Generazione k-CNF

Una formula in logica proposizionale espressa in Conjunctive Normal Form, o CNF, è una congiunzione di clausole costituite da disgiunzioni di letterali. Nel particolare, se espressa come k-CNF, allora ogni clausola contiene al più k letterali. Il progetto le richiede in modo tale che:

1. Ogni clausola ha esattamente k letterali

La creazione di ogni clausola è determinata da una procedura di estrazione di letterali da un insieme, inizialmente di dimensione 2n, contenente, per ciascun simbolo, il letterale positivo e il letterale negato.

Per evitare che in una clausola possano esserci dei letterali duplicati è sufficiente estrarre di volta in volta un letterale e impedire che quest'ultimo possa essere recuperato in un'estrazione successiva. A tal proposito si potrebbe:

- verificare se il nuovo letterale da inserire sia già nella clausola in costruzione, e solo in caso non sia un duplicato procedere con l'inserimento; oppure
- estrarre un letterale dall'insieme ed eliminarlo dal suddetto per evitare a priori che possa essere estratto una seconda volta.

Ovviamente il secondo metodo è più pratico, in quanto garantisce che in k estrazioni venga generata la clausola, e ottimo, poiché non viene perso tempo computazionale per il confronto.

2. Ogni clausola è non banale

Una clausola si dice banale, o tautologia, se e solo se è vera in ogni modello. Contestualizzata alle formule k-CNF in logica proposizionale, una clausola è tautologia se appare almeno una coppia di letterali complementari.

Come nel caso precedente, si hanno due alternative la cui migliore elimina dall'insieme il complementare del letterale appena estratto, per evitare a priori che possa esserci una tautologia.

3. Tutte le clausole sono distinte

Fissati k e n, il numero massimo di clausole generabili sono pari a $\binom{2n}{k}$. Considerate anche le condizioni aggiuntive del problema, il numero totale di clausole distinte che è possibile costruire è $m_{max} = \sum_{i=0}^{i=k} \binom{n-i}{k-i}\binom{n}{i}$. Al variare di n e k, m_{max} rimane un numero confrontabile con $\binom{2n}{k}$, per cui non vengono aggiunte condizioni restrittive ulteriori nella costruzione della clausola, ed eventualmente la si scarta se già presente nella formula proposizionale.

4. Estrazione da una distribuzione uniforme

Ogni linguaggio di programmazione ha funzioni native che consento di scegliere un valore da un insieme mediante una distribuzione uniforme. In genere sono pseudo-casuali e si affidano sull'inizializzazione di un *seme*.

NOTA: n = # simboli proposizionali; m = # clausole; k = # letterali contenute in una clausola.

Davis-Putnam

L'algoritmo di Davis-Putnam richiede in ingresso una formula in logica proposizionale e CNF, i simboli proposizionali di cui è composta e il modello ad essa associata, ovvero un insieme di assegnazioni di valori booleani ai simboli proposizionali. La chiamata iniziale è costituita da:

- La formula proposizionale costruita da *clauseGenerator*;
- I simboli proposizionali che appaiono nella suddetta formula, ottenuti da *symbolDetector*;
- Una lista vuota di coppie.

In uscita l'algoritmo restituisce un valore booleano indicante se il problema è soddisfacibile.
A tal proposito utilizza 3 euristiche:

1. Premature Termination

Si consideri una clausola come una lista di letterali di cui il modello ne specifica il valore di verità. Se nel modello non è definito alcun valore, allora non è ancora stata fatta nessuna interpretazione. Ogni nuova interpretazione di un letterale viene applicata a tutte le clausole che lo contengono:

- Se prende valore *True*, la clausola è *True*, e non viene più considerata. Nel caso non ci siano altre clausole da controllare vuol dire che tutte le clausole sono state interpretate come *True*, pertanto DPLL ritorna *True*; altrimenti continua la normale esecuzione delle istruzioni.
- Se prende valore *False*, viene tolto il letterale dalla clausola e si verifica se la clausola è vuota. In tal caso vuol dire che tutti i letterali hanno assunto valore *False*, per cui la clausola è *False*, per cui DPLL ritorna *False*; in caso contrario continua la normale esecuzione delle istruzioni.

Se non viene fatto tornare nessun valore booleano vuol dire che l'interpretazione appena inserita nel modello non ha fatto scattare l'euristica di terminazione prematura.

2. Pure Symbols

Consiste nella ricerca di tutti i simboli puri che si trovano nella formula proposizionale k-CNF, tramite *findPureSymbol*. Fintanto che ne rimangono, viene fornita l'interpretazione nel modello ad ogni simbolo puro in modo tale che risulti *True*. Dopodiché l'euristica di Premature Termination (*removeClauses*) viene invocata, ed almeno una clausola verrà "semplificata" dal problema.

3. Unit Clauses

Vengono cercate le unit clauses della formula proposizionale. Appena trovata si aggiunge l'interpretazione del suo unico letterale in modo che risulti *True*, e analogamente al caso precedente viene invocato *removeClauses*, che la "semplifica". Ciò garantisce la terminazione dell'euristica con un numero finito di iterazioni.

Implementazioni strutture dati e algoritmi

Letterale: intero con segno, positivo se è un letterale positivo, negativo se è un letterale negato.
Clausola: insieme di letterali, dunque necessariamente tutti differenti, di cardinalità k;
Formula: lista di clausole, implicitamente connesse da and logici, di lunghezza m;

I letterali vengono generati da *literalGenerator*, il quale richiede solo n e restituisce la lista *literals*.
Le clausole sono generate tramite *clauseGenerator*, il quale richiede come parametri k, m e literals, data come copia profonda.
Modello: lista di tuple in cui il primo valore è un simbolo proposizionale, mentre il secondo è il relativo valore booleano datogli nell'interpretazione.

I test

Gli esperimenti su cui concentrarsi devono mettere in luce la *satisfiability threshold conjecture*, secondo la quale, per ogni k≥3, la probabilità che una formula k-CNF in logica proposizionale sia soddisfacibile è 1 per ogni $^m/_n < R_k$ e 0 per ogni $^m/_n > R_k$. R_k è la soglia e diventa sempre più ripida al crescere di n. Inoltre il tempo di esecuzione dell'algoritmo DPLL nell'intorno della soglia aumenta vertiginosamente, e decresce allontanandosene.

Il valore di k viene mantenuto pari a 3, per semplicità. Il valore di n, invece, viene aumentato. Fissati dunque n e k si misurano, al variare di m:

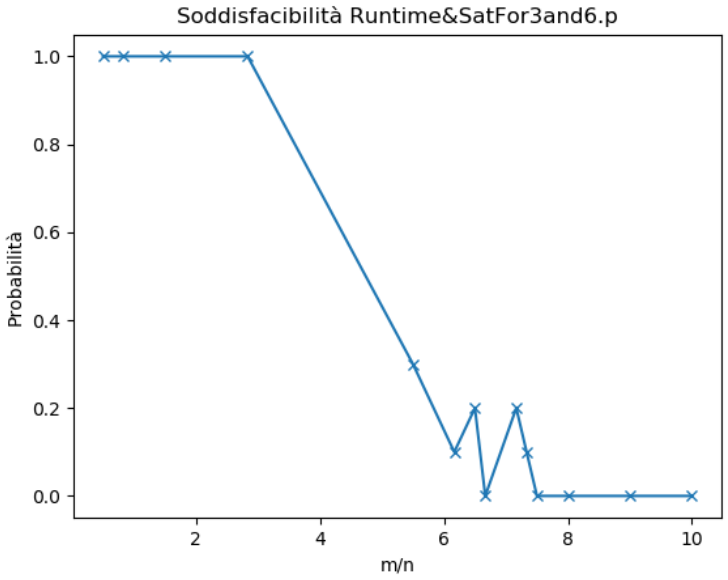
- La probabilità di soddisfacibilità (*prob*), ottenuta come rapporto tra i test ad esito positivo del DPLL e i test eseguiti;
- Il tempo di esecuzione dell'algoritimo (*recTime*), calcolato come media aritmetica tra un certo numero di test *smoothNumber*.

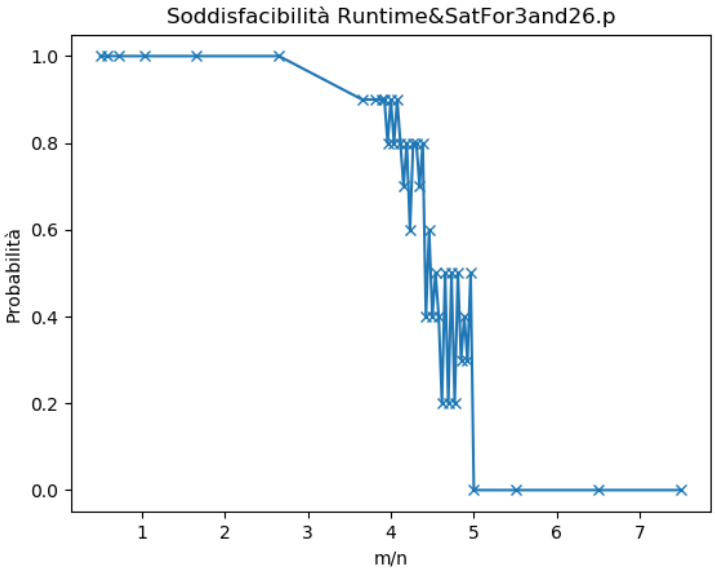
Nella scelta devono rimanere valide due condizioni: $k \leq n$ e $m \leq m_{max}$.

Risultati sperimentali

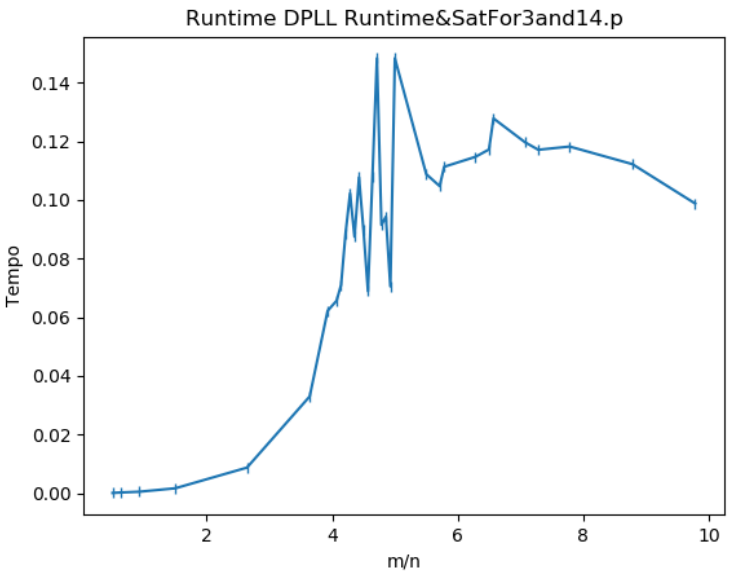
Di seguito vengono mostrati i risultati sperimentali inseriti in appositivi grafici che mostrano come la congettura sia effettivamente reale. Si noti in particolare:

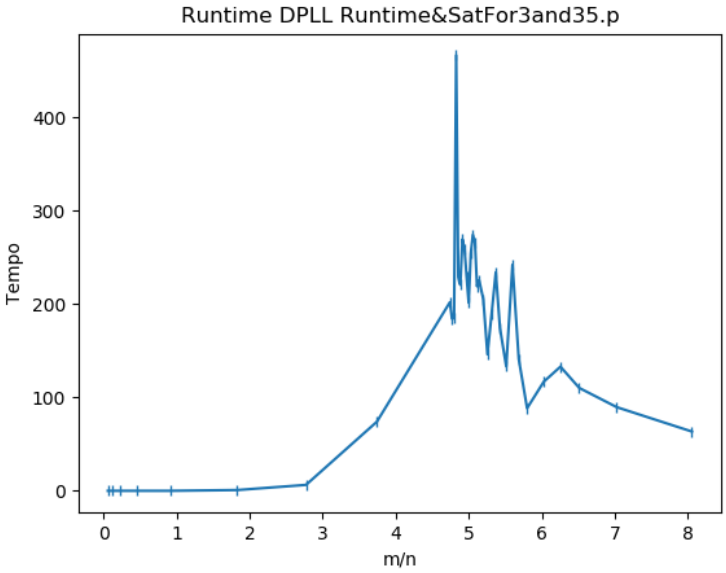
- l'esistenza di una soglia prima della quale la probabilità di soddisfacibilità sia 1 e dopo la quale sia 0;
- la soglia si riduce all'aumentare di n;





- il tempo di esecuzione nell'intorno della soglia aumenta vertiginosamente;
- appena dopo la soglia, cioè ai primi valori nulli di probabilità, il tempo di esecuzione ha un ulteriore picco, dopo il quale si ha l'effettiva caduta del tempo di esecuzione;
- la diminuzione del tempo di esecuzione per n basso non è molto evidente, mentre al suo aumentare diventa sempre più ripido;





Per una visione più dettagliata dei dati si rimanda ai file Runtime&SatForKandN.p (pickle), i quali contengono tutti i dati recuperati dall'esecuzione. K e N sono valori numerici che fanno riferimento al numero di letterali k e simboli proposizionali n utilizzati.

Conclusioni

Per quanto è stato detto e per quanto è stato riportato mediante i risultati sperimentali, si può concludere che effettivamente la congettura della soglia di soddisfacibilità ha riscontro empirico, sebbene non sia stato ancora dimostrato. Un'altra conclusione a cui si può arrivare è che effettivamente ha senso considerare il rapporto di numero di clausole con il numero di simboli proposizionali utilizzati, in quanto l'uno senza l'altro determinerebbe una perdita di informazione fondamentale. Basti pensare che n influenza procedure quali le euristiche precedentemente esposte, mentre m influenza la durata di esecuzione dell'algoritmo. Infine l'aumento esponenziale del tempo di esecuzione dell'algoritmo all'aumentare di n e k mostra che effettivamente il problema SAT è NP-completo.