eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Gradu Amaierako Lana

Informatika Ingeniaritzako Gradua

Konputazioa

Zenbakizko datuetan oinarritutako problema NP-Osoen Boolean SATisfiability problemarako laburketa bidezko ebazpenerako sistema

Mikel Oyarbide Goicoechea

Zuzendaria
Juan Miguel Lopez Gil

2023.eko ekainaren 25

Laburpena

Proiektu honetan NP problemekin egingo dugu lan. NP edo 'non polynomial' problemak teoria konputazionaleko problema multzo bat dira, non, hauen bereizgarri nagusia exekuzio denboran datzan.

Proba kasu txikiekin ordenagailu batek problema hauen emaitza segituan eman badezake ere, problema hauen sarrera datuak asko handituz gero, exekuta ezinak bihurtzen dira, soluzioa topatzeko proba kasu mordoa egin beharra daukalako ordenagailuak. Hainbeste, non, kasu askotan urteak edo mendeak beharko liratekeen emaitza bat lortzeko, superkonputagailu bat erabilita ere.

Problema hauen multzo bat da NP-Osoen multzoa. Hauen bereizgarria ondorengoa da, frogatuta dagoela badagoela modu bat, non, edozein NP problemari sarrera eraldatzeko modu bat dagoen, eta, NP-Osoa diren edozein problemaren sarrera moduan sartu daitekeen. Prozesu hau argiago azalduko da aurrerago, baina, bere izena laburketa da.

Laburketak izango dira proiektu honen muina. Hain zuzen, ikusi nahiko da, ea badagoen modu eraginkorrik NP-Osoak diren bi problema. laburketa bidez ebazteko. Subset Sum eta Knapsack problemak izan dira hautatuak, eta, laburketa hau SAT problemara egitea izango da asmoa.

Behin laburketekin amaitzean, web aplikazio baten bidez aipatutako bi problemak ahalik eta modu eraginkorrenean ebatziko dituen sistema sortzea izango da helburua. Proba kasu handiak, ahalik eta modu eraginkorrenean ebazteko diseinatuko da web aplikazio hau. Emaitza erakusteko, irudi egoki bat sortzea ere izango da erronka, erabiltzaileak emaitza hobeto interpreta dezan.

Gaien aurkibidea

Irudien aurkibidea

Taulen aurkibidea

KAPITULUA 1

SARRERA

Proiektuaren oinarriak eta motibazioa ondo ulertzeko, sarrera teoriko txiki bat azalduko da atal honetan. Hasteko, problemen konplexutasun konputazionalaz hitz egingo dut. Ondoren, laburketak eta NP problemak zer diren azaltzeaz gain, beraien arteko lotura aztertuko dut.

Behin oinarri teoriko hau edukita, proiektu honetan lortu nahi diren helburuak azalduko dira.

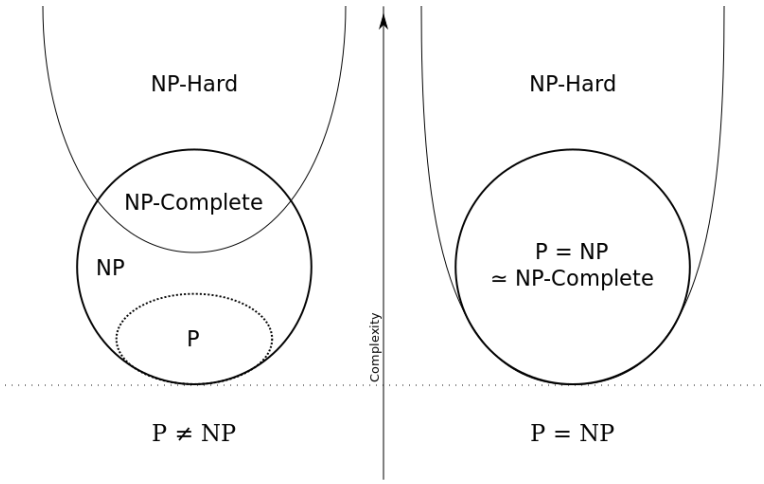
1.1 Konplexutasun konputazionala

Konplexutasun konputazionala, teoria konputazionalaren adar bat da, arazo konputazionalak berezko zailtasunaren arabera sailkatzean oinarritzen dena. Konplexutasun konputazionalaren teoria, baliabide kopuru jakin batekin ebatzi daitezkeen edo ezin diren arazoak sailkatzen saiatzen da, ordenagailuan egin daitekeenaren eta egin ezin denaren muga praktikoak zehaztuz.

Gai honen barruan, denbora polinomialeko algoritmoak eta denbora esponentzialeko algoritmoak desberdindu ditzakegu. Denbora polinomialeko algoritmoak, P (Polinomial) moduan ezagunak, sarrera datuak handitu ahala, algoritmoak exekutatzeko behar duen denbora polinomialki handituko direnak dira. Hau horrela izanda, ordenagailuak algoritmoa exekutatu eta ebatzi ahal izango du, sarrera datuak asko handituta ere.

Proiektu honetan landuko diren problemak, ordea, denbora esponentzialekoak, edo NP (Non Polinomial) motakoak izango dira. Hau da, algoritmoari sarrera datuak handitu ahala, ordenagailuak berau exekutatzeko behar duen denbora esponentzialki igoko da. Beraz, momentu bat iritsiko da, non, sarrera handituz joan ezkero ordenagailuak problema ebatzi ezingo duena.

Errealitatean P problemak, NP problemen azpimultzo bat dira, edozein P problema denbora polinomialean ebatzi daitekeenez, denbora esponentziala baino gutxiago izango delako. NP problemen artean bada beste azpimultzo interesgarri bat, ordea, NP-Osoen multzoa, alegia (ikusi 1.1 irudia).



1.1 Irudia: NP problema multzoa. [1]

NP-Osoak, NP problemen multzo bat dira, non, edozein NP problema, NP-Oso hauetako edozeinetara laburtu daitekeen.

1.2 Laburketak

Bi problemen artean laburketa bat egitea, lehen problemaren sarrera instantzia bat eraldatzea da, bigarren problemaren sarrerara egokitzeko. Modu honetan, bigarren problema ebatziko genuke, eta erantzun moduan lehen problemaren emaitza jasoko genuke, edo lehen problemaren emaitza moduan interpreta dezakegun emaitza bat, behintzat. [2]

Kontuan hartu behar da, laburketa prozesuak denbora polinomikoa izan behar duela, gerora azalduko den arrazoi batengatik.

Demagun p1 eta p2 problemak ditugula, eta p1 problema ebatzi nahi dugula p2 problemaren bidez. Egin beharreko ataza nagusiak bi lirateke.

1. Sarrera moldatu. p1 problemaren sarrera eraldatu egin beharko litzateke, p2 problemak sarrera moduan hartzen duen formatuaren arabera. Beti ere, kontuan hartuz, zer egiten ari garen eta zer den lortu nahi duguna. Lehen esan moduan, prozesu honek denbora polinomikoan egiteko modukoa izan behar du.
2. Emaitza interpretatu. Behin sarrera eraldatuarekin p2 problema ebatzi dugunean, emaitza hau interpretatu behar da. Emaitza honek p2 problemarako zer esan nahi duen ulertu, eta, irteera hau moldatu beharko dugu p1 problemaren emaitza gisa interpretatzeko.

Laburketen azalpen txiki hau, geroago azalduko dut zabalago, egin ditugun laburketak nola egin diren azaltzean.

1.3 NP eta NP-Osoak

Puntu honetan, NP-Osoen multzoa zer den azalduko dut. Kontuan hartu, P problemak, NP problemak eta NP-Osoak dauzkagula, 1.1 irudian ikusten dugun moduan. P problemak,

denbora polinomialean ebatz daitezkeen problemak dira. Hau da, sarrerako elementu kopurua handituta ere, denbora gutxian exekutatu ahalko dira.

Guri interesatzen zaizkigun multzoak, ordea, NP eta NP-Osoen multzoak dira. NP problemak, sarrera datu kopurua handitzen joan ahala, exekuzio denbora esponentzialki handitzen zaien problemak dira. Konbinaketa posible guztiak egin behar diren kasu baten moduan. Adibide moduan, jo dezagun l lista bat daukagula, 2 elementu dituena:

$$\{1, 2\}$$

Gure problemak konbinaketa guztiak egin behako balitu, hauek lirateke konbinaketa posibleak:

1. lista hutsa
2. 1
3. 2
4. 1,2

Konbinaketa kopurua 4koa litzateke. Ikus dezagun, zer pasatzen den elementu bat gehituz gero l lista honi:

$$\{1, 2, 3\}$$

Hauek lirateke konbinaketa posibleak:

1. lista hutsa
2. 1
3. 2
4. 3
5. 1,2
6. 1,3
7. 2,3
8. 1,2,3

Ikus daitekeen moduan, elementu bat gehitzearekin, konbinaketa posible kopurua 4tik 8ra pasa da, bikoitza, alegia. Berdina gertatuko litzateke, beste bat gehituko bagenu, 16ra joango litzateke konbinaketa kopuru posibleen kopurua.

Zer esanik ez, l lista honen elementu kopurua 10, 100, 1.000 edo 10.000 izango balitz. Momentu bat iritsiko litzateke, non, konputagailu batek, urteak edo mendeak beharko lituzkeen exekuzioa osatzeko. Mota honetako problemak dira, beraz, NP problemak.

Kontuan hartu behar den beste gai bat ondorengoa da, ordea. Ez dagoela matematikoki frogatuta, NP problema hauek ebazteko, denbora polinomialeko ebazpenik ez dagoenik. Hau da, nahiz eta, ezaugarri guztiak ikusita, problema hauek denbora polinomikoan ebaztea ezinezkoa dela dirudien, inork ez du lortu oraindik hau frogatzerik. Hemen sartzen da jokoan NP-Osoen multzoa.

NP problema bat, NP-Osoa dela esateko, ezaugarri hau bete behar du. Demagun NP-Osoen multzoan dagoen p problema bat daukagula. Multzo honetan egoteak, esan nahiko luke, NP problemen multzoan dagoen edozein problematik, p problema honetarako laburketa egin daitekeela. Gogoratu, laburketa honek denbora polinomikoan egindakoa izan behar duela.

Demagun, NP-Osoa den p problema honentzat, denbora polinomikoan egin daitekeen ebazpen posible bat topatzen dela. Edozein NP problematik, p problema honetara laburketa polinomikoa egin daitekeenez, NP problema guztientzat soluzio polinomikoa aurkitu dugula esan nahiko luke honek.

Beraz, NP-Osoa den edozein problemarentzat denbora polinomialeko ebazpen bat topatuko balitz, P=NP dela frogatuko litzateke.

1.4 Proiektuaren helburuak

Proiektu honen helburu nagusia NP-Osoak diren bi problema konkretu laburketa bidez ebaztea izango da . Hain zuzen, ondorengo biekin egin da lan proiektu honetan: Subset Sum problema eta Knapsack problema.

Aurrerago zabalago azalduko dira problema hauek, baina, hona hemen azalpen labur bana:

- Subset Sum problema, teoria konputazionaleko problema bat da, non, lista bat eta helburu zenbaki bat ematen diren sarrera moduan. Jakin nahi dena da, ea badagoen lista horren azpimultzorik, non, batura moduan helburu zenbakia izango duen.
- Knapsack problema (motxilaren problema moduan ere ezaguna), teoria konputazionaleko optimizazio problema bat da, non, kasu honetan pisuen lista bat, balioen lista bat eta motxilaren tamaina ematen diren sarrera moduan, pisu bakoitzak balio bat esleituta duelarik. Lortu nahi dena da, motxilaren tamaina pasatzen ez duen pisuen lista bat lortzea, baina, ahalik eta balio handiena lortuz.

Problema bata zein bestea, SAT (Boolean Satisfiability Problem) problemara laburtu, eta, SAT solver baten bidez (ondoren azalduko da SAT solverra zer den), ebaztea izango da helburua.

SAT problema hau ere, aurrerago luze eta zabal azalduko da, baina, laburbilduz ondorengoa litzateke:

- SAT problema honek, sarrera moduan, CNF formatuan (Forma Normal Konjuntiboa) dagoen formula bat hartzen du. Formatu hau, laburbilduz, balio negatibo edo positiboak hartu ditzaketen literalez osaturiko azpilisten lista bat da. Formula honekin, adierazpen boolear posiblerik dagoen begiratzen du, formula egiazko egingo duena.

Esan moduan, ondoren azalpen zabalago bat emango da, adibide eta guzti. Momentuz jakin beharrekoa da, egin nahi dena, Subset Sum eta Knapsack problemak ebaztea dela SAT-erako laburketa erabilita. Laburketa prozesu honetan, SAT problemaren inguruan lan egiteko sorturiko Pysat izeneko liburutegi bat erabili nahi da.

Pysat liburutegi hau, SAT problemen inguruan lan egiteko sorturiko liburutegi bat da. SAT problemen eragiketa ezberdinak errazteko, eta, modu eraginkor batean egin ahal izateko sorturiko Pythoneko liburutegia da, eta, hainbat modulu ditu. Modulu bakoitzak funtzio multzo bat du, problema honen inguruan gauza ezberdinak garatzea ahalbidetzen duena.

Kontuan eduki behar da, NP problemak direnez, denbora eraginkortasuna dela hemen beste erronka garrantzitsu bat. Laburketa hauek, modu eraginkor batean egin nahi dira. Hau da, baliteke ebazpen posible bat topatzea, baina, proba kasu handiak egiteko denbora gehiegi behar izatea, edo zuzenean proba kasu handiak direnean blokeatzea. Gure asmoa, beraz, ebazpen posible bat bilatzea izango da, non, exekuzio denbora aldetik, sistema eraginkorra izango den.

Proiektu honetan egin nahi diren bi laburketek, beraz, bi laburketak egiteaz aparte, ondorengoa ere lortuko lukete. Gure kasuan erabiliko ditugun 3 problemak NP-Osoak direla frogatuta dagoenez, honek esan nahi du, edozein NP problema, Subset Sum eta Knapsack problemara laburtu daitekeela. Beraz, bi problema hauetatik SAT-erako laburketa lortuko bagenu, errealitatean NP problema guztientzat balio duen SAT bidezko ebazpen bat sortzea lortuko genuke.

Behin laburketak osatu direnean, web aplikazio bat garatzea izango da helburu nagusia. Web aplikazio honek, Subset Sum eta Knapsack problemak ebazteko aukera emango du, modu eraginkor batean.

Nik sorturiko laburketak, zenbateraino eraginkorrak diren ikusteko, beste solver espezifiko bat erabili dugu. Hain zuzen, Google-k sortutako Ortools liburutegiko Knapsack solverra. Solver honek, askoz modu eraginkorragoan ebazten ditu aipatutako bi problemak (gerora azalduko dira honen nondik norako guztiak).

Hau ikusita, eta web aplikazio eraginkor bat egiteko asmoa izanda, Ortools liburutegiarekin egindako bertsioak erabili ditut web aplikazioan. Denbora alderaketa guztiak ere ikusgarri izango dira lortutako emaitzen atalean.

Web aplikazioa eraginkorra izateaz gain, emaitzen interpretazio grafiko egoki bat lortzea izango da helburua. Horretarako, sortuko den irteera irudi honek nolakoa izan behar duen hausnartu beharko da.

Gainera, sarrera datu handiak ebatz ditzakeen sistema sortu nahi denez, emaitzak ere elementu asko izango ditu, beraz, irudi moduan emaitzaren interpretazio egoki bat sortzea erronka garbia izango da. Ez baita berdina, proba kasu txiki baterako irudia sortzea, edo, ehunka elementu dituen emaitza baterako sortzea.

KAPITULUA 2

PLANGINTZA

Kapitulu honetan Proiektuaren garapenean egin den plangintza azalduko da. Hasteko, egin beharreko lana, lan-pakete txikiagotan banatu dela erakutsiko da. Gero, lan-pakete hauek guztiak azaltzeaz gain, bakoitza osatzeko pasatako denborak azalduko dira.

Entregatu beharreko emangarriak zein diren eta hauek egiteko lan metodologia azaltzeaz gain, erabilitako informazio eta komunikazio sistemak zeintzuk izan diren esango da.

Arrisku plana egin ondoren, berriz, jarraipena eta kontrola egingo da, denboraren desbiderapen taula bat erakutsiko delarik.

2.1 Lanen Deskonposaketa Egitura diagrama (LDE)

Proiektuaren antolaketa egoki bat egituratzeko, Lanaren Deskonposaketa Egitura (LDE) diagrama osatu da. Horrela, garbiago ikusi ahal izango da proiektu hau garatzeko osatu behar izan diren lan multzo ezberdinak (ikus 2.1 irudia)).

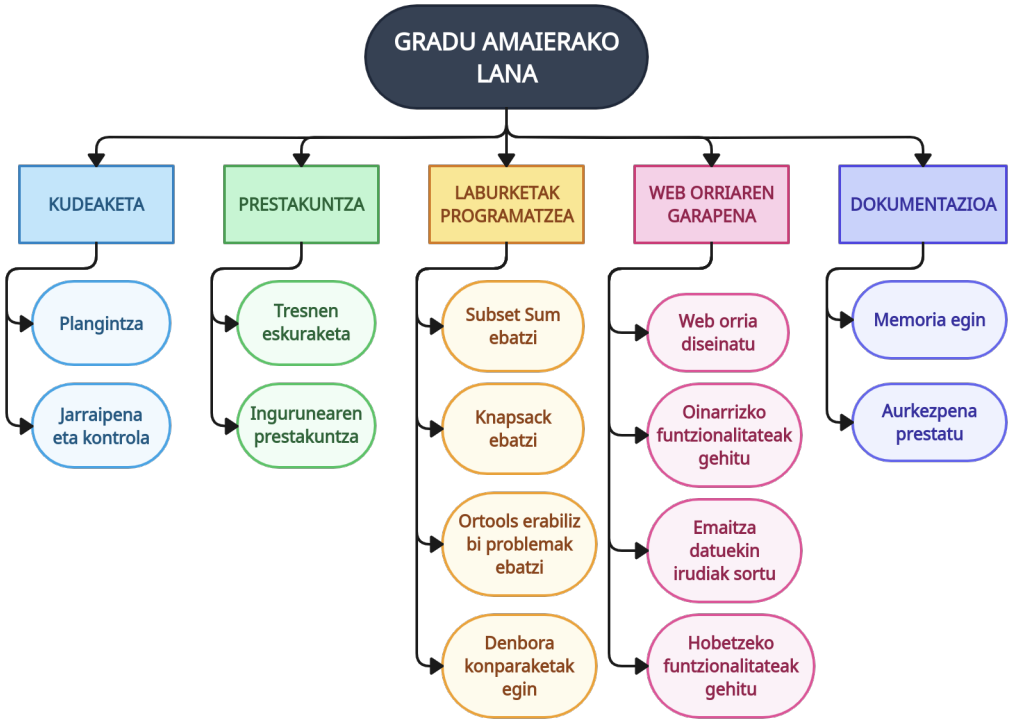
2.2 Lan-paketeak

Atal honetan, ataza bakoitza osatzeko hasieran egindako estimazioa agertuko da. Jarraian, benetan lanean pasatako ordu errealak agertuko dira. Diferentzia dago estimaziotik ordu errealetara, hasieran aurreikustea zaila den arazoak egon direlako.

Ondorengo lan-paketeetan banatu da proiektuaren garapena:

1. Kudeaketa: Lan-pakete honetan proiektuaren antolamendua eta kudeaketa izango dira helburuak. Ataza hauetan banatu da lan-paketea:

    a) Plangintza: Proiektuaren hasieran, jarraitu beharreko urratsen zehaztapen bat egin da. Proiektuaren garapena antolatzeko lan-paketeak eta hauetako bakoitzaren atazak planteatu dira. Lan-metodologia definitzeaz aparte, orduen estimazioa egin eta arriskuen kudeaketa zehaztu dira.



2.1 Irudia: LDE diagrama

b) Jarraipena eta kontrola: Proiektuaren garapen egokia bermatzeko definitu da atal hau. Proiektuaren garapenaren puntu bakoitzean, esperotako helburuak betetzen ari diren, eta, hurrengo urratsak argi definituta dauden begiratuko da.

2. Prestakuntza: Lan-pakete honetan proiektua garatzeko beharrezko ezagutza eta prestakuntza eskuratzeaz gain, behar diren tresnak definitu eta prestatzea izango da helburua. Ataza hauetan banatu da lan-paketea:

   a) Tresnen eskuraketa: Proiektuan zehar beharko ditugun tresnak identifikatuko dira atal honetan. Beharrak aztertuta, kasu bakoitzerako egokienak irizten diren tresnak identifikatu eta eskuratuko dira.

   b) Ingurunearen prestakuntza: Behin tresna egokiak identifikatu eta eskuratuta, lan egiteko behar dugun ingurunea prestatu beharko dugu.

3. Laburketak programatzea: Lan-pakete honetan proiektuaren muinetako bat den atala osatzea izango da helburua. Subset Sum eta Knapsack problemak laburtu nahi dira SAT problemara, SAT solver baten bidez ebatzi ahal izateko. Ataza hauetan banatu da lan-paketea:

   a) Subset Sum ebatzi: Atal honetan Subset Sum problema ebaztea izan da helburua SAT problemara laburketa eginez. Ondoren azalduko dira prozesu honen nondik norakoak.

   b) Knapsack ebatzi: Atal honetan, aldiz, Knapsack izan da ebatzi nahi izan den problema. Hau ere SAT problemarako laburketa eginez, ondoren azalduko den moduan.

4. Web orriaren garapena: Lan egin nahi den bi problema hauek ebazteko erabiliko den web aplikazioa sortuko da hemen. Erabiltzaileak sarrera datu handiak ahalik eta modu eraginkorrenean ebatzi ahal izateko. Ataza hauetan banatu da lan-paketea:

   a) Web orria diseinatu: Lan-pakete honen hasieran web orriaren diseinua definitu beharko da. Horretarako, kontuan hartuko dira garatu nahi diren funtzionalitateak zeintzuk diren.

   b) Oinarrizko funtzionalitateak gehitu: Behin diseinua eginda edukita, martxan jartzea izango helburua. Oinarrizko funtzionalitateak gehituko dira, hala nola, sarrera moduan datuak idatziz sartzeko aukera eman, eta, aplikazioak emaitza datuak itzultzea. Horretarako, Subset Sum eta Knapsack problemak ebatziko dituen algoritmo benetan eraginkor bat topatu beharko dugu.

   c) Emaitza datuekin irudiak sortu: Irudiaren sorkuntza izango da beste ataza bat. Emaitza datuekin irudi bat sortu nahi da, emaitza ulergarriago egiteko. Datu handiekin lan egingo denez, datu askorekin ondo ikusiko den irudi egokitua sortzea ezinbestekoa izango da.

   d) Hobetzeko funtzionalitateak gehitu: Oinarrizko web aplikazioak ondo funtzionatzen duela ikustean, funtzionalitate aurreratuagoak sartzea izango da asmoa. Sarrera moduan fitxategiak sartzeko aukera ematea, emaitza deskargatzeko aukera ematea eta sortutako irudia deskargatzeko aukera ematea izango dira hauetako batzuk.

5. Dokumentazioa: Lan-pakete honetan proiektuaren dokumentazioaren garapena egingo da. Ataza hauetan banatu da lan-paketea:

   a) Memoria: Atal honetan proiektuaren azalpenak, emaitzak eta nondik norako guztiak azalduko diren dokumentua sortuko da. Besteak beste, sortutako algoritmoen eraginkortasuna azaltzen den taula bat egin beharko da bertan.

   b) Aurkezpena: Aurkezpenaren atalean proiektuaren defentsan erabiliko den materiala prestatuko da. Bertan proiektuaren laburpen bat jasoko da, egindako lana eta lortutako emaitzak azalduko direlarik.

2.3 Emangarriak

Ondorengoak izango dira entregatuko diren bi emangarriak:

- Memoria
- Aurkezpena

2.4 Denboraren kudeaketa eta Gantt diagrama

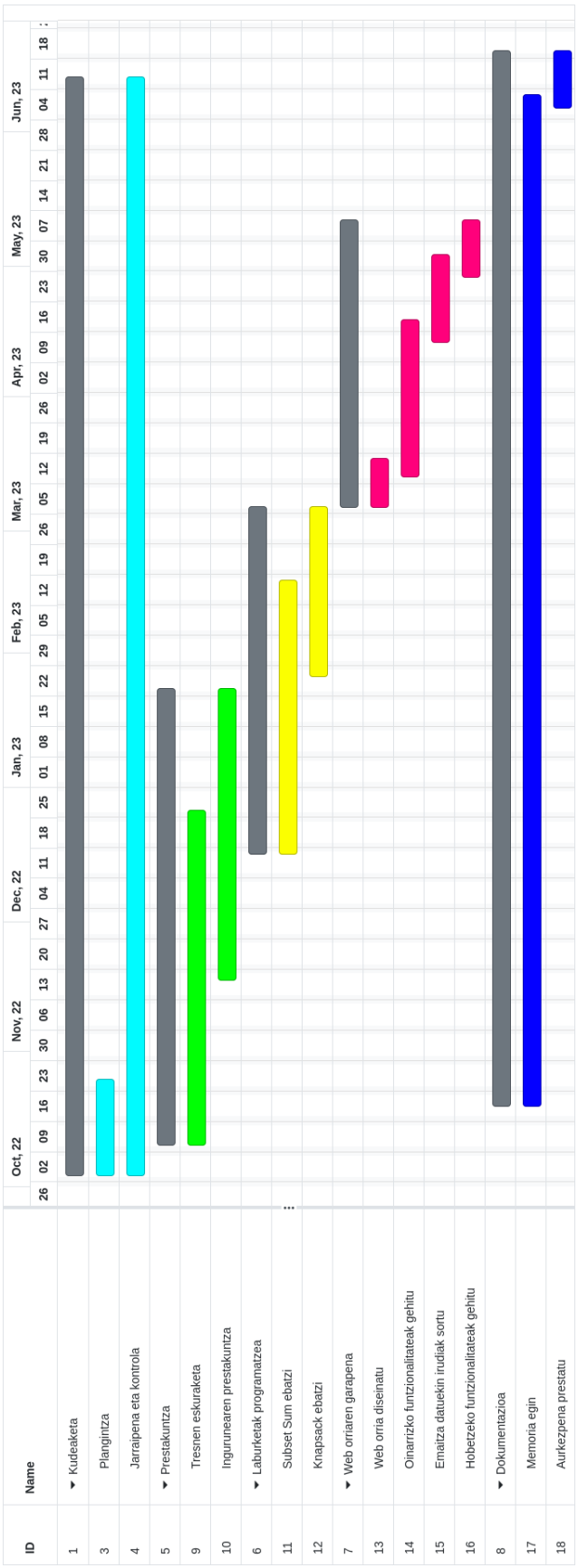
Atal honetan proiektua amaitzeko eta entregatzeko mugarriak azalduko dira. Hau osatzeko, proiektuaren garapenaren inguruko ataza bakoitza noiz hasi eta amaitu den azalduko da. Horretarako, Gantt diagrama bat osatu da, lan-pakete bakoitza kolore ezberdinez sailkaturik dagoelarik (ikusi 2.1 irudia).

Diagraman deigarria gerta daiteke Kudeaketa eta Prestakuntza lan-paketeetan aste asko pasa direla. Hau azaltzeko, esan behar da proiektu hau garatzeko ezagutza zati handia lehen lauhilekoan eman den Konputagailu Eredu Abstraktuak (KEA) irakasgaian eman zela. Beraz, naiz eta, proiektuaren garapena urrian hasi, lehen aste guzti horietan ezagutza hau barneratzea izan zen helburua.

Horretaz aparte, plangintza antolatu, lan ingurunea prestatu eta proiektuko zuzendariarekin bilerak egin ziren ezagutza hau ahalik eta egokien barneratzeko eta proba ezberdinak egiten joateko.

Entrega datak:

- Memoria: 2023/06/25
- Aurkezpena: 2023/07/03-2023/07/14



2.2 Irudia: LDE diagrama

2.5 Lan metodologia

Proiektuaren garapen lan gehiena ehu-ko Donostiako liburutegian egin da. Horrela, ziurtatu nahi izan dut liburutegian nagoen denbora guztian proiektuan zentratuta nagoela. Egia da etxetik eginda bidaia denbora aurreztu nezakeela, baina, liburutegia lan egiteko prestatutako toki egokia denez, nahiago izan dut bertan lan egin, inongo distrakziorik gabe.

Lana aurrera eramateko moduari dagokionez, momentu bakoitzean momentuko probleman zentratzea izan da asmoa. Laburketetan, problema bakoitzean, ebazpen modu eraginkorragoak bilatu eta proba kasu ezberdinak egitea izan da asmoa.

Ondoren, ebatzi nahi diren bi problemak ebazteko sistema eraginkorrena lortu, eta web aplikazioa sortzea izan da helburua.

2.6 Informazio-sistemak

Proiektua garatzean sortutako material guztia lokalean eta hodeian gordeko da. Honako iturri hauek erabiliko dira, hain zuzen:

I. Ordenagailu pertsonala: Dokumentazioa egiten joan ahala, segurtasun kopiak ordenagailuan lokalean gordeko dira. Berdina egingo da, laburketen eta web aplikazioaren artxibo guztiekin.

II. Google Drive: Lokalean gordetako guztia Google Drive-en ere gordeko da. Horrela, segurtasun bikoitza izango da. Ordenagailuan arazoren bat izango balitz, hodeian izango genukeelako segurtasun kopia bakoitza.

III. Overleaf: Dokumentazioa egiteko, Latex-en oinarritutako Overleaf konpiladorea erabiliko da. Dokumentu zientifikoak idazteko eta editatzeko sorturiko tresna honek, hodeian lan egiten duenez, dokumentazioa aldi oro bertan gordeta egongo da.

2.7 Komunikazio-sistemak

Proiektuaren garapen egokia bermatzeko, zuzendariarekin komunikazioa mantentzea ezinbestekoa da. Horretarako, ondorengo bi moduak erabili dira:

I. Bilera presentzialak: Proiektuaren jarraipen egokia egiteko bi astez behin proiektuko zuzendariarekin bilerak egin dira bere bulegoan. Proiektuaren nondik norakoak azaldu eta jarraitzeko hurrengo pausoak zehazten joan dira.

II. Posta elektronikoa: Zalantzak galdetzeko, intereseko materiala partekatzeko edo bilera orduak adosteko erabili da. Horrez gain, proiektuaren inplementazioa partekatzeko ere erabili da.

2.8 Arrisku plana

Atal honetan proiektuaren garapenean aurki daitezkeen arriskuak zerrendatuko dira eta arrisku hauek minimizatzeko hartu diren neurriak azalduko dira.

2.8.1 Arriskuak

Bost arrisku identifikaztu ditut denera. Hona hemen zerrendatuta:

i. Laburketa egokirik ez lortzea. Proiektu honetako arrisku garbi bat laburketarik ez lortzea izango litzateke. Proiektuaren helburu nagusietako bat Subset Sum-etik eta Knapsack-etik SAT-erako laburketak lortzea da, eta, hauekin web aplikazio eraginkor bat sortzea. Hainbat proba egin ondoren, hau egitea lortuko ez balitz, honek proiektuko helburu nagusi bat lortu gabe geratzea ekarriko luke.

ii. Erabilitako ordenagailuaren limitazioak. Nahiz eta, SAT problemara ebatzi nahi diren bi problemak ebaztea lortu, problema hauek duten konplexutasun maila handia dute. Gerta daiteke, gure problema ebazten duen algoritmo bat sortzea, baina, nire ordenagailuan errekurtso gehiegi hartzen dituenez, egindako sistema ondo probatu ezin ahal izatea.

iii. Egindako lanaren galera: Proiektuan egin den lana galtzea, ezabatua izan delako edo bertsioren bat ondo gorde ez delako.

iv. Ordenagailu galera: Proiektua garatzeko erabili den ordenagailua galtzeak, lokalean gordetako babes kopia guztiak galtzea ekarriko luke. Gainera, sistema-eragilea eta programa guztiak berriz instalatu beharrak denbora galera garrantzitsua ekarriko luke.

v. Denbora falta: Atalen bat garatzeak espero baino denbora gehiago hartzeak edo informazio zatiren bat galtzeak kontuan hartzeko moduko denbora atzerapena ekar dezake.

2.8.2 Irtenbideak

Arrisku bakoitzarentzat irtenbide bana aurreikusten saiatu naiz. Horrela, arrisku hauetakoren bat egi bihurtuko balitz, irtenbidearen planteamentua eginda edukiko nuke.

i. Laburketa egokirik ez lortzea. Kasu honetan, hainbat proba egin eta gero, laburketarik lortuko ez balitz, edo behar adina eraginkorra izango ez balitz, laburketatik aparte, problema bakoitzarentzat ebazpen metodo zuzen bat bilatu beharko litzateke web aplikazioa egiteko. Gainera, lortu ez den laburketaren zergatiak eta ondorioak jarri beharko lirateke memorian.

ii. Erabilitako ordenagailuaren limitazioak. Problema mota hauek errekurtso asko eskatzen dutela jakitun, lanerako ordenagailu egoki bat erabiltzen saiatuko naiz. Ebazpen algoritmo bat sortzea lortu badut, lehendabizi proba kasu txikiekin probatuko dut. Honez gain, proba kasu handiekin arazoak ematen badizkit, ikusi beharko da beste ordenagailu bat erabilita exekuzioa osatu ahal izango litzatekeen, edo, algoritmoa bera den hobetu daitekeena.

iii. Egindako lanaren galera: Egindako lana ordenagailuan bertan (lokalean) eta Google Drive-en (hodeian), gorde da, aldaketa nabarmen bakoitzeko segurtasun kopia bat eginez.

iv. Ordenagailu galera: Hau gertatuz gero, fakultatean lanerako prestatuta dauden ordenagailuetan jarraitu beharko litzateke beste ordenagailu pertsonal bat lortu arte, honek dakarren denbora galera guztiarekin.

v. Denbora falta: Lehentasunak finkatu eta proiektuan garrantzi gehien duen ataletik hasi. Horrela, denbora faltagatik atalen bat ez bada erabat osatuta geratu, hau konplementu bat izan dadila eta ez proiektuaren muina.

2.9 Jarraipena eta Kontrola (K.J)

Amaitzeko, atal honetan, proiektuaren garapenaren eta arazoen jarraipena eta kontrola ikusiko dugu. Hasieran egindako denboraren planifikazioa ikusi ahal izango da eta desbiderapen taula baten bidez (ikus 2.1 taula), denboran izan diren desbiderapenak azalduko dira.

ATAZAK	ORDU ESTIMAZIOA	ORDU ERREALAK	DESBIDERAPENA
Kudeaketa	10	10	0
Plangintza	3	3	0
Jarraipena eta kontrola	7	7	0
Prestakuntza	40	30	-10
Tresnen eskuraketa	30	20	-10
Ingurunearen prestakuntza	10	10	0
Laburketak programatzea	75	103	+28
Subset Sum ebatzi	45	75	+35
Knapsack ebatzi	30	28	-2
Web orriaren garapena	105	113	+8
Web orria diseinatu	5	5	0
Oinarrizko funtzionalitateak gehitu	40	43	+2
Emaitza datuekin irudiak sortu	35	43	+5
Hobetzeko funtzionalitateak gehitu	25	22	-3
Dokumentazioa	78	76	-2
Memoria egin	70	68	-2
Aurkezpena prestatu	8	8	0
Guztira	**308**	**332**	**+24**

2.1 Taula: Orduen desbiderapen taula

Ikusten den moduan, Prestakuntza atalean espero baino 10 ordu gutxiago eman dira. Hasiera batean, prestakuntzan ordu gehiago pasako zirela pentsatzen genuen, baina, esan bezala, proiektu honek KEA irakasgaiarekin lotura zuzena duenez, bertan jaso da behar izan den ezagutza zati garrantzitsu bat.

Bestalde, laburketak programatzen uste baino askoz denbora gehiago pasa dela esan behar da. Hasieran, Subset Sum-erako lehendabizi pentsaturiko algoritmoa nahikoa eraginkorra izango zela pentsatzen zen. Ondoren zabalago azalduko den moduan, uste baino limitazio handiagoak zituenez, laburketa beste modu batean egitea pentsatu zen. Knapsack ebazteko ere, Subset Sum-erako sorturiko algoritmoa erabili nahi zenez, knapsack-eko bertsio hau ere ez zen batere egokia gertatu.

Prozesu honetan espero baino denbora gehiago joan zen azkenean algoritmo egoki bat osatu genuen arte. Hau da beraz, desbiderapen honen zergati nagusia. Irudiaren sorketak ere, espero baino denbora gehiago kendu dit, irudi oso handien bistaraketa denez, hainbat arazo izan baititut, irudiaren pisua laukitxo bakoitzaren barnean atera gabe agertzea edo Zoom-a egoki aplikatzea, besteak beste. Beste atazak espero moduan joan dira gutxi gora behera.

KAPITULUA 3

PROBLEMEN AZALPENA ETA SOLVER BIDEZKO EBAZPENA

Proiektu honetan Subset Sum eta Knapsack problemak, SAT problemara laburtu nahi direnez, hiru problema hauek azalduko dira jarraian, adibideak jarriz. Ondoren SAT solver bat zer den eta laburketak nola egiteko asmoa dagoen erakutsiko da.

3.1 Subset Sum problema

Subset Sum problema teoria konputazionaleko problema bat da [3]. Sarrerako azalpenean problema honen inguruko azalpen bat eman da. Hobeto uler dadin, adibide sinple bat jarriz azalduko dut jarraian.

Esan moduan, problema honek lista bat eta zenbaki bat hartzen ditu sarrera moduan. Demagun ondorengo sarrera datuak ditugula:

- l=[3,2,4,8,6]
- z=15

Ikusi beharko litzateke, ea badagoen l listako azpimultzorik, non, beraien batura 15 izango den. Gure kasuan, listak 5 elementu besterik ez dituenez, aukera bat topatzea ez zaigu zaila egingo. Adibidez, l listako [3,4,8] azpimultzoa hartuko bagenu, 3+4+8=15 emango luke emaitza gisa, hau da, lortu nahi dugun emaitza.

Beste emaitza posible bat [3,2,4,6] azpimultzoa litzateke. Edozein kasutan ere, konbinaketa guztiak egin beharko lirateke emaitzarik baduen ikusteko, kasu txarrenean behintzat. Konbinaketa hauek guztiak egin ondoren, emaitzarik topatuko ez balu, ezezkoa itzuliko luke emaitzak.

Emaitza bat topatzen duen arte konbinaketa posible guztiak egin beharrak bihurtzen du problema hau NP problema. Izan ere, sarrera lista handitzen joan ahala, egin beharreko konbinaketa kopurua esponentzialki igotzen baita.

3.2 Knapsack problema

Problema hau, motxilaren problema moduan ere ezagutzen da [4]. Aurreko problemaren ildo beretik doa, baina, kasu honetan zenbakiaz gain, bi lista izango ditugu sarrera moduan. Pisuen lista, batetik, eta, balioen lista, bestetik.

Problema ondo ulertzeko, irudika dezagun motxila bat daukagula, tamaina konkretu bat duen motxila bat hain zuzen. Motxila honetan nahi adina elementu sartu ditzakegu, motxilaren tamaina pasatzen ez dugun bitartean.

Elementu bakoitzak motxilan okupatzen duen lekua pisuen listak definitzen du. Beraz, motxilan sartzen ditugun elementuen pisuen baturak, ezingo du inolaz ere motxilaren tamaina gainditu.

Elementu bakoitzak, pisu bat izateaz gain, balio bat ere badu beregan. Hau, balioen listak definitzen du, noski. Beraz, esan daiteke bi lista hauetan, elementu bakoitzaren pisua eta balioa, posizio berdinean dauden pisuen eta balioen listako elementuak ematen dutela.

Helburua, beraz, motxilan zer elementu sartu erabakitzea izango da, non, motxilaren tamaina pasa gabe, ahalik eta balio handiena lortzen den. Jar dezagun adibide bat hau ondo ulertzeko.

Demagun, hauek direla sarrera datuak:

- pisuLista=[3,2,7,4]
- balioLista=[8,3,3,1]
- z=10

Kasua aztertuta, ikus daiteke motxilaren tamaina z=10 dela. Beraz, pisuen listako elementu batzuk hartu beharko ditugu, non, tamaina hau pasatzen ez duten. [3,7] pisuak hartzen baditugu, pisua=10 izango litzateke, baliozkoa beraz. Lortutako balioa, 8+3=11 izango litzateke.

Emaitza egokia dirudien arren, badago beste hobeago bat. [3,2,4] pisuak hartuta, pisua=9 izango litzateke, motxila bete gabe utziz. Baina, lortutako balioa handiagoa litzateke, 8+3+1=12 hain zuzen.

Ikusi moduan, ez dago zertan motxila erabat bete beharrik, hau balio hobea lortzeko modua bada. Beraz, kasu honetan ere, konbinaketa posible guztiak egin beharrean aurkitzen gara, honek dakarren denbora kostuarekin. Subset Sum problemaren antzera, sarrera handiagoak hartzean, konbinaketa kopurua esponentzialki handitzen denez, exekuzio denbora ere maila berean handitzen da.

3.3 SAT problema (Boolean Satisfiability problem)

Azaldutako bi problema hauek, SAT problema honetara laburtu nahi dira. SAT problema, NP-Oso bezala identifikatu zen lehena izan zen. 1971 urtean, Stephen Cook zientzialaria izan zen hau frogatu zuena. Bere lana, artikulu zientifiko batean argitaratu zuen [5], eta, artikulu hau oso famatua egin zen momentuan.

Goazen problema hau azaltzera. SAT problema, erabakitze problema bat da, non, aldagaiez osaturiko adierazpen boolear bat emanda, emaitza moduan, aldagai hauentzako asignazio posiblerik badagoen itzultzen duen, adierazpena egia egingo duena.

Ondoren jarriko dudan adibidearekin askoz argiago ulertu ahal izango da zehazki zer den SAT problema honek egiten duena. Hau ulertzeko, ordea, sarrera datuen inguruko termino batzuk azaltzea ezinbestekoa da.

SAT problema honek sarrera moduan CNF (Forma Normal Konjuntiboa) formatuan dagoen formula bat hartzen du. CNF formatua, klausulen konjuntzio multzo bat da, eta, klausula bakoitza, literalen disjuntzioz osaturiko multzoa. Adibide batekin azalduko ditut termino hauek. Hona hemen, CNF formatuan dagoen formula bat:

$$(x1 \vee \neg x2) \wedge (x1 \vee x3) \wedge (x2 \vee \neg x3) \wedge (\neg x1 \vee x3)$$

Konjuntzioz separatuta dauden lau ataletako bakoitza klausula bat izango litzateke. Lehen klausula, $(x1 \vee \neg x2)$ litzateke, esaterako. Klausula barruan dagoen elementu bakoitza, aldiz, literala izango litzateke. Klausula hau x1 eta ¬x2 literalez osatuta dago, kasu honetan.

Literalak, izatez, aldagai bakoitzak hartu dezakeen balio positiboa edo negatiboa dira, hau da, aurreko adibidean, 3 aldagai izango genituzke: x1, x2 eta x3. Klausula bakoitzean, aldagai hori ordezkatzen duen literal bat egon daiteke, edo ez. x1-en kasuan, esaterako, klausulan bere literalik baldin badago, balio positiboa, x1, edo negatiboa, ¬x1, izan dezake.

Behin CNF formatua, klausula, aldagaia eta literala zer diren jakinda, hauek nola eragiten duten azalduko dut. Esanda bezala, CNF formatu honetan dagoen formula bat hartzen du sarrera moduan SAT problemak. Jo dezagun, aipaturiko adibide hori pasatzen diogula SAT problemari.

Formula hau egiazkoa egin daitekeen edo ez adieraziko digu SAT problemak. Formula egiazkoa egiteko, bere klausula guztiek egiazkoak izan behar dute. Klausula egiazko egiteko, aldiz, bere literal batek gutxienez egiazkoa izan behar du. Literalaren balioa kalkulatzeko, bi gauza ikusi behar dira: batetik, literal horren aldagaiari asignatu diogun balioa, eta, bestetik, literala ezeztatuta dagoen edo ez. Ezeztatuta badago, aldagaiaren asignazio balioa alderantzikatuko luke.

Demagun, ¬x1 literala dugula eta x1 aldagaiari True balioa asignatuta daukagula. Aldagaia ezeztatuta dagoenez, literalaren balioa False edo negatiboa litzateke.

Goazen gure adibidera. Esan dugu, 4 klausulako formula bat dugula. Klausulaz klausula joango naiz adibidea azaltzen. Horretarako aldagaien asignazio posible bat behar dugu: x1=True, x2=False eta x3=True asignazioarekin probatuko dut, honek formula positibo izatea ahalbidetzen ote duen ikusteko.

1. $(x1 \vee \neg x2)$ klausula. Egiazko egiteko bi aukera ditugu: x1 aldagaiak True balioa izatea edo x2 aldagaiak False balioa izatea. x1=True eta x2=False direnez, bi literalak positibo izango lirateke, klausula positiboa, beraz.

2. $(x1 \vee x3)$ klausula. x1=True eta x3=True direnez, klausula hau ere positiboa izango da.

3. $(x2 \vee \neg x3)$ klausula. Kasu honetan, x2=False eta x3=True direnez, bi literalak faltsuak izango dira. Ondorioz, azken klausula begiratu beharrik gabe, esan dezakegu, asignazio posible honetarako formula negatiboa dela.

4. $(\neg x1 \vee x3)$ klausula. Esan moduan, berdin digu klausula hau positiboa edo negatiboa den, jada klausula negatibo bat dugulako, baina, azken kasua ere probatuko dugu jarrita gaudenez gero. x1=True denez, literala negatibo bihurtuko litzateke, beraz, klausula positibo izateko aukera bakarra, bigarren literala positiboa izatea litzateke. x3=True denez, eta ezeztatuta ez dagoenez, literal honek klausula egiazko egingo luke.

Esan bezala, formula honek emaitza negatiboa emango du asignazio konkretu honetarako. Asignazioa aldatu, eta, x1=True, x2=True eta x3=True balioak esleitzen baditugu, aldiz, klausula guztiak positibo izango direla ikus daiteke.

Honetan datza problema honen funtsak. CNF formatuan dagoen formula bat sarrera moduan hartuta, formula hau positibo egingo duen asignazio posiblerik badagoen bilatzea, alegia.

3.4 SAT solverra

Problema konputazionaletan emaitza bat lortzeko solverrak egoten dira. Solver hauek, problema bakoitzerako sortzen diren sistemak dira, emaitza bat lortzeko.

Problema bakoitzerako solver ezberdinak egon daitezke, batzuk eraginkorragoak izango direlarik. Kasu askotan, gainera, sarrera datuen arabera solver bat eraginkorragoa izan daiteke beste bat baino, eta, beste kasu batzuetarako okerragoa. Beraz, garrantzitsua izaten da solver ezberdinak probatzea, kasu bakoitzerako.

SAT problema hau ebazteko, hainbat solver daude. Hemendik aurrera, SAT solver terminoa aipatzen dudanean, SAT problema ebazteko solverren batetaz ariko naiz.

Kasu honetan, SAT solver bati, sarrera moduan, CNF formatuko formula bat pasa ezkero, emaitza moduan, aldagaien lista bat pasako liguke, aldagai bakoitza positiboan edo negatiboan egongo litzatekeelarik. Honetarako, aurretik azaldu moduan, asignazio posible guztiak probatzen joan beharko luke, harik eta, emaitza bat topatzen duen arte.

SAT solver batzuek, gainera, emaitza posible guztiak itzultzen dituzte. Noski, emaitza posiblerik ez badago, hori adieraziko duen emaitza itzuliko luke.

3.5 Subset Sum-etik eta Knapsack-etik SAT-erako laburketak

Sarrerako oinarri teorikoan aipatu moduan, edozein NP problema, NP-Osoa den batera laburtu daiteke. Hau da, edozein NP problema hartu, eta sarrera modu egokian eraldatuta, NP-Osoa den edozein problemari sarrera moduan sartzea posible da. LAipatutako 3 problemak NP-Osoak direnez, beraien artean laburketak egin daitezkeela esan nahi du horrek.

Proiektu honen helburu nagusia, Subset Sum eta Knapsack problemak ebaztea da, SAT solver bat erabilita. Hau gauzatzeko, bi problema hauek SAT-era laburtu behar dira, hau

da, Subset Sum eta Knapsack-en sarrerak eraldatu beharko dira, SAT problema moduan ebazteko.

SAT problemak, sarrera moduan CNF formatuan dagoen formula bat hartzen duenez, Subset Sum eta Knapsack problemen sarrak eraldatzea izango da helburua CNF formatu bat lortzeko. Sarreren eraldaketa honek zentzua zian behar du, noski, eta ondo planteatu behar da.

Trantsizio hau egitea nahikoa ez, eta, laburketa hau, ahalik eta modu eraginkorrenean egin ahal izatea nahi dugu. Orokortuz, modu eraginkor bat topatu nahi da, Subset Sum eta Knapsack problemak SAT solver baten bidez ebazteko.

Pysat liburutegia erabili nahi da horretarako, Pythonen SAT problemen inguruan modu eraginkor batean lan egiteko diseinatua dagoelako. Subset Sum problemaren laburketa azalduko dut lehendabizi.

3.5.1 Subset Sum-etik SAT-erako laburketa

Laburketa hau egiteko, lehendabizi ikusi behako dugu Subset Sum-en sarrera zein den. Aurretik azaldu moduan, l lista bat eta z zenbaki bat hartzen ditu sarrera moduan. Beraz, lista eta zenbaki hau eraldatu beharko dira, CNF formatuko formula bat sortuz.

Hau lortzeko, Pysat paketea erabiliko da. Bertan, ikusi beharko da, dauden modulu eta funtzioetatik zein erabili genezakeen gure helburua lortzeko. Badakigu, hainbat datu mota ezberdin sarrera moduan hartu eta CNF formatuko formula bat itzultzen duen funtzioak badaudela. Baina, ez da hain agerikoa kasu honetan funtzionatuko lukeen bat topatzea.

Lista bat eta zenbaki bat pasata, batura moduan zenbaki hau duen azpilista bat topatuko digun murrizketak sortzen dituen funtzio bat topatu beharko da, CNF formatuko formula bat itzultzen duena. Hau hobeto ulertzeko jar dezagun adibide sinple bat.

Demagun sarrera datu hauek ditugula Subset Sum-erako:

- l=(1,2,3)
- z=4

Pysat-eko funtzio bat topatu behar dugu, non, l eta z sarrera datuek emanda, honen murrizketak aplikatuko dituen CNF formula sortuko duen. Kasu hau oso sinplea da, aukera bakarra dago, 1 eta 3 zenbakiak hartzea da aukera bakarra. Behar dugun murrizketa hiru klausuletan sor daiteke. Azpian ikus daitekeen moduan, nahikoa izango dugu klausula bakoitzean, zenbakiak hartu beharra daukan balioa jartzea.

$$CNF = ((1), (-2), (3))$$

Kasu hau oso sinplea da, baina, hau egingo duen funtzio bat aurkitzea izango litzateke asmoa. Lista eta zenbaki batetik, murrizketak aplikatuko dizkion funtzio bat topatzea, non, emaitza moduan CNF formula bat itzuliko digun.

Behin hau edukita, SAT solver bat aplikatzea geratuko litzateke. Pysat paketean hainbat solver daude, eta, eraginkorrena zein den ikertu beharko genuke, hasteko.

Behin SAT solver egoki bat topatuta, emaitzen interpretazioa ikusi beharko genuke. Hau da, oraingoan, SAT solverrak, CNF formula hau egiazko izateko, 1 eta 3 aldagaiak positiboak eta 2 aldagaia negatiboa izan behar duela esango du. Kasu honetan, interpretazioa erraza litzateke, positiboak diren aldagai hauek izango lirateke Subset Sum problemako emaitza.

3.5.2 Knapsack-etik SAT-erako laburketa

Knapsack problema honetan ere, antzeko pausoak jarraitu beharko lirateke. CNF formatuko formula bat sortuko digun moduren bat topatu, Pysat-eko funtzioez lagunduta.

Hasierako ideia, ordea, Subset Sum-en ebazpena erabiltzea izango da, Knapsack ebazteko. Hau da, Subset Sum-en ebazpena eginda edukiko dugunez, eta bi problemak antza badutenez, Subset Sum-en ebazpena erabiliz, Knapsack ebazteko modu bat topatzea da lehenengo ideia.

Horretarako, batetik, pisuen lista eta motxilaren tamaina dauzkagu, beraz, lista bat eta zenbaki bat ditugu. Subset Sum-ek behar duen sarrera, hain zuzen. Knapsack-en azalpenean azaldu moduan, ordea, ez du zertan emaitzarik hoberenak motxilan gehien okupatzen duen emaitza posiblea izan behar. Gerta daiteke, motxila bete gabe utzi, eta balio handiagoa lortzea. Beraz, ikusi beharko da modu honetan soluziorik lortzeko modurik badagoen.

Bestetik, balioen lista daukagu, eta, Subset Sum ebazpenari, sarrera moduan balioen lista hau sartzeaz aparte, zenbaki moduan balio ezberdinak sartzea posible izango litzateke, emaitza posiblerik baduen ikusteko.

Hau da, Knapsack ebazteko, Subset Sum exekutatu genezake iteratiboki, aldi bakoitzean balioen lista eta balio ezberdinak sartuz. Balio hau, zerrendako balio guztien baturatik hasiko litzateke (aukerarik onena izango delako), eta, iterazio bakoitzean banaka jaisten hasi, emaitza bat topatzen duen arte. Iterazio bakoitzean Subset Sum aplikatu beharrak, ordea, eraginkortasunari nola eragingo dion aztertu beharko da.

Beste modu bat, Pysat-eko funtzio bat topatzea litzateke, Subset Sum-ean egiten dugun antzeko zerbait egiteko. Kasu honetan, ordea, konplexuagoa izango da funtzio bidez emaitza bat topatzea. Izan ere, Subset Sum erabakitze problema bat da, emaitza bai edo ez izango da, eta balio batzuk itzuliko ditu.

Knapsack ordea, optimizazio problema bat da, beti emango du emaitza bat, baina, emaitza posible bat edukitzeak ez du esan nahi emaitza onena denik. Horretarako, kalkulu gehiago egin beharko dira eta honek errendimenduan eragina izango du.

KAPITULUA 4

ERABILITAKO TRESNAK

Atal honetan proiektuaren garapenean erabili diren tresnak aipatuko dira. Tresna bakoitza zergatik aukeratu den eta zertarako erabili den azalduz.

4.1 Python

Proiektuko laburketak programatzeko Python programazio lenguaia erabili da. Lenguaia hau erabiltzearen arrazoi nagusia Pysat liburutegia erabili nahi izan dela da. Egin diren bi laburketak Pysat liburutegiko funtzioak erabiliz egin dira, beraz, Python erabiltzea ezinbestekoa izan da.

Pysat Pythoneko liburutegi bat da [6], zeinak interfaze sinple eta bateratu bat eskaintzea duen helburu Boolear satisfiability (SAT) ebazteko. Pysat-en helburua da SAT eta bere aplikazioetan lan egiten duten ikertzaileek Python-en SAT prototipoak erraz ebazteko aukera ematea, SAT soluzio modernoen maila baxuko jatorrizko inplementazioen boterea pixkanaka ustiatuz.

Pysat-en funtzio bidez sortu dira CNF formulak eta hauek ebazteko modu onena zein den probatzeko erabili diren solver ezberdinak ere Pysat-ek dauzkan solverrak izan dira.

Pysat-ez aparte honako liburutegiak ere erabili dira:

i. OS liburutegia. Fitxategiekin lan egiteko erabiltzen da, irakurri eta idazteko, besteak beste.

ii. TIME liburutegia erabili da denbora neurketak egiteko.

iii. Multiprocessing liburutegia erabili da behar izan diren paralelizazioak egiteko.

4.2 Ortools

Googlek konbinatoria optimizaziorako garatutako liburutegi bat da Ortools. Hainbat solver ditu beregan, problema ezberdinetarako. C++, Python, C# edo Java programazio lenguaietan sortutako problemak ebazteko sortu zen, eta hainbat sari irabazi ditu 2013az geroztik [7].

4.3 Flask

Flask-ek Python erabilita web aplikazioak sortzea ahalbidetzen du. Sinplea da eta web aplikazioak sortzea errazteko dago garatua [8]. Ezagunak diren hainbat aplikazio flask esparru honekin eginak dira, hala nola, Netflix edo Pinterest web aplikazio ezagunak.

Flask liburutegia erabili da web aplikazioa egiteko. Python lenguaia erabiltzeaz gain, ematen dituen erraztasunengatik web aplikazioa sortzeko aukerarik egokiena dela hausnartu da.

4.4 HTML, CSS eta Javascript

HTML (HyperText Markup Language) webaren osagairik oinarrizkoena da [9]. Web edukien esanahia eta egitura zehazten ditu. HTML-az gain, beste teknologia batzuk erabiltzen dira orokorrean web orri baten itxura/aurkezpena (CSS) edo funtzionalitate/portaera (Javascript) deskribatzeko.

CSS (ingelesez Cascading Style Sheets) web baten estiloa aldatzeko erabiltzen den tresna da. CSS lenguaiaren helburu nagusia dokumentu baten edukia eta aurkezpena bereiztea da. HTML-ko elementuak orrian nola ikusiko diren zehazten du.

Javascript (edo JS) goi-mailako programazio-lenguaia bat da [10]. Web-orri interaktiboak sortzeko aproposa da eta funtsezkoa web-aplikazioetan. Script-lenguaia bat da, atazen exekuzioa ingurune berezietan automatizatzeko diseinatua.

HTML, CSS eta Javascript erabili dira web orriaren diseinua eta oinarrizko funtzionalitateak sortzeko, oso erabiliak direlako eta aukera asko ematen dutelako.

4.5 D3.js

D3.js datuetan oinarritutako dokumentuak manipulatzeko Javascript liburutegia da. D3-k HTML, SVG eta CSS erabiliz datuak bistaratzen laguntzen dizu [11].

Javascript-eko liburutegi honek datuetatik irudiak sortzeko ematen dituen aukerengatik erabili da D3.js. Subset Sum eta Knapsack problemetan erabili diren pisuen irudia bistaratu nahi izan da. Sortzen den irudian pisu bakoitzak bere proportzioko zatia okupatzen duelarik.

4.6 Ubuntu sistema-eragilea

Ubuntu idazmahairako Linux sistema eragile oso bat da, Debian banaketan oinarritutakoa. Banaketa honen ezaugarrietako bat erabilerraztasuna da ("for human beings"du leloa): aplikazio erabilerrazak lehenesten ditu. Bestetik mahaigaineko ingurunetan Unity du.

Lehen erabakia sistema-eragilea aukeratzea izan da. Windows eta Ubunturen artean, Ubuntu izan da aukera ondoren aipatu diren tresnak erabiltzerako orduan erraztasun handiagoa ematen duelako.

4.7 Spyder

Spyder Python-en eta Pythonentzat idatzitako kode irekiko eta doako ingurune zientifiko bat da, zientzialariek, ingeniariek eta datu-analistak diseinatua [12]. Garapen tresna integral baten edizio, analisi, arazketa eta profilaren funtzionalitate aurreratuen konbinazio berezia eskaintzen du, datuen esplorazio, exekuzio interaktiboa, ikuskapen sakona eta pakete zientifiko baten bistaratzeko gaitasunarekin.

Spyder ingurunea erabili da, batetik, Python-en lan egiteko egokia delako eta, bestetik, KEA irakasgaian hau erabili delako eta proiektu honek zerikusi zuzena duelako irakasgaian landutakoarekin.

KAPITULUA 5

PROIEKTUAREN GARAPENA

Proiektuaren garapenak fase ezberdinak izan ditu. Lehendabizi, Subset Sum-etik SAT-erako laburketan sortu diren bertsio ezberdinak eta hauen zergatia azalduko dut. Ondoren, berdina egingo dut Knapsack-etik SAT-erako laburketarekin.

Honez gain, bi problema hauek ebazteko erabili dudan zuzeneko solver batekin egindako bertsioak azalduko ditut. Jarraian, web aplikazioaren garapena nola joan den erakutsiko dut, eta, amaitzeko, web orrian sortu ditudan funtzionalitateak bistaratuko ditut.

5.1 Subset Sum-etik SAT-erako laburketa

Proiektuaren garapeneko lehen helburua Subset Sum problematik SAT problemarako laburketa egitea izan da. Saiakera bat baino gehiago egin dira liburutegi eta algoritmo ezberdinak erabilita.

5.1.1 Bitarrera pasatzeko sistema

Lehen aukera zenbaki guztiak bitarrera pasa eta sistema baten bidez SAT-ek behar dituen CNF formatuak dagoen formula bat sortzea izan da. Listako zenbaki bakoitzari aldagai bat esleitzeaz aparte, sistema honek, zenbakiak, bai listakoak, eta bai helburu zenbakia bitarrera pasatzen ditu eta bit bakoitzari aldagai bat esleitzen dio.

Aldagai hauekin batuketa eta eragiketa ezberdin batzuk egitearen bidez beste aldagai batzuk sortzen ditugu, emaitza egokia lortzeko komeni zaizkigunak. Horrela, CNF formatuan dagoen formula bat sortzen dugu eta Pysat liburutegiko solver bat aplikatzen diogu sortu dugun CNF formulari.

Erantzuna aldagai guztien lista bat izango da, balio positiboz eta negatiboz betea. Hau interpretatzeko listako zenbaki bakoitzari esleitu diegun aldagaiei begiratu behar diegu. Hauetatik positiboak direnak aukeratu eta ikusi dezakegu emaitza egokia ematen duela.

Nahiz eta emaitza egokia eman, errendimendu aldetik ez da oso egokia izan. Izan ere, bit bakoitzarentzat eta hauen arteko eragiketa bakoitzarentzat aldagai bana sortzeak klausula kopurua oso handia izatea dakar. Nahiz eta, CNF klausulak sortzeko prozesua nahiko azkar egin, denbora gehien hartzen duen atala solverrarena izan da.

CNF formula solver ezberdinekin ebaztea probatu da, baina, azkarrenak ere denbora gehiegi ematen zuen exekuzioan (profiler baten bidez ikusi ahal izan da hau). Hainbat aldaketa egiten saiatu arren, ez da lortu hobekuntza nabarmenik, beraz, klausula gutxiago sortzen dituen beste modu bat bilatzen saiatu gara.

5.1.2 Pysat.pb modulua

PBEnc modulua (Pysat.pb) Pysat liburutegiaren parte da [13], eta, gure kasurako interesgarria izan zitekeela ikusi genuen, listetatik CNF formula sortzeko aukerak ematen baititu. Guri interesatzen zaiguna equals funtzioa izan da.

```
PBEnc.equals(lits, weights, bound, encoding)
```

Equals funtzio honek sarrera moduan ondorengo balioak hartzen ditu:

1. lits: Zenbaki zerrenda bat, emaitza bezala itzultzean zerrenda honetako zenbakiak itzuliko ditu. Nire kasuan lista bat pasatzen diot 1etik listaren tamaina artekoa.
2. weights: Lits listako balio bakoitzari pisu bat esleitzen dio. Hemen pasatzen diot nire Subset Sum-eko lista.
3. bound: pisuen limitea izango litzateke. Nire kasuan helburu zenbakia pasatzen diot.
4. encoding: Aukeratu den kodeketa jarri behar da. Nire kasurako 'adder' kodeketa jarri diot pisuen batura egin dezan.

Behin balio hauekin funtzioa exekutatuta, funtzioak ondorengoa egingo du: weight listako elementu multzo bat sortuko du, non, hauen batura bound-en berdina izango den. Hau lortzerik badu, pisu horiei dagozkien lits aldagaiekin sortutako CNF formatuko formula itzuliko du.

Behin CNF formatuko formula edukita, jada laburketa osatuta dago eta SAT solver bat aplika diezaiokegu gure formulari. Horrela, Pysat-eko SAT solver bat aplikatzen diot formula honi eta honek, lortu nahi dudan emaitzaren Subset Sum-eko listako indizeak itzuliko dizkit. Beraz, indize hauei dagozkien pisuak zerrendatu eta Subset Sum-en emaitza lortuta edukiko genuke.

Modu honetan, askoz ere errendimendu hobea lortu dezakegu aurreko bertsioarekin alderatuta. Sortzen dugun CNF formulako klausula kopurua askoz ere txikiagoa denez, SAT solverrak askoz exekuzio denbora gutxiago hartzen baitu.

5.2 Knapsack-etik SAT-erako laburketa

Knapsack problematik SAT-erako laburketaren prozesua ere antzekoa izan zen. Izan ere, Knapsack-etik SAT-erako laburketa egiteko hasierako ideia Subset Sum erabiltzea zen.

5.2.1 Subset Sum-en ebazpena erabiliz bilaketa dikotomikoa egitea

Subset Sum-eko lehen bertsioa eginda edukita (bit-en prozesaketarena), Knapsack-erako geneukan ideia inplementazen hasi nintzen. Asmoa zen, bilaketa dikotomiko bat egitea, non, Subset Sum aplikatuko zen aldi bakoitzean.

Knapsack-ean motxilan sartu daitekeen balio kopurua maximizatzea da helburua. Horretarako, balio maximoa hartzen genuen bilketa dikotomikoaren hasieratzat, hau da, balio guztien batura. Beraz, balio maximo eta minimoaren tarteko balioa, eta, balioen lista pasatzen nizkion Subset Sum-en inplementazioari. Honek itzultzen zuena azpilista posible guztien lista bat zen.

Azpilista bakoitzarekin ondorengoa begiratu beharra zegoen, azpilista horiei dagozkien pisuen baturek, motxilaren tamaina ez zutela pasatzen. Azpilistaren bat bazegoen motxilan sartzen zena, momentuko emaitza bageneukan jada. Bilaketa dikotomiko bitartez hortik aurrerako aukerak begiratzea geratzen zen, ordea.

Laister konturatu ginen, ordea, hau ez zela benetan eraginkorra, izan ere, bilaketa dikotomikoak emaitza posiblerik ez itzultzeak, ez zuen zertan esan nahi hori baino hobeagorik ez zegoenik. Gerta daitekeena da lista batek baturarik ez edukitzea zenbaki baterako, baina, bai zenbaki handiago baterako.

Esandakoarengatik, proba kasu asko egin beharra zeuzkan algoritmoak eta, beraz, errendimendu arazo garbia zegoela iritzita, beste modu bat probatu genuen.

5.2.2 Subset Sum-erako egindako bertsio bat aldatu Knapsack sortu nahian

Knapsack-etik SAT erako laburketa sortzeko hurrengo estrategia, Subset Sum-en bitarrera pasatzeko bertsioari aldaketa bat egitea izan zen, Pysat.card moduluaren erabilpena gehituz.

Pysat.card Pysat-eko modulu bat da eta, erabili nahi genuen funtzioa atmost zen. Funtzio honek, lista batetik lortu nahi ditugun gehienezko aldagai positibo kopurua zehazen du, eta CNF formula sortzen du informazio horrekin.

Asmoa zen bitarrera pasatzerako prozesu amaieran klausula hauek gehitzea eta moduren bat bilatzea zuzenean motxilaren pisutik pasako ez ziren konbinaketak soilik itzul zitzan. Hainbat proba egin nitue, baina, ez zen lortu emaitza egokirik.

5.2.3 Knapsack Pysat.pb bidez

Momentu honetan aurkitu genuen Pysat.pb modulua eta lehen Subset Sum-erako egin moduan Knapsack-erako ere oso erabilgarria izan zitekeela ikusi genuen.

Kasu honetan, Subset Sum-ean egindakoa baino konplexuagoa da laburketa prozesua. Oraingoan, PBEnc.atleast eta PBEnc.atmost funtzioak erabiltzeaz gain, bilaketa dikotomiko bat egin behar izan da emaitza lortzeko.

Knapsack-eko laburketaren lehen saiakeran aipatu den bilaketa dikotomikoan eraginkortasun eza nabarmena zen. Bilaketa honetan emaitza posible bat topatzeak, hori baino emaitza hobeagoak begiratzera eramaten gintuen, baina, bilaketan emaitzarik ez topatzeak ez zigun ziurtatzen hori baino hobeagorik egongo ez zenik, Subset Sum-ek balio konkretuarekin posible den soilik begiratzen baitu.

Nahiz eta eraginkortasun falta hori eduki, ideia hartzeko balio izan zigun eta PBEnc modulu hau topatzean, bilaketa dikotomikoaren ideiarekin lotu genuen segituan. Ondoren azalduko dut laburketa prozesu hau.

Esan moduan PBEnc.atmost eta PBEnc.atleast funtzioak erabiliko ditugu. Funtzio hauek, PBEnc.equals-en antzera funtzionatzen dute. Hau da, lista bat eta zenbaki bat emanda CNF

formula bat itzuliko dute, non, gehienez edo gutxienez (funtzioaren arabera), pasatako balioa ez pasatzeko CNF formula itzuliko duten.

Gauzak horrela, lehendabizi pisuen lista eta motxilaren tamainari lotutako Klausulak sortzen ditut. Horretarako, PBEnc.atmost funtzioa erabiltzen dut.

```
PBEnc.atmost(lits, weights, bound, encoding)
```

Pisuen lista eta motxilaren tamaina sartuta, CNF formula bat lortzen dut, motxilaren pisutik ez pasatzeko klausulak ematen dizkidana. CNF formula honi Pysat eko SAT solver bat pasa eta sortu diren aldagai laguntzaileak kendu ondoren, motxilaren pisu kapazitatez ez pasatzeko konbinaketa posible guztiak edukiko genituzke.

Pisu maximotik ez pasatzeko konbinaketa posible guztietatik balio handiena ematen duena lortu behar dugu. Horretarako, balioen lista hartu eta balio guztien batura lortzen dut lehendabizi. Batura hau izango litzateke lortu ahalko genukeen balio maximoa. Bilaketa dikotomiko baten bidez balio maximo honen eta minimoaren tarteko balioa lortu, eta gutxienez balio hori izango duen konbinaketa posiblerik badagoen begiratzen dut.

Hau egiteko PBEnc.atleast funtzioa erabiltzen dut:

```
PBEnc.atleast(lits, weights, bound, encoding)
```

Funtzio honi balioen lista eta momentuko balioa sartuta, gutxienez balio hau emango duen, balioen listako elementuez osaturiko CNF formula lortzen dut. CNF formula honi Pysat eko SAT solver bat pasatzen diot emaitza posible guztiak lortzeko. Ikusi moduan prozesu zati hau pisuekin egindakoaren oso antzekoa da.

Puntu honetan, bi lista dauzkagu lortuta, pisuen murrizketen emaitza eta balioen murrizketen emaitza. Lista hauetako bakoitza azpilistaz osatuta dago, non, azpilista bakoitza 1etik listaren tamaina arteko zenbakirartekoa den. Kontua da, zenbaki bakoitza positiboa edo negatiboa izango dela, motxilan sartu edo ez sartzearen arabera.

Azpilista hauetako zenbakiak indize modukoak direnez, alderatu daitezke pisuen listakoak balioen listakoekin. Hau da, pisuekin lortu diren azpilista bakoitzak, motxilan sartu daitezkeen elementuen konbinaketa posible guztien indizeak adierazten ditu. Bestalde, balioekin lortu den emaitzan, azpilista bakoitzak, balio hori edo handiagoa lortzeko konbinaketa posible guztien indizeak adierazten ditu.

Puntu honetan, egin beharreko bakarra azpilistak konparatzea izan da. Bi emaitzetan azpilista berdinik badagoen begiratzea, alegia. Bi emaitzetan konbinaketa berdina egoteak zera esan nahiko luke, badagoela gutxienez aukera bat, non, motxilaren tamaina pasa gabe, adierazitako balioaren berdina edo handiagoa lortu daitekeen.

Gauzak horrela, erantzuna baiezkoa balitz, bilaketa dikotomikoa balio horretatik maximora arte egingo litzateke hurrengo aldian. Erantzuna ezekoa izanez gero, aldiz, balio minimotik balio horretara artekoa izango litzateke hurrengo bilaketa. Prozesu hau errepikatuko genuke, kasu nabarian jarraian dauden bi balio posiblek batak, emaitza posible dela, eta, hurrengoak balio horrekin emaitza ez dela posible esaten duen arte. Balio maximoa aurkitu dugula esan nahiko baitu horrek.

Azken aldaera bat ere egin diot algoritmo honi, paralelizazioa aplikatzea, hain zuzen. Bilaketa dikotomikoa egiteko garaian, balio minimoa eta maximoaren artean bilaketa bat egin beharrean, hainbat bilaketa batera egiten ditut. Lehendabizi ordenagailuak dituen cpu

kopurua lortzen dut eta bilaketa dikotomikoan balio minimo eta maximoaren arteko tartea kopuru horrekin zatitzen dut.

Nire kasuan ordenagailuak 12 cpu dauzkanez, balio minimo eta maximoaren arteko 12 bilaketa batera egiten ditu. Horrela, momentuko balio minimo eta maximo berria askoz zehatzago kalkulatzen dira eta hobekuntza nabarmena eragiten da kasu proba handietan.

5.3 Ortools liburutegiaren garapena

Subset Sum problemaren azken ebazpena nahiko eraginkorra bada ere, Knapsack problemaren ebazpenean ez da gauza bera gertatu. Aurrerago, lortutako emaitzen atalean zerrendatzen ditut, bai, Subset Sum, bai, Knapsack problemetarako egindako proba kasu ezberdinak.

Planifikazioan, arriskuen atalean agertu moduan, Knapsack-en ebazpena ez da nahikoa eraginkorra izatea lortu. Eta, beraz, honi soluzioa jarri nahi izan zaio. Arrisku konkretu horren soluzioan jartzen duen moduan, zuzeneko ebazpen bat bilatzen saiatu naiz.

5.3.1 Knapsack problemaren ebazpena Ortools liburutegiaren bidez

Gure helburua, bi problema hauek modu eraginkorrean ebatziko dituen web aplikazio bat sortzea denez, Knapsack problema ebazteko zuzeneko ebazpen bat bilatzen saiatu naiz. Gauzak horrela, Ortools liburutegia topatu dut, zuzenean Knapsack problema ebazteko funtzio bat duena. Ortools, Googlek sorturiko liburutegi bat da, eta konbinatoriako optimizazio problemak ebazteko garatu zen.

Liburutegi honek, badu solver bat, problema ezberdinetarako balio duena, eta, tartean, Knapsack problemarako solverra dauka. Hau da solver honen funtzionamendua:

Lehendabizi solverra definitu behar da:

```
from ortools.algorithms import pywrapknapsack_solver

solver = pywrapknapsack_solver.KnapsackSolver(
    pywrapknapsack_solver.KnapsackSolver.
    KNAPSACK_MULTIDIMENSION_BRANCH_AND_BOUND_SOLVER,
    'Knapsack')
```

Behin, solverra Knapsack problemarako definitu dugularik, balioak esleitu, eta, solverra bera exekutatzea geratzen zaigu:

```
solver.Init(values, weights, capacity)
values = solver.Solve()
```

Beraz, solver honi, balioen lista, pisuen lista eta motxilaren kapazitatea sartu, eta, Knapsack problemak emaitza moduan itzultzen dituen elementuen lista itzuliko liguke.

Gero, 6.2 taulan ikusi ahal izango dira eraginkortasun probak, baina, kontua da, zuzeneko ebazpen bat denez, eta ez laburketa bidezkoa, oso eraginkorra dela. Hau ikusita, Subset Sum problemarako ere antzeko zerbait egitea pentsatu da.

5.3.2 Subset Sum problemaren ebazpena Ortools liburutegiaren bidez

Hasiera batean, web aplikazioan nik sortutako Subset Sum bertsioa jartzea pentsatuta baneukan ere, Knapsack-erako zuen eraginkortasun diferentzia ikusita, Ortools-ek, Subset Sum ebazteko funtziorik bazeukan begira hasi nintzen.

Denbora batez bila ibili eta gero, konturatu nintzen ez zeukala inongo funtziorik hau zuzenean egiteko, baina, Ortools-en Knapsack solver hau erabili nezakeela konturatu nintzen.

Horretarako, egin beharrekoa, Knapsack solverrari, bai pisuen lista, eta, bai balioen lista, berdinak pasatzea da, Subset Sum-ek hartzen duen lista bera, alegia. Modu honetan, pisu bakoitzak, balio bera izango du, eta emaitza moduan balioen lista optimoa itzuliko liguke solverrak.

Kontuan hartu behar da, ordea, Knapsack solverrak itzultzen duen emaitza, balitekeela motxilaren tamaina baino txikiagoa izatea. Adibide bat jarriko dut hau ikusteko.

Hona hemen, adibide moduan hartutako sarrera datuak, Subset Sum problemarentzat:

$$lista = (6, 3, 9, 5)$$

$$z = 21$$

Knapsack solver bidez exekutatu ahal izateko, honako sarrera sartu beharko genioke Knapsack solverrari:

$$balioLista = (6, 3, 9, 5)$$

$$pisuLista = (6, 3, 9, 5)$$

$$z = 21$$

Honen emaitza (6,9,5) azpilista litzateke. Aurretik esan moduan, begiratu behar dena azpilista honen emaitzak z=21 balioa duen da. 6+9+5=20 dela ikusten dugunez, kasu honetan, Subset Sum problemak emaitza posiblerik ez dagoela itzuli beharko luke.

Beraz, hau programatu ondoren, jada Subset Sum problemaren ebazpen eraginkorrago bat edukiko genuke. Aurrerago garatu den 6.1 taulan ageri dira denbora eraginkortasun aldeak.

5.4 Web Aplikazioaren garapena

Proiektuko bigarren helburu nagusia Subset Sum eta Knapsack problemak modu eraginkor batean ebazten dituen web aplikazio bat sortzea izan da. Web aplikazio honek, problema bakoitzaren funtzioak eduki beharko ditu ebazpena egin ahal izateko, eta, hemen etorri da lehen erabakia hartu beharra, nik sortutako funtzioak erabili edo Ortools erabiliz sorturikoak erabili.

Nik sortu ditudan algoritmoak ahalik eta eraginkorren izan daitezen sortu ditut, baina, kontua da, nire algoritmoak laburketa sistema erabili behar zuela, hori zelako proiektuaren gako garrantzitsu bat. SAT problemarako laburketa sistema honek denbora gehien pasatzen

duen puntua SAT solverra aplikatzean da, eta denbora konparaketak egin eta gero Ortools-en bertsioak erabiltzea erabaki da.

Hau erabakitzeko arrazoia, denbora eraginkortasuna izan bada ere, ez da arrazoi bakarra izan. Proiektuko beste helburu bat, bi problemek emandako irteera datuekin, irudi egoki bat sortzea zen, emaitzak interpretatzen lagunduko zuena.

Ortools erabilita, proba kasu handiagoak exekuta daitezkeenez, irteera datuak ere handiagoak izango dira. Horrela, emaitza zenbat eta handiagoa izan, orduan eta zailagoa izango da irudi egokia sortzea. Beraz, erronka hau hartu nahi izan da irudiaren egokitasuna probatzeko.

5.4.1 Web aplikazioa sortzen

Web aplikazioa sortzeko HTML, Javascript eta CSS-z aparte Flask erabiltzea erabaki da. Pythonen egindako funtzioekin ari garenez, web aplikazio bat sortzeko modu onena Flask erabiltzea zela erabaki genuen.

Lehendabizi Subset Sum problemarako atala garatu nuen, erabiltzaileak zenbakiak sartu eta emaitza ematen zuena. Emaitza moduan sarrera eta irteera datuak ematen ditu eta hauek .txt fitxategi batean deskargatzeko aukera ere ematen du. Honez gain, emaitzari interpretazio egoki bat emateko eta ikus errazagoa izateko irudi bat sortzea erabaki zen.

5.4.2 Emaitzen interpretaziorako irudia sortzea

Emaitzako listarekin SVG irudi bat sortzen da. Irudia lauki handi bat da, non, lauki txikiz osatuta dagoen. Lauki txiki bakoitza, emaitzako elementu bakoitzari dagokion zatiaren tamainakoa da. Hau da, emaitza moduan [2,5,3] emango balu, laukian hiru laukitxo egongo lirateke, bata 2 zenbakiduna %20 okupatuko lukeena, bestea 3 zenbakiduna %30 okupatuko lukeena eta bestea 5 zenbakiduna %50 okupatuz (ikus 5.3 irudia).

Irudia sortzeko, Javascript-eko D3 liburutegia erabili da. D3 liburutegi hau, datuekin irudiak sortzeko garatuta dago, eta hainbat aukera ematen ditu SVG irudien sorkuntzan.

Kasu txikietan irudirik gabe ere erraz interpreta daiteke emaitza, baina, emaitza ehunka edo milaka elementuko lista bat denean asko laguntzen du irudia ikusteak. Batez ere, Knapsack probleman, motxilan sartzen diren elementuak lauki horren barruan bakoitzak okupatzen duen proportzioa ikusteak errealistagoa egiten du emaitza.

Irudi egokia sortzeak ere bere lana eman du, ordea. Nahi nuen laukia sortzea lortu nuen hasieran, proportzioak mantentzen zituena zenbakiaren arabera. Baina, aipatu moduan, kasu handienetan ehunka edo milaka elementu egon daitezke emaitzaren listan. Kasu hauetan sortzen zuen irudia ez zen batere egokia, izan ere ezinezkoa baitzen laukian ezer ikustea.

Bi arrazoi nagusi zeuden arazo honen oinarrian. Batetik, agerikoa den moduan, irudi txiki askorekin, ez zen ezer garbirik ikusten. Bestetik, aldiz, iruditxo bakoitzaren gainean, bere pisua idatzi nahi zen. Hau ere, emaitza txikien kasuan, ez zen arazo bat, baina, datu askoren irudia osatzerakoan, laukitxo bakoitzaren zenbaki tamaina ez zen laukitxoaren beraren tamainaren proportzionala.

Ondorioz, laukitxotik ateratzen zen, eta, ondokoan sartzen zen. Honek, ondoko zenbakia tapatzea eragiten zuen. hasierako irudia, beraz, ez zen egokia emaitza handien kasurako.

Zerbaitengatik aukeratu zen ordea irudiaren formatua SVG izatea. JPEG, PNG eta antzeko formatuek pixeletan gordetzen dute informazioa, baina SVG-k ez. SVG formatuko irudiak, artxibo bektorial modukoak dira eta formula matematiko batzuen bidez puntuak eta lerroak gordetzen dituzte.

Lehenik, laukitxo bakoitzaren zenbakia, laukitxo horren tamainakoa izatea garatu da. Horretarako, laukitxoaren altuera eta zabalera neurtzen dira, eta hauen arteko minimoa aukeratu. Bien arteko minimo hau jartzen da letra tamaina bezala, kasu bakoitzean.

Behin arazo bat konponduta, emaitza handietarako irudiak ondo ikusi ahal izatea garatu nahi izan da. Gure irudia SVG irudi bat da, eta, honek abantaila nabarmena ematen digu, irudia handitzea lortuko bagenu ez dela bere kalitatea galtzen. Hau ikusita, Zoom egin ahal izatea ahalbidetu nahi izan zen.

Bere lana emanagatik, lortu dut funtzionalitate hau ere aplikatzea. D3 liburutegiak, aukera ematen du, svgPanZoom funtzioaren bidez SVG irudi bati Zoom-a aplikatzeko. Beraz, saguarekin 'scroll' egiten dudan bakoitzean, Zoom-a aplikatzen dio irudiari. Horrela, irudia handitu edo txikitu daiteke, laukirik txikiena ere ikusgarri eginez. Honez gain, sortutako irudia deskargatzeko aukera ere jarri da.

5.4.3 Datuak fitxategitik irakurtzea

Lehen esan moduan, web aplikazioaren helburu nagusi bat sarrera handietarako eraginkorra izatea zen. Horretarako, datuak eskuz sartzeak zentzu gehiegirik ez zuela konturatu ginen. Sarrera datuak oso handiak izango badira, banan-banan idazteak baino, fitxategi batetik zuzenean irakurtzeko aukera ematea egokiagoa zela ikusi genuen.

Beraz, bi aukerak garatu dira. Zuzenean eskuz idatzi daiteke sarrera, edo .pkl fitxategi bat (edo bi knapsack-en kasuan) kargatu listentzat, eta .txt fitxategi bat nahi den zenbakiarekin. Modu honetan, asko errazten da sarrera handiak sartzeko prozesua.

Errendimenduan hobekuntza gehiago sartzeko hainbat fitxategi batera exekutatzeko aukera ematea pentsatu genuen. Horretarako, Pythonen daukadan Ortools-eko funtzioa paralelizatu egin nuen hainbat prozesu batera egiteko. Behin paralelizazio prozesu hau eta fitxategitik datuak irakurtzeko sistema prest edukita, web aplikazioan funtzionalitate hau betetzeko atala gehitu nuen.

Erabiltzaileak .zip fitxategi bat igo dezake, barruan hainbat fitxategirekin. Subset Sum-en kasuan fitxategiek izen bera izan behar dute, bata .pkl eta bestea .txt formatuarekin. Knapsack-en kasuan aldatzen dena da balioen listak "_bal.pkl" amaiera izan behar duela eta pisuenak "_pis.pkl" amaiera.

Horrela, behin exekuzio prozesua eginda dagoenean, erabiltzaileak lista batetik aukeratu ahal izango du zein sarreraren emaitza ikusi nahi duen. Emaitzak deskargatzeko momentuko aukeraren fitxategia deskarga daiteke, edo, fitxategi bakar bat deskargatzeko aukera ere badago emaitza guztiak barne dituena.

5.4.4 Azken xehetasunak

Azkenik diseinu aldetik CSS egokiago bat aplikatu da, web aplikazioa erabilerrazagoa egiteko. Gainera, kasu handienetan exekutatzen denbora behar duenez "loading" moduko irudi bat gehitu da exekuzioan ari dela argitzeko.

5.5 Web aplikazioan sortu diren funtzionalitateak

Proiektu honen helburuetako bat, Subset Sum eta Knapsack problemak ebatziko dituen web aplikazioa sortzea izan da. Horrez gain, emaitzak irudikatzen lagunduko duen irudi egoki bat sortzea izan da asmoa.

Bi problemak ebazteko aukerak eta funtzionalitateak antzekoak direnez, bakarraren funtzionamendua azalduko da, Knapsack-ena kasu honetan.

Lehenik eta behin, web aplikazioa martxan jartzean, hasiera menua ikusiko dugu (ikus 5.1 irudia).

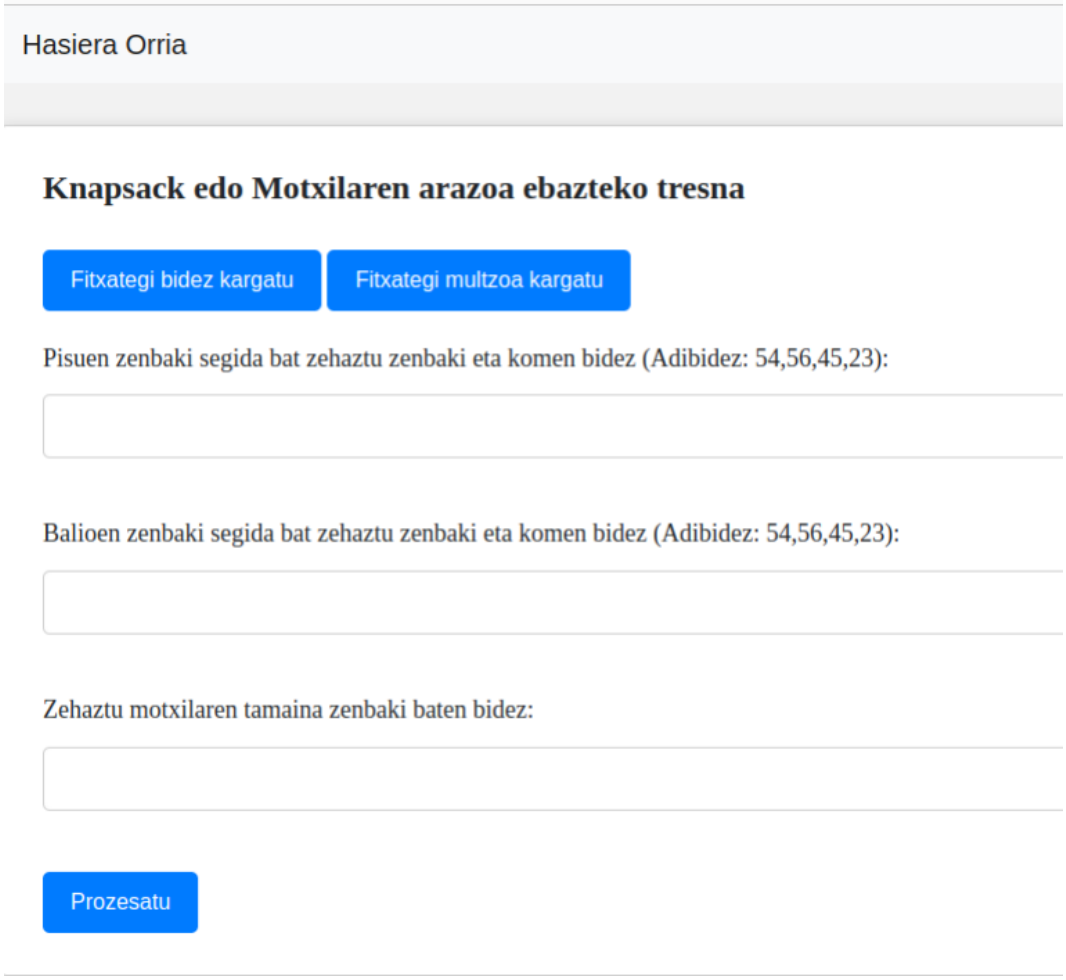
5.1 Irudia: Web aplikazioaren hasiera orria

Bertan, ebatzi nahi dugun problema hautatu beharko dugu. Knapsack problema azalduko dugunez, atal honi dagokion botoia sakatuko dugu.

Behin aukeratutako problemaren orrian gaudenean, zuzenean sarrera bat sartu dezakegu (ikus 5.2 irudia). Horretarako, lehenik pisuen lista bat sartu beharko genuke, balioen lista batez jarraituta joango litzatekeelarik. Bi lista hauek luzera berekoak izan beharko dute, noski. Pisu bakoitzak balio bat izan behar baitu esleituta. Hau horrela ez bada, abisu bat erakutsiko luke. Amaitzeko, motxilaren tamaina sartu beharko dugu.

Behin eremu guztiak definituta, Prozesatu botoia sakatu eta emaitza orrira joango litzateke web aplikazioa. geroago ikusiko dugu emaitza orri hau, izan ere, fitxategi batetik datuek kargatuta ere emaitza orri berdinera joaten baita.

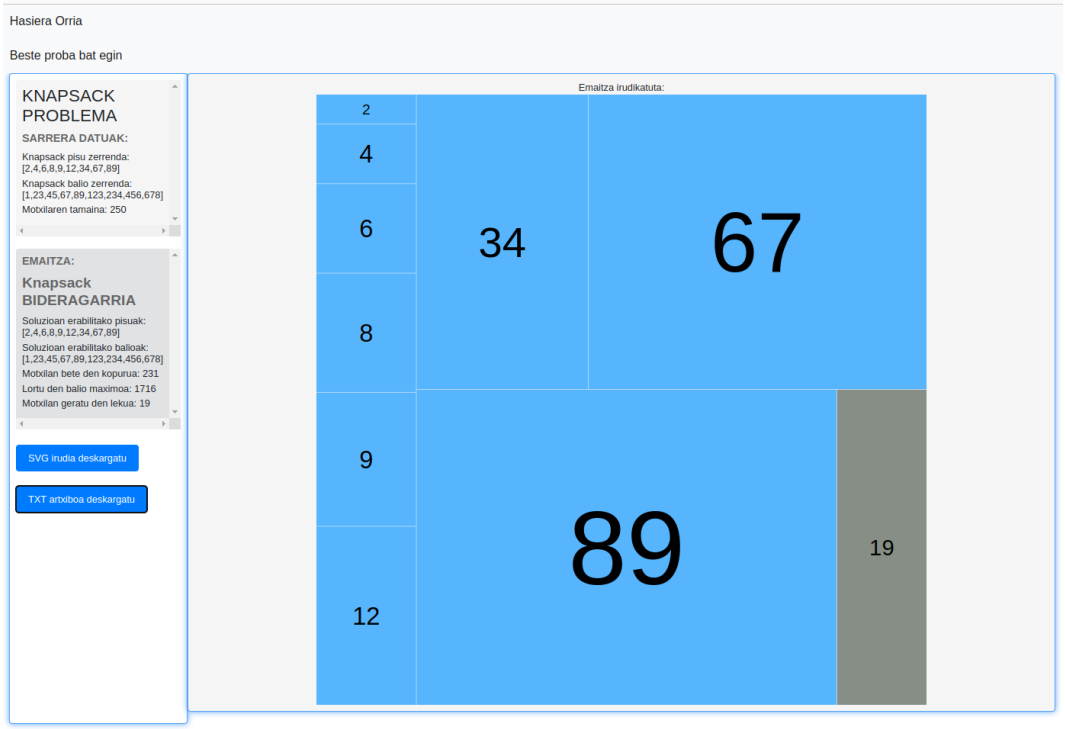
Beraz, datuak idatziz sartu beharrean, Fitxategi bidez kargatu sakatuko bagenu, horretarako prestatuta dagoen orrira joango litzateke.



5.2 Irudia: Knapsack problema ebazteko atala

Fitxategi bidez kargatzeko pickle artxiboak (.pkl) sortuta eduki behar dira. Horrela, lista bakoitzerako .pkl artxibo bat kargatuko genuke. Motxilaren tamaina kargatzeko, berriz, .txt fitxategi bat igoko genuke, zenbaki batez osatuta egongo dena. Hau prozesatzeko, Artxiboak igo sakatu beharko da.

Bai datuak eskuz sartuta, edo bai fitxategi bidez sartuta, emaitza orrira joango da web aplikazioa (ikus 5.3 irudia).



5.3 Irudia: Proba kasu txiki baten emaitza

Bertan bi atal nagusi ikus daitezke. Bata, sarrera datuak eta emaitza idatziz agertzen diren atala, ezker aldean. Bertan, sarrera moduan pasatako bi listez aparte, motxilaren tamaina ere agertzen da.

Emaitza datuen atalean, erabili diren pisuak, beraien balioak eta lortu den balio maximoaz gain, motxilan bete den kopurua eta libre geratu den kopurua azaltzen dira. Beraz, erabiltzaileak eskuragarri izango ditu sarrera-irteerako datu guztiak.

Sarrera-irteera datu hauen azpian, bi botoi ikus daitezke. Agerikoa den moduan, lehenak, sortu den SVG irudia deskargatzeko balio du. Bigarrenak, aldiz, sarrera eta irteera datu guzti hauekin .txt fitxategi bat sortu eta deskargatzen du (ikus 5.4 irudia).

Erabiltzailea emaitza orrian dagoenean, goiko menuan bi aukera agertzen dira: horrela, erabiltzaileari hasiera orrira itzultzeko aukera emateaz gain, bigarrena dagoen *Beste proba bat egin* aukera ageri da. Beraz, ez da zertan hasiera orrira itzuli beharrik izango, beste sarrera batekin proba egin nahi bada.

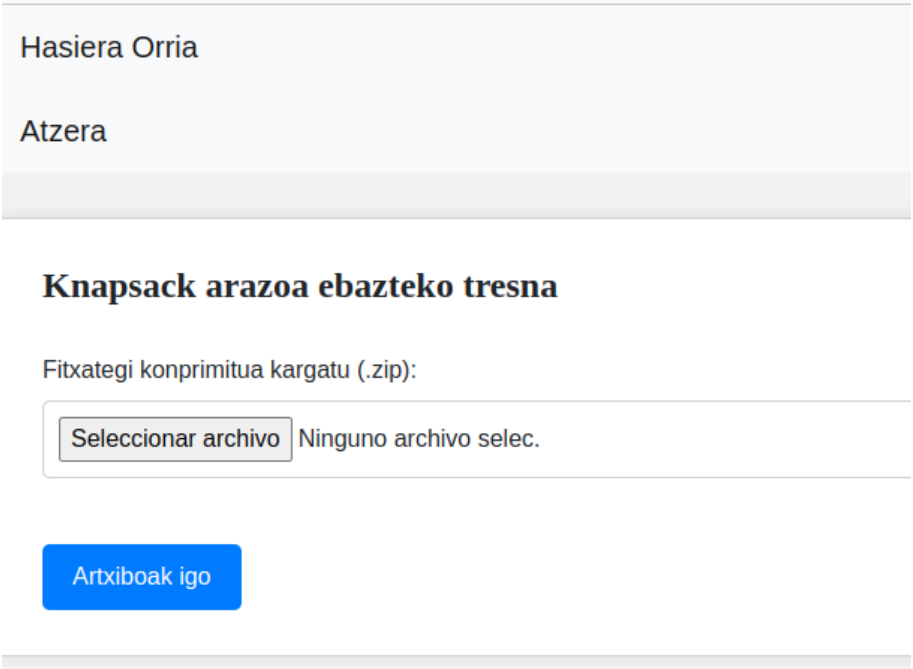
```
KNAPSACK PROBLEMA

SARRERA DATUAK:
Knapsack pisu zerrenda: [2,4,6,8,9,12,34,67,89]
Knapsack balio zerrenda: [1,23,45,67,89,123,234,456,678]
Motxilaren tamaina: 250

EMAITZA:
Knapsack BIDERAGARRIA
Soluzioan erabilitako pisuak: [2,4,6,8,9,12,34,67,89,19]
Soluzioan erabilitako balioak: [1,23,45,67,89,123,234,456,678]
Motxilan bete den kopurua: 231
Lortu den balio maximoa: 1716
Motxilan geratu den lekua: 19
```

5.4 Irudia: Proba kasuaren datuekin deskargatu den emaitzaren .txt fitxategia

Emaitza orriko 5.3 irudi eskuinaldean dagoen beste zatia irudiaren atala da. Emaitzako pisu bakoitzari, pisu horrek emaitzan duen portzentailaren araberako laukia sortzen dio.

Hau da, emaitza moduan lauki txikiz osatutako karratu nagusi bat emango da. Karratuko laukitxo bakoitzak duen tamainarekin adierazi nahi da, zati bakoitzak motxilan okupatuko lukeen lekua, bere pisua adieraziz. Motxilan libre geratu den zatia, gainera, kolore ilunez ezberdinduta dago.

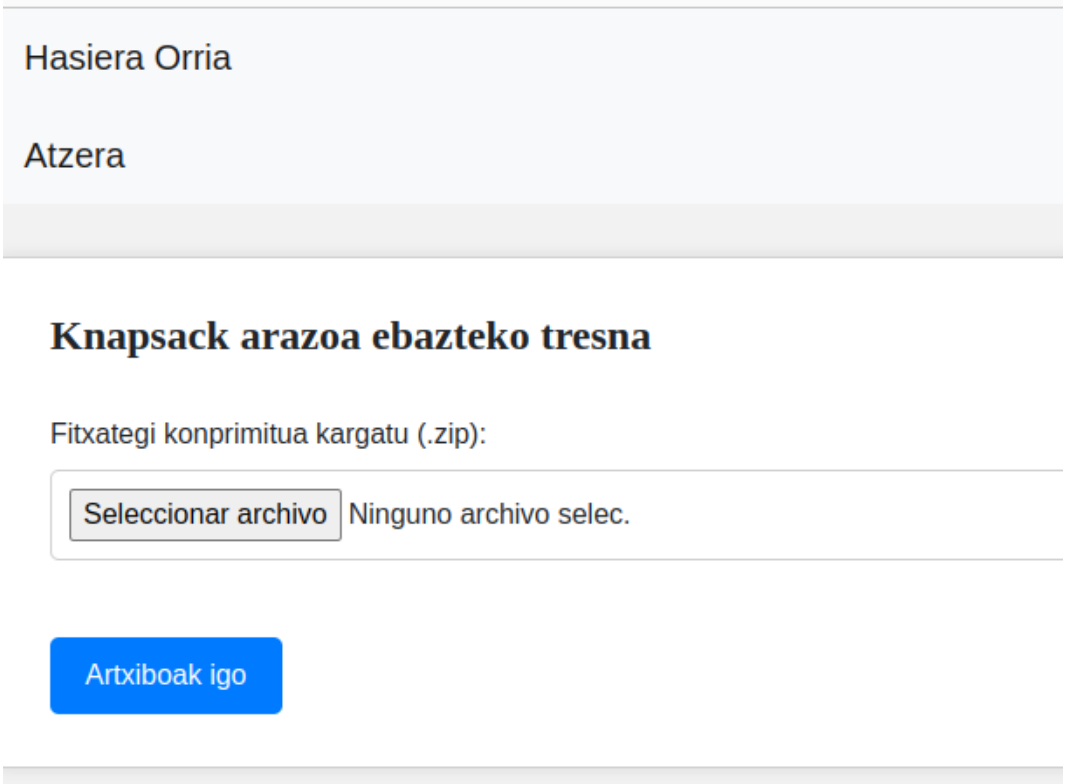
Modu honetan, erabiltzaileari ahalik eta gehien erraztu nahi zaio emaitza ulertzea. Interpretazio grafiko honekin, emaitza ikusgarriago egiten bada ere, sarrera datu handiak direnean, lauki kopurua izugarri handitzen da eta, beraz, ondorengo atalean azalduko den moduan Zoom egin ahal izango da irudia handitu eta laukiak ondo ikusteko.



5.5 Irudia: .zip fitxategia kargatzeko orria

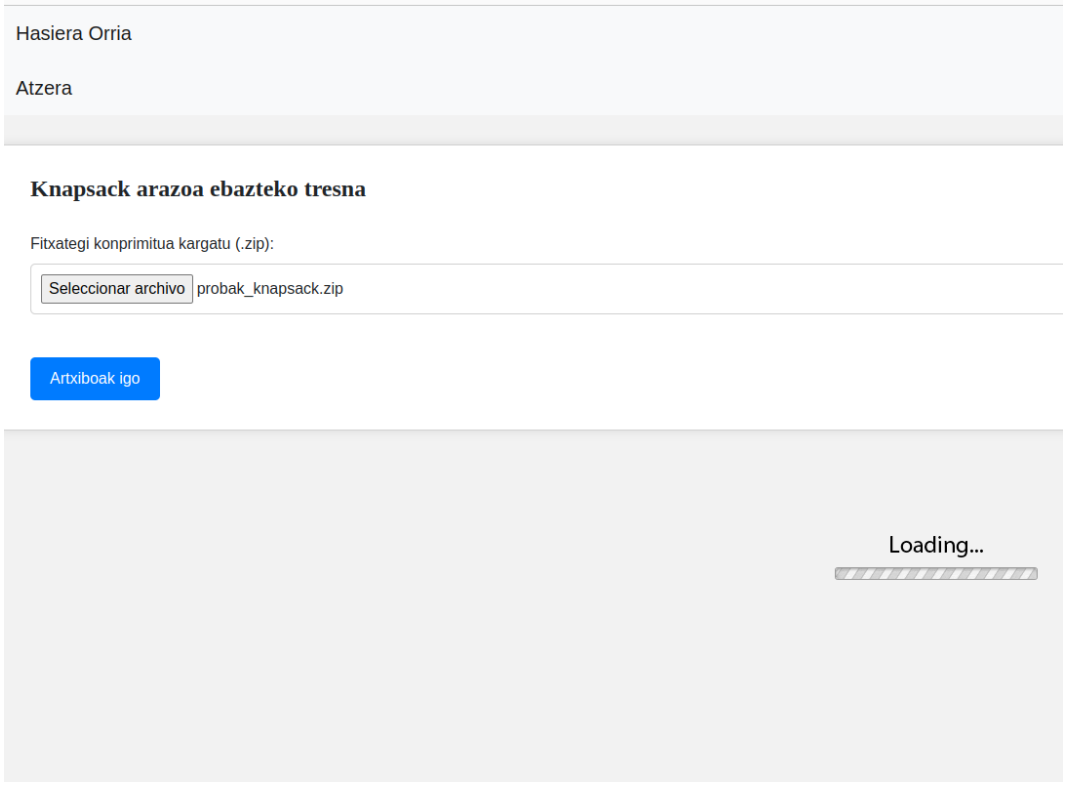
Oinarrizko funtzionalitatea hauez gain, fitxategi bat baino gehiago batera exekutatzea ahalbidetzen duen atala aipagarria da. Horretarako, 5.2 irudian ikusten den web aplikazioaren orrian, fitxategi multzoa kargatu sakatu beharko litzateke.

Honek, .zip fitxategi bat kargatu daitekeen orrira eramango gaitu (ikus 5.6 irudia). Bertan, aurreko atalean azaldu den izen formatuz osatutako .zip fitxategi bat kargatu ahalko da. Horrela, problema ezberdinak paraleloan exekutatuko dira eta emaitza guztiak ikusi ahal izango dira.



5.6 Irudia: .zip fitxategia kargatzeko orria

Aipatu behar da, proba kasu asko eta handiak direnean, denbora bat behar duela aplikazioak exekuzioa amaitu arte. Denbora horretan, erabiltzaileak emaitza kargatzen ari dela jakin dezan, loading moduko ikonoa agertuko da exekuzioan dirauen denboran (ikus 5.7 irudia). Behin exekuzioa amaitzean, emaitza orria agertuko litzateke.



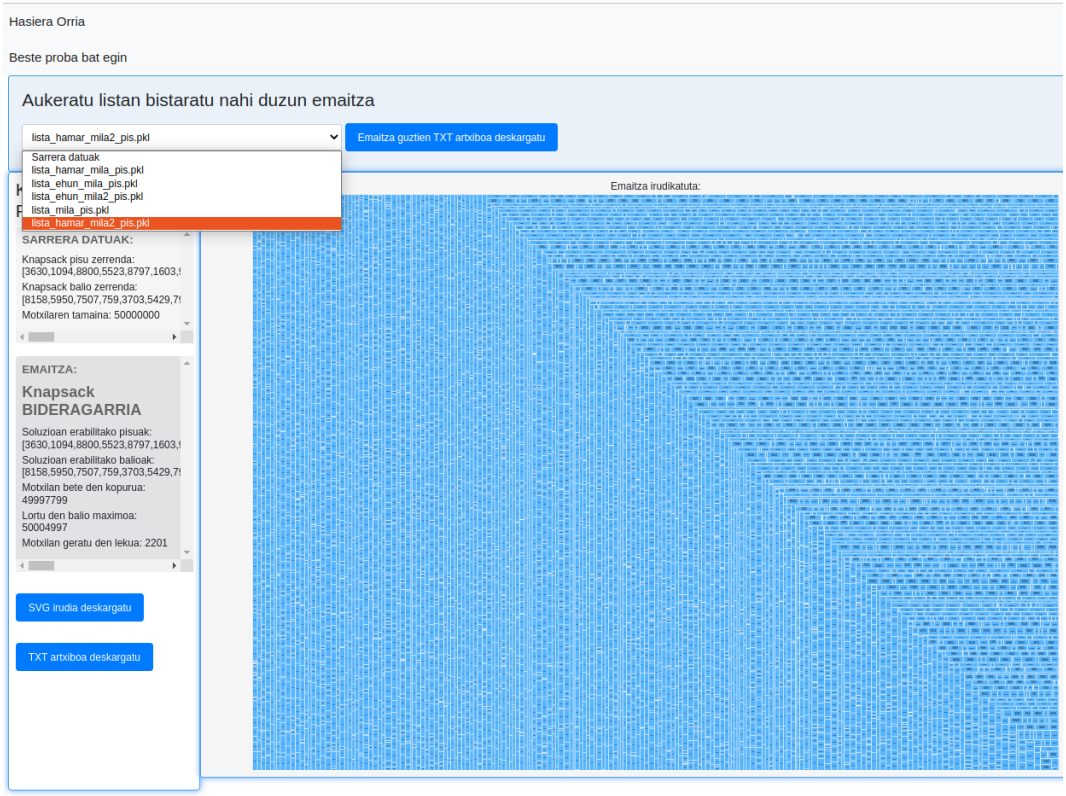
5.7 Irudia: Emaitza kargatzen ari dela ikusi dezake erabiltzaileak

Emaitza orri honetan atal bat gehitzen da. 5.8 irudian ikus daitekeen moduan, goikaldean dagoen atalean, zer sarreraren emaitza ikusi nahi dugun aukeratu ahalko dugu.

Bertan sarrera aukeratzeko lista zabalgarri bat agertzen zaigu. Exekuzioa eginda dagoenez, listako edozein proba kasu aukeratu bezain laster, fitxategi bakarrekin ematen zen moduko emaitza ikusi ahal izango da. Ageriko den moduan, emaitza ezberdinak ikusten joan ahalko du erabiltzaileak listan aukera aldatzen duen bakoitzean.

Goiko atal honetan ikus daiteke, gainera, botoi berri bat gehitu dela. Botoi honen bidez, erabiltzaileak emaitza guztiak jasotzen dituen .txt fitxategi bat deskargatu ahal izango du. 5.4 irudiaren fitxategi berdin bat deskargatuko luke, ezberdintasun bakarrarekin, emaitza guztiak bata bestearen azpian egongo direla.

Emaitza orrian, bai eskuz sartutakoan, eta baita fitxategi bidez sartutakoan ere, orriaren goikaldean beti emango du aukera beste proba bat egiteko edo hasiera orrira itzultzeko.



5.8 Irudia: Hainbat fitxategiren exekuzio emaitza

KAPITULUA 6

LORTUTAKO EMAITZAK

Lortu diren bi helburuak azalduko dira ondoren. Batetik sortu den laburketaren, eta, Googleren Ortools liburutegiarekin egindako bertsioen arteko alderaketak egingo dira. Froga guztiak egiteko, ASUS fx506hcb-hn200 ordenagailu bat erabili da. Bere hardware-aren datuak hauek direlarik: Intel Core i5-11400H (6 nukleo, 12 hari, 2.7-4.4 GHz), 16 GB RAM memoria (DDR4), Nvidia GeForce RTX 3050 txartel grafikoa (12GB GDDR6) eta SSD 500GB memoria.

Konparaketez gain, sortzen den irudiaren inguruko xehetasunak eta adibideak jarri dira, irudi bidez.

6.1 Bi bertsioen alderaketak

Laburketa bidez lortutako exekuzio denborak, Ortools liburutegiarekin egindako bertsioarekin alderatu ditut hainbat proba kasu eginda. Gure helburua, bai, Subset Sum, eta, bai, Knapsack problemak laburketa bidez ebaztea izan da. Ortools liburutegia, aldiz, bi problema hauek ahalik eta modu eraginkorrenean ebaztera bideratuta dagoela jakinda, aldea nabaria izango dela aurreikusi daiteke.

Ondoren ageri dira problema bakoitzean egin diren proba kasuak, exekuzio denborekin. Sarrera datuak, ausaz sorturiko listak dira, eta, taulan sartzeko handiegiak direnez, n=xxx bidez adierazi da sarrera datuen tamaina kasu bakoitzean. Horrela, n=100 jarriko balu, 100 zenbakiko lista bat sartu dela esan nahiko luke. Emaitzak aldiz, segundotan daude kasu guztietan.

Bi kasuetan ere, paralelizazioa aplikatu dut. Hau da, hainbat fitxategi kargatu eta batera exekutatu daitezen nahi dudanez, proba kasu multzo batzuk paraleloan exekutatzu probak egin ditut. Proba hauek guztiak ere, taulan daude ikusgarri, noski.

6.1.1 Subset Sum problema

Hasteko, proba kasu txiki batzuekin probatu dut, eta, handiagoekin gero. Bai laburketaren bertsioa, eta, bai Ortools-ena paralelizatuta dauzkat, esan bezela, hainbat proba kasu batera

exekuta ditzan. Beraz, proba kasuak banaka exekutatuta eta paraleloan exekutatuta lortzen den aldea ere ikusgarri jarri dut.

SUBSET SUM	n	PYSAT	Paralelizatuta	ORTOOLS	Paralelizatuta
1	10	4.84e-4		7.57e-5	
2	100	8.84e-3		8.23e-5	
3	1000	0.136	114.76	4.98e-4	0.37
4	10,000	3.76		4.46e-3	
5	100,000	113.45		0.03	
6	10,000	2.53		3.54e-3	
7	10,000	3.13		3.25e-3	
8	10,000	2.86		3.65e-3	
9	10,000	2.57		3.65e-3	
10	10,000	2.76		3.29e-3	
11	10,000	2.70	7.68	3.25e-3	0.19
12	10,000	2.16		3.65e-3	
13	10,000	3.06		3.49e-3	
14	10,000	2.42		3.98e-3	
15	10,000	2.99		3.79e-3	
16	10,000	2.74		3.45e-3	
17	10,000	2.62		3.52e-3	
18	100,000	125.45		14.05	
19	100,000	115.54		0.02	
20	100,000	97.45		10.75	
21	100,000	101.15		0.12	
22	100,000	135.47		12.24	
23	100,000	99.65	—	0.02	17.38
24	100,000	105.68		0.04	
25	100,000	122.02		0.02	
26	100,000	126.52		10.18	
27	100,000	101.89		0.03	
28	100,000	139.54		0.02	
29	100,000	109.58		10.79	
30	1,000,000	—		122.25	
31	1,000,000	—	—	118.58	—
32	1,000,000	—		119.87	
33	1,000,000	—		124.59	
34	10,000,000	—	—	3,178.00	

6.1 Taula: Subset Sum denbora konparaketak (segundotan)

6.1 taulan ikusten den moduan, proba kasu txikietarako aldea ez da hainbestekoa, baina, sarrera datuak handitzen goazen moduan, aldea asko handitzen da. n=10.000 den kasuan jada aldea handia da, baina, alde hau nabarmenagoa bihurtzen da n=100.000 kasurako.

6. lerrotik aurrera, datu kantitate berarekin egin ditut hainbat proba ezberdin. Honen arrazoiak bi dira. Batetik, datuak ausaz sortzen ditudanez, ikusi nahi izan dut, benetan denborak mantentzen direla antzeko kasuetan ere.

Bestetik, web aplikazioa ahalik eta eraginkorrena izan dadin, paralelizatu egin dut. Hau da, hainbat fitxategitatik jaso ditzake datu ezberdinak, eta, batera exekutatu. Paralelizazioaren eraginkortasuna probatu nahi izan dut, beraz.

Paralelizario kasuetan, emaitzak argiak dira. Proba kasu txikiak direnean denbora gehiago pasatzen du paralelizatuta dagoen bertsioak, oso azkar egiteko eragiketak baitira, eta, paralelizatzen denbora gehiago pasatzen du.

Proba kasuak zenbat eta handiagoak izan, ordea, orduan eta eraginkorragoa da sistema hau, eta, orokorrean esan daiteke, denborak asko hobetzen direla.

n=1.000.000 kasuan Pysat-en bertsioak ez du emaitzarik ematen nire ordenagailuan,

denbora asko utzi arren. Ortools bertsioa, aldiz, n=10.000.000 proba kasua ere exekutatzeko gai da ia ordubete pasa eta gero.

6.1.2 Knapsack problema

Oraingo honetan ere, kasu proba txikiekin hasi naiz lehendabizi. Subset Sum-en bertsioak, Pysat-eko SAT solverra behin exekutatzen du emaitza lortzeko. Kasu honetan, ordea, bilaketa dikotomiko bat egiten da, aldi bakoitzean SAT solverra aplikatuz.

Kontuan hartuta, algoritmoan denbora gehien pasatzen duen atala SAT solverra dela, aurreikusi genezake denboretan aldea handiagoa izango dela, Subset Sum probleman baino.

Knapsack	n	PYSAT	Paralelizatuta	ORTOOLS	Paralelizatuta
1	**10**	2.65e-2		4.07e-4	
2	**10**	2.02e-2		3.54e-4	
3	**10**	2.56e-2	0.20	8.56e-5	0.13
4	**10**	2.84e-2		1.25e-4	
5	**10**	2.25e-2		8.75e-5	
10	**15**	11.391		2.56e-4	
11	**15**	10.635		2.15e-4	
12	**15**	11.524		2.56e-4	
13	**15**	11.254		2.87e-4	
14	**15**	11.365		2.64e-4	
15	**15**	11.589	22.98	2.56e-4	0.14
16	**15**	10.985		2.58e-4	
17	**15**	10.856		2.35e-4	
18	**15**	11.025		2.67e-4	
19	**15**	11.305		2.36e-4	
20	**15**	10.992		2.33e-4	
21	**15**	11.52		2.29e-4	
22	**16**	40.55		1.75e-4	
23	**17**	168.25		8.75e-5	
24	**18**	912.25		9.89e-5	
25	**19**	—	—	2.05e-4	0.52
26	**20**	—		1.98e-4	
27	**100**	—		9.51e-5	
28	**1,000**	—		3.34e-3	
29	**10,000**	—		0.366	
30	**100,000**	—		51.76	
31	**100,000**	—		53.46	
32	**100,000**	—		49.26	
33	**100,000**	—	—	52.59	156.51
34	**100,000**	—		50.06	
35	**100,000**	—		50.86	
36	**100,000**	—		51.65	
36	**100,000**	—		51.07	

6.2 Taula: Knapsack denbora konparaketak (segundotan)

Espero moduan, proba kasu txikienetan ez dago izugarrizko alderik. Sarrera datuak handitzean, ordea, segituan handitzen da bi bertsioen arteko denbora diferentzia.

Pysat bertsioak, proba kasuak 19 elementu dituenean ez du jada emaitzarik ematen. Ortools-ekin egindako bertsioak, aldiz, 100.000 elementuko listekin ere emaitza ematen du. Aldea nabaria da, eta aurretik esan bezala, SAT-era laburtzeko bilaketa dikotomiko bat egin beharra dagoelako gertatzen da hau.

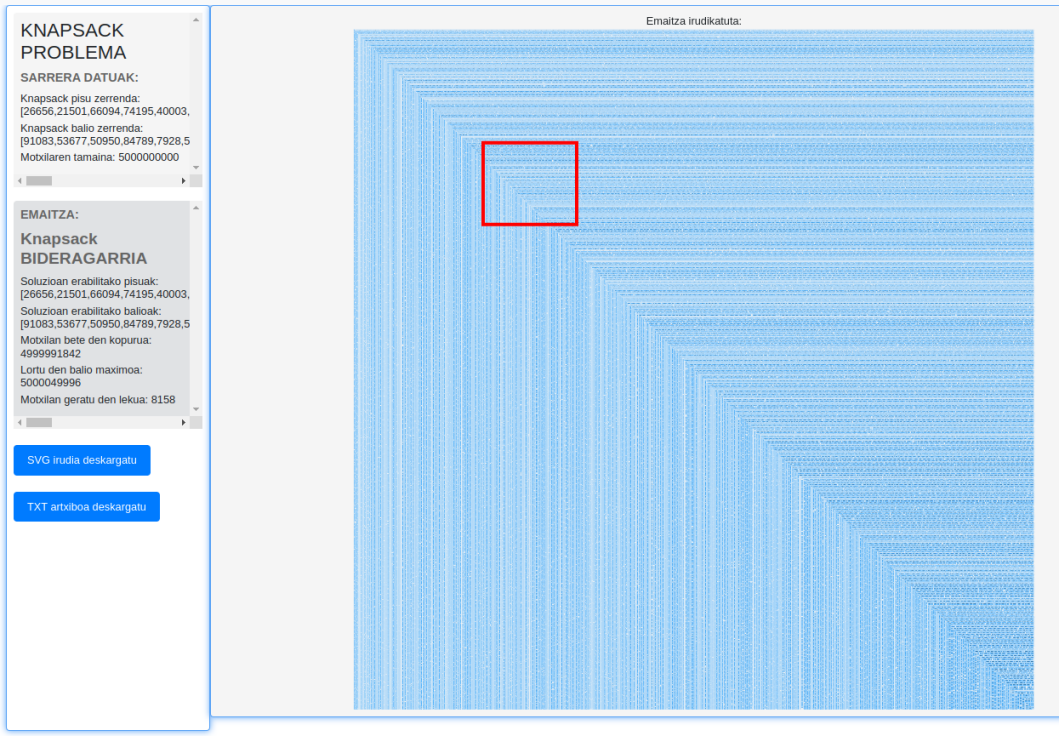
Paralelizazioan, Subset Sum-en antzera kasu txikiak direnean ez da eraginkorra, denbora gehiago pasatzen duelako paralelizatzen. Proba kasuak zenbat eta denbora gehiago pasa,

ordea, orduan eta eraginkorragoa da teknika, bai Pysat bertsioan eta baita Ortools-ekin egindako bertsioan ere.

6.1.3 Sortutako irudiaren azken emaitza

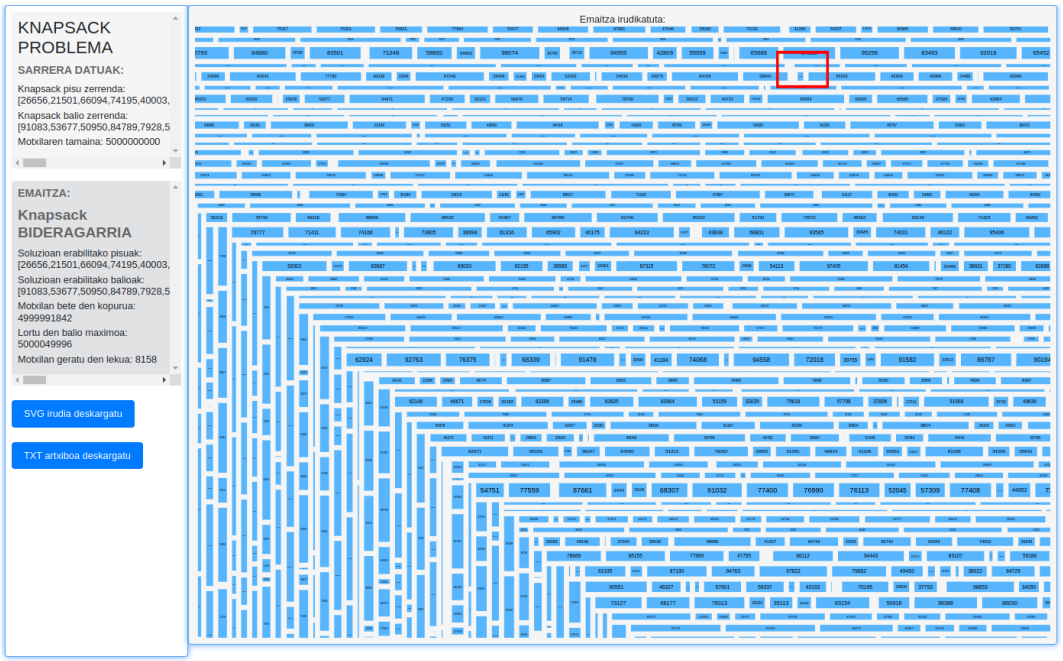
Proiektu honetako beste erronka bat irudi egoki bat sortzea zen. 5.3 irudian ikusten den moduan proba kasu txikietan emaitza ondo irudikatzen da, baina, proba kasu handia denean, 5.8 irudia kasu, irudi hau interpreta ezina gertatzen da, datu asko daudelako.

Lortu den emaitzaren eraginkortasuna ikusteko adibide handi bat jarriko dut. Sarrera moduan ehun milako luzera duen bi lista pasa dizkiot Knapsack ebazteko sistema honi. 6.1 irudian ikus daiteke emaitza.



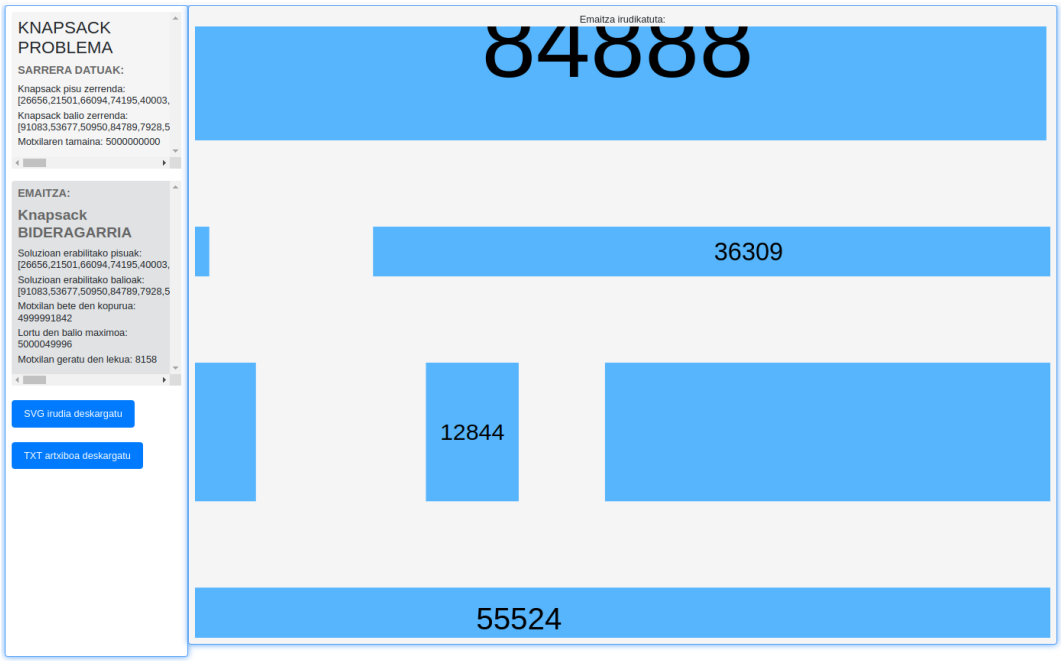
6.1 Irudia: Ehun milako luzerako Knapsack problemaren emaitza

Ikusten denez, ezinezkoa da emaitzako irudia interpretatzea, hainbeste datu daudenez oso txikiak baitira sortzen diren laukiak. Zoom-a egin diot eta 6.1 irudian gorriz markatu dudan laukia ikusiko da orain.



6.2 Irudia: Ehun milako luzerako Knapsack problemaren emaitza zoom eginda

Oraingo honetan, emaitzak ikusten hasi gintezke, baina, badago oraindik Zoom gehiago egiterik. Oraingo honetan ere gorriz markatu dut Zooma aplikatu diodan atala 6.2 irudian.



6.3 Irudia: Ehun milako luzerako Knapsack problemaren emaitza zoom gehiago eginda

Ikusten den moduan, Zoom-a aplikatuta laukiak ikusgarriak egiten dira eta ez du inongo kalitaterik galtzen. Hau da SVG irudia aukeratzearen arrazoi nagusia, aurretik aipatu

moduan, irudiak artxibo bektorial modukoak baitira. Behin Zoom-a egiteko inplementazioa gehituta irudi guztiak ikusteko adina Zoom egin genezake, datu guztiak ondo ikusteko.

KAPITULUA 7

ONDORIOAK

Amaitzeko, proiektu honen inguruko ondorio batzuk atera ditut.

Proiektu honen hasieran, bi helburu nagusi planteatu ziren. Batetik, Subset Sum eta Knapsack problemetatik SAT problemarako laburketak egitea. Laburketa hauek, proba kasu handientzat, eraginkorrak izatea helburu moduan jarri zen, eta horretarako Pysat paketea erabiliko zela erabaki zen. Bestetik, aldiz, laburketa hauek erabiliz, web aplikazio eraginkor bat sortu nahi zen problema hauek ebatziko zituena.

Lehen helburua, bi laburketak egitearena, lortu da, baina, ez nahi bezala. Bi kasuetan, hasieran beste bertsio batzuk egitea lortu nuen, aurretik azalduta utzi dudan moduan. Bertsio hauek, laburketa egiten zuten, baina, ez ziren batere eraginkorrak.

Hainbat aldaketa, eta, Pysat-eko beste modulu batzuekin lan eginda, bi kasuetan eraginkortasuna nabarmen hobetzea lortu dut. Subset Sum-en kasuan, emaitza oso eraginkor bat sortu dudala esango nuke. Garbi dago, Ortools-en bertsioarekin alderatuta eraginkortasun okerragoa duela, baina, laburketa bidez lortu dela kontuan hartuta, hasieran planteatu zen eraginkortasun helburua lortu dela esan daiteke.

Knapsack kaproblema, aldiz, optimizazio problema dela kontuan hartu behar da. Honek esan nahi du, emaitza posible guztietatik hoberena aurkitu behar dela. Beraz, bilaketa dikotomiko bat egiteko beharra ikusi da, eta honek, eraginkortasunean eragin handia izan du. Behin eta berriz, SAT solverra erabili beharra, exekuzio denbora aldetik emaitza txarrak lortzera eraman nau. Laburketa, izatez, lortu da, baina, Ortools-en bertsioaren alderatuta, emaitzetan aldea oso nabaria da.

Laburbilduz, Subset Sum problemarako, laburketa bat sortu da, eta eraginkorra da. Knapsack problemarako, berriz, laburketa sortu da, baina, ez da behar bezain eraginkorra izan.

Sortu den web aplikazioaren inguruan ere, hainbat gauza esan daitezke. Hasteko, bi problema hauek ebazteko, Pysat-ekin sortuko nituen bertsioak erabiltzea zen asmoa. Denbora alderaketak egin ostean, ordea, Knapsack kasuan batez ere, web aplikazio eraginkor bat ezingo zela egin ikusi nuen.

Arriskuen atalean planteatu nuen moduan, laburketa eraginkorrik sortzea lortu ezean, zuzeneko solver bat bilatzea izan behar zuen hurrengo pausoak. Horretan jardun, eta,

Ortools liburutegia topatu ostean, konturatu nintzen, Subset Sumerako ere errendimendua hobetzen zuela. Beraz, liburutegi honen bidez ebazten ditu web aplikazioak bi problemak.

Hasierako helburua hau ez izanagatik, amaierako emaitza ikusita, web aplikazio benetan interesgarri bat sortu dela esan daiteke. Izan ere, Pysat bidez sortu ditudan laburketak erabili izan banitu, denbora gehiago tardatuko zukeen exekuzioan, eta, proba kasu handienak ez lituzke egingo.

Ortools-en exekuzioa paralelizatzeak ere beste kalitate puntu bat eman diola esango nuke. Posible baita, datu handiak dituzten hainbat fitxategi batera exekutatzea, eta emaitzak ikusi ahal izatea. Web aplikazio eraginkorra lortzeko helburan zati garrantzitsua dela deritzot.

Web aplikazio eraginkor bat sortzeaz gain, emaitza grafikoki ondo interpretatzea lortu nahi zen bestetik. Kasu honetan SVG irudia sortzea, benetan arrakastatsua izan dela uste dut. Aurreko atalean azaldu moduan, irudi handienekin ere, irudia handitu daiteke, eta grafiko egoki bat ikusi.

Ez nuke aipatu gabe utzi nahi, Subset Sum problematik SAT problemarako laburketa eraginkor bat sortu denez, izatez, NP problema guztientzat SAT solver bidezko ebazpen bat sortu dela. Izan ere, hasierako oinarri teorikoan azaldu moduan, Subset Sum, NP-Osoa den problema bat da, eta, horrek esan nahi du, frogatuta dagoela edozein NP problema, Subset Sum problemara laburtzea dagoela.

Etorkizunera begira, proiektu honek hobekuntza batzuk izan ditzake. Hobekuntza nagusiena agerikoa da, Knapsak-etik SAT-erako laburketa hobetzea. Laburketa hau egiteko beste moduren bat topatu beharko litzatake horretarako, agian Pysat liburutegitik kanpo, beste moduren batera planteatuta. Nahiz eta esan moduan optimizazio problema bat denez, ez den erraza izango.

Hontaz aparte, web aplikazioaren denbora errendimendua asko hobetzea ez dut bideragarria ikusten, batez ere paralelizatuta dagoelako. Baina, ziur , zer edo zer aldatzea egongo dela errendimendu hau apur bat hobetzeko. Bestalde, diseinua eta funtzionalitateren bat gehitzea izango litzateke hobetzeko modua.

Laburbilduz, beraz, proiektu honetan jarri ziren helburuetatik, batzuk ez da nahi bezala osatzea lortu. Baina, edozein NP problemarentzako SAT solver bidezko ebazpena lortu izanak, eta, web aplikzaioaren emaitza ona ikusteak, proiektu egoki bat garatu dudala pentsatzera narama.

Bibliografia

[1] Np-completeness. https://en.wikipedia.org/wiki/NP-completeness#/media/File:P_np_np-complete_np-hard.svg/, 2007. Ikusi v, 2 orrialdeak.

[2] Richard M. Karp. *Reducibility Among Combinatorial Problems*, pages 85–103. Complexity of Computer Computations, 1972. Ikusi 2 orrialdea.

[3] Subset sum problem. https://en.wikipedia.org/wiki/Subset_sum_problem. Ikusi 15 orrialdea.

[4] Knapsack problem. https://en.wikipedia.org/wiki/Knapsack_problem. Ikusi 16 orrialdea.

[5] Stephen A. Cook. The complexity of theorem-proving procedures. http://4mhz.de/cook.html, 1971. Ikusi 16 orrialdea.

[6] Pysat: Sat technology in python. https://Pysathq.github.io/. Ikusi 21 orrialdea.

[7] Google ortools. https://developers.google.com/optimization?hl=es-419. Ikusi 21 orrialdea.

[8] Qué es flask (python) y cuáles son sus principales ventajas. https://www.epitech-it.es/flask-python/, 2021. Ikusi 22 orrialdea.

[9] Html: Lenguaje de etiquetas de hipertexto. https://developer.mozilla.org/es/docs/Web/HTML, 2023. Ikusi 22 orrialdea.

[10] Javascript: The world's most misunderstood programming language. http://www.crockford.com/javascript/javascript.html. Ikusi 22 orrialdea.

[11] D3.js. https://livebook.manning.com/book/d3-js-in-action/chapter-10/5, 2022. Ikusi 22 orrialdea.

[12] Spyder - scientific python development environment. https://github.com/spyder-ide/spyder/tree/v1.0.0, 2017. Ikusi 23 orrialdea.

[13] Pysat.pb. https://pysathq.github.io/docs/html/api/pb.html. Ikusi 26 orrialdea.