Kombinatorikai feladatok megoldása SAT-solverrel

Önálló projekt, szakmai gyakorlat II. dolgozat

Dankó Dorottya
Témavezető: Damásdi Gábor

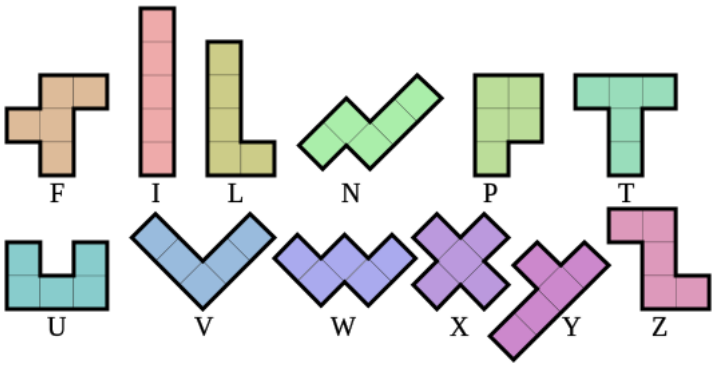
2023. május 26.

1. Bevezetés

A félévi munkám fő célja az volt, hogy kombinatorikai problémákat vezessek vissza a SAT-feladatra és Python programnyelvben implementált SAT-solverekkel keressek rájuk megoldást. Ebben a félévben csempézési feladatokkal foglalkoztam, azon belül a pentominó feladat variánsaival.

2. A pentominó feladat

A pentominó egy régi kirakósjáték. Adott 12 db 5-rendű poliominó, vagyis öt egységnégyzetből kirakható sokszög úgy, hogy az egységnégyzetek teljes oldallal illeszkedhetnek egymáshoz. A játék célja, hogy egy $n \times m$ méretű négyzetrácsot lefedjünk úgy, hogy minden darabot pontosan egyszer használunk fel, az alakzatok ne fedjék egymást és ne maradjon fedetlen cella. A megoldás létezésének triviális szükséges feltétele, hogy $n \times m = 60$ teljesüljön. A pentominó darabok tükrözése és a forgatása megengedett. Az alábbi ábrán látható a klasszikus pentominó szett.



1. ábra. A pentominó szett

2.1. A feladat története

A pentominó feladat először 1907-ben jelent meg Henry Dudeney The Canterbury Puzzles and Other Curious Problems című könyvében, bár ekkor még nem ez volt a neve. Először Solomon W. Golomb definiálta formálisan, aki a Polyominoes: Puzzles, Patterns, Problems, and Packings című könyvében írt a puzzle-ről 1965-ben. A pentominó szó és az 1. Ábrán látható egybetűs elnevezések is az ő nevéhez fűződnek.

2.2. Ismert eredmények

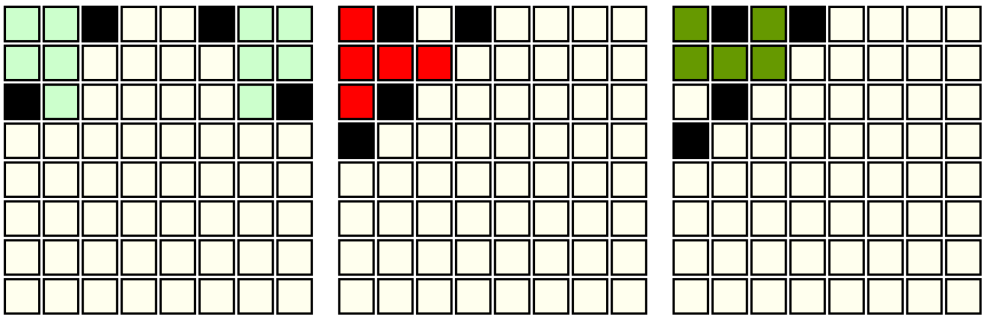
A pentominó feladat megoldásai már ismertek. Két megoldást akkor tekintünk különbözőnek, ha az egyik nem kapható meg a másik teljes lefedett téglalapjának elforgatásával és tükrözésével. Az adott méretű lefedendő téglalapokra a különböző megoldások száma az alábbi táblázatban látható.

Téglalap mérete	Különböző megoldások száma
6×10	2339
5×12	1010
4×15	368
3×20	2

1. táblázat. A pentominó feladatok különböző megoldásainak száma

A 6×10-es esetet Colin Brian Haselgrove és Jenifer Haselgrove oldotta meg 1960-ban [1].

Érdekes kérdés az is, hogy a 8×8-as sakktábla négy mezejét kihagyva a kapott alakzat mikor fedhető le a pentominó készlettel. A középső négy mező kihagyásával kapott esetet Dana S. Scott oldotta meg 1958-ban [2]. Ekkor 65 különböző megoldás van. Scott programja az egyik első volt a világon, ami a backtracking módszert alkalmazta. A sakktábla fedési probléma két egyszerű eseten kívül mindig megoldható. Az egyik megoldhatatlan eset, ha a sarkok környékén úgy választjuk az üresen maradó mezőket, hogy két P pentominóra is szükség legyen. A másik esetben a T vagy U pentominót kényszerítjük az egyik sarokba úgy, hogy ezzel egy újabb mezőt lefedhetetlenné tegyen. Ezeket szemlélteti az alábbi ábra.



Ma már léteznek nagyon hatékony algoritmusok a pentominó puzzle-re. Például Knuth Algorithm X nevű módszere jól használható csempézési feladatok megoldására [3].

3. SAT-solverek

A SAT-probléma elnevezése az angol satisfiability szóból ered. Adott egy logikai formula, kérdés, hogy létezik-e ennek olyan kiértékelése (a változóinak igaz-hamis értékadása), amire a formula értéke igaz lesz. Ez a probléma NP-teljes még akkor is, ha a logikai formula alakjára jelentős megszorításokat teszünk (például konjunktív normálforma alakjában minden klózban legfeljebb három változó szerepeljen, ezt nevezzük 3-SAT problémának).

Többféle algoritmust implementáltak már különböző programnyelvekben a SAT-problémára, ezeket SAT-solvereknek nevezzük. Pythonban a PySAT [4] könyvtárat arra a célra készítették, hogy logikai formulákkal és a SAT-problémával könnyen foglalkozhasson a felhasználó. Ebben többféle SAT-solver is rendelkezésünkre áll, a projektben a Glucose3 nevűt használtam. A solver inputja egy konjunktív normálforma, outputja egy igazságérték. Ha kiértékelhető a normálforma, akkor visszaadhat egy vagy akár az összes jó kiértékelést is.

4. A feladat megoldása

4.1. A probléma átfogalmazása SAT-feladatra

A félév során egy programot fejlesztettem Python nyelven, ami a pentominó feladatot a Glucose3 SAT-solver használatával oldja meg. A munkám legfontosabb része az volt, hogy a csempézési feladatból felépítsek egy konjunktív normálformát, aminek egy igaz kiértékelése megfelel a pentominó egy megengedett lerakásának.

Minden alakzatot egy 5×5 méretű, 0-1 elemekből álló mátrixban tároltam el. Ez után minden alakzatra eltároltam a 90°-os forgatásokkal és tengelyes tükrözésekkel kapható különböző lehelyezéseket, amiknek a száma az alakzat szimmetriáitól függően eltérő lehet. A programom inputjának része az n és az m számok, amik rendre a lefedendő téglalap sorainak és oszlopainak számát jelölik.

A SAT-feladat logikai változói a következők: $x_{i,j,s,o}$ „igaz” pontosan akkor, ha az i-edik sor j-edik oszlop a bal felső sarka annak az 5×5-ös négyzetnek, amiben az s-edik sorszámú alakzat az o-adik különböző orientációban helyezkedik el. Vegyük észre, hogy itt i és j nem feltétlenül esik a lefedendő négyzetbe, mert lehet, hogy abban a sorban és oszlopban az alakzatunkhoz tartozó mátrix minden eleme 0. Ezért egy $(n + 4) \times (m + 4)$ méretű téglalapon dolgoztam, amiben a megoldás a jobb alsó sarokba szorított $n \times m$-es téglalapon helyezkedik el. Az elkódoláshoz a pysat.card részkönyvtár CNF osztályát használtam.

Ahhoz, hogy megengedett fedését kapjuk a téglalapnak, a következő feltételeket kellett felírni konjunktív normálforma alakban:

- minden alakzat a lefedendő téglalapon belül helyezkedik el (nem „lóg ki”),
- minden alakzat pontosan egyszer fordul elő a fedésben,
- minden cellát pontosan egy alakzat fed.

Az első feltételhez létrehoztam egy ellenőrző függvényt, aminek bemeneti paraméterei a lefedendő téglalap méretei és a változó négy paramétere (i, j, s, o), outputja pedig pontosan akkor

igaz, ha van olyan 1-eleme az alakzathoz tartozó mátrixnak ebben a lehelyezésben, ami a lefedendő téglalapon kívül esik. Ezt a függvényt meghívtam minden i, j, s, o-ra, és ha azt kaptam, hogy az alakzat nincs teljesen a téglalapon belül, akkor a lehelyezésnek nem is hoztam létre logikai változót.

Az utóbbi két feltétel felírásához létrehoztam két könyvtárat. A `same_shapes` kulcsai az alakzatok sorszámai voltak, értékei pedig azoknak a változóknak a sorszámai, amik ahhoz az alakzathoz tartoznak. A `covering_shapes` könyvtár kulcsai a téglalap mezői, értékei pedig azok a változók, amik fedik azt. Ezek létrehozása után már csak azokat a normálformákat kellett legenerálni, amik azt jelentik, hogy a kigyűjtött logikai változók közül pontosan egy legyen igaz. A pysat.card CardEnc nevű absztrakt osztályához tartozó .equals() metódus pont erre való.

A Scott [2] cikkében szereplő feladatra is alkalmaztam a programomat. Ekkor a lefedendő téglalap mérete 8×8, és négy mezőt nem fedünk le. Scott azzal az esettel foglalkozott, amikor ez a négy mező pont középen helyezkedik el, az én programom paraméterként veszi be a négy cellát. Ezt úgy valósítottam meg, hogy ezekre a cellákra ahelyett a feltétel helyett, hogy pontosan egy alakzat fedje őket, azt adtam hozzá a normálformához, hogy maradjanak fedetlenek.

4.2. A különböző megoldások kiszűrése

Ahhoz, hogy a program az összes megoldás megkeresésekor tényleg különböző fedéseket találjon, több trükköt is be kellett vetni. Ahogy azt már a 2.2 Fejezetben leírtam, két megoldás nem számít különbözőnek, ha az egyik megkapható a másik megoldás teljes téglalapjának tükrözéseivel és forgatásaival. Ezt úgy követeltem meg, hogy kiválasztottam a P pentominót, aminek nincs semmilyen forgás- vagy tengelyes szimmetriája, és ezt lerögzítettem. Pontosabban, miután minden alakzathoz eltároltam, hogy annak mik a különböző lehelyezései, ha a lefedendő téglalap nem négyzet volt, akkor a P listájába csak kettőt raktam a nyolc helyett. Ez a két alakzat egymásnak 90°-os elforgatottjai. Azért volt szükség mindkettőre, mert ekkor a lefedendő téglalapoknak nincs 90°-os forgásszimmetriája. Abban az esetben, amikor négyzet alakú volt az alap feladat, akkor elég volt a P-nek egy orientációja.

A másik ok, ami miatt több egyforma megoldást is kaphattunk, a CardEnc.equals() függvény használata volt. Ahogy azt a 4.1 Fejezetben írtam, ez a metódus azt a feltételt kódolja el, hogy az adott változók közül pontosan egy legyen igaz. Ehhez segédváltozókat vesz fel, emiatt a végső normálformánkban több olyan változó is szerepelt, ami a feladat szempontjából nem lényeges. Tehát, ha két kiértékelés csak a segédváltozókban különbözött, az ugyanazt a fedést adta. Ezt a problémát úgy hidaltam át, hogy minden új megoldás megtalálása után a kiértékelésben „igaz”-ként szereplő, számunkra lényeges változókból létrehoztam egy új klózt, amiben ezek mindegyike negálva szerepel.

4.3. A megoldás kiszámítása és kirajzolása

A megoldás kiszámításához létrehoztam egy `solve` függvényt, aminek bemeneti paraméterei a konjunktív normálforma és egy `eachsol` nevű logikai változó, ami igaz, ha az összes megoldást szeretnénk megkapni, hamis, ha csak egyet. A függvény meghívja a Glucose3 SAT-solvert és a megoldásban „igaz” értékkel szereplő változókból létrehoz egy $n \times m$ méretű mátrixot, amiben minden alakzathoz tartozik egy különböző pozitív egész szám. Ha tehát például az L

nevű pentominó kapta a kettes számot, akkor a megoldás mátrixban öt darab kettes elem lesz, és ezek L alakban helyezkednek el. A vizuális megjelenítéshez a matplotlib.pyplot könyvtárának .imshow() nevű függvényét használtam. A `solve` függvény visszatérési értéke a talált különböző megoldások száma, ha az összeset kértük, különben pedig 1.

5. Eredmények

A program a félév során jelentős javításokon és gyorsításokon esett át. Az első működő változat körülbelül 30 perc alatt talált egy jó megoldást a 6×10-es téglalap fedésére, a mostani már kevesebb, mint két másodperc alatt. Az összes eredmény megtalálása viszont még mindig több percet vesz igénybe azoknál az eseteknél, ahol sok van. A program mind a négy lefedhető téglalap méretre megtalálta az összes megoldást, a `solve` függvény pontosan azzal az értékkel tért vissza minden esetben, ami az 1. táblázatban szerepel.

Az alábbi ábrákon mutatok egy-egy példát mind a négy megoldható téglalapfedésre, amiket a program kirajzolt.

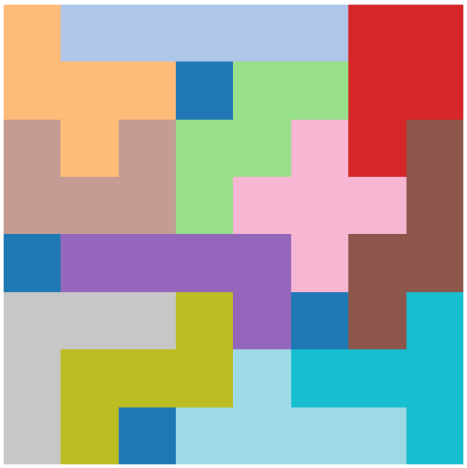
2. ábra. A 6×10-es téglalap

3. ábra. Az 5×12-es téglalap

4. ábra. A 4×15-ös téglalap

5. ábra. A 3×20-as téglalap

A 8×8-as feladatra is jól működött a program. Scott [2] eredményét is sikerült reprodukálni, a négy középső mező kihagyásával 65 különböző megoldást talált. A 8×8-as feladat futási eredményei alább láthatók. A 6. ábrán a fedetlenül hagyott négy cellát véletlenszerűen választottam, a 7. ábrán pedig a középső négyet hagytam ki.



6. ábra. Négy random kihagyott cella

7. ábra. A középső négy cella kihagyása

Hivatkozások

[1] *C. B. Haselgrove* and *Jenifer Haselgrove*: A Computer Program for Pentominoes. Eureka. 23: 16–18. (October 1960)

[2] *Dana S. Scott*: Programming a combinatorial puzzle. Technical Report No. 1, Department of Electrical Engineering, Princeton University (1958)

[3] *Donald E. Knuth*: Dancing links. Millennial Perspectives in Computer Science. P159. 187. (2000)

[4] *A. Ignatiev* and *A. Morgado* and *J. Marques-Silva*: PySAT: A Python Toolkit for Prototyping with SAT Oracles. Theory and Applications of Satisfiability Testing – SAT 2018, 428–437. (2018)