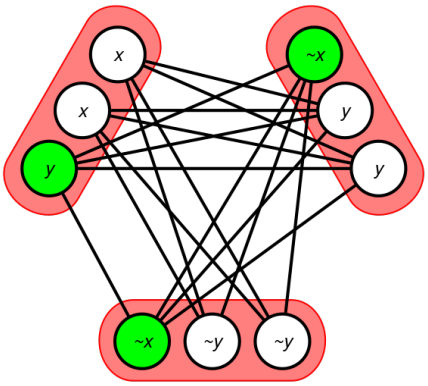
Data Structures and Algorithms : Maximum Satisfaction
Homework 2

Julien Baudru

December 10, 2021

Introduction

The goal of this project is to develop a Las Vegas type algorithm that maximizes the number of satisfied clauses in a boolean formula of conjunctive nominal form (CNF).



In order to give an example, the picture above illustrates an example of a 3-SAT formula reduced to the CLIQUE problem. The corresponding formula in this picture is the following :

$$\phi = (y \vee x \vee x) \wedge (\neg x \vee y \vee y) \wedge (\neg x \vee \neg y \vee \neg y) \tag{1}$$

In these pages, we will deal with the specific case of boolean formula of conjunctive normal in which every clause has exactly three distinct literals, and no clause simultaneously contains a literal and its complements, this particular problem is called 3-SAT.

Requierements

This project was written in **Julia 1.7.0**, the environment and the Julia language can be installed via this link. Note that this project uses the standard packages as well as the ***Plots*** package which you will need to install as follows :

```
julia
using Pkg
Pkg.add("Plots")
```

To start the execution of the program you will just have to enter the following command in your terminal :

```
julia main.jl
```

The entire source code of this project is available via this GitHub repository.

Dataset

This algorithm has been tested on instances of the 3-SAT problem available via this link. The dataset on which this algorithm was tested consists of 1000 files each containing different instances of the 3-SAT problem encoded in DIMACS cnf format. Each of these files contains formula with 100 variables and 91 clauses, they differ by the order of the variables present in the clauses.

This dataset was chosen for practical reasons of execution time. Indeed, the family of Las Vegas type algorithms provides an guarantee on the fact of having a result but none on the time taken to have this result. This is why I have chosen to limit my experiments to a small number of variables and clauses.

Algortihm

Below you will find the pseudo-code explaining the functioning of the main loop used to produce the two graphs of the experimental results section :

```
for each file in folder:
    formula = ParseData(file)
    res = LasVegas(formula)
```

The global behavior of the algorithm to solve the given problem is explained in pseudo-code below :

```
function LasVegasRunner(formula){
    nb_try = 1
    success = LasVegas(formula)
    while sucess < 7/8:
        success = LasVegas(formula)
        nb_try += 1
    return success
}
```

The above pseudo-code calls an instance of the Las Vegas algorithm as many times as necessary to obtain a result in which 7/8 of the clauses are satisfied.

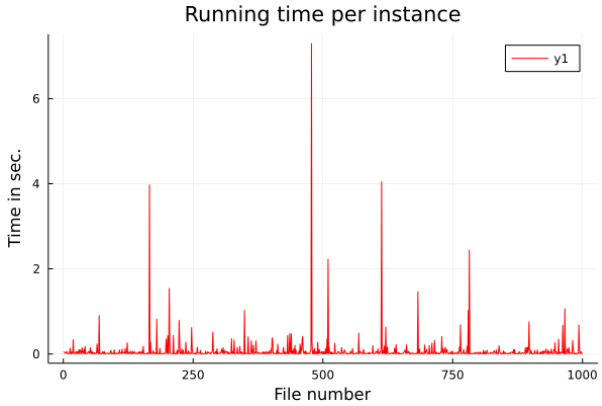
```
function LasVegas(formula){
    variables = generateRandomTruthValue()
    num_clauses_satisfied = 0
    for each clause in formula:
        clauseIsSatisfied = checkClause(clause, variables)
        if (clauseIsStatified){
            num_clauses_satisfied += 1
        }
    return num_clauses_satisfied / num_clauses
}
```

Above, the codes of the $generateRandomTruthValue()$ and $checkClause(clause, variables)$ functions are not detailed for the sake of space and clarity.

Experimental results

The graph below shows the number of satisfied clauses return by the algorithm for each instance (file) of the problem. We notice that this number is always strictly higher than 79 in this case because this algorithm must satisfy at least $\frac{7k}{8}$ of the clauses, here k being 91, that gives us that the algorithm must satisfy at least 79,625 of the clauses.

For each instance, the graph below shows the time taken by the algorithm to satisfy at least $\frac{7}{8}$ of the clauses. We notice that for some instances of the problem, depending on the position of the variables in the clauses of the initial formula, the execution time will be particularly high compared to the average which is 0.08 seconds.



Moreover we notice that all the instances of the 3-SAT problem have obtained a solution which satisfies the lemma that says that there always exists an assignment of truth which satisfies at least $\frac{7k}{8}$ of the clauses.

Finally, on average the algorithm implemented in this project obtains an execution time of 0.0818 seconds, a number of trials of 2060.498 and the number of satisfied clauses is 80.342.

Analysis

Given a 3-SAT formula with k clauses, the expected number of clauses satisfied by a random assignement of the truth values to their variables is $\frac{7k}{8}$. Indeed, if we put X_j the random variable equal to 1 if the clause C_j is satisfied, 0 otherwise. Then we can put $Pr[X_j = 1] = 1 - (\frac{1}{2})^3 = 0.875$, the probability that the clause C_j is satisfied. Experimentally, we can see that the average over the 1000 files gives a value of 0.8841259 for the probability that a clause is satisfied which is close enough to the expected value to consider this result as correct.