WalkSAT e la Transizione di Fase in Problemi SAT Casuali

Andrea Ferritti

Corso di Laurea in Ingegneria Informatica
Università degli Studi di Firenze
N° Matricola: 7113347

Anno Accademico 2024/2025

1 Elaborato Assegnato

La presente relazione illustra l'implementazione e l'analisi dell'elaborato assegnato relativo alla parte di logica, con particolare riferimento all'algoritmo `WalkSAT` il quale viene applicato a formule in forma normale congiuntiva (`k-CNF`) generate rispettando alcuni vincoli. Nello specifico, dati il numero di simboli proposizionali n, il numero di clausole m e la lunghezza delle clausole k:

1. Ogni clausola deve contenere k letterali, evitando duplicazioni al suo interno.
2. Nessuna clausola deve essere banale (non è una tautologia).
3. Tutte le clausole devono essere uniche, evitando ripetizioni.

I letterali sono stati selezionati da una distribuzione uniforme sui $2n$ letterali possibili. I risultati ottenuti sono stati visualizzati seguendo il modello della Figura 7.19 del testo di riferimento *Artificial Intelligence: A Modern Approach* (Russell e Norvig, 2020), che illustra, attraverso dei grafici, il fenomeno della transizione di fase.

1.1 Struttura del Progetto

Il progetto, sviluppato in Python, è stato organizzato in modo modulare attraverso quattro file sorgente. Ecco una breve descrizione delle loro funzionalità:

- `kcnf_generator.py` Generazione di formule k-CNF casuali che rispettano i vincoli.
- `walksat.py` Implementazione dell'algoritmo WalkSAT, utilizzato per verificare che le formule k-CNF siano soddisfacibili.
- `phase_transition.py` Generazione dei grafici che illustrano il fenomeno della transizione di fase nel contesto dei problemi SAT casuali.

- `common.py` Fornisce librerie di utilizzo comune (Tabella 1).

Libreria	Utilizzo
`random`	Generazione di valori casuali.
`string`	Accesso alle lettere dell'alfabeto.
`itertools`	Generazione combinazioni di lettere.
`matplotlib` (alias `plt`)	Creazione dei grafici.
`numpy` (alias `np`)	Operazioni matematiche.
`tqdm`	Barra di avanzamento.

Tabella 1: Librerie utilizzate nel progetto.

Infine, nel file `common.py` è stata definita anche una costante `NEG = '¬'`, utilizzata come simbolo di negazione nelle formule k-CNF.

2 Implementazione

2.1 Generazione di Formule k-CNF Casuali

La classe `KCNFGenerator` nel file `kcnf_generator.py` è responsabile della generazione di formule k-CNF casuali che rispettano i vincoli specificati. I $2n$ letterali vengono generati dal metodo `generate_literals`, utilizzando combinazioni di lettere maiuscole dell'alfabeto. La combinazione di lettere è possibile grazie alla funzione `product` della libreria `itertools`, che genera tutte le combinazioni cartesiane di lettere dell'alfabeto per una lunghezza specificata. In questo modo, i letterali vengono creati in sequenza partendo da simboli di una sola lettera (`A`, `B`, ..., `Z`) e, successivamente, combinando le lettere per formare simboli più lunghi (`AA`, `AB`, ..., `AZ`, `BA`, ...). Le clausole vengono create dal metodo `generate_clause`, che seleziona k letterali distinti in modo casuale e rigenera la clausola finché ne ottiene una che non contiene tautologie. Quest'ultime sono verificate attraverso il metodo `is_tautology` il quale controlla se una clausola contiene contemporaneamente un letterale e la sua negazione. Successivamente, il metodo `generate_formula` costruisce l'intera formula con m clausole, verificando l'unicità di ciascuna clausola tramite un insieme `seen_clauses`. Il risultato è una formula k-CNF composta da m clausole che può essere convertita in una stringa con i connettivi logici grazie al metodo `formula_to_string`.

2.2 Algoritmo WalkSat

La classe `WalkSAT`, implementata nel file `walksat.py`, realizza l'algoritmo WalkSAT, utilizzato per verificare la soddisfacibilità di una formula. L'algoritmo cerca un modello, ovvero un'assegnazione di valori di verità ai simboli proposizionali, che soddisfi la formula, qualora tale modello esista. I simboli vengono estratti dalla formula con il metodo `extract_symbols`, che identifica le variabili uniche nella formula, ignorando le negazioni. L'assegnazione iniziale dei valori di verità viene generata casualmente tramite il metodo `initialize_assignment` che restitusce un dizionario dove la chiave è il simbolo e il valore è true/false. Per valutare l'assegnazione, il metodo `is_clause_satisfied` verifica se almeno un letterale in una clausola è soddisfatto, mentre `count_satisfied_clauses` calcola il numero totale di clausole soddisfatte per un'assegnazione data. Durante l'esecuzione

dell'algoritmo, il metodo `find_unsatisfied_clause` identifica una clausola non soddisfatta nella quale viene selezionato un simbolo da "flippare" (invertire valore di verità). Il simbolo viene scelto casualmente in base a una probabilità `p` oppure, in alternativa, si seleziona quello che massimizza il numero di clausole soddisfatte dopo un flip, utilizzando il metodo `count_satisfied_clauses_after_flip`. L'algoritmo procede iterativamente nel metodo `solve`, eseguendo al massimo `max_flips` per tentare di trovare una soluzione. Se tutte le clausole vengono soddisfatte prima di raggiungere il limite di iterazioni, restituisce il modello trovato insieme al numero di iterazioni effettuate `num_flip`. In caso contrario, termina segnalando un fallimento e riportando comunque il numero totale di iterazioni eseguite che coinciderà con `max_flips`.

```
Formula:  [frozenset({'¬Z', 'C', '¬AR'}), frozenset({'¬Y', 'AO', 'AN'}), frozenset({'B', '¬H', '¬N'}), frozenset({'¬AT', '¬M', 'C'})]

Con connettivi logici:  (¬Z v C v ¬AR) ∧ (¬Y v AO v AN) ∧ (B v ¬H v ¬N) ∧ (¬AT v ¬M v C)

Modello:  {'M': False, 'AR': True, 'Y': False, 'AO': False, 'AT': False, 'H': True, 'B': True, 'N': True, 'AN': False, 'Z': False, 'C': True}
```

Figura 1: Test su una $CNF_3(4, 50)$

2.3 Analisi Transizione di Fase

Il file `phase_transition.py` contiene la funzione `analyze_phase_transition`, che analizza il fenomeno della transizione di fase nei problemi SAT casuali. Questo fenomeno descrive il cambiamento repentino della probabilità di soddisfacibilità di una formula in funzione del rapporto m/n, dove m è il numero di clausole e n è il numero di simboli proposizionali. La funzione riceve i parametri: `num_symbols`, `len_clauses` numero di letterali per clausola, `num_samples` numero di campioni e il numero di rapporti da analizzare `num_ratios`. Utilizzando la funzione `linspace` della libreria `numpy`, il rapporto m/n viene suddiviso uniformemente in un intervallo come quello utilizzato per i grafici del libro ($[0.5, 8]$), e per ciascun valore vengono calcolati due risultati principali: la probabilità di soddisfacibilità (`sat_prob`) e il numero medio di iterazioni (`avg_iterations`) richieste dall'algoritmo WalkSAT per trovare un modello o restituire `None`. Per ogni valore di m/n, la funzione genera `num_samples` formule casuali k-CNF utilizzando la classe `KCNFGenerator`, quindi applica l'algoritmo WalkSAT (con `max_flips`=2000 e `p`=0.5) per determinare se ciascuna formula è soddisfacibile oppure no. I risultati vengono raccolti in due liste principali: `sat_probabilities`, che memorizza la probabilità di soddisfacibilità, e `avg_iterations_list`, che memorizza il numero medio di iterazioni per ciascun rapporto. La funzione visualizza i risultati attraverso due grafici generati con `matplotlib`. Il primo rappresenta la probabilità di soddisfacibilità in funzione del rapporto m/n mentre il secondo mostra il numero medio di iterazioni richiesto dall'algoritmo WalkSAT in funzione dello stesso rapporto.

Infine, il file include una funzione ausiliaria, `run_analysis`, che definisce i parametri utilizzati ($n = 50$, $k = 3$, 100 formule per rapporto e 50 rapporti da analizzare) e avvia l'analisi chiamando `analyze_phase_transition`. Quando il file viene eseguito, la funzione `run_analysis` viene invocata tramite il costrutto `if __name__ == "__main__"` e lo stato dell'esecuzione può essere monitorato grazie alla barra di caricamento fornita dalla libreria `tqdm`.

3 Risultati e Conclusioni

I grafici (Figura 2) sono generati su $CNF_3(m, 50)$ casuali , ovvero formule in forma normale congiuntiva costruite a partire da 50 simboli proposizionali e composte da clausole contenenti esattamente 3 letterali ciascuna. L'analisi è stata condotta in funzione del rapporto m/n, in linea con l'approccio descritto nel libro.

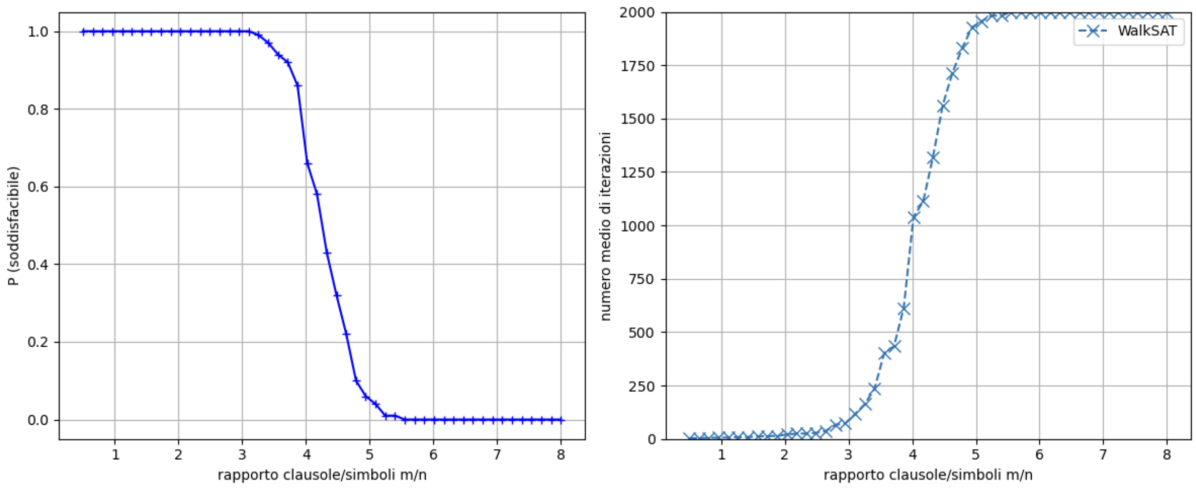

Figura 2: Grafici generati

In particolare, il grafico a sinistra mostra la probabilità che una formula $CNF_3(m, 50)$ casuale sia soddisfacibile rispetto al rapporto m/n, mentre quello a destra rappresenta il numero medio di iterazioni dell'algoritmo WalkSAT sullo stesso tipo di formule e in funzione dello stesso rapporto.

Per rapporti m/n piccoli la probabilità si avvicina a 1, mentre al crescere del rapporto la probabilità diminuisce fino a 0. Nello specifico si può notare che la probabilità di soddisfacibilità diminuisce bruscamente intorno a un valore $r_3 \simeq 4,3$ detto rapporto di soglia. Il fenomeno osservato nell'esperimento è la cosiddetta `transizione di fase`, in accordo con quanto previsto dalla teoria. Anche il numero medio di iterazioni dell'algoritmo mostra un andamento significativo in prossimità di r_3: aumenta rapidamente fino a raggiungere il limite massimo di flip consentiti nelle regioni dove la probabilità di soddisfacibilità si avvicina a zero.