Címkézett fák kompakt reprezentációja

Simon Máté
3. féléves alkalmazott matematika hallgató
Operációkutatás szakirány

Témavezető: Madarasi Péter

2022. december 19.

1. Bevezetés

Ebben a félévben az új témám a Címkézett fák kompakt reprezentációja.

A feladat a következő: Legyenek $A_1, A_2, \ldots, A_n$ kevés elemből álló halmazok. Vegyük ezek Descartes-szorzatának egy részhalmazát, amelyet általában valamilyen szabályrendszer definiál. A cél az, hogy ezen kombinációkat a minél hatékonyabban kódoljuk el úgy, hogy azokkal hatékonyan végezhessünk bizonyos műveleteket, például a reprezentált kombinációk módosítását vagy a köztük való keresést. Ezt a problémát az irodalomban nagy részt úgynevezett decision diagramokkal [1] reprezentálják.

A fő kérdés az, hogy milyen egyéb módon lehetne egy-egy ilyen kombináció halmazt hatékonyabban reprezentálni. Az elméleti vizsgálatok mellett célunk heurisztikus algoritmusok kidolgozása és implementálása is, és ezek összevetése a korábbi megoldásokkal, többek között a CUDD nevű C++ [2] és a dd [3] nevű python könyvtárakkal.

1.1. A félév menete

A félév első részében az előző, 1-2-3 sejtéssel kapcsolatos téma lezárásával foglalkoztunk: Befejeztük a cikkírást a témából, majd kiegészítve a tavalyi TDK dolgoza-

Témavezető: Madarasi Péter

tomat az új eredményekkel, beküldtük az OTDK dolgozatomat. Továbbá előkészítettük a témát arra, hogy a 2023-as Japán-Magyar [4] konferencián előadhassunk vele. Majd ezeket követően áttértünk az új téma, a címkézett fák kompakt reprezentációja, vizsgálatára.

A fent említett új téma kapcsán először az irodalom felkutatásával foglalkoztunk, hogy kialakuljon bennünk a kép, hogy milyen irányokba érdemes indulni és új fajta lehetséges reprezentációkon gondolkoztunk.

A beszámoló további részét a volt témában elért, új eredmény bemutatásával kezdjük. Ezután áttérve az új témára szemléltetjük a főbb megközelítési módszereket a feladatra az eddigi irodalomban, majd vázolunk új fajta megközelítési módszereket. Ezután ismertetünk nehézségi eredményeket a témával kapcsolatban, emellett nyitott nehézségi eredményeket is és potenciális megközelítési irányokat. Ezután megvizsgáljuk, hogy egy Sudoku tábla megoldásait miként tudjuk elkódolni minél tömörebben. Végezetül bemutatjuk, hogy a következő félévben milyen irányokban fogjuk folytatni a kutatást, mivel diplomamunkámnak a témája is ez lesz.

2. Az antifaktor feladat

Ebben a fejezetben bemutatjuk, hogy egy korábbi eredményünket felhasználva, lásd [5]-ben (3.5. Tétel), egy alternatív polinomiális algoritmus adható az úgynevezett Antifaktor problémára. Az antifaktor probléma a fokszám-előírt részgráf feladat egy változatát vizsgálja, ahol minden csúcson pontosan egy fok van megtiltva. Azaz adott egy összefüggő G gráf, egy $f : V \to \mathbb{Z}_+$ hozzárendelés és egy olyan H részgráfját keressük G-nek, hogy $d_H(v) \neq f(v)$ teljesül minden $v \in V$-re. Ezt a problémát Lovász megoldotta [6] és később általánosították [7]-ben és [8]-ban. Egyszerűen megmutatható, hogy a problémának mindig van megoldása, ha a gráf tartalmaz kört, ezért feltehetjük, hogy G egy fa.

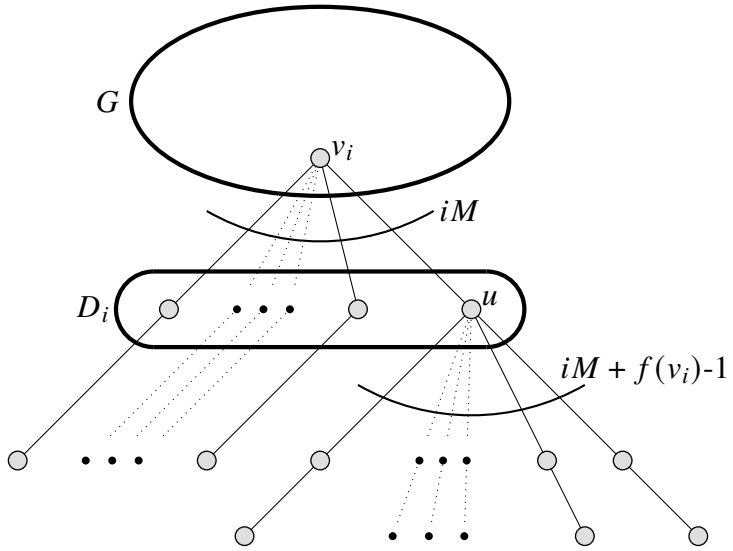
Mindenekelőtt tekintsük a következő definíciót, majd feladatot:

2.1. Definició. Egy adott G gráf $\{a, b\}$-élsúlyozásán egy $w : E \to \{a, b\}$ függvényt értünk, melyet megengedettnek nevezünk, ha a $z : V \to \mathbb{Z}$ indukált címkék egy megengedett színezését adják G-nek, ahol a, b különböző egész számok és $z(v) = \sum_{e \in \Delta(v)} w(e)$.

2.2. Feladat. *Adott egy G gráf, és néhány éle meg van súlyozva az $\{a, b\}$ halmazból, ahol a és b két különböző racionális szám. A kérdés, hogy befejezhető-e ez a súlyozás, azaz tudunk-e a G gráfunk nem súlyozott éleihez olyan súlyokat rendelni az $\{a, b\}$ halmazból, hogy az így kapott $\{a, b\}$-élsúlyozás megengedett legyen.*

A [5]-es beszámolóban beláttuk a következő tételt:

2.3. Tétel. *A 2.2. Feladat polinom időben megoldható fákon tetszőleges a, b esetén.*



1. ábra. A 2.4. Tétel bizonyításában szereplő konstrukció illusztrációja.

Most megmutatjuk, hogy az antifaktor probléma fákon megoldható a 2.3. Tétel bizonyításában adott dinamikus programozási algoritmussal, amivel adunk egy alternatív polinomiális algoritmust az antifaktor probléma megoldására.

2.4. Tétel. *Az antifaktor probléma fákon polinomiálisan visszavezethető annak az eldöntésére, hogy létezik-e egy adott fának megengedett* $\{0, 1\}$*-élsúlyozása.*

Bizonyítás. Vegyük az antifaktor problémának egy példányát, azaz adva van egy G fa és egy $f : V \to \mathbb{Z}_+$ függvény. Ezek alapján polinom időben konstruálni fogunk egy G' gráfot, melynek akkor és csak akkor lesz megengedett $\{0, 1\}$-élsúlyozása, ha az antifaktor probléma adott példánya megoldható.

Kezdjük a G' gráf konstrukciójával. A G gráf egy másolatából indulunk ki, majd minden $v_i \in V$-re módosítsuk G'-t a következőképpen. Adjunk hozzá új iM darab v_i-be csatlakozó levelet, ahol $M = n+1$. Legyen D_i ezen újonnan hozzáadott csúcsok halmaza. Legyen u egy csúcs D_i-ből és adjunk hozzá a gráfhoz $(iM + f(v_i) - 1)$ darab új kettő hosszú utat, melyeket csatlakoztassunk be u-ba, majd minden D_i-beli csúcshoz, kivéve az u-t, adjunk hozzá egy levél élt. A 1. ábrán megtekinthetjük a konstrukciót egy adott v_i-re. Nyilvánvalóan G' előállítható polinomiális időben. A bizonyítás hátra levő részében belátjuk, hogy az antifaktor probléma pontosan akkor oldható meg G-re és f-re, ha G'-nek létezik megengedett $\{0, 1\}$-élsúlyozása.

Először tegyük fel, hogy az antifaktor feladat megoldható G-re és f-re, és az eredmény részgráfot jelöljük H-val. Ahhoz, hogy megkapjuk G'-nek egy megengedett $\{0, 1\}$-élsúlyozását, állítsuk be G' éleinek súlyát a következőképpen: ha az e él része H-nak akkor $w(e)$ legyen 1, egyébként legyen 0. Súlyozzuk az összes maradandó élt — melyek pontosan az újonnan G'-höz adott élek — eggyel. Vizsgáljuk meg, hogy az így kapott $\{0, 1\}$-élsúlyozás valóban megengedett.

Az G' gráf G-beli csúcsai között nem lehet ütközés, mert az indukált címkéje bármely két ilyen csúcsnak különböző: Mivel az újonnan hozzáadott élek hozzájárulása a címkéhez iM és az eredeti éleké pedig legfeljebb $n-1$, a v_i csúcs legnagyobb lehetséges címkéje $U_i = iM + n - 1 = (i+1)n + i - 1$. Ezzel szemben a v_{i+1} csúcs legkisebb lehetséges címkéje $L_{i+1} = (i+1)M = (i+1)n+i+1$, mely hasonló érveléssel megmutatható. Láthatjuk, hogy mindkét esetben L_{i+1} szigorúan nagyobb, mint U_i, azaz $z_{v_{i+1}} > z_{v_i}$ minden megengedett $\{0,1\}$-élsúlyozásban minden $i \in \{1,\dots,n-1\}$-re, mivel $L_{i+1} \leq z_{v_{i+1}}$ és $U_i \geq z_{v_i}$.

Figyeljük meg, hogy a v_i és a D_i-ben lévő csúcsok között nem lesz ütközés, mivel H egy megengedett megoldása volt az antifaktor problémának, így a v_i-re illeszkedő 1 súlyú élek száma az eredeti gráfban G-ben nem egyezhet meg $f(v_i)$-vel a súlyok beállítása miatt. A többi újonnan hozzáadott maradék él mentén szintén nem lesz ütközés, ami nagyon könnyen átgondolható.

Másodjára tegyük fel, hogy G'-nek létezik megengedett $\{0,1\}$-élsúlyozása. Ekkor legyen H a G gráf 1 súlyú éleiből álló részgráfja. Most megmutatjuk, hogy H egy megengedett megoldása az antifaktor problémának G-re és f-re. Minden v_i és $D_i \setminus \{u\}$ között menő e él súlyának muszáj 1-nek lennie, mivel minden $D_i \setminus \{u\}$-beli csúcsra illeszkedik egy levél él, amely mentén ütközés alakulna ki, ha az él súlya 0 lenne. Hasonló érveléssel minden $\Delta(u) \setminus \{v_iu\}$-beli élnek is 1 súlyúnak kell lennie. Ezek alapján $z(v_i) = d_H(v_i) + |D_i \setminus \{u\}| + w(v_iu) = d_H(v_i) + iM - 1 + w(v_iu)$ és $z(u) = |\Delta(u) \setminus \{v_iu\}| + w(v_iu) = iM + f(v_i) - 1 + w(v_iu)$, és a $z(v_i) \neq z(u)$-ből következik, hogy $d_H(v_i) \neq f(v_i)$, amivel beláttuk a tételt. □

Az előző tétel bizonyításában megkonstruált fa pontosan akkor 0-1 tulajdonságú, ha a hozzá tartozó antifaktor feladatnak van megoldása. Ebből következik, hogy az antifaktor feladat megoldhatóságára vonatkozó Lovász-féle karakterizáció [6] egyben jól jellemzi a bizonyítás szerinti 0-1 tulajdonságú fákat is. Nyitott kérdés, hogy ez a karakterizáció kiterjeszthető-e tetszőleges fákra.

3. Címkézett fák kompakt reprezentációja

Ebben a fejezetben áttérünk az új témámra, a címkézett fák kompakt reprezentációjára. Vizsgáljuk a következő feladatot:

3.1. Feladat. *Legyenek $A_1, A_2, \dots, A_n$ kevés elemből álló halmazok. Vegyük ezek Descartes-szorzatának egy részhalmazát, amelyet általában valamilyen szabályrendszer definiál. A cél az, hogy ezen kombinációkat a lehető leghatékonyabban kódoljuk el úgy, hogy azokkal hatékonyan végezhessünk bizonyos műveleteket, például a reprezentált kombinációk módosítását vagy a köztük való keresést.*

A fejezet további részében először is bemutatjuk a feladat motivációját. Ezután bemutatjuk, hogy milyen reprezentációk a legelterjetebbek az irodalomban, és vá-

zolunk néhány új, az irodalomban nem megtalálható reprezentációt a 3.1. Feladatra. Ezt követően ismertetünk néhány nehézségi eredményt, és bemutatjuk a fontosabb megoldatlan kérdéseket. Végezetül figyelmünket arra fordítjuk, hogy egy sudoku megoldásai hogyan tömöríthetőek össze minél jobban.

3.1. Motiváció

A kutatást ebben a témában egy hétköznapi probléma ihlette, amelyet bemutatok ebben az alfejezetben. Autók előállításához tömérdek dolgot kell specifikálni. Például gyártás előtt meg kell adni hány lóerős motor legyen benne, kerekei méretet, a színét, vagy, hogy klímával felszerelt legyen-e. Ezenfelül vannak kritériumok, mint például, hogy egy erős 150 lóerős kocsi nem állítható elő a rendelkezésre álló legkisebb kerékkel. A valóéletben néhény tulajdonság megengedett kombinációit táblázatokban tárolják. Egy teljes beállítást pedig akkor nevezünk jónak, ha minden táblázatra megszorítva megengedett. Tehát ezen kombináció táblázatok eltárolása adta a motivációt.

Tehát a 3.1. Feladattal összhangba hozva legyen n a tulajdonságok száma. Ekkor legyenek $A_1, A_2, \ldots, A_n$ a tulajdonság halmazok, azaz mindegyik halmaz feleljen meg egy tulajdonságnak, mint például a fenti példával élve, lesz i index, hogy az A_i halmazban az összes lehetséges kerék méret van. Ekkor minden halmazból kiválasztva egy elemet, vagy egy érvényes, autóként előállítható kombinációját kapjuk a tulajdonságoknak, vagy nem. Az érvényes kombinációk az $A_1, \ldots, A_n$ halmazok Descartes-szorzatának egy részhalmazát alkotják. Tehát ha ezen jó kombinációkat szeretnénk elkódolni, akkor visszakapjuk a bevezetésben bemutatott problémát.

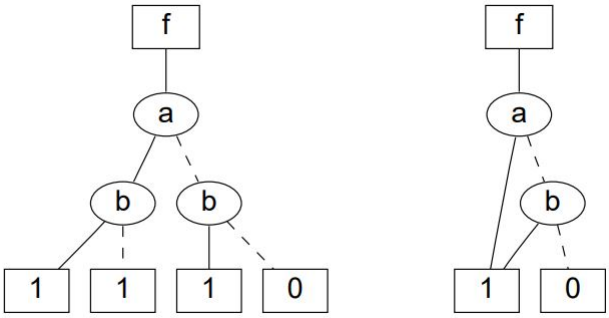
3.2. Reprezentációk a feladatra

Ebben az alfejezetben bemutatunk többféle — az irodalomban is használt, és általunk javasolt — reprezentációt, amellyel modellezhető a 3.1. Feladat.

3.2.1. Decision diagramok

Először azt a megközelítést mutatjuk be, melyhez tartozó adatstruktúrának a legnagyobb az irodalma, az úgynevezett Decision Diagramokat [9] (röviden DD).

A digitális rendszerek tervezése és verifikációja során számos feladat hatalmas logikai formulákkal felírható problémákra vezet. A Binary Decesion Diagramok (BDD) ilyen formulák megoldásainak a halmazának a reprezentációjára és manipulálására az egyik legelterjedtebb megközelítés. A BDD-okat speciálisan Boole formulák, és azzal felírható feladatok reprezentálására találták ki. A BDD-ok gyakorlatilag redukált bináris döntési fák, ami alatt azt értjük, hogy a feladathoz tartozó bináris döntési



2. ábra. $f = (a \vee b)$-hez tartozó döntési fa, és BDD

fából kaphatóak meg az úgynevezett *redukciós* eljárással, amit a következőkben bemutatunk.

Az algoritmus a következő három szabály egymás utáni alkalmazásából áll a döntési fára, egészen addig, amíg egyik szabály sem alkalmazható már:

1. Ugyanolyan címkével (0 vagy 1) rendelkező levél csúcsok összevonhatóak.
2. Két belső csúcs (nem gyökér, vagy levél), melyek gyerekei megegyeznek összevonhatóak.
3. Ha egy belső csúcsnak egyetlen gyereke van, akkor kitöröljük a gráfból és a szüleiből egyből a gyerekeibe irányítjuk az éleket.

A 2. ábrán megtekinthetjük az $f = (a \vee b)$ logikai függvényhez tartozó bináris döntési fát bal oldalon, míg a hozzá tartozó BDD-ot jobb oldalt.

A megoldásai a reprezentált logikai függvénynek a következő módon olvashatóak ki a BDD-ból: tekintsük a gyökérből az 1-es terminálba vezető utakat, és ha szaggatott élen megyünk át, akkor a változó értékét 0-ra, míg ha telin, akkor a változó értékét 1-re állítva egy megoldását kapjuk a függvénynek. Lehetnek olyan utak, melyek nem tartalmaznak minden szintről csúcsot, ebben az esetben a kimaradó szintekhez tartozó változók értékét tetszőlegesen megválaszthatjuk. De fordítva is igaz, hogyha adva van a változóknak egy kiértékelése, akkor a BDD-on végig menve megkapjuk, hogy így igaz vagy hamis lesz a logikai függvény értéke.

A feladatunk reprezentálásához a BDD-ok egy általánosítására, a Multi Decision Diagramokra (MDD) lesz szükségünk. Az MDD-okban minden változó értékkészlete legalább két elemű, viszont a későbbiekben egy ennél is általánosabb adatszerkezetre lesz szükségünk, mivel tulajdonság halmazaink nem azonos méretűek. De ha minden halmazt kiegészítünk segédváltozókkal egy jó reprezentációt kaphatunk, melyben minden szintet megfeleltethetünk annak, hogy melyik tulajdonság halmazból választunk ki éppen elemet.

A 3.4. Fejezetben bemutatjuk, hogy egy gyakorlati példában miképp működtek a Decision Diagrammok.

3.2.2. DAG reprezentáció

A feladatunkra tekinthetünk úgy is, hogy egy DAG reprezentálja a jó kombinációkat, mégpedig úgy, hogy a jó kombinációk pontosan az n hosszú utak. A következő módon egy jó DAG reprezentációt kaphatunk: Rögzítsük az n szint mindegyikére, hogy melyik tulajdonság halmaznak fog megfelelni. Az egyszerűség kedvéért tegyük fel, hogy az i. szinthez tartozó tulajdonság halmaz A_i. Tegyük fel, hogy az i. szintig már előállítottuk a reprezentációt. Ekkor az i. szint összes csúcsára vegyünk fel $|A_i|$ új élt, majd az összes v csúcson végig menve töröljük ki azokat az éleket, amelyekhez tartozó A_i érték kombinálva a v-be vezető út éleinek megfelelő értékekkel olyan kombinációt adnak, mely nem része egyik jó kombinációnak sem.

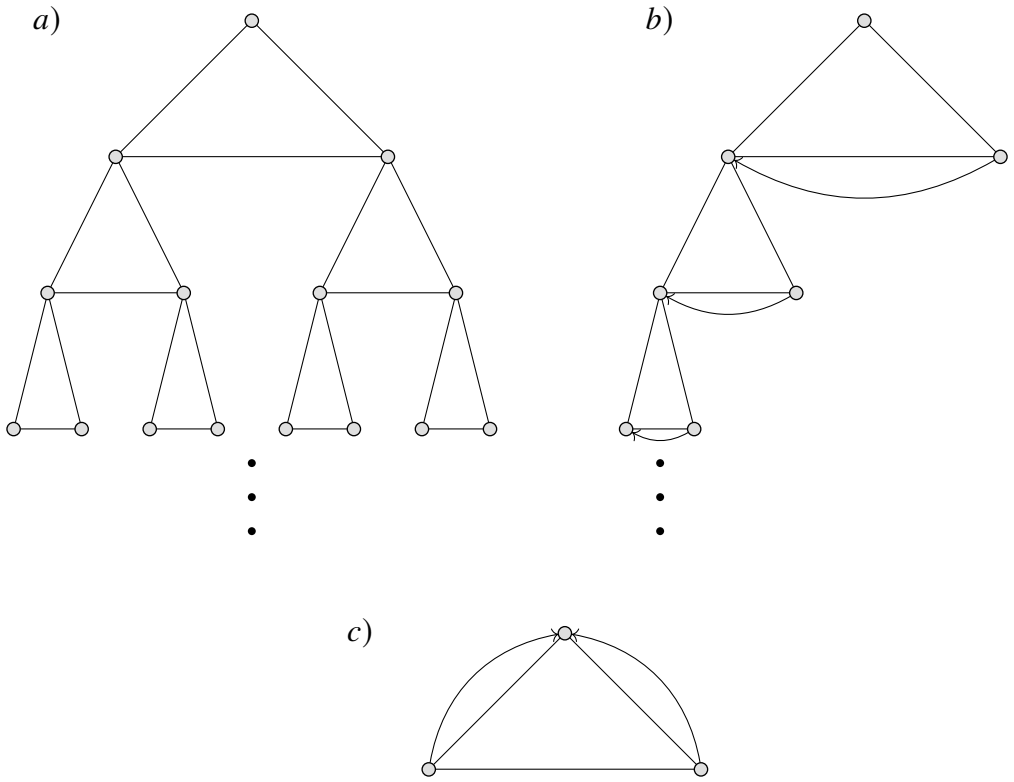
Az így kapott fa természetesen jó reprezentációja a jó kombinációknak, de a mi célunk minél tömörebb reprezentáció előállítása. Ebből a fából az identikus részfákat egy hash-elős algoritmussal azonosítva, majd ezeket egyesítve egy tömör reprezentáció kapható [10]. Ezen algoritmus működését a 3.4. Fejezetben bemutatjuk egy példán keresztül.

3.2.3. Séta reprezentáció

A 3.2.2. Fejezetben láttuk, hogy a DAG reprezentációban egy n hosszú gyökér-levél út felel meg egy konkrét kombinációnak. Ezen reprezentáció megismerése után felmerült bennünk a kérdés, hogy milyen eredmények érhetőek el, hogyha egy olyan reprezentációt szeretnénk előállítani, ahol az n hosszú utak helyett pontosan az n hosszú séták felelnek meg a jó kombinációknak. A 3. ábra segítségével adunk egy konstrukciót, melyből kiderül, hogy létezik olyan D_n gráfsorozat, melyhez tartozó DAG reprezentáció mérete n-szerese egy séta reprezentációjának, azaz, ha n tart a végtelenbe, akkor akármekkora lehet a két reprezentáció méretének az aránya.

Tekintsük a 3. ábrát. Az a) ábrán illusztráljuk a kiindulási D_n irányított aciklikus gráfot, mely álljon egy adott H gráf $1+2+4+\ldots+2^n = 2^{n+1}-1$ darab másolatából az ábrán bemutatott módon, ahol a háromszögek reprezentálják a H gráf másolatait. Emellett ezen H rendelkezzen azzal a tulajdonsággal, hogy a gyökeréből egyetlen egy él megy lefelé. Továbbá a H gráfok legyenek olyan DAG-ok, hogy tovább már nem tömöríthetőek az előző fejezetben említett hash-elős algoritmussal. Ekkor D_n tömörítése a DAG reprezentációban a b) esetben látható G_1 gráf lesz, ahol n egymás alatti H gráf lesz és az irányított, hajlított uv élek annak felelnek meg, hogy a reprezentációban az u csúcs azonosítva van a v csúccsal. Ezzel szemben D_n tömörítése a séta reprezentációban, G_2, bármely n-re úgy fog kinézni ahogyan a c) ábrán láthatjuk.

Tehát láthatjuk, hogy $|V(G_1)| \leq (|V(H)|-2)n$, míg $|V(G_2)| = |V(H)|-2$ minden n-re. Könnyen átgondolható, hogy az élek számánál is hasonlót kapunk, tehát tetszőleges nagy is lehet a két reprezentáció méreténekaz aránya, ezzel motiválva ezen



3. ábra. Az a) ábrán látható a D_n gráf sematikus ábrája, a b) ábrán a hozzá tartozó DAG sematikus ábrázolása, míg a c) ábrán a hozzá tartozó séta reprezentáció. Minden háromszög a H gráf egy másolata.

reprezentáció vizsgálatát.

Ezen reprezentáció tanulmányozását is irodalomkutatással kezdtük. Konkrétan a séta reprezentációval még nem foglalkozott az irodalom, viszont jelentek meg cikkek, melyek a fa tömörítést úgynevezett „top tree"-k segítségével csinálták [11, 12]. A top tree-k gyökereztetett, címkézett, bináris fák, melynek csúcsai úgynevezett „clustereket" reprezentálnak. Ezen clusterek pedig az eredeti fa néhány részfájának feleltethetőek meg. Megismerve ezen munkákat arra jutottunk, hogy ezekben a cikkekben vizsgált módszer alapjaiban véve hasonló, de összeségében más, és érdekes feladatnak tűnik ezen séta reprezentáció megértése is.

3.2. Megjegyzés. Sok hasonló érdekes reprezentáció is szóba jöhetne még, egy ilyen példa a párosítás reprezentáció, ahol a jó kombinációknak pontosan a teljes párosítások felelnek meg. A jövőben érdemes lehet ezen reprezentációk szélesebbkörű vizsgálata és összehasonlítása a fejezettben látottakhoz hasonlóan.

3.3. Nehézségi problémák

Egy logikai függvényhez tartozó BDD előállításakor az egyetlen szabadságunk ami van, az a változók sorrendje. Amint a 3.4. Fejezetben is látni fogjuk az, hogy melyik szintre melyik változót rakjuk a BDD előállításakor lényegesen változtatni tudja a BDD méretét. Nehézségi eredmények a témakörben a BDD-ok minimális méretével foglalkoznak. Első eredmények között [13]-ban belátták, hogy a döntési változata a minimális méretű BDD keresésének NP-teljes. Ennek ellenére találhatóak egzakt megoldási módszerek is az irodalomban [14, 15, 16]. Ezen (és további) nehézségi eredmények indokolják nem egzakt algoritmusok alkalmazását a témában. A legtöbb heurisztikus algoritmus lokális keresést hajt végre [17, 18].

Térjünk át approximálhatósági feladatok bemutatására. A [19]-es cikkben belátják, hogy tetszőleges $c > 1$ konstansra, ha létezne c-közelítő polinomiális algoritmus a minimális méretű BDD feladatra, akkor $P = NP$ teljesülne. Approximálhatósági irányban, legjobb tudásunk szerint még megoldatlan feladat, hogy létezik-e nem konstans approximációs faktor. Sejtésünk szerint a következő állítás is igaz lehet: Minden $\epsilon > 0$-ra NP-nehéz $n^{1-\epsilon}$-on belül közelíteni a problémát, ahol n a változók száma. Ezen sejtés bebizonyítását először a max klikk nem közelíthetési problémáról próbálhatjuk visszavezetni.

3.4. Sudoku megoldásainak tömör reprezentációja

A félév során szerettünk volna az elméleti fa tömörítési eredményeket gyakorlatban is tesztelni. Ennek érdekében kiválasztottuk azt a feladatot, hogy vegyünk egy sudoku táblát, és kódóljuk el minél tömörebben a megoldásait. Ezt a feladatot annak a reményében választottuk, hogy az ebben a problémában elért eredményeket a későbbiekben is fel tudjuk használni, valóélet-beli táblázatok tömörítésénél.

Első lépésként szükségünk volt egy olyan sudoku solverre, amely egy megkezdett sudoku táblához (lehet üres is) fel tudja sorolni az összes lehetséges megoldást. Ez egy backtracking algoritmussal a következőképpen oldható meg: Legyen megadva a cellák feldolgozási sorrendje; T. Az i. szinten az $T[i]$ cellát dolgozzuk fel: a lehetséges értékek mindegyikére kipróbáljuk, hogy azt beírva megoldható-e, azaz beírjuk az értéket, és meghívjuk a rekurziót a következő szintre. Így ezen algoritmussal el tudjuk érni, hogy meghatározza az adott input sudoku összes megoldását. Ezt az algoritmust tömören azért írtuk le, mert a következőkben bemutatjuk, hogy hogyan alakítható át úgy, hogy ne csak az összes megoldást sorolja fel, hanem a futása végén az összes megoldás DAG reprezentációját adja vissza.

Legyen S egy új szótár, melynek kulcsai halmazok, amelyekben tuple-ök vannak, az értékek pedig egészek. A szótár minden eleme a DAG reprezentáció egy csúcsa lesz. Az értékek felelnek meg annak, hogy az adott csúcsban hányas sorszámú részfa gyökerezik, a kulcsok pedig, hogy a gyerekei milyen sorszámú részfák (pontosabban

az algoritmus leírása során mutatjuk be).

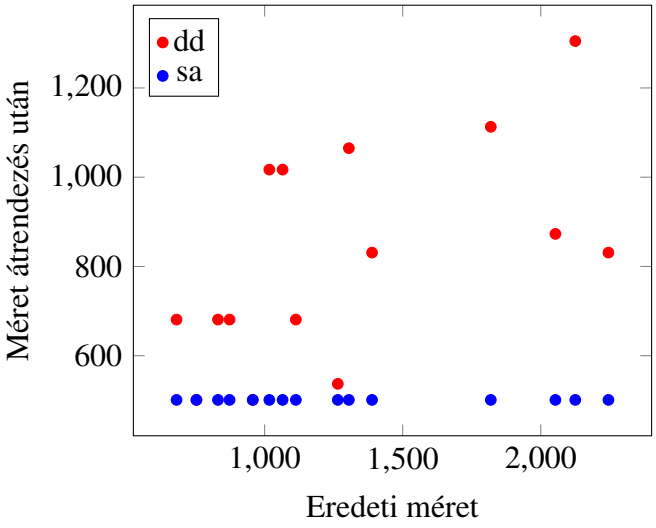
A tömörített DAG reprezentációnkat a backtracking során építjük fel a következőképpen. Tegyük fel, hogy a backtracking fa egy v csúcsában, v-ben vagyunk, és a backtracking most fejezte be a bejárását a gyerekeiben gyökerező részfáknak, azaz most lépett vissza az utolsó gyerekéből is. Ekkor indukció szerint tudjuk, hogy a gyerekekbe milyen részfák gyökereznek, legyenek a gyerekek $v_1, \ldots, v_k$, az értékek amelyekkel a gyerekekbe léptünk rendre $e_1, \ldots, e_k$, és az alattuk gyökerező részfák, pedig rendre $T_1, \ldots, T_k$. Legyen $w := \{(T_1, e_1), (T_2, e_2), \ldots, (T_k, e_k)\}$. Ekkor két lehetőség van: 1) $w \in S$, 2) $w \notin S$. Az első esetben olvassuk ki S-ből w értékét, és azzal térjünk vissza, míg a másodikban, bővítsük S-et a w kulccsal, és a hozzá tartozó értékkel térjünk vissza, amely pedig legyen az eddigi legnagyobb részfa sorszáma plusz egy.

Tehát ezen backtracking algoritmus a futása során építi az S szótárat, amelyből a legvégén kiolvasható a DAG reprezentáció. Mivel a szülő-gyerek kapcsolatok miatt, amelyek a kulcsokban vannak eltárolva szépen sorban felépíthető a DAG. Viszont a szótár struktúrájából adódóan nem egy átlátható reprezentációt kapunk, ezért ezen felül az algoritmus során egy átlátható reprezentációt is építeni kell. Egy 3D tömb megfelel ezen reprezentációnak, mivel ennek azt kell tudnia, hogy az első index szerinti i. helyen azon csúcsokat kell tartalmaznia, amellyek i távolságra vannak a gyökértől az S szótár szerint.

Ezen reprezentáció kitalálásánál a legfontosabb dolog amire ügyelni kellett, hogy ha cserélni akarunk a változók sorrendjén, akkor az új reprezentáció könnyen számolható legyen. Ebben a reprezentációban két szomszédos szint cseréje megoldható $O(n^2)$ időben, de ezen beszámoló keretibe ennek a bemutatása nem fér bele [20].

Miután megvan a reprezentáció, figyelmünket a változók sorrendjének a meghatározására fordíthatjuk. Tehát a feladatunk, hogy kapunk egy sudokut, határozzuk meg a megoldásait, és reprezentáljuk a lehető legtömörebb módon. Jelen esetben 4x4-es Sudokukat vizsgáltunk. Első lépésként implementáltunk egy hasonló eljárást a Sifting algoritmushoz [21], amely az irodalomban a legtöbbet használt átrendezési algoritmus. Majd ezután egy saját, szimulált lehűlésen alapuló heurisztikus algoritmust is implementáltunk. Ezen két algoritmus összevetését a 4. grafikonon tekinthetjük meg. Ebben a példában vettük a teljesen üres sudoku megoldásait, és különböző változó sorrendekhez tartozó DAG reprezentációjuk méreteit vizsgáltuk, átrendezés előtt, és után. A vízszintes tengelyen szemléltetjük a reprezentáció eredeti méretét átrendezés előtt, míg a függőlegesen átrendezés után. A kék pontok felelnek meg a szimulált lehűléses algoritmusnak, míg a pirosak a Sifting féle algoritmusnak

Amint láthatjuk, a szimulált lehűlésen alapuló algoritmusunk minden esetben 501 méretű reprezentációt talált, míg a Sifting algoritmusához hasonló heurisztika csak kevés esetben találta meg, ezt a futtatásaink közötti legjobb reprezentációt.



4. ábra

Ahhoz, hogy a DD nevű python könyvtárban szereplő state-of-the-art átrendezési algoritmussal vethessük össze algoritmusunkat, át kellett térnünk a BDD féle reprezentációra. Ehhez modelleztük a feladatot logikai formulák segítségével is, majd az algoritmusokat is átalakítottuk ennek megfelelően. A 5. ábrában található táblázatokban az így kapott futási eredményeket szemléltetjük.

Amint láthatjuk, kisebb méretű BDD-ok esetén nincsen nagy eltérés az átrendezések között, de ahogy növeljük a BDD méretét, a mi heurisztikánk egyre jobban teljesít általában a Sifting féle heurisztikával szemben.

Futási időket nem tárgyalunk ebben a beszámolóban, mert a reprezentációnkból még hiányzik a hatékony szint csere implementációja, ezért heurisztikáink futási idejét sokat lassítja, hogy minden egyes cserénél újra futtatjuk a backtracking algoritmust az új sorrendhez tartozó reprezentáció méretének meghatározásához.

4. Terv a következő félévre

Az MSc-s diplomamunkámat a címkézett fák kompakt reprezentációjából fogom írni, így ebben a fejezetben bemutatom, hogy a következő félévre milyen terveink vannak. Egyrészt a 3.2.3. Fejezetben bemutatott reprezentációk vizsgálata érdekes kutatási irány lehetne. Másrészt a 3.3. Fejezetben említett nem approximálhatósági problémával is szeretnénk foglalkozni, miszerint igaz-e, hogy minden $\epsilon > 0$-ra NP-nehéz $n^{1-\epsilon}$-on belül közelíteni a minimális BDD méretét, ahol n a változók száma. Továbbá szeretnénk a 3.4. Fejezetben bemutatott tömörítési, átrendezési algoritmusokat tovább fejleszteni, valóélet-beli adathalmazokon is tesztelni.

Eredeti	BDD	SA
105	79	81
336	196	198
336	196	219
336	196	198
336	196	186
336	196	225
336	196	192
1,336	843	879
1,336	843	880
1,336	843	876
1,336	843	873
1,336	843	935
1,515	933	889
1,515	936	901
2,496	1,241	1,148

Eredeti	BDD	SA
2,496	1,225	1,222
2,496	1,260	1,183
2,496	1,330	1,103
2,496	1,284	1,368
2,496	1,196	1,270
4,340	2,325	2,274
4,340	2,220	2,252
5,991	4,017	4,112
5,991	3,958	2,765
5,991	3,978	3,290
5,991	3,911	4,112
5,991	3,938	3,904
5,991	4,032	3,069
5,991	3,973	4,007
5,991	4,094	2,923

5. ábra. A szimulált lehűlésen alapuló heurisztikus átrendezési algoritmus összeveztve a DD könyvtárban található átrendezési algoritmussal.

Hivatkozások

[1] Henrik Reif Andersen, Tarik Hadzic, and David Pisinger. Interactive cost configuration over decision diagrams. *Journal of Artificial Intelligence Research*, 37:99–139, 2010.

[2] Fabio Somenzi. Cudd: Cu decision diagram package release 2.5. 0. *University of Colorado at Boulder*, 2012.

[3] Python dd package. `https://github.com/tulip-control/dd`. Accessed: 2022-12-19.

[4] 12th japanese-hungarian symposium on discrete mathematics and its applications march 21-24, 2023 in budapest, hungary. `http://cs.bme.hu/~jh2023/`.

[5] Péter Madarasi and Máté Simon. On vertex-coloring {a, b}-edge-weightings of graphs. `egres.elte.hu`, *TR-2022-06, Egerváry Research Group, Budapest*, 2022.

[6] László Lovász. Antifactors of graphs. *Periodica Mathematica Hungarica*, 4(2-3):121–123, 1973.

[7] Gérard Cornuéjols. General factors of graphs. *J. Comb. Theory Ser. B*, 45(2):185–198, 1988.

[8] András Frank, Lap Chi Lau, and Jácint Szabó. A note on degree-constrained subgraphs. *Discrete Mathematics*, 308(12):2647–2648, 2008.

[9] Sheldon B. Akers. Binary decision diagrams. *IEEE Transactions on computers*, 27(06):509–516, 1978.

[10] Sherif Sakr. Xml compression techniques: A survey and comparison. *Journal of Computer and System Sciences*, 75(5):303–322, 2009.

[11] Philip Bille, Inge Li Gørtz, Gad M Landau, and Oren Weimann. Tree compression with top trees. *Information and Computation*, 243:166–177, 2015.

[12] Lorenz Hübschle-Schneider and Rajeev Raman. Tree compression with top trees revisited. In *International Symposium on Experimental Algorithms*, pages 15–27. Springer, 2015.

[13] Beate Bollig and Ingo Wegener. Improving the variable ordering of obdds is np-complete. *IEEE Transactions on computers*, 45(9):993–1002, 1996.

[14] Rolf Drechsler, Nicole Drechsler, and Wolfgang Günther. Fast exact minimization of bdds. In *Proceedings of the 35th Annual Design Automation Conference*, pages 200–205, 1998.

[15] Nagisa Ishiura, Hiroshi Sawada, and Shuzo Yajima. Minimazation of binary decision diagrams based on exchanges of variables. In *ICCAD*, volume 91, pages 472–475, 1991.

[16] She Woong Jeong, Tae Sun Kim, et al. An efficient method for optimal bdd ordering computation. In *ICVC: International Conference on VLSI and CAD*, volume 3, pages 252–256, 1993.

[17] Beate Bollig, Martin Löbbing, and Ingo Wegener. Simulated annealing to improve variable orderings for obdds. In *In Int'l Workshop on Logic Synth*. Citeseer, 1995.

[18] Beate Bollig, Martin Löbbing, and Ingo Wegener. On the effect of local changes in the variable ordering of ordered decision diagrams. *Information Processing Letters*, 59(5):233–239, 1996.

[19] Detlef Sieling. The nonapproximability of obdd minimization. *Information and Computation*, 172(2):103–138, 2002.

[20] Donald Ervin Knuth et al. *The art of computer programming*, volume 3. Addison-Wesley Reading, MA, 1973.

[21] Karl S Brace, Richard L Rudell, and Randal E Bryant. Efficient implementation of a bdd package. In *Proceedings of the 27th ACM/IEEE design automation conference*, pages 40–45, 1991.