Logikai feladványok megoldása korlátprogramozással

A minta dolgozat alcíme

Szerzők: Papp Ádám, Sós Nikolett

Mérnökinformatika BSc., I. évfolyam

Témavezető: Ősz Olivér, doktorandusz

Széchenyi István Egyetem, GIVK, Informatika Tanszék

2018

Kivonat

A vizsgált feladatok az „Einstein fejtörője” néven ismert logikai feladvány, és annak különböző változatai. Ezekben a feladatokban adottak bizonyos személyek vagy objektumok, ezeknek néhány tulajdonságuk, melyek adott értékeket vehetnek fel. A feladatok hasonlítanak a szakirodalomban hozzárendelési feladatként ismert feladatosztályhoz, azzal a különbséggel, hogy nem az optimális hozzárendelést keressük egy adott szempont szerint, hanem speciális korlátozások vannak megadva a hozzárendelésre, és olyan megoldást keresünk, ami ezeket a feltételeket kielégíti. Míg a hozzárendelési feladatok megoldására a szakirodalomban léteznek hatékony algoritmusok, a logikai feladványok, különösen a korlátozások leírása valamilyen általánosabb modellezési módszert igényel.

Az általunk választott módszer a korlátprogramozás (constraint programming), ami egy modellezési és egy megoldási módszertan is egyben. Az utóbbi néhány évben egyre elterjedtebbé vált a korlátprogramozás használata különböző optimalizálási és kielégíthetőségi feladatok megoldásában.

Munkánk során megvizsgáltuk a logikai feladványok szerkezetét, összegyűjtöttük a korlátozások fajtáit. Különböző módokon modelleztük a korlátozásokat, és összehasonlítottuk őket megoldási teljesítmény szempontjából. Többféle megoldó szoftver teljesítményét is összevetettük egymással, és azonosítottuk, hogy az egyes megoldók hatékonyságát hogyan befolyásolta a használt modellezési módszer.

**Kulcsszavak**: logikai fejtörők, korlátprogramozás, hozzárendelési feladat

Tartalomjegyzék

[1. Bevezetés 1](#_Toc507505129)

[1.1. Korlátprogramozás bemutatása 1](#_Toc507505130)

[1.2. Logikai feladványok bemutatása 3](#_Toc507505131)

[2. Feladatok modellezése korlátprogramozással 5](#_Toc507505132)

**[2.1.](#_Toc507505133)****[Gardens](#_Toc507505133)** [5](#_Toc507505133)

[**2.2.** **Zebra feladatok** 9](#_Toc507505134)

[**2.3.** **Megoldók összehasonlítása** 9](#_Toc507505135)

[3. Redundáns megkötések megkeresése 9](#_Toc507505136)

[4. Összefoglalás (Címsor1 fejezet) 9](#_Toc507505137)

[Irodalomjegyzék 11](#_Toc507505138)

# Bevezetés

## Korlátprogramozás bemutatása

A feladatunkat úgynevezett korlátprogramozással oldottuk meg. Ennek a módszernek a legkorábbi verziója – a Sketchpad – 1963-ra tehető és [Ivan Sutherland](https://en.wikipedia.org/wiki/Ivan_Sutherland) nevéhez fűződik. Az 1980-as évektől egyre keresettebb lett, és mivel a logikai programozást szerették volna kiterjeszteni, így sok helyen korlát-logikai programozásként hivatkoznak rá. Az első praktikus verziókat – amiket üzleti célokra alkottak és már eladásra is bocsájtottak ­– az 1990-es években készítették. [1]

A projektünkben a MiniZinc nevű grafikus szerkesztő programot használtuk – magát a nyelvet is így hívják –, aminek a fordítója az mzn2fzn, amely FlatZinc-re fordítja a MiniZinc modellt. Ezt a legtöbb megoldó által támogatott formátumú fájlt adja tovább a megoldónak, ami végül kiadja a megoldást. Az egyik legnépszerűbb megoldó a Gecode, melynek fő alkotója Christian Schulte. A munkát 2002-ben kezdték meg, 2005 decemberében adták ki az első verziót, és onnantól kezdve több évben is aranyérmes lett a kategóriájában. [2] [3] Ezen kívül több megoldóval is képes együttműködni a MiniZinc, ilyen például a Gurobi, a Chuffed, és a CBC.

A megoldó működését legkönnyebben egy feladaton keresztül lehet szemléltetni. A feladványt a szakirodalomban „négyszín tételként” szokták említeni. Adottak bizonyos országok és ezeket úgy kell kiszínezni adott számú színnel, hogy a szomszédos területek ne legyenek azonosak.

A modell elején megadjuk hány színt szeretnénk használni és külön megjegyezzük, hogy a megoldás során is ezeket vegyék fel az egységek. A szomszédsági mátrixot – ami egyben a feltételek listája is itt esetünkben – kikötésekben adjuk meg. A legprimitívebb mód erre az, hogy leírjuk páronként a szomszédokat, amiknek más és más értéket kell felvenniük, azaz két egymás mellett lévő rész nem lehet egyenlő tulajdonságú.

MiniZinc modell

Térkép színezés

**%adatok**

**int: maxSzin = 4;**

**%megadjuk milyen értékeket vehetnek fel**

**var 1..maxSzin: Belgium;**

**var 1..maxSzin: Dánia;**

**var 1..maxSzin: Franciaország;**

MiniZinc logó

**var 1..maxSzin: Németország;**

**var 1..maxSzin: Hollandia;**

**var 1..maxSzin: Luxemburg;**

**%kikötések**

**constraint Belgium != Franciaország;**

**constraint Belgium != Németország;**

**constraint Belgium != Hollandia;**

**constraint Belgium != Luxemburg;**

**constraint Dánia != Németország;**

**constraint Franciaország != Németország;**

Gecode logó

**constraint Franciaország != Luxemburg;**

**constraint Németország != Hollandia;**

**constraint Németország != Luxemburg;**

**solve satisfy;**

A megoldó úgy dolgozik, hogy kiválaszt egy változót – a példában egy országot – és beállítja egy lehetséges értékre – színre. Ezután a korlátozások alapján következtetve csökkenti a többi változó lehetséges értékkészletét. Ezt a műveletet propagációnak nevezzük. A példában ez azt jelenti, hogy a szomszédos területeknél kizárja azt a lehetőséget, amit már felhasználtunk, így már csak a megmaradt színekből választhat.

Ha a propagáció során egy változó lehetséges értékkészlete üressé válik, akkor a megoldó visszavonja a legutolsó értékadást, és az adott értéket kizárja a változó értékkészletéből, mert ellentmondáshoz vezet. Ezután egy másik értéket ad neki, vagy egy új változót választ ki.

Amikor minden változónak sikerült értéket adni, akkor az egy lehetséges megoldása a feladatnak. Természetesen nem csak egy megoldás létezhet, ha például a piros és a fekete színt felcseréljük egymással, akkor az már másik megoldásnak fog számítani. Ha több megoldásra vagyunk kíváncsiak, folytathatjuk a keresést az utolsó értékadás visszavonásával. A korlátprogramozásnak ez egy hatalmas előnye más módszerekhez képest, hogy az összes lehetséges megoldást kiadja nekünk, nem pedig csak egyet.

A módszer optimalizálásra is használható, ahol a megoldások értékét egy célfüggvény adja meg. A keresés során egy további korlátozást kell figyelembe venni: hogy a megoldás értéke az eddig megtalált megoldásoknál jobb legyen.

## Logikai feladványok bemutatása

A címben lévő logika szóról az első érdemleges információnk Arisztotelésztől származik. Fontos megjegyezni, hogy nem tartotta külön tudománynak, csupán egy eszközként tekintett rá más szakirányokhoz. Szerinte ennek az elsajátítása ugyanannyira nem „tudatos”, mint az anyanyelv megtanulása. [4]

A összes vizsgált feladatunk a logikára épül, ezért is kapták a „logikai feladvány” nevet. Mi részletesebben az „Einstein-féle” esetekkel dolgoztunk. Ezekben a feladatokban adottak személyek és hozzájuk több személyes tulajdonság vagy tárgy, melyek mindegyike csak egy bizonyos emberhez tartozik, de hogy melyik kihez, annak a meghatározása maga a feladat. A feladvány megad néhány állítást a személyekről és tulajdonságaikról, melyekből levezethető a helyes hozzárendelés.

A leghíresebb feladata a következő. Adva van öt különböző színű ház egymás mellett, melyekben más és más nemzetiségű lakó él. Mind az öt ház tulajdonosa egy bizonyos italt iszik, egy bizonyos márkájú cigarettát vesz, és egy adott háziállata van. Mindegyik tulajdonos más háziállatot tart, más márkájú cigarettát szív, és más italt fogyaszt. A mi dolgunk, hogy kitaláljuk a kikötések alapján, hogy mi a helyes megoldás. [5]

A hozzárendelési feladatok nagyon hasonlítanak az általunk vizsgált feladatokhoz, de sok dologban el is térnek tőlük. Az egyik talán legszembetűnőbb különbség, hogy ezek a modellek optimalizálásra lettek kitalálva, ahol valamilyen célfüggvény segítségével a legjobb megoldást keressük a sok lehetőségből. A kikötések itt egyáltalán nem szerepelnek, minden hozzárendelés egy lehetőség, maximum csak azt szabják meg, hogy nem lehet mindent mindenhez hozzárendelni. A legismertebb ilyen modell a magyar módszer. [6]

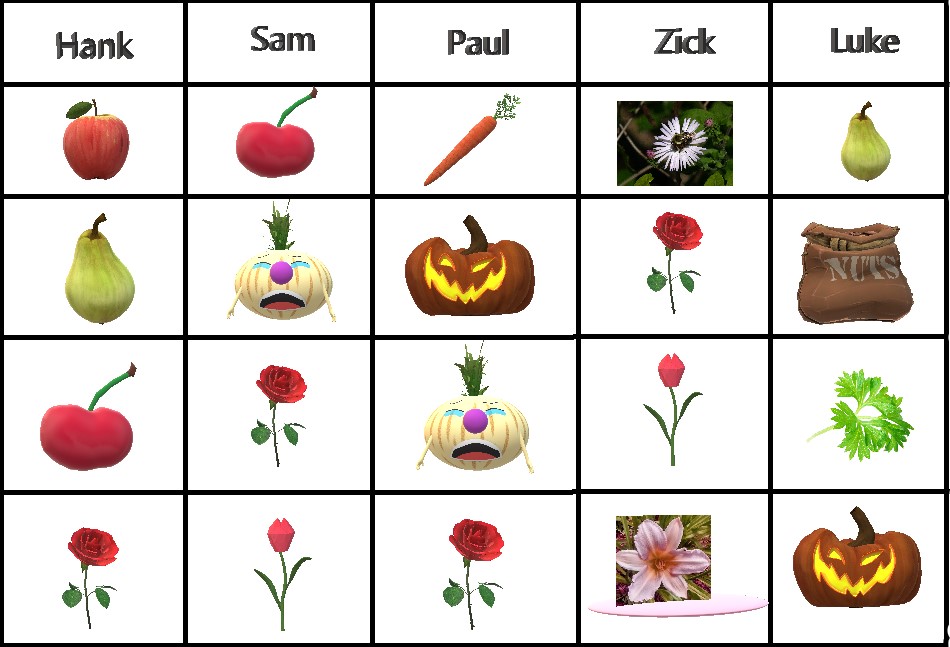
Ilyen feladatra példa az egyszerű munkamegosztás is a következőket figyelembe véve. Adott meghatározott számú gép és ugyanannyi független munka. Bármelyik gép bármelyik munkát képes elvégezni. Ismertek a gépek adott munkákra vonatkozó költségei. A feladat az, hogy minden géphez rendeljünk pontosan egy munkát úgy, hogy minden munka el legyen végezve és az összköltség minimális legyen. [7]

Einstein feladatát egyes források szerint az emberiség csupán 2%-a képes megoldani, ami nem tudományosan alátámasztott adat, de azt mindenképpen kifejezi, hogy bizony jó logikára van szükség hozzá. Ez is bizonyítja, hogy egyes feladatoknál ahhoz, hogy megkapjuk az eredményt sok időt és energiát kell belefektetni a munkába. Éppen ezért vizsgáltuk azt, hogy hogyan lehet az ilyen feladványok megoldását számítógépes segítséggel meghatározni.

# Feladatok modellezése korlátprogramozással

## **Gardens**

A munkánk során részletesen bizonyos előre „legyártott” feladatokat vizsgáltunk. Az első ilyen neve „Gardens”, azaz „Kertek”, amit az Einstein-féle logikai feladványokhoz sorolnak kategóriája szerint.

Adott öt barát, akiknek a kertjei egymás mellett helyezkednek el. Ezeken a területeken tizenkettő féle növényből termesztenek négyet-négyet fejenként. Azt is tudjuk, hogy ezekből a terményekből négy gyümölcs, négy zöldség és négy virág van. Azt, hogy ki melyik kertben dolgozik és azt, hogy mit tartalmaznak ezek, azt kikötések sora után tudjuk csak meg, aminek a végeredményét az alábbi kép szemlélteti.

Előszőr is az általunk használt parancsokat mutatnám be a kikötések fajtái alapján a megoldásunk első verzióján. Itt bináris mátrix segítségével dolgoztunk, azaz, ha megtalálható valami a kertben, akkor a helyére egyest, ha nem akkor nullát raktunk a táblázatba. A tulajokat egy külön tömbben tároljuk, melyeket összekötöttünk a kertekkel.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Hank** | **Sam** | **Paul** | **Zick** | **Luke** |
| **alma** | **1** | 0 | 0 | 0 | 0 |
| **körte** | **1** | 0 | 0 | 0 | **1** |
| **mogyoró** | 0 | 0 | 0 | 0 | **1** |
| **cseresznye** | **1** | **1** | 0 | 0 | 0 |
| **sárgarépa** | 0 | 0 | **1** | 0 | 0 |
| **petrezselyem** | 0 | 0 | 0 | 0 | **1** |
| **tök** | 0 | 0 | **1** | 0 | **1** |
| **hagyma** | 0 | **1** | **1** | 0 | 0 |
| **őszirózsa** | 0 | 0 | 0 | **1** | 0 |
| **rózsa** | **1** | **1** | **1** | **1** | 0 |
| **tulipán** | 0 | **1** | 0 | **1** | 0 |
| **liliom** | 0 | 0 | 0 | **1** | 0 |

A legfontosabb hogy minden utasítást „constraint”-el kell kezdeni. A szó maga is nagyon jól szemlélteti, hogy amit mögé írunk, azt úgymond „kikényszerítjük”, hogy tartsa be a munka során.

A képeken is jól látszik, hogy Zick például csak virágokat termel, míg Hank-nél vagy Sam-nél mind három fajta termény megtalálható, tehát nagyon különbözőek a kertek. Direkt ezekre az esetekre használtuk programunk során az „alldifferent”-et, ami kifejezi, hogy mint a kertek, mint a tulajok csak és kizárólag egyszer szerepelnek és eltérnek egymástól.

% egy embernek pontosan 1 kertje van(azaz minden tulaj különböző)

constraint alldifferent(tulaj);

A leírásban sok olyan kikötés szerepel, ami kijelenti, hogy mi hányszor szerepel. Ennek három speciális esetét különböztettük meg. Az első mikor konkrétan megadják hogy melyik kertben vagy melyik tulaj termeszti/nem termeszti az adott dolgot, de eme módon akár a tulajt is megadhatják. Ilyenkor egyszerűen csak megadjuk az adott cella értékét.

%Paul kertje a középső, liliom nélkül.

constraint termeszt[12,3]=0;

constraint tulaj[3]=4;

A második mikor csak annyit tudunk, hogy az adott növényt hányszor termeljük. Ilyen esetekben azt az előnyt használtuk ki, hogy mivel a táblázatunkban számok vannak, így lehet sima összeadás műveletet használni, így ilyenkor a sorösszegeket adtuk meg neki.

% Az alma csak egy kertben szerepel.

constraint forall(k in Kertek)(sum(k in Kertek)(termeszt[1,k])=1);

A harmadik pedig nagyon hasonlít a másodikhoz, de a szövegben valamilyen formában még jobban hangsúlyozza az állítást. Ilyen esetekben az „exactly”-t használtuk, aminek konkrét felépítési kritériuma van.

% Csak egy kertben található egy fajta növényből mind a négy darab.

constraint exactly(1, [bool2int(sum(z in Zoldsegek) (termeszt[z, k])=4 \/ sum(gy in Gyumolcsok) (termeszt[gy, k])=4 \/ sum(v in Viragok) (termeszt [v,k])=4) | k in Kertek], 1);

Végül, de nem utolsó sorban van még egy kikötési formánk, de ez nem volt elég hatékony megoldás a többi megoldásunknál már nem ezt használtuk.

% 21. Paul pontosan három fajta zöldséget termeszt.

constraint forall(t in Tulaj where tulaj[t]=4)

(sum(z in Zoldsegek)(termeszt[z,t])=3);

Ezt a „where”-es megoldást implikációra cseréltük le a projektünk további részeiben.

constraint forall(t in Tulaj)(tulaj[t]=4 -> sum(z in Zoldsegek)(termeszt[z,t])=3);

A programunk második verziója csupán az ilyen kikötések felépítésében tér el, de mégis sokkal hatékonyabb lett.

A harmadik verzióban már a meglévő mátrix mellé felvettünk még egyet a tulaj-kert kapcsolatokhoz. Az új mátrixot szintén egyesekkel és nullákkal töltöttük fel.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Luke** | **Sam** | **Hank** | **Paul** | **Zick** |
| **Kert1** | 0 | 0 | **1** | 0 | 0 |
| **Kert2** | 0 | **1** | 0 | 0 | 0 |
| **Kert3** | 0 | 0 | 0 | **1** | 0 |
| **Kert4** | 0 | 0 | 0 | 0 | **1** |
| **Kert5** | **1** | 0 | 0 | 0 | 0 |

A kikötések felépítésében itt nincs változás, tehát nem sok különbség van a két változat között.

A következő ötletünk az volt, hogy a kétdimenziós tömb helyett halmazokkal reprezentáljuk a megoldást. A tulajokat csak egy egyszerű tömbben tároltuk itt el. Két fontos új kifejezés is megjelent a kikötésekben ennél a munkánál. Az első a „card”, ami megadja a részhalmaz elemszámát.

% Mindenki négy különböző növényt termeszt.

constraint forall(k in Kertek)(card(termeszt[k])=4);

Ehhez a „card”-hoz használtunk még helyenként „intersect” parancsot is, ami kifejezi, hogy az éppen vizsgát halmaz részünk milyen másik halmazzal legyen keresztezve.

% Paul pontosan három fajta zöldséget termeszt.

constraint forall(t in Tulaj)(tulaj[t]=4 -> card (Zoldsegek intersect termeszt[t])=3);

A másik újdonság az „array\_union”, ami igazából egyesíti a kívánt dolgokat egymással.

% Minden egyes variáció minimum egy kertben megtalálható.

constraint card(array\_union(termeszt))=termenyek;

A Kertek utolsó verziójában ismét visszatértünk a mátrixos megoldáshoz, ám itt már nem bináris értékekkel töltöttük fel, hanem konkrétan a terményekkel. A leghasznosabb változtatás itt az volt, hogy a tömböket „enum” kulcsszóval töltöttük fel, így akár maga az objektum nevét, akár csak a sorszámát írtuk le a kikötésben, mind kettő variációt felismerte a rendszer. Egy másik fontos változtatás volt, hogy itt mivel nem tudtunk sorösszegeket számolni, így a „count”-ot kellett használnuk.

% Hank nem termel őszirózsát.

constraint forall(k in Kertek)(tulaj[k]=Hank -> count([termeszt[k, n] | n in Noveny], oszirozsa)=0);

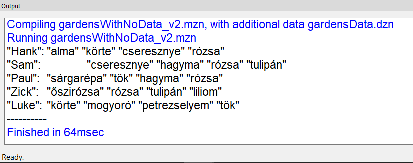
Egy plusz kikötést kellett még írnunk ehhez a változathoz az egy megoldás megtartása érdekében. Le kellett fixálnunk a termények sorrendjét, hogy ne adjon ki több lehetőséget is a megoldó, így a sorszámuk alapján növekvő sorrendben rakosgattuk be a növényeket a helyükre. Ennek a módszernek pontos szemléltetése volt a feladat elején bemutatott kép.

constraint forall(k in Kertek, n in 1..noveny-1)(termeszt[k,n] < termeszt[k,n+1]);

A kimeneteket egységesen minden verziónál formáztuk a jobb olvashatóság érdekében.

output [show(tulajNev[fix(tulaj[1])]) ++ ":\t"]++[show(termenyNev[t]) ++ " "

| t in Termenyek where fix(termeszt[t,1])=1]++["\n"];



"Output"

Ötből három változatban az adatokat külső fájlban tároltuk, így például, ha csak a növények neveit szeretnénk módosítani, nem kell az egész programunkat átírni hanem csak az adatfájlt. Ez a módszer is egy jó megoldás, ha munkát szeretnénk spórolni később.

## **Zebra feladatok**

## **Megoldók összehasonlítása**

# Redundáns megkötések megkeresése

# Összefoglalás (Címsor1 fejezet)

Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse.

Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse.

Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse. Az összefoglalás szövegtörzse.

Irodalomjegyzék

1. <http://www.constraint.org/en/history.html>
2. <http://www.gecode.org/presentations/Gecode%202011.pdf>
3. <https://en.wikipedia.org/wiki/Gecode>
4. <https://hu.wikipedia.org/wiki/A_logika_t%C3%B6rt%C3%A9nete>
5. <http://www.origo.hu/tudomany/20150911-einstein-fejtoro-lewis-carrol-ket-szazalek.html>
6. <http://www.tankonyvtar.hu/hu/tartalom/tamop425/2011-0001-526_ronyai_algoritmusok/ch06s07.html>
7. http://ait.iit.uni-miskolc.hu/~kulcsar/VirtVall\_2013\_14\_1f/h03/VE\_KGy\_2013\_gy\_03\_Magyar\_modszer\_v4.pdf