

Progetto OC - *Max-Flow* QUBO

Nicola Barbaro (1070668)

Mario Bifulco (881727)

A.A. 2022/2023

Indice

1	Problema del flusso massimo	2
1.1	Descrizione generale del problema	2
1.2	Minimum Cut, Teorema <i>Max-Flow Min-Cut</i>	3
1.3	Riformulazione su programmazione lineare primale-duale	4
2	QAC e formulazione QUBO	6
2.1	Quantum Adiabatic Computing	6
2.2	Quadratic Unconstraint Binary Optimization	7
2.3	Simulated Annealing	8
3	Min Cut come problema QUBO	10
3.1	Inquadramento del problema	10
3.2	Conversione delle variabili di slack	10
3.3	Rilassamento Lagrangiano	11
4	Implementazione	12
4.1	Test	13
4.2	Risultati sperimentali	13

1 Problema del flusso massimo

1.1 Descrizione generale del problema

Introdotta da Harris e Ross nel 1954, il problema del Flusso Massimo (Maximum Flow Problem) richiede di trovare, sulla struttura dati chiamata Rete di Flusso, un flusso plausibile con il rapporto più alto possibile (chiamato, appunto *Flusso Massimo*). Il MF è spesso usato per risolvere problemi legati al trasporto di beni o di instradamento di informazioni su una rete.

Si vuole definire innanzitutto la connotazione della Rete di Flusso. Sia:

- $G = (V, E)$ una rete (rappresentata come un grafo) con $s, t \in V$ rispettivamente la *fonte* e la *foce* di G .
- Dato $(i, j) \in E$ uno qualsiasi spigolo della rete, la sua **capacità** è definita come la massima quantità di flusso che può passare per esso. Il suo valore è definito dalla funzione $c : E \rightarrow \mathbb{R}^+$.

Definizione. Un generico flusso su una rete $G = (V, E)$ è una funzione $f : E \rightarrow \mathbb{R}$ se sono soddisfatti i seguenti:

- *Vincoli di capacità.* Per ogni $(i, j) \in E$, il flusso associato ad esso non supererà mai la sua capacità, ossia $x_{ij} \leq u_{ij}$.
- *Conservazione dei flussi.* Con sola esclusione del nodo fonte s e foce t , la somma dei flussi in entrata in un nodo deve uguagliare la somma dei flussi in uscita dallo stesso nodo. In formula:

$$\forall v \in V \setminus \{s, t\} : \sum_{i:(i,j) \in E} x_{ij} = \sum_{i:(j,i) \in E} x_{ji}$$

Si può osservare come la matrice di adiacenza del flusso G è antisimmetrica: infatti $x_{ij} = -x_{ji}$, $\forall (i, j) \in E$.

Definizione. Il **valore** del flusso è la quantità di flusso passante sulla rete a partire dalla fonte s e terminante nella foce t . Rappresentato in termini formali, un flusso $f : E \rightarrow \mathbb{R}^+$ è una funzione il cui dominio è dato da:

$$|x| = \sum_{j:(s,j) \in E} x_{sj} - \sum_{i:(i,s) \in E} x_{is}$$

A questo punto è possibile fornire una introduzione formale del problema attraverso la seguente:

Definizione. Il **Problema del Flusso Massimo** è il problema di instradare quanto più flusso possibile dalla fonte alla foce, ossia di trovare il flusso con massimo valore x_{max} instradabile sulla rete.

Il problema del flusso massimo ammette unica o infinite soluzioni, dato che ci sono infinite combinazioni lineari possibili del valore x_{max} di partenza.

1.2 Minimum Cut, Teorema *Max-Flow Min-Cut*

Nella teoria dei grafi, un **minimum cut** per un grafo è una partizione dei vertici del grafo in due insiemi disgiunti minimale rispetto ad una qualunque metrica definita dal problema. Nella specifica del problema di Flusso Massimo, su rete G , è interessante porre l'attenzione sull'insieme di tagli chiamati **tagli s-t** $C = (S, T)$: questo insieme è una suddivisione dei vertici V di G tale che il nodo fonte $s \in S$ e il nodo foce $t \in T$, ossia una ripartizione dei vertici della rete in due parti, con la fonte in una parte e la foce in un'altra. Si definisce il **cut-set** X_C di un taglio C l'insieme degli spigoli che connettono i vertici $i \in S$ ai vertici $j \in T$:

$$X_C := \{(i, j) \in E : i \in S, j \in T\} = (S \times T) \cap E$$

È immediato notare che togliere gli spigoli di X_C da E renderebbe impossibile la computazione di qualsiasi flusso a valore positivo, perché non ci sarebbero più cammini dalla fonte alla foce percorribili.

La **capacità** di un taglio $s - t$ è dato dalla somma delle capacità degli spigoli in X_C :

$$c(S, T) = \sum_{(i, j) \in X_C} u_{ij} = \sum_{(i, j) \in E, i \in S, j \in T} u_{ij}$$

Il problema del **taglio s-t minimale** prevede di minimizzare $u(S, T)$, ossia di determinare S, T tali che la capacità del taglio $s - t$ sia minimale.

Teorema *Max-Flow Min-Cut* (1). Il massimo valore di un flusso $s - t$ è uguale al taglio $s - t$ di capacità minima tra tutti i possibili tagli.[2]

1.3 Riformulazione su programmazione lineare primale-duale

È possibile rappresentare il problema del Flusso Massimo come un programma lineare primale il cui duale è il programma lineare associato al problema di Minimum Cut.

Formulazione Primale (a).

$$\text{massimizza} \quad \sum_{(s,i) \in FS_i} x_{si} = x_{ts} \quad (1)$$

$$\text{soggetto a} \quad \sum_{(h,i) \in BS_i} x_{hi} - \sum_{(i,h) \in FS_i} x_{ij} = 0 \quad \forall i \in V \setminus \{s, t\} \quad (2)$$

$$\left(\sum_{(i,t) \in BS_t} x_{it} \right) - x_{ts} = 0 \quad (3)$$

$$x_{ts} - \left(\sum_{(s,i) \in FS_s} x_{si} \right) = 0 \quad (4)$$

$$0 \leq x_{ij} \leq u_{ij} \quad \forall (i, j) \in E \quad (5).$$

I vincoli hanno la seguente interpretazione:

1. Si vuole massimizzare il flusso su un arco *dummy*, senza capacità, che va dalla foce t alla fonte s ;
2. Vincoli che permettono di rispettare la *conservazione dei flussi*;
3. Il flusso massimo trovato dal problema deve combaciare con la somma dei flussi entranti nella foce
4. Il flusso massimo trovato dal problema deve combaciare con la somma dei flussi uscenti dalla fonte
5. Bisogna rispettare il *vincolo di capacità*.

Il duale di un programma lineare primale è definibile attraverso il seguente procedimento:

- Ogni variabile del primale diventa un vincolo del duale;
- Ogni vincolo del primale diventa una variabile del duale;
- Va invertita la funzione obiettivo (da massimo a minimo, o viceversa).

Secondo questo algoritmo, possiamo ottenere a partire da **(a)**, la seguente: **Formulazione Duale (b).**

$$\text{minimizzare} \quad \sum_{(i,j) \in E} \omega_{ij} u_{ij} \quad (1)$$

$$\text{soggetto a} \quad \pi_t - \pi_s \geq 1 \quad (2)$$

$$\pi_i - \pi_j + \omega_{ij} \geq 0 \quad \forall (i, j) \in E \quad (3)$$

$$\omega_{ij} \geq 0 \quad \forall (i, j) \in E \quad (4).$$

Ove:

$$\omega_{ij} = \begin{cases} 1 & \text{se } i \in S \text{ e } j \in T, \text{ ossia lo spigolo } (i,j) \text{ appartiene al taglio } X_C \\ 0 & \text{altrimenti} \end{cases} \quad (1)$$

$$\pi_i = \begin{cases} 1 & \text{se } i \in T \\ 0 & \text{altrimenti} \end{cases} \quad (2)$$

Anche in questo caso è possibile fornire una interpretazione dei vincoli e della funzione obiettivo:

1. La funzione obiettivo vuole minimizzare la capacità totale degli spigoli nel taglio;
2. Il vincolo si assicura che la fonte sia nell'insieme S e la foce sia nell'insieme T ;
3. Garantisce che, per ogni nodo i, j diverso dalla fonte o dalla foce, se $i \in S$ e $j \in T$, allora lo spigolo (i, j) deve essere considerato nel taglio (e quindi $\omega_{ij} \geq 1$);
4. triviale.

Si può osservare come il problema non garantisce quali spigoli non siano nel taglio, ma può solo garantire che, se esso *può* ricadere nel taglio, allora la sua capacità deve essere sommata nella funzione obiettivo: è deducibile che ogni possibile taglio $s - t$ vede la sua capacità come una soluzione ammissibile per **(b)**, e, dalla teoria, ogni soluzione di **(b)** è un limite superiore per **(a)**, e quindi per il valore del flusso che stiamo cercando di massimizzare.

Dalla teoria della programmazione lineare, è noto il

Teorema della dualità forte (2). Dato uno programma lineare primale P, se esso ammette soluzione ottimale x^* , allora anche il programma lineare duale D associato a P ammette soluzione ottima y^* , e in particolare si riscontra $y^* = x^*$.

Segue perciò che l'uguaglianza definita in (1) è data espressamente dal teorema (2), e che risolvere il problema del Minimum Cut fornisce una soluzione ottima per il problema del Flusso Massimo.

2 QAC e formulazione QUBO

2.1 Quantum Adiabatic Computing

La risoluzione di problemi intrattabili in modo efficiente utilizzando il calcolo classico richiede la dimostrazione dell'uguaglianza $P = NP$. Tuttavia, al momento attuale, questa dimostrazione rimane un problema aperto.

Un'alternativa per affrontare tali problemi è sfruttare modelli di calcolo non convenzionali che utilizzano principi di calcolo differenti. Tra questi modelli, si trovano le architetture basate sulla meccanica quantistica.

Attualmente, i due principali approcci per la computazione quantistica sono la programmazione tramite gate, concettualmente simile alla programmazione tramite porte logiche, e la programmazione adiabatica quantistica (QAC). Quest'ultimo approccio si basa sulla ricerca di stati a minore energia per trovare la soluzione desiderata.

Nel modello di calcolo descritto da QAC, si utilizza una matrice Hermitiana con elementi complessi (\mathbb{C}). Gli Hamiltoniani, utilizzati per eseguire la computazione, rappresentano il livello di energia istantaneo di un sistema fisico e descrivono l'evoluzione del suo stato.

L'idea fondamentale è sfruttare le proprietà fisiche per cercare stati a minore energia e quindi più stabili.

La matrice Hamiltoniana è un operatore in uno spazio vettoriale in cui le configurazioni delle variabili sono rappresentate come vettori, che rappresentano gli autovettori della matrice, e i valori assunti dalla funzione da minimizzare corrispondono agli autovalori di \mathcal{H}_P .

Per risolvere problemi espressi tramite la programmazione lineare utilizzando un Quantum CPU, è necessario convertirli in una formulazione QUBO (Quantum Unconstrained Binary Optimization), che presenta una naturale rappresentazione grafica in forma di grafo, successivamente mappato nel grafo dei qubit.

La mappatura viene effettuata mediante l'utilizzo dell'algoritmo *minor embedding* che preserva le relazioni aggiungendo eventualmente nuovi nodi se necessario. Pur appartenendo alla classe NP-complete tale algoritmo, viene risolto in modo efficiente nel caso medio, rendendolo quindi valido nell'utilizzo pratico.

Ci sono alcuni punti critici da considerare in questo approccio computazionale:

- È difficile fornire un limite inferiore al tempo di annealing (il tempo minimo necessario per raggiungere lo stato finale).
- In assenza di tale limite inferiore, la formulazione QUBO può essere considerata come un'euristica e non è possibile valutare in generale l'efficienza e la correttezza dell'algoritmo.
- Questo approccio non è valido in modo generale, poiché ci sono casi in cui gli algoritmi classici mostrano un comportamento lineare, mentre l'implementazione quantistica può degradare all'aumentare delle dimensioni del problema.

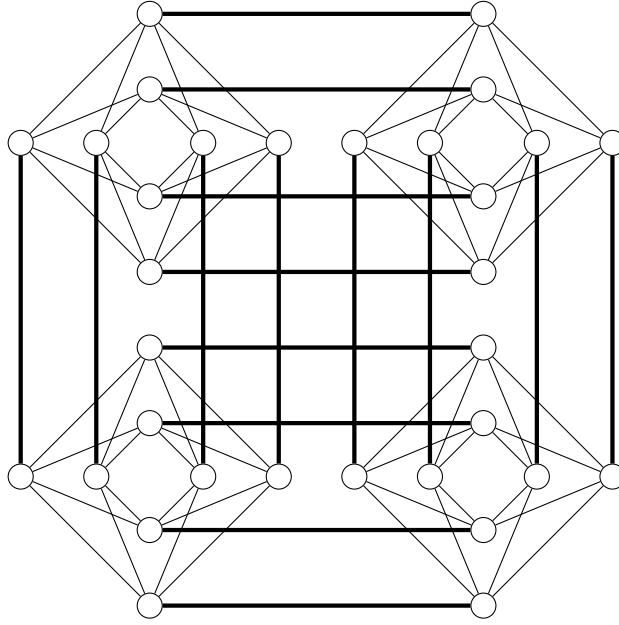


Figura 1: Esempio di QPU a 32 qubit[5]

2.2 Quadratic Unconstraint Binary Optimization

Il problema QUBO, Quadratic Unconstrained Binary Optimization[3], consiste nell'individuare il vettore $x^* = (x_1^*, \dots, x_n^*)$ che minimizza l'equazione $\bar{x}^T Q \bar{x}$, dove \bar{x} è un vettore colonna di variabili binarie e Q è una matrice triangolare superiore che rappresenta i coefficienti della funzione da minimizzare.

In particolare, l'obiettivo del problema QUBO è determinare il valore delle variabili binarie x_i che minimizza l'espressione $\sum_{i=1}^n \sum_{j=1}^n Q_{ij} x_i x_j$, dove Q_{ij} sono gli elementi della matrice Q .

$$\text{minimizzare } \underbrace{\begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix}}_{\bar{x}^T} \underbrace{\begin{bmatrix} a_1 & \cdots & a_n \\ \vdots & \ddots & b_n \\ 0 & \cdots & c_n \end{bmatrix}}_Q \underbrace{\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}}_{\bar{x}}$$

È importante notare che nella funzione obiettivo espansa, è possibile riscrivere i termini quadratici, cioè le variabili al quadrato, utilizzando il loro equivalente lineare. Questa riscrittura è possibile grazie al dominio delle variabili x_i , che assumono solo i valori 0 o 1. Pertanto, il termine x_i^2 può essere sostituito con x_i nella formulazione del problema QUBO.

Il formalismo dei modelli QUBO offre un approccio flessibile per la risoluzione di problemi di ottimizzazione combinatoria mediante l'utilizzo di variabili

binarie e l'espressione quadratica della funzione obiettivo. Questo formalismo è di particolare importanza nella progettazione e nell'implementazione di algoritmi e processori quantistici adiabatici per la risoluzione efficiente di problemi complessi.

2.3 Simulated Annealing

L'algoritmo *Simulated Annealing* fornisce una prospettiva intuitiva sul funzionamento di un processore quantistico adiabatico, QPU.

L'obiettivo di questo algoritmo è raggiungere lo stato finale attraverso un cammino casuale, definito *Random Walk*, esplorando tutti i possibili stati del sistema. La selezione dei passi da compiere è influenzata dalla probabilità che varia nel tempo e determina se accettare o meno le variazioni energetiche del sistema fisico su cui si sta operando.

Nell'implementazione di questo algoritmo, vengono utilizzati due attributi fondamentali: l'insieme degli stati adiacenti a un dato nodo e l'energia associata a tale nodo. Questi parametri influenzano la sequenza di passi che porta al cammino e, dunque, allo stato finale.

Durante l'esecuzione dell'algoritmo, un parametro cruciale è rappresentato dalla temperatura che può decrescere secondo diverse funzioni, più o meno rapidamente. La diminuzione della temperatura comporta una minor propensione al cambio di stato, fino alla stabilizzazione del sistema sulla risposta.

Formalizzazione dell'algoritmo *Simulated Annealing*

Sia $G = (V, E)$ un grafo rappresentante il sistema fisico di interesse, dove V rappresenta l'insieme dei nodi e E l'insieme degli archi. Ogni nodo $v \in V$ ha associata un'energia $E(v)$, mentre l'insieme degli adiacenti di un nodo v è indicato con $N(v)$.

L'algoritmo *Simulated Annealing* può essere descritto attraverso i seguenti passaggi:

1. Inizializzazione: si assegna a ogni nodo del grafo una configurazione casuale iniziale e si fissa una temperatura iniziale T .
2. Ciclo principale:
 - (a) Scelta di un nodo v in modo casuale dal grafo.
 - (b) Selezione di un nodo u in modo casuale dall'insieme $N(v)$.
 - (c) Calcolo della variazione di energia $\Delta E = E(u) - E(v)$.
 - (d) Se $\Delta E \leq 0$, si accetta la mossa e si aggiorna lo stato del sistema considerando u come nuovo stato corrente.
 - (e) Se $\Delta E > 0$, si accetta la mossa con una probabilità $p = \exp(-\frac{\Delta E}{T})$. In caso di accettazione, si aggiorna lo stato del sistema come nel passo precedente.

- (f) Ripeti i passi da (a) a (e) fino a soddisfare una condizione di terminazione.

Durante l'esecuzione dell'algoritmo, la temperatura T viene ridotta nel tempo seguendo una specifica funzione, che dipende dal problema e dall'obiettivo dell'ottimizzazione.

Tale algoritmo rappresenta un importante strumento per l'esplorazione degli spazi di ricerca complessi e ha trovato applicazioni significative nella risoluzione di problemi di ottimizzazione e nell'ambito della computazione quantistica adiabatica.

È importante notare, inoltre, che l'algoritmo può essere implementato anche su processori quantistici adiabatici, sfruttando le peculiarità della computazione quantistica per ottenere potenziali vantaggi prestazionali.

3 Min Cut come problema QUBO

3.1 Inquadramento del problema

Come visto nella sezione 1.2 il problema *Min-Cut* è espresso come:

$$\begin{array}{c|l} \text{minimizzare} & \text{soggetto a} \\ \sum_{(i,j) \in E} \omega_{ij} u_{ij} & \begin{array}{l} \pi_t - \pi_s - 1 = 0 \\ \pi_i - \pi_j + \omega_{ij} - s_2 = 0, \forall (i, j) \in E \\ s_2 \geq 0, w_{ij} \in \{0, 1\}, \pi_i \in \{0, 1\} \end{array} \end{array}$$

Nel quale il primo vincolo è stato trasformato in uguaglianza, in quanto un assegnamento valido non sarà mai diverso da zero, e al secondo vincolo è stata aggiunta un'apposita variabile di slack.

Dove la funzione obiettivo è la somma delle capacità degli archi del taglio, mentre i vincoli garantiscono che il taglio sia valido.

L'obiettivo della riformulazione nel problema QUBO è ottenere un'unica equazione \mathcal{H}_P composta solo da variabili binarie e che incorpori sia la funzione obiettivo che i vincoli del problema.

3.2 Conversione delle variabili di slack

A differenza delle variabili originali del problema, le variabili di slack non hanno la garanzia di assumere valori binari. Pertanto, è necessario trasformare opportunamente tali variabili per poterle esprimere unicamente attraverso variabili binarie.

La strategia adottata consiste nel convertire ogni variabile di slack utilizzando la relativa rappresentazione in codice binario. I coefficienti delle variabili così introdotte saranno le rispettive potenze di due, consentendo così di rappresentare ogni possibile valore della variabile di slack originale.

Per aggiungere un numero appropriato di variabili binarie, è necessario stimare il valore massimo che le variabili di slack possono assumere. A tal fine, si analizza il limite superiore per ciascuno dei vincoli del problema originale.

Nella riformulazione si useranno le variabili y_i^j , dove l'indice i indica il vincolo di riferimento e l'indice j rappresenta la potenza di due corrispondente. Pertanto, la variabile composta, moltiplicata per il suo coefficiente, sarà: $2^j y_i^j$.

Secondo vincolo Riformulando il secondo vincolo come $s_2 = \pi_i - \pi_j + \omega_{ij}$, il valore massimo si ottiene istanziando π_i a uno, π_j a zero e ω_{ij} a uno. In questo modo si ottiene un upper bound pari a due, che viene riformulato come $y_2^0 + 2y_2^1$.

3.3 Rilassamento Lagrangiano

In seguito alla conversione dei vincoli finalizzata a scriverli esclusivamente tramite variabili binarie, è necessario procedere alla riformulazione completa del problema al fine di consentirne l'espressione come un'unica equazione.

A tale scopo è utile applicare una tecnica di rilassamento al problema *Min-Cut*. In particolare, mediante l'uso del rilassamento Lagrangiano, è possibile convertire i vincoli in funzioni penalità che vengono sommate alla funzione obiettivo. Nel problema affrontato le funzioni penalità vengono sommate poiché si sta lavorando su un problema di minimizzazione. Nel caso di problemi di massimo, le penalità verrebbero sottratte.

È importante notare che l'applicazione del rilassamento al problema originale porta all'introduzione di istanziazioni inconsistenti delle variabili, ovvero casi in cui le penalità introdotte non sono sufficientemente elevate e la soluzione minima ottenuta non soddisfa i vincoli originali del problema.

Al fine di evitare tali situazioni, le funzioni penalità vengono moltiplicate per un opportuno parametro, noto come moltiplicatore Lagrangiano. Il suo scopo è garantire che le penalità influiscano in modo significativo sulla formulazione rilassata del problema, in modo da *forzare* l'assegnamento di valori validi che minimizzino la funzione obiettivo.

La riformulazione del problema in forma QUBO diventa quindi:

$$\mathcal{H}_P = \sum_{(u,v) \in E} \omega_{ij} u_{ij} + \lambda(\pi_t - \pi_s - 1)^2 + \lambda \sum_{(i,j) \in E} (\pi_i - \pi_j + \omega_{ij} - y_2^0 - 2y_2^1)^2$$

Dove λ rappresenta il moltiplicatore Lagrangiano. L'obiettivo è trovare un valore di λ adeguato che garantisca che le penalità influiscano in modo significativo sull'assegnamento delle variabili, al fine di ottenere la soluzione ottima anche per il problema non rilassato.

4 Implementazione

Per valutare l'efficacia della riformulazione proposta e dei metodi di calcolo quantistico allo stato dell'arte, l'algoritmo per il problema del taglio minimo è stato implementato utilizzando il linguaggio di programmazione Python e le librerie fornite da D-Wave Systems per interagire con la QPU[6].

La libreria NetworkX[1] è stata utilizzata per confrontare i risultati ottenuti tramite l'algoritmo in formulazione QUBO con quelli derivabili da approcci classici. Ciò ha permesso di gestire grafi e flussi su reti in modo semplice ed efficiente.

L'algoritmo *Capacity Scaling*, noto per la sua efficienza nell'affrontare problemi di flusso massimo su reti, ha una complessità asintotica pari a $O(n^2m \log C)$ ed è stato implementato per garantire un equo confronto con la controparte quantistica. Di seguito, lo pseudocodice dell'algoritmo *Capacity Scaling*:

Algorithm 1 Capacity Scaling

```

1: procedure CAPACITYSCALING( $G$ )
2:   Inizializza il flusso  $f_e$  su ogni arco  $e \in E$  a 0
3:    $C \leftarrow \max_{e \in E} c_e$ 
4:   while  $C > 0$  do
5:     Trova un cammino aumentante  $p$  in  $G_f$ 
6:     while  $p$  esiste do
7:        $f_e \leftarrow f_e + \delta$  per ogni arco  $e \in p$ 
8:        $C \leftarrow C - \delta$ 
9:     Trova un nuovo cammino aumentante  $p$  in  $G_f$ 
10:    Ricalcola i flussi residui e i pesi degli archi in  $G_f$ 
11:     $C \leftarrow C/2$ 
12: return  $f$ 

```

Il codice sviluppato è stato organizzato in un'applicazione da linea di comando che consente di eseguire i vari algoritmi sull'intero dataset e di riportare i risultati. Inoltre, il pacchetto denominato *implementation* contiene il codice relativo all'algoritmo di *Capacity Scaling* e della riformulazione QUBO del problema di taglio minimo.

Questo setup sperimentale ha reso possibile condurre confronti accurati tra l'algoritmo quantistico adiabatico e l'algoritmo *Capacity Scaling* al fine di valutare l'efficacia della riformulazione proposta per il problema del taglio minimo. Inoltre, il setup ha consentito di valutare in modo qualitativo l'efficacia degli strumenti di calcolo non classici attualmente disponibili.

4.1 Test

I test condotti hanno coperto un insieme di grafi, suddivisibili in due macroclassi, i grafi più semplici e quelli di dimensione maggiore. In particolare, sono stati utilizzati i grafi presentati durante le prove d'esame del corso di Ottimizzazione Combinatoria, oltre a un sottoinsieme dei grafi disponibili nel dataset pubblico per i problemi di flusso[4].

I dati relativi ai grafi sono stati rappresentati e salvati nel formato DIMACS. Questo formato è uno standard ampiamente utilizzato per la rappresentazione di grafi, problemi di flusso e problemi di taglio minimo. La struttura di un file DIMACS è organizzata come segue:

- La prima riga del file contiene una descrizione del grafo o del problema.
- Seguono le righe per identificare specifici nodi, ad esempio il nodo sorgente e quello pozzo.
- Le righe successive contengono le informazioni sugli archi del grafo, ovvero le coppie di nodi componenti lo spigolo e la relativa capacità.

Aperto un file in formato DIMACS il formato visualizzabile è analogo a quanto riportato nel listato sottostante.

```
p max #nodes #edges
n node_id label
a source_node_id target_node_id edge_capacity
```

4.2 Risultati sperimentali

Nella tabella 1 sono presentati i tempi di esecuzione relativi alle tre diverse sperimentazioni condotte. Mentre, la tabella 2 riporta i risultati ottenuti, ovvero il massimo flusso, calcolati sia tramite l'approccio di calcolo classico che attraverso l'impiego del calcolo quantistico.

Nella seconda tabella, il valore ottenuto tramite la libreria e quello derivante dall'implementazione dell'algoritmo *Capacity Scaling* sono riportate nella stessa colonna in quanto, essendo entrambi algoritmi deterministici, avere risultati diversi implicherebbe un errore nell'esecuzione.

Tale discorso non si applica invece per la computazione quantistica, che restituisce l'istanziamento delle variabili a minore energia ma, sfruttando un approccio probabilistico, non c'è garanzia che il risultato sia esattamente quello desiderato.

Tabella 1: Tempi d'esecuzione

Grafo	Libreria (s)	Capacity Scaling (s)	QAC (s)
2015-06-10	0,00100	0,00100	11,37926 (0,15433)
2015-09-18	0,00100	0,00100	10,67002 (0,15719)
2015-07-09	0,00100	0,00100	9,03541 (0,15652)
2019-06-21	0,00100	0,00100	10,39780 (0,16976)
2020-09-15	0,00100	0,00100	8,99546 (0,14616)
2021-07-06	0,00100	0,00100	10,66569 (0,15182)
2021-06-14	0,00001	0,00100	10,60231 (0,16220)
2021-09-13	0,00001	0,00200	11,12439 (0,17091)
BVZ-tsukuba8 [†]	88,28760	416,60800	81,99760
BVZ-venus1 [†]	266,89360	1232,29300	144,78604
BVZ-sawtooth15 [†]	160,31860	1554,15700	144,88442
BVZ-tsukuba7 [†]	130,15300	199,88400	86,03621
BVZ-venus0 [†]	255,65700	1522,47000	136,51908
BVZ-venus10 [†]	257,85000	1855,94300	139,38559
BVZ-venus17 [†]	223,88600	1300,08000	136,03417
BVZ-venus13 [†]	86,02160	5486,61894	135,31621
BVZ-venus2 [†]	298,23400	1535,18092	135,78110
BVZ-venus11 [†]	112,00959	6132,16092	151,99681
BVZ-venus14 [†]	346,34100	2173,71200	152,43104
BVZ-venus15 [†]	81,29500	6218,44792	161,83125
BVZ-venus12 [†]	316,60100	2563,73400	133,98577
BVZ-venus16 [†]	318,94800	1415,69300	145,96313
KZ2-venus10 [†]	1188,77692	14554,88358	759,89836
KZ2-venus20 [†]	942,27799	9690,74837	790,60935
KZ2-venus16 [†]	982,49200	11064,19739	810,48272
KZ2-venus12 [†]	719,88700	15539,82438	930,74117
KZ2-venus14 [†]	833,82900	13426,26237	860,68817
KZ2-venus21 [†]	876,03400	9570,96602	878,06673
KZ2-venus17 [†]	1015,76200	10388,83364	863,26866
KZ2-venus11 [†]	439,95100	22397,76110	662,58155
KZ2-venus15 [†]	397,48610	19978,63321	592,02941
KZ2-venus8 [†]	911,93727	11027,99488	942,59513
KZ2-venus13 [†]	473,13800	21195,98017	613,73570
KZ2-venus9 [†]	541,43789	8374,89805	970,97154
Media (BVZ, KZ2)	471,75020	7723,76792	444,71604

Tabella 2: Flusso massimo calcolato

Grafo	Calcolo classico	Formulazione QUBO
2015-06-10	14	14
2015-09-18	7	7
2015-07-09	16	0
2019-06-21	12	12
2020-09-15	13	13
2021-07-06	10	10
2021-06-14	7	7
2021-09-13	12	12
BVZ-tsukuba8 [†]	54458	111389
BVZ-venus1 [†]	274732	78967
BVZ-sawtooth15 [†]	359799	63699
BVZ-tsukuba7 [†]	41095	212757
BVZ-venus0 [†]	360038	85987
BVZ-venus10 [†]	442371	97295
BVZ-venus17 [†]	259189	71971
BVZ-venus13 [†]	996632	162412
BVZ-venus2 [†]	234246	84274
BVZ-venus11 [†]	850805	175915
BVZ-venus14 [†]	448731	90606
BVZ-venus15 [†]	1149848	148969
BVZ-venus12 [†]	340048	82395
BVZ-venus16 [†]	210033	71981
KZ2-venus10 [†]	1375201	2356322
KZ2-venus20 [†]	989615	2026731
KZ2-venus16 [†]	1191635	1611271
KZ2-venus12 [†]	1393079	2417293
KZ2-venus14 [†]	1304192	1585019
KZ2-venus21 [†]	955443	1501347
KZ2-venus17 [†]	1140960	1573083
KZ2-venus11 [†]	1315279	2237533
KZ2-venus15 [†]	1034347	1349735
KZ2-venus8 [†]	1423372	1702522
KZ2-venus13 [†]	1230158	1450437
KZ2-venus9 [†]	1396819	2380765

I risultati sperimentali possono essere suddivisi in due macrogruppi basati sulla dimensione dei grafi utilizzati durante le sperimentazioni:

- Nel primo macrogruppo sono presenti i grafi di dimensione modesta, su cui è stato possibile eseguire l'algoritmo quantistico adiabatico sulla Quantum Processing Unit (QPU). Questi grafi hanno permesso di ottenere i risultati direttamente dall'hardware quantistico, consentendo una valutazione diretta delle prestazioni dell'algoritmo in un ambiente reale.
- Nel secondo macrogruppo sono inclusi i grafi estratti da dataset reali, indicati dal simbolo †. Questi grafi, a causa della loro maggiore dimensione e complessità, non hanno potuto essere eseguiti sulla QPU a causa dei limiti di tempo imposti dalla piattaforma. Pertanto, per valutare l'efficacia dell'algoritmo su tali istanze, abbiamo optato per la simulazione tramite l'utilizzo dell'algoritmo Simulated Annealing. Sebbene questo approccio consenta di ottenere risultati approssimati per i grafi di grandi dimensioni, va sottolineato che tali risposte possono variare rispetto a quelli ottenuti direttamente dalla QPU a causa della natura probabilistica dell'algoritmo quantistico e il diverso approccio all'esplorazione dello spazio degli stati.

Questo raggruppamento dei dati è fondamentale per una corretta interpretazione dei risultati e per comprendere l'effettiva prestazione dell'algoritmo quantistico adiabatico su grafi di diversa dimensione e complessità.

I dati relativi alle sperimentazioni eseguite sulla Quantum Processing Unit presentano due tempi d'esecuzione distinti:

- Il primo indica il tempo totale impiegato per l'intero processo, comprensivo della fase di riformulazione del problema, della connessione alla piattaforma remota, dell'esecuzione dell'algoritmo quantistico adiabatico sulla QPU e della ricezione della risposta. Questo valore rappresenta il tempo complessivo necessario per portare a termine l'intera operazione di calcolo per ogni istanza del problema.
- Il secondo dato, riportato tra parentesi, specifica il tempo impiegato esclusivamente per l'esecuzione dell'algoritmo sulla QPU, escludendo quindi i tempi necessari per le fasi di preparazione e comunicazione dei dati. Questo tempo rappresenta il tempo di esecuzione puro dell'algoritmo quantistico sulla QPU e riflette il contributo effettivo della piattaforma quantistica nella risoluzione del problema.

Questa distinzione tra i due tempi di esecuzione è importante per valutare l'efficacia e le prestazioni dell'algoritmo quantistico adiabatico sulla QPU. Il tempo di esecuzione puro rappresenta una metrica significativa per misurare l'efficienza della piattaforma quantistica nell'affrontare il problema del taglio minimo. D'altra parte, il tempo totale di esecuzione fornisce una visione più completa dell'intero processo, includendo le operazioni di preparazione dei dati e comunicazione con la piattaforma remota, aspetti importanti per valutare l'usabilità dell'algoritmo in scenari reali.

La differenza significativa tra il tempo di esecuzione dell'algoritmo quantistico e il tempo totale, come osservato, indica chiaramente che gran parte del tempo è occupata dalle operazioni di pre e post processamento dei dati. Questi processi aggiuntivi, come la riformulazione del problema e la comunicazione con la piattaforma remota, incidono notevolmente sul tempo complessivo richiesto per ottenere il risultato finale. Questo aspetto sottolinea l'importanza di ottimizzare tali procedure per migliorare l'efficienza globale dell'algoritmo quantistico adiabatico e renderlo più competitivo rispetto alle soluzioni deterministiche.

Tuttavia, anche considerando il solo tempo di esecuzione sull'hardware quantistico, i risultati mostrano che l'algoritmo quantistico non raggiunge i livelli di efficienza dell'implementazione proposta di Capacity Scaling o del metodo fornito dalla libreria NetworkX. Questo fenomeno è intrinsecamente correlato alla natura probabilistica della ricerca nello spazio degli stati dell'algoritmo quantistico. A differenza degli algoritmi deterministici, in cui si possono esplorare le soluzioni in modo sistematico, l'algoritmo quantistico adiabatico si basa su un approccio probabilistico, il che può comportare una maggiore dispersione nella ricerca e influire sulla sua efficienza complessiva.

Queste considerazioni sottolineano l'importanza di valutare attentamente il contesto e i requisiti specifici del problema quando si sceglie l'algoritmo appropriato per la risoluzione del problema del taglio minimo. L'algoritmo quantistico adiabatico può essere vantaggioso in alcuni scenari, soprattutto per problemi complessi con ampie possibilità di ricerca, ma potrebbe non essere la scelta più efficiente in tutti i contesti. La ricerca continua e l'ottimizzazione delle implementazioni potrebbero portare a miglioramenti significativi nelle prestazioni dell'algoritmo quantistico adiabatico e rendere questa tecnologia più competitiva nel panorama delle soluzioni computazionali.

Analizzando le discrepanze tra le due implementazioni deterministiche, emerge chiaramente che l'implementazione proposta di Capacity Scaling non è altrettanto efficiente del metodo fornito dalla libreria NetworkX.

Questo comportamento è principalmente influenzato da due fattori fondamentali. In primo luogo, la libreria NetworkX utilizza un diverso metodo per il calcolo del flusso massimo, noto come *Preflow Push*, il quale garantisce una migliore complessità asintotica di $O(n^2\sqrt{m})$. Questo algoritmo di calcolo del flusso massimo è più efficiente rispetto all'implementazione proposta di Capacity Scaling in termini di tempo di esecuzione, specialmente su grafi di grandi dimensioni e complessità.

Inoltre, va preso in considerazione il contesto dell'implementazione in Python. Python è un linguaggio di alto livello interpretato, mentre la libreria NetworkX è implementata in C++, un linguaggio compilato. A causa della natura interpretata di Python, questo linguaggio risulta essere significativamente più lento rispetto a C++ in termini di velocità di esecuzione delle operazioni. Di conseguenza, l'implementazione in Python dell'algoritmo Capacity Scaling subisce un impatto negativo sulla sua efficienza complessiva rispetto alla versione implementata in C++ di NetworkX.

Questi fattori combinati si traducono in un maggior tempo di computazione dell'implementazione di Capacity Scaling rispetto al metodo di libreria.

In relazione alla Tabella 2, si osserva una differenza sostanziale rispetto alle implementazioni classiche, che producono sempre e solo risultati corretti. Con l'aumento delle dimensioni dei grafi, si evidenzia un significativo aumento dell'errore derivante dall'esecuzione dell'algoritmo Simulated Annealing. Inoltre, non emerge una particolare correlazione tra il risultato ottenuto e la soluzione esatta del problema.

Le motivazioni di questo comportamento possono essere attribuite a due principali fattori:

- Una scelta non ottimale dell'iperparametro Lagrangiano, impostato a 15 nei nostri test. Tuttavia, va sottolineato che il valore di questo iperparametro potrebbe non essere adatto per tutte le istanze del problema. Un valore inappropriato può influenzare negativamente la capacità dell'algoritmo Simulated Annealing di convergere verso una soluzione accurata, portando a risultati meno precisi.
- Un aumento del *rumore* su grafi di grandi dimensioni, proprio dovuto al comportamento probabilistico che caratterizza Simulated Annealing e, in generale, gli approcci quantistici. La natura probabilistica di Simulated Annealing introduce incertezza nel processo decisionale, portando a una maggiore variabilità nei risultati ottenuti su istanze complesse. Questo fenomeno può incidere negativamente sulla precisione delle soluzioni trovate, soprattutto su grafi di grandi dimensioni, dove la complessità del problema può amplificare il rumore e influire sulla convergenza dell'algoritmo verso la soluzione ottima.

Queste considerazioni evidenziano l'importanza di una selezione attenta degli iperparametri e delle strategie di ottimizzazione per l'efficacia dell'algoritmo Simulated Annealing su istanze complesse del problema del taglio minimo. Inoltre, va tenuto presente che la natura probabilistica degli approcci quantistici può richiedere un numero maggiore di iterazioni o esecuzioni ripetute per ottenere risultati accurati e stabili.

In conclusione, il presente lavoro aveva l'obiettivo di esplorare la semplicità d'utilizzo delle tecnologie per la computazione quantistica adiabatica applicata a problemi per i quali è già nota una strategia risolutiva efficiente, nonché di verificare la loro efficacia pratica.

Inoltre, è stata condotta un'indagine comparativa su diversi approcci e implementazioni deterministiche. Tuttavia, come ci si poteva ragionevolmente attendere, a causa delle limitazioni attuali nell'utilizzo delle tecnologie quantistiche, soprattutto i limiti di tempo imposti dalla piattaforma, e considerando la natura stessa del problema affrontato, gli approcci classici risultano ancora la migliore strategia risolutiva. Tale affermazione risulta particolarmente valida per problemi in cui si conosce già una soluzione in tempo polinomiale, soprattutto per i polinomi il cui grado è contenuto (quadratico o cubico).

Pur riconoscendo il grande potenziale delle tecnologie quantistiche e i progressi continui nel campo della computazione quantistica adiabatica, è evidente

che la loro applicazione pratica per problemi di taglio minimo richiede ulteriori sviluppi e ottimizzazioni. Le limitazioni attuali, come i tempi di esecuzione e la complessità degli algoritmi quantistici, rendono ancora più vantaggiosi gli approcci classici per molti scenari.

Riferimenti bibliografici

- [1] NetworkX developers. Networkx. <https://networkx.org/>. Visitato il 2023-06-20.
- [2] L. R. Ford, Jr. and D. R. Fulkerson. Maximal flow through a network. *J-CAN-J-MATH*, 8:399–404, 1956.
- [3] Fred W. Glover and Gary A. Kochenberger. A tutorial on formulating QUBO models. *CoRR*, abs/1811.11538, 2018.
- [4] Patrick Møller Jensen, Niels Jeppesen, Anders Bjorholm Dahl, and Vedrana Andersen Dahl. Min-Cut/Max-Flow Problem Instances for Benchmarking. 3 2022.
- [5] Catherine C. McGeoch. Theory versus practice in annealing-based quantum computing. *Theoretical Computer Science*, 816:169–183, 2020.
- [6] DWave Systems. Dwave systems. <https://github.com/dwavesystems>. Visitato il 2023-06-20.