# 3-SAT Solver proje raporu

## Yapanlar:


- Zehra Sueda Çiğdem 383217
- Yaren Zelal Tosu 383256
- Yusuf Nadaroğlu 365242
- İbrahim sincap 365272
- Erdem özen 390622


31.05.2022

# Proje Açıklaması

## Problem tanımı

3-SAT solver projesi, bir SAT algoritmasının alt işlemidir. SAT işleminde, bir dizi girdi dizisine karşılık gelen değişken atamalarından oluşan fonksiyonu tatmin (SATisfy) edebilen bir girdi durumunun mevcut olup olmadığı bulunur. Bu fonksiyonlara boole formülü denir.

Bir boole formülü, mantıksal olarak VE ilişkisi içinde bir veya daha fazla yan tümcesi varsa, bir bağlaçlı normal formda (conjunctive normal form) olduğu söylenir. Ve her cümle, birbiriyle VEYA ilişkisi içinde 3 değişmezden oluşan bir koleksiyondur (iki olası duruma/değere sahip olabilen herhangi bir mantıksal değişken, yani True(Doğru) veya False(Yalnış)).

Bir boole formülünün, bazı geçerli değişmez değer atamaları için formülün sonucunun True olması durumunda tatmin edici olduğu söylenir.

Örnek Madde :
(A v B v C)
burada A, B, C değişmez değerlerdir ve v VEYA sembolünü temsil eder

Örnek Formül:
(A v B v C) ^ (~A v D v C) ^ (E v F v ~G) ^ (A v B v ~C)
burada ^ VE sembolünü ve ~ mantıksal DEĞİL işlemini temsil eder

~ operatörü, değişkenlerin değerini tersine çevirir; örnek, True olarak False ve False öğesini True olarak değiştirir.

Tatmin Edilebilir (SATisfiable) Formül:
(A v B v C) ^ (D v B v ~E) ^ (D v ~B v C)

Atama :

B - Doğru (Madde 1 ve 2'yi Doğru yapar)
D - Doğru (3. maddeyi Doğru yapar)
Diğer tüm değişmez değerler atanamaz veya false olarak ayarlanabilir ve formül yine de True olacaktır.

Bu sorunu çözmek için ileri seviye algoritmalar geliştirilmiştir. Bunlardan biri, DPLL algoritması, sabit değerlerin tüm olası değerleri için ağaç tabanlı bir arama gerçekleştirmeye çalışır, ancak arama alanı, aşağıdaki adımların yardımıyla önceden azaltılır (budanır):
(1) birim yayılım - yalnızca bir atanıyor, hazır bilgi içeren tümceleri tanımlanıyor ve yan tümceyi DOĞRU yapmak için bir değer atanıyor
(2) saf değişmez eleme - formül boyunca aynı polariteye sahip değişmezleri belirlemek ve tüm bu maddeleri DOĞRU yapmak için değerler atamak

## Projenin tanımı

Bu projenin amacı,

1) SAT işlemini gerçekleştiren algoritma yazarak programlama bilgilerimizi geliştirmek,

2) Bunu paralelleştirerek paralel programlama bilgilerimizi geliştirmek,

3) DPLL Algoritmasını implemente ederek ileri seviye algoritmalara giriş yapmak.

# Projenin yapımı

## Planlama aşaması

Açıklama PDF`ini inceledikten sonra hepimiz için daha tanıdık olması sebebiyle projeyi C++ dilinde yazmaya karar verdik. İlk proje planı şu şekildeydi:

1) Algoritmayı iyice anlamak

Bu projeyi iyi bir şekilde yazmak istiyorsak ne yapmamız gerektiğini iyice anlamamız gerekiyordu. Bu yüzden internet üzerinde farklı dillerde birçok kaynaktan sorunun anlatımlarını araştırdık.

2) Programın yazılması için bir yazılım ortamı oluşturmak

3) Uygulamanın (SAT Problemi) basit girdilerle çalışan ilk versiyonunu yazmak

4) Çalıştırdıktan sonra daha büyük girdilerle test etmek, sorun çıkarsa düzeltmek

5) SAT Problemi çözüldükten sonra hangi durumun SAT çıktısı verdiğini ekrana yazdıran algoritmayı yazmak

6) Bu algoritmayı test ettikten sonra paralelleştirmek

7) Basit paralelleştirme çalıştıktan sonra benchmarking yapmak

8) DPLL Algoritmasını entegre etmek

9) Eski algoritmayla karşılaştırmak

# Adımların gerçekleştirilmesi

## Ve yaşadığımız sorunlar

### 1: Algoritmayı iyice anlamak

İlk adımda her birimizin farklı öğrenme teknikleri ve kaynakları olduğu için sorunu kendi kendimize araştırmaya karar verdik. Öğrenmek için kullandığımız kaynaklar içerisinde ekteki videolar ve makaleler var. UE2`de bahsi geçen dersi de izlemek istedik, ancak kaydını bulamadık.

### 2: Programın yazılması için bir yazılım ortamı oluşturmak

#### 2.1: Derleyici

Çoğunluk oyundan sonra Linux işletim sistemi kullanmanın bu proje için daha uygun olduğuna karar verdik. Programlama dili olaraksa C++ kullanmayı seçtik.

Linuxta derleyici olarak GNU C Compiler setinden g++ uygulamasını kullanıyoruz. g++ kullanarak uygulama derlemek için default komut aşağıdakı şekildedir:

```
g++ main.cpp -o main
```

Burda -o main, çıktı dosyasını, main.cpp ise derlenecek olan kod dosyasını temsil eder. Aynı zamanda g++ derlemesine OpenMP kütüphanesi desteği katmak için -fopenmp, daha optimize kod üretmek içinse -Ofast bayraklarını kullanabiliriz:

```
g++ main.cpp -fopenmp -Ofast -o main
```

Çıkan kodu çalıştırmak içinse ./main yazarak çalıştırabiliriz.

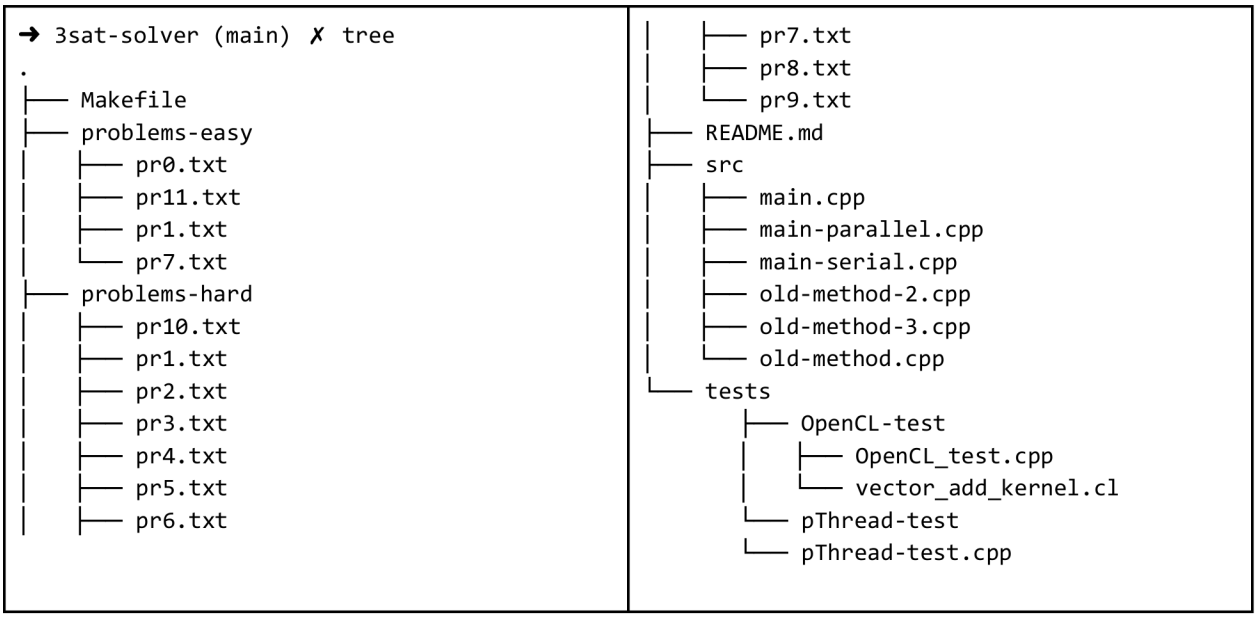
Yazdığımız kodun proje altyapısının düzenli ve kolay anlaşılır olmasını isteriz, o yüzden projedeki dosya ve klasör düzeni önemlidir. Bunun için bir git reposu açtık ve ismini 3sat-solver olarak belirledik. Git, bir versiyon kontrol sistemi. Peki versiyon kontrol sistemi nedir? Basitçe, bir dosyanız var ve bu dosyanın oluşturulma tarihinden başlayarak istediğiniz tüm durumlarını takip edebildiğiniz bir yapı tasarlıyorsunuz. Bu yapının sorun çıkarıp çıkarmaması tamamen bizim elimizde. Ne kadar iyi tasarlarsak ne kadar çok commit atıp geri dönebileceğimiz noktalar oluşturursak, yönetimi kolay bir yapı tasarlamış oluruz. Kısaca yazılım projemizin tüm anlarını kayıt altına alıyormuşuz gibi düşünebiliriz.

Projede aşağıdaki gibi bir yapı kullanmaya karar verdik:

```
➜ 3sat-solver (main) ✗ ls
Makefile  problems-easy  problems-hard  README.md  src
tests
```

`problems-easy` ve `problems-hard`, respektif olarak içinde kolay ve zor DİMACS CNF SAT problem dosyalarını bulunduruyor. src ise kaynak dosyalarının bulunduğu dosya. tests klasörü ise o bilgisayarda OpenCL ve ya pThread çalıştırabiliyor muyuz diye basit uygulamaların olduğu bir klasör.

Bir sonrakı sayfada projenin son halinin ağaç grafı şeklinde halini görebilirsiniz.




### 2.3 Makefile

Bu tarz projelerde derlenme aşamasını kolaylaştırmak için Makefile kullanılıyor. Makefile, make uygulaması tarafından kullanılan bir configurasyon dosyasıdır. make ise, projenin derlenmesini, çalıştırılmasını ve s. sağlayan bir çok komut kümesini tek bir make komutu altına almanızı sağlayan bir uygulamadır. Bizim bu projede bir Makefile`a ihtiyacımız yoktu, ama hepimiz için denemeler yaparken uygulamayı tekrar derleyip çalıştırmaktansa tek bir make yazmak daha kolay olur diye karar verdik:

```
➜ 3sat-solver (main) ✗ cat Makefile
CC=g++

3sat-solver: src/main.cpp
    $(CC) -pthread -o 3sat-solver src/main.cpp
    ./3sat-solver
```

Bu Makefile, uygulamayı g++ kullanarak derliyor, 3sat-solver diye bir uygulama dosyası oluşturuyor ve onu çalıştırıyor. Bir diğer güzel yanı, kaynak dosyasını takip ediyor ve eğer değişmemişse, tekrardan derlemekle uğraşmıyor.

3, 4, 5: Uygulamanın (SAT Problemi) basit girdilerle çalışan ilk versiyonunu yazmak

Çalışmayı ayrı ayrı yapmak yerine, birlikte gelerek group programming şeklinde yapmaya karar verdik.
İlk olarak dosyayı okuyan algoritmayı yazmamız gerekti. Bunun için, bir `std::filesystem::directory_iterator` kullanarak bir dizindeki her dosya için ayrı ayrı çalışmasını sağladık:

```
for( const auto & entry :
std::filesystem::directory_iterator( problems_path ) ){
     cout << "_____________________________________" << endl;
     cout << entry.path() << ": " << endl;
     current_file.open(entry.path());
     if(!current_file.is_open()) {
          perror("Error open");
          exit(EXIT_FAILURE);
     }
          /* her dosya için çalışan kodlar */
}
```

Sonrasında dosyayı satır satır okumak için getline() methodunu kullandık. Bu metodu uygulamanın ana döngüsüne condition (şart) olarak vererek dosyanın okunması bittiğinde döngünün durmasını sağladık. getline() fonksiyonunun bize verdiği satır string`ini ise bir stringstream`a dönüştürmek içerisinden ayrı ayrı kelimeleri alabilmemizi sağladı:

```
while(getline(current_file, line)) {
      stringstream ss(line);
      string word;
      ss >> word;
      if(word == "c")
           continue;
      else if (word == "p")
      {
           ss >> word; // get the word "cnf"
           ss >> word; // get variable count
           int variable_count = stoi(word);
           ss >> word; // get line count
           int line_count = stoi(word);
……
     }
}
```

Bundan sonraki kodlar, tek tek satırları okuyor ve girdilere bakıyor. src klasörü içinde birden fazla dosya var. Ana kodumuz olan main.cpp dosyası yanında, old-method-1, 2, 3 adında dosyalar da var. Bunlar öncesinde yazıp da ya algoritmasını yanlış anladığımız için yanlış yazdığımız, ya da yazmanın ortasında nasıl çalışması gerektiğini değiştirmeye karar verdiğimiz adımlar. Bunlara bakmak isteyebilirsiniz, nasıl ilerlediğimize size bir insight verecektir.

Uzun denemelerden, başarısızlıklardan, daha fazla denemelerden ve çıktılarımızı bu projeyi yapan başka arkadaşlarla karşılaştırdıktan sonra, SAT problemini sağlayan çıktıyı veren algoritmayı yazmayı başardık. Uygulama içerisinde belirtilen klasördeki dosyalar için tüm CNF girdilerini okuyor, her birisini bir dizide tutuyor. Bu problem 3CNF olduğu için, her bir satır maksimum 3 girdiden oluşabilir, bu da demek oluyor ki bizim satırları
`int line_states[line_count][3]` gibi bir array`de tutabiliriz. Sonra bu arrayi o an denediğimiz girdi kombinasyonuna göre deniyoruz. Eğer SAT değişkeni 0a dönmezse, uygulamanın sonunda o girdi kombinasyonunun SATı sağladığını biliyoruz.

### 6: Bu algoritmayı test ettikten sonra paralelleştirmek

Bu kodu paralelleştirmek için elimizde birkaç seçenek vardı:

1. Java/Python/Diğer %25
2. Web teknolojileri: PHP/Django %35
3. OpenMP/pThread %30
4. MPI/CUDA %35
5. OpenCL %40
6. Hadoop/Apache Spark %45
7. Android/ %45

Biz C++ kullandığımız için 3. Seçeneğin, yani OpenMP ve ya pThread kullanmanın daha uygun olacağına karar verdik.

#### 6.1: Paralelleştirme mantığı

Uygulamanın en çok zaman alan kısmı 2^n değişken durumuna göre bizim girdi değişkenlerini deneyen kısım. Bunu paralelleştirmek için de aklımıza ilk olarak en düz method geldi: bu işlemi bir fonksiyona atamak ve o fonksiyonu bir kütüphane yardımıyla birden fazla kere aynı anda multithreaded olarak çalıştırmak. Bu yüzen ana for loopunu checkLinesForSAT diye bir fonksiyona koyduk.

#### 6.2 Paralelleştirme aşaması

pThread kullanarak multithreading yapmak için pthread_create(..) fonksiyonu kullanılıyor. Bu fonksiyonu standart yöntemlerle kullanmayı denedik, ancak karşımıza bir sorun çıktı: pThread fonksiyonu, son argüman olarak çalıştırılacak fonksiyona gönderilecek olan argümanları alıyor, ama sadece tek bir argüman alabiliyor. Bizim tasarladığımız fonksiyon ise birden fazla argüman gerektiriyor:

```
void checkLinesForSAT(bool *variables_array,
int line_states[][3], int line_count, int variable_count)
```

Bunu çözmek için argümanlarımızı bir struct içerisinde tutup struct veriyapısını tek argüman olarak gönderebiliriz. Ancak bizim arraylarımız 2 boyutlu olduğundan, pointer atamada ve s. Sıkıntı yaşadık, durmadan hatalar aldık ve ya uygulama doğru çalışmadı. Uzun bir süre C++`ın pointerları, memory allocationlarıyla uğraştık ama çözemedik.

Sonra başka bir çözüm denemek istedik, C++`ın std::thread yapıları. thread(..) fonksiyonuyla, aday fonksiyonun aldığı argümanları olduğu gibi gönderebiliyorsun, çünki thread() fonksiyonu template olarak çalışıyor. Ama malesef bizim kullandığımız veri yapıları bu ortamda da sıkıntı çıkardılar, bunların da üzerine çalıştık ama çözemedik.

En son denediğimiz metod ise for loopu OpenMP kullanarak paralelleştirmekdi. Bu da çalışmadı, ne kadar private, shared değişkenleri tanımladıysak, ne kadar inline for, nest break flags gibi mantıklar kullanmaya çalışsak da asla doğru çalışmadı.

Sonuç olarak paralelleştirmeyi yapamadık, ama yapamamızın sebebinin Paralel Programlama mantığını bilmememize değil, C++ kaprizlerine bağlı olduğunu temenni ediyoruz. Zamanımız yetişmediği için daha fazla deneme yapamadık, ama src klasörü içindeki kodlarda diğer denemelerimizi de görebilirsiniz. Son sonuçları aşağıda görebilirsiniz. Başta sadece SAT sonuç var mı diye bulan algoritmamızla aşağıdaki sonuçları bulmuştuk, ama sadece pr1 ve pr7 `lerin SAT durumu çıkaran tablolarını bula bildik. Diğer durumlarda bilgisayarın hızıyla işlemler çok uzun sürüyordu.

UNSAT: pr3, pr9
SAT: Diğer hepsi.

```
➜ 3sat-solver (main) ✗ time make -B
g++ -pthread -o 3sat-solver src/main.cpp
./3sat-solver

_____________________________________
"problems-easy/pr0.txt":
vars and lines: 2 4
Variables array state: 11

_____________________________________
"problems-easy/pr7.txt":
vars and lines: 20 91
Variables array state: 01110001111001101111
Variables array state: 10000100000011101001
Variables array state: 10000100100001101001
Variables array state: 10000100100011101001
Variables array state: 10010000010011101001
Variables array state: 10010001010011101001
Variables array state: 10010100000011101001
Variables array state: 10010100010011101001

_____________________________________
"problems-easy/pr1.txt":
vars and lines: 22 73
Variables array state: 11011110110101010000010

_____________________________________
"problems-easy/pr11.txt":
vars and lines: 16 18
Variables array state: 0110111110011001
Variables array state: 0110111110111001
Variables array state: 0110111111011001
Variables array state: 1010111110011001
Variables array state: 1010111110111001
Variables array state: 1010111111011001
Variables array state: 1110111110011001
Variables array state: 1110111110111001
Variables array state: 1110111111011001
make -B  152.27s user 1.21s system 99% cpu 2:33.61 total
➜ 3sat-solver (main) ✗
```

Not: Yukardakı çıktıda Variables array state: dediği her bir dosya için SAT durumunu üreten çıktılar. Küçük dosyalar için elle kontrol ettik ve doğru olduğunu düşünüyoruz.

Not 2: pr0 kendi elimizle yazdığımız test dosyası, pr11 internetten bulduğumuz bir dosya.

# Kaynaklar


- https://medium.com/swlh/introduction-to-the-openmp-with-c-and-some-integrals-approximation-a7f03e9ebb65
- https://www.fayewilliams.com/2015/03/31/pthreads-tutorial-linux-cpp/
- https://www-cs-faculty.stanford.edu/~knuth/programs/sat0.w
- https://math.stackexchange.com/questions/86210/what-is-the-3-sat-problem
- https://en.wikipedia.org/wiki/Boolean_satisfiability_problem#/
- https://www.cse.cuhk.edu.hk/~ericlo/teaching/os/lab/9-PThread/Pass.html
- https://www.generacodice.com/en/articolo/4857376/c-11-std-thread-giving-error-no-matching-function-to-call-std-thread-thread
- https://en.wikipedia.org/wiki/Pthreads
- https://stackoverflow.com/questions/33558556/sending-multiple-arguments-to-pthread-create
- http://www.thinkingparallel.com/2007/06/29/breaking-out-of-loops-in-openmp/
- https://newbedev.com/how-does-openmp-handle-nested-loops