



EÖTVÖS LORÁND TUDOMÁNYEGYETEM  
INFORMATIKAI KAR  
ALGORITMUSOK ÉS ALKALMAZÁSAIK TANSZÉK

---

## Hatékony részstruktúra-kereső algoritmusok a kémiai informatika területén

TUDOMÁNYOS DIÁKKÖRI SZAKDOLGOZAT

*Témavezetők:*

**Dr. Tichler Krisztián**

ELTE IK

Algoritmusok és Alkalmazásai Tanszék  
adjunktus

**Kovács Péter**

ChemAxon Kft.

programtervező matematikus

*Készítette:*

**Vigula Mónika**

Alkalmazott matematikus MSc

I. évfolyam

Budapest, 2012

A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg (a támogatás száma TÁMOP 4.2.1./B-09/1/KMR-2010-0003).

MarvinSketch, felhasználása: kémiai struktúrák rajzolása,  
Verzió: 5.11.4, 2012, ChemAxon (<http://www.chemaxon.com>)

JChem Base, felhasználása: kémiai struktúrák keresése, adatbázis-kezelés, Verzió: 5.11.4, 2012, ChemAxon (<http://www.chemaxon.com>)



## Köszönetnyilvánítás

Köszönöm dr. Fekete István Docens Úrnak, hogy megkedveltette velem az algoritmusok izgalmas érdekes világát, a témafelvetést és a lehetőséget, hogy dolgozatom kutatási projekt keretében készíthettem el.

Köszönetet szeretnék mondani témavezetőimnek, dr. Tichler Krisztiánnak és Kovács Péternek. Köszönöm dr. Tichler Krisztiánnak az értékes szakmai és stilisztikai tanácsait, melyek a szakirodalom megértése és dolgozatom elkészítése során nélkülözhetetlenek voltak. Köszönöm Kovács Péternek, hogy megismerttetett a kémiai informatika alapjaival, és a témérdek információt, melyet az implementáció és a dolgozat készítése során nyújtott. Köszönöm továbbá, hogy a felmerült kérdésekkel mindig fordulhattam hozzá.

Hálával tartozom továbbá a ChemAxon Kft. minden kedves dolgozójának, hogy munkámat építő jellegű kritikákkal illették.

Végül, de nem utolsó sorban szeretnék köszönetet mondani Családomnak és Barátaimnak a tanulmányaim során nyújtott biztatásért és támogatásért.

# Tartalomjegyzék

|                                                             |           |
|-------------------------------------------------------------|-----------|
| <b>Bevezetés</b>                                            | <b>1</b>  |
| <b>1 Elméleti háttér</b>                                    | <b>4</b>  |
| 1.1 A részgráf-izomorfia probléma . . . . .                 | 4         |
| 1.2 A SMILES formátum . . . . .                             | 6         |
| <b>2 Részstruktúra-kereső algoritmusok összehasonlítása</b> | <b>8</b>  |
| 2.1 Előszűrés . . . . .                                     | 8         |
| 2.2 Részstruktúra-kereső algoritmusok . . . . .             | 10        |
| 2.2.1 Az Ullmann-algoritmus . . . . .                       | 11        |
| 2.2.2 A VF2 algoritmus . . . . .                            | 15        |
| 2.3 Várhatóan optimális atomsorrend meghatározása . . . . . | 21        |
| 2.4 Az algoritmusok implementálása . . . . .                | 26        |
| 2.4.1 Kémiai jelenségek a molekulákban . . . . .            | 29        |
| 2.4.2 Heurisztikák alkalmazása . . . . .                    | 31        |
| <b>3 Összefoglalás</b>                                      | <b>37</b> |
| <b>Irodalomjegyzék</b>                                      | <b>41</b> |

# Bevezetés

"A tudomány izgalmas kaland. Ajtókat nyitogatunk, keressük az igazságot, s egyszerre ott van előttünk, mint mesebeli kincs, a maga kézzelfogható, tündöklő valóságában." Kosztolányi Dezső

A gráfok, mint adatmodellek a természettudományos és a mérnöki kutatások során széles körben alkalmazhatóak. Számos esetben – pl.: gyógyszerkutatás, képfelismerés, számítógépes tanítás – szükséges különböző objektumok hasonlóságának eldöntése. Abban az esetben, ha az egyes objektumok reprezentálására gráfot használunk, akkor a feladatot könnyen visszavezethetjük gráfok közti hasonlóság eldöntésére.

Klasszikus feladat annak eldöntése, hogy két adott gráf szerkezetileg azonos-e [3]. A gráfizomorfizmus probléma biztosan NP-beli, azonban nem ismert, hogy P-beli vagy NP-teljes-e. Néhány gráftípusra léteznek polinomiális algoritmusok, például síkbarajzolható gráfokra [17], fokszámkorlátos gráfokra [25], intervallumgráfokra [24] és permutációgráfokra [11]. Ismert és ingyenesen elérhető gráfizomorfia eldönthető programcsomag a nauty [27] és conauto [23].

Objektumok feldolgozása során szükség lehet két objektum hasonlóságának megállapítására. Ebben az esetben az adott feladattól függően a két gráf legnagyobb közös feszített részgráfját (MCS – Maximum Common (Induced) Subgraph) vagy a két gráf maximális élszámú (MCES – Maximum Common Edge Subgraph) közös részgráfját kell megkeresnünk. Adott két gráf esetén az MCS és MCES meghatározása NP-teljes feladat [16]. Két gráf legnagyobb közös részgráfját meghatározó algoritmust találunk [10]-ben, [12]-ben, [21]-ben és [33]-ban. A kémiai informatika területén két egyszerű molekula hasonlóságának megállapítása során a két molekula által reprezentált gráfok MCES-ét érdemes meghatározni.

Egy másik megközelítés szerint két gráfot hasonlónak tekintünk, ha a gráfok szerkesztési távolsága [38] egy előre megadott értéknél kisebb. Két gráf szerkesztési távolsága alatt azon minimális lépésszámot értjük, mely alatt az egyik gráfból megengedett lépések sorozatával a másik gráfot kapjuk meg. Tipikus megengedett lépés egy csúcs vagy él címkéjének módosítása, egy él vagy csúcs beillesztése/törlése (ter-

mészetesen előtte a rá illeszkedő éleket is törölnünk kell). Két adott gráf szerkesztési távolságának eldöntése szintén NP-teljes feladat.

Vizsgálhatjuk továbbá, hogy két adott gráf,  $G_1$  és  $G_2$  esetén  $G_2$  tartalmazza-e részgráfként  $G_1$ -et [5]. Bizonyos alkalmazások esetén feszített részgráfként történő tartalmazást, míg más esetben a nem feltétlen feszített részgráfként történő tartalmazást érdemes vizsgálni. A részgráf-izomorfia probléma NP-teljes. Részgráf-izomorfiaát eldöntő eljárás a J. R. Ullmann 1976-ben publikált algoritmus [35], L. P. Cordella és mtsai. 2001-es algoritmus, a VF2 [13], valamint a H. Shang és mtsai. által 2008-ban publikált algoritmus, a QuickSI [34].

Dolgozatunkban a részgráf-izomorfia-problémát [5] (mely alatt a nem feltétlen feszített részgráfként történő tartalmazást értjük) speciális gráfokra, ún. molekulagráfokra vizsgáltuk. Az ismertetett eljárások többsége azonban kis módosítással könnyen alkalmazható általános értelemben vett címkézetlen vagy csúcs/élcímkézett gráfokra is.

Egy egyszerű molekulát könnyen ábrázolhatunk gráfként. A gráf csúcsai legyenek az egyes atomok, és a gráf élei pedig legyenek címkézve az atomok közötti kötéseknek megfelelően. Molekulagráfok esetén általában nem foglalkozunk a molekulában szereplő hidrogénekkel és a rájuk illeszkedő kötésekkel, hiszen ezeket kémiai tudásunk alapján meghatározhatjuk [8, 26].

A kémiai informatikában (pl. gyógyszerkutatás során) gyakran merül fel az igény arra, hogy egy adott molekula és molekula-adatbázis esetén adjuk meg azon molekulákat az adatbázisból, melyek részstruktúráként tartalmazzák az adott molekulát. Mivel a részgráf-izomorfia probléma NP-teljes, és az adatbázis gyakran több száz-ezer, esetleg millió molekulából áll, ezért azon megoldás, hogy az adatbázis minden egyes molekulájával végezzük el a részgráf-izomorfia-vizsgálatot a gyakorlatban használhatatlan. Ezért az adatbázison előszűrést végzünk, ezáltal kiszűrve azon molekulákat az adatbázisból, melyeknek biztosan nem részstruktúrája az adott molekula. Csak az ezen eljárás után megmaradt adatbázisbeli molekulákkal végezzük el a költséges részgráf-izomorfia-vizsgálatot.

A kémiai informatikában gyakran alkalmazott módszer, hogy minden egyes, az adott adatbázisban szereplő molekulához tárolunk egy fingerprintet [1]. A fingerprint egy 0-kból és 1-esekből álló, jellemzően 1024 hosszú sorozat, mely az adott molekulát jól jellemzi. A fingerprint minden egyes pozícióján attól függően szerepel 0 vagy 1, hogy a molekula rendelkezik-e az  $i$ . pozíciónak megfelelő tulajdonsággal (például tartalmaz-e benzolgyűrűt). Az adatbázisban szereplő molekulákhoz tartozó fingerprinteket csak egyszer számoljuk ki, és a molekulákkal együtt az adatbázisban tároljuk.

Egy jó kémiai fingerprintre teljesül, hogy ha a keresni kívánt molekula fingerprintjének  $i$ . pozícióján 1 szerepel, akkor az  $i$ -t, részstruktúraként tartalmazó molekula fingerprintjének  $i$ . helyén is 1-esnek kell állnia. Ezen tulajdonságot felhasználva gyorsan, egyszerű bitműveleteket alkalmazva végezhetünk előszűrést az adatbázison. Első lépésben kiszámoljuk a keresni kívánt molekulához tartozó fingerprintet. Ezen és az adatbázisban tárolt fingerprintek, valamint a fent említett tulajdonság alapján kiszűrjük az adatbázisból azon molekulákat, melyek biztosan nem tartalmazhatják részstruktúraként az adott molekulát. A szakirodalomban több, kémiai fingerprintet előállító algoritmust találunk [1, 28, 31]. Megjegyezzük, hogy nem volt célunk a különböző fingerprintekkel történő előszűrések hatékonyságának összehasonlítása, ezért a fingerprintekkel a dolgozatban nem foglalkozunk.

A részgráf-izomorfia-problémával 2010 őszén kezdtem el foglalkozni. Az ELTE TTK Matematika BSc Alkalmazott matematikus szakirányán készített szakdolgozatomban [30] az egyes részstruktúra-kereső és előszűrő algoritmusokat vizsgáltam elméleti szempontból. Az ELTE IK Programtervező informatikus BSc szakirányán készített szakdolgozatomhoz [29] az Ullmann- és a VF2 algoritmus implementációk futási idejét hasonlítottam össze. Az újonnan elért eredményeket jelen dolgozat keretében ismertetem. A kutatás során elkészített implementációk helyességét és hatékonyságát valós molekulagyűjteményeken vizsgáltam.

A dolgozat a következőképpen épül fel. Az 1. fejezetben a definíciókat ismertetjük és a részgráf-izomorfia probléma NP-teljességét bizonyítjuk. A 2. fejezetben először néhány előszűrési módszert, majd két, részgráf-izomorfia eldöntő algoritmust vizsgálunk: az Ullmann-algoritmust [35], majd a VF2-t [13]. Ezt követően egy várhatóan optimális atomsorrendet előállító eljárást [34] tekintjük át. Ezután az algoritmusok implementálása során alkalmazott heurisztikákkal és a mért futásidőkkel foglalkozunk. Eredményeinket a 3. fejezetben foglaljuk össze.

# 1. fejezet

## Elméleti háttér

### 1.1. A részgráf-izomorfia probléma

**1.1. Definíció** Legyen  $G_1 = (V_1, E_1)$  és  $G_2 = (V_2, E_2)$  két adott gráf. A  $G_1$  gráf részgráf-izomorf a  $G_2$  gráffal, ha létezik olyan  $f : V_1 \rightarrow V_2$  injektív függvény, melyre a következő teljesül:  $\forall u \in V_1 \forall v \in V_1 ((u, v) \in E_1 \Rightarrow (f(u), f(v)) \in E_2)$ .

**Jelölés:**  $G_1 \subseteq G_2$

Fontosnak tartjuk hangsúlyozni, hogy a szakirodalomban találunk egy másik, a fentivel nem ekvivalens definíciót a részgráf-izomorfia problémára. Abban az esetben  $G_2$  feszített részgráfként [37] kell, hogy tartalmazza  $G_1$ -et.

Csúcs- és élcímkézett gráfok (például a dolgozatban szereplő molekulagráfok) esetén a részgráf-izomorfia definíciójában további feltételekre van szükség.

**1.2. Definíció** Csúcs- és élcímkézett gráf alatt egy  $G = (V, E, \Sigma_V, \Sigma_E, l_V, l_E)$  hatást értünk, ahol  $V$  és  $E$  a gráf csúcs-, illetve élhalmazát jelöli,  $\Sigma_V$  illetve  $\Sigma_E$  a csúcsok és az élek címkeinek halmaza,  $l_V : V \rightarrow \Sigma_V$  és  $l_E : E \rightarrow \Sigma_E$  pedig a gráf címkézőfüggvényei.

A későbbiekben külön hangsúlyozzuk, ha csak egyik (csúcs- vagy él-) címke van jelen. A továbbiakban címkézett gráfok alatt a csúcs- és élcímkézett gráfokat értjük. Címkézett gráfokra az 1.1. definíció a következőképpen módosul.

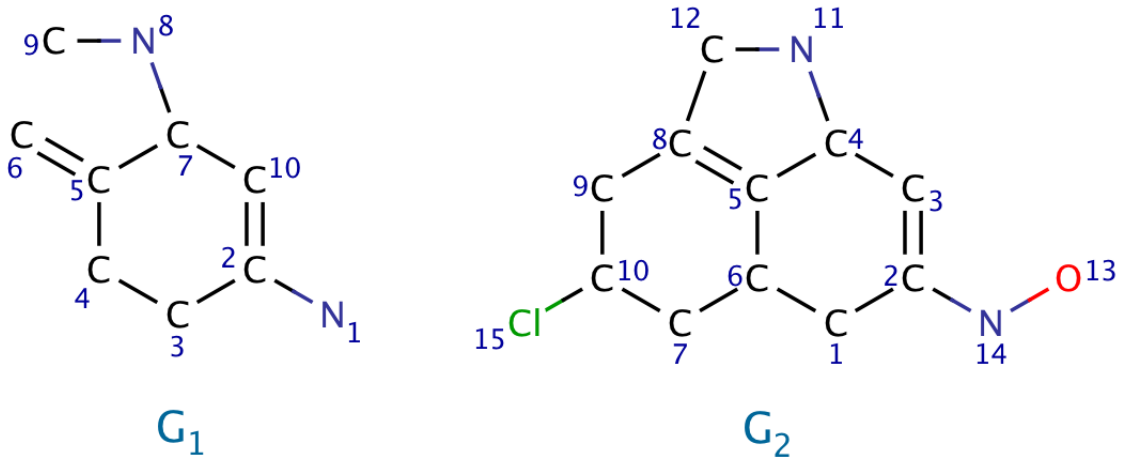
**1.3. Definíció** Legyen adott két címkézett gráf,  $G_1 = (V_1, E_1, \Sigma_{V_1}, \Sigma_{E_1}, l_{V_1}, l_{E_1})$  és  $G_2 = (V_2, E_2, \Sigma_{V_2}, \Sigma_{E_2}, l_{V_2}, l_{E_2})$ .  $G_1$  részgráf-izomorf  $G_2$ -vel, ha létezik olyan  $f : V_1 \rightarrow V_2$  injektív függvény, melyre a következők teljesülnek:

1.  $\forall u \in V_1 (l_{V_1}(u) = l_{V_2}(f(u)))$
2.  $\forall u \in V_1 \forall v \in V_1 ((u, v) \in E_1 \Rightarrow (f(u), f(v)) \in E_2)$

$$3. \forall u \in V_1 \forall v \in V_1 ((u, v) \in E_1 \Rightarrow l_{E_1}((u, v)) = l_{E_2}((f(u), f(v))))$$

A továbbiakban a részgráf-izomorfia definíciója alatt az 1.3. definíciót értjük, míg a definícióban szereplő függvényt részgráf-izomorfizmusnak nevezzük. Az egyes algoritmusok ismertetése során – abban az esetben, ha a szigorúbb, feszített részgráfként történő tartalmazást előíró definíció esetén további feltételnek is teljesülnie kell – megjegyzésként feltüntetjük ezen feltételt is.

### 1.1. Példa



1.1. ábra. Példa a részgráf-izomorfia probléma inputjára

Tekintsük az 1.1. ábrán látható  $G_1$  és  $G_2$  címkézett gráfokat. Egy lehetséges  $f : V_1 \rightarrow V_2$  :

|             |             |             |
|-------------|-------------|-------------|
| $f(1) = 14$ | $f(5) = 5$  | $f(9) = 12$ |
| $f(2) = 2$  | $f(6) = 8$  | $f(10) = 3$ |
| $f(3) = 1$  | $f(7) = 4$  |             |
| $f(4) = 6$  | $f(8) = 11$ |             |

**1.4. Tétel** Legyen adott két címkézett gráf,  $G_1 = (V_1, E_1, \Sigma_{V_1}, \Sigma_{E_1}, l_{V_1}, l_{E_1})$  és  $G_2 = (V_2, E_2, \Sigma_{V_2}, \Sigma_{E_2}, l_{V_2}, l_{E_2})$ . Annak eldöntése, hogy  $G_1 \subseteq G_2$ , NP-teljes feladat.

A tétel bizonyítása előtt bevezetünk néhány további definíciót és közlünk egy tételt is.

**1.5. Definíció** Egy  $G$  gráfban Hamilton-körnek nevezünk egy  $H$  kört, ha  $G$  minden csúcsát pontosan egyszer tartalmazza.

**1.6. Definíció (Hamilton-kör feladat)** *Adott egy  $G$  gráf. Döntsük el, hogy  $G$  tartalmaz-e Hamilton-kört.*

Az 1.4. tétel bizonyításához felhasználjuk az alábbi közismert tételt.

**1.7. Tétel** *A Hamilton-kör feladat NP-teljes feladat.*

*Bizonyítás:* (1.4. tétel) Először megmutatjuk, hogy a részgráf-izomorfia probléma NP-beli: a tanú az  $f : V_1 \rightarrow V_2$  függvény. Polinom időben ellenőrizhető, hogy  $f$  kielégíti-e az 1.3. definícióban szereplő feltételeket.

Ezután a Hamilton-kör feladatot – melyről az 1.7. tétel szerint tudjuk, hogy NP-teljes feladat – polinomiálisan visszavezetjük a részgráf-izomorfia problémára. Legyen  $G$  egy input a Hamilton-kör feladatra,  $C_n$  pedig az  $n$  pontú kör, ahol  $n$  jelöli a  $G$  gráf csúcsszámát. Ekkor  $C_n \subseteq G$  pontosan akkor, ha  $G$  tartalmaz Hamilton-kört. □

Dolgozatunk célja a kémiai gráfok részgráf-izomorfájának vizsgálata. A kémiai informatikában  $G_1$ -t query-nek,  $G_2$ -t pedig target-nek nevezzük, gráf helyett molekula(gráf)ról, csúcs helyett atomról, él helyett kötésről, részgráf helyett pedig részstruktúráról beszélünk. Mivel célunk kémiai gráfok részgráf-izomorfájának eldöntése, ezért a továbbiakban mi is ezen elnevezéseket használjuk.

A kémiai informatikában az egyszerű molekulákat általában SMILES formátumban kódoljuk, melyről a következő alfejezetben egy rövid ismertetőt adunk.

## 1.2. A SMILES formátum

A SMILES (Simplified Molecular-Input Line-Entry System) [4, 36] formátum egy gyakran alkalmazott leírónyelv a molekulák tömör, ASCII karakterekkel történő kódolására. Ezen leírás előnyei:

- tömörség, egyszerűség
- egyetlen sorban leírható a molekula (sok esetben egyszerű inputként kezelni, pl.: command line argumentum egy programnál vagy egy dialógus ablak text input mezőjébe bevihető, nem szükséges fájlt csatolni)
- hordozhatóság
- viszonylag könnyű szerkesztés és interpretálás a vegyészek és a számítógépes programok számára.

A molekulában szereplő atomokat a vegyjelükkel jelöljük (például: C, N, O, Cl). Az egyszeres kémiai kötések nem tüntetjük fel, a kétszeres kémiai kötések az =, háromszoros kötések pedig a # jellel reprezentáljuk. Az egymáshoz csatlakozó atomokat legegyszerűbb esetben a vegyjeleik egymás után írásával reprezentálhatjuk. A molekulában szereplő köröket a következőképpen jegyezhetjük fel. A körben szereplő valamely atomot jelöljük meg egy számmal. Ezen sorszám következő előfordulása esetén tudjuk, hogy a számmal jelölt atom és a legutoljára feldolgozott atom között kötés szerepel. Ezen sorszámot a későbbiekben újra felhasználhatjuk.

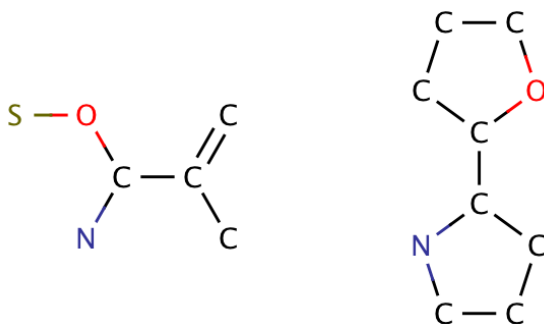
A molekulában szereplő elágazásokat a SMILES formátumban kerek zárójelek között jegyezzük fel. Az elágazásokat egymásba is ágyazhatjuk. Több komponensből álló molekula esetén az egyes komponensek leírását ponttal választjuk el.

A molekula egy tömör SMILES reprezentációja a molekula egy bejárásán alapul. Tekintsük a molekula egy tetszőleges atomját, majd indítsunk innen egy mélységi bejárást. Ezen bejárás során az egyes atomokat és a közöttük lévő kötések a fent említett módon kódoljuk el. Könnyen látható, hogy egy molekula több, ekvivalens SMILES reprezentációval is rendelkezhet.

Sok egyéb információ (pl. hidrogének, aromás gyűrűk) reprezentálására is alkalmas a SMILES formátum [4, 36], de ezeket itt nem tárgyaljuk.

## 1.2. Példa

*Tekintsük az 1.2. ábrán látható általánosított molekulát. Úgy is gondolhatunk molekulák egy halmazára, mint egy több komponensből álló molekulára. A továbbiakban molekula alatt ezen általános értelemben vett molekulát értjük. Ezen molekula két különböző reprezentációja SMILES formátumban: CC(=C)C(N)OS.C1CNC(C1)C1CCC01 és C1CNC(C1)C2OCCC2.SOC(N)C(=C)C.*



1.2. ábra. A reprezentált molekula

## 2. fejezet

# Részstruktúra-kereső algoritmusok összehasonlítása

Az 1. fejezetben beláttuk, hogy a részgráf-izomorfia probléma NP-teljes feladat (1.4. tétel). Emiatt a költséges részgráf-izomorfia eldöntő algoritmusok futtatását általában előszűrés előzi meg. Ezen eljárás feladata, hogy gyorsan ellenőrizhető feltételek segítségével minél több olyan query-target párt kiszűrjön, mely esetén a target molekula nem tartalmazza részstruktúraként a query molekulát.

Az előszűrés után kizárólag a megmaradt molekulagráf-párokra végzünk részgráf-izomorfia vizsgálatot. Gyakran alkalmazott részstruktúra-kereső algoritmus a J. R. Ullmann által 1976-ban publikált algoritmus [35] és az L. P. Cordella és mtsai 2001-es algoritmus [13], a VF2. Szintén részstruktúra-kereső algoritmus a H. Shang és mtsai. által 2008-ban publikált algoritmus, a QuickSI [34]. Ezen algoritmusok implementálása és összehasonlító elemzése jelen dolgozat fő célkitűzése.

A fejezet felépítése a következő. Először az alkalmazott előszűrési módszereket tekintjük át, majd az Ullmann-algoritmussal és a VF2-vel folytatjuk a vizsgáldást. Ezután a QuickSI algoritmus várhatóan optimális atomsorrendet meghatározó részével foglalkozunk. Ezt követően az egyes algoritmusok implementálása során alkalmazott heurisztikákat és a kapott eredményeket ismertetjük.

### 2.1. Előszűrés

Az előszűrés során célunk kiszűrni olyan query-target párokat, melyek esetén a target biztosan nem tartalmazhatja a query molekulát részstruktúraként. Triviális előszűrési feltételek a query és a target atomszámaira és a kötősszámaira vonatkozó egyenlőtlenségek. További könnyen adódó feltétel, hogy a target a különböző rendszámú atomokból legalább annyit tartalmaz, mint amennyi a query-ben meg-

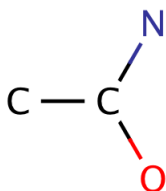
található. Hasonló feltétel mondható el a query és a target molekulában található különböző kötéstípusok számáról is.

Egy további előszűrési módszer a két molekulához egy-egy mátrixot határoz meg. A mátrix sorai a kötéseknek, oszlopai pedig a gyakori atomoknak ( $B, C, N, O, F, P, S, Cl, Se, Br, I$ ) felelnek meg, melyeket az amerikai Nemzeti Rákkutató Intézet [6] nyilvános molekulagyűjteményén készített statisztika alapján választottunk ki. A mátrix kezdetben az azonosan 0 mátrix. A módszer a molekula kötéseit egyesével vizsgálja meg. Amennyiben a kötés végén szereplő mindkét atom gyakori, akkor a kötésnek és az atomoknak megfelelő elemek értékét eggyel megnöveli. Könnyen látható, hogy a részgráf-izomorfia teljesüléséhez a query-hez tartozó mátrix minden eleme legfeljebb akkora lehet, mint a target-hez tartozó mátrix megfelelő eleme.

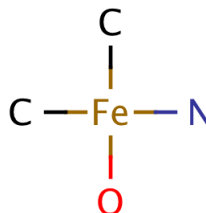
## 2.1. Példa

*Legyenek a query és a target a 2.1. ábrán látható molekulák. Ekkor a következő mátrixot kapjuk. A könnyebb áttekinthetőség érdekében a query-hez és a target-hez tartozó mátrixokat együtt, az egyes elemeket kettősponttal elválasztva közöljük. Fel-tüntettük továbbá a sorok és az oszlopok címkéit, valamint a mátrixok nem nulla elemeit pirossal kiemeltük. Jól látható, hogy targethez tartozó mátrix minden eleme 0, a query-hez tartozó mátrixnak azonban van nem nulla eleme, ezért a target biztosan nem tartalmazza részstruktúráként a query-t.*

|                    | $B$   | $C$          | $N$          | $O$          | $F$   | $P$   | $S$   | $Cl$  | $Se$  | $Br$  | $I$   |
|--------------------|-------|--------------|--------------|--------------|-------|-------|-------|-------|-------|-------|-------|
| <i>egyszeres</i>   | 0 : 0 | <b>4</b> : 0 | <b>1</b> : 0 | <b>1</b> : 0 | 0 : 0 | 0 : 0 | 0 : 0 | 0 : 0 | 0 : 0 | 0 : 0 | 0 : 0 |
| <i>kétszeres</i>   | 0 : 0 | 0 : 0        | 0 : 0        | 0 : 0        | 0 : 0 | 0 : 0 | 0 : 0 | 0 : 0 | 0 : 0 | 0 : 0 | 0 : 0 |
| <i>háromszoros</i> | 0 : 0 | 0 : 0        | 0 : 0        | 0 : 0        | 0 : 0 | 0 : 0 | 0 : 0 | 0 : 0 | 0 : 0 | 0 : 0 | 0 : 0 |



query



target

2.1. ábra. Példa az előszűrés inputjára

A gyakorlatban az előszűrés során a bevezetőben is említett molekulaleíró fingerprinteket [1, 28, 31] is alkalmaznak. Nem volt célunk fingerprintet készítő eljárások implementálása, ezért csak a fent említett módszereket alkalmazva végeztünk előszűrést méréseink során.

## 2.2. Részstruktúra-kereső algoritmusok

A következőkben a két leggyakrabban alkalmazott részstruktúra-kereső algoritmust, az Ullmann-algoritmust [35] és a VF2-t [13] elemezzük. Mindkét esetben elmondható, hogy a szerzők az algoritmust a publikált cikkben címkézetlen gráfokra, egyetlen, a szigorúbb definíciónak (feszített részgráfként történő tartalmazás) megfelelő részgráf-izomorfizmus megkeresésére ismertették. Továbbá a VF2 algoritmus esetén a szerzők a problémát irányított gráfokra vizsgálták. Ezen algoritmusokat módosítottuk úgy, hogy az 1.3. definíció szerinti részgráf-izomorfizmusok számát kapjuk meg az adott query és target molekula esetén.

Amennyiben  $G_1$  részgráf-izomorf  $G_2$ -vel, akkor sok esetben több különböző részgráf-izomorfizmus is megadható közöttük (pl. szimmetrikus struktúrák esetén). Bizonyos alkalmazások esetén elegendő eldönteni, hogy a target molekula tartalmazza-e részstruktúraként a query molekulát (pl. azon adatbázisbeli molekulákat szeretnénk meghatározni, melyek tartalmaznak egy adott funkciós csoportot [2]), azonban számos vizsgálat során (pl. reakciók elemzése esetén) szükséges az összes leképezés meghatározása.

A dolgozatunkban bemutatott részstruktúra-kereső algoritmusok esetén a visszatérési érték nem az összes, 1.3. definíciónak eleget tevő függvény, hanem ezen függvények száma. Abban az esetben, ha a target nem tartalmazza részstruktúraként a query molekulát, ezen érték természetesen 0. Az ismertetett algoritmusok kis módosításával elérhető, hogy az adott algoritmus ne a kérdéses függvények számát, hanem egy vagy az összes függvényt adja vissza.

Az Ullmann- és a VF2 algoritmus visszalépéses keresésen alapuló módszerek, melyek futása során egy-egy keresési fát épít fel. Ezen fa egy csúcsa egy  $f : V_1 \rightarrow V_2$  parciális izomorfizmusnak felel meg. Célunk, hogy ezen leképezést bővítve, az 1.3. definíciónak eleget tevő  $f : V_1 \rightarrow V_2$  függvény(ek)e)t kapjunk, vagy jelezzük, amennyiben az adott parciális izomorfizmust bővítve ilyen függvény biztosan nem létezik. Mivel a teljes fa exponenciális méretű lehet, csak azon részeit építjük fel, amelyeket be kell járnunk. Annak érdekében, hogy a keresés során kihagyható részfákat minél hamarabb kiszűrjünk – és ezáltal az algoritmus minél gyorsabb legyen –, mindkét algoritmus ún. „Finomítási kritériumokat” vezet be. A két algoritmus a keresési fát

lényegesen különböző módon építi fel, valamint az alkalmazott adatszerkezetek is különbözőek. A részleteket az egyes algoritmusok leírásánál tárgyaljuk.

H. Shang munkatársaival egy további részstruktúra-kereső algoritmust, a QuickSI-t (quick subgraph isomorphism) publikált 2008-ban [34]. Ezen algoritmus is egy visszalépéses keresésen alapuló módszer, mely csúcs- vagy élcímkézett gráfok esetén működik hatékonyan. Habár a szerzők a cikkben csúcscímkézett gráfokkal foglalkoztak, az ismertetett eljárás könnyen alkalmazható csúcs- és élcímkézett gráfokra. Ezen algoritmus előre rögzíti a query molekulában a feldolgozandó atomok sorrendjét, majd ezen sorrend alapján építi fel a keresési fa egyes szintjeit. A sorrendet igyekszik úgy meghatározni, hogy a keresési fa várhatóan minél kisebb legyen. Mivel az algoritmus lényeges, a többi algoritmustól eltérő lépése a query atomjainak átrendezése kezdeti lépésként, ezért ezen algoritmust nem a cikknek megfelelően implementáltuk. Mindössze az atomok átrendezését vizsgáltuk, és az Ullmann- és a VF2 algoritmusokat ezen új sorrendre futtattuk.

Egy tipikus keresési feladat során azon molekulákat keressük egy molekula-adatbázisból, melyek részstruktúraként tartalmazzák az adott query-t. Ekkor elegendő egyszer meghatározni a várhatóan optimális atomsorrendet a query-ben, melyet utána az összes, páronkénti részstruktúra-vizsgálat során alkalmazhatunk.

A VF2 algoritmus a keresés egyes szintjeit nem a query előre meghatározott atomsorrendje alapján építi fel, hanem úgy, hogy ha a query már leillesztett atomjainak szomszédsági halmaza nem üres, akkor onnan veszi a minimális indexű atomot, különben pedig a még le nem illesztett atomok közül a legkisebb indexűnek a képét keresi. Látni fogjuk, hogy a QuickSI által meghatározott atomsorrendre teljesül a VF2-nél szükséges invariánstulajdonság, ezért az atomok további átrendezése nincs szükség.

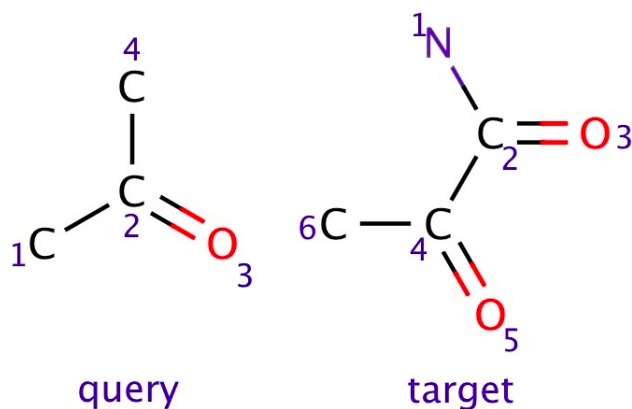
### 2.2.1. Az Ullmann-algoritmus

#### Az algoritmus

A továbbiakban J. R. Ullmann [35] 1976-ban publikált, gyakran alkalmazott algoritmusát elemezzük. Az algoritmus kezdőlépéseként minden  $u_i \in V_1$  csúcshoz hozzárendeljük a lehetséges képek halmazát:

$$\mathcal{H}_i := \{v_j | v_j \in V_2 \wedge l_{V_1}(u_i) = l_{V_2}(v_j) \wedge \deg(u_i) \leq \deg(v_j)\},$$

ahol  $\deg(u_i)$  jelöli az  $u_i$  csúcs fokszámát  $G_1$ -ben.



2.2. ábra. Példa az Ullmann-algoritmus inputjára

## 2.2. Példa

Tekintsük a 2.2. ábrán látható query és target molekulákat. A  $\mathcal{H}_i$  halmazokat meghatározva kapjuk:

$$u_1 \mapsto \mathcal{H}_1 = \{v_2, v_4, v_6\}$$

$$u_2 \mapsto \mathcal{H}_2 = \{v_2, v_4\}$$

$$u_3 \mapsto \mathcal{H}_3 = \{v_3, v_5\}$$

$$u_4 \mapsto \mathcal{H}_4 = \{v_2, v_4, v_6\}$$

Rögzítsünk  $G_1$  csúcsainak egy tetszőleges sorrendjét:  $u_1, u_2, \dots, u_n$ . Ekkor a keresési fát a következőképpen építjük fel. A fa gyökere legyen az üres halmaz. Ez lesz a 0. szint. A fa minden egyes csúcsa egy  $f : V_1' \rightarrow V_2'$  injektív leképezés, ahol  $V_1' \subseteq V_1$  és  $V_2' \subseteq V_2$  és  $|V_1'| = |V_2'|$ . Formálisan, egy, az  $i$ . szinten lévő belső csúcsnak a következő függvény felel meg:

$$f(u_1) = v_{j_1} \in \mathcal{H}_1$$

$$f(u_2) = v_{j_2} \in \mathcal{H}_2$$

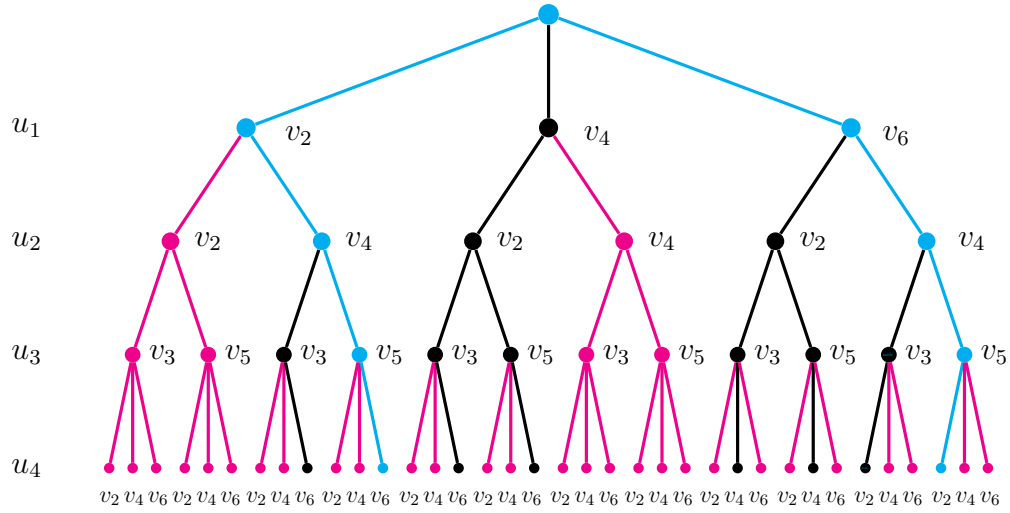
$$\vdots$$

$$f(u_i) = v_{j_i} \in \mathcal{H}_i$$

Ezen leképezést próbáljuk meg kiterjeszteni  $u_{i+1}$ -re úgy, hogy  $f(u_{i+1})$  értéke a  $\mathcal{H}_{i+1}$  halmazból való, és  $f$  továbbra is injektív legyen. Így meghatároztunk egy olyan  $f : V_1 \rightarrow V_2$  injektív leképezést, melyre

$$f(u_i) = v_{j_i} \in \mathcal{H}_i \quad \forall i = 1, \dots, n_1,$$

ahol  $n_1 := |V_1|$ .



2.3. ábra. A keresési fa a 2.2. ábrán látható input esetén

### 2.3. Példa

Legyen az algoritmus inputja a 2.2. ábrán szereplő query-target pár. Ekkor a 2.3. ábrán látható keresési fát kapjuk, ha a csúcsokat az  $u_1, u_2, u_3, u_4$  sorrendben dolgozzuk fel.

Az így kapott leképezések között keressük az összes olyan  $f$ -et, mely teljesíti a részgráf-izomorfia 1.3. definíciójában szereplő feltételeket. Ezen  $f$  függvények keresésére visszalépéses keresést használunk a kialakult fán. Tekintsük a keresési fa egy belső csúcsát:

$$f(u_1) = v_{j_1} \in \mathcal{H}_1$$

$$f(u_2) = v_{j_2} \in \mathcal{H}_2$$

$$\vdots$$

$$f(u_i) = v_{j_i} \in \mathcal{H}_i$$

Az  $u_{i+1}$  csúcshoz hozzárendeljük a  $v \in \mathcal{H}_{i+1}$  csúcsot,

1. ha  $f$  továbbra is injektív marad:  $\forall k \in [1, \dots, i](v \neq v_{j_k})$
2. ha léteznek élek a megfelelő képek között, és ekkor az élcímkék is megegyeznek:

$$\forall k \in [1, \dots, i] ((u_k, u_{i+1}) \in E_1 \Rightarrow ((f(u_k), f(u_{i+1})) \in E_2 \wedge l_{E_1}((u_k, u_{i+1})) = l_{E_2}((v_{j_k}, v))))$$

3. ha  $u_{i+1}$  minden szomszédjának lehet olyan képe, amely szomszédja  $v$ -nek.

Abban az esetben, ha ezen feltételek közül valamelyik nem teljesül, akkor töröljük  $v$ -t  $u_i$  lehetséges képei közül. Amennyiben az összes feltétel teljesül, a parciális izomorfizmust kiegészítjük az  $f(u_{i+1}) = v$  megfeleltetéssel,  $v$ -t töröljük az összes  $\mathcal{H}$  halmazból, és ekkor a módosított parciális izomorfizmust próbáljuk tovább bővíteni.

## 2.4. Példa

*Az injektivitási kritérium alapján a 2.2. példa esetén a bíborvörös színnel jelölt csúcsokat a 2.3. ábrán látható keresési fából törölhetjük. A fában kék színnel jelöltük az összes, olyan parciális izomorfizmust, melyet bővítve végül  $f : V_1 \rightarrow V_2$  részgráf-izomorfizmust kapunk.*

A finomítási eljárás során folyamatosan figyeljük, hogy valamelyik  $\mathcal{H}_i$  üres halmaz-e. Ekkor az algoritmus visszalép az előző szintre. Amennyiben az algoritmus a 0. szintről lépne vissza, az eljárás leáll.

## Alkalmazott adatstruktúrák

Jelölje  $n_1$  a  $G_1$  gráf,  $n_2$  pedig a  $G_2$  gráf csúcshalmazának elemszámát. Az Ullmann-algoritmus során a  $\text{mapping} \in \{1, \dots, n_2\}^{n_1}$  segítségével tároljuk az aktuális parciális izomorfizmust:  $\text{mapping}[i] = j$ , ha  $f(u_i) = v_j$ .

A visszalépéses keresés lehetséges lépéseit az  $M \in \mathbb{L}^{n_1 \times n_2}$  mátrix segítségével írjuk le, ahol  $\mathbb{L} = \{\text{igaz}, \text{hamis}\}$ . A mátrix a keresési fa egy szintjén a lehetséges elágazásokat tárolja. Az  $M$  mátrix segítségével a keresési fa méretét is csökkenthetjük, ahol inicializáláskor

$$m_{ij} := \begin{cases} \text{igaz} & \text{ha } \deg(u_i) \leq \deg(v_j) \wedge l_{V_1}(u_i) = l_{V_2}(v_j) \\ \text{hamis} & \text{különben} \end{cases}$$

Jelöljük  $d$ -vel, hogy a keresési fa épp mely szintjén járunk. Részgráf-izomorfizmust abban az esetben találunk, amennyiben minden  $G_1$ -beli csúcshoz megfeleltettünk egy  $G_2$ -beli csúcsot az 1.3. definíciónak megfelelően, azaz  $d \geq n_1$ . Az 1. algoritmus formájában összefoglalt kód 1 – 3. sora ezen tulajdonságot ellenőrzi. A 4 – 5. sor felsorolja azon fel nem használt csúcsokat, melyek kielégítik a Finomítási kritériumokat. Abban az esetben, ha a Finomítási kritérium igaz az  $(u_d, v_k)$  párra, a tömböt és a mátrixot ennek az állapotnak megfelelően módosítjuk.

Az algoritmus fontos lépése a mátrix ritkítása, melyet a mátrix inicializálása után és minden állapotmentést követően elvégzünk. Ekkor megvizsgáljuk, hogy ha  $m_{ij}$  igaz, akkor a  $G_1$  gráf  $i$ . csúcsának a szomszédainak lehetséges képei között van-e olyan csúcs, mely a  $G_2$  gráf  $j$ . csúcsának szomszédja. Abban az esetben, ha ez a

feltétel nem teljesül,  $m_{ij}$ -t hamisra állítjuk. Ha sikerült a mátrix valamely elemét hamisra állítani, a mátrixot megpróbáljuk újra ritkítani.

Az összefoglalt pszeudokód a 10. sorban visszalép az előző szintre, amennyiben az adott csúcs alatt biztosan nincs részgráf-izomorfizmus függvény.

---

**algorithm 1** *UllmannAlgoritmus*( $G_1, G_2, M, mapping, d$ )

---

**Input** :  $G_1, G_2$  gráfok

$M$  : mátrix

$mapping$ :  $n_1$  hosszú tömb, kezdetben minden értéke -1

$d$ : keresési mélység, értéke kezdetben 0

**Output**: találatok száma  $\in \mathbb{Z}$ : az 1.3. definíciónak eleget tevő  $f : V_1 \rightarrow V_2$  függvények száma, értéke kezdetben 0

```

1: if  $d \geq n_1$  then
2:    $találatok\ száma := találatok\ száma + 1$ 
3: end if
4: for each  $1 \leq j \leq n_2$  do
5:   if FinomításiKritérium( $d, j$ ) igaz then
6:     AdatokMentéseÉsMátrixRitkítása( $d, j$ )
7:     UllmannAlgoritmus( $G_1, G_2, M, mapping, d + 1$ )
8:     AdatokVisszaállítása()
9:   end if
10: end for each

```

---

Fontosnak tartjuk hangsúlyozni, hogy míg a keresés során minden egyes mélység esetén el kell tárolnunk az aktuális  $M$  mátrixot, addig a tömbről nem készítünk másolatot a parciális izomorfizmus minden egyes bővítésénél, hanem ugyanazt a tömböt használjuk végig, bővítés és visszalépés esetén pedig konstans időben módosítjuk a megfelelő elemét.

Az Ullmann-algoritmus memóriaigénye  $O(n^3)$ , ahol  $n := n_1 + n_2$ , mivel a tömb mérete  $O(n_1)$  és minden lépésben eltároljuk az  $M$  mátrixot, a keresési fa konstrukciója alapján  $n_1$  különböző lépés lehet.

### 2.2.2. A VF2 algoritmus

#### Az algoritmus

A következőkben L. P. Cordella, P. Foggia, C. Sansone és M. Vento 2001-ben publikált algoritmusát [9], a VF2-t elemezzük. Az algoritmus alkalmazhatóságának fontos feltétele, hogy a  $G_1$  gráf csúcshalmaza teljesen rendezett legyen. Az algorit-

mus egy visszalépéses keresés, melynek során az egyes állapotok egy-egy parciális izomorfizmusnak felelnek meg.

A parciális izomorfizmus kibővítéséhez az algoritmus először meghatározza azon jelölt  $(u, v)$  párokat, melyekkel ezen  $s$  állapotot bővíthetné. Jelölje  $Jelöltek(s)$  az ezen jelölt párokat tartalmazó halmazt. Ezen párokra nem teljesül, hogy az  $s$  állapotot bármely  $Jelöltek(s)$ -beli párral bővítve továbbra is (parciális) izomorfizmust kapunk  $G_1$  és  $G_2$  között. Ennek a feltételnek az ellenőrzését a  $FinomításiKritérium(s, u, v)$  függvény végzi.  $FinomításiKritérium(s, u, v)$  értéke pontosan akkor igaz, ha az  $s$  állapotot az  $(u, v)$  párral bővítve továbbra is (parciális) izomorfizmust kapunk.

Mielőtt a  $Jelöltek(s)$  halmaz és a  $FinomításiKritérium(s, u, v)$  kiszámítását részleteznénk, bevezetünk néhány jelölést, melyre a későbbiekben szükségünk lesz. Legyen  $M(s) := \{(u, v) | u \in V_1 \wedge v \in V_2 \wedge f(u) = v\}$ , ahol  $f$  parciális izomorfizmus  $G_1$  egy feszített részgráfja és  $G_2$  között, azaz  $M(s)$  az aktuális parciális izomorfizmusnak megfelelő párokat tartalmazza. Jelölje  $s_0$  a kezdőállapotot. Ekkor nyilván  $M(s_0) = \emptyset$ . Jelölje  $G_1(s)$ , illetve  $G_2(s)$  azon  $V_1$  illetve  $V_2$ -beli csúcsokat, melyeket  $M(s)$  tartalmaz, azaz precízen:  $G_1(s) := \{u \in V_1 | \exists v \in V_2 f(u) = v \wedge (u, v) \in M(s)\}$ .  $G_2(s)$  hasonlóan definiálható. Legyen  $u$  egy tetszőleges  $G_1(s)$ -beli csúcs. Jelöljük  $N(u)$ -val az  $u$  csúcs  $V_1$ -beli szomszédait tartalmazó halmazt, tehát  $N(u) := \{u_i \in V_1 | (u, u_i) \in E_1\}$ . Jelöljük  $N(G_1(s))$ -sel a  $G_1(s)$ -beli csúcsok szomszédainak halmazát, vagyis  $N(G_1(s)) := \bigcup_{u \in G_1(s)} N(u)$ . Az  $N(G_2(s))$  halmazt hasonlóan definiálhatjuk.

A következő néhány ábrán a könnyebb áttekinthetőség érdekében a következő színézést alkalmazzuk: a  $G_1(s)$  és  $G_2(s)$  halmazokat kék színnel, az  $N(G_1(s)) \setminus G_1(s)$  és az  $N(G_2(s)) \setminus G_2(s)$  halmazokat pedig zöld színnel jelöljük.

A jelölt párokat tartalmazó halmaz,  $Jelöltek(s)$  meghatározásakor kétféle esetet különböztetünk meg, melyeket a 2.4 és 2.5 ábrán is feltüntettünk.

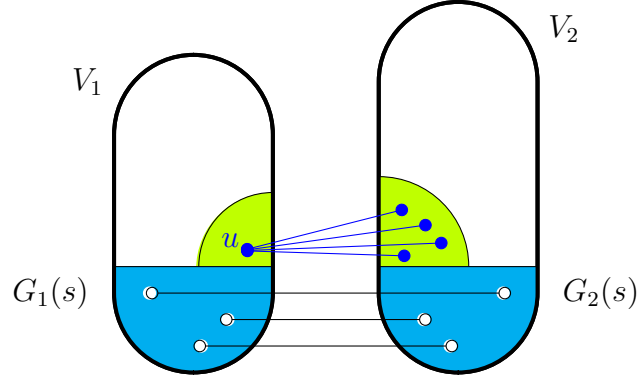
1. eset:  $N(G_1(s)) \setminus G_1(s) \neq \emptyset$

A keresési tér minimalizálása érdekében ekkor  $N(G_1(s)) \setminus G_1(s)$  legkisebb sorszámu csúcsának keressük a képét. Ezen csúcs képe az élstruktúra megtartása miatt csak  $N(G_2(s)) \setminus G_2(s)$  halmazbeli csúcs lehet. Emiatt ezen esetben  $Jelöltek(s) := \{\min(N(G_1(s)) \setminus G_1(s))\} \times (N(G_2(s)) \setminus G_2(s))$ .

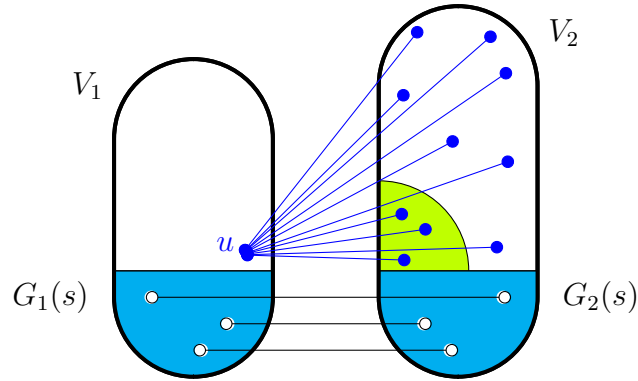
2. eset:  $N(G_1(s)) \setminus G_1(s) = \emptyset$

Amennyiben  $s \neq s_0$ , akkor a  $G_1$  gráf biztosan nem összefüggő. Ebben az esetben  $V_1 \setminus G_1(s)$  legkisebb sorszámu csúcsának keressük a képét. Ezen csúcs

szomszédainak képét még nem ismerjük, emiatt a csúcs képe bármely, korábban még fel nem használt csúcs lehet. Ezen gondolatmenet alapján kapjuk, hogy ekkor  $Jelöltek(s) := \{\min(V_1 \setminus G_1(s))\} \times (V_2 \setminus G_2(s))$ .



2.4. ábra. A  $Jelöltek(s)$  halmaz kiszámításának 1. esete

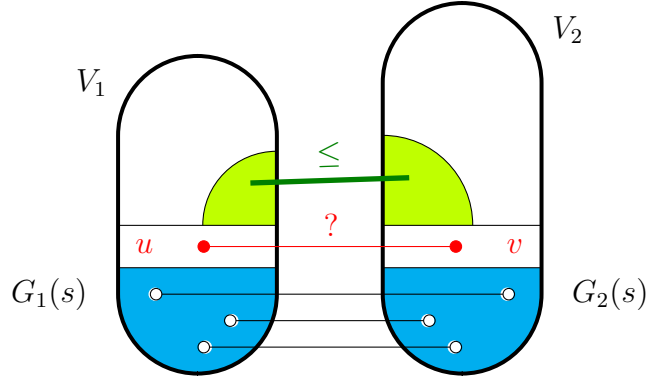


2.5. ábra. A  $Jelöltek(s)$  halmaz kiszámításának 2. esete

*Megjegyzés.* Abban az esetben, ha a részgráf-izomorfia alatt a szigorúbb, feszített részgráfként történő tartalmazást értjük, a  $Jelöltek(s)$  halmaz meghatározása a 2. esetben a következőképpen változik az élstruktúra megtartása érdekében:  $Jelöltek(s) := \{\min(V_1 \setminus G_1(s))\} \times (V_2 \setminus (N(G_2(s)) \cup G_2(s)))$ .

A  $FinomításiKritérium(s, u, v)$  logikai függvény alkalmazásával a keresési fa méretét csökkenthetjük. Értéke pontosan akkor igaz, ha az  $M(s') := M(s) \cup \{(u, v)\}$  (parciális) izomorfizmust tovább bővítve végül egy részgráf-izomorfizmust kaphatunk  $G_1$  és  $G_2$  között. A  $FinomításiKritérium(s, u, v)$  kiszámításánál megvizsgáljuk, hogy  $u$  és  $v$  kompatibilis-e: szükséges, hogy a csúcs- és az élcímkek egyezők legyenek, illetve  $G_2(s')$ -nek legalább annyi még le nem illesztett szomszédja kell, hogy legyen, mint  $G_1(s')$ -nek (2.6. ábra). Formálisan a következő feltételeknek kell teljesülniük:

1.  $l_{V_1}(u) = l_{V_2}(v)$
2.  $\forall (u, u_i) \in E_1 (u_i \in G_1(s) \Rightarrow l_{E_1}(u, u_i) = l_{E_2}(v, f(u_i)))$
3.  $|N(G_1(s) \cup \{u\}) \setminus (G_1(s) \cup \{u\})| \leq |N(G_2(s) \cup \{v\}) \setminus (G_2(s) \cup \{v\})|$



2.6. ábra. *FinomításiKritérium*( $s, u, v$ ) 3. feltétel

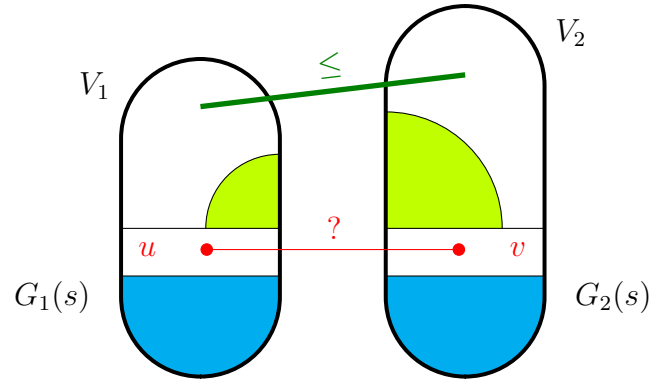
*Megjegyzés.* Ha a részgráf-izomorfia alatt a szigorúbb, feszített részgráfként tartalmazást értjük, akkor a *FinomításiKritérium* függvény kiszámításakor a 2.7. ábrán jelölt halmazok számosságára vonatkozó egyenlőtlenségnek is teljesülnie kell, formálisan:

$$4. |V_1 \setminus (N(G_1(s) \cup \{u\}) \cup G_1(s) \cup \{u\})| \leq |V_2 \setminus (N(G_2(s) \cup \{v\}) \cup G_2(s) \cup \{v\})|.$$

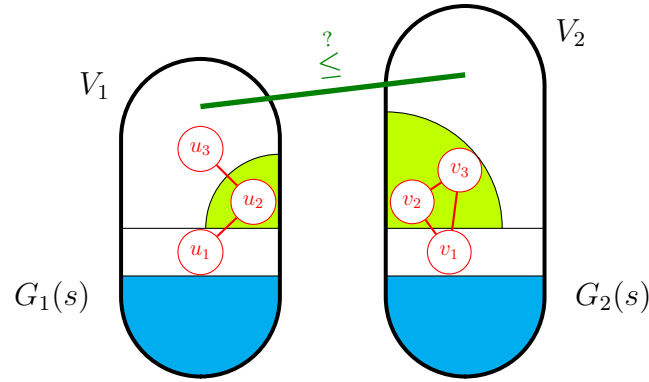
Ezen egyenlőtlenség nem teljesül abban az esetben, ha részgráf-izomorfia alatt a dolgozatban is alkalmazott, 1.3. definíciót értjük. Tekintsük erre az alábbi egyszerű példát.

## 2.5. Példa

Legyen  $G_1$  és  $G_2$  a 2.8. ábrán látható gráfok. Ekkor  $s = s_0$ , azaz a kezdőállapotban vagyunk, és a *FinomításiKritérium*( $s_0, u_1, v_1$ ) értékét szeretnénk meghatározni. Ekkor az előbb említett 4. feltétel nem teljesül, de  $G_1 \subseteq G_2$  fennáll. Egy lehetséges  $f : V_1 \rightarrow V_2$  részgráf-izomorfizmus:  $f(u_1) = v_1$ ,  $f(u_2) = v_2$  és  $f(u_3) = v_3$ .



2.7. ábra. További feltétel a  $FinomításiKritérium(s, u, v)$  kiszámításakor feszített részgráf-izomorfia esetén



2.8. ábra. Nem teljesül a  $FinomításiKritérium(s, u, v)$  4. feltétele nem feszített részgráf-izomorfia esetén

### Alkalmazott adatstruktúrák

Legyen a csúcsok sorszáma  $G_1$ -ben  $1, \dots, n_1$ ,  $G_2$ -ben pedig  $1, \dots, n_2$ . Az aktuális parciális izomorfizmust ezen algoritmus esetén is – mint a korábban bemutatott Ullmann-algoritmusnál – a `mapping[1, ..., n1]` tömb segítségével tároljuk: `mapping[i]=j`  $\Leftrightarrow f(u_i) = v_j$  ( $u_i \in V_1, v_j \in V_2$ ). Továbbá a részgráf-izomorfizmus injektivitása miatt szükséges feljegyeznünk, hogy mely  $V_2$ -beli csúcsok vannak benne az aktuális parciális izomorfizmus értékkészletében. Ezen információt az `isUsed[1, ..., n2]` tömbben tároljuk: `mapping[i]=j` és `isUsed[j]=true`  $\Leftrightarrow f(u_i) = v_j$  ( $u_i \in V_1, v_j \in V_2$ ). Kezdetben  $\forall i \in [1, \dots, n_1]$  `mapping[i]=-1` és  $\forall j \in [1, \dots, n_2]$  `isUsed[j]=false`.

A szomszédsági halmazok tárolásához további két tömböt, `querysNeighbours[1, ..., n1]`-t és `targetsNeighbours[1, ..., n2]`-t használunk. `querysNeighbours[i]` értéke azon keresési mélység, melynek során az  $u_i \in V_1$

---

**algorithm 2** *VF2Algoritmus*( $s$ )

---

**Input** :  $G_1, G_2$  gráfok

**Output**: találatok száma  $\in \mathbb{Z}$ : az 1.3. definíciónak eleget tevő  $f : V_1 \rightarrow V_2$  függvény(ek) száma, értéke kezdetben 0

```
1: if  $d \geq n_1$  then
2:   találatok száma := találatok száma + 1
3: else
4:   Jelöltek( $s$ ) kiszámítása
5:   for each  $(u, v) \in \text{Jelöltek}(s)$  do
6:     if FinomításiKritérium( $s, u, v$ ) igaz then
7:        $M(s') := M(s) \cup \{(u, v)\}$ 
8:       AdatokMentése( $u, v$ )
9:       VF2Algoritmus( $s'$ )
10:      AdatokVisszaállítása()
11:    end if
12:  end for each
13: end if
```

---

csúcs bekerült a  $G_1(s) \cup N(G_1(s))$  halmazba. A *targetsNeighbours* tömb elemei hasonlóan definiálhatóak. Kezdetben ezen két tömb minden eleme 0.

Ezen tömbök fontos tulajdonsága, hogy ha az  $i$ . tömbelem nem nulla az  $s$  állapotban, akkor az összes állapot során, melyet  $s$  kibővítésével értünk el, továbbra sem lesz nulla. Ezért nem szükséges minden állapothoz külön tárolni ezen négy tömböt. Amennyiben visszalépésre kerül sor, az eltárolt információk alapján vissza tudjuk állítani az előző állapotot tároló tömböket.

Az algoritmus során folyamatosan nyilvántartjuk az  $N(G_1(s)) \setminus G_1(s)$  és az  $N(G_2(s)) \setminus G_2(s)$  halmazok számosságát. Ezen két érték tárolásával elkerülhető, hogy a  $G_1$  illetve a  $G_2$  összes atomját minden állapot során meg kelljen vizsgálni, hogy megállapíthassunk ezen számosságokat, ehelyett elegendő ezt az ellenőrzést az  $u$  illetve  $v$  atomok szomszédaira elvégezni, s a korábbi értéket szükség esetén módosítani. Minden állapot esetén feljegyezzük azon  $(u, v)$  párt, mellyel a parciális izomorfizmust kiegészítettük. A különböző keresési mélységekhez tartozó információkat a *history*[1, ...,  $n_1$ ] tömbben tároljuk.

Mivel a tömbök mérete  $n$ -nel arányos, továbbá minden állapot során konstans méretű memóriát használunk, és legfeljebb  $n_1$  állapot van, melyet tárolnunk kell, így az algoritmus memóriaigénye  $O(n)$ . Az algoritmus implementálása során alkalmazott heurisztikákat a későbbiek során ismertetjük. Látni fogjuk, hogy ezen heurisztikák

alkalmazása esetén az algoritmus memóriaigénye továbbra is  $O(n)$ .

## 2.3. Várhatóan optimális atomsorrend meghatározása

A továbbiakban a H. Shang, Y. Zhang, X. Lin és J. Xu Yu cikke [34] által 2008-ban publikált részstruktúra-kereső algoritmus, a QuickSI (quick subgraph isomorphism) a korábban ismertetett algoritmusoktól eltérő lépésével, az atomok feldolgozási sorrendjének a kiválasztásával foglalkozunk. A szerzők az algoritmust csúscímkezt gráfokra ismertették, melyet úgy módosítottunk, hogy címkézett gráfokra oldjuk meg a problémát, mivel a kémiai molekulagráfok ilyenek. Megjegyezzük, hogy az eredeti algoritmus csak összefüggő, legalább kétcsúcú gráfok esetén alkalmazható, melyet úgy módosítottunk, hogy több komponensből álló, izolált csúcsokat tartalmazó gráf esetén is használható legyen.

Célunk, hogy a query molekulában az atomokat olyan sorrendben dolgozzuk fel, hogy a keresési fa mérete minél kisebb legyen. Az algoritmus első lépésben a keresésre használt adatbázison az egyes atom- és kötéstípusok előfordulásával foglalkozik, majd ezen adatokat felhasználva igyekszik minél jobb atomsorrendet meghatározni. Jelöljük  $D$ -vel az adatbázisban található gráfokat tartalmazó halmazt.

**2.1. Definíció** *Adott egy  $x$  csúscímke. Definiáljuk  $x$  átlagos belső tartóját,  $\phi(x)$ -et a következőképpen:  $\phi(x) := \frac{|\{u \mid \exists G \in D(u \in V(G) \wedge l_{V(G)}(u) = x)\}|}{|\{G \mid G \in D \wedge \exists u \in V(G)(l_{V(G)}(u) = x)\}|}$ .*

Az előszűrés során könnyen kiszűrhetjük azon gráfokat, melyek nem tartalmaznak szénatomot, ezért a nevezőben  $|D|$  helyett azon gráfok száma szerepel, melyek tartalmaznak szénatomot. Például  $\phi(C)$  az az érték, melyet úgy kapunk, ha a  $D$ -beli  $C$ -atomok számát elosztjuk azon gráfok számával, melyek tartalmaznak  $C$ -t. Emiatt  $\phi(C)$  nem azt fejezi ki, hogy mennyire gyakori ezen csúscímke, hanem azt, hogy ha egy gráfban van  $C$ , akkor mennyire várható, hogy lesz még egy  $a$  gráfban.

Hasonlóan definiálhatjuk egy csúcs-él-csúscímke sorozat átlagos belső tartóját. Ezen esetben figyelembe vesszük az él két végén szereplő csúcsok címkéit is. Például  $C = O$  átlagos belső tartójának kiszámítása során az adatbázisban előforduló  $C = O$  kapcsolódások számát osztjuk azon gráfok számával, melyek tartalmaznak  $C = O$  kapcsolódást. A továbbiakban a „csúcs-él-csúscímke átlagos belső tartója” helyett a rövidebb „él átlagos belső tartója” elnevezést használjuk, nem elfelejtve, hogy értéke függ az él két végén szereplő két csúcs címkéjétől is. Jelöljük  $\phi_e(y)$ -nal az  $y$  él átlagos belső tartóját.

## 2.2. Definíció

$$\phi_e(x \diamond y) := \frac{|\{(u, v) \mid \exists G \in D((u, v) \in E \wedge l_V(u) = x \wedge l_V(v) = y \wedge l_E((u, v)) = \diamond)\}|}{|Aut(x \diamond y)| \cdot |\{G \mid G \in D \wedge \exists u, v \in V((l_V(u) = x \wedge l_V(v) = y \wedge l_E((u, v)) = \diamond)\}|},$$

$$ahol |Aut(x \diamond y)| = \begin{cases} 1 & ha \ \phi(u) \neq \phi(v) \\ 2 & ha \ \phi(u) = \phi(v) \end{cases}$$

**2.3. Definíció** Adott a  $G_1$  gráf és egy  $u \in V_1$  csúcs. Definiáljuk az  $u$  csúcs átlagos belső tartóját,  $\phi(u)$ -t a következőképpen:  $\phi(u) := \phi(l_{V_1}(u))$ .

**2.4. Definíció** Adott a  $G_1$  gráf és egy  $(u, v) \in E_1$  él. Definiáljuk az  $(u, v)$  él átlagos belső tartóját,  $\phi_e(u, v)$ -t a következőképpen:  $\phi_e(u, v) := \phi_e(l_{V_1}(u), l_{E_1}((u, v)), l_{V_1}(v))$ .

A  $G_1$  gráf várhatóan optimális atomsorrendjének meghatározásához készítsük el egy súlyozott, címkézett  $G_w$  gráfot a következőképpen. Számítsuk ki minden  $G_1$ -beli csúcsra és élre az átlagos belső tartót, majd minden csúcshoz és élhez rendeljük hozzá ezen értékeket.

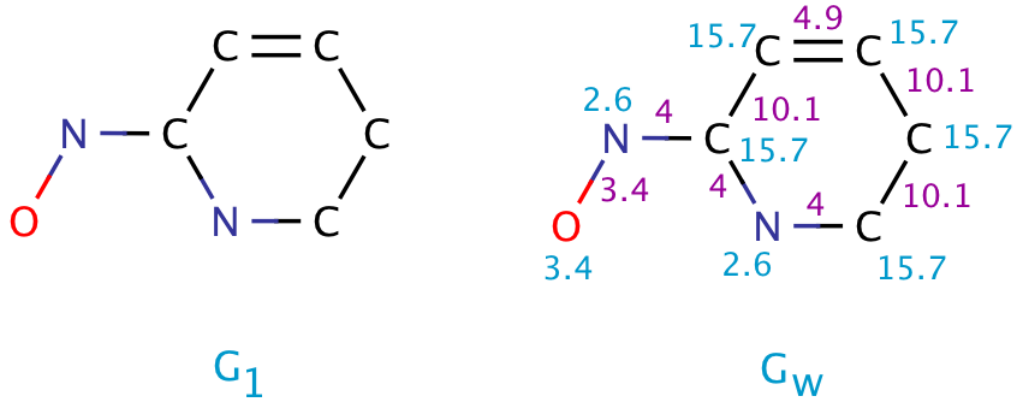
## 2.6. Példa

Az amerikai Nemzeti Rákkutató Intézet publikus adatbázisát [6] felhasználva kiszámoltuk az egyes belső tartókat. A kapott értékek egy része látható a 2.9 ábrán. A 2.10. ábrán látható  $G_1$  gráf esetén a kapott súlyozott gráf tekinthető meg ugyanazon az ábrán. A könnyebb áttekinthetőség érdekében a csúcsok átlagos belső tartóját kék, az élek átlagos belső tartóját pedig lila színnel jelöltük.

| $x$ | $\phi(x)$ | $y$     | $\phi_e(y)$ |
|-----|-----------|---------|-------------|
| $N$ | 2.6       | $N - C$ | 4           |
| $O$ | 3.4       | $N - O$ | 3.4         |
| $C$ | 15.7      | $C - C$ | 10.1        |
|     |           | $C = C$ | 4.9         |

2.9. ábra. Néhány csúcs- és élcímkék átlagos belső tartója

Ezen  $G_w$  gráfot felhasználva készítjük el a várhatóan optimális atomsorrendet a  $G_1$  gráf esetén. Jelöljük  $T[1, \dots, n_1]$ -gyel az ezen eljárás által meghatározott atomsorrendet tartalmazó tömböt. Kezdetben az izolált csúcsokkal foglalkozunk. Az előszűrés során kiszűrtük azon gráfokat, melyek nem tartalmaznak elegendő számú csúcsot az egyes csúcscímkék esetén, valamint a részgráf-izomorfia 1.3. definíciója alapján egy izolált csúcs képe bármely, vele azonos csúcscímkéjű  $G_2$ -beli csúcs lehet.



2.10. ábra. Az input és az előállított súlyozott gráf

Ez azonban nem teljesül nem izolált csúcs esetén. Emiatt célszerű a visszalépések keresései során legutoljára az izolált csúcsok képét meghatározni. Ennek következtében kezdetben az izolált csúcsokkal foglalkozunk, melyeket a  $T$  utolsó pozícióin helyezünk el. Ezt követően a még fel nem dolgozott csúcsokat tartalmazó részgráf várhatóan optimális csúcssorrendjének meghatározásához egy minimális feszítőerdőjét határozzuk meg a Prim-algoritmust [14, 18] alkalmazva, melyet kiegészítünk azon esetekre vonatkozó élválasztással, melyek esetén a Prim-algoritmus véletlenszerűen választana egy élt a minimális súlyú élek közül. A csúcssorrendet aszerint állapítjuk meg, hogy a csúcsok milyen sorrendben kerültek be a feszítőerdőbe.

Ezen algoritmus és a Prim-algoritmus kezdőlépése lényegesen különbözik. Míg a Prim-algoritmus esetén egy tetszőleges csúcsból kiindulva építjük fel a feszítőerdőt, addig ezen esetben egy minimális átlagos tartójú élt választunk ki – abban az esetben, ha több ilyen él is van, akkor ezen élek közül válasszuk azt, melynek két végén lévő csúcsok fokszámösszege a legkisebb –, és ezen két él végpontja lesz a feszítőerdőbe bekerülő első két csúcs. Az első feszítőerdőbeli él,  $(u, v)$  kiválasztása után meghatározzuk, hogy mely csúcs legyen  $T[1]$  és  $T[2]$ . Amennyiben  $\phi(u) \neq \phi(v)$ , akkor  $T[1]$  legyen az a csúcs  $u$  és  $v$  közül, melynek az átlagos belső tartója a kisebb. Abban az esetben, ha ezen értékek megegyeznek, akkor  $T[1]$  legyen  $u$  és  $v$  közül az csúcs, melynek fokszáma nagyobb. Amennyiben a fokszámok is egyenlőek, véletlenszerűen válasszunk. Az élek és a csúcsok közti sorrend megválasztását az indokolja, hogy korábban határozzuk meg a ritkábban előforduló csúcsok képét, s csak ezután keressük a gyakori csúcsok képét. Mivel egy nagyobb fokszámú csúcsnak várhatóan kevesebb lehetséges képe van, mint egy kisebb fokszámú csúcsnak, ezért preferáljuk a nagyobb fokszámú csúcsokat.

Jelöljük  $P$ -vel azon minimális súlyú éleket tartalmazó halmazt, melyek közül kiválasztjuk azt az élt, amivel a  $G_f$  feszítőerdőt kibővítjük. Amennyiben  $P$  legalább

kételemű, válasszuk azt élt, melynek új, nem feszítőerdőbeli csúcsa maximális fokszámú  $G_f$ -ben. Abban az esetben, ha több ilyen él is van, akkor válasszuk azt, mely új csúcsának a fokszáma  $G_w$ -ben a legkisebb. Ha ezek után is több jelölt él van, akkor válasszunk véletlenszerűen.

Abban az esetben, ha a kapott feszítőerdőt nem tudjuk kiegészíteni feszítőerdőből kiinduló éllel, választunk egy újabb, minimális élt a fent leírtaknak megfelelően, kiegészítjük a feszítőerdőt ezen éllel, majd a fenti gondolatmenetet folytatva továbbépítjük a feszítőerdőt.

Ezen, várhatóan optimális atomsorrendet meghatározó algoritmus pszeudokódját tartalmazza a 3. algoritmus. A könnyebb áttekinthetőség érdekében az algoritmust több részre bontottuk. Az egyes részalgoritmusok esetén eltekintünk az input- és outputparaméterek ismertetésétől.

---

**algorithm 3** VárhatóanOptimálisCsúcsSorrend( $G_w$ )

---

**Input** :  $G_w$ :  $G_1$ -ből az átlagos belső tartó alapján elkészített súlyozott gráf

**Output**:  $T[1, \dots, n_1]$ : a várhatóan optimális atomsorrendet tartalmazó tömb

```

1: IzoláltCsúcsokFeldolgozása( $G_w$ )
2:  $\text{kövIndexATömbben} := 1$ 
3: while  $|V_{G_f}| < n_1$  do
4:    $(u, v) := \text{KezdőélKiválasztása}(G_w)$ 
5:    $\text{KezdőCsúcsokSorrendje}(G_w, (u, v))$ 
6:   while van még feldolgozatlan csúcs a komponensben do
7:      $\text{KövÉlKiválasztásaÉsIllesztése}(G_w)$ 
8:   end while
9: end while
10: return  $T[1, \dots, n_1]$ 
```

---

## 2.7. Példa

*Tekintsük a 2.11. ábrán látható gráfot. Mivel a gráf tartalmaz izolált csúcsokat, ezért a várhatóan optimális atomsorrendet meghatározó algoritmus először ezen csúcsok feldolgozási sorszámát határozza meg úgy, hogy ezen sorszámok minél nagyobbak legyenek. Emiatt lett az izolált  $N$  és  $O$  csúcsok sorszáma rendre 13 és 14. Ezt követően kiválasztjuk a minimális átlagos belső tartójú él(ek)e)t. Mivel két ilyen él (a két  $N - O$ ) is van, és ezen élek végpontján lévő csúcsok fokszámösszege megegyezik, ezért véletlenszerűen választunk ezen két él közül egyet. Tegyük fel, hogy a 3 csúcsból álló komponensben található  $N - O$  élt választottuk. Ezután az él két végpontján lévő csúcsok között döntjük el a sorrendet. Mivel  $2.6 = \phi(N) < \phi(O) = 3.4$ , ezért*

---

**algorithm 4** IzoláltCsúcsokFeldolgozása( $G_w$ )

---

**Input** :  $G_w$ :  $G_1$ -ből az átlagos belső tartó alapján elkészített súlyozott gráf

```
1:  $V_{G_f} := \emptyset$ 
2:  $E_{G_f} := \emptyset$ 
3:  $kövIndexATömbben := n_1$ 
4: for  $i = n_1, \dots, 1$  do
5:   if  $\deg_{G_w}(u_i) = 0$  then
6:      $T[kövIndexATömbben] := i$ 
7:      $kövIndexATömbben := kövIndexATömbben - 1$ 
8:      $V_{G_f} := V_{G_f} \cup \{u_i\}$ 
9:   end if
10: end for
```

---

---

**algorithm 5** KezdőélKiválasztása( $G_w$ )

---

**Input** :  $G_w$ :  $G_1$ -ből az átlagos belső tartó alapján elkészített súlyozott gráf

```
1:  $P := \{(u, v) | (u, v) \in E(G_w) \setminus E(G_f) \wedge \forall (u', v') \in E(G_w) (\phi_e(u, v) \leq \phi_e(u', v'))\}$ 
2: if  $|P| > 1$  then
3:    $P := \{(u, v) | (u, v) \in P \wedge \forall (u', v') \in P (\deg(u) + \deg(v) \leq \deg(u') + \deg(v'))\}$ 
4: end if
5: if  $|P| > 1$  then
6:    $(u, v) :=$  válasszunk  $P$ -ből véletlenszerűen
7: else
8:    $(u, v) :=$   $P$ -beli elem
9: end if
10: return  $(u, v)$ 
```

---

a  $N$  csúcs sorszáma 1, az  $O$  csúcs sorszáma pedig 2. Az algoritmus a következő lépésben a komponensben található  $C$  sorszámát határozza meg, mely a 3-as érték. Mivel még nem határoztuk meg az összes csúcs sorszámát, ezért az algoritmus ismét meghatározza a minimális átlagos belső tartójú él(eke)t, mely jelen esetben az ábra közepén látható  $N - O$ . Hasonlóan a korábbiakhoz, a kiválasztott él két végpontján lévő csúcs átlagos belső tartóját megvizsgálva kapjuk, hogy  $2.6 = \phi(N) < \phi(O) = 3.4$ , emiatt a  $N$  csúcs sorszáma 4, az  $O$  csúcs sorszáma pedig 5 lesz. Ezután a komponensben található többi csúcs sorszámát határozzuk meg. A 9-es sorszámú csúcs meghatározásakor két – az ábrán zöld téglalappal jelölt – olyan csúcs közül választhatunk, melyek átlagos belső tartója és az eddigi meghatározott  $G_f$  gráf az új csúccsal kiegészítve ugyan megegyezik, de ezen csúcsok fokszáma  $G_w$ -ben különböző. Ezen két csúcs közül válasszuk azt, melynek a fokszáma  $G_w$ -ben a kisebb. A 10-es

---

**algorithm 6** KezdőCsúcsokSorrendje( $G_w, (u, v)$ )

---

**Input** :  $G_w$ :  $G_1$ -ből az átlagos belső tartó alapján elkészített súlyozott gráf

```
1: if  $\phi(u) = \phi(v)$  then
2:   if  $\deg_{G_w}(u) = \deg_{G_w}(v)$  then
3:      $T[kövIndexATömbben] := u$ 
4:      $kövIndexATömbben := kövIndexATömbben + 1$ 
5:      $T[kövIndexATömbben] := v$ 
6:      $kövIndexATömbben := kövIndexATömbben + 1$ 
7:   else
8:      $T[kövIndexATömbben] := \arg \max\{\deg_{G_w}(u), \deg_{G_w}(v)\}$ 
9:      $kövIndexATömbben := kövIndexATömbben + 1$ 
10:     $T[kövIndexATömbben] := \arg \min\{\deg_{G_w}(u), \deg_{G_w}(v)\}$ 
11:     $kövIndexATömbben := kövIndexATömbben + 1$ 
12:   end if
13: else
14:    $T[kövIndexATömbben] := \arg \min\{\phi(u), \phi(v)\}$ 
15:    $kövIndexATömbben := kövIndexATömbben + 1$ 
16:    $T[kövIndexATömbben] := \arg \max\{\phi(u), \phi(v)\}$ 
17:    $kövIndexATömbben := kövIndexATömbben + 1$ 
18: end if
19:  $V_{G_f} := V_{G_f} \cup \{u, v\}$ 
20:  $E_{G_f} := E_{G_f} \cup \{(u, v)\}$ 
```

---

sorszámú atom meghatározása során szintén két – az ábrán sárga színnel jelölt – csúcs közül választhatunk. Az előbbiekhöz hasonló megfontolás alapján kapjuk, hogy ezen két csúcs közül a  $G_w$ -ben kisebb fokszámú csúcsot válasszuk.

Az algoritmus által meghatározott, várhatóan optimális csúcssorrendet az ábrán lila színnel tüntettük fel.

## 2.4. Az algoritmusok implementálása

Célunk az egyes részgráf-izomorfiaát eldöntő algoritmusok implementálása és összehasonlító elemzése volt. Az implementációk Java programozási nyelven készültek. A molekulák SMILES formátumból történő felépítéséhez és reprezentálásához a Kovács Péter által készített keretrendszert használtuk [32]. A fejlesztés és a futásidők mérése Lenovo T520 laptopon (Intel Core i7 processzor, 8 GB memória), Windows

---

**algorithm 7** K v lKiv laszt sa sIlleszt se( $G_w$ )

---

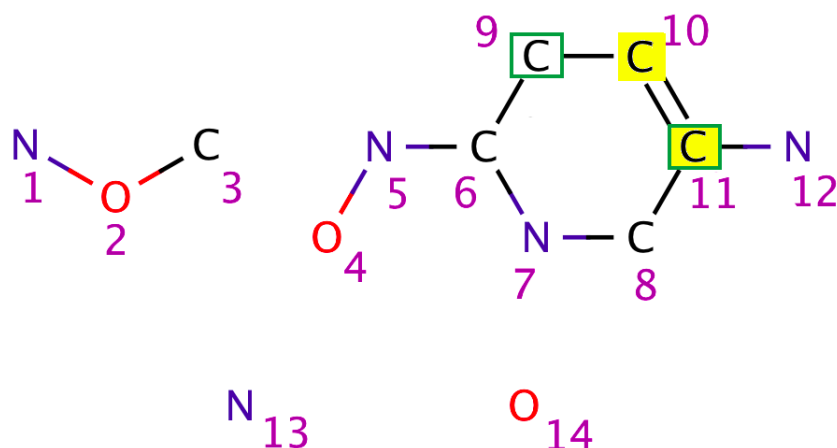
**Input** :  $G_w$ :  $G_1$ -b l az  tlagos bels  tart  alapj n elk sz tett s lyozott gr f

```
1:  $P := \{(u, v) | (u, v) \in E(G_w) \wedge u \in V_{G_f} \wedge v \notin V_{G_f}\}$ 
2:  $P := \{(u, v) | (u, v) \in P \wedge \forall (u', v') \in P (\phi(u, v) \leq \phi(u', v'))\}$ 
3: if  $|P| > 1$  then
4:    $P := \{(u, v) | (u, v) \in P \wedge \forall (u', v') \in P (|Ind_{G_w}(V_{G_f} \cup \{u, v\})| \leq |Ind_{G_w}(V_{G_f} \cup \{u', v'\})|),$ 
     ahol  $Ind_{G_w}(V_{G_f} \cup \{u, v\})$  jel li a  $V_{G_f} \cup \{u, v\}$  cs csok  ltal fesz tett r szgr f
      lhalmaz t  $G_w$ -ben
5: end if
6: if  $|P| > 1$  then
7:    $P = \{(u, v) | (u, v) \in P \wedge \forall (u', v') \in P (deg_{G_w}(v) \leq deg_{G_w}(v'))\}$ 
8: end if
9: if  $|P| > 1$  then
10:   $(u, v) :=$  v lasszunk  $P$ -b l v letlenszer en
11: else
12:   $(u, v) :=$   $P$ -beli elem
13: end if
14:  $T[k vIndexAT mbben] := v$ 
15:  $k vIndexAT mbben := k vIndexAT mbben + 1$ 
16:  $V_{G_f} := V_{G_f} \cup \{v\}$ 
17:  $E_{G_f} := Ind_{G_w}(V_{G_f})$ 
```

---

7 oper ci s rendszeren t rt nt.

Az implement lt r szstrukt ra-keres  algoritmusok fut sidej t t bb teszt szle-  
ten is megm rt k. Vizsg lt k, hogy az egyes implement ci k mennyire gyorsak kis  
query - kis target, kis query - nagy target, nagy query - nagy target molekulap rokon.  
(A k miai informatik ban kicsi molekula alatt a legfeljebb 15 nem-hidrog n atomot  
tartalmaz  molekul t  rtj k, m g a 30 nem-hidrog n atommal rendelke   molekula  
m r nagynak sz m t.) A m r sek sor n alkalmazott molekul kat az amerikai Nem-  
zeti R kkut t  Int zet publikus molekulagy jtem ny b l [6] v lasztott k ki. Adott  
query  s target molekul k esetén az  sszes query-target molekulap rra m rt k a  
fut sid t. A kis query - kis target molekulap rok esetén 20 kis, gyakori molekul t  
– melyek k z l n h ny a 2.12.  br n l that  – kerest nk r szstrukt rak nt a fent  
 ml tett molekulagy jtem ny 11-15 atomot tartalmaz  molekul in ( sszesen 48.243  
molekula). A kis query - nagy target molekulap rok esetén a query molekul k a  
kor bban kiv lasztott 20 kis molekula, a target molekul k pedig a fenti molekula-  
gy jtem ny legal bb 76 nem-hidrog n atomot tartalmaz  molekul i ( sszesen 829



2.11. ábra. Példa a várhatóan optimális atomsorrendre

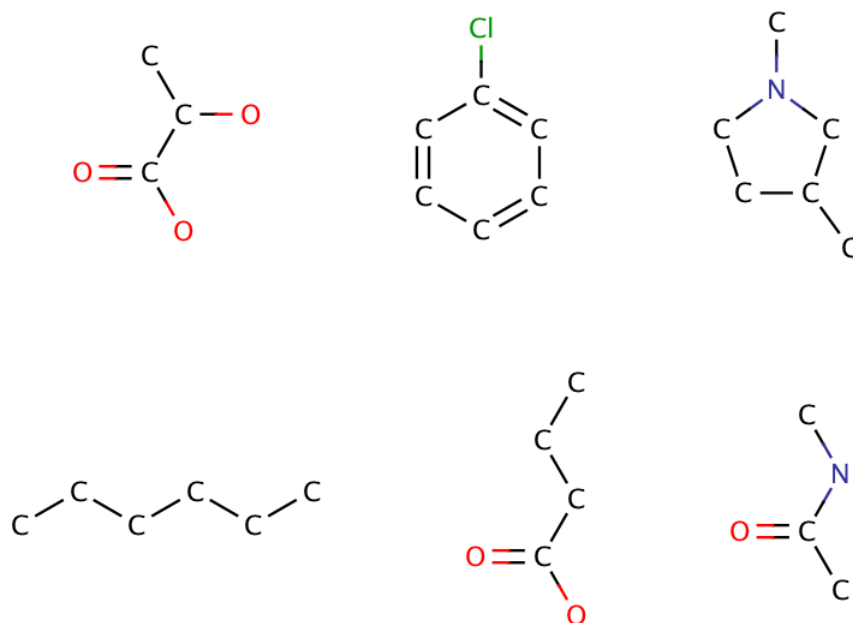
molekula) voltak. Nagy query - nagy target molekulapárok esetén a futásidőt összesen 11 molekulakészletre mértük, és az így kapott futásidők összegét hasonlítottuk össze. Minden esetben a query és target molekulák az adott molekulakészlet molekulái (25, 50, 75 vagy 200 molekula) voltak. Egy tesztkészleten belüli két molekula nagyon hasonló volt. Két nagy, azonos készletben szereplő tesztmolekula látható a 2.13. ábrán. A különböző tesztesetek inputjait foglaljuk össze a 2.1 táblázatban. Az egyes cellákban rendre a teszteléshez használt query és target molekulák leírása szerepel.

Méréseink során a query és a target molekulák SMILES formátumban álltak rendelkezésünkre. Minden esetben megmértük, hogy az egyes implementációk mennyi idő alatt találják meg az első, illetve az összes, 1.3. definíciónak eleget tevő részgráfizomorfizmust.

A továbbiakban a kémiai struktúrák néhány érdekes és fontos jelenségével foglalkozunk, majd az implementálás során alkalmazott heurisztikákat tekintjük át.

|                  |       | Query molekulák                                              |                         |
|------------------|-------|--------------------------------------------------------------|-------------------------|
|                  |       | kicsi                                                        | nagy                    |
| Target molekulák | kicsi | 20 kicsi molekula<br>11-15 atomot tartalmazó molekulák       | –                       |
|                  | nagy  | 20 kicsi molekula<br>legalább 76 atomot tartalmazó molekulák | nagy, hasonló molekulák |

2.1. táblázat.



2.12. ábra. Néhány kis query molekula

### 2.4.1. Kémiai jelenségek a molekulákban

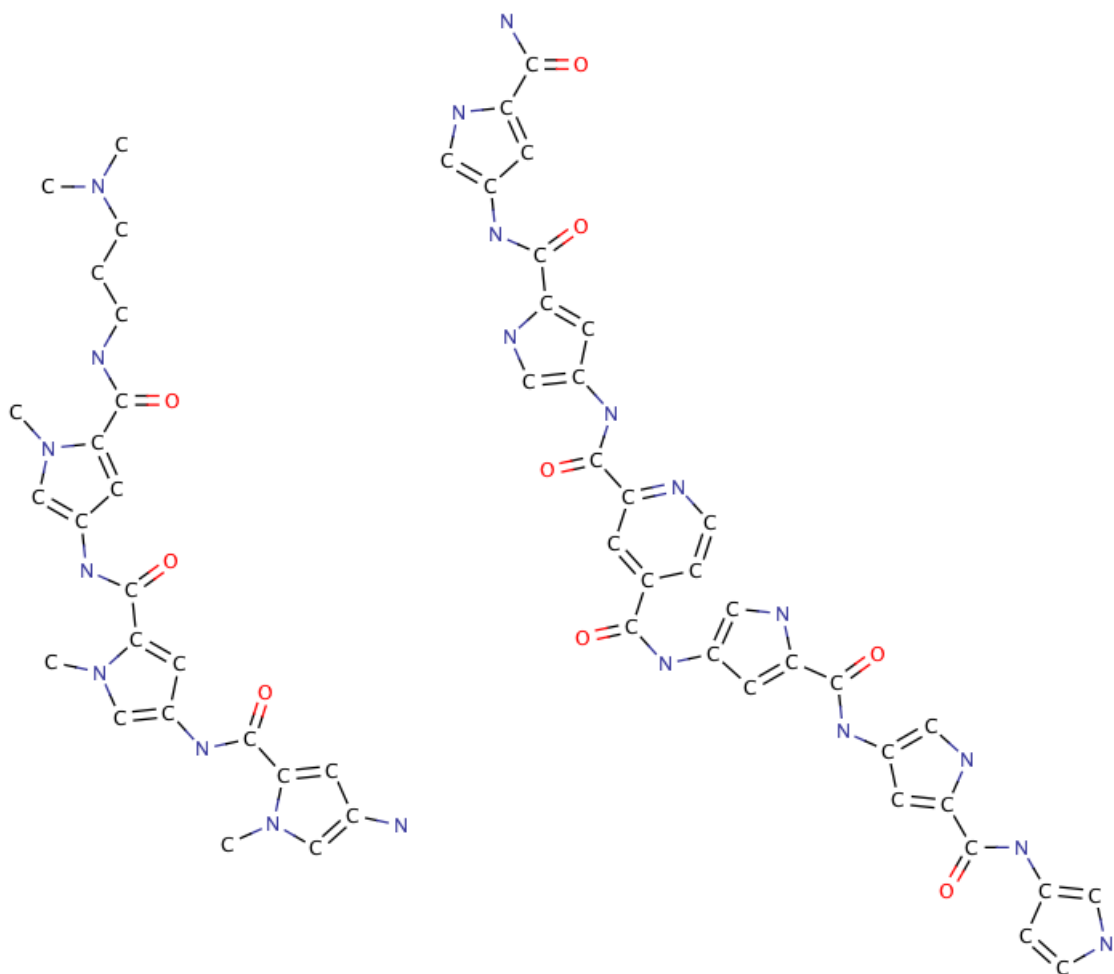
A molekulák kémiai értelemben vett izomorfája nem pontosan esik egybe a nekik megfelelő gráfok izomorfájával. A fejlesztés során nem volt célunk a számos kémiai jelenség korrekt kezelése, csak az egyes részstruktúra-kereső algoritmusok futási idejének összehasonlítása. A vizsgált módszerek hasonló elvek szerint épülnek fel, ezáltal további kémiai intelligencia könnyen beépíthető az egyes algoritmusokba. A következőkben ezek közül néhányat mutatunk be röviden a teljesség igénye nélkül.

Az aromás gyűrűket kialakító atomok delokalizált elektronrendszert alkotnak. Az aromás gyűrűket többféleképpen is ábrázolhatjuk: a gyűrűkben résztvevő atomok közé felváltva rajzolunk egyszeres és kétszeres kötést (Kekulé-formula), vagy az aromás gyűrűket egy körrel jelöljük. Az aromás kötésekkel kapcsolatban bővebb leírás található [8]-ban és [19]-ben.

Természetesen egy gyakorlati alkalmazásban az aromás rendszerek helyes kezelése elengedhetetlen, hiszen a felhasználók a kémiai értelmezésre kíváncsiak. Egy kézenfekvő megoldási módszer egy előfeldolgozási lépés, ami az aromás rendszerekben szereplő egyszeres és kétszeres kötések egy speciális, aromás típusú kötésre cseréli le, így az algoritmusok változatlan formában alkalmazhatóak.

## 2.8. Példa

A 2.14. ábrán látható molekulák az eltérő rajz ellenére ugyanazt a molekulát jelölik.



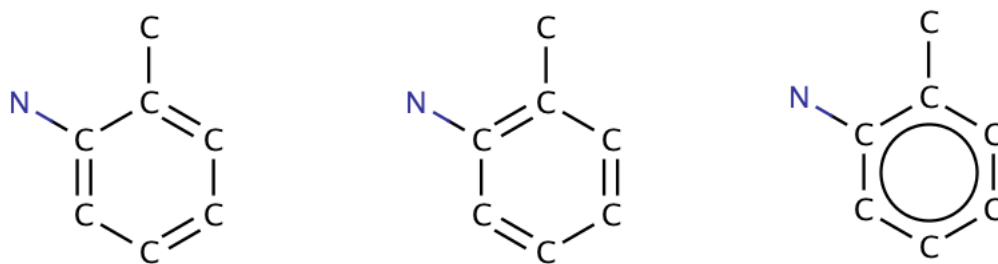
2.13. ábra. Két nagy tesztmolekula

*Tekintsük a 2.15. ábrán látható query és target molekulát. A részgráf-izomorfia 1.3. definíciója szerint a target molekula nem tartalmazza részstruktúraként a query molekulát, azonban kémiai értelemben igen.*

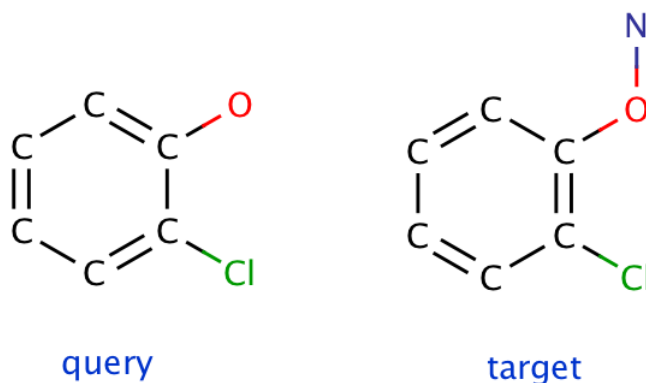
Kémiailag fontos jelenség továbbá a tautomerizáció [26], melynek során normál körülmények között két molekula szabadon átalakul egymásba. A 2.17. ábrán egy molekula két különböző tautomerjét láthatjuk.

A sztereoizomer molekulák [26] esetén a molekulák által meghatározott gráfok izomorfak, azonban a molekulák 3-dimenziós szerkezete mégis különböző. A 2.18. ábrán két ilyen molekula látható.

Mivel a molekulák reprezentálásához használt keretrendszer nem biztosított lehetőséget molekula aromatiszálására, tautomerizációra valamint sztereoizomerek megkülönböztetésére, ezen funkcionalitások elkészítésétől a kellően összetett kémiai szabályrendszer miatt eltekintettünk, ezért nem a kémiai, hanem matematikai értelemben vett részstruktúra-tartalmazást vizsgáltuk.



2.14. ábra. Példa egyazon aromás vegyület többféle lerajzolására



2.15. ábra. Példa arra, hogy a target molekula részstruktúraként tartalmazza a query molekulát az aromás gyűrűszerkezetek kémiai értelmezése miatt (az 1.3. definíció alapján ez nem teljesül)

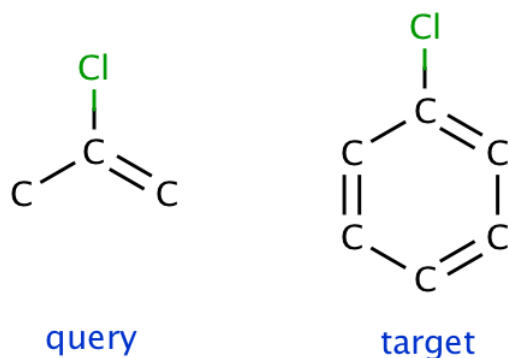
Ezen jelenségek helyes vizsgálata is alapvető fontosságú a gyakorlati alkalmazásokban. A ChemAxon Kft. MolSearch [20] implementációja képes megfelelően kezelni ezeket és még számos más kémiai jelenséget, emiatt lényegesen összetettebb és sok esetben szükségszerűen lassabb, mint a dolgozat keretében vizsgált implementációk.

A továbbiakban az egyes algoritmusok implementálása során hozott döntésekkel foglalkozunk. A molekulagráfok ún. SMILES formátumban (1.2. alfejezet, 6. oldal) álltak rendelkezésünkre.

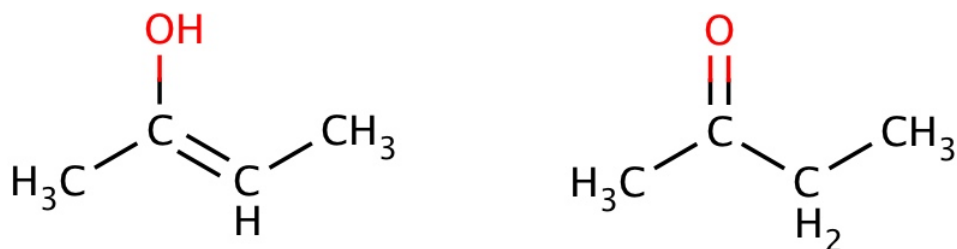
### 2.4.2. Heurisztikák alkalmazása

A részstruktúra-kereső implementációk futásideje jelentősen csökkent abban az esetben, ha a következő heurisztikát alkalmaztuk. A query molekula minden egyes  $u$  atomjához tartsuk nyilván – amennyiben van neki – a szülő atomját [14], melyet a későbbiekben  $parent(u)$ -val jelölünk.

**2.5. Definíció** Ha az  $u_i$  atomnak van olyan  $u_j$  szomszédja, melyet a részstruktúra-kereső algoritmus előbb illeszt le, akkor ezen szomszédok közül a legkisebb sorszámú atom legyen  $parent(u_i)$ .



2.16. ábra. Példa arra, hogy a target molekula nem tartalmazza részstruktúraként a query molekulát az aromás gyűrűszerkezetek kémiai értelmezése miatt (habár az 1.3. definíció alapján ez teljesül)

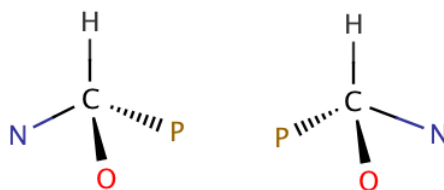


2.17. ábra. Példa egy molekula két különböző tautomerjére

A definícióból következik, hogy a query molekula minden összefüggő komponensében pontosan egy atomnak (amelyiket először illesztjük le az adott komponensből) nem lesz szülője. Abban az esetben, ha az  $u_i$  atom képét keressük, elegendő megvizsgálni, hogy  $f(\text{parent}(u_i))$  szomszédai között van-e olyan atom, melyet megfeleltethetünk  $u_i$ -nek.

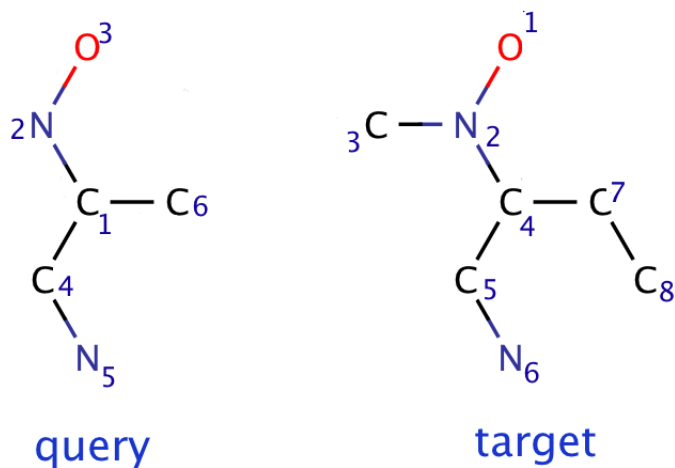
## 2.9. Példa

Legyen a részstruktúra-keresés inputja a 2.19. ábrán látható query és target molekula. Tegyük fel, hogy az aktuális parciális izomorfizmus:  $f(u_1) = v_4$  és  $f(u_2) = v_2$ . Az ezen állapothoz tartozó szülő-kapcsolatokat tekinthetünk meg a 2.20. ábrán. A szülő-gyerek kapcsolatokat a szülő atom felé mutató zöld nyilakkal jelöltük.

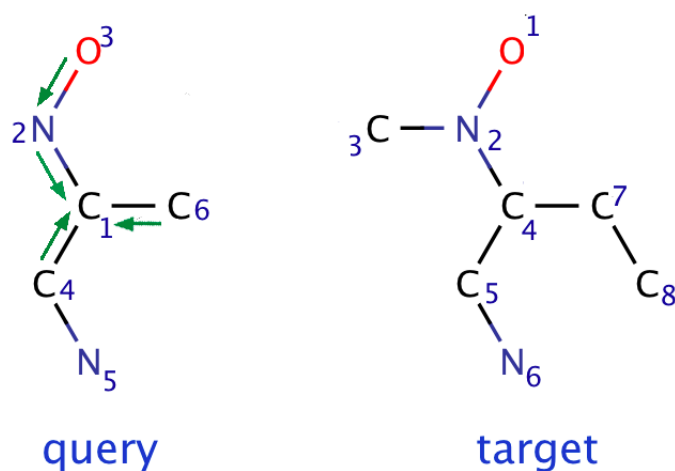


2.18. ábra. Példa két különböző sztereoizomer molekulára

A következő lépésben az  $u_3$  atom képét keressük. Mivel  $\text{parent}(u_3) = u_2$  atom, ezért  $u_3$  lehetséges képe az  $f(u_2) = v_2$  atomnak szomszédja. Emiatt csak a  $v_1$  és  $v_3$  atomot kell megvizsgálni.



2.19. ábra. A részstruktúra-kereső algoritmus egy lehetséges input query-target molekulapárja



2.20. ábra. Szülő csúcsok alkalmazása

A részstruktúra-kereső algoritmusok esetén megvizsgáltuk, hogy miképpen módosul a futásidő abban az esetben, ha az ismertett algoritmusokat nem rekurzív, hanem iteratív módon implementáljuk. Az iteratív implementáció átlagosan 10%-kal gyorsabbnak bizonyult, mint a rekurzív változat.

Az Ullmann- és a VF2 algoritmus futásidejének mérésekor megvizsgáltuk, hogy miként változik az első és az összes részgráf-izomorfizmus megtalálásához szükséges idő abban az esetben, ha a query molekula atomsorrendjét a 2.3. alfejezetben (21. oldal) tárgyalt várhatóan optimális atomsorrendet meghatározó algoritmus alapján

megváltoztatjuk. A várhatóan optimális atomsorrendet meghatározó algoritmus kis módosítással az új atomsorrend alapján képes a szülő atomok meghatározására is. Elmondható, hogy a futásidő mind az új atomsorrend, mind a szülő atomok alkalmazása esetén az összes tesztkészleten nagymértékben csökkent.

Az Ullmann-algoritmus futásidejét az összes tesztkészlet esetén kb. 15-20%-kal sikerült csökkenteni az alábbi egyszerű észrevétel alapján. Az algoritmus futása során a keresési fa  $i$ . szintjét a query molekula  $i$ . atomja alapján építjük fel. Ekkor a mátrix első  $i - 1$  sorában soronként pontosan egy *igaz* érték található. Az aktuális parciális izomorfizmust az  $f(u_i) = v_j$  megfeleltetéssel bővítve a mátrix  $i$ . sorában, valamint a  $j$ . oszlopában lévő elemeket az  $(i, j)$ . elem kivételével *hamis*-ra változtatjuk, ezért az állapotmentést követően a mátrix ritkítását elegendő az  $i + 1$ . sortól vizsgálni.

Az Ullmann-algoritmus futásidejét kb. 10%-kal és memóriaigényét egy nagyságrenddel csökkentettük az alábbi heurisztika alkalmazásával. Az algoritmus futása során minden egyes állapotmentés során nem az egész mátrixot tároljuk el, hanem a mátrix azon indexpozícióit jegyezzük fel, ahol a mátrix elemét *igaz*-ról *hamis*-ra változtattuk. Mivel a mátrixnak összesen  $n_1 \cdot n_2$  eleme van, és állapotmentés során elemet nem változtatunk *hamis*-ról *igaz*-ra, ezért egyszerre legfeljebb  $n_1 \cdot n_2$  indexpozíció tárolására van szükség. Ennek következtében az Ullmann-algoritmus memóriaigénye  $O(n^3)$ -ról  $O(n^2)$ -re csökkent.

Az Ullmann-algoritmus esetén elmondhatjuk, hogy érdemes először kiszámolni a várhatóan optimális atomsorrendet és a szülő atomokat, majd az eredeti algoritmust a korábban ismertetett – a mátrix ritkítására és az állapotmentésre vonatkozó – heurisztikákkal módosítani.

A VF2 algoritmus esetén észrevettük, hogy abban az esetben, ha lerögzítjük az atomok sorrendjét a query molekulában, akkor a keresési fa  $i$ . szintjét mindig ugyanazon atom szerint építjük fel. A  $Jelöltek(s)$  kiszámításának kétféle esetre vonatkozó képletét elsőre tekintve (16. oldal) ez nem tűnik nyilvánvalónak, azonban a két esetet külön kezelve könnyen igazolható ezen megállapítás. Emiatt a  $Jelöltek(s)$  meghatározása során nem szükséges a parciális izomorfizmus minden egyes bővítése esetén minimumot számolni, hanem elegendő egyszer, a legelején ezen sorrendet meghatározni. Elmondható továbbá, hogy a korábban ismertetett várhatóan optimális atomsorrendet meghatározó algoritmus által kiszámolt sorrendet nem szükséges átrendezni, hiszen teljesíti a szükséges feltételt. (Nevezetesen, ha  $N(G_1(s)) \setminus G_1(s) \neq \emptyset$ , akkor  $N(G_1(s)) \setminus (G_1(s))$ , különben pedig  $V_1 \setminus G_1(s)$  minimális sorszámú atomjának keressük a képét.)

Elmondhatjuk, hogy a VF2 algoritmus futásideje jelentősen, kb. 20-25%-kal csökkent akkor, ha a parciális izomorfizmus bővítése során a teljes  $Jelöltek(s)$  halmazt

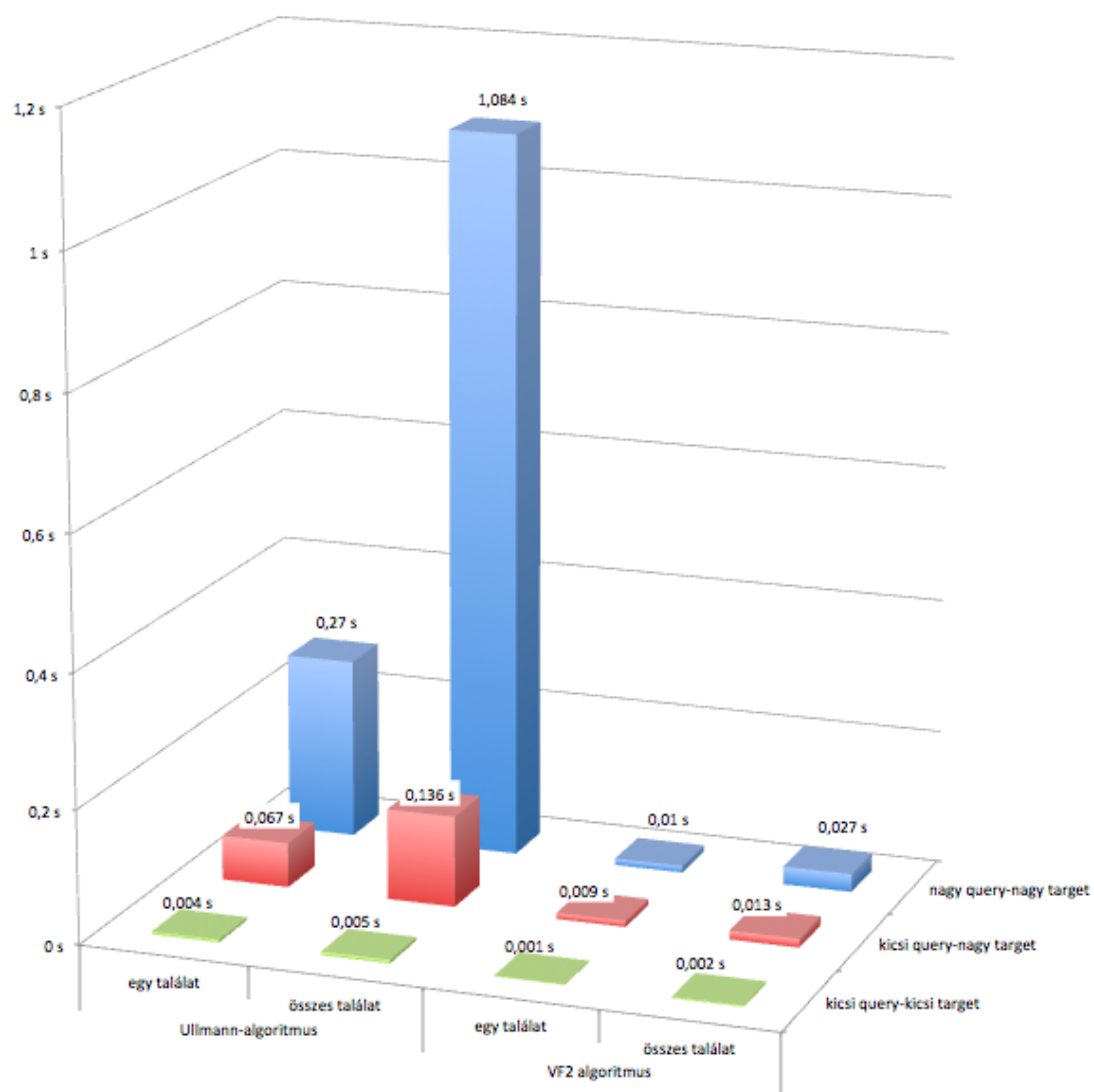
nem számítjuk ki előre és a memóriában sem tároljuk, hanem csak egy jelöltet határozunk meg, a vizsgálódást ezen jelölttel folytatjuk tovább, majd visszalépést követően egy újabb jelöltet határozunk meg, ha van ilyen, egyébként visszalépünk az eggyel korábbi szintre.

A VF2 algoritmus esetén a legjobb futásidőt minden esetben (kicsi query - kicsi target, kicsi query - nagy target, nagy query - nagy target) akkor kaptuk, ha először meghatároztuk a query molekula várhatóan optimális atomsorrendjét és a szülő csúcsokat. Ezt követően az algoritmus során a *Jelöltek(s)*-t nem tároljuk, hanem szükség esetén – első részgráf-izomorfizmus keresése során nem feltétlen kell az összes jelölt – a következő jelöltet kiszámoljuk. Habár jó ötletnek tűnik az eredeti algoritmusban szereplő szomszédsági halmazok méretének összehasonlítása (2.6. ábra, 18. oldal), méréseink során azt tapasztaltuk, hogy ezen elgondolás a futásidőt átlagosan 20%-kal növelte, ezért ezen feltétel ellenőrzését célszerű mellőzni (az algoritmus helyességét nem módosítja).

Az Ullmann-algoritmus és a VF2 heurisztikákkal kiegészített változatait összehasonlítottuk. A VF2 méréseink során minden esetben (első vagy összes részgráf-izomorfizmus keresése, query-target molekulák különböző mérete) gyorsabbnak bizonyult, mint az Ullmann-algoritmus.

A heurisztikákkal kiegészített Ullmann-algoritmus és a VF2 futásidejét összehasonlítottuk egy ipari alkalmazás, a ChemAxon Kft. **MolSearch** részstruktúra-kereső alkalmazásának [20] futásidejével. Ezen mérésekhez felhasználtuk a ChemAxon Kft. Marvin és JChem könyvtárainak 5.11.4-es verzióját, melyekhez akadémiai licencet biztosítottak számunkra.

Elmondhatjuk, hogy mindhárom esetben az összes részgráf-izomorfizmus megtaláláshoz 1.5-3-szor annyi idő szükséges, mint ha csak az első részgráf-izomorfizmust keresnénk. Méréseink során azt tapasztaltuk, hogy a VF2 gyorsabb, mint az egyébként lényegesen gazdagabb funkcionálist nyújtó **MolSearch**. Az általunk implementált VF2 jelentősen gyorsabbnak bizonyult kicsi query - kicsi target molekulákon történő keresés esetén, mint a másik két alkalmazás. Ezen tesztmolekulák esetén az Ullmann-algoritmus is gyorsabb volt, mint a **MolSearch**. Kis query - nagy target tesztkészlet esetén az általunk implementált Ullmann-algoritmus és a **MolSearch** az első részgráf-izomorfizmust közel azonos idő alatt találja meg, azonban az összes találat keresése, valamint nagy query molekulák esetén a **MolSearch** gyorsabb. Az Ullmann- és a VF2 algoritmus különböző tesztkészleteken mért 1000 query-target molekulapárra eső átlagos futási idejét a 2.21. ábrán tekinthetjük meg.



2.21. ábra. Az Ullmann- és a VF2 algoritmus átlagos futásideje 1000 query-target molekulapárra a különböző tesztkészleteken

## 3. fejezet

# Összefoglalás

A dolgozatban az NP-teljes részgráf-izomorfia problémát vizsgáltuk kémiai molekulagráfokra. Először a problémát, majd néhány előszűrési módszert és a két leggyakrabban alkalmazott részgráf-izomorfizmus-kereső algoritmust, az Ullmann-algoritmust [35] és a VF2-t [13] tekintettük át. Ezután egy újabb, kifejezetten részstruktúra-keresésre publikált algoritmus, a QuickSI [34] egy olyan részével foglalkoztunk, amely a várhatóan optimális atomsorrendet határozza meg.

Ezt követően az implementált algoritmusok futási idejét hasonlítottuk össze több tesztkészleten, az első és az összes részgráf-izomorfizmus megtalálásához szükség időt is figyelembe véve. Az egyes algoritmusok esetén több heurisztikát is implementáltunk. Vizsgáltuk, hogy ezen heurisztikák együttes alkalmazása esetén hogyan módosul a kereséshez használt idő. Az Ullmann-algoritmus és a VF2 esetén meghatároztuk azon heurisztika-együttállásokat, melyek használata esetén méréseink során a legkisebb futásidőt kaptuk.

Összefoglalva elmondhatjuk, hogy az új heurisztikákkal az ELTE IK Programtervező informatikus BSc szakán készített szakdolgozatban [29] szereplő első részgráf-izomorfizmus megtalálásához szükséges időt az Ullmann-algoritmus és a VF2 esetén is jelentősen sikerült csökkenteni. Ezen felül a dolgozatban nem csak az első, hanem az összes részgráf-izomorfizmus megkereséséhez szükséges idővel is foglalkoztunk (a szakdolgozatban ez nem szerepelt).

Ezen két implementáció futásidejét összehasonlítottuk a ChemAxon Kft. bővebb funkcionalitást biztosító részstruktúra-kereső alkalmazásával, a MolSearch-csel [20]. Méréseink során a VF2 gyorsabbnak bizonyult a másik két alkalmazásnál. Az Ullmann-algoritmus és a MolSearch közül nem egyértelmű, hogy melyik a jobb. Az Ullmann-algoritmus kicsi query - kicsi target molekulák esetén, míg a MolSearch a nagy query - nagy target molekulák esetén volt gyorsabb.

# Irodalomjegyzék

- [1] Daylight theory.  
<http://www.daylight.com/dayhtml/doc/theory/theory.finger.html>.
- [2] Funkciós csoport. [http://hu.wikipedia.org/wiki/Funkciós\\_csoport](http://hu.wikipedia.org/wiki/Funkciós_csoport).
- [3] Graph isomorphism. [http://en.wikipedia.org/wiki/Graph\\_isomorphism](http://en.wikipedia.org/wiki/Graph_isomorphism).
- [4] Simplified molecular-input line-entry system. [http://en.wikipedia.org/wiki/Simplified\\_molecular-input\\_line-entry\\_system](http://en.wikipedia.org/wiki/Simplified_molecular-input_line-entry_system).
- [5] Subgraph isomorphism problem. [http://en.wikipedia.org/wiki/Subgraph\\_isomorphism\\_problem](http://en.wikipedia.org/wiki/Subgraph_isomorphism_problem).
- [6] Downloadable Structure Files of NCI Open Database Compounds, 2000.  
<http://cactus.nci.nih.gov/download/nci/>.
- [7] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, 1974.
- [8] Villányi Attila. *Kémia a kétszintű érettségire*. Kemavill Bt., 2009.
- [9] Nathan Brown. Chemoinformatics – an introduction for computer scientists. *ACM Computing Surveys*, pages 8:1–8:38, 2009.
- [10] Horst Bunke, Pasquale Foggia, C. Guidobaldi, Carlo Sansone, and Mario Vento. A Comparison of Algorithms for Maximum Common Subgraph on Randomly Connected Graphs. In *Proceedings of the Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*, pages 123–132. Springer-Verlag, 2002.
- [11] C. J. Colbourn. On testing isomorphism of permutation graphs. *Networks*, 11:13–21, 1982.

- [12] Donatello Conte, Pasquale Foggia, and Mario Vento. Challenging Complexity of Maximum Common Subgraph Detection Algorithms: A Performance Analysis of Three Algorithms on a Wide Database of Graphs. *Journal of Graph Algorithms and Applications*, 11:99–143, 2007.
- [13] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. An Improved Algorithm for Matching Large Graphs. In: *3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, Cuen*, pages 149–159, 2001.
- [14] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.
- [15] Thomas Engel. Basic overview of chemoinformatics. *Journal of Chemical Information and Modeling*, 46:2267–2277, 2006.
- [16] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.
- [17] J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, STOC '74, pages 172–184. ACM, 1974.
- [18] Fekete István. Gráfalgoritmusok, 2004. [http://people.inf.elte.hu/fekete/docs\\_2/grafalg/grafalg.pdf](http://people.inf.elte.hu/fekete/docs_2/grafalg/grafalg.pdf).
- [19] ChemAxon Kft. Aromaticity detection. <http://www.chemaxon.com/marvin/help/sci/aromatization-doc.html>.
- [20] ChemAxon Kft. MolSearch, 2012. [https://www.chemaxon.com/jchem/doc/user/query\\_searchtypes.html](https://www.chemaxon.com/jchem/doc/user/query_searchtypes.html), <http://www.chemaxon.com/jchem/doc/dev/java/api/chemaxon/sss/search/MolSearch.html>.
- [21] Evgeny B. Krissinel and Kim Henrick. Common subgraph isomorphism detection by backtracking search. *Software-Practice & Experience*, 34:591–607, May 2004.
- [22] Tichler Krisztián. Részgráf-izomorfizmus és kapcsolódó problémák a kémiai informatikában, 2011. Formális eszközök az informatikában szeminárium.
- [23] José Luis López-Presa and Antonio Fernández Anta. Fast Algorithm for Graph Isomorphism Testing. In *SEA*, pages 221–232, 2009.
- [24] George S. Lueker and Kellogg S. Booth. A Linear Time Algorithm for Deciding Interval Graph Isomorphism. *Journal of ACM*, 26:183–195, April 1979.

- [25] E. M. Luks. Isomorphism of Graphs of Bounded Valence Can Be Tested in Polynomial Time. *Journal of Computer and System Sciences*, 1982.
- [26] J. March. *Advanced organic chemistry: reactions, mechanisms, and structure*. Wiley, 6th edition, 2007.
- [27] Brendan D. McKay. Nauty, 1981. <http://cs.anu.edu.au/~bdm/nauty>.
- [28] H. L. Morgan. The Generation of a Unique Machine Description for Chemical Structures – A Technique Developed at Chemical Abstracts Service. *Journal of Chemical Documentation*, 5:107–113, 1965.
- [29] Vigula Mónika. Hatékony algoritmusok kémiai gráfok részgráf-izomorfia vizsgálatára, 2012. ELTE IK, Programtervező informatikus BSc, szakdolgozat, Algoritmusok és Alkalmazásaik Tanszék.
- [30] Vigula Mónika. A részgráf-izomorfia probléma adatbázisokban, 2012. ELTE TTK, Matematika BSc, szakdolgozat.
- [31] B. Nisius and J. Bajorath. Molecular fingerprint recombination: generating hybrid fingerprints for similarity searching from different fingerprint types. *ChemMedChem*, 4:1859–63, 2009.
- [32] Kovács Péter. Molekulakezelő-keretrendszer (Java keretprogram), 2011. <http://people.inf.elte.hu/kpeter/algmsc1.html>.
- [33] John W. Raymond and Peter Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design*, 16:521–533, 2002.
- [34] Haichuan Shang, Ying Zhang, Xuemin Lin, and Jeffrey Xu Yu. Taming Verification Hardness: An Efficient Algorithm for Testing Subgraph Isomorphism. *Proceedings of the VLDB Endowment*, pages 364–375, 2008.
- [35] J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the Association for Computing Machinery*, 23:31–42, 1976.
- [36] David Weininger. SMILES, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Modeling*, pages 31–36, 1988.
- [37] Katona Gyula Y., Recski András, and Szabó Csaba. *A számítástudomány alapjai*. Typotex kiadó, 2006.

- [38] Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: on approximating graph edit distance. *Proceedings of the VLDB Endowment*, pages 25–36, 2009.