Debreceni Egyetem
Informatika Kar

Mesterséges Intelligencia algoritmusok: A SAT probléma

Témavezető:
Dr. Nagy Benedek
Egyetemi adjunktus

Készítette:
Debreceni Béla
Programtervező-matematikus

Debrecen
2008

Tartalomjegyzék

1. Bevezetés

A SAT-probléma a mesterséges intelligencia kutatások egyik kiemelkedő területe, olyan témakörök alapja, mint az automatikus tételbizonyítás, VLSI áramkörök helyességének biztosítása, tudásalapú ellenőrzés és érvényesítés. Már 1971-ben bizonyította róla Stephan Cook, hogy ez egy *NP-teljes* bonyolultságú feladat. Maga a probléma nem más, mint egy ítéletlogikai állítás kielégíthetőségének a vizsgálata. Mai ismereteink szerint nem létezik algoritmus, mely a legrosszabb esetre, például, ha kielégíthetetlen a formulánk, polinomiális időben megoldaná a feladatot.

Minden olyan módszer, mely átlagos esetre megközelíti a gyors algoritmusok sebességét figyelmet érdemel, hiszen ha találunk a SAT-problémát átlagosan polinomiális időben megoldó algoritmust, akkor a feladat teljességéből kifolyóan sok más, hasonlóan nehéz feladatra is találtunk gyors megoldást. Egy ilyen új módszer kerül bemutatásra dolgozatomban, melynek alapja a mesterséges intelligencia algoritmusok egy kevésbé ismert területéről, az evolúció elvén működő genetikus algoritmus (a továbbiakban genTsat).

Összehasonlításul két ismert algoritmus is kidolgozásra kerül. Az ismertebb, általánosan használt teljes algoritmus a DPLL, mely a formula szemantikus fájában visszalépéses kereséssel igyekszik megoldást találni. A DPLL képes a kielégíthetetlen formulák azonosítására is, hiszen ha nem talál megoldást, akkor bejárja a teljes keresési fát. A második bemutatandó módszer a WalkSAT algoritmus, mely egy lokális keresést valósít meg, kiegészítve a mohó és sztochasztikus heurisztikák erejével. Mint általában a lokális keresési módszerek a WalkSAT sem tudja befejezni a futását, amíg nem talál megoldást, így ha a vizsgált formula nem kielégíthető nem tudjuk eldönteni vele a problémát.

A genTsat algoritmus egyik lehetséges előnye az, hogy egyszerre több megoldásjelöltet értékel ki, majd kiválasztva a legjobbat, továbbörökíti az értékes bitjeit a következő populációnak, így konvergálva a tényleges megoldás felé. További pozitívuma az algoritmusnak, ellentétben a másik két vizsgált módszerrel, hogy Boole formulákra működik, így elkerülhető a költséges átalakítás konjunktív normálformára. Mivel a genTsat is egyfajta lokális keresésen alapul, a WalkSAT-hoz hasonlóan, nem kielégíthető formulára nem tud dönteni. A populációk kiértékelése megadja annak a lehetőségét, hogy mérlegeljük a kialakult megoldásjelöltek jóságát, és úgymond megjósoljuk a formula kielégíthetőségét kudarc esetén is.

Fontos, hogy megismerjük és megértsük a feladat hátterét, az ítéletlogikai formulát, az ábrázolási lehetőségeket, a Boole formulát, a konjunktív normálformát, hiszen ezek segítségével könnyebb átlátni a bemutatásra kerülő módszereket, jobban megérthető az algoritmusok működése. Megismerhetjük a mesterséges intelligencia különböző probléma megoldó módszereit. Minden algoritmusra kidolgozásra kerül egy részletes példa, mely betekintést nyújt a választott adatszerkezetbe, és működésének rejtelmeibe.

Az algoritmusok implementálása C nyelven történik, mely programozási nyelv teljes szabadságot enged a kiválasztott adatmodell megvalósításában, de kellőképpen gyors és hatékony.

A következő fejezet a SAT probléma matematikai megfogalmazása, megismerjük a különböző ábrázolási módokat és megalapozzuk az algoritmusok adatmodelljeit.

2. A matematikai háttér

Ebben a fejezetben definiáljuk az ítéletlogikai formulát, a szerkezeti felépítését, a szemantikus modelljét. Bemutatásra kerül a konjunktív normálforma, a 3SAT formula, és ennek levezetése egy formulából.

2.1 Az ítéletlogika

Ez az egyik legegyszerűbb eszköztárral rendelkező formális logika és nyelve egy olyan nulladrendű nyelv, melynek ábécéje csak logikai szimbólumokból áll. Ítéletváltozókat, logikai összekötőjeleket és elválasztójeleket tartalmaz. A probléma definiálásakor, és az egyes algoritmusok adatszerkezetének a bemutatásánál az ítéletváltozókat jelöljük $X, Y, Z...$ betűkkel, majd a programozás során, a könnyebb kezelhetőség és nagyobb feladatok egyszerűbb feldolgozhatósága érdekében, természetes számokkal. Alkalmazhatóak a $\neg$ unér és a $\wedge, \vee, \supset$ binér logikai összekötőjelek, és elválasztójelként a (és) jelek szerepelnek majd.

V_v: jelölje az ítéletváltozók halmazát

V_0: az ítéletlogika nyelvének ábécéje

Definíció (1): Az **ítéletlogika nyelve** a V_0 ábécé feletti legszűkebb olyan tulajdonságú szóhalmaz, amelynek

1. V_v minden betűje egyúttal szava is,
2. ha S eleme a szóhalmaznak, akkor $\neg S$ is eleme, és
3. ha S és T elemei a szóhalmaznak és $\circ$ binér logikai összekötőjel, akkor $(S \circ T)$ is eleme a szóhalmaznak.

Definíció (2): $\mathcal{L}_0$ **szintaxisa**

1. Minden ítéletváltozó ítéletlogikai formula, ezeket a formulákat (atomi vagy) prímformuláknak is nevezzük.
2. Ha A ítéletlogikai formula, akkor $\neg A$ is az.
3. Ha A és B ítéletlogikai formulák, és $\circ$ binér logikai összekötőjel, akkor $(A \circ B)$ is ítéletlogikai formula.
4. Minden ítéletlogikai formula az 1-3. szabályok véges sokszori alkalmazásával áll elő.

Példa (1): Egy *ítéletlogikai formula*: $(\neg X \supset Y) \wedge (Z \wedge X)$

Ha egy formula $\neg A$ alakú, negációs, ha $(A \wedge B)$ alakú, konjunkciós, ha $(A \vee B)$ alakú, diszjunkciós, ha pedig $(A \supset B)$ alakú, akkor implikációs formulának nevezzük. Ezek az *összetett formulák.*

Definíció (3): egy ítéletlogikai formulában előforduló minden olyan összefüggő betűsorozatot, mely maga is ítéletlogikai formula **részformulának** nevezünk.

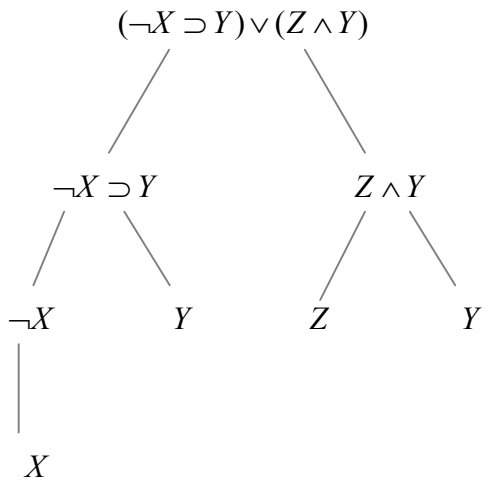
Definíció (4): (**közvetlen részformula**) $\mathcal{L}_0$-ban

1. egyetlen prímformulának sincs közvetlen részformulája,
2. a $\neg A$ egyetlen közvetlen részformulája az A formula,
3. az $(A \circ B)$ formula közvetlen részformulái az A és a B formulák.

Definíció (5): Egy C **formula szerkezeti fája** egy olyan véges rendezett fa, melynek csúcsai formulák,

1. gyökere C,
2. a $\neg A$ csúcsának pontosan egy gyermeke van, az A formula,
3. az $(A \circ B)$ csúcsának pontosan két gyermeke van, rendre az A és a B formulák,
4. levelei prímformulák.

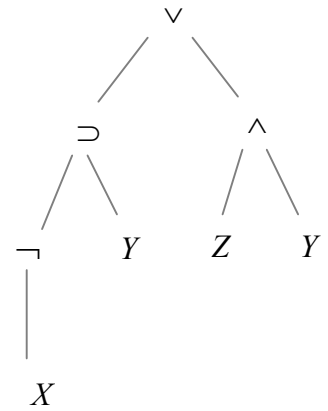
Példa (2): $(\neg X \supset Y) \vee (Z \wedge Y)$ formula *szerkezeti fája:*



1. ábra Formula szerkezeti fája

Megjegyzés: Az 1. ábrán látható fa szerkezet lesz az alapja a genTsat algoritmusnak, annyi módosítással, hogy a csúcsokon a részformulák helyett a részformulák fő logikai összekötő jelét ábrázoljuk, és az adott csúcs azt a részformulát jelzi, melynek ő a gyökere, mint részfa.

Példa (3): $(\neg X \supset Y) \vee (Z \wedge Y)$ formula *egyszerűsített szerkezeti fája:*



2. ábra Formula egyszerűsített szerkezeti fája

2.2 Az interpretáció

Egy ítéletváltozó lehet igaz vagy hamis, ennek az információnak a rögzítését nevezzük interpretációnak.

Definíció (6): $\mathcal{L}_0$ **interpretációján** egy $\mathcal{I} : V_V \to \{i, h\}$ függvényt értünk.

Definíció (7): ($\mathcal{L}_0$ **szemantikája**) $\mathcal{L}_0$-beli formulák $\mathcal{I}$ *interpretációbeli Boole-értékelése* a következő $\mathcal{B}_\mathcal{I} : \mathcal{L}_0 \to \{i, h\}$ függvény:

1. ha A prímformula, akkor $\mathcal{B}_\mathcal{I}(A)$ legyen $\mathcal{I}(A)$,
2. $\mathcal{B}_\mathcal{I}(\neg A)$ legyen $\neg\mathcal{B}_\mathcal{I}(A)$
3. $\mathcal{B}_\mathcal{I}(A \wedge B)$ legyen $\mathcal{B}_\mathcal{I}(A) \wedge \mathcal{B}_\mathcal{I}(B)$,
4. $\mathcal{B}_\mathcal{I}(A \vee B)$ legyen $\mathcal{B}_\mathcal{I}(A) \vee \mathcal{B}_\mathcal{I}(B)$,
5. $\mathcal{B}_\mathcal{I}(A \supset B)$ legyen $\mathcal{B}_\mathcal{I}(A) \supset \mathcal{B}_\mathcal{I}(B)$.

A 3. ábrán látható táblázat segítségével könnyen megérthetjük, hogyan is kapják a szemantikai értéküket az összetett formulák, és a negáció. A továbbiakban jelöljük az igaz értéket *1*-el, a hamis értéket *0*-val, vagyis, ha A igaz, akkor $\mathcal{B}_\mathcal{I}(A)$=1, ha A hamis, akkor $\mathcal{B}_\mathcal{I}(A)$=0.

A	B	$A \wedge B$	$A \vee B$	$A \supset B$	$\neg A$
1	1	1	1	1	0
1	0	0	1	0	0
0	1	0	1	1	1
0	0	0	0	1	1

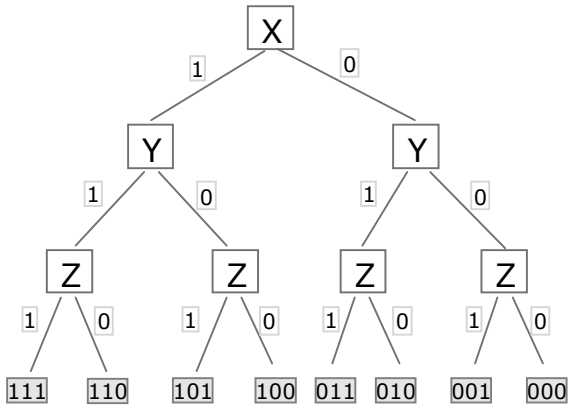
3. ábra A logikai jelek szemantikája

Egy formula igazságértéke csak a benne előforduló ítéletváltozók interpretálásától függ, meghatározásához elegendő csak a kérdéses formula változóihoz rendelt igazságértékeket

ismerni, azaz a formula egy interpretációját megadni. Egy n-változós formulának 2^n különböző interpretációja van.

Ábrázolhatjuk a formula különböző interpretációit **szemantikus fa** segítségével. Az ítéletváltozók egy rögzített sorrendjét nevezzük *bázisnak*. Felépítünk egy binárisfát úgy, hogy a bázis első eleme a fa csúcsa, a bal ág az ítéletváltozó igaz értéke, a jobb ág az ellentettje. A következő bázis elem lesz a két ágból induló részfa csúcsa. Iteráljuk a lépéseket, míg van báziselemünk A levélelemek értéke pedig az egyes ágak értékei a gyökér elemtől a levélig összefűzve.

Példa (4): X, Y, Z ítéletváltozókhoz tartozó *szemantikus fa*:



4. ábra Szemantikus fa

Bármely 3 változót tartalmazó ítéletlogikai formulának teljesen hasonló módon épül fel a szemantikus fája, a változók kiválasztásától függetlenül. A fának n változó esetén 2^n különböző levele lesz, melyek megadják a formula összes lehetséges interpretációját. Ebből következik, hogy minden eset megvizsgálása polinomiális időben, a bemenetek polinomiális függvényében, nem valósulhat meg.

Ez az ábrázolás az alapja a DPLL algoritmusnak. Rekurzívan bejárjuk a szemantikus fa minden ágát, és ellenőrizzük, hogy a kialakult interpretáció megoldja-e az eredeti feladatot.

2.3 Ítéletlogikai törvények:

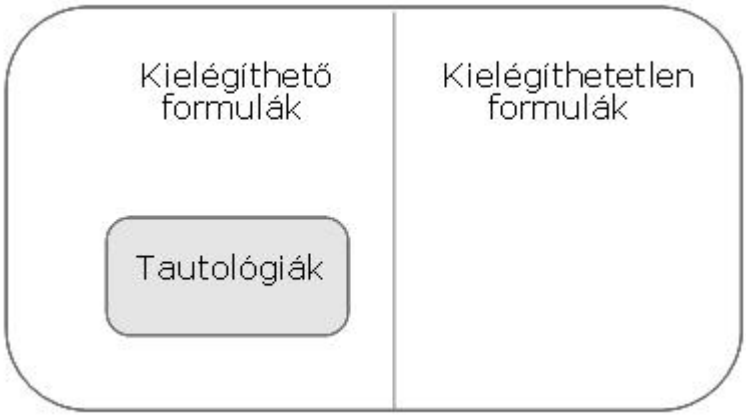
Definíció (8): Az $\mathcal{L}_0$ nyelv egy A **formulája kielégíthető**, ha van $\mathcal{L}_0$-nak olyan $\mathcal{I}$ interpretációja melyet behelyettesítve, az A formula igaz eredményt ad. Jelölje ezt: $\mathcal{I} \models_0 A$. Egy ilyen interpretációt A **modelljének** nevezzük. Ha nincs A-nak modellje, az A formulát kielégíthetetlennek mondjuk.

Példa (5): Tekintsük az $A = (\neg X \supset Y) \wedge (Z \wedge X)$ formulát. Az A egy kielégítő interpretációja és egyben modellje a következő behelyettesítés: $\mathcal{I}(A \mid X = 1, Y = 1, Z = 1) = 1$

Definíció (9): Az A formula **ítéletlogikai törvény** vagy másképpen **tautológia**, ha $\mathcal{L}_0$ minden interpretációjára $\mathcal{I} \models_0 A$.

Egy tautológiának minden interpretáció modellje. Tehát egy $\mathcal{L}_0$-beli formula lehet olyan, hogy nincs modellje, tehát *kielégíthetetlen*, ellenkező esetben van modellje, azaz *kielégíthető*, és ilyen esetben lehet, hogy minden interpretáció modellje, vagyis *tautológia*.

Az ítéletlogikai formulák szemantikai tulajdonságuk alapján az alábbi ábra szerint osztályozhatók:



5. ábra Ítéletlogikai formulák osztályozása

Definíció (10): Azt mondjuk, hogy az A és B ítéletlogikai formulák **tautologikusan ekvivalensek**, ha minden $\mathcal{I}$ interpretációban $\mathcal{B}_{\mathcal{I}}(A) = \mathcal{B}_{\mathcal{I}}(B)$. Jelölje ezt: $A \sim_0 B$.

Igazságtáblák segítségével könnyen eldönthető, hogy két formula tautologikusan ekvivalens-e egymással vagy sem. Ha a közös igazságtáblájukban a formulákhoz tartozó oszlopokban minden sorban ugyanaz az igazságérték található, a két formula tautologikusan ekvivalens. Két formula akkor és csak akkor tautologikusan ekvivalens, ha az általuk leírt logikai művelet ugyanaz. Meg kell jegyezni, hogy az igazságtáblák felépítése már nem ilyen egyszerű feladat, hiszen általában exponenciális idő szükséges hozzá.

Példa (6): Az $X \supset Y$ formula tautologikusan ekvivalens a $\neg X \vee Y$ formulával, amit a közös igazságtáblájuk mutat:

X	Y	$X \supset Y$	$\neg X \vee Y$
1	1	1	1
1	0	0	0
0	1	1	1
0	0	1	1

6. ábra Tautologikus ekvivalencia

A továbbiakban az implikációt minden esetben ezzel a formulával fogjuk helyettesíteni a könnyebb kezelhetőség érdekében. Szükségünk van még néhány számítási szabályra, melyek segítségével át tudunk térni a konjunktív normálformára.

Tétel (1): Ha *A,B,C* ítéletlogikai formulák $\top$ tautológia, $\bot$ pedig kielégíthetetlen formula, akkor a következő formulák rendre tautologikusan ekvivalensek egymással:

asszociativitás:

$A \wedge (B \wedge C) \sim_0 (A \wedge B) \wedge C$ $\qquad$ $A \vee (B \vee C) \sim_0 (A \vee B) \vee C$

kommutativitás:

$A \wedge B \sim_0 B \wedge A$ $\qquad$ $A \vee B \sim_0 B \vee A$

disztributivitás:

$A \wedge (B \vee C) \sim_0 (A \wedge B) \vee (A \wedge C)$ $\qquad$ $A \vee (B \wedge C) \sim_0 (A \vee B) \wedge (A \vee C)$

idempotencia:

$A \wedge A \sim_0 A$ $\qquad$ $A \vee A \sim_0 A$

De Morgan törvényei:

$\neg(A \wedge B) \sim_0 \neg A \vee \neg B$ $\quad\quad$ $\neg(A \vee B) \sim_0 \neg A \wedge \neg B$

Kiszámítási törvények:

$A \wedge \top \sim_0 A$ $\quad\quad$ $A \wedge \bot \sim_0 \bot$

$A \vee \top \sim_0 \top$ $\quad\quad$ $A \vee \bot \sim_0 A$

Logikai jelek közötti összefüggések:

$A \wedge B \sim_0 \neg(\neg A \vee \neg B)$ $\quad\quad$ $A \vee B \sim_0 \neg(\neg A \wedge \neg B)$

Kétszeres tagadás:

$\neg\neg A \sim_0 A$

Bizonyítás (1): a tétel állításainak bizonyítása igazságtáblázatok segítségével könnyen belátható.

Definíció (11): A logikai műveletek egy halmazát **funkcionálisan teljesnek** nevezzük, ha e műveletek megfelelő logikai összekötőjeleknek és ítéletváltozóknak a felhasználásával tetszőleges $\{1,0\}^n \to \{1,0\}$ logikai művelethez meg lehet konstruálni egy formulát.

Bizonyítható, hogy a $(\neg, \wedge, \vee)$ logikai műveletekkel bármilyen ítéletlogikai formula előállítható. [1]

Definíció (12): Nevezzük a $(\neg, \wedge, \vee)$ logikai összekötőjelekkel előállított formulák halmazát **Boole-formuláknak**.

Példa (7): A $(\neg X \supset Y) \vee (Z \wedge Y)$ formulával ekvivalens *Boole-formula*: $(X \vee Y) \vee (Z \wedge Y)$

2.4 Konjunktív normálforma

Definíció (13):

1. egy prímformulát (ítéletváltozót) vagy annak negáltját közös néven **literálnak** nevezzük. A prímformulát a literál alapjának hívjuk. Egy literált esetenként egységkonjunkciónak vagy éppen egységdiszkjunkciónak (**egységklóznak**) is nevezünk.
2. Elemi konjunkció, az egységkonjunkció és a különböző alapú literálok konjunkciója, elemi diszjunkció vagy **klóz,** pedig az egységdiszkjunkció és a különböző alapú literálok diszjunkciója.
3. Diszjunktív normálformának – röviden DNF-nek – nevezzük az elemi konjunkciók diszjunkcióját, **konjunktív normálformának** – röviden ***KNF***-nek – pedig az elemi diszjunkciók (klózok) konjunkcióját.

2.5 3SAT

Definíció (14): **3SAT formuláknak** nevezzük azokat a konjunktív normálformában lévő formulákat, ahol minden klóz pontosan 3 literált tartalmaz.

A következőkben nézzük meg, hogyan állítjuk elő a *3SAT* formulákat.

Legyen *A* egy *Boole-formula*. Konstruálhatunk lineáris időben *A*-val ekvivalens *A'* formulát, hogy a negációk csak változókra vonatkoznak. Ehhez az (1) tételben bemutatott összefüggéseket használjuk fel.

Az *A'* formula konjunktív normál formában:

$\bigwedge_{i=1}^{k} \bigvee_{j=1}^{k_i} \alpha_{ij}$, ahol α_{ij} egy literál.

Példa (8): Alakítsuk át a következő formulát: $\big((A \vee B) \wedge \neg(C \vee A)\big) \vee (C \wedge \neg D)$ konjunktív normálformába!

A negáció csak a változókra vonatkozzon:

$\big((A \vee B) \wedge \neg C \wedge \neg A\big) \vee (C \wedge \neg D)$

Alkalmazzuk a disztributív tulajdonságot:

$\big((A \vee B) \wedge \neg C \wedge \neg A \vee C\big) \wedge \big(\big((A \vee B) \wedge \neg C \wedge \neg A\big) \vee \neg D\big)$

Hagyjuk el a felesleges zárójeleket, és a biztosan igaz részformulákat:

$(A \vee B \vee C) \wedge (\neg A \vee C) \wedge (A \vee B \vee \neg D) \wedge (\neg C \vee \neg D) \wedge (\neg A \vee \neg D)$

Tétel (2): 3SAT NP-teljes.

Bizonyítás: A bizonyítás megtalálható [3]-ban. Ez a levezetés az alapja a következő átalakításnak.

Alakítsuk át polinomiális időben a KNF formulát 3SAT-ra.

Minden klózra:

- ha *C* egyetlen α literált tartalmaz, kreáljuk új *x* és *y* változókat és klózokat:
  1. $\alpha \vee x \vee y$
  2. $\alpha \vee \neg x \vee y$
  3. $\alpha \vee x \vee \neg y$
  4. $\alpha \vee \neg x \vee \neg y$
- ha $C = \alpha \vee \beta$, kreáljunk új *x* változót és klózokat:
  1. $\alpha \vee \beta \vee x$
  2. $\alpha \vee \beta \vee \neg x$
- ha $C = \alpha \vee \beta \vee \gamma$, nem teszünk semmit
- ha $C = \alpha_1 \vee ... \vee \alpha_k$, k > 3 készítsünk új $x_1, ..., x_{k-3}$ változókat és klózokat:
  1. $\alpha_1 \vee \alpha_2 \vee x_1$
  2. $\neg x_i \vee \alpha_{i+2} \vee x_{i+1}$ minden $1 \le i \le k-4$
  3. $\neg x_{k-3} \vee \alpha_{k-1} \vee \alpha_k$

Példa (9): Alakítsuk a következő formulát 3SAT alakúra:

$(A \vee B \vee C) \wedge (\neg A \vee C) \wedge (A \vee B \vee \neg D) \wedge (\neg C \vee \neg D) \wedge (\neg A \vee \neg D)$.

$(A \vee B \vee C), (A \vee B \vee \neg D)$ már eleve 3 literált tartalmaz, így nem bántjuk.

$(\neg A \vee C), (\neg C \vee \neg D), (\neg A \vee \neg D)$ részformulákat a fentieknek megfelelően kiegészítjük egy új literállal.

$(\neg A \vee C \vee E), (\neg A \vee C \vee \neg E)$

$(\neg C \vee \neg D \vee E), (\neg C \vee \neg D \vee \neg E)$

$(\neg A \vee \neg D \vee E), (\neg A \vee \neg D \vee \neg E)$

Az előállított formula 3SAT alakban:

$$(A \vee B \vee C) \wedge (A \vee B \vee \neg D) \wedge (\neg A \vee C \vee E) \wedge (\neg A \vee C \vee \neg E) \wedge$$
$$\wedge (\neg C \vee \neg D \vee E) \wedge (\neg C \vee \neg D \vee \neg E) \wedge (\neg A \vee \neg D \vee E) \wedge (\neg A \vee \neg D \vee \neg E)$$

3. DIMACS CNF

Az egyes algoritmusok teszteléséhez megfelelő méretű és bonyolultságú tesztfeladatokat is szükséges előállítani. A fejezetben megismerhetünk egy egyszerű módszert a 3SAT formulák véletlenszerű előállítására.

Általában a 3SAT formulákat **DIMACS CNF**-ben szokás az algoritmusok bemeneteként megadni. Ekkor a literálok pozitív, vagy negatív egészszámként vannak jelölve, és a bemenet minden egyes sora egy klóznak felel meg. A sor végére nulla kerül, mely jelzi a feldolgozás végét, arra az esetre, ha változó elemszámú literálból állnak a klózok. Az ábrázolás ilyen módon történő megválasztásánál lehetővé válik igen nagy változó számú feladatok kezelése, és megkönnyíti az adatstruktúrák létrehozását.

Példa (11): A következő formula *DIMACS CNF* alakja:

$$(A \vee B \vee C) \wedge (A \vee B \vee \neg D) \wedge (\neg A \vee C \vee E) \wedge (\neg A \vee C \vee \neg E) \wedge$$
$$\wedge (\neg C \vee \neg D \vee E) \wedge (\neg C \vee \neg D \vee \neg E) \wedge (\neg A \vee \neg D \vee E) \wedge (\neg A \vee \neg D \vee \neg E)$$

1 2 3 0

1 2 -4 0

-1 3 5 0

-1 3 -5 0

-3 -4 5 0

-3 -4 -5 0

-1 -4 5 0

-1 -4 -5 0

3SAT formulákra használhatjuk még a következő egyszerűsített ábrázolást is:

(1,2,3),(1,2,-4),(-1,3,5),(-1,3,-5),(-3,-4,5),(-3,-4,-5),(-1,-4,5),(-1,-4,-5)

A formulák konstruálása véletlenszerű kiválasztással történik. Minden klózba válasszunk 3 változót egyenletes eloszlással, majd 50%-os valószínűséggel negáljuk őket. Figyelni kell arra, hogy a klózokon belül különféle változóink legyenek. [6]

Ha azt szeretnénk, hogy az előállított tesztpéldák még bonyolultabbak legyenek, akkor érdemes tiltani a klózok ismételt előfordulását. A következő egyszerű függvénnyel tudjuk számszerűsíteni a klózokat:

$f(x_1, x_2, x_3) = 2^{x_1} + 2^{x_2} + 2^{x_3}$, ahol x_1, x_2, x_3 literálok egészszámként ábrázolva.

Példa (10): Tekintsük a következő formulát:

$(X \vee Y \vee V) \wedge (\neg V \vee Y \vee \neg Z) \wedge (\neg Y \vee \neg Z \vee X)$

Ez az egyszerűsített ábrázolásban a következő alakra módosul:

$(1,2,4),(-4,2,-3),(-2,-3,1)$

Ebből már számolhatóak az *f* függvény értékei az egyes klózokra:

$f(1,2,4) = 2^1 + 2^2 + 2^4 = 22$

$f(-4,2,-3) = 2^{-4} + 2^2 + 2^{-3} = 4{,}1875$

$f(-2,-3,1) = 2^{-2} + 2^{-3} + 2^1 = 2{,}375$

Ezen értékek segítségével, pedig könnyedén eldönthető, hogy az aktuális klóz szerepel-e már a listánkban, vagy sem.

4. Az algoritmusok

Ebben a fejezetben bemutatásra kerülnek az egyes megvizsgált algoritmusok. A WalkSAT és a DPLL algoritmusok alapjai a Russel, Norvig: *Mesterséges Intelligencia modern megközelítésben* könyvben találhatóak meg. [2] Mindkét algoritmus működik KNF-ben lévő formulákra, de a modell és a tesztfeladatok generálásának egyszerűsítése érdekében 3SAT került választásra bemenetként.

A genTsat algoritmus a Genetikus Algoritmusok c. tantárgy féléves feladataként készült, ahol az eredeti feladat a SAT tömör formájának megoldása volt. Szerencsére ezt könnyedén át lehetett alakítani a Boole formulák feldolgozására.

Az algoritmusok működése egy részletesen kidolgozott példa segítségével kerül bemutatásra. Mindhárom algoritmussal a $\big((A \vee B) \wedge \neg(C \vee A)\big) \vee (C \wedge \neg D)$ Boole formulát oldjuk meg, de a DPLL és a WalkSAT algoritmusok meghívásához előbb 3SAT formulára át kell alakítani a feladatot, ahogyan a (8) és a (9) példákban láthattuk.

5. WalkSAT algoritmus

A WalkSAT egy egyszerű és nagyon hatékony lokális kereső algoritmus a SAT problémára, mely nagyszerűen egyesíti a mohó algoritmusok célratörő hatékonyságát, és a véletlen kiválasztás lehetőségeit. [2]

Az algoritmus bemenete egy konjunktív normál formában lévő formula. Kezdeti lépésként minden változóhoz egy véletlen értéket rendel, létrehozva a formula egy véletlen interpretációját. Ha ez modellje a formulának, az algoritmus megáll és visszaadja a modellt. Máskülönben kiválaszt egy változót, és negálja az értékét. *Hogyan történik ez a kiválasztás?*

A WalkSAT véletlenszerűen választ a ki nem elégített klózok közül és ebből kiválaszt egy változót. Ez a változó kiválasztás igen érdekes, mivel keveredik a mohó algoritmusok maximalizmusa és a véletlen választás. Adott *p* valószínűséggel folytatódhat az algoritmus a mohó kereséssel, vagy *1-p* valószínűséggel véletlenszerű választással. Az első esetben az algoritmus azt a változót választja ki, melynek megváltoztatásával a lehető legtöbb klózt elégíti ki. A véletlen választás esetén a WalkSAT lehetőséget teremt, hogy elhagyjuk egy lokális maximum pontot, ha az algoritmusunk egy nem kielégítő interpretációt részesítene előnyben.

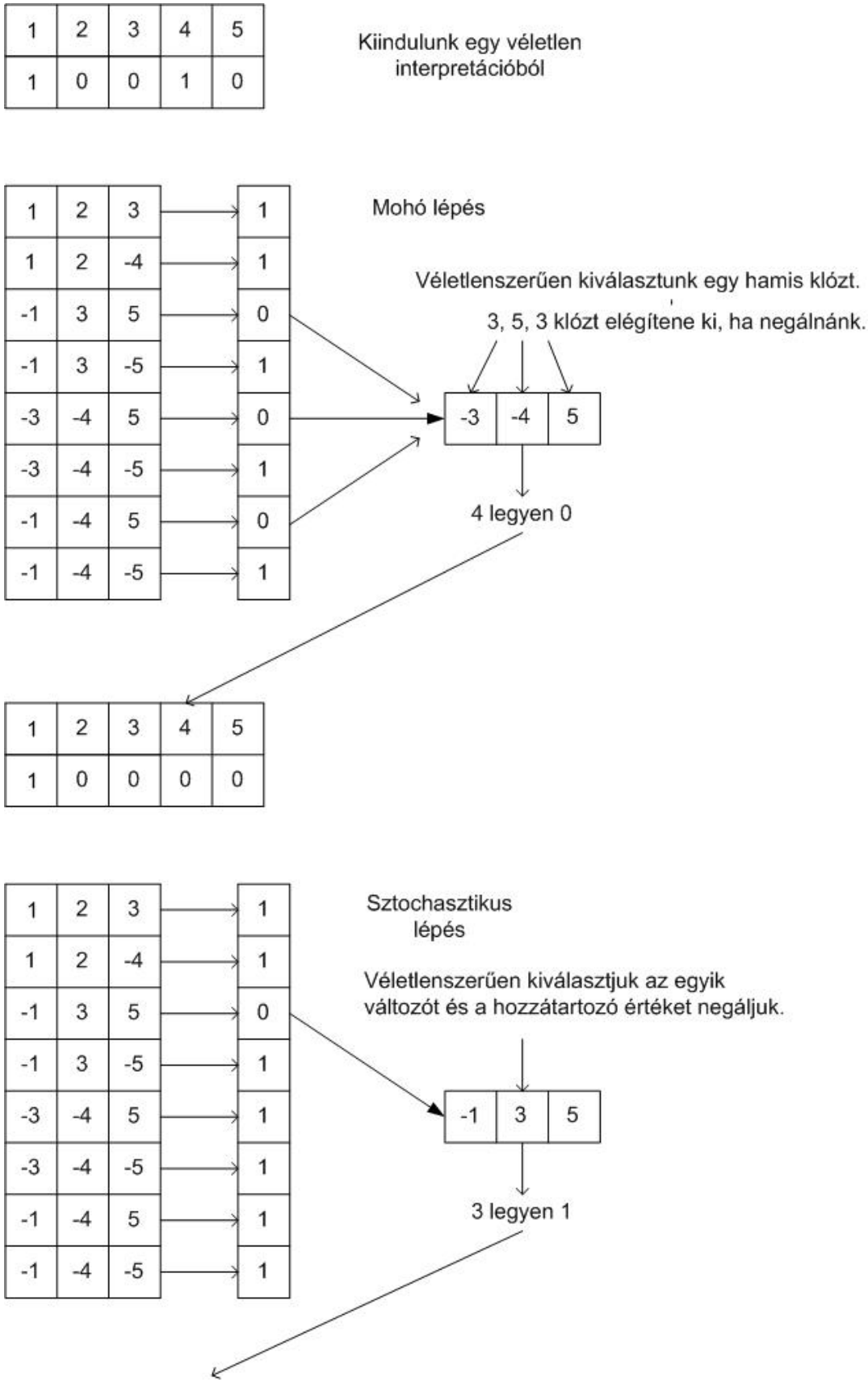
Ezt a kiválasztást iteráljuk adott ideig, és ha ezen idő alatt nem talál modellt az algoritmus, előfordulhat, hogy a kielégítetlen klózok számának egy lokális minimumát érte el. Ekkor újraindíthatjuk egy következő véletlen interpretációval.

Az algoritmus működését a 7.-8. ábrákon követhetjük figyelemmel. Az implementáció során a klózok egy mátrixban kerülnek tárolásra. Felveszünk még, egy a klózok számával megegyező méretű segédtömböt, amiben a vizsgált interpretáció eredményeit tároljuk. Ha az adott interpretáció kielégíti a klózt, akkor a segédtömbbe 1 kerül, ha nem akkor 0. A segédtömb elemeinek a segítségével könnyen ki tudjuk választani a hamis klózokat, és ha az elemeinek összege megegyezik a klózok számával, akkor egy kielégítő interpretációt találtunk, vagyis egy modellt.

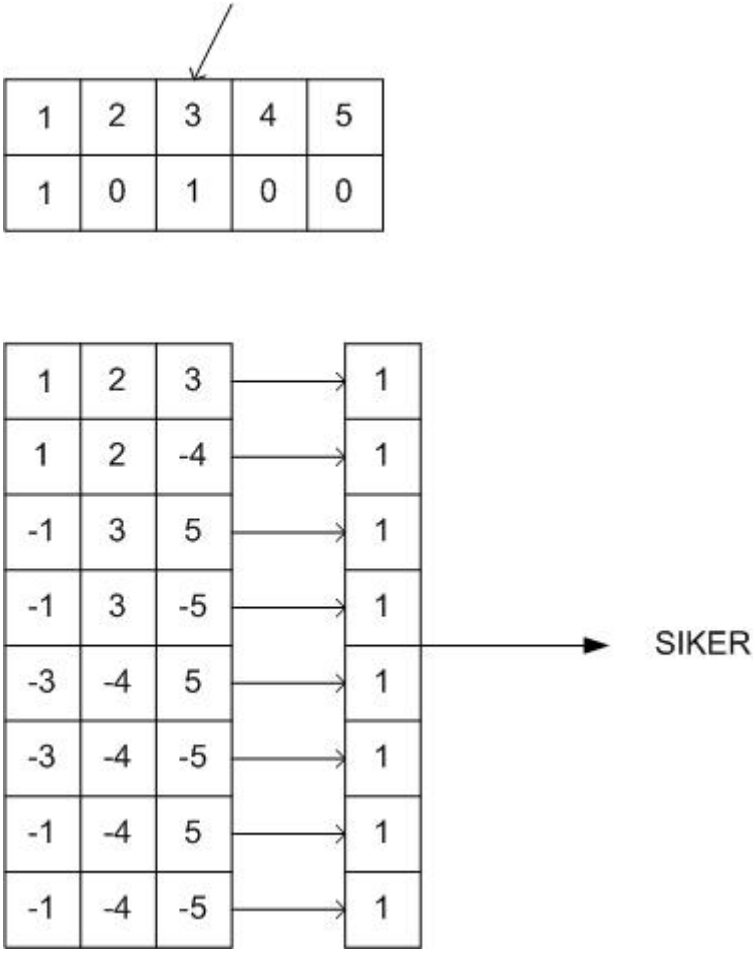
5.1 A WalkSAT algoritmus pszeudókódja

```
function WalkSAT(klózok,p,max_csere) returns egy MODELL vagy kudarc
Bementek:
    klózok DIMACS CNF formában
    p: a véletlen lépés valószínűsége
    max_csere: a megengedett cserék száma, mielőtt feladná az algoritmus

    jelölt<- T/F értékek véletlen hozzárendelései a klózok szimbólumaihoz
    for i = 1 to max_csere do
        if (jelölt egy megoldás) then return jelölt
            klóz<-véletlenszerűen választott klóz a klózok elemei közül,
                  amely hamis a modell-ben
            with probapility p érték csere egy véletlenszerűen választott
                  klóz-ban, amely hamis a jelöltben
            else bármely szimbólum, amelyik maximalizálja a kielégített
                  klózok számát
        return kudarc
```



7. ábra A WalkSAT algoritmus működése



8. ábra A WalkSAT algoritmus működése (folytatás)

6. A DPLL algoritmus

A DPLL/Davis-Putnam-Logemann-Loveland egy teljes, visszalépéses keresésen alapuló algoritmus, a SAT kielégíthetőségi problémára. Hasonlóan a WalkSAT algoritmushoz a bemenet itt is konjunktív normálformában lévő klózok halmaza. [2]

Az egyszerű visszalépéses keresési algoritmus úgy oldaná meg a feladatot, hogy választ egy literált, igazra állítja, majd rekurzívan vizsgálja, hogy a leegyszerűsített feladat kielégíthető-e. Ha ez igaz, akkor az eredeti formula is kielégíthető, ha nem, akkor a literál ellentett értékével hajtja végre a rekurzív ellenőrzést.

Ez az úgynevezett vágási szabály, mely két egyszerűbb részfeladatra osztja az eredeti feladatot. Ez az egyszerűsítési lépés alapvetően kitörli a kielégített klózkat és a maradék klózokból a literál előfordulásait.

A DPLL algoritmus kibővíti ezt a metódust a következő szabályok használatával:

Egységterjesztés:

Ha egy klóz egységklóz, vagyis egyetlen olyan literált tartalmaz, amelyik még nem kapott értéket, csak akkor elégíthető ki, ha ezt a literált olyan értékre állítjuk, ami igazzá teszi. Ebben az esetben nincs szükség választásra és sokszor elkerülhető egyes interpretációk felesleges vizsgálata.

Tiszta szimbólum keresés:

Ha egy ítéletváltozó csak egyféle előjellel fordul elő a formulában, akkor tiszta szimbólumnak nevezzük. A tiszta szimbólumnak mindig olyan értéket kell adni, hogy kielégítse az őt tartalmazó klózokat. Ebből kifolyólag ezekkel a klózokkal egyszerűsíthető a feladat.

Az algoritmus nagy előnye a *korai leállás*, észreveszi, ha egy mondat igaz vagy hamis a még részben elkészült modell alapján. Adott részleges modell kielégíthetetlensége eldönthető, ha egy klóz üressé válik, vagyis a benne szereplő összes literál oly módon kapott értéket, hogy a klóz hamissá vált.

A *kielégíthetőség* adódik akkor, ha minden klóz igazzá vált a modell alapján.

A teljes *formula kielégíthetetlensége* csak teljes keresés alapján dönthető el, ha nincs üres klóz.

6.1 Mohó literál eliminációó

Nézzünk egy DPLL megvalósítást, mely az egyes választott literálok fizikai törléséből áll. A klózok egy listában tárolódnak, ahol minden listaelem tartalmazza, hogy mennyi literálból áll az adott klóz, és magukat a literálokat egy újabb listán növekvő sorrendben.

A kiválasztás a WalkSAT algoritmusnál megismert mohó módszeren alapul. A legtöbbször előforduló literált választjuk ki, mivel ez csökkenti a legnagyobb mértékben a formulát. Ez a megoldás egyúttal a tiszta szimbólumok megkeresésére is alkalmas, külön függvény meghívása nélkül.

Amelyik klózban a kiválasztott literál előfordul, az törlődik a klózok listájából. Ha ellentett értékkel szerepel, akkor a klózhoz rendelt literálok listájáról törlődik, és csökken az adott klózhoz rendelt literálok száma. Az eliminált literálhoz tartozó igazságértéket felvesszük a *MODELL*-be.

Ha elfogytak a vizsgálandó literálok, akkor a függvény igaz értékkel tér vissza.

Az egységklóz keresés fontos szerepet kap ebben a megvalósításban. A választott literál kezelése után végignézzük a klózok listáját, hogy szerepel-e rajta egységklóz. Ha ezek között egy literál pozitív és negatív előjellel is szerepel, akkor az eddig kialakított *MODELL* alapján egy nem kielégítő interpretációt találtunk. Ez felelne meg az eredeti DPLL algoritmus üres klóz definíciójának. Ha nem így van, akkor az első egységklózban szereplő literállal ismét meghívjuk a DPLL algoritmust. A klózokat tartalmazó lista mindig rendezett a literálok száma szerint, hogy az egységklózok hamarabb megtalálhatóak legyenek.

A DPLL egy egyszerű, de nagyon hatékony teljes algoritmus a 3SAT problémára. A legnagyobb kihívás a megfelelő adatszerkezet megtalálása, mivel minden függvényhíváskor be kell járni a teljes formulát, hogy megvizsgálhassuk a már kialakult modell jóságát.

Az algoritmus működését a 9. ábrán tekinthetjük meg. A példában a visszalépés nem jelenik meg, de látható, hogy a korai leállás miatt, részben kialakult modell esetén is találhatunk megoldás. Ebben az implementációban nem szükséges ellenőrizni a kialakult modellt, hiszen a *MODELL*-be került változók kvázi behelyettesítődnek a formulába, a nekik megfelelő értékkel és ezzel párhuzamosan minden híváskor csökken a feladat mérete. A „mohóságnak” köszönhetően a tiszta szimbólum kiválasztására nem kell külön függvényt meghívni, ez automatikusan megkapható kiválasztáskor.

Mivel fizikailag is törlődnek a literálok és a klózok, ezért minden elágazás előtt meg kell duplázni a klózok listáját, hogy a literál ellentett értékére is meghívhassuk a választás előtti

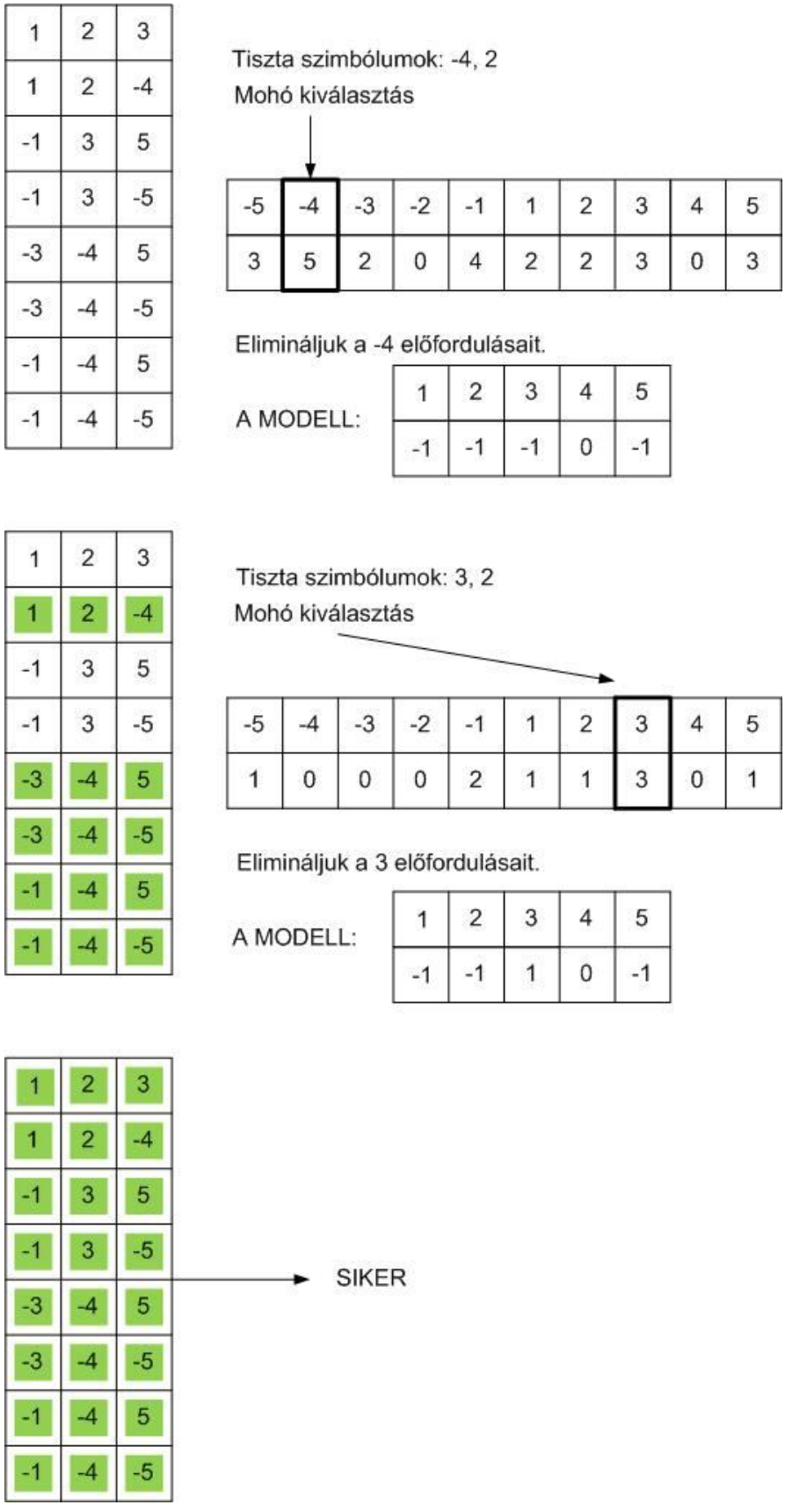
klózok listáját. Sajnos emiatt nagy feladatoknál memória korlátokba ütközik ez a DPLL megvalósítás.

6.2 A DPLL algoritmus pszeudókódja

A DPLL algoritmust már a főprogramból két listával kell meghívni:

```
Φ<-Klózok listája
Φ2 <- másol_lista(Φ)
l=valasztas(Φ)
if(DPLL(Φ, l) OR DPLL(Φ2, -l)) Φ-t KIELÉGITI a MODELL
else Φ KIELÉGITHETETLEN
   function DPLL(Φ, literal)
   MODELL<- literálhoz tartozó igaz érték
   Φ <-torol_kloz(Φ, literal)
   torol_literal(Φ, -literal)
   if(lu<-egyseg_kloz(Φ))
        if(!ellenoriz_egyseg(lu))
             Return FALSE
        else
             l <- lu->literal
             return DPLL(Φ, l)
   if(!(l=valasztas(Φ))
        return TRUE
   Φ2 <- masol_lista(Φ)
   return(DPLL(Φ, l) or DPLL(Φ2, -l))
```

:a modell felépítése
:a kielégített klózok törlése
:az egyes feleslegessé vált literálok törlése
:van-e egység klóz
:ha volt benne ellentett előjellel szereplő literálpár, a MODELL nem kielégítő
:ellenkező esetben az első egységklózban szereplő literálra meghívjuk a DPLL-t
:a tiszta szimbólum vagy a legtöbbször előforduló literál kiválasztása, ha nincs végeztük
:lista másolása
:a választott literálra meghívjuk a DPLL-t, vagy ha ez az ág nem teljesül, akkor a literál negáltjára.

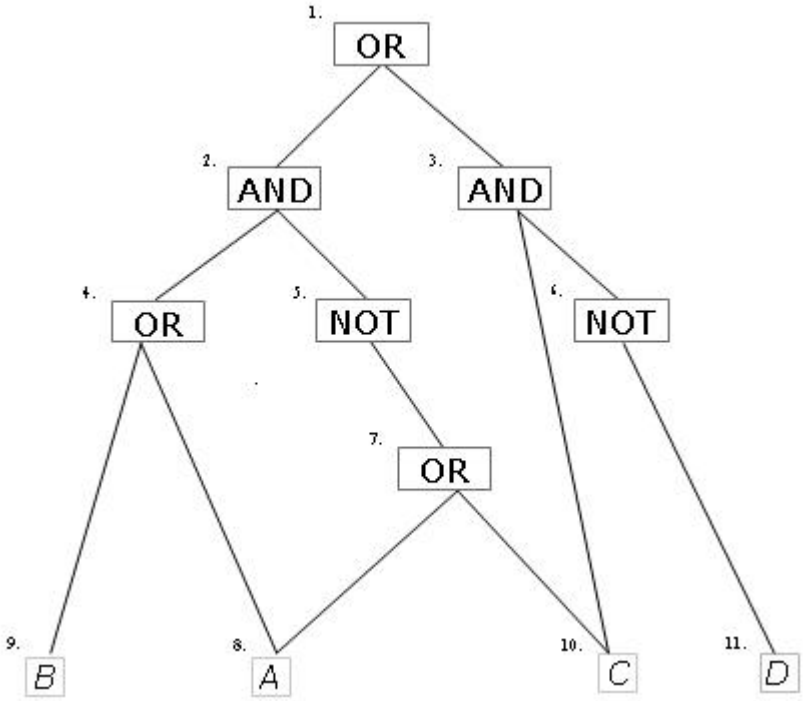


9. ábra A DPLL algoritmus működése

7. GenTsat algoritmus

A genTsat genetikus algoritmus alapja egy ítéletlogikai formula *egyszerűsített szerkezeti fája*, annyi módosítással, hogy a levélelemek több csúcshoz is tartozhatnak, így nem szükséges ismételten tárolni őket a reprezentációban. Ekkor egy irányított körmentes gráfhoz jutunk, az úgynevezett *DAG*-hoz. Minden irányított körmentes gráfnak van egy topológiai rendezése, ami a csúcsainak egy olyan sorozata, amelyben minden csúcs a belőle elérhető csúcsok előtt szerepel. Szerencsére ez a rendezés a jelen reprezentációban egyértelmű, így az adatmodell kezelése az implementáció során a fákéhoz hasonlóan történhetett.

Példa (11): $\big((A \vee B) \wedge \neg(C \vee A)\big) \vee (C \wedge \neg D)$ formula *DAG* reprezentációja:



10. ábra Formula DAG reprezentációja

Fontos megemlíteni a jelentőségét annak, hogy a genTsat algoritmus Boole formulákkal dolgozik. Láthattuk, hogy a Boole-formuláról 3SAT formulára való átalakítás költséges művelet és jelentősen megnövelheti az eredeti feladat méretét. Előfordulhat, hogy exponenciálisan nő a feldolgozandó formula mérete az átalakítás után.

A genetikus algoritmusok egyik legfontosabb pontja a reprezentáció, vagyis a kromoszóma előállítása. Mivel az egyedek létrehozása a kromoszómán végzett műveletek segítségével

történik, ezen múlik, hogy a szülőegyedek, mint lehetséges megoldások, hogyan örökítik tovább a génjeiket az új egyedek - mint új, lehetséges megoldások - felé. A feladatban a kromoszómákat rögzített hosszúságú bináris sztringek reprezentálják. A kromoszóma egy génje, egy bináris sztring egy bitje.

$$x_i,\ i\text{: } \{0,1,\ldots,L\text{-}1\},\ x_i \in \{0,1\}$$

Az L hossz az adott gráfban a csúcsok száma. Minden csúcshoz rendelünk egy bitet. A bitek sorrendjét úgy határozzuk meg, hogy az első csúcs (továbbiakban *gyökércsúcs*) - az adott Boole formula fő logikai összekötő jele - az egyes, majd a következő szinten balról jobbra haladva nőnek az indexek. A szint ezen gráf reprezentáció esetén azt a leghosszabb utat jelenti, mely összeköti a gyökércsúcsot a vizsgált csúccsal. Mivel rendezett csúcsokról van szó, ez az út egyértelmű. A kiértékelés során azt vizsgáljuk, hogy a gyökércsúcsból kiindulva, az adott csúcsnál lévő bit helyes értéket mutat-e. Ha igen, akkor egy a kromoszómához tartozó segédváltozóban igazra állítjuk ezt a bitet. Ezzel a módszerrel haladunk végig a teljes gráfon. Ha a segédváltozóban csak igaz értékek vannak, akkor megkaptuk a megoldást. Ekkor a modellt úgy nyerjük, hogy a kromoszómából az ítéletváltozókhoz rendelt csúcsok bitjeit vesszük.

A továbbiakban a Boole formulát és a logikai hálózatot (röviden hálózat) szinoním fogalmakként kezeljük. Hasonlóképpen a logikai összekötőjel és a logikai kapu (röviden kapu) megnevezések is szinonímák.

A feladatban a logikai hálózatot paraméterként lehet megadni, oly módon, hogy megadjuk az első kaput névvel, majd a kapcsolódó kapuk számát, és így tovább, míg van a hálózatban elem.

Az 10. ábrához tartozó paraméter a következő:

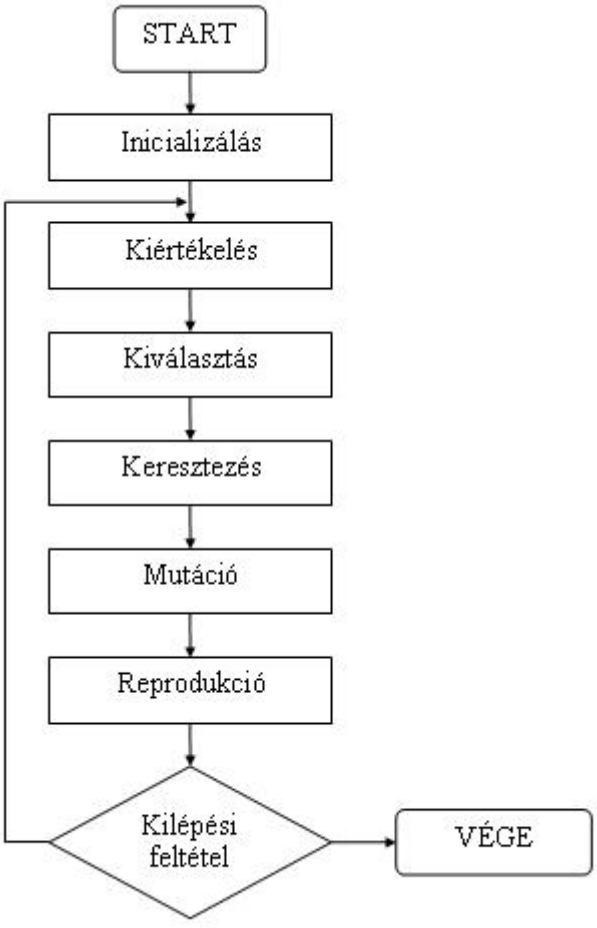
```
Logikai_Halozat <-
c('or',2,3,'and',4,5,'and',6,10,'or',8,9,'not',7,0,'not',11,0,'or',8,10)
```

A teljes populáció egy mátrixban tárolódik, melynek minden sora egy egyednek feleltethető meg. Az irányított körmentes gráf ábrázolásához egy kétdimenziós kapcsolódási mátrixot használunk, melyben az egyes oszlopok adják, hogy az adott csúcs mely következő csúcsokkal kapcsolódik. A NEM logikai kapuhoz tartozó csúcshoz egyetlen csúcs kapcsolódik, a másik ága üres helyre (0) mutat. Az ábrázolás ilyen megválasztása az alkalmazott programnyelv lehetőségéből adódik.

A genetikus algoritmusoknak nincs szigorú definíciója, így a feladat megoldása során a Holland-féle kanonikus genetikus algoritmus [4] által javasolt módszer került alkalmazásra, melynek folyamatábrája a 11. ábrán látható.

Egy általános genetikus algoritmus működésének főbb lépései:

1. Inicializálás
2. Kiértékelés
3. Kiválasztás
4. Keresztezés
5. Mutáció
6. Reprodukció
7. Kilépési feltétel vizsgálata.



11. ábra Genetikus algoritmus folyamatábrája

7.1 Inicializálás

Az egyedeket egy a csúcsok számával egyező hosszúságú bitsztring, vagy kromoszóma reprezentálja, melyek az inicializálás során kapnak értéket. Az első bit, vagy gén a feladat megoldhatósága miatt automatikusan 1-re van állítva. A kromoszóma többi génje 50%-os valószínűséggel 1-re vagy 0-ra állítódik be.

A populáció számossága paraméterként adható meg (`Populacio_Szamossaga`), ami az adott hálózat méretéhez igazítható. Egy egyed jelenti a populáció egy elemét, tagját

7.2 Kiértékelés

A populáció minden egyedéhez hozzárendelunk egy jósági értéket, jósági függvény használatával. A módszer lényege, hogy az adott génhez megvizsgáljuk, hogy milyen kapu tartozik és egy ehhez tartozó *kiértékelő táblázat* (12. ábra) segítségével eldöntjük, hogy a kapcsolódó kapukat reprezentáló gének értéke megfelel-e. *Mit is jelent ez a táblázat?*

Az egyes kapukat úgy értékeljük ki, hogy a hozzátartozó gén értékéhez megvizsgáljuk milyen bemenetre lenne szükség. Például, ha egy AND kapunál 0 érték van, akkor ez előállhat úgyis, hogy az adott kapuhoz tartozó gyermekkapuk génjei mind 0, vagy 0 és 1, vagy 1 és 0 értékekkel rendelkeznek. Ekkor mindkét gyermekkapu génjének értéke elfogadható számunkra, hiszen ezen értékeket egy logikai *'és'*-el összekapcsolva 0-t kapnánk, így a segédtömbünkbe mind a két kapcsolódási pontra 1-et írunk. Viszont ha egy AND kapunál 1 érték van, akkor részben elfogadható az az állapot is, amikor csak az egyik gyermekkapu génje rendelkezik 1 értékkel. Ez azért lehetséges, mert az ilyen géneket megtartjuk a kromoszómából, és előfordulhat, hogy a következő keresztezésnél a másik gyermekkaput reprezentáló génhez is használható érték kerül. Vagyis, ha a csúcshoz tartozó két részgráfot jelölő gének közül egy is még elfogadható értékkel rendelkezik, akkor azt jónak kell megítélni, és ha lehet tovább kell örökíteni.

A vizsgált gén értéke	A logikai kapu	A kapcsolódó csúcs/csúcsok egy/két gén értéke	A segédtáblába írandó érték
1	AND	11	11
		10	10
		01	01
		00	00
0	AND	11	00
		10	11
		01	11
		00	11
1	OR	11	11
		10	11
		01	11
		00	00
0	OR	11	00
		10	01
		01	10
		00	11
1	NOT	1	0
		0	1
0		1	1
		0	0

12. ábra Kiértékelő táblázat

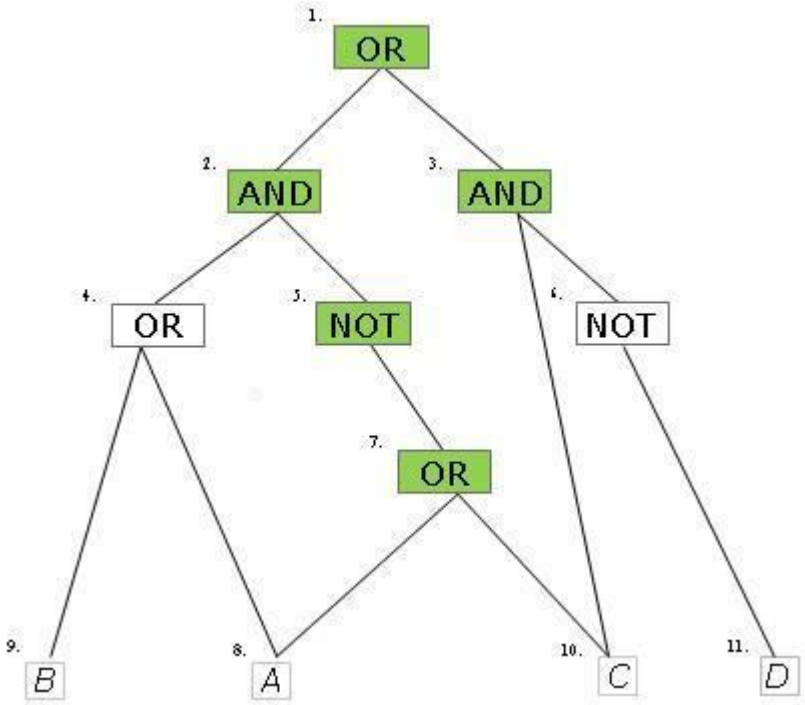
Egy egyed jóságértékének a meghatározása:
A logikai kapuk számával megegyező ciklus fut le.

Megvizsgáljuk IGAZ érték van-e a segédtömbben az adott bithez, ha nem, kilépünk a ciklusból, vége az egyed vizsgálatának.

Első lépésnél a segédtömb első eleme automatikusan IGAZRA van állítva. Az ehhez génhez tartozó logikai kapu szerint a 12. ábrán látható kiértékelő táblázat segítségével beállítjuk a segédtömbbe a kapcsolódó kapuk elfogadhatóságának értékeit.

A**1 1 1 0 1 0 0 1 1 0 1** kromoszómához tartozó segédtömb,

1 1 1 0 1 0 1 0 0 0 0, ami a gráfot a következőképpen színezi ki.



13. ábra Adott kromoszómára elfogadott kapuk

A segédtömb alapján megtudom határozni, hogy az adott egyed milyen mértékben jut el a gyökércsúcstól a magasabb szintekre. Minél tovább jut, annál jobb értéket fog kapni.

A ciklus futása után a segédtömb alapján kiszámoljuk az egyes egyedek jósági értékeit. Az érték meghatározásához minden IGAZ értéknél összegezzük a $2^{(\text{maximális szint} + 1 - \text{az adott kapu szintje})}$ értékeket. Mint korábban említettük a kapu szintjét, a gráfban hozzátartozó csúcs gyökércsúcstól való leghosszabb távolsága adja (a 10. kapu szintje 5). A 13. ábrához tartozó jóságérték: 44.

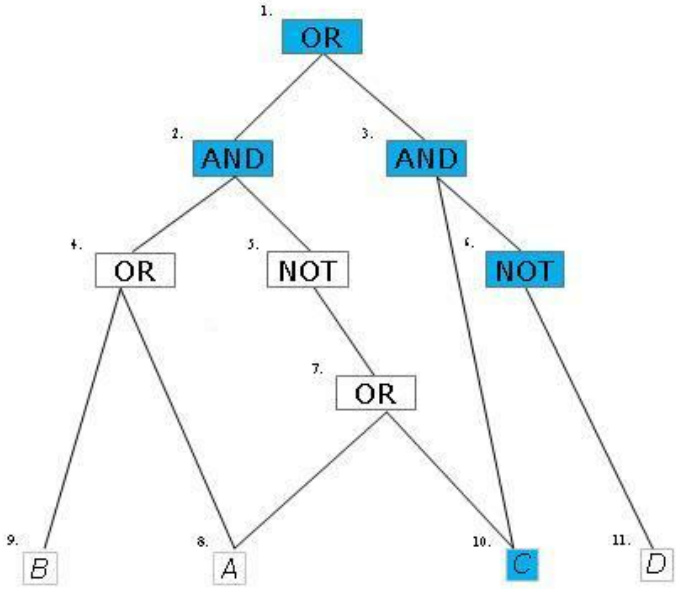
7.3 Kiválasztás

Az eredeti genetikus algoritmusban fellelhető skálázásnak a kiértékelés miatt nincs szerepe a feladatban, mert egyszerű sorrendet állítunk fel a jóság értékek alapján, egyedül a legjobb egyedet kiválasztva a későbbi keresztezéshez.

7.4 Keresztezés

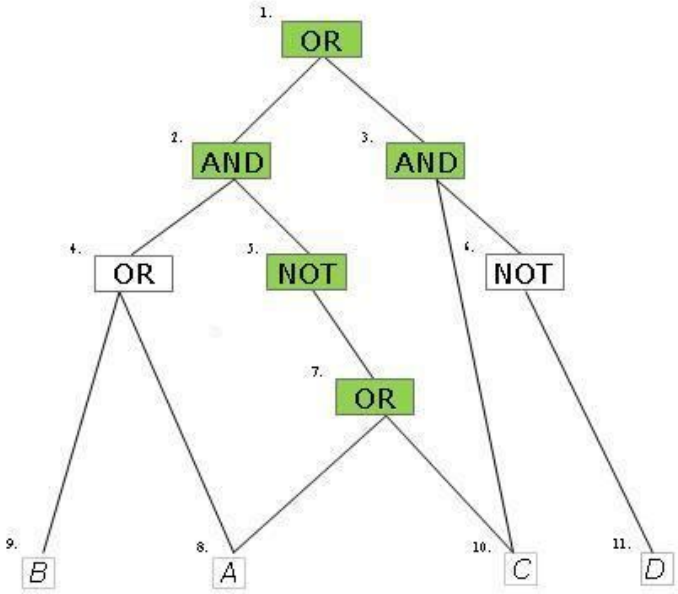
A legjobb egyed (HÍM) az, amelyik a gráfban, mely a logikai hálózatot reprezentálja, a legtöbb csúcsot színezi ki. Célszerű ezeket a részfákat továbbörökíteni a következő generációra, ezért csak ezt az egyedet választjuk ki és keresztezzük, az összes többi egyeddel (NŐSTÉNY).

Ehhez az úgynevezett uniform keresztezést használjuk, oly módon, hogy ahol a NŐSTÉNY egyednek, a segédtömb szerint elfogadható génje volt, az továbbörökítődik, ahol nem, ott a HÍM egyed elfogadható génjei öröklődnek tovább. Ha egyik sem volt elfogadható, akkor 50% eséllyel vagy egyiket, vagy másikat fogadjuk el. Ezzel a módszerrel lehet elérni azt, hogy olyan génváltozatok is továbböröklődjenek, amik jónak tűnnek, de az adott populációban nem biztosították a legjobb megoldást. A keresztezés után a HÍM egyed minden bitje paraméterezhető valószínűséggel (`Legjobb_Egyed_Mutacios_Valoszinusege`) mutálásra kerül (negálódik), új génstruktúrát injekciózva így a következő generációhoz.

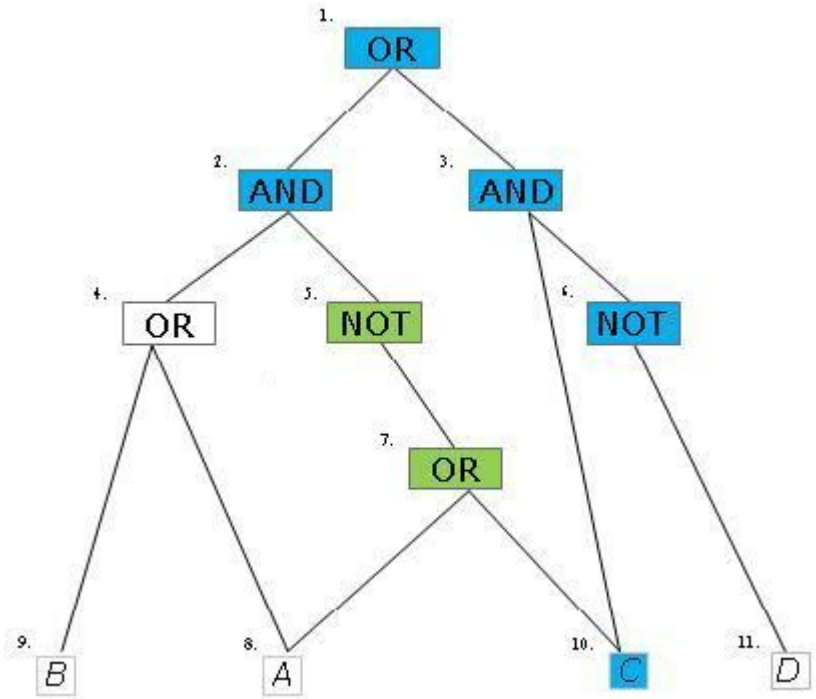


14. ábra 1 1 0 0 0 1 0 1 1 0 1 kromoszómához tartózó segédtömb: 1 1 1 0 0 1 0 0 0 1 0, és színezett gráfja

15. ábra 1 1 1 0 1 0 0 1 1 0 1 kromoszómához tartozó segédtömb: 1 1 1 0 1 0 1 0 0 0 0, és színezett gráfja.



Az 14. ábrán szereplő gráf a keresztezés során a NŐSTÉNY egyednek felelne meg, míg a 15. ábrán látható a HÍM-nek. A kettő uniform keresztezéséből a következő egyedjelölt jön létre:



16. ábra Keresztezés után.

A világosabb szürkével jelölt kapukon a HÍM egyed génjei öröklődtek, míg a sötétebb szürkével jelölteken a NŐSTÉNY egyedé. A fehér színű kapuknál 50%-os valószínűséggel öröklődhet a gén bármelyik szülőtől.

7.5 Mutáció

Feltételes mutációt alkalmazunk, oly módon, hogy folyamatosan figyeljük a legjobb egyed jósági értékét. Ha ez, egy paraméterként megadható (`Mutacio_Ciklus_Szama`) szint után sem változik, az azt jelenti, hogy több egyed is elérte ugyanazt a jósági szintet, (a legjobb egyed nagy valószínűséggel teljesen megváltozott) de a keresztezések már nem tudnak segíteni az egyes egyedek fejlődésében. A populáció konvergál egy megoldásjelölthöz, de azt még nem lehet megoldásként elfogadni. A mutáció során az összes egyed minden bitjét,

paraméterezhető valószínűség (`Bitszintu_Mutacio_Valoszinusege`) mellet negáljuk, így frissítve fel az egyedek génállományát.

7.6 Reprodukció

A reprodukciós szakaszban a keresztezés során kapott egyedekkel töltjük fel a következő ciklushoz a populációt és a segédmátrixhoz is beállítjuk az alap értékeket.

7.7 Kilépési feltétel

A programból két különböző kilépési feltétel teljesítésével lehet kilépni. Az első, hogy a populációban az egyik egyed eléri a maximális jóságot és ezáltal megadja az eredeti feladat megoldását. A másik lehetőség pedig az, ha a futási ciklusszám elér egy paraméterként (`Generacio_Maximum`) beállított értéket. Abban az esetben, ha a megadott logikai hálózat kielégíthetetlen volt, akkor a feladat megoldása általában ugyanahhoz a jósági értékhez konvergál. Egy választott paraméterrel (`Josag_Szint`), igen jól be lehet állítani a programot, hogy ennek az észrevételnek hangot adjon. Ellenkező esetben, ha az összes legjobb jósági értékkel rendelkező egyed százalékosan nem haladja meg az adott paramétert, akkor a legjobb jósági értékű egyedei közül az első megfelelő génjeit kínálja fel a program megoldásként.

8. Összefoglalás

A dolgozatban megismerhettünk három különböző elven működő algoritmust a SAT problémára. A matematikai háttér segítségével megfigyelhettük, milyen módszerek alapján dolgoznak az egyes algoritmusok, milyen módon lehet hasznosítani a különböző ábrázolási módokat, definíciókat az adatmodellek elkészítésében.

A SAT problémára nem létezik gyors megoldás, de ha jobban megismerjük a feladatot, akkor látható, hogy léteznek olyan részek, ahol lehetséges az egyszerűsítés, így csökkenthetjük az eredeti feladat méretét, bonyolultságát.

A DPLL algoritmus és variánsai, ma is az egyik leggyakrabban használt algoritmusok a kielégíthetőségi vizsgálatoknál. A három bemutatott algoritmus közül ez az egyetlen, amelyik nem kielégíthető formulákra is megáll. Előnye még az is , hogy a korai leállás miatt gyakran a teljes MODELL kialakulása előtt eldönti a feladatot. A futtatások során megfigyelhető volt, hogy sokszor lineáris időben megoldja a feladatot.

A WalkSAT algoritmus lokális kereső algoritmus, mely a mohó és véletlen keresési heurisztikákat egyesíti. Mint azt a bevezetőben említettem az algoritmus nem alkalmas kielégíthetetlen formulák ellenőrzésére és ebben az implementációban nagyon lassan oldotta meg a feladatokat. Léteznek még további heurisztikák, melyek segítségével egyes feladattípusokra még a DPLL-nél is gyorsabban fut le, de ezek megvizsgálása túlmutat a dolgozat lehetőségein. [8]

A genTsat algoritmus nagy előnye, hogy Boole formulára építve oldja meg a feladatot, és nem szükséges a költséges átalakítás 3SAT vagy KNF alakra. A kiválasztás mindig egy lokális maximum felé tereli a populációt, de a véletlenszerű választások, és mutációk miatt, az algoritmus igen nagy részét bejárja a lehetséges megoldások terének.

A dolgozatban sajnos nem sikerült az eredetileg tervezett versenyeztetés, a genTsat algoritmus a másik két algoritmussal való összehasonlítása. Nagyon sok időt töltöttem a genTsat algoritmus bemenetének 3SAT formára való átalakításának lehetőségeivel, de sajnos nem találtam meg a szükséges adatszerkezetet. Most már látom, hogy a Boole formulák 3SAT-ra alakítása lett volna a kulcs a versenyeztetéshez. Továbbá a genTsat algoritmus R nyelven lett implementálva, ellentétben a DPLL és WalkSAT algoritmusokkal, ahol elég kötöttek a választható adatszerkezetek.

9. Irodalomjegyzék

[1] Pásztorné Varga Katalin, Várterész Magda *A matematikai logika alkalmazásszerű tárgyalása* PANEM 2003

[2] Stuart Russel, Peter Norvig *Mesterséges Intelligencia modern megközelítésben* PANEM 2005

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest *Algoritmusok* Műszaki Könyvkiadó 2003

[4] Álmos Attila, Győri Sándor, Horváth Gábor, Várkonyiné Kóczy Annamária *Genetikus algoritmusok* Typotex 2002

[5] Juhász István, Kósa Márk, Pánovics János *C példatár* PANEM 2005

[6] David Mitchell, Bart Selman, Hector Levesque *Hard and easy distribution of SAT problems* In Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92), 459-465

[7] Gilles Audemard, Said Jabbour, Lakhdar Sais *SAT graph-based representation: A new perspective* Journal of Algorithms: Algorithms in Cognition, Informatics and Logic 63 (2008) 17-33

[8] Brian Ferris, Jon Froehlich *WalkSAT as an Informed Heuristic to DPLL in SAT solving* Department of Computer Science, University of Washington

10. Köszönetnyilvánítás

Köszönetet szeretnék mondani Dr. Nagy Benedeknek az ötletekért, mellyekkel segítette munkámat, a genTsat algoritmus reprezentációja az útmutatása nélkül nem valósulhatott volna meg, és amiért fáradhatatlanul lektorálta beadott vázlataimat.

Köszönettel tartozom Dr. Aszalós Lászlónak, amiért a nagyszerű óráin megismertette velem a Mesterséges Intelligencia algoritmusok izgalmas világát, és Jeszenszky Péternek, amiért megszeretetette velem az R programozási nyelvet.

Köszönettel és hálával tarozom feleségemnek, Krisztinek, hogy mindvégig támogatott és bíztatott a dolgozat írása közben.