Relazione Elaborato

WalkSAT e problemi SAT casuali

Studente

- Nome: Alessio Falai
- Matricola: 6134275
- E-mail: alessio.falai@stud.unifi.it

Prima parte

Nella prima parte di questo elaborato si scrive un programma (in un linguaggio di programmazione a scelta) che genera formule k - CNF in logica proposizionale modo uniformemente casuale, come esposto in classe e descritto in R&N 2009 §7.6.3. In particolare, dato il numero di simboli proposizionali, n, il numero di clausole, m, e la lunghezza di clausola, k, si generano formula tali che: (1) ogni clausola ha esattamente k letterali (ossia si rifiuta una clausola se contiene letterali duplicati), (2) ogni clausola è non banale (cioè non è una tautologia), e (3) tutte le clausole sono distinte. I letterali devono essere estratti da una distribuzione uniforme sui 2n letterali possibili.

Per l'implementazione di questa prima parte è stata scritta una classe $KCNF$, contenuta nel file $kcnf.py$. La classe prevede l'utilizzo di due strutture dati principali, $literals$ e $clauses$, istanziate utilizzando le collezioni $list$ e set, rispettivamente. Nel costruttore $\_\_init\_\_(k, m, n)$ viene specificato il seguente ordine di esecuzione:

1. $generate\_literals(n)$: La funzione prende in input il parametro n che rappresenta il numero di simboli proposizionali della formula da generare e restituisce la lista formata dai $2n$ possibili letterali, dove con letterale si intende un simbolo proposizionale o la sua negazione. Ciascun simbolo è rappresentato da una o più lettere maiuscole, generate intuitivamente in questo modo: $A, B, ..., Z, AA, AB, ..., AZ, BA, BB, ..., BZ, ..., ZZ, AAA, ...$, e così via. La funzione che si occupa della generazione dei simboli è $next\_symbol(current)$ che, dato il simbolo attuale in input, restituisce il prossimo simbolo, come illustrato sopra.
2. $generate\_clauses(k, m, n)$: La funzione prende in input i parametri k, m, n e restituisce una formula casuale $k-CNF(m, n)$ sotto forma di insieme di clausole, dove con clausola si intende una disgiunzione di letterali e con insieme di clausole una congiunzione tra clausole. La funzione è stata implementata con due cicli while annidati, in cui il ciclo più esterno controlla che il numero di clausole dell'insieme sia esattamente m e quello più interno

controlla che la lunghezza di ogni clausola sia proprio n e che tale clausola non sia una tautologia. Nel ciclo più interno viene estratto in modo casuale, senza rimessa, un campione di k letterali, da una distribuzione uniforme sui $2n$ letterali possibili (Funzione $numpy.random.choice$). Dopodichè, viene creata una clausola, implementata con la struttura $frozenset$, contenente quei k letterali estratti. A questo punto, viene controllata la lunghezza della clausola e la sua validità. Nel caso in cui la clausola abbia cardinalità strettamente minore di k oppure[1] la clausola sia una tautologia, questa viene scartata. Il procedimento viene ripetuto finchè l'insieme di clausole non ne contiene esattamente m. L'utilizzo delle strutture dati di tipo set e $frozenset$ consente di automatizzare il processo di distinzione dei letterali presenti in una clausola. Nella funzione $generate\_clauses$ è inoltre presente un quarto parametro di input max_time, il cui valore di default è 2000 millisecondi. Tale parametro viene utilizzato per definire un limite superiore nel tempo che la funzione può impiegare a generare l'insieme di clausole e risulta utile, per esempio, quando è molto probabile che vengano generate delle tautologie, cioè quando n è piccolo.

Seconda parte

Nella seconda parte, si implementa l'algoritmo WalKSAT esposto in classe e descritto in R&N 2009 §7.6.2 e lo si applica a diversi problemi SAT generati casualmente dal programma preparato nella prima parte dell'elaborato. Si visualizzino i risultati come nella Figura 7.19 del libro di testo (ignorando il caso della procedura DPLL per la Figura 7.19b).

Per l'implementazione di questa seconda parte è stata scritta una classe $WalkSAT$, contenuta nel file $walksat.py$. La classe prevede l'utilizzo di due strutture dati principali, $clauses$ e $model$, istanziate utilizzando le collezioni set e $dict$, rispettivamente. Nel costruttore $\_\_init\_\_(clauses)$ viene specificato il seguente ordine di esecuzione:

1. $get\_model()$: La funzione si occupa di estrarre tutti i simboli dall'insieme di clausole $clauses$ passato al costruttore e di popolare la struttura dati $model$ con tali simboli, come chiavi, e con una scelta casuale (Funzione $numpy.random.randint$) tra i valori $True/False$, come valori.

Dopo aver creato un'istanza della classe $WalkSAT$ è possibile richiamare su tale oggetto la seguente funzione:

- $solve()$: La funzione prende in input i parametri opzionali max_flips e p, i cui valori di default sono 1000 e 0.5, rispettivamente. Se l'algoritmo restituisce un modello per la formula in esame allora tale formula è soddisfacibile, mentre se

[1] In questo caso, la disgiunzione viene utilizzata in modo non esclusivo.

ritorna $FAILURE$ non si può sapere se la formula è insoddisfacibile oppure[2] se l'algoritmo ha bisogno di più tempo per processarne un modello. La funzione è implementata con un ciclo for, che permette di iterare max_flips volte. Possiamo considerare l'algoritmo come una sorta di ricerca locale di tipo $hill\ climbing$, la cui funzione di valutazione $unsatisfied\_clauses()$ conta il numero di clausole non soddisfatte. A ogni iterazione, l'algoritmo seleziona una clausola non soddisfatta in modo uniformemente casuale (Funzione $get\_random\_clause()$) e cambia il valore di verità di uno dei suoi simboli. Ci sono due modi per scegliere tale simbolo. Il primo, effettuato con probabilità p, sceglie un simbolo in modo uniformemente casuale fra tutti i simboli presenti nella clausola (Funzione $get\_random\_symbol(clause)$). Il secondo, effettuato con probabilità $1-p$, sceglie un simbolo che minimizza il numero di clausole non soddisfatte nel nuovo stato (Funzione $get\_maximizing\_symbol(clause)$).

Il file $randomkcnfsolver.py$ contiene le funzioni utilizzate per applicare ciò che è stato implementato nelle classi sopra-descritte. In particolare:

1. $model(k, m, n, max\_flips, p)$: La funzione prende in input i parametri necessari a creare in modo casuale una singola formula $k-CNF(m,n)$, utilizzando un'istanza della classe $KCNF$ sopra-descritta, e restituisce un modello che la soddisfa oppure $FAILURE$, in base ai valori ritornati dalla funzione $solve()$ della classe $WalkSAT$.

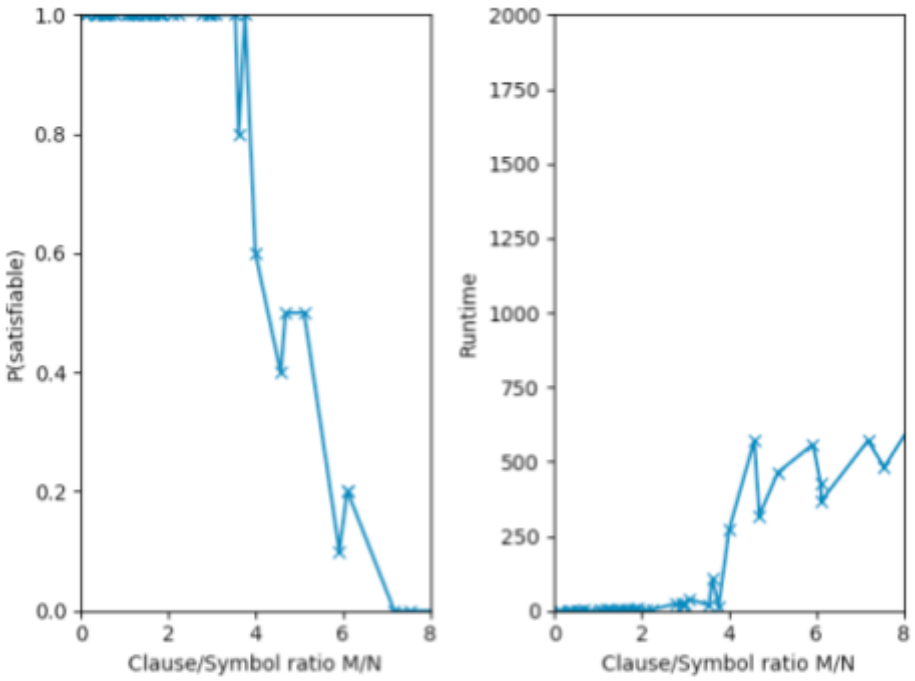
```
$ python3 randomkcnfsolver.py -o model -k 3 -m 15 -n 5
Clauses set:
{frozenset({'-D', 'E', 'B'}), frozenset({'E', 'C', 'A'}), frozenset({'-B', '-D', 'A'}), frozenset({'
D', 'E', 'B'}), frozenset({'D', 'C', '-A'}), frozenset({'E', 'A', 'B'}), frozenset({'-B', 'D', 'E'})
, frozenset({'-B', 'C', '-A'}), frozenset({'D', 'A', 'B'}), frozenset({'-C', '-A', 'B'}), frozenset(
{'-A', '-E', 'B'}), frozenset({'-D', 'C', 'B'}), frozenset({'-D', '-E', 'B'}), frozenset({'D', 'C',
'A'}), frozenset({'-B', 'D', 'C'})}
Model: {'E': False, 'A': True, 'B': True, 'C': True, 'D': True}
```

Funzione *model* che mostra una formula 3-CNF(15, 5) generata casualmente e un possibile modello che la soddisfa.

2. $plot(loops, tries, k, max\_m\_value, max\_n\_value, max\_flips, p)$: La funzione prende in input i parametri necessari a creare due datasets. Il primo descrive la probabilità che una formula proposizionale $k-CNF(m,n)$ generata casualmente, dove $1 \le m < max\_m\_value$ e $k \le n < max\_n\_value$, con rapporto $\frac{m}{n}$ sia soddisfacibile. Il secondo descrive il tempo necessario, in millisecondi, all'algoritmo $WalkSAT$ per ritornare un output, data in input una formula proposizionale $k-CNF(m,n)$ con rapporto $\frac{m}{n}$ generata casualmente. I valori dei due datasets sono calcolati effettuando una media per ogni rapporto $\frac{m}{n}$. Il numero di diversi rapporti $\frac{m}{n}$ generabili è specificato dal parametro $loops$, mentre il numero di diverse formule generabili con rapporto $\frac{m}{n}$ è specificato

[2] In questo caso, la disgiunzione viene utilizzata in modo esclusivo.

dal parametro *tries*. La funzione restituisce due modelli affiancati, che rappresentano graficamente i datasets calcolati. Le *'x'* sui grafici (marker) rappresentano i valori effettivamente calcolati.



Il grafico di sinistra mostra la probabilità che una formula 3-CNF generata casualmente sia soddisfacibile, come una funzione del rapporto m/n.
Il grafico di destra mostra il runtime medio di WalkSAT, calcolato su formule 3-CNF generate casualmente. Si può vedere che in questo caso i problemi più difficili hanno un rapporto m/n maggiore di circa 4.3.

```
$ python3 randomkcnfsolver.py -l 50 -t 10
M = 46 / N = 27
Test #1
Test #2
Test #3
Test #4
Test #5
Test #6
Test #7
Test #8
Test #9
Test #10

M = 61 / N = 10
Test #1
Test #2
Test #3
Test #4
Test #5
Test #6
Test #7
Test #8
Test #9
Test #10

M = 64 / N = 23
Test #1
Test #2
Test #3
Test #4
```

Funzione *plot* in corso di esecuzione, con parametri loops = 50 e tries = 10, per un totale di 500 iterazioni.