BACKTRACK TEKNIKA SENDOTZEN

BETEGARRITASUN PROBLEMA [SAT]

- 3 INPLEMENTAZIO ARIKETA -

ALGORITMOEN DISEINUA

2017-2018

Egilea:

Iker Otxoa De Latorre

12/05/2018

AURKIBIDEA

0. SARRERA

Txosten honetan *Algoritmoen Diseinua* ikasgaiko hirugarren inplementazioari dagozkion azalpenak emango ditugu.

Hautatutako problema SAT izan da:

Forma Normal Konjuntiboan dagoen formula boolear bat emanik (CNF), bertako literalei (hau da, aldagaiei edo hauen ukapenei) true/false balioen esleipenik existitzen duen formula beteko duena (formula true egingo duena) erabakitzea da; eta balego, hura ematea da.

Egituraren aldetik, honako puntuak jorratuko dira:

- Lehenik eta behin CNF formula boolearrak irudikatzeko erabili dugun errepresentazioa aurkeztuko dugu. Kodeketa Java programazio lengoaian egin dugunez, Java klaseak erabili ditugu formuletako elementuak adierazteko (klausulak, literalak, ...).

- Ondoren, problema ebazteko erabilitako prozedurak aurkeztuko ditugu. Guztira, bi backtrack metodo garatu dira:
  - Lehenengoa sinplea, non begi-bistako backtracka erabiliz aldagai bakoitzari balio bat esleitzen zaion. Esleipenak egiteko, Dasguptaren liburuak emandako gomendioetan oinarritu gara.
  - Bigarrena nolabait landuagoa eta "iteligenteagoa", DPLL algoritmoan oinarrituta egongo dena. Lehenengo algoritmoaren hedapen bat dela esan genezake, formulak sinplifikatzeko bi metodo ezagun gehituz.

- Azkenik, algoritmoak praktikan jarri eta analisi enpiriko bat egingo dugu tamaina-desberdinetako proba-datuak erabiliz, exekuzio-denborak neurtuz eta ateratako ondorioak aurkeztuz.

Garatutako inplementazio guztia bi proiektutan entregatu da, bat algoritmo bakoitzeko (*SAT-DasguptaSinplea* eta *SAT-DasguptaAurreratua*). Java kode guztia proiektuetako *src* karpetetan eskuragarri dago. Hala eta guztiz ere, txostenean funtzio garrantzitsuenen kode zatiak agertuko dira, azalpenak osatzeko asmoz.

1. PROBLEMAREN IRUDIKAPENA

Inplementazio honetan, formulak *DIMACS* izeneko formatua jarraitzen duten fitxategietan ematen zaizkigu. Hauek irakurtzerakoan, lortutako informazioa datu-egitura aproposotan gorde beharko dugu, ondoren SAT problema ondo ebazteko aukerak izan ditzagun.

Gure kasuan, Java programazio lengoaia erabili dugunez, Java klaseak erabili ditugu formulak (eta orokorrean problema) irudikatzeko. Sortutako klaseak honakoak dira:

- *KlausulaLiterala*: Formuletako literalak irudikatzeko erabiliko den klasea. Bi atributu ditu:
    - *literalZenbakia*: literalaren zenbaki osokoa gordetzeko.
    - *zeinua*: literalaren zeinua irudikatzeko erabiliko dugun aldagai boolearra (*true* + bada, eta *false* - kasuan).

- *Klausula*: Formuletako klausulak irudikatzeko erabiliko den klasea. Bere atributu bakarra *literalZerrenda* da, *HashSet* motakoa. "Zerrendan" (*HashSet* berez multzoa adierazten du) klausula osatzen duten literalak gordeko dira (hots, *KlausulaLiterala* motako objektuak).

- *CNFFormula*: Forma normal konjuntiboan (CNF) dauden formulak irudikatzeko erabiliko den klasea. Honako atributuak ditu:
    - *klausulaZerrenda*: *LinkedList* motako zerrenda, formula osatzen duten klausulak gordetzeko erabiliko dena.
    - *klausulaTxikiena*: Klausula motako objektua, formulako klausula txikienaren erreferentzia gordetzen duena. Aurrerantzean ikusiko dugun bezala, baliagarria izango zaigu formulako klausula txikiena zein den jakiteak.
    - *literalenEsleipenak*: Literalei egindako esleipenak (*true/false*) gordetzeko erabiliko dugun zerrenda. Zerrendan *KlausulaLiterala* motako objektuak gordeko ditugu. Interpretaziorako, honako ideia izan behar dugu kontuan:
        - Zeinu negatiboko (*false*) *KlausulaLiterala* izatekotan, literal horri *false* balioa esleitu beharko zaio.
        - Zeinu positiboko (*true*) *KlausulaLiterala* izatekotan, literal horri *true* balioa esleitu beharko zaio.

- *SAT*: Klase honetan proposatutako algoritmoen kodeketa dago. Klaseko atributuak aldagai global moduan erabiliko ditugu, eta honakoak dira:
    - *formula*: Betegarria den ala ez aztertu beharko dugun formula (CNFFormula klasearen bidez irudikatuta).
    - *klausulaKopurua*: Formulak daukan klausula kopurua.
    - *literalKopurua*: Formulak dauzkan literal kopurua.
    - *adabegiKop*: backtrack zuhaitz inplizituan eraikitako adabegiak zenbatzeko erabiliko den aldagaia.

- *Frogak*: *main()* metodoa duen klasea, frogak inplementatzeko erabiliko dena.

- *Kronometroa*: Algoritmoen exekuzio denbora neurtzeko erabiliko dugun klasea.

Aurkeztuko dugun lehenengo algoritmorako aipatutako klaseak esan bezala mantenduko dira. Bigarrenerako, aldiz, aldaketak egongo dira *SAT eta CNFFormula* klaseetan, eta bigarren algoritmo bertsioa azaltzerakoan aipatuko dira.

2. ALGORITMO PROPOSAMENAK

Atal honetan garatutako kodeketa azalduko da. Lehenik eta behin hainbat ideia orokor aurkeztuko ditugu, algoritmoak egiteko zertan oinarritu garen adieraziz. Ondoren, eraikitako bi algoritmoak aurkeztuko dira. Algoritmo bakoitzarentzat, gutxienez, honako atalak garatuko dira: erabilitako datu-egiturak, soluzioen egitura, kasu nabariak, kimak, aldagaiak eta parametrizazioa, zuhaitza eta kodea.

2.1. IDEIA OROKORRAK

Hasi baino lehen komeni da gogoraraztea SAT ez dela optimizazio problema bat, baizik eta erabaki problema. CNF formatuan dagoen formula bat emanda SAT edo UNSAT den aztertu eta aipatu behar da, eta SAT izatekotan literalei eman behar zaien balioak (*true/false*) itzuli.

Bi algoritmoak sortzeko, *Dasguptaren* liburuan agertzen diren gomendioak erabili izan ditugu. Izan ere, liburuan agertzen den backtrack eskema iteratibo bera erabili dugu kodeketetan:

1. Hasierako formularekin hasi F_0

2. Izan bedi $S=\{F_0\}$ *azpiformulen multzoa*

3. S hutsa ez den bitartean:

- *3.1.* $F \in S$ *multzoko azpiformula bat hautatu eta multzotik ezabatu*
- *3.2. F azpiformula, azpiformula berrietan hedatu* $F_1,F_2,\ldots,F_k$
- *3.3.* F_i *azpiformula bakoitzarentzat:*
  - *3.3.1. Baldin test*(F_i)*=ondo* → *bilaketa amaitu eta soluzioaren berri eman*
  - *3.3.2. Baldin test*(F_i)*=gaizki* → *baztertu*
  - *3.3.3. Bestela* → F_i *S multzora gehitu*

4. Soluziorik ez dago

Liburuan agertzen diren azalpenak, ideiak eta adibideak irakurriz, *hautatu*, *hedatu* eta *test* funtzioak honela ulertu ditugu:

Test funtzioa

Lortutako azpiformula hutsa baldin bada → Formula betegarria (SAT) da.

Lortutako azpiformulan klausula huts bat baldin badago → Azpiformula baztertu, ezinezkoa izango delako betegarria izatea.

Hedatu funtzioa

Formula bat emanda, klausula txikiena hartu. Formula hedatzerakoan beti klausula txikienari erreparatuko diogu, hedapena gauzatzerakoan formula hutsa azkarrago lortzeko "esperantzarekin".

- Klausula hutsa baldin bada, ez egin ezer (*test* funtzioaren arabera formula baztertu beharko da).
- Klausulak literal bakar bat baldin badu, literal hori *true* egiten duen balioa esleitu eta horren arabera formula hedatu, azpiformula berri bat lortuz.
- Klausularen literal kopurua bat baino haundiagoa bada, klausulako edozein literal hartu eta *true* eta *false* balioak esleituz formula hedatu, bi azpiformula berri lortuz.

Hortaz, gure interpretazioaren arabera, formula bakoitzeko gehienez bi azpiformula berri lortuko ditugu.

Jarraian, hedapena nola egiten den ikusteko, adibide bat aurkeztuko dugu.
Demagun honako formula daukagula:

$$F0 = (\neg X \vee Y \vee Z) \wedge (X \vee \neg Y) \wedge (Y \vee \neg Z)$$

Klausula txikiena $(X \vee \neg Y)$ da ($(Y \vee \neg Z)$ ere izan zitekeen). Jarraian klausula txikieneko literal bat hautatuko dugu, demagun X. Literalari *true* eta *false* balioak esleituz lortuko diren bi formula berriak honako hauek izango dira:

$$\{X = true\} \Rightarrow F1 = (false \vee Y \vee Z) \wedge (true \vee \neg Y) \wedge (Y \vee \neg Z) \Rightarrow$$

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

$$\{X = false\} \Rightarrow F2 = (true \vee Y \vee Z) \wedge (false \vee \neg Y) \wedge (Y \vee \neg Z) \Rightarrow$$

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

Garrantzitsua da hedapena gauzatzerakoan bi formula berrien klausula minimoa topatzea (*klausulaTxikiena* atributua) eta *literalenEsleipenak* atributua eguneratzea. Azken hau bi formula berrietan eguneratzeko jatorrizko formularen *literalenEsleipenak* zerrendaren kopia egin beharko da, eta formula berri bakoitza lortzeko egin den esleipena gehitu. Hau da:

F1.literalenEsleipenak = kopia(F0.literalenEsleipenak) eta gehitu(X)

F2.literalenEsleipenak = kopia(F0.literalenEsleipenak) eta gehitu(¬X)

Hautatu funtzioa

Hedatu funtzioaren atalean esandakoarekin bat etorriz, hedapena egiteko klausula txikienak dituzten formulak hautatu nahiko genituzke, formula huts batera azkar heltzeko "esperantzarekin". Hori dela eta, S multzoko formulak ordenatuta egotea erabaki dugu, aldi bakoitzean azpiformularen hautaketa "erraza" izan dadin. Ordenazioa egiteko baliagarria izango da formula bakoitzeko zein den klausula txikiena jakiteak.

Formulen arteko konparazioak honako ideian oinarritzen dira: Formula bat beste batekin konparatuta txikiagoa izango da baldin eta klausula txikiagoak baditu (gutxienez bat). Adibidez:

$$F1 = \{ (A \vee B \vee \neg C) \wedge (\neg C \vee \neg A) \wedge (\neg B \vee A) \}$$

$$F2 = \{ (A \vee \neg B \vee C \vee D \vee E) \wedge (\neg A) \}$$

Aurreko formulen artean, F2 F1 baino txikiagoa da.

2.2. LEHENENGO PROPOSAMENA: ALGORITMO SINPLEA

Lehenengo kodeketa honetan aurreko atalean aipatutako ideiak garatu eta kodetu egin ditugu, gure lehenengo SAT algoritmoa lortuz.

2.2.1. Erabilitako datu-egiturak

Jarraian algoritmoa eraikitzeko erabili ditugun datu-egitura desberdinak aipatuko ditugu, aldi berean justifikazioa emanez.

- *HashSet*: Txostenaren 1 atalean (Problemaren irudikapena), *Klausuletan KlausulaLiteralak* gordetzeko *HashSet* bat erabiliko genuela adierazi genuen. Arrazoia honakoa da: Fomulen hedapena gauzatzerakoan, "*KlausulaLiteral* hau klausularen barruan dago?" galdera formulako klausula bakoitzerako egin behar da. Gainera ere, kasu batzuetan *klausulaLiterala* klausulatik ezabatu behar da. Beraz, kontuan izanez klausulen tamaina ez dela finkoa eta handiak izan daitezkeela, datu-egitura bat behar genuen zeinetan kontsulta (elementu bat dagoen ala ez) eta kendu (elementu bat egituratik atera) eragiketak denbora konstantean egiten ziren.
  HashSet-ak aipatutako bi eragiketak gehi sartu (elementu berri bat sartu) operazioa denbora konstantean egiten ditu, eta horregatik pentsatu dugu egokia zela gure kodeketarako.

- *PriorityQueue*: 2.1. atalean (Ideia Orokorrak) S azpiformulen multzoa aurkeztu da eta berarekin batera *hautatu()* funtzioa. Azken honen azalpenean azpiformulak ordenatuko genituela aipatu, eta formulak konparatzeko irizpidea eman dugu.
  Gauzak horrela, azpiformulen artean orden bat ezartzeko meta d.m.a. erabiltzea erabaki dugu (minimoena hain zuzen ere). Horrela, metaren erroan beti azpiformula guztietatik txikiena izango dugu (klausula txikienak dituena). Erabiliko ditugun bi metodoak honakoak dira:
  - *poll*(): Metaren erroa itzultzen du (hura ezabatuz) + minimoaren propietatea berreskuratu.
  - *add*(): Metan elementu berri bat sartu + minimoaren propietatea berreskuratu.

  Bi prozeduren kostua O(lg n) da, non n metak dituen elementu kopurua den.

- *LinkedList*: Formulak irudikatzeko *Klausulaz* osatutako *LinkedList*ak erabiliko ditugu. Arrazoiak honakoak dira:
  - Bektoreak ezin ditugu erabili, formula baten hedapena egiterakoan ez dakigulako lortuko ditugun azpiformula berrien tamaina.
  - Momentuz klausulak ez ditugu formulatik zuzenean hartu behar. Hau behar izatekotan *LinkedLista* ez litzateke batere egokia izango, elementuen atzipena denbora linealean egiten duelako.
  - *ArrayListak* ekidin egin ditugu, jakin badakigulako betetzen direnean tamainaz handitzen direla, dagokion kopia eginez.

2.2.2. Soluzioen egitura

Aurreko ataletan aipatu dugun bezala, azpiformula bakoitzak bere *literalenEsleipenak* atributuan (zerrenda da) jatorrizko formulatik azpiformula horretara ailegatzeko egin behar izan diren literal esleipenak izango ditu (*KlausulaLiteral* objektuen bidez adierazita). Honen adibidea 2.2.6. atalean aurkeztuko dugun backtrack zuhaitzean dago.

2.2.3. Kasu nabariak

Aurkeztutako backtrack eskema iteratiboa eta *test()* funtzioaren azalpenak kontuan hartuta, identifikatu ditugun kasu nabariak hauek dira:

- Formula hedatuz formula hutsa lortzen bada, jatorrizko formula betegarria izango da.
- S azpiformulen multzoa huts gelditzen bada, jatorrizko formula ez da betegarria izango.

2.2.4. Kimak

Algoritmo sinple honetan honako kimak kodetu ditugu, alferrikako adarkatzeak ekiditzeko:

- Azpiformula bat eraikitzean klausula huts bat agertzen bada, formula baztertu eta S multzora ez sartu, azpiformula hura hedatuz ezinezkoa izango delako formula hutsa lortzea.
- Formula baten hedapena gauzatzerakoan, klausula txikienak *KlausulaLiteral* bakarra badu adarkatze bakarra egin beharko da, hots, klausula hori *true* egiten duena (bi adarkaketak egitekotan formula batean klausula hutsa geldituko zaigu, alferrikako hedapen bat eginez).

2.2.5. Aldagaiak eta parametrizazioa

Atal honetan algoritmoa kodetzeko behar izan ditugun aldagaiak aurkeztuko ditugu, globala edo lokala den adieraziz. Informazio guzti hau jarraian agertzen den taulan jaso da (Taula 1):

ALDAGAIA	DESKRIPZIOA	LOKALA/GLOBALA
`CNFFormula formula`	CNFFormula klaseko objektua, aztertu behar dugun formula irudikatzeko erabiltzen dena. Betegarria izatekotan, bere `literalEsleipenak`	Globala

	atributuan literalei esleitu behar zaien balioak izango ditugu.	
`int klausulaKopurua`	Formulak dauzkan klausula kopurua duen aldagaia. Formularen irakurketarako erabiltzen da bakarrik (ez backtrack prozeduran)	
`int literalKopurua`	Formulak dauzkan literal kopurua	
`long adabegiKop`	Backtrack esplorazio zuhaitzean eraiki diren adabegiak zenbatzeko erabiliko den aldagaia	

Taula 1: Lehenengo algoritmorako erabili diren aldagaiak

Kasu honetan aldagai guztiak global bezala jarri izan ditugu, eta backtrack funtzioari ez zaio parametrorik pasa behar: `backtrack();`

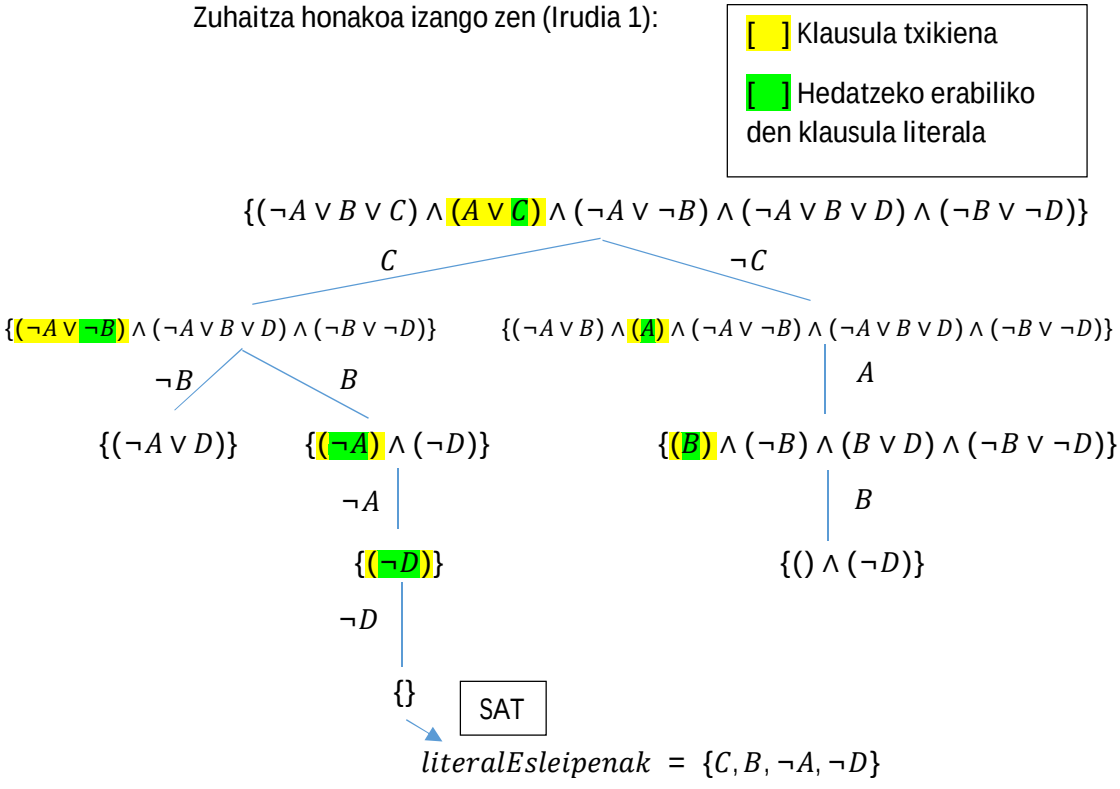
2.2.6. Zuhaitza

Atal honetan formula bat emanda algoritmoak eraikiko lukeen zuhaitza bistaratuko dugu, adibide gisa.

Demagun honako formula daukagula:

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

Zuhaitza honakoa izango zen (Irudia 1):



*Adabegi bakoitzak berea du

Irudia 1: Lehenengo algoritmoak sortutato backtrack zuhaitz adibidea

2.2.7. Kodea

Jarraian aurkeztu dugun lehenengo algoritmo honen inplementazioa erakusten dugu. Bere ulermena errazteko hainbat argipen jarri dira komentario moduan.

DASGUPTA LIBURUKO SASIKODEA JAVAN

```
private LinkedList<KlausulaLiterala> backtrack(){
	/* Esplorazio zuhaitzaren erpinak zenbatzeko aldagaia */
	adabegiKop = 0;
	/* Azpiformulak (jatorrizko formularen "azpiproblemak") dituen meta */
	PriorityQueue<CNFFormula> azpiformulak = new PriorityQueue<CNFFormula>();
	azpiformulak.add(formula);
	while (!azpiformulak.isEmpty()){
		/* Formula "txikiena" atera (formula
		guztien artetik klausula txikiena duena) */
		CNFFormula formula = azpiformulak.poll();
		LinkedList<CNFFormula> azpiformulaBerriak = hedatu(formula);
		adabegiKop = adabegiKop + azpiformulaBerriak.size();
		for(CNFFormula azpiformula : azpiformulaBerriak){
			if(azpiformula.hutsaDa()){
				/* Bilaketa amaitu da: Formula SAT da */
				return azpiformula.getLiteralenEsleipenak();
			}else if(!azpiformula.getKlausulaTxikiena().hutsaDa()){
				/* Azpiformula multzora sartu */
				azpiformulak.add(azpiformula);
			}
		}
	}
	return null; /* Formula UNSAT da */
}
```

Hautatu funtzioa. Meta erabili dugunez erroa hartzearekin nahikoa da

Test funtzioa

HEDATU FUNTZIOA

```
private LinkedList<CNFFormula> hedatu(CNFFormula formula){
	/* Gehienez bi azpiformula berri sortuko dira */
	LinkedList<CNFFormula> azpiproblemaBerriak = new LinkedList<CNFFormula>();
	if(!formula.hutsaDa() && !formula.getKlausulaTxikiena().hutsaDa()){
		/* Split: literal bat hartu eta TRUE FALSE balioak esleituz bi
azpiformula berri lortu */
		KlausulaLiterala literala =
formula.getKlausulaTxikiena().getLiteralBat();
		CNFFormula f1 = literalarenBalioaFormulanJarri(formula, literala);
		azpiproblemaBerriak.add(f1);
		if(formula.getKlausulaTxikiena().literalKopurua()>1){
			/* Klausula txikiena ez da literal bakarrekoa --> Bigarren
adarketa egin */
			KlausulaLiterala literalaEzeztua = new
KlausulaLiterala(literala.getLiteralZenbakia(), !literala.getZeinua());
			CNFFormula f2 = literalarenBalioaFormulanJarri(formula,
literalaEzeztua);
			azpiproblemaBerriak.add(f2);	}
	}else{
		azpiproblemaBerriak.add(formula);
	}
	return azpiproblemaBerriak;
}
```

LITERALAREN BALIOA FORMULAN JARTZEN DUEN FUNTZIOA (hura sinplifikatuz)

```
private CNFFormula literalarenBalioaFormulanJarri(CNFFormula formula,
KlausulaLiterala literala) {
	CNFFormula formulaBerria = new CNFFormula();
	/* Literalei esleitutako balioak kopiatu */
	formulaBerria.setLiteralenEsleipenak(new
LinkedList<KlausulaLiterala>(formula.getLiteralenEsleipenak()));
	formulaBerria.getLiteralenEsleipenak().add(literala);
	/* Formula berria eraiki */
	KlausulaLiterala literalaEzeztua = new
KlausulaLiterala(literala.getLiteralZenbakia(),!literala.getZeinua());
	for(Klausula klausula : formula.getKlausulaZerrenda()){
		/* Klausula formula berrira gehituko da literala ez badago */
		if(!klausula.klausulaLiteralaDago(literala)){
			Klausula klausulaBerria = klausula.kopia();
			if(klausula.klausulaLiteralaDago(literalaEzeztua)){
				/* Literala ezetua egotekotan klausulatik kendu */
				klausulaBerria.klausulaLiteralaEzabatu(literalaEzeztua);
			}
			formulaBerria.klausulaBerriaGehitu(klausulaBerria);
			/* Formula berriko klausula txikiena zein den harrapatu */
			if(formulaBerria.getKlausulaTxikiena()==null ||
formulaBerria.getKlausulaTxikiena().literalKopurua()>klausulaBerria.literalKopuru
a()){
				formulaBerria.setKlausulaTxikiena(klausulaBerria);
			}
		}
	}
	return formulaBerria;
}
```

2.3. BIGARREN PROPOSAMENA: DPLL-en OINARRITUTAKO ALGORITMOA

Bigarren kodeketa lehenengoaren hedapena dela esan dezakegu. Izan ere, erabilitako ideiak eta algoritmoaren egitura berdina da, baina azpiformulak sinplifikatzeko bi teknika berri gehituz.

Bi teknika hauek DPLL (*Davis-Putnam-Logemann-Loveland*) algoritmotik hartu ditugu eta propagazio unitarioan (*unit propagation-UP*) eta literal puruen ezabapenean (*pure literal elimination-PLE*) oinarritzen dira. Bi teknika hauek nolabait DPLL algoritmoaren nukleoa dira, eta nahiz eta algoritmoa 1962 urtean argitaratu, oraindik ere gaur egungo *SAT Solver* algoritmo berri askoren basea izaten jarraitzen dute.

2.3.1. Formulak sinplifikatzeko bi teknika: UP eta PLE

Jarraian lehen aipatutako bi teknikak (propagazio unitarioa eta literal puruen ezabapena) azalduko ditugu adibideak erabiliz, bien garrantziaz ohartarazteko eta hurrengo ataletan agertuko den kodea hobeto uler dadin. Gainera, bi hauek inplementatzeko izan ditugun ideiak aipatuko ditugu.

2.3.1.1. Propagazio Unitarioa (Unit Propagation – UP)

Kontzeptua

Propagazio unitarioa formulan literal bakarreko klausulak ditugunean ematen da. Klausula hauei klausula unitarioak deritze, eta barnean duten literalari balio bakar bat esleitu ahal zaio: klausula *true* egiten duena (ez dago beste aukerarik). Literalari dagokion balioa esleituz, formula hedatu eta sinplifikatuko dugu. Hedatzerakoan, klausula unitario berriak agertzekotan, prozesua berriro errepikatu beharko da. Jarraian adibide bat aurkeztuko dugu.

Demagun honako formula daukagula:

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

Klausula unitarioa: $(\neg D)$. Esleitu behar zaion balioa: *false*. Hedatuz:

$$\{(\neg A \vee B \vee C) \wedge (A) \wedge (\neg A \vee \neg B) \wedge (\neg A \vee B)\}$$

Klausula unitarioa: (A). Esleitu behar zaion balioa: *true*. Hedatuz:

$$\{(B \vee C) \wedge (\neg B) \wedge (B)\}$$

Klausula unitarioa: (B). Esleitu behar zaion balioa: *true*. Hedatuz:

$$\{(\,)\}$$

Lortutako azpiformulak klausula hutsa dauka. Hasierako formula ez da SAT.

Inplementaziorako ideiak

Lehenengo algoritmo bertsioan formula bakoitzeko klausula txikiena zein zen gordetzen genuen. Klausula txikiena klausula unitarioa denean bakarrik gauzatu beharko dugu propagazio unitarioa. Aplikatuz goazen bitartean, formula berriaren klausula txikiena zein den erregistratu beharko dugu, eta klausula unitarioa izatekotan, UPa berriro aplikatu (eta horrela behin eta berriz klausula unitariorik ez egon arte). Teknika hau *hedatu()* funtzioaren hasieran aplikatu beharko da.

2.3.1.2. Literal Puruen Ezabapena (Pure Literal Elimination – PLE)

Formula bateko literal bat purua dela esan dezakegu, formulan bere ezeztapena/ukapena agertzen ez denean. Kasu horretan, literalari esleitu behar zaion balioa begi-bistakoa da, eta ondorioz, formula sinplifikatu daiteke (literal purua dauzkaten klausulak formulatik ezabatuko dira). Hemen ere, adibide bat erabiliko dugu azalpena osatzeko.

Demagun honako formula daukagula:

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

Literal purua: C. Esleitu behar zaion balioa: *true*. Hedatuz:

$$\{(\neg A \vee \neg B) \wedge (\neg A \vee B \vee D) \wedge (\neg B \vee \neg D)\}$$

Literal purua: $\neg A$. Esleitu behar zaion balioa: *false*. Hedatuz:

$$\{(\neg B \vee \neg D)\}$$

Literal purua: $\neg B$. Esleitu behar zaion balioa: *true*. Hedatuz:

$$\{\}$$

Formula hutsera heldu gara. Hasierako formula SAT da.

OHARRA: Formula hau lehenengo algoritmoaren zuhaitza adibidea egiteko erabili da. Sortzen diren adabegi kopuruaren desberdintasunak erraz ikus daiteke kasu honetan. Lehenengo algoritmoa erabiliz 8 adabegi sortu dira, eta teknika hau erabiliz zuzenean formula hutsera heldu gara, adabegi bakarra sortuz.

Inplementaziorako ideiak

Literal puruen teknika eraginkorki kodetzeko honako ideia hartu dugu. Literal bakoitzeko bi kontagailu izango ditugu: lehenengoan literalaren agerpenak “era positiboan” (formulan) zenbatuko ditugu, eta bigarrenean aldiz, literalaren agerpenak “era negatiboan”

(ezeztapenak/ukapenak). Formula sinplifikatzerakoan, literalen kontagailuak eguneratu egin beharko dira. Literal baten bietako kontagailu bat zerora heltzen denean, bestea zero ez bada, literal puru baten aurrean egongo gara.

Adibidez:
$\{(\neg A \vee B \vee C) \wedge (A \vee C) \wedge (\neg A \vee \neg B) \wedge (\neg A \vee B \vee D) \wedge (\neg B \vee \neg D)\}$

Kontagailuak	A	B	C	D
Positibo eran	1	2	2	1
Negatibo eran (ukapena/ezeztua)	3	2	0	1

→ Literal purua

Teknika hau ere *hedatu()* funtzioaren hasieran aplikatu beharko da, baina propagazio unitarioa gauzatu eta gero. Hau horrela egin behar da propagazio unitarioak literal puruak sor ditzakeelako, eta ez alderantziz. Arrazoia: propagazio unitarioan klausula batzuk txikitu eta beste batzuk formulatik kentzen dira, baina literal puruen ezabapenean bakarrik klausulak formulatik kentzen dira. Hala eta guztiz ere, kontuan izan behar dugu literal puru baten ezabapenak beste berri batzuk sor ditzakela. Beraz, prozesua behin eta berriz aplikatu beharko da litera pururik ez egon arte.

2.3.2. Erabilitako datu-egiturak

Lehen azaldutako bi teknikak gure algoritmoan sartzeko ez da lehen egindako ezer desegin behar. Erabilitako datu-egiturak berdin mantenduko dira. Klaseetan, aldiz, aldaketak egin beharko dira, atributuak gehituz.

CNFFormula klasean honako atributuak gehitu behar izan ditugu:

- *baiLiteralKop*: literal kopuruen tamainako bektorea. Literal bakoitzaren "agerpen positiboak" zenbatzeko erabiliko da.
- *ezLiteralKop*: literal kopuruen tamainako bektorea. Literal bakoitzaren "agerpen ezeztuak/ukatuak" zenbatzeko erabiliko da.
- *literalPuruak*: *HashSet* motako multzoa, formularen literal puruak gordetzeko erabiliko dena.
  HashSet eta ez zerrenda arrunt bat erabiltzearen arrazoia honakoa da. Lehen azaldutako kontagailuen ideia literal puruak aurkitzeko ona ematen zuen, baina bere arazoa badauka. Suposa dezagun honako egoeran gaudela:

$$\{(P) \wedge (\neg Q \vee P) \wedge (\neg P \vee \neg X) \wedge (Q \vee P \vee X)\}$$

Kontagailuak	P	Q	X
Positibo eran	3	1	1
Negatibo eran (ukapena/ezeztua)	1	1	1
Literal puruak = {}			

Propagazio unitarioa aplika dezakegu, (P) klausula unitarioa baitaukagu.
Lehenengo bi klausulak ezabatuz (P literala baitute), honela gelditzen dira kontagailuak:

$$\{(\neg P \vee \neg X) \wedge (Q \vee P \vee X)\}$$

Kontagailuak	P	Q	X
Positibo eran	1	1	1
Negatibo eran (ukapena/ezeztua)	1	0	1
Literal puruak = {Q}			

Lehenengo klausulako $\neg P$ literala ezabatuz:

$$\{(\neg X) \wedge (Q \vee P \vee X)\}$$

Kontagailuak	P	Q	X
Positibo eran	1	1	1
Negatibo eran (ukapena/ezeztua)	0	0	1
Literal puruak = {Q,P}			

Azkeneko klausula ezabatuz:

$$\{(\neg X)\}$$

Kontagailuak	P	Q	X
Positibo eran	0	0	0
Negatibo eran (ukapena/ezeztua)	0	0	1
Literal puruak = {Q,P,X}			

Propagazio unitarioa aplikatu eta gero, ikus dezakegunez, literalen zerrendan baliteke puruak ez diren literalak geratzea. Nolabait, zerrendan literal purua izateko aukera zuten literalak sartu egin dira, baina azkenean batzuk formulatik desagertu dira. Hau konpontzeko, literal baten kontagailua eguneratzen den bakoitzean honakoa frogatu behar da: kontagailu bat zerora iritsi bada eta bestea ere zero bada, orduan literala literal puruen zerrendatik ezabatu egin behar da (egotekotan).

Beraz, kasu txarrenean bai kontsulta (literala literal puruen zerrendan dago) eta bai ezabapena (literala literal puruen zerrendatik ezabatu) egin beharko da. *HashSet*ak bi eragiketa hauek denbora konstantean egiten ditu, guretzat ezagunak diren zerrendak (*ArrayList*, *LinkedList*,...) ezin dutena egin (lineala izango zen). Hau izan da, hortaz, berriro *HashSetak* erabiltzera eraman gaituen arrazoia.

2.3.3. Soluzioen egitura

Lehen bertsioko algoritmoaren berdina da. Formula bakoitzak bere *literalenEsleipenak* atributuan jatorrizko formulatik azpiformula horretara ailegatzeko egin behar izan diren literal esleipenak izango ditu (*KlausulaLiteral* objektuen bidez adierazita).

2.3.4. Kasu nabariak

Bigarren algoritmo honen kasu nabariak lehenengoaren bezalakoak dira:

- Formula hutsa lortzen bada, bilaketa amaitu jatorrizko formula SAT baita.
- S azpiformulen multzoa huts gelditzen bada, jatorrizko formula ez da SAT izango.

2.3.5. Kimak

Bigarren algoritmo honetan sartutako "kima" berriak propagazio unitarioa eta literal puruen ezabapena izan dira. Berez, ez dute esplizituki backtrack zuhaitzaren adarrak kimatzen, baina formula sinplifikatzen laguntzen dute, adarkatze dezente aurreztuz.
Lehenengo bertsioko algoritmoan geneukan klausula hutsen kima ere badaukagu. Hau da, formula bat hedatzerakoan klausula hutsa sortzen bada, lortutako azpiformula baztertu egingo da.

2.3.6. Aldagaiak eta parametrizazioa

Erabilitako aldagaiak eta backtrack funtzioaren parametrizazioaren aldetik aurkeztutako lehenengo algoritmoaren egitura mantendu dugu. Aldagai guztiak global bezala jarri izan ditugu eta backtrack funtzioari dei egiterakoan parametrorik ez zaio pasa behar. Zehaztasun gehiagorako *Taula 1* berrikusi.

2.3.7. Zuhaitza

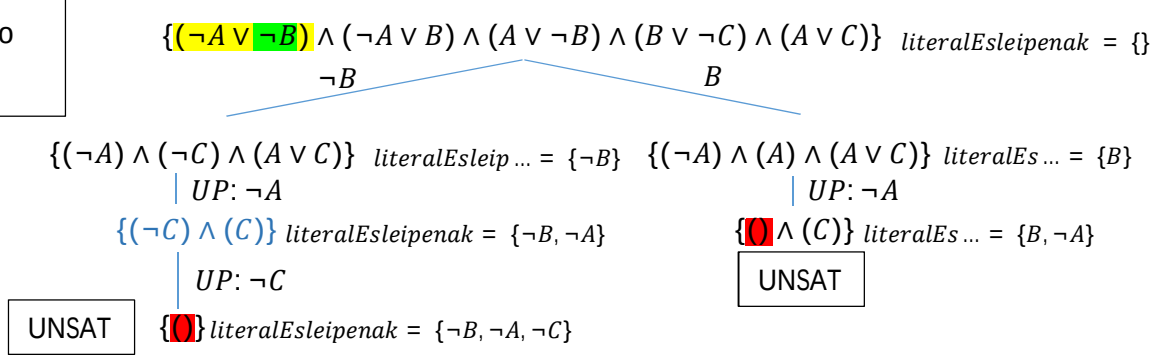
Atal honetan bigarren algoritmo honekin garatzen diren backtrack zuhaitzen adibide bat erakutsiko dugu.
Demagun honako formula daukagula:

$$\{(\neg A \vee \neg B) \wedge (\neg A \vee B) \wedge (A \vee \neg B) \wedge (B \vee \neg C) \wedge (A \vee C)\}$$

Zuhaitza honakoa izango zen (Irudia 2):

[ ] Klausula txikiena

[ ] Hedatzeko erabiliko den klausula literala



Irudia 2: Bigarren algoritmoak sortutato backtrack zuhaitz adibidea

2.3.8. Kodea

Azkenik, bigarren algoritmo bertsio hau inplementatzen duen Java kodea erakutsiko dugu. Aurrekoan bezala, hainbat komentario utzi ditugu kodearen ulermena errazteko.

DASGUPTA LIBURUKO SASIKODEA JAVAN [LEHENENGO ALGORITMOAN BEZALA]

```
private LinkedList<KlausulaLiterala> backtrack(){
	/* Esplorazio zuhaitzaren erpinak zenbatzeko aldagaia */
	adabegiKop = 0;
	/* Azpiformulak (jatorrizko formularen "azpiproblemak") dituen meta */
	PriorityQueue<CNFFormula> azpiformulak = new PriorityQueue<CNFFormula>();
	azpiformulak.add(formula);
	while (!azpiformulak.isEmpty()){
		/* Formula "txikiena" atera (formula
		guztien artetik klausula txikiena duena) */
		CNFFormula formula = azpiformulak.poll();
		LinkedList<CNFFormula> azpiformulaBerriak = hedatu(formula);
		adabegiKop = adabegiKop + azpiformulaBerriak.size();
		for(CNFFormula azpiformula : azpiformulaBerriak){
			if(azpiformula.hutsaDa()){
				/* Bilaketa amaitu da: Formula SAT da */
				return azpiformula.getLiteralenEsleipenak();
			}else if(!azpiformula.getKlausulaTxikiena().hutsaDa()){
				/* Azpiformula multzora sartu */
				azpiformulak.add(azpiformula);
			}
		}
	}
	return null; /* Formula UNSAT da */
}
```

Hautatu funtzioa. Meta erabili dugunez erroa hartzearekin nahikoa da

Test funtzioa

HEDATU FUNTZIOA

```
private LinkedList<CNFFormula> hedatu(CNFFormula formula){
	LinkedList<CNFFormula> azpiproblemaBerriak = new LinkedList<CNFFormula>();
	/* Unit propagation */
	propagazioUnitarioa(formula);
	/* Pure literal elimination */
	literalPuruenEzabapena(formula);
	if(!formula.hutsaDa() && !formula.getKlausulaTxikiena().hutsaDa()){
		/* Split: literal bat hartu eta TRUE FALSE balioak esleituz bi
azpiformula berri lortu */
		KlausulaLiterala literala =
formula.getKlausulaTxikiena().getLiteralBat();
		CNFFormula f1 = literalarenBalioaFormulanJarri(formula, literala);
		azpiproblemaBerriak.add(f1);
		KlausulaLiterala literalaEzeztua = new
KlausulaLiterala(literala.getLiteralZenbakia(), !literala.getZeinua());
		CNFFormula f2 = literalarenBalioaFormulanJarri(formula,
literalaEzeztua);
		azpiproblemaBerriak.add(f2);
	}else{
		azpiproblemaBerriak.add(formula);
	}
	return azpiproblemaBerriak;
}
```

LITERALAREN BALIOA FORMULAN JARTZEN DUEN FUNTZIOA (hura sinplifikatuz)

```
private CNFFormula literalarenBalioaFormulanJarri(CNFFormula formula,
KlausulaLiterala literala) {
	CNFFormula formulaBerria = new CNFFormula();
	/* Aldagaiei esleitutako balioak kopiatu */
	formulaBerria.setLiteralenEsleipenak(new
LinkedList<KlausulaLiterala>(formula.getLiteralenEsleipenak()));
	formulaBerria.getLiteralenEsleipenak().add(literala);
	/* Literalen agerpenak (baiezkoak eta ezezkoak) kopiatu */
	formulaBerria.setBaiLiteralKop(formula.getBaiLiteralKop().clone());
	formulaBerria.setEzLiteralKop(formula.getEzLiteralKop().clone());

	/* Formula berria eraiki */
	KlausulaLiterala literalaEzeztua = new
KlausulaLiterala(literala.getLiteralZenbakia(),!literala.getZeinua());
	for(Klausula klausula : formula.getKlausulaZerrenda()){
		/* Klausula formula berrira gehituko da literala ez badago */
		if(!klausula.klausulaLiteralaDago(literala)){
			Klausula klausulaBerria = klausula.kopia();
			/* Literalaren ukapena egotekotan, klausula berritik ezabatu */
			if(klausula.klausulaLiteralaDago(literalaEzeztua)){
				klausulaBerria.klausulaLiteralaEzabatu(literalaEzeztua);
				literalarenAgerpenKopuruaEguneratu(formulaBerria,
literalaEzeztua);
			}
			formulaBerria.klausulaBerriaGehitu(klausulaBerria);
			/* Formula berriko klausula txikiena harrapatzeko */
			if(formulaBerria.getKlausulaTxikiena()==null ||
formulaBerria.getKlausulaTxikiena().literalKopurua()>klausulaBerria.literalKopuru
a()){
				formulaBerria.setKlausulaTxikiena(klausulaBerria);
			}
		}else{
			/* Literala egotekotan klausula formula berrian ez sartu eta
			 * klausulan zeuden literalen agerpenak eguneratu */
			literalenAgerpenakEguneratu(formulaBerria,klausula);
		}
	}
	return formulaBerria;
}
```

PROPAGAZIO UNITARIOA GAUZATZEN DUTEN METODOAK

```
private void propagazioUnitarioa(CNFFormula formula){
	Klausula klausulaTxikiena = klausulaUnitarioaDago(formula);
	while(klausulaTxikiena!=null){
		/* Klausula txikiena zein den berriro bilatuko dugu */
		formula.setKlausulaTxikiena(null);
		/* Klausulan dagoen literal bakarra hartu */
		KlausulaLiterala literala = klausulaTxikiena.getLiteralBat();
		formula.getLiteralenEsleipenak().add(literala);
		KlausulaLiterala literalaEzeztua = new
KlausulaLiterala(literala.getLiteralZenbakia(), !literala.getZeinua());
		Iterator<Klausula> iteradorea =
formula.getKlausulaZerrenda().iterator();
		/* Formulako klausula guztiak korritu */
		while(iteradorea.hasNext()){
			Klausula klausula = iteradorea.next();
			if(klausula.klausulaLiteralaDago(literala)){
				literalenAgerpenakEguneratu(formula, klausula);
				iteradorea.remove();
			}else{
				if(klausula.klausulaLiteralaDago(literalaEzeztua)){
					literalarenAgerpenKopuruaEguneratu(formula,
literalaEzeztua);
					klausula.klausulaLiteralaEzabatu(literalaEzeztua);
				}
				/* Formulako klausula txikiena harrapatzeko */
				if(klausula!=klausulaTxikiena &&
(formula.getKlausulaTxikiena()==null ||
klausula.literalKopurua()<=formula.getKlausulaTxikiena().literalKopurua())){
					formula.setKlausulaTxikiena(klausula);
				}
			}
		}
		/* Klausula unitario gehiago daude?(prozesuan berriak sor daitezke) */
		klausulaTxikiena = klausulaUnitarioaDago(formula);
	}
}

* Klausula unitarioak dauden ala ez jakinarazten digun metodoa
private Klausula klausulaUnitarioaDago(CNFFormula formula){
	Klausula klausulaTxikiena = formula.getKlausulaTxikiena();
	if(!formula.hutsaDa() && klausulaTxikiena.literalKopurua()==1){
		return klausulaTxikiena;
	}else{
		return null;
	}
}
```

LITERAL PURUEN EZABAPENAZ ARDURATZEN DIREN METODOAK

```
private void literalPuruenEzabapena(CNFFormula formula){
	HashSet<KlausulaLiterala> literalPuruak = literalPuruakDaude(formula);
	while(literalPuruak!=null){
		/* Formularen literal puruen zerrenda hasieratu berriak hartzeko */
		formula.setLiteralPuruak(new HashSet<KlausulaLiterala>());
		/* Literal puruak ezabatu */
		for(KlausulaLiterala literalPurua : literalPuruak){
			formula.getLiteralenEsleipenak().add(literalPurua);
			Iterator<Klausula> iteradorea =
formula.getKlausulaZerrenda().iterator();
			while(iteradorea.hasNext()){
				Klausula klausula = iteradorea.next();
				if(klausula.klausulaLiteralaDago(literalPurua)){
					literalenAgerpenakEguneratu(formula, klausula);
					iteradorea.remove();
				}
			}
		}
		/* Literal puru gehiago daude? */
		literalPuruak = literalPuruakDaude(formula);
	}
}

* Literal puruak dauden ala ez jakinarazten digun metodoa
private HashSet<KlausulaLiterala> literalPuruakDaude(CNFFormula formula) {
	if(formula.getLiteralPuruak().isEmpty() || formula.hutsaDa() ||
formula.getKlausulaTxikiena().hutsaDa()){
		return null;
	}else{
		return formula.getLiteralPuruak();
	}
}
```

LITERALEN KONTAGAILUAK EGUNERATZEKO METODOAK

```
* Literal puruak aurkitzeko metodo laguntzailea.
* Klausula bat formulatik ezabatzen denean literalen agerpenak eguneratu behar dira.
private void literalenAgerpenakEguneratu(CNFFormula formulaBerria, Klausula klausula) {
	for(KlausulaLiterala klausulaLiterala : klausula.getKlausularenLiteralZerrenda()){
		literalarenAgerpenKopuruaEguneratu(formulaBerria, klausulaLiterala);
	}
}

* Literal puruak aurkitzeko metodo laguntzailea.
* Literal bat klausula batetik ezabatzen denean bere agerpenak eguneratu behar dira.
private void literalarenAgerpenKopuruaEguneratu(CNFFormula formula, KlausulaLiterala
literala) {
	int [] aux; /* Literalaren zeinu bereko kontagailu bektorea */
	int [] aux1; /* Literalaren kontrako zeinuko kontagailu bektorea */
	if(literala.getZeinua()){
		aux = formula.getBaiLiteralKop(); aux1 = formula.getEzLiteralKop();
	}else{
		aux = formula.getEzLiteralKop(); aux1 = formula.getBaiLiteralKop();
	}
	aux[literala.getLiteralZenbakia()-1]--;
	if(aux[literala.getLiteralZenbakia()-1]==0 && aux1[literala.getLiteralZenbakia()-
1]!=0){
		KlausulaLiterala literalaEzeztua = new
KlausulaLiterala(literala.getLiteralZenbakia(),!literala.getZeinua());
		formula.literalPuruaGehitu(literalaEzeztua);
	}else if(formula.getLiteralPuruak().contains(literala)){
		formula.getLiteralPuruak().remove(literala);
	}
}
```

3. AZTERKETA ESPERIMENTALA

Atal honetan aurrekoetan aurkeztutako bi algoritmoak praktikan jarriko ditugu, hainbat exekuzio eginez sarrera datu desberdinekin.

Sarrera datuak enuntziatuan jarritako estekan zeuden fitxategietatik lortu ditugu gehienbat, honako ezaugarriak izanez:

- Fitxategi bakoitzak honako aurrizkietako bat izango du: *uf* edo *uuf*. *uf* fitxategietako formulak betegarriak izango dira, eta *uuf* fitxategietakoak, aldiz, ez.
- Fitxategiak fitxategi trinkotuetan antolatuta daude, eta bakoitzean tamaina bereko fitxategi ugari daude. Hau da, fitxategi trinkotu bakoitzean literal eta klausula kopuru bereko fitxategien bilduma dago. Gure kasuan, multzo bakoitzeko 4 fitxategi hartu ditugu (100, 099, 098 eta 097 zenbakia dutenak).
- Aurrizkiaren arabera, fitxategi bakoitzak literal eta klausula kopuru finko bat du. Balio hauek Taula 3 taulan jaso dira.

Fitxategi aurrizkia	Literal kopurua	Klausula kopurua
[uf/uff]20-...	20	91
[uf/uff]50-...	50	218
[uf/uff]75-...	75	325
[uf/uff]100-...	100	430
[uf/uff]125-...	125	538
[uf/uff]150-...	150	645
[uf/uff]175-...	175	753
[uf/uff]200-...	200	860
[uf/uff]225-...	225	960
[uf/uff]250-...	250	1065

Taula 3: Aurrizkiaren arabera fitxategi bakoitzak duen literal eta klausula kopurua

Behin fitxategiak izanda, bi algoritmoak fitxategi guztiekin exekutatu ditugu. Helburua bikoitza izan da: alde batetik algoritmoen zuzentasuna frogatzea, eta bestetik, bi algoritmoen arteko desberdintasunak ikustea (eraginkortasunaren aldetik). Gainera, gerora analisiak egiteko, exekuzio bakoitzak hartu duen denbora (milisegundotan) eta sortutako adabegi kopurua erregistratu ditugu. Datu hauek formatu tabularrean atxikitu ditugu txosten honetara, eta eranskineko atalean aurki daitezke.

Zuzentasunaren aldetik, algoritmoak itzulitako emaitza guztiak zuzenak izan dira. Xehetasun gehiagorako, proiektuarekin batera igo diren exekuzio *log*-ak ikus daitezke.

Analisiaren aldetik, lortutako emaitzak behin aztertu ondoren, klasean ikusitako “algoritmo-exekuzio ezaugarriekin” topatu eta hainbat ondoriotara heldu gara. Hauek hurrengo orrietan aurkeztuko ditugu. Bistaratuko ditugun grafiketan “DS” (*Dasgupta-Simplea*) hitzak (legenda eta tauletan agertuko dena) garatutako lehenengo algoritmoari erreferentzia egingo dio, eta “DA” (*Dasgupta-Aurreratua*) hitzak, aldiz, bigarrenari.

Ondorioak:

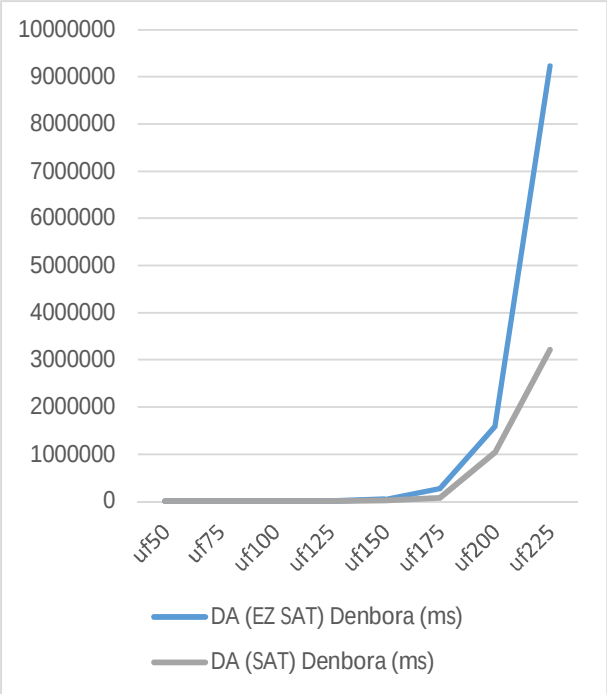
1. Lehenengo puntu hau, ondorioa baino, klasean ikusitako ezaugarri bat da, gure exekuzioetan erraz ikusi izan dena: garatutako algoritmoak, nahiz eta tamaina bereko sarrerak jaso, hauen datuekiko sentikorrak dira. Adibide moduan, Taula 4 taulan jasotako denborak ditugu. Exekuziorik gabe ere hau erraz ikus liteke, formula baten zailtasuna ez delako bere tamainaren arabera neurtzen.

DA algoritmoa

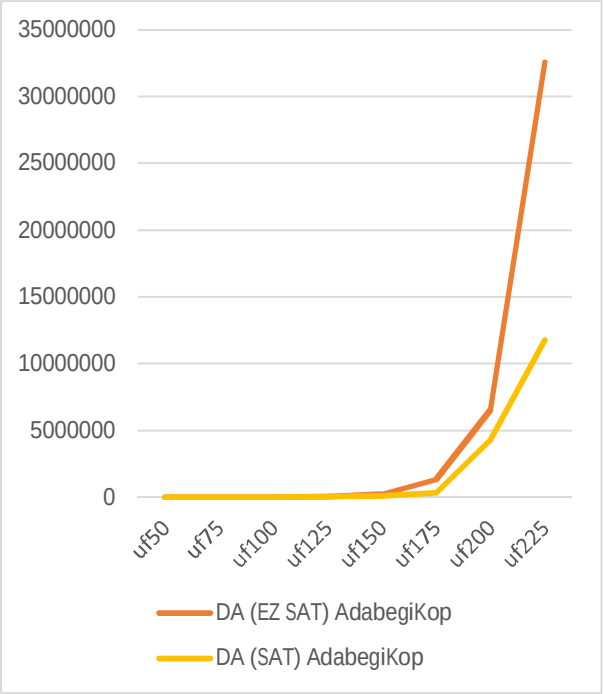
Fitxategia	*Denbora(ms)*
uf175-0100.cnf	184045
uf175-097.cnf	18572
uf175-098.cnf	39027
uf175-099.cnf	16867

Taula 4: Tamaina bereko formulentzako jasotako denborak

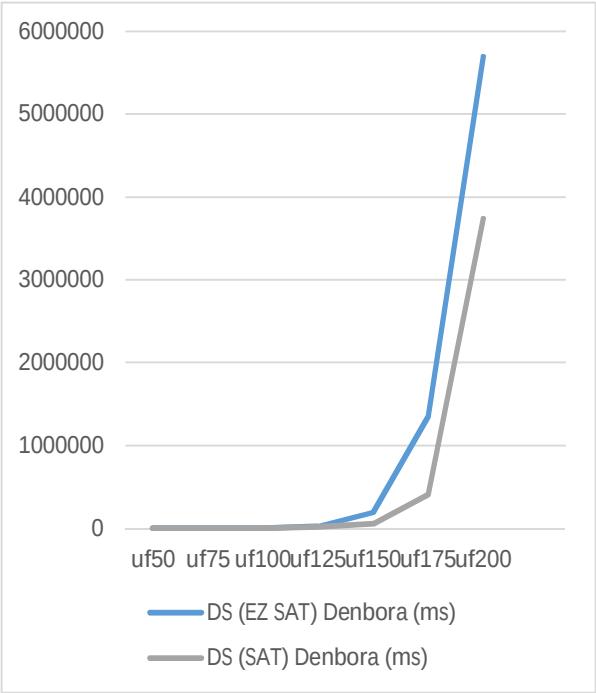
2. Exekuzio denborak eta sortutako adabegi kopurua ikusita, betegarriak (SAT) diren formulen kostua betegarriak ez diren formulen kostua baino txikiagoa da. Ondorio hau erraz suposa genezakeen SAT problemaren ezaugarriak gogoratuta. Optimizazio problema ez denez, formula betearazten duten literal esleipenak aurkitzekotan bilaketa geldiaraziko da, eta bestela, backtrack zuhaitz osoa aztertu beharko da. Bai denbora eta bai adabegi kopuru desberdintasunak ikusteko Grafiko 1, Grafiko 2, Grafiko 3 eta Grafiko 4 grafikak prestatu ditugu (bi algoritmoen exekuzioetan nabarmentzen baitira).



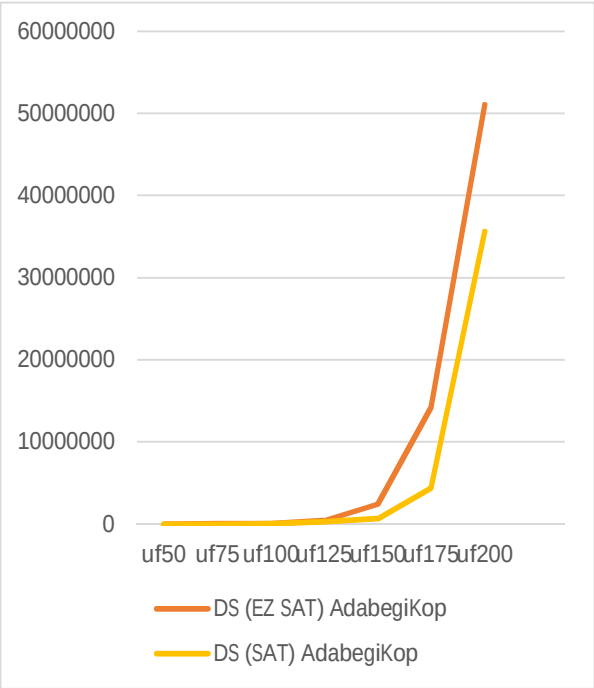
Grafiko 1: SAT formula VS. UNSAT formula denborak [DA algoritmorako]



Grafiko 2: SAT formula VS. UNSAT formula adabegi kopurua [DA algoritmorako]



Grafiko 3: SAT formula VS. UNSAT formula denborak [DS algoritmorako]

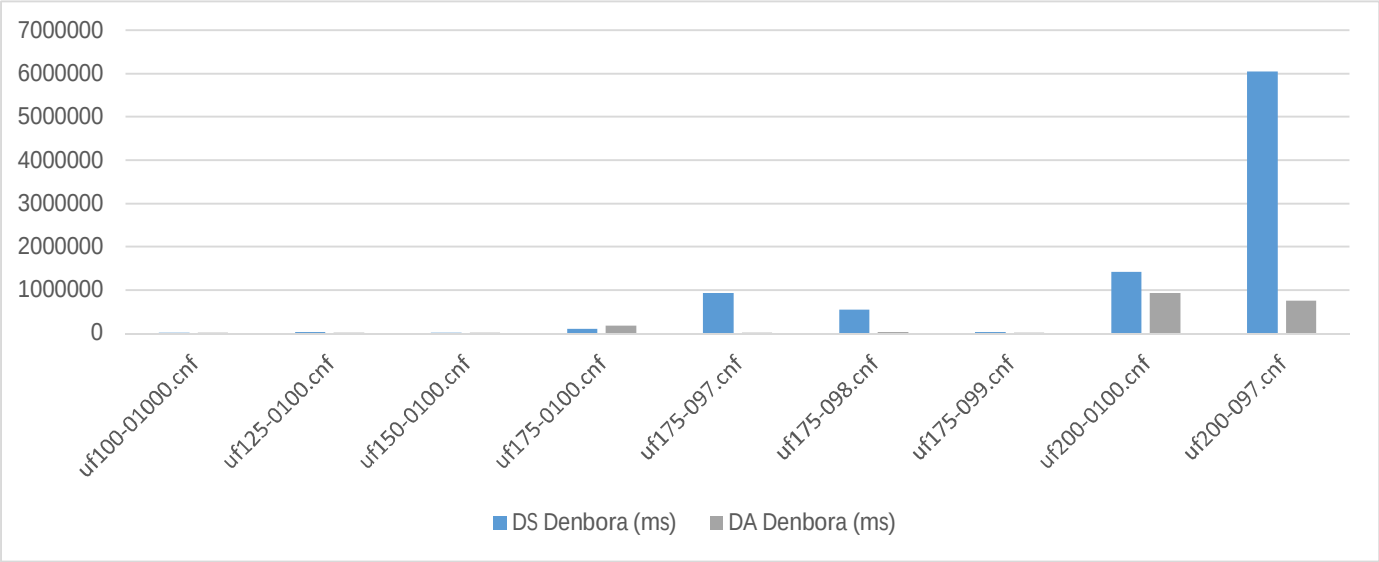


Grafiko 4: SAT formula VS. UNSAT formula adabegi kopurua [DS algoritmorako]

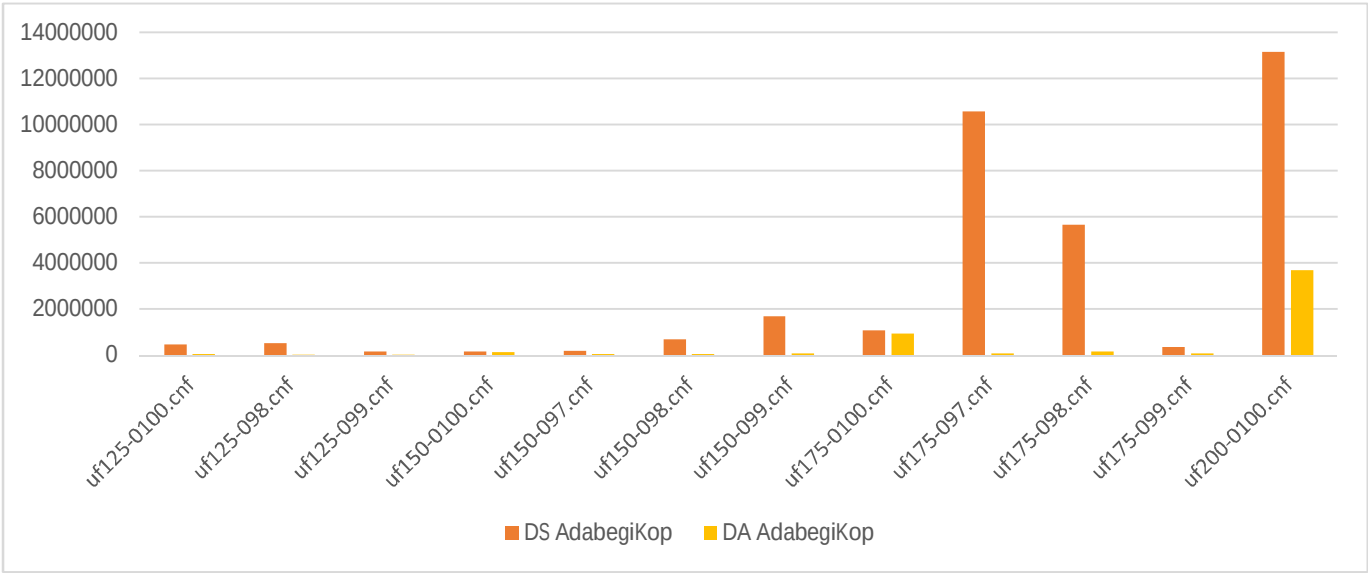
OHARRA: Aurreko grafikoak egiteko nolabait datuak "orokortu" behar ziren. Horretarako, aurrizki berdinetako fitxategien denborak hartu eta algoritmo bakoitzeko batezbestekoa kalkulatu dugu. Berdina adabegi kopuruarekin.

Gainera, SAT eta UNSAT formulen arteko exekuzio denborak are eta gehiago bereizten laguntzen duen zerbait erabilitako heuristikoa/jalea da. Formulen hedapena beti klausula txikienetik gauzatuko dugu, formula hutsa lortzeko probabilitate handiagoa izateko. Backtrack zuhaitzaren adar zuzenean egotekotan, jale honek formula SAT dela erabakitzera azkar eramango gaitu.

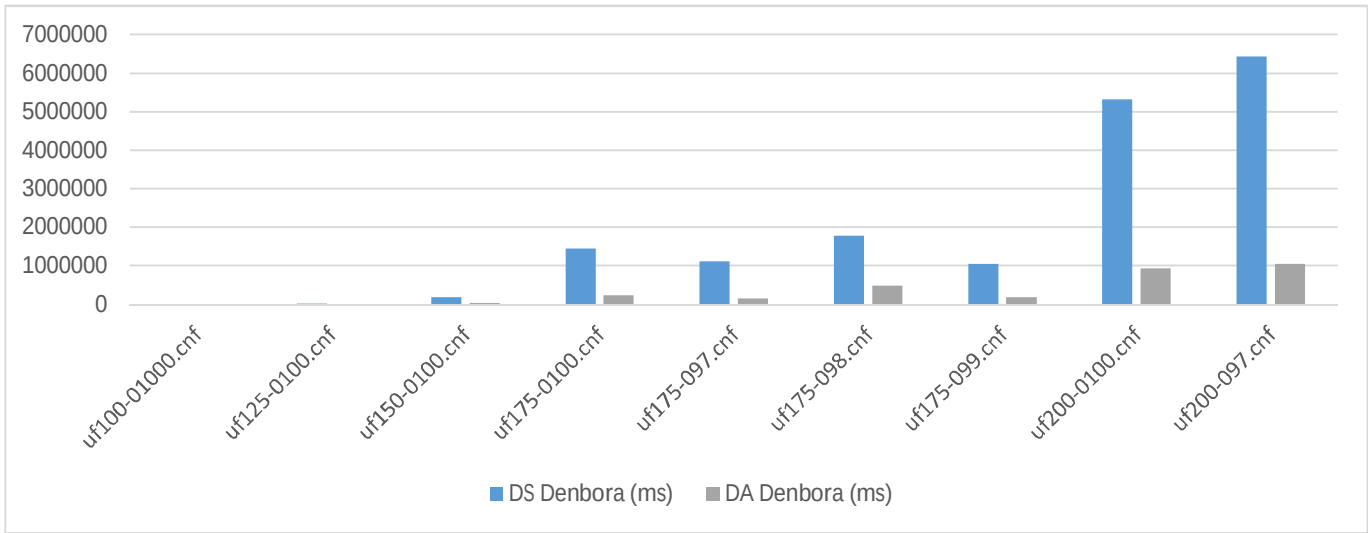
3. Propagazio unitarioaren eta literal puruen ezabapenaren teknikak oso onak dira praktikan formulak sinplifikatzeko. Adabegi kopurua (hau da, azpiformula kopurua) eta ondorioz, exekuzio denbora izugarri murrizten dute. Azken hau erraz ikus daiteke Grafika 5, Grafika 6, Grafika 7 eta Grafika 8 grafiketan. Beraz, *SAT Solverrak* eraikitzeko garaian kontuan izan behar diren teknikak dira hauek.



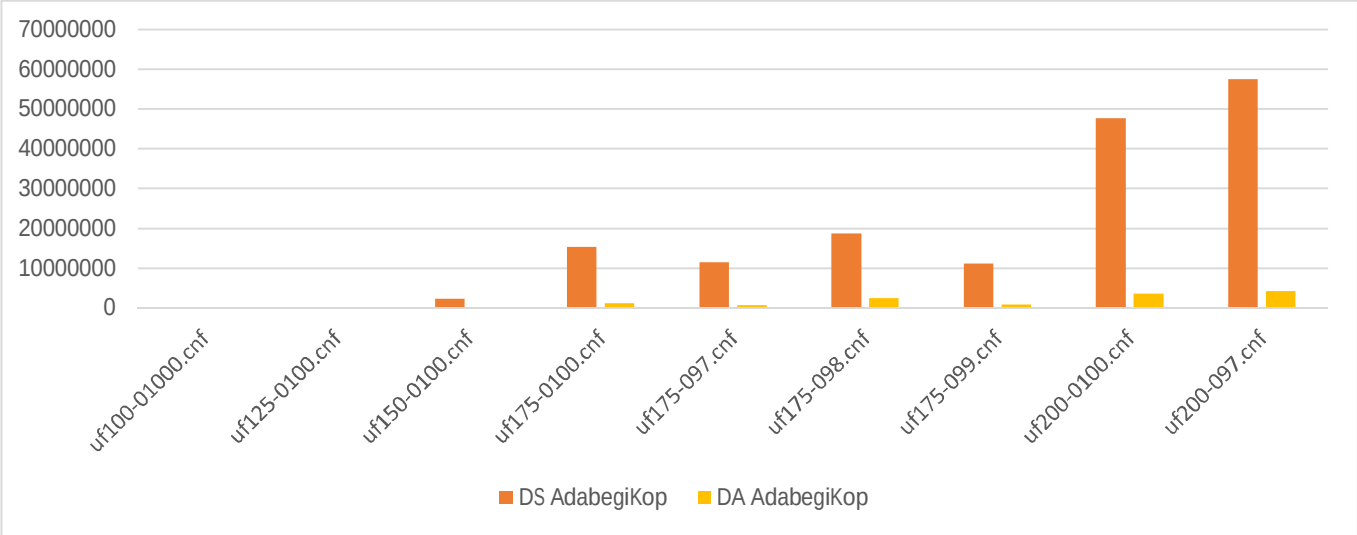
Grafiko 5: DA VS. DS exekuzio denborak [SAT formulekin]



Grafiko 6: DA VS. DS adabegi kopurua [SAT formulekin]

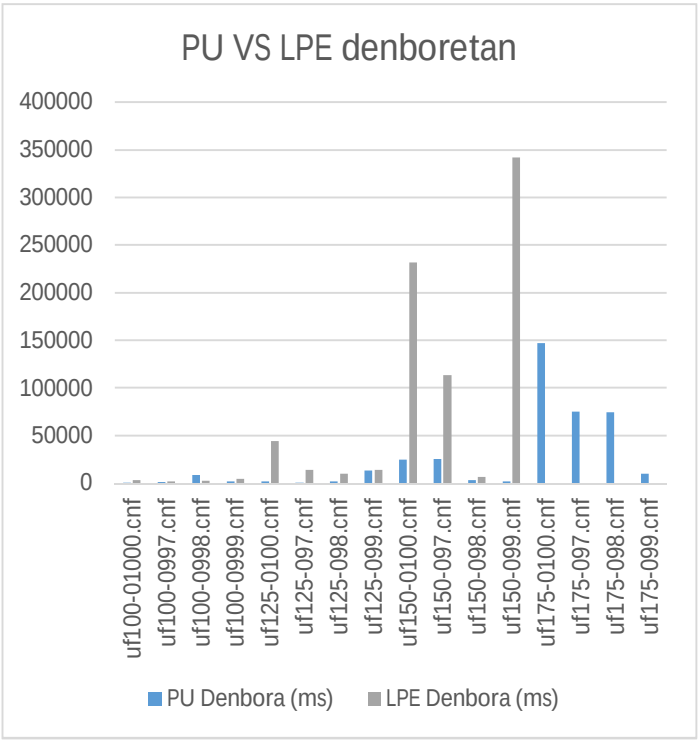


Grafiko 7: DA VS. DS exekuzio denborak [Ez SAT formulekin]

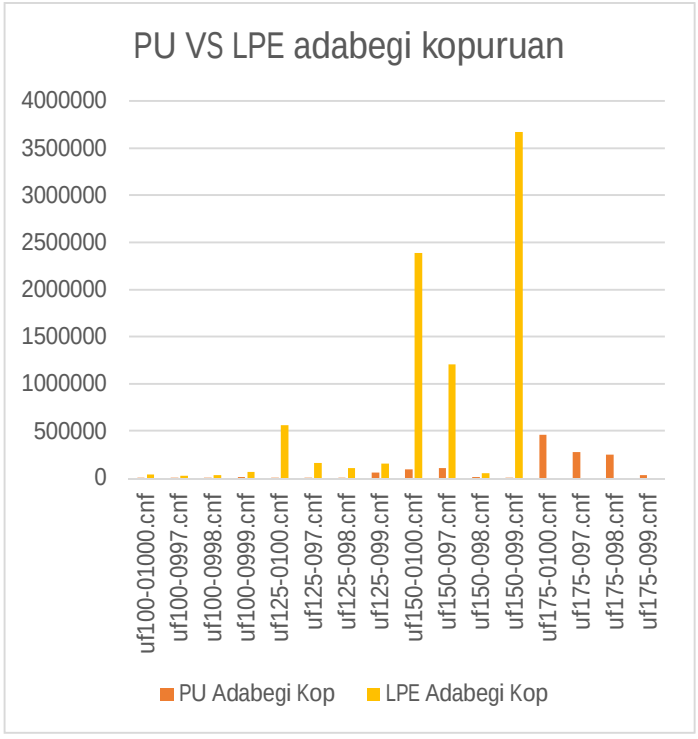


Grafiko 8: DA VS. DS adabegi kopurua [Ez SAT formulekin]

Behin aurreko guztia ikusita, jakin beharreko beste gauza bat propagazio unitarioa literal puruen ezabapena baino "indartsuagoa" dela (web sarrera batzuetan ikus daiteke). Izan ere, propagazio unitarioan klausulak ezabatu eta txikitu egiten ditugu, eta literal puruen ezabapenean, soilik ezabatu. Hau esplizituki ikusteko DA algoritmoaren beste bi bertsio egin ditugu (bakarrik frogak egiteko): batean PU utzi dugu soilik eta bestean LPE bakarrik. SAT formulekin frogak egin dira, eta PU unitarioa teknika zeukan algoritmoak azkarrago aztertu izan ditu formulak. Emaitzak Grafika 9 eta Grafika 10 grafiketan ikusgarri daude. Honekin ez dugu esan nahi LPE txarra denik, izan ere aurreko ataletan ikusi dugun bezala bi teknikak konbinatuz (algoritmo berean sartuta) emaitza onak ateratzen dira, formulak asko sinplifikatzen direlako.



Grafiko 9: PU VS LPE denboretan



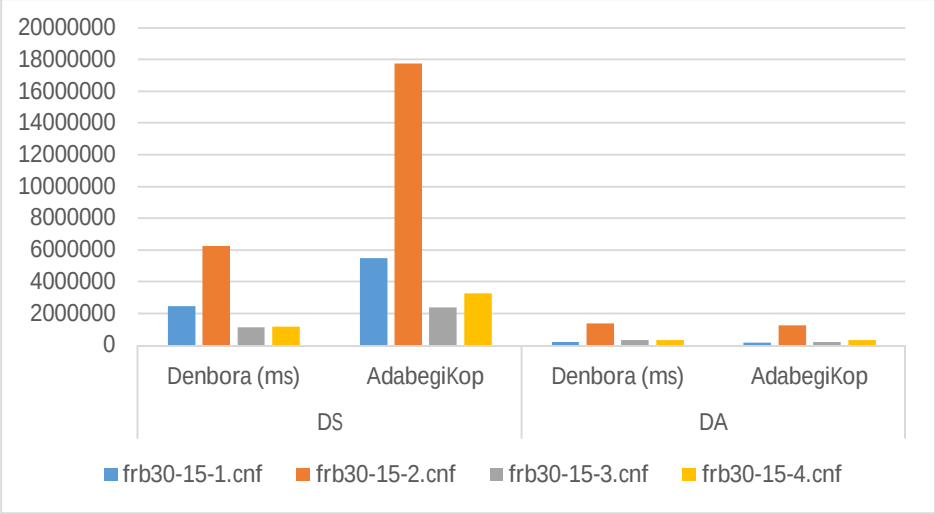
Grafiko 10: PU VS LPE adabegi kopuruan

4. *uf/uff250* aurrizkiko fitxategiak dituzten formulak izan dira gure mugak, hauek erabiltzerakoan exekuzioak ez baitu bukatu (lehenengoarekin *-uf250-0100.cnf-* programa eten egin behar izan dugu). Fitxategi hauetan 250 literaletako eta 1065 klausuletako formulak daude. Kuriositatez, handiagoak diren beste fitxategi batzuk hartuta (esteka honetatik), exekuzioak bukatu du. Beste fitxategi hauen ezaugarriak Taula 5 taulan jaso ditugu.

Fitxategi aurrizkia	Literal kopurua	Klausula kopurua
frb30-15-...	450	19084
frb30-17-...	595	29707

Taula 5: Tamaina handiko fitxategien ezaugarriak

Kasu honetan *frb30-17* fitxategiekin *OutOfMemory* errorea atera zaigu, Javaren makina birtualak erreserbatuta daukan memoria betetzen delako. Besteak aldiz, bi algoritmoekin exekutatu izan ditugu, eta hemen ere DA (*Dasgupta-Aurreratua*) algoritmoa garaile atera da. Emaitzak Grafiko 11 grafikan ikusgarri daude.



Grafiko 11: DA VS. DS tamaina handiko fitxategiekin

OHARRA: Baliteke aurreko formulekin exekuzioak amaitu izana klausula gehienak 2 literal dituztelako (nahiz eta formulak oso handiak izan). Formula bat bakarrik bi literaleko klausulak baldin baditu, orduan 2-SAT izenaz ezagutzen den probleman egongo gara, denbora polinomialean ebatz daitekeena. Beraz, kasu hauetan problema P klaseko problema multzoan erortzen da.

4. ETORKIZUNERAKO LANA

Praktika honen bidez *SAT Solver* txiki bat garatu izan dugu, problema famatu hau ezagutzeko eta bere ezaugarrietan murgiltzeko balio izan diguna. Ala eta guztiz ere, jakin badakigu *SAT Solver* onetatik nolabait urrun gaudela oraindik, heuristiko eta teknika gehiago sar zitekeelako, eraginkortasuna gehiago hobetzeko. Adibidez, badago betegarritasun problema ebazten duen *open source* algoritmo bat, miniSAT izenekoa (C++ lengoaian idatzia). Argitaratu dituzten artikuluak ikusita, erraz ohartaraziko gara problema honen ebazpena bizkortzeko teknika ugari daudela (*Conflict clause learning, Non-Chronological backtracking, two watched literals* teknika propagazio unitarioa eraginkorki kodetzeko, ...) gure algoritmoan sartu ez ditugunak.

Denbora gehiago izatekotan, honako bi puntuak garatzea gustatuko litzaidake:

- Formulen hedapena egiterakoan, nolabait formulen kopia aurreztea.
- Propagazio unitarioa, literal puruen ezabapena eta formula baten hedapena gauzatzerakoan formula guztia ez korritzea. Hau egiteko ideia batzuk izan ditut, adibidez, klausula guztiei identifikadore bat esleitu eta literal bakoitzeko zein klausuletan agertzen den gorde (klausula identifikadore zerrenda bat izango zuen literal bakoitzak).

5. ERANSKINAK

Atal honetan exekuzioetatik lortutako datuak formatu tabularrean atxikituko ditugu.

- DS algoritmoa, UNSAT formulekin

Fitxategia	Denbora (ms)	AdabegiKop
uuf50-01000.cnf	83	1707
uuf50-0997.cnf	32	1038
uuf50-0998.cnf	56	1803
uuf50-0999.cnf	64	2295
uuf75-0100.cnf	455	12130
uuf75-097.cnf	587	16128
uuf75-098.cnf	224	5161
uuf75-099.cnf	323	7735
uuf100-01000.cnf	5586	103809
uuf100-0997.cnf	4723	88645
uuf100-0998.cnf	3838	72518
uuf100-0999.cnf	2099	38381
uuf125-0100.cnf	26356	389927
uuf125-097.cnf	20510	296154
uuf125-098.cnf	38113	578810
uuf125-099.cnf	25500	391057
uuf150-0100.cnf	192959	2425330
uuf150-097.cnf	209958	2698770
uuf150-098.cnf	149486	1840287
uuf150-099.cnf	223268	2791645
uuf175-0100.cnf	1459373	15392300
uuf175-097.cnf	1123044	11514785
uuf175-098.cnf	1789482	18751707
uuf175-099.cnf	1051079	11277010
uuf200-096.cnf	5321275	47823761
uuf200-097.cnf	6441903	57632119
uuf200-098.cnf	5321275	47823761
uuf200-099.cnf	DENBORA ASKO	
uuf225-0100.cnf		
uuf225-097.cnf		
uuf225-098.cnf		
uuf225-099.cnf		

- DS algoritmoa, SAT formulekin

Fitxategia	Denbora (ms)	AdabegiKop
uf20-01000.cnf	3	95
uf20-0997.cnf	3	112
uf20-0998.cnf	3	88
uf20-0999.cnf	2	91
uf50-01000.cnf	60	1048
uf50-0997.cnf	32	817
uf50-0998.cnf	27	877
uf50-0999.cnf	18	456
uf75-0100.cnf	203	3721
uf75-097.cnf	218	5496
uf75-098.cnf	246	6151
uf75-099.cnf	162	3629
uf100-01000.cnf	429	6130
uf100-0997.cnf	2594	51728
uf100-0998.cnf	3066	58439
uf100-0999.cnf	2546	45397
uf125-0100.cnf	27618	482503
uf125-097.cnf	3853	54866
uf125-098.cnf	32607	541468
uf125-099.cnf	10642	164122
uf150-0100.cnf	13961	181687
uf150-097.cnf	17739	202350
uf150-098.cnf	52618	683569
uf150-099.cnf	127651	1686776
uf175-0100.cnf	104278	1072650
uf175-097.cnf	942465	10578033
uf175-098.cnf	557759	5665245
uf175-099.cnf	34110	368646
uf200-0100.cnf	1431166	13174976
uf200-097.cnf	6050850	58122979
uf200-098.cnf	DENBORA ASKO	
uf200-099.cnf		
uf225-0100.cnf		
uf225-097.cnf		
uf225-098.cnf		
uf225-099.cnf		

- DA algoritmoa, UNSAT formulekin

Fitxategia	Denbora (ms)	AdabegiKop
uuf50-01000.cnf	38	325
uuf50-0997.cnf	13	184
uuf50-0998.cnf	21	352
uuf50-0999.cnf	29	463
uuf75-0100.cnf	174	1792
uuf75-097.cnf	117	1531
uuf75-098.cnf	127	1771
uuf75-099.cnf	108	1450
uuf100-01000.cnf	922	7939
uuf100-0997.cnf	1362	13033
uuf100-0998.cnf	994	9412
uuf100-0999.cnf	532	4789
uuf125-0100.cnf	5284	37765
uuf125-097.cnf	4232	29638
uuf125-098.cnf	10417	75418
uuf125-099.cnf	5392	40198
uuf150-0100.cnf	45452	273619
uuf150-097.cnf	32300	200209
uuf150-098.cnf	37633	218824
uuf150-099.cnf	37030	223312
uuf175-0100.cnf	240616	1197973
uuf175-097.cnf	162184	757273
uuf175-098.cnf	496548	2528761
uuf175-099.cnf	185707	916300
uuf200-096.cnf	943703	3763672
uuf200-097.cnf	1055723	4321612
uuf200-098.cnf	2432639	10408942
uuf200-099.cnf	1923823	7792624
uuf225-0100.cnf	7029419	25444363
uuf225-097.cnf	11433268	39727864
uuf225-098.cnf	DENBORA ASKO	
uuf225-099.cnf		

- DA algoritmoa, SAT formulekin

Fitxategia	Denbora (ms)	AdabegiKop
uf20-01000.cnf	3	30
uf20-0997.cnf	3	24
uf20-0998.cnf	2	11
uf20-0999.cnf	3	11
uf50-01000.cnf	8	138
uf50-0997.cnf	10	185
uf50-0998.cnf	16	338
uf50-0999.cnf	13	264
uf75-0100.cnf	15	181
uf75-097.cnf	57	813
uf75-098.cnf	4	29
uf75-099.cnf	18	202
uf100-01000.cnf	79	492
uf100-0997.cnf	1022	9443
uf100-0998.cnf	295	2815
uf100-0999.cnf	1504	14028
uf125-0100.cnf	7134	56576
uf125-097.cnf	366	2909
uf125-098.cnf	1705	13784
uf125-099.cnf	5683	41975
uf150-0100.cnf	21328	128730
uf150-097.cnf	6501	44984
uf150-098.cnf	8485	52023
uf150-099.cnf	11706	72277
uf175-0100.cnf	184045	940884
uf175-097.cnf	18572	94084
uf175-098.cnf	39027	179608
uf175-099.cnf	16867	95046
uf200-0100.cnf	930741	3695278
uf200-097.cnf	761188	3367687
uf200-098.cnf	1337553	5631858
uf200-099.cnf	1093538	4620436
uf225-0100.cnf	10561513	38955538
uf225-097.cnf	991272	3288910
uf225-098.cnf	1312868	4692256
uf225-099.cnf	36864	143315

- DS algoritmoa, fitxategi haundiekin

Fitxategia	Denbora (ms)	AdabegiKop
frb30-15-1.cnf	2482830	5518422
frb30-15-2.cnf	6274084	17760302
frb30-15-3.cnf	1149723	2384256
frb30-15-4.cnf	1199786	3266516
frb35-17-1.cnf	OUT OF MEMORY	

- DA algoritmoa, fitxategi haundiekin

Fitxategia	Denbora (ms)	AdabegiKop
zfrb30-15-1.cnf	209905	179829
zfrb30-15-2.cnf	1383690	1261820
zfrb30-15-3.cnf	319793	236871
zfrb30-15-4.cnf	353943	334415
zfrb35-17-1.cnf	OUT OF MEMORY	

- PU VS PLE desberdintasunak ikusteko erabilitako denborak

Fitxategia	PU		LPE	
	Denbora (ms)	Adabegi Kop	Denbora (ms)	Adabegi Kop
uf100-01000.cnf	114	501	3209	41765
uf100-0997.cnf	1803	9897	2246	30133
uf100-0998.cnf	8727	1834	2829	36585
uf100-0999.cnf	2467	13603	5166	69505
uf125-0100.cnf	2175	8868	44368	564871
uf125-097.cnf	117	469	14266	160367
uf125-098.cnf	1933	8826	9912	107809
uf125-099.cnf	13581	61638	14535	154781
uf150-0100.cnf	25107	94516	231846	2390395
uf150-097.cnf	25792	109707	113742	1206409
uf150-098.cnf	3585	15047	6895	56331
uf150-099.cnf	2216	9001	342611	3673395
uf175-0100.cnf	147349	464069	DENBORA ASKO	
uf175-097.cnf	75725	276007		
uf175-098.cnf	74689	253297		
uf175-099.cnf	10029	35551		