghozzá megkötöm, hogy a változók csak és kizárólag binárisak lehetnek, és több korlát nem adható meg. Mivel a szakirodalom egyszerűen csak QUBO (Quadratic Unconstrained Binary Optimization) néven hivatkozik erre a fajta felírásra, én is így teszek a továbbiakban.

Az általános n változós, feladat így a következő alakban írható.

Paraméterek:

Q	$\in \mathbb{R}^{n \times n}$	$n \times n$-es valós, szimmetrikus* mátrix, a négyzetes tagok együtthatói
c	$\in \mathbb{R}^n$	A lineáris tagok együtthatói

Változók:

x	$x \in \mathbb{B}^n$	változók

Célfüggvény:

$$\min_x \frac{1}{2} x^T Q x + c^T x \tag{2.3}$$

Korlátok:

$$\emptyset \tag{2.4}$$

A bináris változók alatt szokásosan 0 vagy 1 értékeket jelölünk, így a továbbiakban is $\mathbb{B} = \{0, 1\}$. Ugyanakkor itt érdemes kitérni, hogy bizonyos alkalmazásoknál inkább a $\{-1, 1\}$ alaphalmazt tekintik. Erre elterjedt módon Ising modellként hivatkozhatunk, a fizikai spin irányultságok miatt. Bizonyos esetekben ezt könnyebb lehet elméleti síkon is kezelni, azonban a kettő között (QUBO és Ising modell) egyszerű lineáris transzformáció ad átjárást, így lényegi különbséget végül nem ad. A D-Wawe Systems gyűjtőnéven, BQM-ként (Binary Quadratic Model) hivatkozik a két problémára együttesen.

A továbbiakban tehát feltesszük, hogy a bináris változóink 0 vagy 1 értéket vesznek fel. Ekkor a korábban felírt általános alakot rögtön egyszerűbb alakra hozhatjuk, hiszen bármely változó megegyezik saját négyzetével. Így elég a négyzetes tagokat felírni, mert az esetleges lineáris tagokat belevehetjük a négyzetes tagok közé. Továbbá az $\frac{1}{2}$-del való szorzás sem tesz hozzá így már érdemben a felíráshoz, ennek ellenére benne hagytam.

Paraméterek:

$Q \quad \in \mathbb{R}^{n \times n}$ $n \times n$-es valós, szimmetrikus* mátrix, a négyzetes tagok együtthatói

Változók:

$x \quad x \in \mathbb{B}^n$ változók

Célfüggvény:

$$\min_x \frac{1}{2} x^T Q x \tag{2.5}$$

Korlátok:

$$\emptyset \tag{2.6}$$

2.3. Korlátolt illetve korlátmentes programozás

A korábbi alfejezetben röviden kitértünk arra, hogy a kvantumszámítógépek segítségével elméletileg hatékonyan megoldhatóak a bináris kvadratikus programozási feladatok, amennyiben nem szabunk további korlátokat. Továbbiakban egyszerűen QUBO (Quadratic Unconstrained Binary Optimalization) néven hivatkozhatunk erre a problémára[9].

Azonban felmerül a kérdés, hogy miként tudjuk mégis használni a gyakorlatban ezt a technikát, nem szorítja meg a kezünket túlságosan az, hogy nem adhatunk meg korlátot, és egy "egyszerű" függvény szélsőértékét keressük? A válasz szerencsére nemleges, mely látszik a következő, röviden bemutatott technikából [3].

Általánosan elmondható, hogy egy optimalizációs esetben kétféle követelményt állítunk a rendszerrel szemben. Ezek közül az egyik típus a erős (hard) követelmény. Ezeket mindenképpen szeretnénk, hogy teljesítse a rendszer, a követelmény sérülése használhatatlanná tenné az eredményt. Általában ezeket a típusú követelményeket korlátként adjuk hozzá a programozási feladathoz.

A másik típus a gyenge (soft) követelmény. Ezeket is szeretnénk minél jobban teljesíteni, de nem követeljük meg az összes teljesítését. Ez olykor nem is lenne lehetséges, hiszen elég valószínű egy olyan való életbeli probléma, hogy minden gyenge követelmény hozzávétele inkonzisztenssé tenné a rendszert. (Hiszen nem lehet minden tökéletes.) Ezeket a követelményeket általában igyekszünk az optimalizálandó célfüggvénybe belefogalmazni.

A kettő követelménytípus között azonban van átjárás. Hiszen csak annyiról van szó, hogy az erős követelményeket is bele tehetjük a célfüggvénybe, egyszerűen csak szorozzuk meg őket valamilyen jó nagy együtthatóval (úgynevezett büntetőtaggal), hogy bármelyik erős követelmény sérülése esetén az optimumtól nagyon távol essen a függvény értéke. Ezt a technikát én is felhasználom a megoldásokban, a képletekben és kódmintákban inf szimbólummal jelölve ezt az alkalmasan megválasztott nagy konstans számot.

Ugyanakkor a konstans(ok) megválasztása koránt sem triviális feladat. Bár elméletileg az eredmény ugyanaz, ha túl nagyra választjuk a büntetőtagot, akkor a nem optimális megoldások nagyon messze esnek az optimálistól, ha túl kicsire, akkor pedig túl közel. Egy heurisztikus elven működő optimalizáló szoftver mind a két szélsőséges eset károsan befolyásolhat. Hiszen ha a helytelen megoldások túl messze vannak az optimumtól, akkor lehet, hogy rossz helyen keresgetünk, és nem találjuk meg a "szűk" kis optimumot, amíg

ha túl közel esnek, akkor egy nagyon rossz megoldásra is rámondhatjuk, hogy már ”elég jó”.

Ráadásul tapasztalatok azt is mutatják, hogy az explicit korlátokat megoldó programok sokkal jobban tudják kezelni, amíg ha a célfüggvénybe fogalmazunk bele mindent, az meglehetősen ”ködösít” a megoldó számára, hiszen nem tud lényegi különbséget tenni a különböző funkciójú változók között.

3. fejezet

Vágások

Ebben a fejezetben különböző nehézségű vágási problémákat tekintek, illetve megvizsgálom, hogy miként fogalmazhatók meg valamilyen optimalizációs programként.

3.0.1. Jelölések

Ha máshogy nem jelzem, a továbbiak $G = (V, E)$ jelölje a gráfot, ahol szokásos módon $n = |V|$ és $m = |E|$. Jelölje továbbá $[k]$ az 1-től k-ig tartó egészek halmazát. inf pedig egy elegendően nagy számkonstanst jelöljön.

3.1. Minimális vágás

3.1.1. Kombinatorikus megoldással

Kombinatorikusan is "könnyen" megoldható feladat. Ha ismerünk két csúcsot s, t, melyeknek külön részbe kell kerülniük, akkor a feladat megoldható a jól ismert max-folyamot előállító algoritmussal. (Amennyiben nem ismerünk két ilyen csúcsot, akkor a gráf egy s csúcsát fixálhatjuk, és az algoritmust lefuttatjuk minden st párra.)

3.1.2. Lineáris programozással

3.1.3. QUBO

A feladat felírható QUBO formában is. Az ötlet, hogy minden csúcsot jelöljünk egy x_i bináris változóval, amelynek értéke mutatja, hogy az i. csúcs melyik csoportba tartozik. A célfüggvényt úgy konstruáljuk, hogy egy él súlyát csak akkor számoljuk bele az összegbe, ha az a két csoport közt fut, és ezt az összeget szeretnénk minimalizálni.

Ekkor persze a minimum az a triviális megoldás lenne, hogy minden csúcsot egy csoportba osztunk be. Vagyis azt kell elérni még, hogy egyik csoport sem üres. Ezt tudjuk kezelni, csakúgy mint a kombinatorikus esetben, azaz tegyük fel, hogy tudjuk egy s, t csúcspárról, hogy nekik külön kell kerülniük. (Ha ezt nem tudjuk, akkor s fixálásával $(n-1)$-szer kell futtassuk a megoldó algoritmust.) Ha s és t külön csoportba kell kerüljön, akkor feltehetjük, hogy $x_s = 0$ és $x_t = 1$. Vezessünk be büntető tagokat a célfüggvénybe, hogy ezt mindenképpen kikényszerítsük. (Megjegyzés: lényegtelen, hogy a változókra direkt módon, vagy a változók négyzetére szabjuk a büntetést.)

$$\min_x \left\{ \sum_{\{i,j\}\in E(G)} w_{ij} \cdot (x_i - x_j)^2 + x_s \cdot inf + (1 - x_t) \cdot inf \right\} \tag{3.1}$$

3.2. Maximális vágás

3.2.1. Kombinatorikus megoldás

Kombinatorikusan is ”nehéz” feladat. Jellemzően valamilyen kombinatorikus közelítő algoritmust alkalmaznak rá. 2-approximációt viszonylag gyors algoritmus segítségével is lehet már adni, de tetszőlegesen közeli approximációt már nem.

3.2.2. Lineáris programozással

IP felírható rá, ezúttal minden $\{i, j\} \in E(G)$ élre (pontosabban később látjuk majd, hogy minden csúcspárra) vezetünk be egy d_{ij} bináris változót, melynek értéke akkor 1, ha az él benne van a vágásban. Ekkor a célfüggvényben világos, hogy azon élekre adjuk össze a súlyokat, melyek benne vannak a vágásban, és ezt szeretnénk maximalizálni.

$$\max_d \left\{ \sum_{\{i,j\}\in E(G)} w_{ij} \cdot d_{ij} \right\} \tag{3.2}$$

További korlátokat is fel kell vegyünk, különben a maximum akkor áll elő, ha a változók azonosan 1 értéket vesznek fel. Amit ki szeretnénk fejezni további korlátokkal, hogy bármely háromszögből pontosan nulla vagy kettő él lehet benne a vágásban. Ez nyilvánvalóan egy szükséges feltétele egy helyes vágásnak, viszont az elégségesség is következik, ha a gráf teljes. Ehhez viszont nem létező éleket kell hozzávegyünk a gráfhoz (és így a változók száma is $n \cdot n$ -re növekszik.) Az élsúlyokat pedig vagy úgy kell megválasszuk, hogy ne zavarják a maximális vágást, (például ha az élsúlyok nemnegatívak, akkor az azonos 0 választás megfelelő) de általában még az sem probléma, ha nem rendelünk az új élekhez súlyokat, hiszen elég ha a célfüggvényben csak az eredeti élekre összegezzük a súlyokat.

$$d_{ij} \leq d_{ik} + d_{kj} \tag{3.3}$$
$$d_{ij} + d_{ik} + d_{kj} \leq 2 \tag{3.4}$$

Érdemes pár szót ejteni a relaxált feladatról, ugyanis azzal 2-közelítő algoritmust kapunk. Ez persze egyáltalán nem biztos, hogy gyakorlatban is hasznos, hiszen egyszerű mohó algoritmussal is elérhető 2-approximáció [1, 7].

3.2.3. QUBO

QUBO-ban természetesen felírható. Az élekre (csúcspárokra) felírt változók helyett, ismét csak a csúcsokra definiáljuk a bináris változóinkat. Csakúgy, mint a minimális vágás QUBO-ként felírt alakjában, annyi különbséggel, hogy ezúttal maximalizálni szeretnénk.

Sőt, a helyzetünk ezúttal sokkal könnyebb, hiszen nincsenek elfajuló esetek, melyek esetén beállna a maximum, illetve további korlátok felvételére, így büntető tagok hozzávételére sincs szükség.

$$\max_x \left\{ \sum_{\{i,j\}\in E(G)} w_{ij} \cdot (x_i - x_j)^2 \right\} \tag{3.5}$$

Ugyanez a négyzetes részt kibontva a következőképpen alakul:

$$\max_x \left\{ \sum_{\{i,j\}\in E(G)} w_{ij} \cdot (x_i^2 + x_j^2 - 2x_i x_j) \right\} \tag{3.6}$$

3.3. Maximális K-vágás

3.3.1. Lineáris programozással

Az ötlet itt is hasonlít a sima maximális vágáséhoz. Az élekre ezúttal is bevezetünk egy bináris változót, mely azt jelzi, hogy az él két különböző csoport között fut. Vagyis $y_{uv} = 1$, akkor és cs ak akkor, ha u, v él csoportok között fut. Ezen felül viszont azonosítanunk kell minden egyes csúcsot, hogy ő melyik csoportba fog kerülni. Ezeket szintén binárisan kódoljuk $n \cdot K$ db változóban, ahol x_{vi} akkor és csak akkor 1, ha a v. csúcs az i. halmazba kerül.

A célfüggvény egyértelműen elkészíthető, egyszerűen azon élekre kell összeszámolnunk a súlyokat, melyek két különböző csoport között futnak.

A korlátoknál az elsőként felveendő legfontosabb dolog, hogy bármely v, pontosan egy csoportba tartozhasson, ezt egy szummával adhatjuk meg.

A további korlátok pedig azt biztosítják, hogy a változók konzisztensek, azaz y_{uv} valóban csak akkor legyen 1, amikor u és v különböző csoportba kerülnek.

$$\max_y \sum_{\{u,v\}\in E} w_{uv} y_{uv} \tag{3.7}$$

$$\sum_{i\in[K]} x_{vi} = 1, \qquad v \in V, \tag{3.8}$$

$$x_{ui} - x_{vi} \le y_{uv}, \qquad \{u,v\} \in E,\ i \in [K], \tag{3.9}$$

$$x_{vi} - x_{ui} \le y_{uv}, \qquad \{u,v\} \in E,\ i \in [K], \tag{3.10}$$

$$x_{ui} + x_{vi} + y_{uv} \le 2, \qquad \{u,v\} \in E,\ i \in [K], \tag{3.11}$$

$$x_{vi} \in \{0,1\}, \qquad v \in V,\ i \in [K], \tag{3.12}$$

$$y_{uv} \in \{0,1\}, \qquad \{u,v\} \in E, \tag{3.13}$$

Egy ehhez nagyon hasonló (több korlátot igénylő) probléma felírása megtalálható az internetet, részletes indoklással [5].

3.3.2. QUBO-val

3.3.2.1. One-hot encoding

A maximális vágásnál a QUBO forma segítségével jelentősen le tudtuk csökkenteni a változók számát, hiszen nem kellett az élekre bevezetnünk változókat. Ezt a trükköt próbáljuk meg itt is. Vagyis a lineáris programhoz hasonlóan, a x_{vi} változókat megtartjuk, az a tény pedig, hogy egy uv él kettő különböző csoport között fut, pedig egyértelműen következik, ha $x_{vi} * x_{uj} = 1$ valamely $i \neq j$ és $u \neq v$-re.

Ez alapján meg is konstruálhatjuk és fel is írhatjuk a maximalizálandó célunkat. (Egyelőre megengedjük további korlátok létezését is.) Világos, hogy minden, csoportok között futó élet pontosan egyszer fogunk megszámolni, mivel minden csúcs pontosan egy csoportnak lehet az eleme.

$$\max_{x} \sum_{\substack{\{u,v\}\in E \\ \{i,j\}\in[K]}} w_{uv}(x_{vi} \cdot x_{uj}) \tag{3.14}$$

$$\sum_{i\in[K]} x_{vi} = 1, \qquad v \in V \tag{3.15}$$

A korlátokat viszont szeretnénk kiiktatni a felírásból, így azokat valamilyen büntető tagként kell hozzáadni a célfüggvényhez. Amit így büntetni szeretnénk, azok azok az esetek, amikor egy csúcs két különböző csoportba is bekerül. Egy v csúcs pontosan akkor kerül bele két különböző csoportba, ha $x_{vi} \cdot x_{vj}$ szorzat értéke 1 valamilyen $i \neq j$-re. Így a célfüggvényünk kiegészül még a következő taggal:

$$\sum_{\substack{u\in V \\ i,j\in[K] \\ i\neq j}} (-inf) \cdot x_{vi} \cdot x_{vj} \tag{3.16}$$

Szerencsére elég a kisebb-egyenlő esetet büntetni, hiszen impliciten teljesül, hogy az egy csúcshoz tartozó bináris változók közül legalább az egyik 1 értéket vesz fel. Hiszen ha lenne egy változóbehelyettesítés, melynél egy csúcs nem kerülne bele egyetlen csoportba sem, a belőle kimenő élek nem kerülnének bele a vágásba. Így bármelyik csoportba is kerüljön bele, így a célfüggvény értéke biztosan nőni fog.

Így a végső QUBO forma:

$$\max_x \left\{ \sum_{\substack{\{u,v\}\in E \\ i,j\in[K]}} w_{uv}(x_{vi}x_{uj}) + \sum_{\substack{u\in V \\ i,j\in[K] \\ i\neq j}} (-inf)\cdot x_{vi}\cdot x_{vj} \right\} \tag{3.17}$$

3.3.2.2. Binary encoding

Más megoldást is megpróbáltunk keresni, amellyel a QUBO-t tovább egyszerűsíthetjük, a változók számát csökkenthetjük. Erre egy lehetséges irányzat, hogy azt, hogy a csúcs mely csoportba tartozik nem úgy kódoljuk, hogy minden csoport-csúcs párhoz egy bináris változót rendelünk, és a csúcs csoportját "one-hot" módon kódoljuk, hanem kifinomultabb módon, a csoport sorszámát binárisan kódoljuk le. Ennek előnye lehet, hogy így nem szükséges $n \cdot K$ darab bit, hanem elég volna $n \cdot \log K$, ráadásul a problémás korlátot, amely azért szükséges, hogy biztosítsuk, hogy egy csúcs pontosan egy csoportba kerüljön, ezt impliciten teljesítené. Motivációt ad egy friss kutatás, ahol ezt a módszer sikeresen alkalmazták kvantumgép esetében, azonban ezt natív módon tették meg. A mi kérdésünk, hogy vajon ezt meg lehet-e csinálni eggyel magasabb absztrakciós szinten, csupán QUBO felírást használva [2].

A továbbiakban az egyszerű szemléltetés kedvéért feltesszük, hogy a készítendő csoportok száma valamilyen kettő hatvány, vagyis $K = 2^m$, így $log(K)$ egész. Ezzel mindig feltehető, hogy minden bit szabadon lehet 0 és 1, mert nem lesznek "tiltott" csoportok. Később kitérek arra az esetre, ha ez nem így volna.

A megoldáshoz elsősorban minden csúcshoz definiálunk kell az ő csoportját tartalmazó változókat. Az u. csúcs csoportját jelölje x_u, illetve ennek bináris felírását tekintve, az x_u i. bitjét jelölje x_{ui}.

Amire szükségünk van, az egy bináris függvény, amely megmondja, hogy két csúcs különböző csoportba esik-e. Ha igen, akkor természetesen a köztük lévő él súlya hozzá kell adódjon a maximalizálandó célfüggvényhez. Ellenkező esetben viszont figyelmen kívül kell hagyni ezt a súlyt.

$$D_{uv} = diff(x_u, x_v) \tag{3.18}$$

$$\max_x \left\{ \sum_{\{u,v\}\in E} D_{uv}\cdot w_{uv} \right\} \tag{3.19}$$

Annak jelölésére, hogy az u és v csúcsok különböző csoportokba kerülnek-e, használjuk a D_{uv} bináris változót. Két egyedi bit összehasonlítására pedig d_{uvi} változót definiáljuk, melynek jelentése, hogy az x_u és x_v számok i. bitjük különböző. d_{uvi} így a KIZÁRÓ VAGY műveletének eredménye x_{ui} és x_{vi}-re alkalmazva.

Mivel két szám pontosan akkor különböző, ha legalább egy helyiértékükön eltérnek, ezért a D_{uv}-t érdemes úgy előállítani, hogy minden bit különbözőségét VAGY kapcsolatba állítjuk.

$$D_{uv} = \bigvee_{i \in [\log K]} d_{uvi} \tag{3.20}$$

$$d_{uvi} = x_{ui} \oplus x_{vi} \tag{3.21}$$

Kiszámolható, hogy ezzel összesen hány változóra van szükségünk. A csoportok elkódolásához kell $n \cdot \log K$ db. Minden csúcspár összehasonlításához kell $\log K + 1$ darab, ez összesen $\binom{n}{2} \cdot (\log K + 1)$.

(Spórolhatnák, ha csak azokat a csúcspárokat néznénk, amelyek között fut él?)

3.3.2.3. Logikai kapuk megvalósítása

EGYENLŐ:

$$(x - y)^2 = -2xy + x + y \tag{3.22}$$

NEMEGYENLŐ:

$$1 - (x - y)^2 = 1 + 2xy - x - y \tag{3.23}$$

ÉS:

$$xy - 2(x + y)z + 3z \tag{3.24}$$

VAGY:

$$(x + y) + z - 2(x + y)z + xy = \tag{3.25}$$
$$x + y + z - 2xz + -2yz + xy \tag{3.26}$$

x	y	z	büntetés
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	3
1	1	1	0

3.1. táblázat. VAGY

NEMVAGY:

$$1 - (x + y) - z + 2(x + y)z + xy = \tag{3.27}$$
$$-x - y - z + 2xz + 2yz + xy \tag{3.28}$$

XOR:

$$4(xy - 2(x + y)z + 3z) + (...) = \tag{3.29}$$
$$-2xy - 2(x + y)t + (x + y) + t + 4zt + 4xy - 8z(x + y) + 12z = \tag{3.30}$$
$$(x + y) + t + 12z + 2xy - 2(x + y)t + 4zt - 8z(x + y) \tag{3.31}$$

4. fejezet

Gyakorlati eredmények

Ebben a fejezetben alkalmazom a korábbiakban megismert elméleti módszereket, megvizsgálom milyen futásidőt érnek el különböző megoldóprogramok használatával, illetve összevetem az így kapott eredményeket a kombinatorikus algoritmusok hatékonyságával.

Az implementációkat a Google által üzemeltetett Colaboratory segítségével készítettem Python nyelven. A választás többek között azért esett erre a rendszerre, mert a D-Wave Systems által, Python nyelven publikált könyvtárat szerettem volna felhasználni, és ez így nem csak minimális konfigurációval megoldható, de az eredmények könnyen rekonstruálhatók a konténer technológia miatt, és nem befolyásolja őket a lokális gépemen eszközölt idő közbeni módosítások.

A teszteléshez generált gráfok előállításához és reprezentálásához a NetworkX könyvtárat használtam fel [6].

4.1. QUBO-k megoldása

A D-Wave Ocean nevű programcsomagja több lehetőséget kínál QUBO formák megoldására. Van egzakt megoldója, ez azonban nagyon kicsi bemenetekre is használhatatlan a gyakorlatban, hiszen működési elvét tekintve, minden lehetőséget kipróbál. Ezen kívül van lehetőség kvantum számítás szimulálására, vagy egy valós kvantumgépnek is elküldhetjük a problémát a megfelelő API használatával [8].

Lényeges észrevétel, hogy a D-Wave megoldója csak minimalizálni tud, ezért a korábban bemutatott QUBO-kat is ilyen alakra kell hozni. Ez szerencsére könnyen megy, mert ha eredetileg maximalizálni szerettünk volna, egyszerűen minden tagot -1-gyel kell megszoroznunk, és az így kapott függvényt minimalizálni.

A helyes konfiguráció után van lehetőségünk a korábban bemutatott QUBO modellek tényleges próbájára.

A szimulált kvantumgépen való megoldás mellett két fő iránya van a D-Wave számítógépek használatának. Ebből az egyik a direkt módon beágyazása a problémának kvantumszámítógépre, a mások pedig egy hibrid megoldót használ, mely klasszikus optimalizációkat is végez.

A direkt QPU-t (Quantum processing unit) használatával sajnos probléma ütköztünk, ahogy nőtt a probléma mérete, nem túl nagy, vagyis pár száz csúcsot tartalmazó gráfokra is túl hosszú időnként túl sokat kellett kvantum erőforrásra várni. Sajnos nem is teljesen sikerült kideríteni, hogy ennek miért van köze a probléma méretéhez, de nagyobb

bemenetek esetén jellemzőbb volt ez a viselkedés. Azért is érdekes, mert a probléma megoldása a QPU-n hamar elkészül (amennyiben eljut odáig), amely kiderül a jelentésekből. Az erőforrásra várakozás azért tűnik valószínűnek, mert a kliens, ahonnan a hívást végzem többször egy `get_solvers` nevű függvény belsejében várakozik. Ugyanakkor viszont nem ez tűnik az egyetlen oknak, hiszen kisebb problémák megoldásánál nem jelentkezik ez a probléma, így azt is sejtjük, hogy a probléma beágyazása valamiért nem történik meg hatékonyan.

A QUBO-kat, és általánosságban a kvadratikus programozási feladatokat persze más, klasszikus megoldóprogramokkal is használhatunk. Egy kereskedelmi forgalomban lévő, egyik piacvezető lineáris programozás megoldó szoftver a Gurobi, melynek bár nem fő iránya a kvadratikus programok megoldása, ezeket is támogatja [4].

Sajnos az eredmények viszont itt sem voltak bíztatóak, viszonylag kevés változószámnál is reménytelennek tűnt például egy QUBO megoldása, illetve nem sokkal később az ingyenes licenc korlátaiba ütköztünk, amennyiben túl sok változót szerettünk volna hozzáadni a modellhez.

4.2. Gráf generálása

Az alkalmazások ellenőrzésére speciális gráfokat generáltam, melyen így előre meg tudtam határozni a várható minimális, illetve maximális vágásokat. A generált gráf rendelkezik négy különböző paraméterrel, ezek N, K, p és q. A gráfot ezek után jelöljük $G(N, K, p, q)$-val. Az ötlet csupán annyi, hogy előre meghatározunk K darab diszjunkt halmazt, hogy mindegyik csoportban pontosan N darab csúcs legyen. Ezután minden élről egyénileg döntünk, hogy őt hozzávesszük-e a gráfhoz. Amennyiben egy él egy csúcs N-esen belül fut, ez a valószínűség legyen q, amennyiben két különböző csúcs N-es között fut, akkor legyen ez a valószínűség p. Világos, hogy ha p kicsi, q pedig nagy, akkor K klasztert kapunk, ahol az élek sűrűek, a csoportok között viszont ritkák. Ellenkező esetben K darab egymástól majdnem független ponthalmazt kapunk, ahol a halmazok között viszont sok él hozzá van adva a gráfhoz. Az előbbivel egy minimális vágás várhatóan a K klasztert szétválasztó vágás lesz, utóbbi esetben pedig ugyanígy a maximális vágásra lesz egy jó becslésünk.

Érdemes megvizsgálni az elfajuló eseteket. $G(N, K, 0, 1)$ esetén K darab különálló egyenként N csúcs teljes gráfot kapunk. $G(N, K, 1, 0)$ esetén pedig ennek a komplementerét kapjuk.

Az élsúlyokat pedig egy véletlenszámmal generáltam minden élre. Jellemzően 0 és 1 közötti egyenletes eloszlással.

Érdemes észben tartani, hogy $n = |V| = N \cdot K$, tehát a kicsi n és nagy N nem keverendő.

4.3. Minimális vágás

4.3.1. Klasszikus

A probléma klasszikusan is megoldható, például folyamalgoritmus segítségével. Referenciaként a NetworkX-ben ilyen módon implementált `minimum_cut` függvényt fogom használni.

```
  cut_value, partition = nx.minimum_cut(G, s, t, 'weight')
```

4.1. kódrészlet. Min-cut flow

4.3.2. QUBO

Egy n változós bináris kvadratikus kifejezést megadhatunk tömören mátrixos formában is, ahol a Q mátrix q_{ij} eleme az $x_i x_j$ együtthatója. Mivel ezt a formát képes kezelni a legtöbb a megoldó, (vagy ha nem, akkor könnyen átalakítható olyan formára) ezért a továbbiakban ilyen formában adjuk meg.

A minimális vágás problémája könnyen felírható QUBO-ként, ahogyan azt korábban is láttuk. Az egyetlen "trükk", hogy két speciális pontot ki kell jelölni, amelyeknek külön csoportba kell kerülniük. Mivel azonban ez a klasszikus folyamalgoritmusnál is így működik, ezért ilyen tekintetben ez a megoldás sem alsóbbrendűbb.

```
  for u, v in G.edges:
    w=G.edges[u,v]['weight']
    Q[(u,u)]+= 1*w
    Q[(v,v)]+= 1*w
    Q[(u,v)]+= -2*w

  Q[(s,s)]+=inf
  Q[(t,t)]-=inf
```

4.2. kódrészlet. Min-cut QUBO

4.4. Maximális vágás

A maximális vágás klasszikus példája a QUBO-nak, ezt könnyen elintézhetjük. Éppen úgy működik, mint a minimális vágás, de nem szükségesek a büntető tagok. Érdemes megfigyelni, hogy a változók száma pontosan $|V| = n$.

```
  for u, v in G.edges:
    w=G.edges[u,v]['weight']
    Q[(u,u)]+= -1*w
    Q[(v,v)]+= -1*w
    Q[(u,v)]+= 2*w
```

4.3. kódrészlet. Max-cut QUBO

4.5. Maximális K-vágás

A maximális K-vágásnál több trükköt is be kellett vetnünk. Az elméleti megfontolásokon felül, itt már csupán annyit kell még kezelni, hogy a változókat jól osszuk ki. Vagyis az x_{ui} változót kettő indexszel indexeltük, azonban most egy index kell, ezt pedig úgy oldjuk meg, hogy a változókat sorba rakva blokkonként következnek az egy csúcshoz tartozó változók, azaz x_{ui} az $(uK + i)$. változó lesz. Ennek megfelelően használjuk tehát az indexeket.

Ebből továbbá következik az a fontos megfigyelés is, hogy a változók száma itt $n \cdot K = N \cdot K^2$.

```
for u, v in G.edges:
  for i in range(K):
    for j in range(K):
      if (i!=j):
        w=G.edges[u,v]['weight']
```

```
        Q[(u*K+i,v*K+j)] += -1*w

for u in range(K*N):
  for i in range(K):
    for j in range(K):
      if (i!=j):
        Q[(u*K+i,u*K+j)] += inf
```

4.4. kódrészlet. Max-K-cut QUBO

```
for u, v in G.edges:
  for i in range(K):
    for j in range(i+1, K):
      w=G.edges[u,v]['weight']
      Q[(u*K+i,v*K+j)] += -1*w
      Q[(v*K+i,u*K+j)] += -1*w

for u in range(K*N):
  for i in range(K):
    for j in range(i+1, K):
      Q[(u*K+i,u*K+j)] += inf
```

4.5. kódrészlet. Max-K-cut QUBO (alternatív)

A maximális K-vágásra pár futásidőt szemléltetően, egy táblázatot készítettem néhány gráfra. A táblázat első két sorában a $K = 2$ eset miatt, az egyszerű maximális vágásnál látott módon oldottam meg, amíg a 3-4. sorban az általános K-vágást alkalmaztam ugyanarra a gráfra. A gráfoknál $p = 0.95$ és $q = 0.05$ paramétereket használtam, így ezeket nem jelöltem külön.

A futásidőknél sokszor jellemzőek voltak akár 1-2 másodperces eltérések is ugyanarra a bemenetre, a sok külső befolyásoló tényező miatt. (Kezdve csak azzal, hogy a Hibrid esetnél interneten keresztüli kommunikációval oldjuk meg a problémát, ezért csak a hálózati kommunikáció egy elég jelentős állandón változó tényező.) Mindezért csak közelítőleg másodperces pontosságot használtam.

A vágások várható optimumát is feltüntettem ezresekre kerekítve. Ez egyrészt kijön a saját méréseimből, másrészről, ha $p = 1, q = 0$ közelítést használjuk, akkor $N \approx (N-1)$ miatt, expliciten kiszámolható a $\binom{K}{2} \cdot N^2 \cdot E(W)$, képlettel, ahol $E(W)$ az élsúlyok várható értéke.

inf értékére egységes mindenhol 10000 használtam.

Gráf	Vágás optimuma	Szimulált kvantum (s)	Hibrid (s)
$G(N = 100, K = 2, p, q)$	5000	0.37	2.5
$G(N = 1000, K = 2, p, q)$	500000	32.5	11.5
$G(N = 100, K = 2, p, q)$	5000	0.9	3.5
$G(N = 1000, K = 2, p, q)$	500000	73	21
$G(N = 100, K = 3, p, q)$	15000	6	2.5
$G(N = 50, K = 4, p, q)$	7500	6	3
$G(N = 100, K = 4, p, q)$	30000	25	8

4.1. táblázat. Futásidők különböző gráfokra

Sajnos a futási idők elemzése önmagában nem elég, fontos megvizsgálni, hogy valóban jó (közelítő) eredményeket kaptunk-e. $K = 2$ esetre minden esetben megtaláltuk az optimumot, bármely elkódolást is használjuk. Sajnos $K > 2$ esetre viszont elképesztően rossz vágásokat kaptunk vissza. Például az utolsó esetben a 30000 körüli vágásérték helyett a szimulált kvantum esetén 23000 körüli értéket kaptunk. Amely nem csak, hogy az optimumtól nagyon távol esik, (több mint 20%), hanem ha végig gondoljuk, egy véletlenszerű vágás, amikor a K csoport mindegyikébe minden eredeti csoport K-ad részét

tesszük, akkor is várhatón ilyen körüli vágást kell kapjunk. Ugyanis ha $p = 1, q = 0$, akkor $K \cdot \frac{N}{K} \cdot (N - \frac{N}{K}) \cdot \binom{K}{2}$ adja az élek számát, amelyet még szorozni kell az élsúlyozás várható értékével. Ez az utolsó példára 22500-at ad. A kapott felosztáson továbbá "szemmel végignézve" is úgy tűnik, mintha véletlen felosztásról lenne szó. Hibrid számításnál általában valamivel jobb a eredményeket kapunk, és a felosztás megvizsgálásánál is olykor látszik, hogy a csúcsok mintha lassan elkezdenének megfelelően csoportokba tömörülni, azonban még mindig alig kapunk jobbat egy véletlenszerű vágásnál.

Azt sejtjük hogy valamilyen paraméterek konfigurálásával javítható lenne ez az eredmény, de egyelőre, a viszonylag kicsi felhasználóbázis és elérhető dokumentáció miatt nem sikerült ezt kideríteni. Az is elképzelhető, hogy az általam adott elkódolásban van a hiba, viszont ekkor érdekes, hogy $K = 2$ esetre mindkét módszer hibátlanul működik. Az viszont sajnos kiderült, hogy ebben a formájában a megoldó nem alkalmazható, hiszen mint láttuk, viszonylag kicsi (néhány száz csúcsból álló) gráfokra sem ad még jó közelítést sem.

Irodalomjegyzék

[1] Wenceslas Fernandez de la Vega – Claire Kenyon-Mathieu: Linear programming relaxations of maxcut. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07 konferenciasorozat. USA, 2007, Society for Industrial and Applied Mathematics, 53–61. p. ISBN 9780898716245. 9 p.

[2] Franz Georg Fuchs – Herman Øie Kolden – Niels Henrik Aase – Giorgio Sartor: Efficient encoding of the weighted max k-cut on a quantum computer using QAOA. 2. évf. (2021. 01) 2. sz., 1–14. p.

[3] Fred W. Glover – Gary A. Kochenberger: A tutorial on formulating QUBO models. *CoRR*, abs/1811.11538. évf. (2018). URL `http://arxiv.org/abs/1811.11538`.

[4] LLC Gurobi Optimization: Gurobi optimizer reference manual, 2021. URL `http://www.gurobi.com`.

[5] Christopher Hojny – Imke Joormann – Hendrik Lüthen – Martin Schmidt: Mixed-integer programming techniques for the connected max-k-cut problem. *Mathematical Programming Computation*, 13. évf. (2021. Mar) 1. sz., 75–132. p. ISSN 1867-2957. URL `https://doi.org/10.1007/s12532-020-00186-3`.

[6] NetworkX developers: NetworkX. `https://networkx.org/`.

[7] Svatopluk Poljak – Zsolt Tuza: The expected relative error of the polyhedral approximation of the max-cut problem. *Operations Research Letters*, 16. évf. (1994) 4. sz., 191–198. p. ISSN 0167-6377. URL `https://www.sciencedirect.com/science/article/pii/016763779490068X`.

[8] D-Wave Systems: D-Wave Ocean Software Documentation. `https://docs.ocean.dwavesys.com/`.

[9] Wikipedia contributors: Quadratic unconstrained binary optimization — Wikipedia, the free encyclopedia, 2021. URL `https://en.wikipedia.org/w/index.php?title=Quadratic_unconstrained_binary_optimization&oldid=1020700695`. [Online; accessed 28-Feb-2021].