UNIVERSITÀ DI PISA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Laurea Triennale in Ingegneria Informatica

Modellazione del problema MAX3-SAT in forma QUBO e sua risoluzione tramite Simulated Annealing

Relatore:

Prof. Marco Cococcioni

Candidato:

Francesco Vesigna

ANNO ACCADEMICO 2024/2025

Indice

1 Introduzione **3**
- 1.1 Importanza del problema 3-SAT . . . 4
- 1.2 Applicazioni del problema 3-SAT . . . 4
- 1.3 Computer Quantistici e QUBO . . . 5
- 1.4 Obbiettivi della tesina . . . 6

2 Trasformazione da Istanza MAX3-SAT a Istanza QUBO **7**
- 2.1 Trasformazioni Esatte . . . 8
  - 2.1.1 Trasformazione di Choi . . . 8
  - 2.1.2 Trasformazione di Chanchellor . . . 10
- 2.2 Trasformazioni Approssimate . . . 14
  - 2.2.1 Approssimazioni Naive . . . 14
  - 2.2.2 Pattern QUBO Search . . . 14
- 2.3 Implementazione . . . 18

3 QUBO Solvers **20**
- 3.1 Simulated Annealing . . . 21
  - 3.1.1 Criterio di Metropolis . . . 21
  - 3.1.2 Catene di Markov e SA . . . 22
  - 3.1.3 Cooling schedule . . . 23
  - 3.1.4 Memoization . . . 25
- 3.2 Implementazione . . . 26

4 Valutazione della soluzione **28**
- 4.1 Implementazione . . . 28
- 4.2 Benchmarks . . . 29

A Matrici risultanti dalla trasformazione di Chanchellor **30**

Capitolo 1

Introduzione

Il problema SAT è uno dei problemi fondamentali nella teoria della complessità computazionale. In generale, consiste nello stabilire se esista un assegnamento delle variabili booleane che renda vera una formula logica. In ambito teorico e pratico, è comune considerare formule espresse in forma normale congiuntiva (CNF), una rappresentazione standard che facilita l'analisi e la risoluzione algoritmica. Questa scelta è giustificata dal fatto che qualunque formula booleana può essere trasformata in una formula logicamente equivalente in CNF attraverso una procedura con complessità polinomiale.

Definizione (1.1 | Formula CNF)
Una formula CNF ha la seguente forma:

$$\bigwedge_i \left(\bigvee_j y_{ij} \right)$$

Ogni y_{ij} è un letterale e rappresenta una precisa variabile booleana x_i o una sua negazione $\neg x_i$, mentre $(\wedge y_{ij})$ è una clausola e nel caso di una fomula N-CNF ciscuna di esse contiene esattamente N letterali.

In questa tesina ci concentreremo su un sottoproblema del problema SAT, ossia il problema 3-SAT. Questa variante riguarda solo formule booleane in forma 3-CNF.

Esempio (1.1 | Problema 3-SAT)

$$\phi(x_1, x_2, x_3) = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee x_3)$$

ϕ è una formula booleana in forma 3-CNF. Elenchiamo ora tutti i possibili assegnamenti e vediamo se è soddisfacibile o meno

$$\phi(0,0,0) = \text{false}, \quad \phi(0,0,1) = \text{true}, \quad \phi(0,1,0) = \text{true}, \quad \phi(0,1,1) = \text{false},$$

$$\phi(1,0,0) = \text{true}, \quad \phi(1,0,1) = \text{true}, \quad \phi(1,1,0) = \text{true}, \quad \phi(1,1,1) = \text{true}$$

Abbiamo trovato 6 soluzioni, la formula è dunque soddisfacibile.

1.1 Importanza del problema 3-SAT

Malgrado possa sembrare più semplice del problema SAT, il problema 3-SAT è comunque NP-completo. Infatti, è stato dimostrato che ogni problema 3-SAT può essere ridotto a un problema SAT tramite un algoritmo polinomiale. Inoltre, la natura NP del problema è evidente, poiché un qualsiasi assegnamento può essere verificato in tempo polinomiale. Per questo motivo, il 3-SAT è diventato un punto di riferimento fondamentale per la comunità di ricerca che si occupa di problemi NP.

Per analizzare gli algoritmi che svilupperemo, ci concentreremo su una variante chiamata MAX3-SAT. A differenza del 3-SAT, che è un problema di riconoscimento, il MAX3-SAT è un problema di ottimizzazione: si cerca la configurazione delle variabili booleane che massimizza il numero di clausole soddisfatte. Sebbene sembri un problema diverso, in realtà è strettamente collegato, perché una volta trovata la configurazione ottimale sappiamo anche se la formula originale è soddisfacibile.

Esempio (1.2 | Problema MAX3-SAT)
Rifacendoci all'esempio (1.1), se lo riformulassimo come un problema MAX3-SAT otterremmo:

$$\phi(0,0,0) = 1, \quad \phi(0,0,1) = 2, \quad \phi(0,1,0) = 2, \quad \phi(0,1,1) = 1,$$

$$\phi(1,0,0) = 2, \quad \phi(1,0,1) = 2, \quad \phi(1,1,0) = 2, \quad \phi(1,1,1) = 2$$

Osservazioni:

1. Abbiamo due massimi locali e sei massimi globali. Il massimo globale coincide con il numero totale di clausole, quindi possiamo concludere che la formula è soddisfacibile.
2. La funzione $f(x_{as})$ corrisponde al numero di clausole soddisfatte; ogni clausola soddisfatta da x_{as} **contribuisce** ad aumentare $f(x_{as})$. Questa osservazione sarà fondamentale nelle sezioni successive.

1.2 Applicazioni del problema 3-SAT

Nonostante il problema 3-SAT sia una formulazione astratta e teorica, rappresenta la base per modellare e risolvere molteplici problemi concreti e complessi. Di seguito riportiamo alcune delle sue applicazioni più rilevanti:

1. **Crittoanalisi**
   Gli attacchi crittografici brute-force, cioè quelli che cercano di trovare una

chiave enumerando e testando tutte le possibili combinazioni, possono essere formulati come problemi 3SAT. Velocizzare la risoluzione di questi problemi aiuterebbe i crittoanalisti a rompere cifrari più rapidamente. In modo realistico, questo metterebbe in crisi gran parte della crittografia utilizzata nelle applicazioni considerate "sicure".

2. **Logistica**
   l problema del routing per voli e tratte, più in generale il vehicle routing problem, può essere ridotto a un problema 3-SAT, così come la creazione di un orario in contesti lavorativi (job-packet scheduling).

3. **Progettazione e Test di circuiti digitali**
   Progettare un circuito digitale efficiente spesso si riduce alla minimizzazione di un'espressione logica, che può essere formulata come un problema 3-SAT. Il problema 3-SAT è inoltre rilevante nella verifica del corretto funzionamento del circuito: possiamo infatti rappresentare il circuito e le condizioni di malfunzionamento come un'istanza 3-SAT. La soluzione a questo problema fornirà un testcase in grado di evidenziare eventuali malfunzionamenti.

1.3 Computer Quantistici e QUBO

Sappiamo che i problemi NP-completi sono notoriamente difficili da risolvere, nonostante ciò i computer quantistici possono esservi d'aiuto in quanto capaci di ridurre esponenzialmente il tempo di calcolo della soluzione. I solver implementati su modelli quantistici lavorano con due tipi di input isomorfi:

1. QUBO (Quadratic Unconstrained Binary Optimization)
2. Ising Spin Glass

È dunque necessario trovare un modo per trasformare un istanza di un problema 3-SAT in un problema QUBO o Ising Spin Glass. Noi ci concentreremo su QUBO, anche se sono del tutto equivalenti.

Definizione (1.2 | QUBO)
Data una matrice $Q \in \mathbb{R}^{n\times n}$ e $x \in \mathbb{F}_2^n$ un vettore booleano n-dimensionale. Il problema QUBO è definito nel seguente modo:

$$\min \quad H_{\text{QUBO}}(x) = x^T Q x = \sum_i^n Q_{ii} x_i^2 + \sum_{i<j}^n Q_{ij} x_i x_j$$

Nota: siccome $a^2 = a \ \ \forall a \in \mathbb{F}_2$, si può riscrivere $H_{\text{QUBO}}(x)$ come

$$H_{\text{QUBO}}(x) = \sum_i^n Q_{ii} x_i + \sum_{i<j}^n Q_{ij} x_i x_j$$

$H_{\text{QUBO}}(x)$ è detta l'energia del vettore x.

1.4 Obbiettivi della tesina

L'obbiettivo di questa tesina è lo sviluppo di uno script in linguaggio MATLAB che, data un'istanza del problema MAX3-SAT, la converta in una formulazione QUBO e ne calcoli una soluzione, restituendo l'assegnamento trovato e il numero di clausole soddisfatte. Verranno analizzate diverse tecniche di trasformazione, sia esatte che approssimate. Come metodo di risoluzione sarà adottato il simulated annealing, scelto tra i vari approcci disponibili per la sua semplicità ed efficacia. Infine, i risultati ottenuti verranno riportati nella sezione dedicata al benchmark.

Capitolo 2

Trasformazione da Istanza MAX3-SAT a Istanza QUBO

Per capire come trasformare una problema MAX3-SAT in un problema QUBO dobbiamo prima differenziare due tipi di trasformazioni:

- Trasformazioni esatte: La matrice Q, che rappresenta il problema QUBO, descrive perfettamente il problema originale
- Trasformazioni approssimate: La matrice Q, che rappresenta il problema QUBO, non descrive perfettamente il problema originale, questo vuol dire che esistono assegnamenti ottimi non codificati nella matrice Q, in altre parole potremmo avere assegnamenti ottimi che non hanno energia minima ($H_{\text{QUBO}}(x_o) > E$) dove E è il minimo globale di H e x_o è un assegnamento ottimo.

Sebbene i vantaggi delle trasformazioni esatte siano evidenti, è importante chiarire i benefici derivanti dall'utilizzo di matrici Q approssimate:

- La matrice Q è di dimensioni più piccole o più sparsa, il che rende il problema più trattabile da un punto di vista computazionale.
- Le macchine quantistiche sono, per natura, probabilistiche, in particolare restituiscono soluzioni campionate da uno spettro di configurazioni, e non sempre il minimo assoluto. Di conseguenza, una formulazione esatta non garantisce di ottenere sempre la soluzione ottimale.
- I solver più popolari implementati su macchine quantistiche (i.e. tabu search e quantum annealing) sono euristici, ovvero non forniscono nessun tipo di garanzia sulla qualità del risultato. In questo contesto, una buona approssimazione del problema può essere sufficiente per ottenere risultati utili, con un costo computazionale significativamente inferiore.

2.1 Trasformazioni Esatte

In questa sezione introduciamo alcune delle trasformazioni esatte più utilizzate per la conversione di problemi MAX3-SAT in problemi QUBO.

2.1.1 Trasformazione di Choi

La prima trasformazione che andremo ad analizzare è la trasformazione di Choi, una delle più note nel contesto della trasformazione di problemi SAT in problemi QUBO. Questa trasformazione si basa sul problema MWIS (Maximum Weight Independent Set), che consente di rappresentare l'istanza MAX3-SAT attraverso un grafo e di convertirla in un'istanza QUBO risolvibile successivamente.

Definizione (2.1 | MWIS)
Dato un grafo non orientato $G = (V, E)$, dove V è l'insieme dei vertici e E è l'insieme degli archi, un sottoinsieme $V' \subset V$ è un'insieme indipendente di vertici se tutti i vertici appartenenti a V' sono a coppie non adiacenti ($\forall u, v \in V', \quad (u, v) \notin E$). Assegnamo a ogni vertice un peso che definiamo come $w_v \in \mathbb{R}^+$.
Il problema MWIS consiste nel trovare un set indipendente $I \subset V$ tale che la somma dei pesi dei vertici in I ($w_{tot} = \sum_{v \in I} w_v$) sia massima tra tutti i possibili set indipendenti $J \subset V$.

Un'istanza del problema MWIS può essere trasformata in un'istanza QUBO come segue:

$$H(x_1, ..., x_n) = -\sum_{i \in V} w_i x_i + \sum_{(i,j) \in E} Q_{ij} x_i x_j$$

- $x_i \in \{0, 1\}$ è una variabile booleana che indica se il vertice i è incluso nel set indipendente
- w_i è il peso associato al vertice i
- $Q_{ij} \geq |\min(w_i, w_j)| \quad \forall (i, j) \in E$

Il primo termine $-\sum_{i \in V} w_i x_i$ premia la selezione di vertici ad alto peso.
Il secondo termine $\sum_{(i,j) \in E} Q_{ij} x_i x_j$ penalizza la selezione simultanea di vertici adiacenti. Impostando $Q_{ij} \geq |\min(w_i, w_j)|$, ci si assciura che la penaltà superi il beneficio di includere due vertici adiacenti, scoraggiando soluzioni non indipendenti. Segue che qualsiasi soluzione ottima del problema QUBO non può non essere un set indipendente, e, tra tutti quelli possibili, è il set con peso totale massimo.

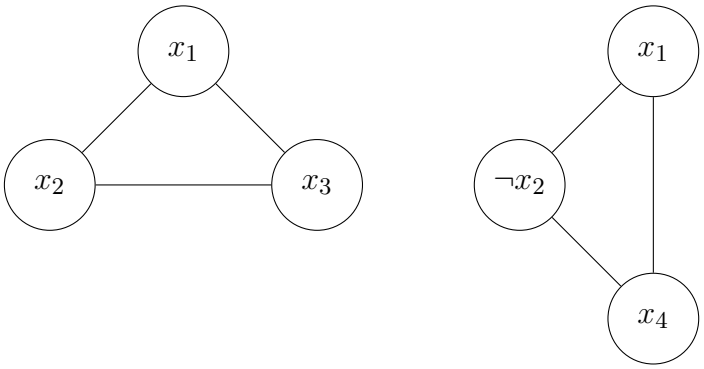
La struttura del grafo G deve codificare i vincoli e le interazioni tra le clausole della formula booleana.

Costruzione di G[1]
Dati un istanza 3-SAT $\phi(x_i, ..., x_n) = C_1 \wedge ... \wedge C_m$ con n variabili booleane, m clausole e $G_{\text{SAT}} = (V_G, E_G)$ un grafo vuoto, si procede come segue:

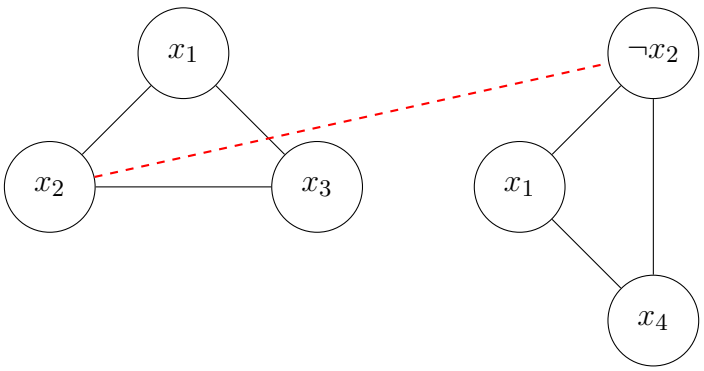
- Per ogni clausola $C_i = y_{i1} \vee y_{i2} \vee y_{i3}$, aggiungiamo una clique contenente 3 vertici al grafo. I vertici sono etichettati dai letterali y_{ij}. Poiché in una clausola basta che uno solo dei lettrali sia vero affinché essa stessa sia soddisfatta, la clique modella questa alternativa: la selezione di un singolo vertice nella clique è sufficiente per "attivare" la clausola.
- Si aggiunge un arco tra due clique se le etichette dei due nodi sono in conflitto, ovvero se $y_{is} \neq y_{jt}$ con $i \neq j$ (due clique associate a clausole differenti). L'arco collega i due letterali in conflitto e li rende adiacenti l'uno all'altro, l'assegnamento ottimo di conseguenza non potrà contenere entrambi i vertici.

A ogni vertice del grafo risultante viene associato lo stesso peso $w_v \in \mathbb{R}^+$ che può essere scelto arbitrariamente.

Esempio (2.1 | Trasformazione di Choi)
A partire dalla formula booleana $\phi(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_4)$, si costruiscono le due clique relative alle due clausole.



Si nota che i letterali y_{12} e y_{22} sono in conflitto dunque si inserisce l'arco nel grafo.



Il grafo risultante G rappresenta esattamente l'istanza MAX3-SAT, sottoforma di instanza MWIS. Impostiamo infine il peso $w_i = 1$. La matrice QUBO risultante è:

$$Q_{\text{choi}} = \begin{bmatrix} -1 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 1 \\ 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

matrice Q_f

Si osserva che abbiamo $3m$ vertici, dunque la matrice finale dovrà essere di dimensioni $3m \times 3m$.

2.1.2 Trasformazione di Chanchellor

Alla base della trasformazione di Chancellor c'è l'osservazione fatta nell' esempio (1.2): ogni clausola contribusce a massimizzare la funzione obbiettivo. Consideriamo, ad esempio, la clausola $(x_1 \vee x_2 \vee x_3)$, possiamo dire che la seguente equazione è valida:

$$(x_1 \vee x_2 \vee x_3) = \psi(x_1, x_2, x_3) = x_1 + x_2 + x_3 - x_1x_2 - x_2x_3 - x_1x_3 + x_1x_2x_3$$

Infatti per $x_{as} \neq (0, 0, 0)$ abbiamo che la clausola è soddsfatta e la funzione obbiettivo è massimizzata. Noi sappiamo che sia QUBO che Ising Spin Glass sono problemi di minimizzazione dunque occorre invertire la funzione obbiettivo.

$$-\psi(x_1, x_2, x_3) = \phi(x_1, x_2, x_3) = -x_1 - x_2 - x_3 + x_1x_2 + x_2x_3 + x_1x_3 - x_1x_2x_3$$

Si nota che questo non può essere un problema QUBO perchè è presente il termine cubico $x_1x_2x_3$. Per risolvere questo problema passiamo al problema dell'ising spin glass che è del tutto equivalente e vediamo se è possibile modellare il termine cubico di modo che diventi quadratico.

Definizione (2.2 | Ising Spin Glass)
Data una matrice $J \in \mathbb{R}^{n \times n}$, un vettore $s \in \{1, -1\}^n$ n-dimensionale e $h \in \mathbb{R}^n$. Il problema Ising Spin Glass è definito nel seguente modo:

$$\min \; H_{ISG} = \sum_{i}^{n} h_i s_i + \sum_{i<j}^{n} J_{ij} s_i s_j$$

Le variabili s_i sono dette di spin e hanno la stessa funzione delle varaibili booleane del problema QUBO. Per trasformare un problema di Ising Spin Glass in un problema QUBO basta sostituire:

$$x_i = \frac{s_i + 1}{2}$$

Ritornando al problema di prima possiamo espriemere la funzione obbiettivo utilizzando le variabili di spin (s_1, s_2, s_3) invece delle variabili booleane (x_1, x_2, x_3)

$$\phi(s_1, s_2, s_3) = -\frac{1}{8}s_1 - \frac{1}{8}s_2 - \frac{1}{8}s_3 + \frac{1}{8}s_1s_2 + \frac{1}{8}s_2s_3 + \frac{1}{8}s_1s_3 - \frac{1}{8}s_1s_2s_3 - \frac{7}{8}$$

Questa è la rappresentazione con le variabili spin della clausola. La rappresentazione con le variabili spin di una clausola generica può essere calcolata come segue:

$$\begin{aligned}(*)\ \ \phi(c(1), c(2), c(3), s_1, s_2, s_3) = &-\frac{1}{8}\big(c(1)s_1 + c(2)s_2 + c(3)s_3\big) \\ &+ \frac{1}{8}\big(c(1)c(2)s_1s_2 + c(1)c(3)s_1s_3 + c(2)c(3)s_2s_3\big) \\ &- \frac{1}{8}c(1)c(2)c(3)s_1s_2s_3 - \frac{7}{8}\end{aligned}$$

La variabile $c(i)$ indica il segno della variabile x_i, nel caso di $(x_1 \vee x_2 \vee x_3)$ abbiamo $c(i) = 1\ \ \forall i \in \{1, 2, 3\}$, siccome tutte le x_i sono non negate.
Rimane il problema per il termine cubico $s_1s_2s_3$. Chanchellor propone vari approcci per modellare il termine cubico[2]. In questa tesina proponiamo il seguente metodo, basato sul controllo di parità, con l'utilizzo di una sola variabile ausiliaria s_a detta *ancilla* spin[3].

$$I_{cubico}(s_1, s_2, s_3) = J\sum_{i=1}^{3}\sum_{j=1}^{i-1} s_is_j - c(1)c(2)c(3)\sum_{i=1}^{3} s_i + 2J\sum_{i=1}^{3} s_is_a - 2c(1)c(2)c(3)s_a$$

J è un parametro arbitrario e deve essere ≥ 1, per i risultati riportati nell'appendice si è scelto $J = 1$, un'altra scelta comune è $J = 5$. Si nota che, fissando s_1^*, s_2^*, s_3^*, minimizzare I_{cubico} su s_a equivale a fare un controllo di partà per le variabili $c(1)s_1^*, c(2)s_2^*, c(3)s_3^*$. I termini quadratici s_is_j sono fissati perciò la matrice J relativa al I_{cubico} non cambia al mutare della clausola analizzata, risulta dunque che l'energià di I_{cubico} dipende esclusivamente dalla parità. Infatti, dalla minimizzazione di I_{cubico} su s_a risulta che, per un numero dispari di 1 l'energia è $E_{\min} < 0$, mentre, per un numero pari di 1 l'energia è $E_{\min} + 2$. I termini rimanenti, presenti nell'equazione $(*)$ implicano un energia $-7 + E_{\min} + 2 + 6 = E_{\min} + 1$ per l'assegnamento che non soddisfa la clausola. Al contrario gli assegnamenti che soddisfano la clausola producono un energia minima di $E_{\min} - 7$. La funzione obbiettivo finale, modellata senza termini cubici, risulta:

$$\begin{aligned}(\#)\ \ \phi_{ISG}(c(1), c(2), c(3), s_1, s_2, s_3) = &-(c(1)s_1 + c(2)s_2 + c(3)s_3) \\ &+ (c(1)c(2)s_1s_2 + c(1)c(3)s_1s_3 + c(2)c(3)s_2s_3) \\ &+ I_{\text{cubico}} - 7\end{aligned}$$

Si ricorda che è possibile togliere le costanti K e moltiplicare per una costante C, per avere funzioni obbiettivo aventi coefficienti più semplici da manipolare. Supponiamo

ora di avere più clausole. Come possiamo costruire la funzione obiettivo dell'intera formula booleana evitando l'introduzione di termini cubici? Per farlo, calcoliamo la funzione obbiettivo di ciascuna clausola utilizzando le variabili di spin, semplifichiamo i termini cubici e **sovrapponiamo** [4] i contributi ottenuti per ottenere la funzione obbiettivo complessiva. Infine, possiamo riapplicare la trasformazione $s_i = 2x_i - 1$ per convertire le variabili di spin in variabili booleane e ricavare così la matrice Q corrispondente.

Sovrapposizione

Questo metodo sarà utilizzato nell'implementazione e consiste nel calcolare la funzione obbiettivo del problema (ISG o QUBO) a partire dalle funzioni obbiettivo associate alle singole clausole.

- Per ogni variabile spin s_i: denotiamo come K il set di tutte le clausole dove x_i appare (negata o non negata). L'elemento i del vettore h_f del problema finale si calcola come somma di tutti gli elementi i dei vettori h_l dei singoli problemi ISG che descrivono le singole clausole appartenenti a K.
- Per tutte le coppie di variabili spin s_i,s_j con $i < j$: denotiamo come K il set di tutte le clausole dove x_i e x_j appaiano entrambe (negate o non negate). L'elemento ij della matrice J_f del problema finale si calcola come somma di tutti gli elementi ij delle matrici J_l dei singoli problemi ISG che descrivono le singole clausole appartenenti a K.
- Tutti i termini quadratici o lineari contenenti le variabili *ancilla* non vengono modificati.

Conversione a QUBO

Una volta calcolato l'ISG finale, è possibile convertirlo nel corrispondente problema QUBO e, come già illustrato, ottenere la matrice Q. Schematizzando, l'intero procedimento può essere riassunto nei seguenti passaggi:

1. $(\phi_1, ..., \phi_K) \implies (\phi_{ISG_1}, ..., \phi_{ISG_K})$
2. $(\phi_{ISG_1}, ..., \phi_{ISG_K}) \implies \phi_{ISG_f}$
3. $\phi_{ISG_f} \implies \phi_{QUBO_f} \implies Q_f$

Un approccio alternativo consiste nel trasformare ciascuna clausola nella sua corrispondente formulazione QUBO singolarmente, e successivamente sovrapporre tutte le istanze QUBO ottenute per costruire l'istanza QUBO finale..

1. $(\phi_1, ..., \phi_K) \implies (\phi_{ISG_1}, ..., \phi_{ISG_K}) \implies (\phi_{QUBO_1}, ..., \phi_{QUBO_K})$
2. $(\phi_{QUBO_1}, ..., \phi_{QUBO_K}) \implies \phi_{QUBO_f}$
3. $\phi_{QUBO_f} \implies Q_f$

Anche se il procedimento può sembrare complesso, in realtà consiste semplicemente nella somma delle funzioni obiettivo: tutti i coefficienti vengono sommati, ad eccezione di quelli relativi alle variabili ancilla.

matrice Q_f

Utilizzando la trasformazione di Chanchellor, si ottiene una matrice Q_f di dimensioni $n+m \times n+m$, dove n è il numero di variabili booleane e m è il numero di clausole.

Esempio (2.2 | Sovrapposzione)
Prendiamo come esempio la formula logica $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_4)$.

La matrice Q_1, relativa alla prima clausola $(x_1 \vee x_2 \vee x_3)$, è la seguente:

$$Q_1 = \begin{bmatrix} -2 & 1 & 1 & 1 \\ 0 & -2 & 1 & 1 \\ 0 & 0 & -2 & 1 \\ 0 & 0 & 0 & -2 \end{bmatrix}$$

La matrice Q_2, relativa alla seconda clausola $(x_1 \vee \neg x_2 \vee x_4)$, è la seguente:

$$Q_2 = \begin{bmatrix} -1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

Per trovare la matrice Q_{1+2} dobbiamo **sovrapporre** le due matrici Q_1 e Q_2.

$$Q_{1+2} = \begin{bmatrix} -2-1 & 1+0 & 1 & 1 & 1 & 1 \\ 0 & -2+0 & 1 & 0 & 1 & 1 \\ 0 & 0 & -2 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

invarianza delle clausole

Ogni clausola 3-SAT rientra in uno dei quattro casi possibili, in base al numero di negazioni. È quindi possibile precalcolare le corrispondenti matrici Q_i e assegnarle alle clausole in fase di costruzione del modello QUBO.

- 0 negazioni: $(x_i \vee x_j \vee x_k)$
- 1 negazione: $(x_i \vee x_j \vee \neg x_k)$
- 2 negazioni: $(x_i \vee \neg x_j \vee \neg x_k)$
- 3 negazioni: $(\neg x_i \vee \neg x_j \vee \neg x_k)$

2.2 Trasformazioni Approssimate

Come anticipato in precedenza, le approssimazioni QUBO per problemi MAX3-SAT permettono di ottenere formulazioni più compatte o sparse, accettando però una perdita di precisione nella corrispondenza tra soluzioni ottime e minimi globali.

In questa sezione analizziamo alcune tecniche di approssimazione, partendo da metodi intuitivi basati sulla rimozione di termini (approssimazioni naive) fino a metodi più strutturati come il Pattern QUBO Search, che punta a ridurre sistematicamente la dimensione delle matrici associate alle clausole.

2.2.1 Approssimazioni Naive

Il metodo più intuitivo per creare approssimazioni QUBO per istanze MAX3-SAT consiste nell'eliminare entrate da una matrice QUBO esatta, ovvero una matrice che descrive fedelmente l'istanza originale di MAX3-SAT.
Utilizzando la trasformazione di Choi, si ottiene una matrice Q_f di dimensioni $3m \times 3m$, mentre con la trasformazione di Chanchellor la matrice Q_f ha dimensioni $n + m \times n + m$. Idealmente vorremmo ridurre il più possibile le dimensioni della matrice Q_f oppure renderla il più possibile sparsa. In generale, ogni algoritmo naive si inserisce in una di due categorie operative principali, ciascuna delle quali adotta un approccio diverso alla semplificazione della matrice.

- Min Pruning: Data una matrice Q esatta, rimuoviamo le N entrate più piccole di Q per ottenere una matrice Q approssimata.
- Random Pruning: Il random pruning lavora allo stesso modo con l'eccezione che non discrimina le entrate in base a segno o valore assoluto.

2.2.2 Pattern QUBO Search

Il pattern QUBO search è un' approssimazione sistematica e si basa sulla riduzione di dimensioni delle matrici Q_i associate alle clausole C_i. Sappiamo che, usando la trasformazione di Chanchellor, si trovano matrici di dimensioni 4×4, il pattern QUBO search mira a trovare versioni approssimate di dimensioni 3×3. Nella matrice Q_f finale le variabili ancilla portano a un incremento di m righe e m colonne, eliminandole del tutto si ha una riduzione drastica delle dimensioni di Q_f.
Si riporta ora un teorema fondamentale per la formulazione del Pattern QUBO Search approssimato [4].

Teorema (2.1)
Ipotesi: Date x_1, x_2, x_3 le variabili booleane contenute in una clausola di un istanza MAX3-SAT. Dati S_{SAT} l'insieme contenente tutti gli assegnamenti che soddisfano la clausola e S_{UNSAT} l'insieme di tutti gli assegnamenti che non soddisfano la clausola.

Consideriamo

$$f(x_1, x_2, x_3) = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_2 x_3 + \alpha_{12} x_1 x_2 + \alpha_{13} x_1 x_3 + \alpha_{23} x_2 x_3$$

un polinomio con $\alpha_i \in \mathbb{R}$ rappresentante la nostra approssimazione QUBO.

Tesi: Non esiste un'assegnamento di α_i e E tale che $f(s) = -E$ e $f(u) > E$, per ogni $s \in S_{\text{SAT}}$ e $u \in S_{\text{UNSAT}}$.

Dimostrazione: Poniamo per assurdo che possiamo scegliere $\alpha_i, E \in R$ tale che $f(s) = -E$ e $f(u) > -E$ per ogni $s \in S_{\text{SAT}}$ e $u \in S_{\text{UNSAT}}$. Per semplificare la dimostrzione, ipotizziamo di analizzare una clausola di tipo 0 (senza negazioni). Si nota che $100, 010, 001$ sono tutti assegnamenti che soddisfano la clausola, dunque $f(100) = f(010) = f(001) = -E$. Sostituiendo nel polinomio f

$$f(x_1, x_2, x_3) = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_2 x_3 + \alpha_{12} x_1 x_2 + \alpha_{13} x_1 x_3 + \alpha_{23} x_2 x_3$$

troviamo $\alpha_1 = \alpha_2 = \alpha_3 = -E$. Notiamo inoltre che 110 è un assegnamento appartenente a S_{SAT}, dunque come prima sostituiamo nel polinomio f

$$f(110) = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_{12} x_1 x_2 = -E \implies -E - E + \alpha_{12} = E \implies \alpha_{12} = E$$

In modo analogo si trova $\alpha_{23} = E, \alpha_{13} = E$. Consideriamo adesso l'assegnamento $111 \in S_{\text{SAT}}$. Sostituendo nel polinomio si trova

$$f(111) = -E - E - E + E + E + E = 0 \neq -E$$

Questa è una contraddizione in quanto avevamo ipotizzato $\forall s \in S_{\text{SAT}}, f(s) = -E$.

In una clausola MAX3-SAT ci sono sempre 3 variabili booleane, ciò significa che abbiamo 8 assegnamenti possibili per la clausola. Per ogni clausola 7 di questi 8 appartengono a S_{SAT} e uno appartiene a S_{UNSAT}. Il teorema (1.1) mostra che è possibile codificare fino a 6 delle 7 soluzioni nella funzione obbiettivo approssimata. Questo significa che esiste sempre una soluzione ottima che non minimizza $H_{\text{QUBO}}(x)$. In altre parole abbiamo ridotto le dimensioni della matrice Q e eliminato di conseguenza svariati termini quadratici al costo della perdita di una della 7 soluzioni ottime. Il guadagno in termini di riduzione di termini quadratici e dimensioni implica un minor numero di QUBIT utilizzati dal computer quantistico. Si ipotizza, per i motivi osservati a inizio capitolo, che il tradeoff sia complessivamente conveniente.

Sulla base di quanto osservato è possibile costruire un algoritmo che ricerchi esaustivamente nello spazio di matrici 3×3. La matrice 3×3 finale (output) deve codificare 6 delle 7 soluzioni ottime. Si nota che lo spazio su cui vogliamo ricercare non è discreto se assumiamo $\alpha_i \in \mathbb{R}$, risulta che è impossibile analizzare "tutti" gli assegnamenti. Si risolve ponendo $\alpha_i \in S$ con S finito.

```
Require: Set S of values the search method can insert into
the QUBO matrix as values

Require: Set A = {(a1,a2,a3,a12,a13,a23) in S^6} containing
all possible six tuples of values of S.


procedure SEARCH QUBO APPROXIMATION(clause_type)
    FoundQUBOS = {}
    for tuple in A do:
        if six satisfying assignments of clause of type
           clause_type have minimal energy then

            FoundQUBOS.insert(tuple)
        end if
    end for
    return FoundQUBOS
end procedure
```

L'algoritmo, come anticipato, esplora tutte le possibili funzioni obbiettivo con coefficienti in S. In base alla tipologia di clausola testiamo gli assegnamenti che la soddisfano, se 6 dei 7 hanno energia minima allora abbiamo trovato una delle matrici 3×3 adatte all'approssimazione QUBO per la clausola.

L'algoritmo trova 4 approssimazioni per ogni tipo di clausola. Le approssimazioni usate nel codice sono [4]:

- clausola di tipo 0:
$$\begin{bmatrix} -1 & 1 & 1 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \end{bmatrix}$$
- clausola di tipo 1:
$$\begin{bmatrix} 0 & 1 & -1 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$
- clausola di tipo 2:
$$\begin{bmatrix} 1 & -1 & -1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

- clausola di tipo 3:

$$\begin{bmatrix} -1 & 1 & 1 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \end{bmatrix}$$

Nel caso generale di una formula MAX3-SAT, si associa una matrice QUBO approssimata per clausola di tipo i a ogni clausola. Successivamente, si calcola la matrice finale sovrapponendo le matrici associate a ogni clausola presente nella formula.

Note:

- Le approssimazioni usate nel codice sono state trovate avendo fissato $S = \{-1, 0, 1\}$
- Pattern QUBO search può essere utilizzato per trovare trasformazioni esatte. Invece di esplorare lo spazio delle matrici 3×3, possiamo considerare matrici 4×4, in quanto, come mostrato nella trasformazione proposta da Chancellor, queste possono codificare tutte le soluzioni ottime. In questo caso, stiamo anche assegnando i coefficienti dei termini lineari e quadratici che coinvolgono le variabili ancilla.

2.3 Implementazione

```
for c = 1:size(clauses,1)      % we loop over the rows
    clause = clauses(c,:);     % we take the whole row

    % Extract variables and their signs
    vars = abs(clause);
    signs = clause > 0;


    % Determine clause type based on the number of nots
    num_neg = sum(~signs);

    switch num_neg
        case 0 % Type 1: (xi OR xj OR xk)
            local_Q = [-1,  1,  1;
                        0, -1,  1;
                        0,  0, -1];
            % order doesn't matter %
        case 1 % Type 2: (xi OR xj OR NOT xk)
            local_Q = [0,  1,  -1;
                       0,  0,  -1;
                       0,  0,   1];
            % we exchange literals wlog
            % we want the not literals at the end
            neg_indx = find(signs == false);
            if neg_indx ~= 3
                vars([neg_indx,3]) = vars([3, neg_indx]);
            end
        case 2 % Type 3: (xi OR NOT xj OR NOT xk)
            local_Q = [1, -1, -1;
                       0,  0,  1;
                       0,  0,  0];
            % we exchange literals wlog
            % we want the not literals at the end
            pos_indx = find(signs == true);
            if pos_indx ~= 1
                vars([1,pos_indx]) = vars([pos_indx, 1]);
            end

        case 3 % Type 4: (NOT xi OR NOT xj OR NOT xk)
            local_Q = [-1,  1,  1;
                        0, -1,  1;
                        0,  0, -1];
            % order doesn't matter
        otherwise
            error('Invalid clause type');
    end
```

Ogni clausola deve essere mappata all'approssimazione QUBO corrispondente. Si usa un vettore `vars` che memorizza per ogni clausola le varaibili booleane che essa contiene e un vettore `signs` che memorizza il segno di queste variabili. In base al numero di variabili negate le viene assegnata la Q_i corrispondente. Nel caso di clausola di tipo 1 o 2 dobbiamo spostare in fondo le varaibili negate (nel tipo 2 si sposta la varaibile non negata all'inizio).

```
    % Map local_Q to global QUBO matrix Q %
    % we know that is a UT matrix so we just need to check
    % the entries for i <= j
    for i = 1 : 3
        for j = i : 3
         vi = vars(i);
         vj = vars(j);
         % we take the entrance for the current clause
         global_entry = local_Q(i,j);
         %  Q has to be upper triangular
         if vj < vi
             % adding the contribute of the clause to the
             % global Q matrix
             Q(vj,vi) = Q(vj,vi) + global_entry;
         else
             Q(vi,vj) = Q(vi,vj) + global_entry;
         end
         end
    end
end
```

Ora sovrapponiamo la matrice locale a quella globale che è stata inizializzata a O_n. La matrice risultante è triangolare superiore e $n \times n$. Questa sarà la matrice da dare in input al quantum annealer.

Capitolo 3

QUBO Solvers

Una volta trasformato il problema MAX3-SAT nella sua forma QUBO (Quadratic Unconstrained Binary Optimization), il passo successivo consiste nell'individuare una strategia efficace per risolverlo, ovvero per trovare un'assegnazione binaria che minimizzi la funzione obiettivo quadratica associata.

Esistono diverse classi di metodi risolutivi per problemi QUBO, ciascuna con vantaggi e limiti specifici:

- Approcci esatti: comprendono tecniche di programmazione intera binaria (BIP) risolvibili con solutori come CPLEX o Gurobi. Questi metodi garantiscono soluzioni ottimali, ma diventano rapidamente impraticabili per istanze di grandi dimensioni a causa della complessità computazionale.
- Algoritmi euristici e metaeuristici classici: includono algoritmi genetici, tabu search, greedy randomized adaptive search procedure (GRASP), e local search. Queste tecniche non garantiscono l'ottimalità ma offrono soluzioni approssimate in tempi contenuti, con buona efficacia su istanze complesse.
- Metodi basati su rilassamenti continui, come la programmazione quadratica continua o semidefinite programming (SDP), che offrono approssimazioni efficienti della soluzione binaria rilassando il dominio delle variabili.
- Ottimizzazione quantistica, in particolare il quantum annealing, che rappresenta la motivazione teorica principale per la trasformazione in forma QUBO. Tali metodi sfruttano dispositivi fisici (come i quantum annealer di D-Wave) per esplorare in parallelo lo spazio delle soluzioni sfruttando effetti quantistici come il tunneling.

Nonostante il legame diretto tra QUBO e quantum annealing, in questo lavoro si è optato per un approccio alternativo, più accessibile e riproducibile in ambiente classico: il simulated annealing.

3.1 Simulated Annealing

Il Simulated Annealing (SA) pianta le sue radici in ambito metallurgico. Il processo di annealing, in metallurgia, consiste nel riscaldare un materiale e poi raffreddarlo lentamente per modificarne le proprietà fisiche e chimiche. Questo trattamento viene effettuato per aumentarne la duttilità e ridurne la durezza, rendendolo così più lavorabile.

Simulated Annealing è un algoritmo probabilistico che si ispira a questo processo fisico per esplorare lo spazio degli assegnamenti possibili e cercare un assegnamento ottimo. L'idea alla base è semplice: si parte da un assegnamento iniziale x_i, al quale è associata un'energia, calcolata nel nostro caso tramite la funzione obiettivo $H_{\text{QUBO}}(x_i)$.

A ogni passo, l'algoritmo esplora gli assegnamenti "vicini" modificando leggermente quello corrente, cambiando (flippando) il valore di una variabile. Se il nuovo assegnamento trovato ha un'energia più bassa ($H_{\text{QUBO}}(x_{\text{new}}) \leq H_{\text{QUBO}}(x_i)$), viene accettato come nuovo punto di partenza $x_i = x_{\text{new}}$. Se invece x_{new} ha un'energia più alta, potremmo essere tentati di scartarlo. Tuttavia, facendo così si rischia di rimanere bloccati in minimi locali, ossia soluzioni sub-ottimali che sembrano buone solo nel loro intorno ristretto.

Per evitare questo blocco, l'algoritmo può decidere di accettare temporaneamente soluzioni peggiori, seguendo un criterio probabilistico che dipende dalla differenza di energia e dalla temperatura (duttilità nella ricerca). Questo criterio si chiama criterio di Metropolis.

3.1.1 Criterio di Metropolis

Per decidere se accettare o meno un nuovo assegnamento con energia più alta rispetto a quella corrente, il Simulated Annealing utilizza una regola chiamata criterio di Metropolis. L'idea è che, anche se il nuovo stato è peggiore, può comunque essere accettato con una certa probabilità decrescente, che dipende da:

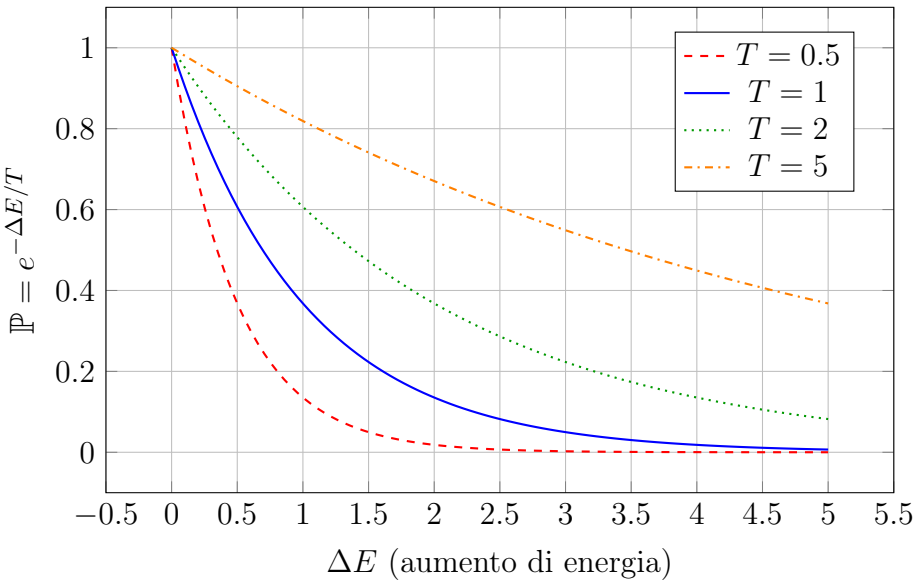
- quanto peggiore è il nuovo stato (cioè dalla differenza di energia ΔE)
- dalla temperatura (duttilità nella ricerca), che descresce nel tempo. T

La probabilità di accettare il nuovo assegnamento e cambiare stato è:

$$\mathbb{P}(x_i = x_{\text{new}}) = e^{-\frac{\Delta E}{T}}$$

Vediamo coma cambia la probabilità al descrescere della temperatura.

Probabilità di accettazione al variare della temperatura



Si nota che per temperature alte, anche per grandi sbalzi di energia, la probabilità di accettazione è molto alta. Per temperature basse la probabilità di accettazione è bassa anche per sbalzi di energia minimi. Dunque per temperature alte l'algoritmo è più tollerante, mentre al descrescere della temperatura l'algoritmo diventa man mano più severo. L'algoritmo si evolve in due fasi differenti, la prima, quella iniziale, durante la quale cerca di esplorare il più possibile lo spazio degli assegnamenti possibili (caratterizzato da temperature più alte) e la seconda (caratterizzato da temperature più basse) in cui l'algoritmo cerca di convergere verso un minimo (che speriamo essere globale).

3.1.2 Catene di Markov e SA

Definizione (3.1 | Catena di Markov)
Una catena di Markov è un processo stocastico a tempo discreto che evolve su uno spazio di stati finito o numerabile S, in cui la probaiblità di transizione da uno stato all'altro dipende solo dallo stato attuale e non dalla storia passata del processo. Questa proprietà è detta proprietà di Markov.
Formalmente, una catena di Markov, è definita da:

- uno spazio degli stati S
- una matrice di transizione $P = [P_{ij}]$ tale che:

$$P_{ij} = \mathbb{P}(X_{t+1} = j \mid X_t = i)$$

dove $X_t \in S$ rappresenta lo stato al tempo t, e:

$$\sum_{j \in S} P_{ij} = 1 \quad \forall i \in S$$

L'algoritmo del simulated annealing su QUBO, può essere descritto come una catena di Makrov definita sullo spazio degli assegnamenti binari del problema $S = \mathbb{F}_2^n$. A ogni passo, viene generato uno stato candidato "vicino" (flippando il valore di una variabile booleana), e si decide se accettarlo secondo una certa probabilità. Questa dinamica, genera una matrice di transizione P_T dipendente dalla temperatura.

$$P_T(x_i \to x_{\text{new}}) = \begin{cases} 1 & \text{se } H(x_{\text{new}}) \leq H(x_i) \\ \exp\left(-\frac{H(x_{\text{new}}) - H(x_i)}{T}\right) & \text{se } H(x_{\text{new}}) > H(x_i) \end{cases}$$

$\sum_{j \in S} P_{ij} = 1 \quad \forall i \in S$ non sarebbe soddisfatta usando questo modello, dovremmo considerare la probabilità che x_{new} sia stato generato $G(x_i \to x_{\text{new}})$.

- se dovessi flippare più di una variabile booleana per raggiungere x_{new} allora $G(x_i \to x_{\text{new}}) = 0$.
- In caso contrario avremmo $G(x_i \to x_{\text{new}}) = \frac{1}{n}$ dove n è il numero di variabili booleane (G è uniforme per x_{new} vicini).

Supponendo che G sia indipendente rispetto a $P_{T\text{old}}$:

$$P_T(x_i \to x_{\text{new}}) = G(x_i \to x_{\text{new}}) \cdot P_{T\text{old}}(x_i \to x_{\text{new}})$$

Per garantire che la catena sia ben definita dovremmo includere anche la possibilità di rimanere nello stesso stato, vediamo perchè:

$$\sum_{x_{\text{vicino}(i)}} \frac{1}{n} P_{T\text{old}}(x \to x_{\text{vicino}(i)})$$

L'ipotesi è confermata se e solo se tutti gli stati vicini hanno energia minore o uguale allo stato iniziale, questo in generale non è sempre vero. Si risolve osservando che la probaiblità di rifiutare l'assegnamento è:

$$P_T(x \to x) = \sum_{x_{\text{vicino}(i)}} G(x \to x_{\text{vicino}(i)}) \cdot (1 - P_{T\text{old}}(x \to x_{\text{vicino}(i)}))$$

che implica

$$\sum_j P_{ij} = \frac{1}{n} \sum_{x_{\text{vicino}(i)}} P_{T\text{old}}(x \to x_{\text{vicino}(i)}) + \frac{1}{n} \sum_{x_{\text{vicino}(i)}} 1 - P_{T\text{old}}(x \to x_{\text{vicino}(i)}) = 1$$

La catena di Markov è ora ben definita.

3.1.3 Cooling schedule

Uno degli aspetti fondamentali del Simulated Annealing è il modo in cui la temperatura T viene diminuita nel tempo, ovvero il cooling schedule. Questa temperatura controlla la probabilità di accettare soluzioni peggiori, e quindi influenza il comportamento esplorativo dell'algoritmo.

All'inizio, con temperature alte, è più facile accettare soluzioni peggiori: questo aiuta ad esplorare lo spazio delle soluzioni ed evitare minimi locali. Man mano che la temperatura si abbassa, l'algoritmo diventa sempre più selettivo e tende a "bloccarsi" su una soluzione.

Il risultato teorico importante alla base del Simulated Annealing è che, se la temperatura viene abbassata abbastanza lentamente (cioè secondo un cooling schedule sufficientemente lento), allora l'algoritmo converge con probabilità 1 alla soluzione ottima globale.[5]

Questo accade perché, come abbiamo visto, l'algoritmo può essere modellato come una catena di Markov con probabilità di transizione dipendenti dalla temperatura. Quando il cooling è sufficientemente lento, la catena ha il tempo di "equilibrarsi" a ogni temperatura, e alla fine si concentra sulle configurazioni con energia minima.

Ovviamente, in pratica non è possibile usare un cooling infinitamente lento: il tempo richiesto sarebbe troppo lungo. Per questo motivo, nella realtà si scelgono strategie di cooling più aggressive (ad esempio temperature che si dimezzano ogni tot passi), accettando un compromesso tra qualità della soluzione e tempo di calcolo.

Il codice utilizza una progressione geometrica decrescente, $T_k = \alpha T_{k-1}$. Dove α viene calcolato in base alla T_i temperatura iniziale, T_f temperatura finale e N_{iter} il numero di iterazioni.

$$T_f = \alpha^{N_{\text{iter}}} T_i \implies \alpha = \left(\frac{T_f}{T_i}\right)^{\frac{1}{N_{\text{iter}}}}$$

Il comportamento del simulated annealing è congruo a quanto descritto teoricamente, man mano che la temperatura descresce esplora sempre di meno.

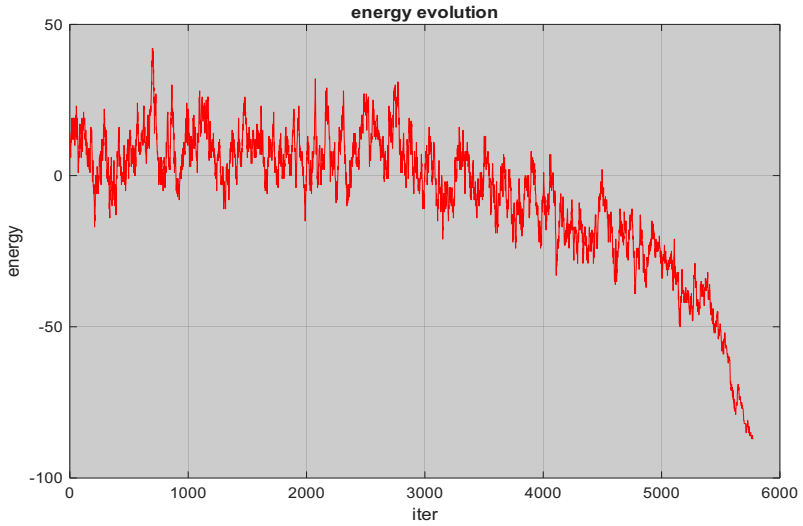
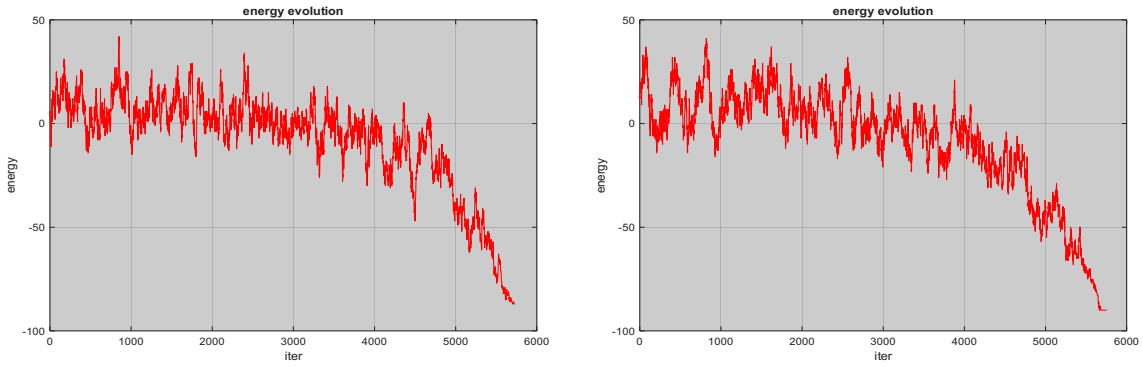


Figura 3.1: SA evolution (538-clauses 125-vars 10000-maxiter 100-initial temp 0.1-final temp)

3.1.4 Memoization

Sono state implementate e analizzate due varianti dell'algoritmo: con e senza memoizzazione, per confrontare l'efficacia nell'esplorazione.

```
% variable declarations
d = configureDictionary("string","double");
% code
%
% generate random neighbor_solution
% check dictionary
vettoreStr = num2str(neighbor_solution);
Str = convertCharsToStrings(vettoreStr);
if isKey(d,Str)
    continue;
end
% code
%
if delta_energy < 0 || rand() < exp(-delta_energy/temperature)
    d = insert(d,Str,neighbor_energy); % mark as accepted
```



(a) Memoization (b) No Memoization

Figura 3.2: Memoization vs No Memoization

La tecnica della memoization è risultata apprezzabile per istanze di piccola dimensione ($10 - 12$ vars). Per istanze grandi (> 20 vars), la versione senza memoization ha prodotto risultati più soddisfacenti. In figura troviamo un confronto tra i due simulated annealing sulla stessa istanza QUBO.

```
vector = int32([]);
% code
if delta_energy < 0 || rand() < exp(-delta_energy/temperature)
    vector(end+1) = neighbor_energy;
```

3.2 Implementazione

```
max_iter = 5000;
initial_temp = 100;
final_temp = 0.1;
alpha = (final_temp/initial_temp)^(1/max_iter);

% Initialize assignment randomly
current_solution = randi([0,1], n, 1);
current_energy = current_solution' * Q * current_solution;
best_solution = current_solution;
best_energy = current_energy;
temperature = initial_temp;
iter = 0;
while iter < max_iter
    % Generate neighbor by flipping a random bit
    neighbor_solution = current_solution;
    idx = randi(n);
    % flipping the random bit
    neighbor_solution(idx) = 1 - neighbor_solution(idx);
    % check dictionary
    vettoreStr = num2str(neighbor_solution);
    Str = convertCharsToStrings(vettoreStr);
    if isKey(d,Str)
        continue;
    end
    % energy of the qubo matrix
    neighbor_energy = neighbor_solution' * Q *
   neighbor_solution;
    % energy gap
    delta_energy = neighbor_energy - current_energy;

    % Metrpolis criterion
    if delta_energy < 0 || rand() < exp(-delta_energy/
   temperature)
        % we add the current energy to the bucket
        d = insert(d,Str,neighbor_energy);
        % we accepted the new energy as current
        current_solution = neighbor_solution;
        current_energy = neighbor_energy;
        if current_energy < best_energy
            best_solution = current_solution;
            best_energy = current_energy;
        end
    end
    % Decrease temperature
    temperature = temperature * alpha;
    iter = iter + 1;
end
```

```
% plotting of the energy
E = entries(d);
V = E.Value;
plot(0:(length(V)-1),V,'-r');
xlabel('iter');
ylabel('energy');
title('energy evolution');
% this represents the evolution of the simulated annealing
```

Il codice riportato sopra mostra l'implementazione con memoization. In particolare, l'approccio adottato consiste nell'evitare di rianalizzare soluzioni già accettate, al fine di prevenire la formazione di cicli e ridurre il numero di computazioni ridondanti. Al termine dell'algoritmo, viene tracciato l'andamento dell'energia delle soluzioni accettate nel corso delle iterazioni.

Nota: L'attuale implementazione fa un uso piuttosto significativo della memoria. Un possibile miglioramento potrebbe consistere nel limitare il numero di soluzioni memorizzate, ad esempio conservando solo le N peggiori, riducendo di fatto la dimensione della cache e il consumo di memoria. La versione priva di memoization risulta naturalmente più leggera dal punto di vista dell'uso della memoria, poiché non richiede l'impiego di alcuna struttura dati. In questa variante, infatti, il vettore è stato utilizzato esclusivamente per comodità in fase di visualizzazione dell'evoluzione.

Capitolo 4

Valutazione della soluzione

4.1 Implementazione

Lo script riporta infine il numero di clause soddsfatte e l'assegnamento trovato tramite le variabili `solution` e `satisfied_clauses`. Il calcolo necessario per la determinazione della seconda variabile viene fatto come segue:

```
% we recall the best solution marked by SA
solution = best_solution;

% Evaluate the number of satisfied clauses
satisfied_clauses = 0;
for c = 1:size(clauses,1)
    clause = clauses(c,:);
    vars = abs(clause);
    signs = clause > 0;
    % selecting the correct part of the assignment for the
   clause
    vals = solution(vars);
    clause_satisfied = any(vals == signs'); % we check if the
   clause is staisfied
    if clause_satisfied
        satisfied_clauses = satisfied_clauses + 1;
    end
end
% print the results %
fprintf('Number of satisfied clauses: %d out of %d\n',
   satisfied_clauses, size(clauses,1));
end
```

4.2 Benchmarks

Lo script è stato testato su formule 3-CNF con 20 variabili booleane, 91 clausole (S) e 125 variabili booleane, 538 clausole (L). Le istanze (S) sono state tutte analizzate con numero massimo di iterazioni = 5000;

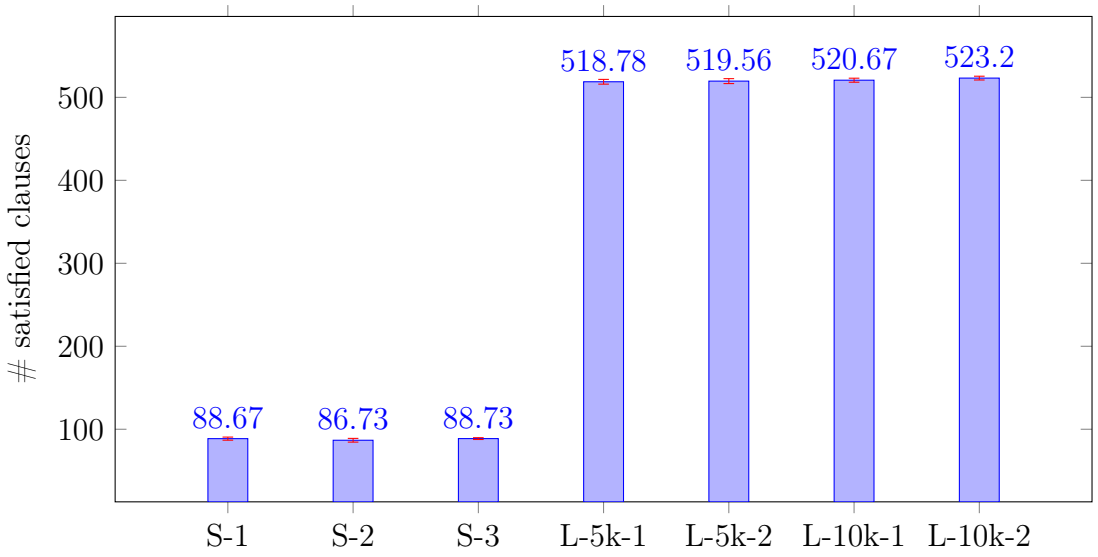


Figura 4.1: Media delle clausole soddisfatte con stdev

Appendice A

Matrici risultanti dalla trasformazione di Chanchellor

Si elencano di seguito le matrici risultanti dal procedimento associato alla trasformazione di Chanchellor, descritto nel capitolo 2 della tesina. Si ricorda che, per queste rappresentazioni QUBO, è supposto che le variabili negate siano in fondo alla clausola.[4]

- Clausola di tipo 0:

$$\begin{bmatrix} -2 & 1 & 1 & 1 \\ 0 & -2 & 1 & 1 \\ 0 & 0 & -2 & 1 \\ 0 & 0 & 0 & -2 \end{bmatrix}$$

- Clausola di tipo 1:

$$\begin{bmatrix} -1 & 1 & 0 & 1 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

- Clausola di tipo 2:

$$\begin{bmatrix} -1 & 0 & 0 & 1 \\ 0 & -1 & 1 & 1 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

- Clausola di tipo 3:

$$\begin{bmatrix} -1 & 1 & 1 & 1 \\ 0 & -1 & 1 & 1 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

Il parametro libero $J \geq 1$ è stato posto $= 1$, si consiglia inoltre di moltiplicare per 8 la formula $(*)$ presentata nella sezione 2 per usare la formula $(\#)$ senza incongruenze.

Bibliografia

[1] Zielinski S., Nüßlein J., Stein J., Gabor T., Linnhoff-Popien C., and S. Feld. "algorithmic construction of 3sat-to-qubo transformations". *Electronics*, 12:3492, 2023. doi:10.3390/electronics12163492.

[2] Chancellor N., Zohren S., Warburton P.A., Benjamin S.C., and Roberts S. A. "direct mapping of max k-sat and high order parity checks to a chimera graph". *Scientific Reports*, 6:37107, 2016. doi:10.1038/srep37107.

[3] Munch C., Schinkel F., Zielinsky S., and Walter S. "transformation-dependent performance-enhancement of digital annealer for 3-sat". *arXiv preprint*, 2023. doi:10.48550/arXiv.2312.11645.

[4] Feld S., Nusslein J., Gabor T., Linnhoff-Popien C., Zielinski S., and Kolle M. "solving max-3sat using qubo approximation". *arXiv preprint*, 2024. doi:10.48550/arXiv.2409.15891.

[5] Granville V., Krivanek M., and Rasson J.-P. "simulated annealing: A proof of convergence". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6):652–656, 1994. doi:10.1109/34.295910.