CS 463: SAT Assignment

Derek Jones

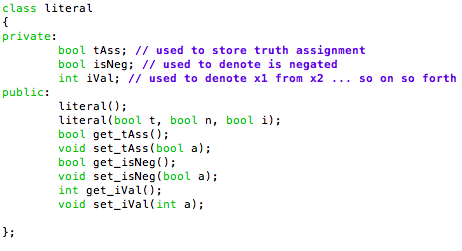
10/12/2015

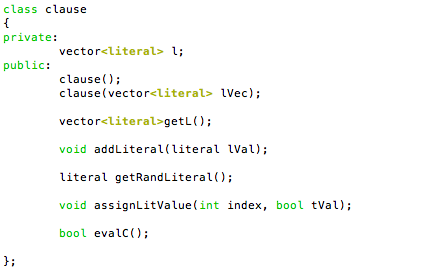
**Problem Statement**

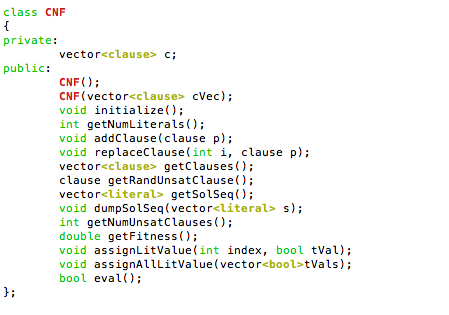
In this programming assignment, the goal was to program a set of algorithms that were able to solve CNF logical formulas. This problem is classified as NP-Hard as there may not be in some cases, a satisfying set of variable values (0 or 1) to satisfy all clauses in a given CNF, and thus a search for a solution could very well venture into exponential time. For this assignment, the instructor (Dr. Judy Goldsmith) requested that three out of four possible algorithms be implemented by the student, and to collect data on each of the runs on the provided testing files. In the case of this implementation of the assignment, each algorithm incorporated some degree of randomness, thus data on each algorithm was collected for 10 runs each.

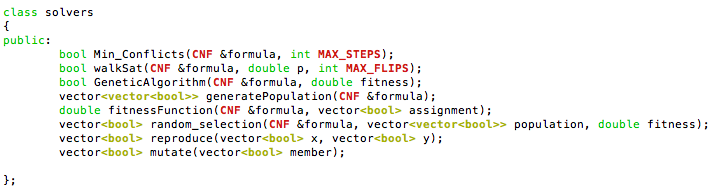
**Description of Data Structure**

The general idea for the data structure for this assignment was derived from the class discussion board, however the code for the data structures is completely original work by the student. Three data structures were created to store the contents of a given CNF.









The data structures are related but do not inherit from one another as neither a clause or a literal are necessarily a CNF, but literals make up the clauses that make up a given CNF, so it is important that these data structures are able to work together to form the foundation of this programming assignment. A solvers class was created to store the logic for the solvers.

**Description of Algorithms**

The implementations of the algorithms listed differ slightly from the pseudo-code given by the author of the text [1].

WalkSAT

The following psuedocode came from Artificial Intelligence, A Modern Approach[1]

**Function**: WalkSAT(*clauses, p , max\_flips*) returns a satisfying model or failure

{

**inputs:** *clauses*, a set of clauses in propositional logic

*p*, the probability of choosing to do a “random walk” move, typically around 0.5

*max\_flips*, number of flips allowed before giving up

*model* <- a random assignment of true/false to the symbols in clauses

for i = 1 to *max\_flips* do

**if** *model* satisfies *clauses* **then return** *model*

*clause* <- a randomly selected clause from *clauses* that is false in *model*

**with probability** *p* flip the value in *model* of a randomly selected symbol from *clause*

**else** flip whichever symbol in *clause* maximizes the number of satisfied clauses

**return** *failure*

}

In my implantation, I return a boolean for each possible state (Satisfiable assignment = true, Unsatisfiable assignment = false). In the else loop I call the MIN-Conflicts local search algorithm as the WalkSAT algorithm performs a local search in this stage and MIN-Conflicts accomplishes just that. I also have set the probability p = 0.5.

Min-Conflicts (Local Search)

The following psuedocode came from Artificial Intelligence, A Modern Approach

**Function:** MIN-CONFLICTS(*csp,max\_steps*) returns a solution or failure

{

**inputs:** *csp*, a constraint satisfaction problem

*max\_steps*, the number of steps allowed before giving up

*current* <- an initial complete assignment for *csp*

**for** i = 1 to *max\_steps* do

**if** *current* is a solution for *csp* **then return** *current*

*var* <- a randomly chosen conflicted variable from *csp*.VARIABLES

*value* <- the value *v* for *var* that minimizes CONFLICTS(*var,v,current,csp*)

set *var* = value in *current*

**return** *failure*

}

My implementation of Min-Conflicts follows very closely the pseudo-code given by the author [1].

Genetic Algorithm

The following pseudocode came from Artificial Intelligence, A Modern Approach [1].

**Function:** GENETIC-ALGORITHM(*population*,FITNESS-FN) returns an individual

{

**inputs:** *population*, a set of individuals

FITNESS-FN, a function that measures the fitness of an individual

**repeat**

*new\_population* <- empty set

**for** i = 1 to SIZE(*population*) **do**

*x* <- RANDOM-SELECTION(*population*,FITNESS-FN)

*y* <- RANDOM-SELECTION(*population*,FITNESS-FN)

child <- REPRODUCE(*x , y*)

**if** (small randomly probability) **then** *child* <- MUTATE(*child*)

add *child* to *new\_population*

*population* <- *new\_population*

**until** some individual is fit enough, or enough time has elapsed

**return** the best individual in *population*, according to FITNESS-FN

}

**Function:** REPRODUCE(x,y) **returns** an individual

{

**inputs:** *x, y*, parent individuals

*n* <- LENGTH(*x*); c <- random number from 1 to *n*

**return** APPEND(SUBSTRING(*x*, 1, *c*),SUBSTRING(*y*, c+1, *n*))

}

My implementation of the genetic algorithm follows very closely the pseudo-code given by the author. I add implementation for RANDOM\_SELECTION not given by the book, as well as a method to generate an initial population of truth assignments. I also add implementation for MUTATE not given by the book.

**Note:** While not an algorithm specified for implantation, the code for the parser was developed from pseudo-code given by a Dr. Tom Allen from a previous semester that A.I. was offered. The code written is original work, only the general structure of what needed to be done (and how) was taken from this source [2].

**DATA from testing**

The testing files were renamed (test[i], 0<=i<=10) in order to make it easier to pass the files to the program.

The fitness was calculated as number of satisfied clauses divided by total number of clauses. A fitness measure closer to 1 means that the program has found an assignment of variables to satisfy the given CNF.

Data sheets are attached to back of the write-up for your reading convenience.

**Statement of Educational Value**

This assignment was very difficult to approach at first, however it became very exciting to work on the assignment after the initial learning curve of working with these types of algorithms. I feel that this assignment helped me grow as a computer scientist as I have since developed a much deeper interest in the field of AI. I took a risk in this assignment by attempting to implement the genetic algorithm. At first sight I felt overwhelmed by the algorithm and I didn’t try to implement it because the pseudo-code from the book wasn’t specified for a SAT implementation. However, after being granted extra time on the assignment, in the wee hours of Thursday morning (4am to approximately 6am) I felt a spark of inspiration and implemented most of the genetic algorithm. I feel the most important things that I have learned from this assignment are better time management skills as well as allowing oneself to take risks and try seemingly difficult things. I hope to continue learning from future assignments in order to apply my skills and curiosity to the field of AI.

**Reference**

[1]: Russell, S. "Artificial intelligence: A modern approach author: Stuart Russell, Peter Norvig, publisher: Prentice hall pa." (2009).

[2]: Allen, Tom. "CS 463, Spring 2014, Assignment 3." *CS 463G Assignment 3*. 2014. Web. 12 Oct. 2015.