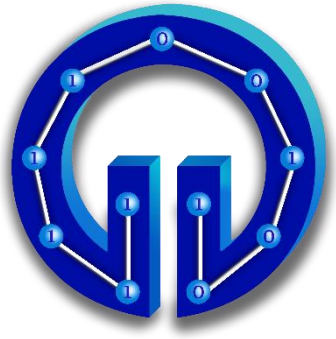
KARADENİZ TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ



SATisfiability NP-Tam(NP-Complete) Problemi

PARALEL BİLGİSAYARLAR PROJE RAPORU

365233 Nurullah ÖZTÜRK (I.Öğr.)
383188 Buğra UZUNOSMANOĞLU (I.Öğr.)
365299 Kader NUR TEKİN (I.Öğr.)
376943 Merve AYDIN (I.Öğr.)

2021-2022

İÇİNDEKİLER

1. GİRİŞ

Programımızın çalıştırılması ve sonuçlar ile ilgili bir video çekilmiştir. Bunun haricinde her bölüm için ayrı ayrı anlatım videoları da mevcuttur. Bunların linkleri ilgili bölümlerin eklerinde verilmiştir. Öncelikle giriş bölümünü okuduktan sonra program çalıştırılması videomuzu izlemenizi daha sonra yine raporda her bir bölümün sonunda linki verilen kısa anlatım videolarını izlemenizi daha sonra raporda sonuçlar kısmını incelemenizi tavsiye ederiz.

1.1. Rapor düzeni

Raporda 3 ana bölüm vardır. Seri çözüm, paralel çözüm ve resolution. Her bir bölümde ise yapılan çalışmalar özeti ve linklerin bulunduğu ekler bölümü vardır.

1.2. Proje iş bölümü

Nurullah – Kaba kuvvet yöntemi algoritma kurulması ve implementasyon, video anlatım, rapor yazımı(giriş, seri bölümü, sonuç)
Buğra – Kaba kuvvet yöntemi çözümün paralele çevrilmesi, video anlatım, rapor yazımı(paralel bölümü)
Kader ve Merve – Resolution çalışmaları, rapor(resolution bölümü)

Grubumuzdaki her bir üyenin projeye katkısı olmuştur. İletişim kanalımızda devamlı fikir alışverişleri yapılmıştır.

1.3. Uyarılar

Programın çalışması esnasında her hangi bir beklenmedik durum ile karşılaşılırsa programı kapatıp açınız.

1.4. Program çalıştırılması videosu

https://drive.google.com/file/d/1SkbMR7bwe_QtiZ_x83Ib4sLszX9ecty5/view?usp=sharing

2. KABA KUVVET YÖNTEMİ SERİ ÇÖZÜM (NURULLAH)

2.1. Yapılan çalışmalar özet (Nurullah)

Öncelikle problemin herkes tarafından anlaşıldığından emin olmak için problem hakkında bir anlatım videosu çektim ve üyelere paylaştım. Kaba kuvvet seri çözümü için algoritma kurdum ve implementasyonu yaptım. Grup arkadaşlarımın daha iyi anlayabilmesi için algoritma ve kod hakkında yardımcı video çekildi. Ortak drive klasöründe grup üyelerine kod ile birlikte paylaşıldı. İş bölümü yapıldı videoda ayrıca iş bölümünden bahsedildi. Böylece erkenden projenin ilk kısmını bitirmiş ve herkesin yapacağı işi de belirlemiş oldum. Daha sonraki aşamalarda diğer üyelerin çalışmalarıyla ilgili grubumuzda elimden geldiğince yardımcı olmaya çalıştım. Fikir alışverişinde bulunduk. En son da yazılan kodları toparladım ve final anlatım videosunu çektim.

2.2. Kaba kuvvet çözümü(seri) algoritması

a[3] //cümle dizisi (cümledeki değerleri tutar. Örn cümle “a b c” ise a[0] = a, a[1] = b, a[2] = c tutar. Buna göre en fazla 3 elemanlı cümleler işlenebilir.)

Counter = 0

Döngü: SAT==true veya Finished=true olana kadar dön {
Counter ile binary değerler dizisini(gBinary) setle. (counter=1 ise 000001 gibi)
Döngü: Dosyanın sonuna kadar satır satır oku {
c = 0 // cümle dizisinin(a[3]) indisi
Döngü: Her satır için kelime kelime oku {
Okunan “c” ise döngüden çık
Okunan “p” ise döngüden çık
Okunan “0” değilse kelimeyi gBinary dizisinin idisi olarak kullan.(
örn. 18 okunduysa gBinary[18]) değeri al ve cümle dizisine(a[c]) yaz.
Okunan “0” ise döngüden çık.
c++
}
Cümle disizindeki elemanları “veya” işleminden geçir ve sonuca “ve” işlemiyle ekle (sonuç = sonuç && (a[0] || a[1] || a[2]))
}
Counter++
}

2.3. Kaba kuvvet çözümü(seri) implementasyon

Aşağıdaki gibi fSeri() fonksiyonu tanımlanmıştır. Başlangıçta kullanıcıdan alınmış dosya adını tutan global değişken(gFileName) ile bir dosya açıldı. Satır ve kelime okuma işmeleri için line ve field stringleri tanımlandı. Sonuç bool değişkeni cnf nin sonucunu tutar. && işlemini etkilememesi için 1 olarak tanımlanmıştır. a[3] dizisi cümle dizisidir. “a b c” şeklindeki bir cümlenin elemanlarının değerlerini tutar(örn 1 0 1). Counter değişkeni denenecek olan değeri ifade eder. Örn 5 ise …0000101 binary sayısı değer olarak verilir. Flag değişkeni döngüde bazı kısımları atlamak için oluşturulmuştur. Videoda anlatılmıştır.

```
void fSeri() {

    ifstream pr(gFileName);
    string line, field;
    bool sonuc = 1;
    int i = 1, c, k;
    bool a[3] = { 0 }; // eger bir elemani tanimli
    unsigned long long counter = 0;
    int flag = 0;

    double itime, ftime, exec_time;
    itime = omp_get_wtime();
    std::cout << "hesapliyor..." << endl;

```

Problem SAT bulunana kadar veya limite ulaşılana kadar 1.döngü döner. Bu döngü değer döngüsüdür. Her adımda bir değer dizisini dener. Sonuç değişkeni her döngüde 1'e setlenir ki hesaplamayı etkilemesin. gBinary dizisi global bir değişkendir. Değerleri bir string olarak tutar("10101001…" gibi).

```
    while (!isSatisfiable && !isFinished) {
        sonuc = 1;
        gBinary = bitset<1000>(counter).to_string();

        while (getline(pr, line))
        {
            stringstream stream(line);
            // her satir icin cumle dizisi sifirlanmali
            for (int i = 0; i < 3; i++) {
                a[i] = 0;
            }
            c = 0; // a dizisinin indisi. her satir icin 0'dan basliyor
            flag = 0;
            // mevcut satirdaki bosluga kadar olan bir kelimeyi field'a yaz.
            while (getline(stream, field, ' ')) {
                // satirin ilk karakteri c veya p ise sonraki satira atla
                if (field == "c") {
                    std::cout << "(yorum satiri)" << endl;
                    flag = 1;
                    break;
                }
                if (field == "p") {
                    std::cout << "(p cnf satiri)" << endl;
                    flag = 1;
                    break;
                }
```

Bitset, counter değişkenini bir binary sayıya çevirir ve 1000 elemanlı bir string olarak gBinary de tutar(en buyuk counter değerini tutabileceğinden emin olmak için 1000 elemanlı tanımlandı). 2.döngü başlar. Bu döngü satır döngüsüdür. Dosyanın sonuna kadar satır satır okuma yapar. Bu döngüde önce cümle dizisi(a[3]) sıfırlanır. Bu dizinin indisi(c) de sıfırlanır. Ve flag sıfırlanır. 3.döngü başlar. Bu döngü kelime döngüsüdür. Her satırdaki boşluk ile ayrılmış kelimeleri okur. Okuduğu kelimelere göre karşılaştırmalar yapar ve buna göre gerekli eylemi gerçekleştirir. Okunan kelime c veya p ise kelime döngüsünden çıkar. Yani bir sonraki satıra geçeriz. Flag tam da bu sırada iş görüyor. C veya p okunmuşsa flag 1'e setleniyor. Böylece sonuç işlemlerini flag'ın değerini kontrol ederek atlayabiliyoruz.
Eğer okunan değer c veya p değilse ve sıfır da değilse bir sayı demektir. Bu sayı field değişkenine string olarak okunduğu için önce tamsayıya çeviriyoruz ve k değişkenine atıyoruz. Daha sonra eğer k değişkeni negatif ise önce pozitife çeviriyoruz.

gBinary dizisinde binary sayı dizinin sonundan dolmaya başlar. Örn 1 sayısı "…0000001" şeklinde bir dizi olarak tutulur. En sağdan başlar yani. Bu yüzden değişken sayımız(gVarNum, önceden dosyadan okunmuştur örn. P cnf 3 4 ise gVarNum 3'tür. Değişken sayısıdır yani) kaç ise değer dizisi o kadar elemanlı olması gerekir. Örn 20 değişkenli bir cnf için en fazla 20 basamaklı bir değer dizisi olabilir. Bu da gBinary dizimizin son 20 elemanıyla işlem yaptığımız anlamına gelir. 45.satırdaki ifadeye gelirsek, örneğin değişken sayımız 22 ve dosyadan okuduğumuz değer de -18 olsun. gBinary dizisinin uzunluğu(1000) – değişken sayısı(22) - 1 + 18 sonucu 995 yapar. gBinary[995]'in değerini alırız. Bu da 18.değişkenin değerine karşılık gelir. Bu değer eğer "1" ise cümle değer dizisine 0 koyarız (negatif bir sayı olduğundan değilini aldık). Sonraki kelimeye geçmeden c++ arttırız.
Eğer okuduğumuz kelime 0 ise kelime döngüsünden çıkarız.

```
39
40              if (field != "0") {
41                  // dosyadan string olarak okunan degeri tamsayiya cevirir
42                  k = stold(field);
43                  if (k < 0) {
44                      k = k * (-1); //pozitife cevir. orn: -6 --> 6
45                      if (gBinary[gBinary.length() - gVarNum - 1 + k] == '1') {
46                          a[c] = 0;
47                      }
48                      else {
49                          a[c] = 1;
50                      }
51                  }
52                  else {
53                      if (gBinary[gBinary.length() - gVarNum - 1 + k] == '1') {
54                          a[c] = 1;
55                      }
56                      else {
57                          a[c] = 0;
58                      }
59                  }
60              }
61              if (field == "0") {
62                  break;
63              }
64              c++;
```

Flag 1 değilse, yani c veya p okunmadıysa ve 0 okunduysa artık cümleyi okumuş ve cümledeki değişkenlerin değerlerlerini cümle değer dizimize(a[3]) almışız demektir. Bundan sonra sonuç işlemini hesaplamaya geçeriz. Cümlede kaç eleman okunduysa ona göre işlemimiz değişecektir. Örneğin eğer cümlede 1 eleman varsa c == 1 dir dolayısıyla ilgili işlem yapılır. Biz dosyadan genellikle 3 elemanlı cümleler okuyoruz bu yüzden genellikle c==3 işlemi yapılır. Ayrıca sonda sonucun 0a eşit olup olmadığı kontrol edilir. Eğer 0 ise diğer cümlelere(satırlara) bakmadan direk satır döngüsünden çıkılır ve 1.döngüye devam edilir.

```
            c++;
        }

        // eger c veya p satiri ise hemen sonraki satira gec (counteri arttirmamis oluyoruz)
        if (flag == 1) {
            continue;
        }

        // tek degiskenli 2 degiskenli ve 3 degiskenli cumleler icin sonuc hesaplamalari
        if (c == 0) {
            break;
        }
        else if (c == 1) {
            sonuc = sonuc && a[0];
            cout << a[0] << " " << endl;
        }
        else if (c == 2) {
            sonuc = sonuc && (a[0] || a[1]);
            cout << a[0] << " " << a[1] << " " << endl;
        }
        else if (c == 3) {
            sonuc = sonuc && (a[0] || a[1] || a[2]);
            cout << a[0] << " " << a[1] << " " << a[2] << endl;
        }
        // sonuc 0 ise digerlerine bakmaya gerek olmadigi icin bir sonraki degerler dizine geciyoruz
        if (sonuc == 0) { cout << sonuc << endl; break; }


    }

```

Satır döngüsü bittikten sonra tekrar 1.döngünün kapsamına gireriz. Sonraki değer dizisine geçmeden önce sonuç kontrolü yaparız. Eğer sonuç 1 ise SAT bulunduğu bildirilir. Binary sayı limitine ulaşıldıysa döngü bitirilir. Veya sayma sınırına ulaşıldıysa döngü bitirilir.

```

        // sonuc 1 ise sonlandir
        if (sonuc == 1) {
            isSatisfiable = true;
            std::cout << "Problem is SAT." << endl;
            std::cout << "Solution: " << gBinary.substr(gBinary.length() - gVarNum, gVarNum) << endl;
        }
        // sonuc 0, ancak butun degerler denendiyse
        if (gBinaryLimit == gBinary.substr(gBinary.length() - gVarNum - 1, gVarNum)) {
            std::cout << "Binary limit reached (" << gBinaryLimit << ")" << endl;
            std::cout << "Problem is UNSAT." << endl;
            isFinished = true;
        }
        if (counter == ULLONG_MAX) {
            isFinished = true;
            std::cout << "Counter limit reached! (" << counter << ")" << endl;
        }

        counter++;
        cout << "counter " << counter << endl;
        //std::cout << sonuc << endl;
        // dosyanin basina konumlan
        pr.clear();                 // clear fail and eof bits
        pr.seekg(0, std::ios::beg);
    }

    ftime = omp_get_wtime();
    exec_time = ftime - itime;
    printf("\ngecen sure %f", exec_time);
}
```

2.4. Ekler

NP-Tam problemi anlatım videosu
https://www.youtube.com/watch?v=Cc0pmYV6jk0&t=314s

Kaba kuvvet çözümü(seri) algoritma ve kod anlatım videosu
https://drive.google.com/file/d/1B3jG26ngZjCeJsKNJEqtPnE3onI6M224/view?usp=sharing

3. KABA KUVVET PARALEL ÇÖZÜMÜ (BUĞRA)

3.1 Yapılan çalışmalar özet (Buğra)

Seri kodda çalışan kısmın, paralel olarak icra edilmesi. Özetle, her bir thread, thread sayısına bölünmüş işlem uzayında çalışarak çözümü arayacak. (2^n/p) formülü ile her thread'in aramaya yapacağı büyüklüğü belirtebilirim. Bunları gerçekleyip, anlatımı için video çektim. Bu videoya aşağıdaki ekten ulaşabilirsiniz.

3.2 Paralel kısmın anlatılması

Paralel kısımda seri kısmın algoritması genel anlamda korunarak paralelleştirildi. Burada bazı noktalarda paralel koşulabilmesi için değişiklik yapılması gerektiğinden bu değişiklikleri kod üzerinden açıklayarak ilerleyeceğim.
Öncelikle, paralel kısımda birbirinden farklı thread'ler birbirinden farklı sıradan değişkenleri denemeye başlayacaktı. Bunun için aşağıdaki kod bloğunu yazdık.

```
59
60  void setTLimit() {
61      thLimit = pow(2, gVarNum);
62      thLimit = thLimit / tNums;
63  }
```

Bu blokta esasında, iş yükünü her bir thread'e eşit bölmek için (2^n/p) işlemini gerçekleştirdik. Tek sorun, işlemin tam sayı çıkıp tam sayılarla çalışılabilmesi için 2'nin kuvvetleri şeklinde thread sayısı seçilmesi gerekiyor.
Paralel bloğun başlangıcı şu şekilde yapılıyor:

```
#pragma once
void fParalel() {
    omp_set_num_threads(tNums);
    setTLimit();
    double itime, ftime, exec_time;
    itime = omp_get_wtime();
    std::cout << "hesapliyor..." << std::endl;
    #pragma omp parallel private(gBinary, gBinaryLimit, isFinished)
    {
```

Burada thread sayısını ayarlayıp, yukarıda açıklamasını yaptığımız bloğu çağırıp thLimit'in ayarlanmasını sağlıyoruz. Zaman hesabı için gerekli değişkenleri de ayarlayıp başlatarak işleme başlıyoruz.

```
    unsigned long long counter = omp_get_thread_num() * thLimit; //tnum * thLimit
    unsigned long long limit = (omp_get_thread_num()+1) * thLimit;
    int flag = 0;
    string limitforThread = bitset<1000>(limit).to_string();
```

Bu kısımda, thread'lerin counter'i kendi başlaması gerektiği yere ayarlanıyor. Limitleri bir sonraki thread'in başladığı noktaya ayarlanıyor. Mesela 4 Thread'li bir işleyişte Thread 3 için bu limit 2^n oluyor esasında. Bunları da programda daha sonrasında karşılaştırmak için bir string'e çevirerek saklıyoruz.

Private yapmadığımız ve threadler tarafından paylaşılan tek global değişken ise "isSatisfiable" değişkeni. Bu değişken zaten tek bir sefer değiştirilip, tüm thread'ler arasında herhangi bir thread'in doğru değeri bulup bulmadığının kontrolü için tutulduğundan private olması gerekmiyor.

Paralel kısmın işleyişi, esasında thread'lere bölünmüş olması hariç seri kısımla aynı icra ediliyor. Ondan dolayı, algoritmayı seri kısımda açıkladığımız için burada tekrar açıklamamıza gerek kalmıyor.

3.3 Paralel'in sağladığı hızlanma hakkında

Programdaki hızlanmayı 4 threadli bir kurgu için, "pr1.txt" üzerinde denediğimizde seri kaba kuvvet yönteminde 900 saniye süren bir işlem, paralel işlendiğinde 160 saniye gibi oldukça kısa bir sürede çözüldü. Esasında burada asıl hızlanmayı elde etmemiz çözümün bulunduğu değere de bağlı oluyor. Biz tüm işlem uzayını 4'e bölerek işlem yapmaya başlıyoruz. Bu noktada, thread'lerin herhangi birinin başlangıç noktasına yakın çözümü bulunan problemler için aslında verimlilik çok daha fazla artmış oluyor. Gördüğümüz gibi burada 4 thread için en idealde 4 kat hızlanma beklerken yaklaşık 5,6 kat hızlanmış bulunuyoruz. Bu hızlanma, bazı problemlerde kat kat fazla olabilir, bu açıkladığımız üzere probleme göre değişen rastlantısal bir durum.
Aynı şekilde, "pr7.txt"deki problem için ise 50 saniye süren seri çözüm yerine, paralel çözüm 4.5 saniye sürerek 11 kat gibi bir hızlanmayı beraberinde getirdi.
Bunun tam tersi için bir örnek vererek durumu açıklayalım. Örneğin 100 sayılık çözüm uzayı bulunan bir problemde, sırayla çözümü denediğinizde eğer çözüm 24. sayıda ise tek threadle de, dört thread ile de aynı hızda çözmüş bulunursunuz.

3.4 Ekler

Paralel kısım anlatım videosu
https://drive.google.com/file/d/1gw5NmQqC5pk1CtIMmcU94eDLwlyvgf0D/view?usp=sharing

4. RESOLUTION ÇALIŞMALARI (MERVE VE KADER)

4.1. Yapılan çalışmalar özet (Merve ve Kader)

DPLL algoritmasının araştırdık ve çalışma mantığını anladık. Algoritmanın implementasyonunu araştırıp bulduk(bkz:ekler). Bulduğumuz kodun tahlilini yaptık. İmplementasyonu çalıştırarak tüm dosyaları okuyup tek tek sonuçları elde ettik. Elde ettiğimiz sonuçları alarak, online ortamda direk sonuç veren araçlar ile elde ettiğimiz sonuç değerleri arasında karşılaştırma yaparak algoritmanın doğruluğunun kontrolünü sağladık.

4.2. Resolution DPLL algoritması

4.2.1. DPLL

Bir CNF'nin tatmin edilebilir olup olmadığını belirleyen algoritmadır.Mevcut SAT çözücülerinin temelini oluşturur. DPLL algoritması çoğunlukla mantıksal bir önermenin tatmin edici olup olmadığını anlamak için kullanılır. DPLL algoritması, kontrol etmemiz gereken toplam durum sayısını azaltmamıza yardımcı olarak hesaplamalarımızı çok daha hızlı hale getirir.

4.2.2. DPLL Algoritması Özellikleri

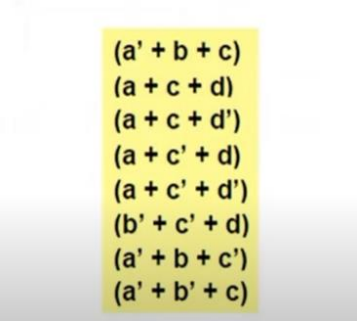
DPLL eksiksiz, doğru ve sonlandırılacağı garantili bir algoritmadır.
DPLL, varsa bir model oluşturabilir..
Genel olarak, DPLL üstel zaman gerektiren bir algoritmadır.
Şimdiye kadar tüm SAT yarışmalarında DPLL tabanlı prosedürler en iyi performansı göstermiştir.

4.2.3. DPLL algoritması, backtracking algoritmasına göre çalışan bir algoritmadır.

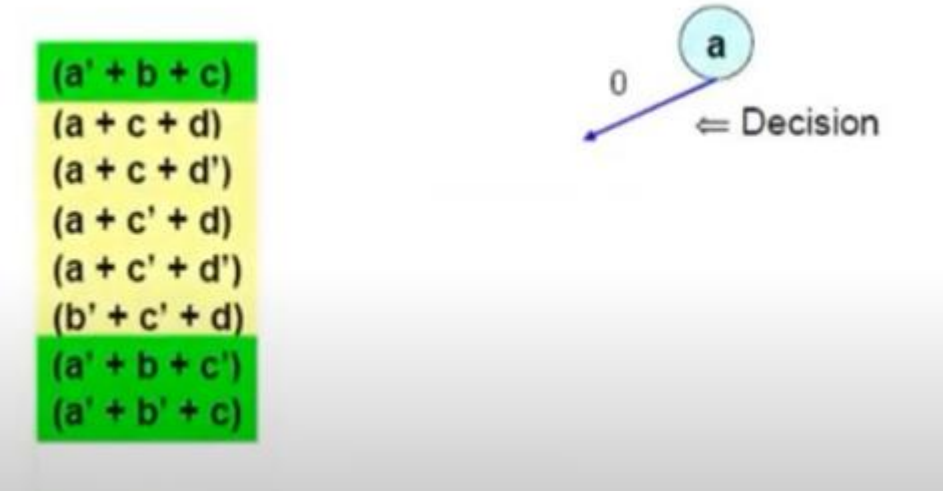
- algoritma adımları aşağıdaki gibidir.
  - ➔ 1. Atanmış doğruluk değeri olmayan bir değişken seçeriz. Hiçbiri yoksa, SAT'ı döndürür.
  - ➔ 2. Bu değişkene bir doğruluk değeri atarız(true/false).
  - ➔ 3. Formülümüzdeki tüm maddelerin hala satisfiable olup olmadığını kontrol ederiz.

  1. Eğer satisfiable ise, 1. adıma gideriz.
  2.Eğer satisfiable değillerse, 2'. adıma gidip ve diğer doğruluk değerini seçeriz.
  3. Eğer satifiable değillerse ve her iki doğruluk değeri de denenmişse, geri adım atarız.
  4. Geri izlenecek bir yer yoksa, UNSAT değerini döndürürüz.

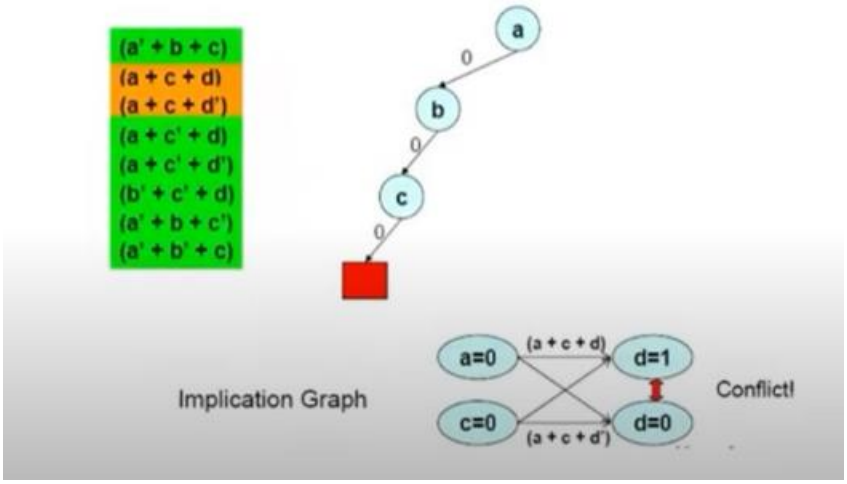
Bir örnek üzerinden algoritmayı açıklayalım.
Bir CNF formülü oluşturan tüm maddeler aşağıdaki gibi verilmiş olsun.



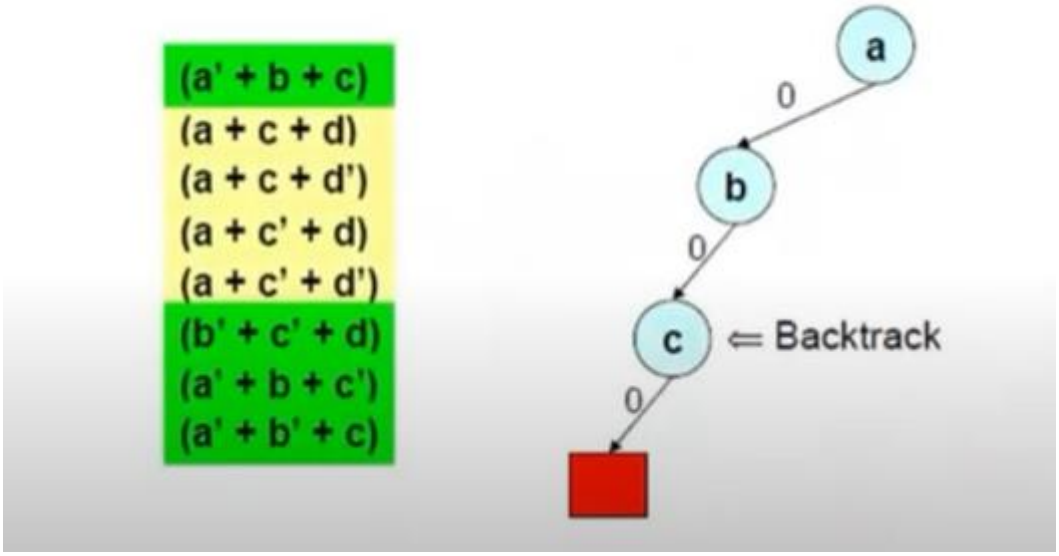
Atanmış doğruluk değeri olmayan bir değişken olarak a değişkenini seçelim(1.adım).
Ve bu değişkene herhangi bir doğruluk değeri atayalım.(2.adım). bu değer false olsun. a=0 olması durumunda aşağıda gösterilen yeşil ile boyalı olan tüm cümleler true olmuş olacaktır.



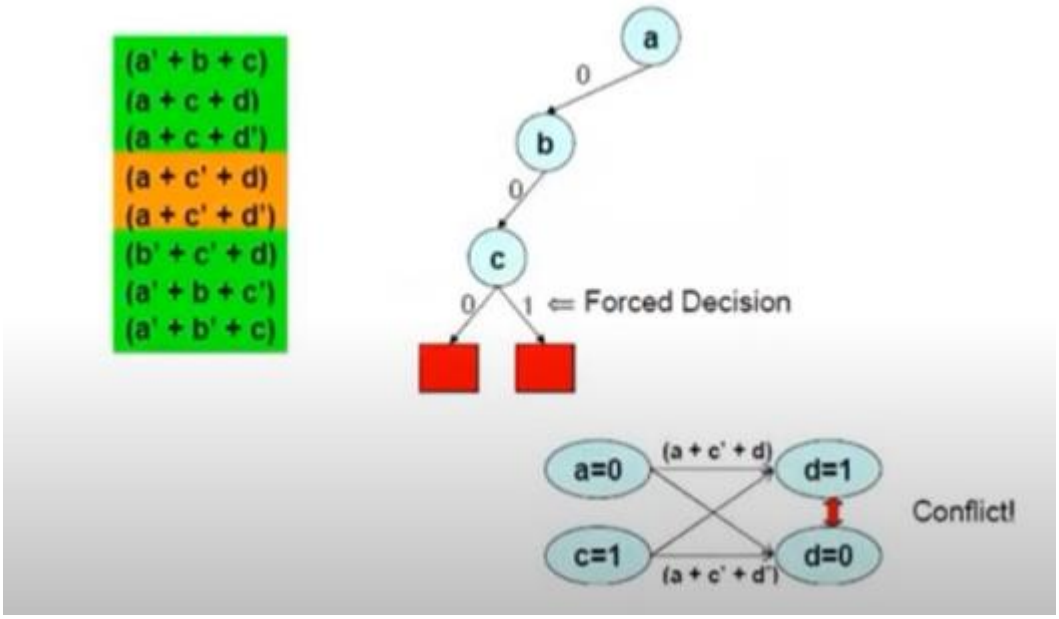
Birkaç karar verdikten sonra c değerine 0 değerini verince turuncu ile gösterilen cümlede bir çatışma ortaya çıkıyor ve buda c değerinin 0 olmaması gerektiğini bize göstermiş oluyor.



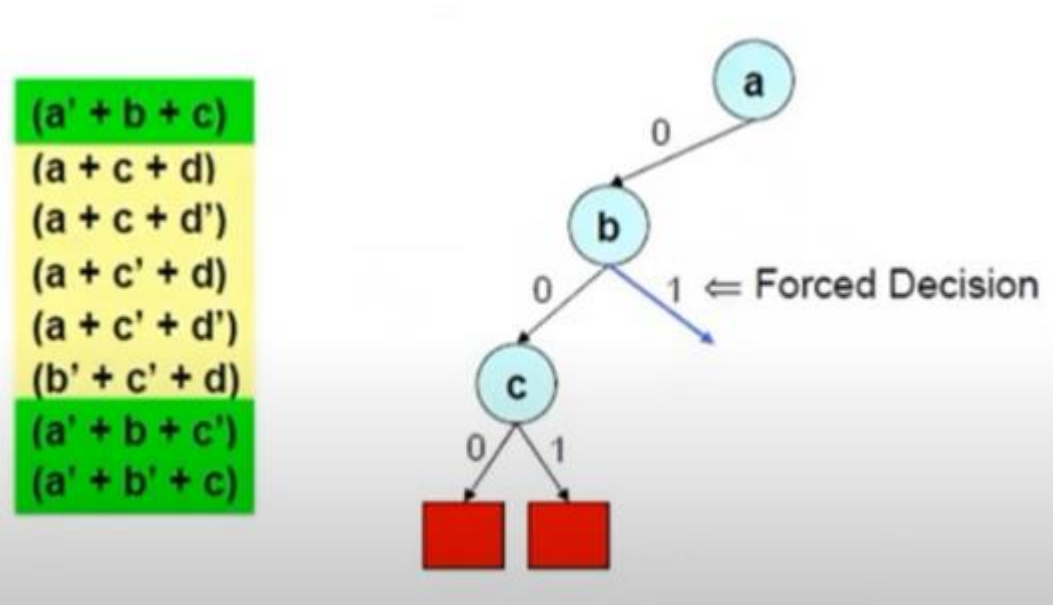
c değerinin olduğu düğüme dönerek c değerini 1 yaparız.



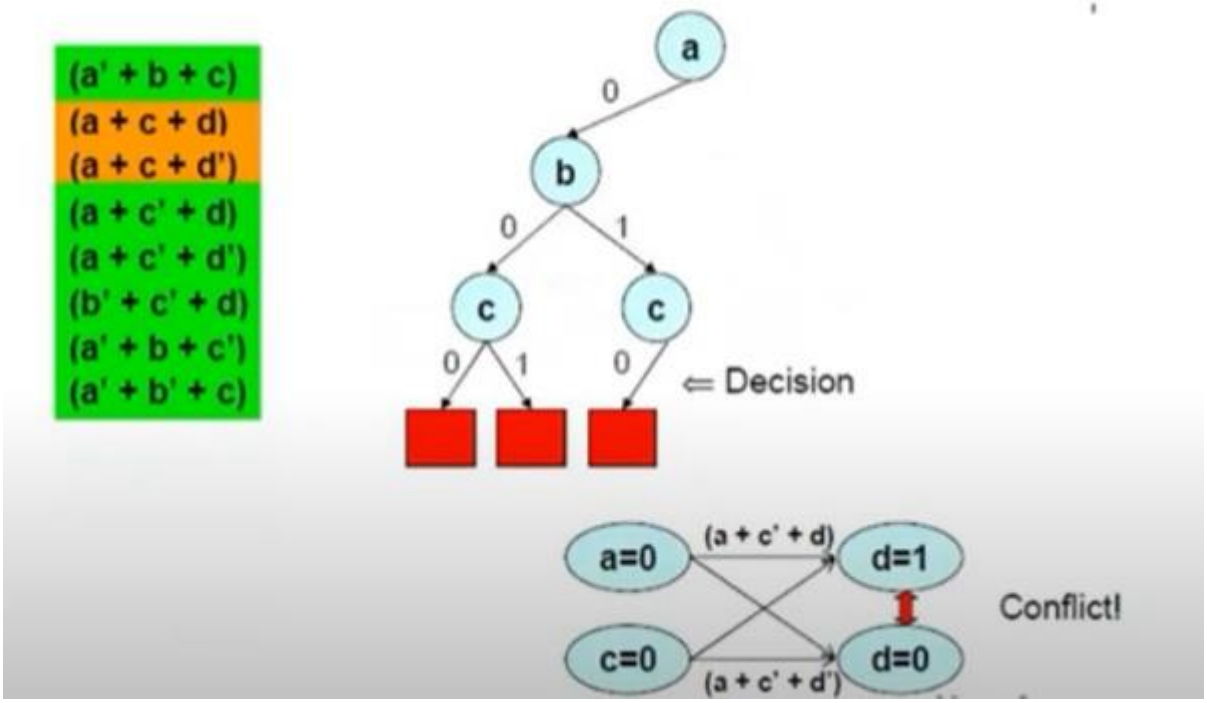
Fakat bu c değerini 1 yaptığımız zaman da başka bir çatışma ortaya çıkmaktadır. Aşağıdaki şekilde turuncu ile gösterilen kısımlarda çatışmaya yol açan cümleler gösterilmektedir.



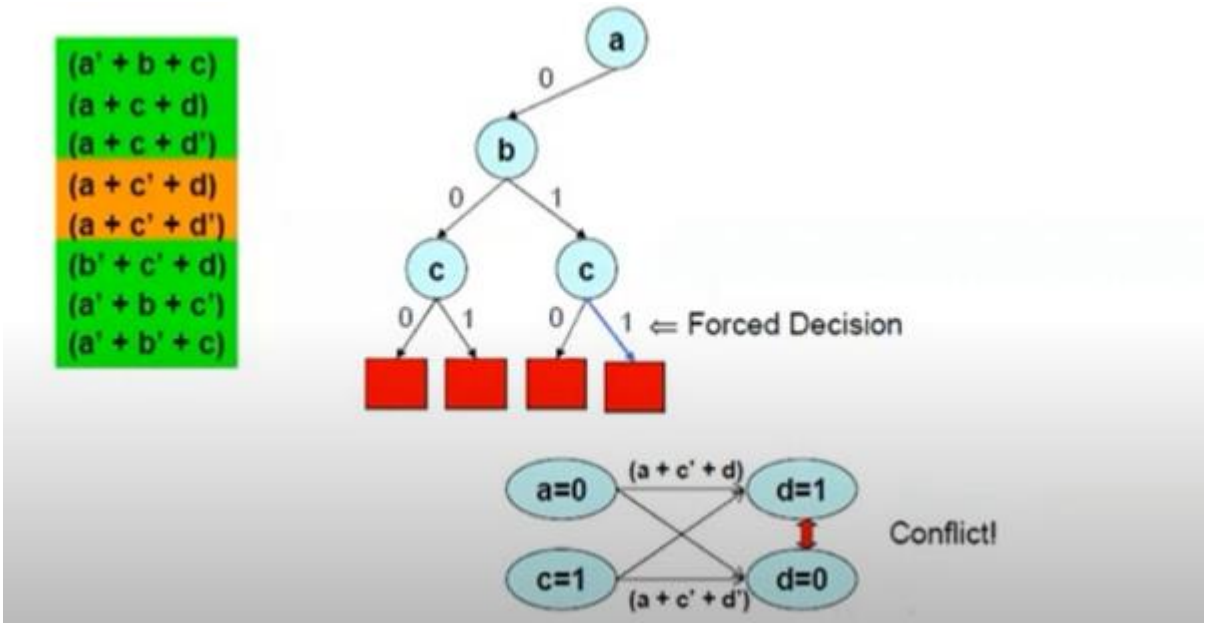
Bu durumda bir önceki düğüme dönülerek yani b değişkeninin olduğu düğüme gidilerek b değişkeninin değeri değiştirilir. b = 0 değeri b = 1 yapılır.



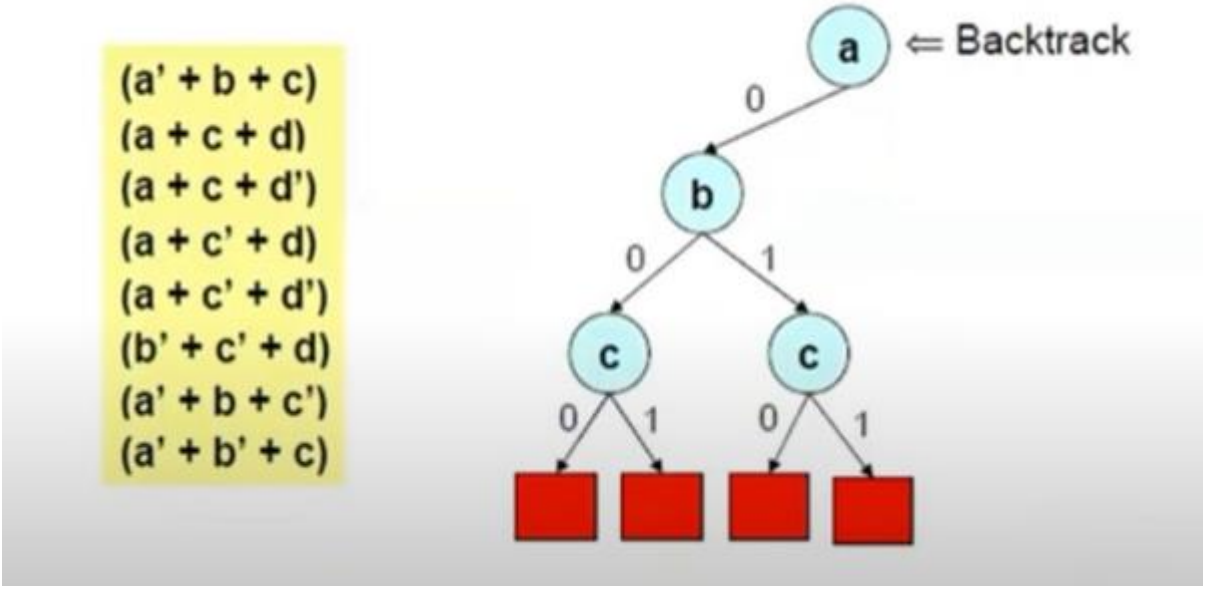
Bu b değişkeninde yaptığımız değer değişikliği de tekrardan başka bir çatışmaya yol açmaktadır.



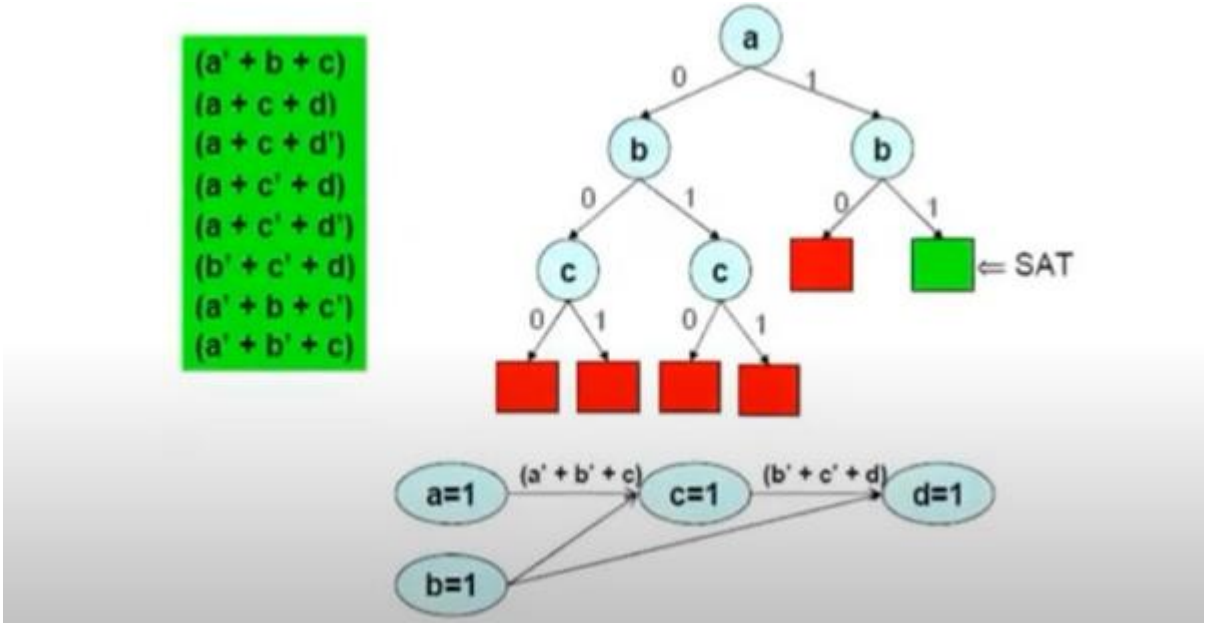
bu durumda bir üst düğüme değilde b değerinin 1 olduğu durumdaki c değerlerini ele alırız ve ilk olarak c değerini sırasıyla 0 ve 1 seçeriz fakat bu seçimlerde de cümlelerde çatışma meydana gelmiş oluyor.



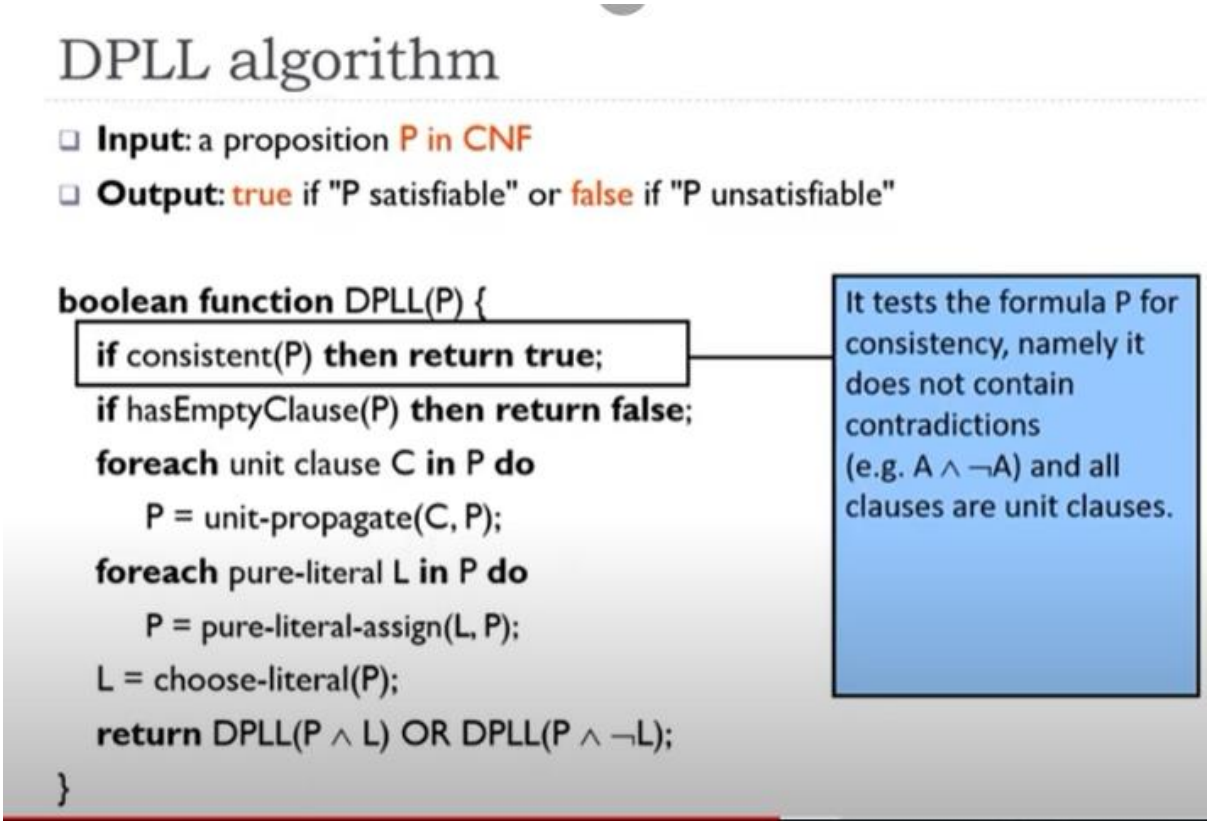
Bu çatışmalardan dolayı bir üst düğüme yani a değişkeninin olduğu düğüme geri dönüş sağlarız.



burada a değişkenini 1 olduğu durumu ele alarak ilerleme sağlarız. a'nın 1 olduğu durumda b değerine 0 vererek ilerlediğimiz taktirde yine bir çatışma ortaya çıkmış oluyor bu nedenle b değişkenine 1 vererek SAT sonucuna ulaşmış oluruz. Final grafiğimiz aşağıdaki gibi olacaktır.



yukarıda örnek üzerinden açıklamış olduğumuz algoritma kodu aşağıdaki şekildedir.



DPLL prosedürünün birkaç adımı, bir formülden çıkarılacak değişmez değerleri yeniden derler. Bu doğal görünse de, milyonlarca cümleyle uğraştığınızda mümkün olmaz. Bunun yerine, bir formüldeki değişkenlerin tümüne değil bazılarına doğruluk değerleri atayan kısmi değerlendirmeler kullanırız.

4.3. Ekler

DPLL kodu: https://github.com/sukrutrao/SAT-Solver-DPLL/blob/master/solver.cpp

5. ÇIKARIM VE SONUÇLAR

5.1. Çıkarımlar

Verilen problemler için Kaba kuvvet yöntemiyle(seri veya paralel olsun) yalnızca iki dosya çözülebilmektedir. Diğer dosyalar en az 50 değişkenli olduklarından kaba kuvvet ile çözülmeleri imkansızdır. Geliştirdiğimiz paralel kod ile ~20 değişkenli dosyalar için seri çözümün hesaplanma süresini ciddi miktarda düşürdük. Seri çözüm ile Pr1 900sn gibi bir süre alırken 4 threadli paralel çözüm ile 96sn'de çözüldü. Bunun gibi Pr7.txt nin hesaplama süresi de 60sn'den 5sn'e kadar düşürülmüştür. Diğer problemlerin çözümleri DPLL algoritması kullanılarak bulunmuştur. DPLL algoritmasıyla Pr1 ve Pr7 problemlerini çözdüğümüzde kaba kuvvet yöntemimizle bulduğumuz çözümlerle aynı olduğunu görüyoruz. Bu da kaba kuvvet algoritmamızın doğru çalıştığını göstermektedir. Yalnız kaba kuvvetin seri versiyonu ile Pr7 için farklı bir sonuç daha bulduk. Bu sonucu problemde yerine koyduğumuzda gerçekten de doğru olduğunu gördük. Böylece bir problemin birden fazla çözümü olabileceğini de öğrenmiş olduk.

DPLL algoritmasının bu denli iyi performans göstermesi gerçekten dikkat çekicidir. Bu, NP problemlerde semantik yaklaşımların ne kadar önemli olduğunu göstermektedir.

Sonuç tablosunda Pr1 ve Pr7 kaba kuvvet ile bulduğumuz çözümlerdir. Diğerleri DPLL ile bulunmuştur. Pr7 için kaba kuvvetin seri yöntemiyle bulduğumuz farklı çözüm:
-1 2 3 4 -5 -6 -7 8 9 10 11 -12 -13 14 15 -16 17 18 19 20.

5.2. Sonuç tablosu

No	**Pr1**	**Pr2**	**Pr3**	**Pr4**	**Pr5**	**Pr6**	**Pr7**	**Pr8**	**Pr9**	**Pr10**
1	**1**	1	UNSAT	1	1	1	**1**	1	UNSAT	-1
2	**2**	2		2	-2	-2	**-2**	2		2
3	**-3**	-3		3	3	3	**-3**	-3		-3
4	**4**	4		-4	-4	-4	**-4**	-4		4
5	**5**	-5		5	-5	-5	**-5**	-5		5
6	**6**	-6		-6	6	6	**6**	6		6
7	**7**	7		-7	7	7	**-7**	-7		7
8	**-8**	8		-8	-8	-8	**-8**	-8		8
9	**9**	-9		-9	-9	-9	**9**	9		9
10	**10**	10		10	-10	-10	**-10**	-10		10
11	**-11**	11		11	-11	-11	**-11**	-11		-11
12	**12**	12		12	-12	-12	**-12**	-12		12

13	**-13**	-13		-13	-13	-13	**13**	13		-13
14	**14**	14		14	14	14	**14**	14		14
15	**-15**	-15		15	-15	-15	**15**	15		15
16	**16**	-16		16	16	16	**-16**	-16		-16
17	**-17**	17		17	-17	-17	**17**	17		-17
18	**-18**	18		-18	-18	-18	**-18**	-18		-18
19	**-19**	-19		19	19	19	**-19**	-19		19
20	**-20**	20		-20	20	20	**20**	20		20