UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

TESI DI LAUREA

Applicazione di un Algoritmo Quantistico per la Selezione dei Casi di Test di Regressione

RELATORE

Prof. Andrea De Lucia

Università degli Studi di Salerno

CANDIDATO

Antonio Trovato

Matricola: 0522501270

Anno Accademico 2022-2023

Questa tesi è stata realizzata nel

Abstract

Una delle aree di studio più importanti nell'ambito dell'ingegneria del software consiste nello sviluppo di attività atte ad incrementare il più possibile l'affidabilità dei sistemi software. Il test di regressione è una fase di test del ciclo di sviluppo software che comporta la riesecuzione di test sul software ogni qualvolta esso venga modificato. Tale attività è necessaria poiché qualsiasi tipo di cambiamento al software che si sta sviluppando potrebbe comportare l'introduzione non intenzionale di errori nelle componenti non direttamente interessate dalla modifica stessa. In particolare, l'aggiunta, l'eliminazione o la semplice modifica di funzionalità potrebbe portare sia alla ricomparsa di vecchi bug già risolti sia alla comparsa di errori mai riscontrati prima. Eseguire, ogni volta che introduciamo del nuovo codice, un test totale del sistema diventa però presto molto tedioso e costoso. Infatti, anche per modifiche estremamente piccole, saremo costretti ad eseguire l'intera suite di test di regressione; risulta evidente che l'entità del problema venga amplificata anche dalla crescita del sistema, in quanto comporta anche la crescita della suite di test di regressione. Uno dei principali metodi per la riduzione dei costi della fase di test di regressione consiste nella selezione di un sottoinsieme (possibilmente minimo) di casi di test a partire dall'intera suite. Gli algoritmi proposti che riescono nell'obiettivo sono di diversa natura ma tra essi spicca l'algoritmo Diversity based Genetic Algorithm (DIV-GA). Si tratta di un algoritmo genetico multi-obiettivo (MOGA) che sfrutta i meccanismi della progettazione ortogonale e dell'evoluzione ortogonale per introdurre nuovi individui ortogonali durante ogni ciclo della ricerca genetica, ottenendo così un aumento della diversità nelle generazioni. Obiettivo di questo lavoro è mostrare che gli avanzamenti in atto nell'ambito della computazione quantistica non solo possano essere sfruttati nell'ambito dell'ingegneria del software, in particolare per lo sviluppo di sistemi software il più affidabili possibile, ma che inoltre essi possano essere sfruttati per ottenere un processo che riesca nell'intento di risolvere il problema di selezione dei casi di test di regressione tramite l'utilizzo del Quantum Annealing; per poi confrontare il metodo proposto in questo lavoro con il metodo DIV-GA.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

1.1 Contesto Applicativo

La rivoluzione più importante degli ultimi cento anni, secondo alcuni quasi equiparabile all'invenzione della ruota o addirittura a quella della scrittura, consiste senza alcun dubbio nell'invenzione dei computer. I computer non sono altro che macchine elettroniche il cui scopo consiste nel memorizzare, manipolare e comunicare dati con altri computer o esseri umani; in altre parole, i computer sono strumenti che supportano gli esseri umani nell'elaborazione di informazioni anche estremamente complesse.

La rivoluzione rappresentata dai computer è stata tale che, nel corso del secolo scorso, si è arrivati alla nascita di una nuova disciplina che studiasse nello specifico la progettazione e l'applicazione di strumenti di elaborazione delle informazioni. Dunque, nasce, a causa di queste nuove esigenze, l'informatica, ad oggi disciplina senza dubbio centrale in quasi ogni ambito della vita quotidiana e lavorativa delle persone.

L'impatto che l'informatica ha avuto sulle vite degli esseri umani può essere ana-

lizzata seguendo l'evoluzione che i computer stessi hanno subito nel corso della loro storia, in particolare dagli anni '20 sino ai giorni nostri. Essi sono passati dall'essere progettati come sistemi valvolari le cui dimensioni richiedevano stanze intere per essere contenute, agli odierni personal computer, le cui componenti di elaborazione hanno dimensioni inferiori al micron.

L'obiettivo che attualmente si sta perseguendo nella progettazione fisica delle componenti dei computer, riguarda la costruzione e la produzione di chip di dimensioni talmente ridotte da comportare porte logiche grandi come pochi atomi. La rivoluzione di questa direzione consiste nel fatto che la materia, quando si arriva al livello atomico, segue delle leggi fisiche ben diverse da quelle a cui ci ha abituato la fisica classica; ciò significa, che le porte logiche di tali dimensioni dovranno seguire regole diverse da quelle che sono attualmente sfruttate.

Lo studio di soluzioni appropriate ad un tale cambiamento ha portato alla nascita di una nuova branca dell'informatica: l'informatica quantistica. Sulla base delle scoperte nell'ambito della meccanica quantistica, si è stati in grado di sviluppare sistemi di computazione quantistica, ossia sistemi che utilizzino, al posto del bit convenzionale il cui valore logico può essere rappresentato solo tramite due valori possibili (0 o 1), i così detti bit quantistici (o qubit); essi rappresentano sistemi quantistici, in grado quindi di potersi trovare in uno stato equivalente alla sovrapposizione degli stati classici (spiegazioni più dettagliate saranno fornite nella Sezione 2.2).

Lo scopo della nascita dell'informatica quantistica è quello di poter progettare sistemi che siano in grado di risolvere problemi in ambiti specifici con un'accelerazione ed un aumento delle prestazioni estremamente rilevanti rispetto alle soluzioni classiche.

Di pari passo con i cambiamenti fisici dei computer ha proceduto l'evoluzione del software che tali macchine sono in grado di eseguire e che rappresenta il principale protagonista degli studi dell'informatica. Con l'avanzare delle capacità fisiche dei dispositivi di elaborazione, le esigenze di complessità nello sviluppo di sistemi software sono cresciute senza freni. Ciò ha portato, a cavallo tra gli anni '60 e '70 del secolo scorso, alla nascita di un nuovo ramo dell'informatica: l'ingegneria del

software. Si tratta di una disciplina che risponde alla necessità degli sviluppatori che si trovavano di fronte al problema di dover gestire sistemi software con livelli di complessità notevoli.

L'ingegneria del software, oltre al problema della complessità crescente dei sistemi da sviluppare, ha risolto altre problematiche importanti come:

1. **Fallimento dei Progetti:** per colpa di errori o ritardi imprevisti ed a causa di stime non corrette dei costi di produzione, troppi progetti finivano per fallire durante la loro stessa produzione. Ciò ha reso evidente la necessità di un approccio che fosse più rigoroso ed ingegneristico allo sviluppo di tali sistemi.

2. **Riusabilità del Software:** al fine di risparmiare costi, tempi e risorse, si è pensato di sviluppare una sola volta molte componenti software per poi riutilizzarle in altri progetti quando necessario.

3. **Collaborazione:** maggiori erano le dimensioni dei progetti, crescente era il numero del personale al lavoro, il che si traduceva in una maggiore difficoltà di comunicazione fra gli addetti ai lavori. Ciò poteva essere risolto solo tramite una standardizzazione delle strutture e dei processi di comunicazione del personale.

L'ingegneria del software ha dunque permesso la nascita di strumenti che fossero in grado di affrontare tutte le problematiche derivanti dall'esplosione che la produzione di software stava causando in quel periodo.

1.2 Motivazioni ed Obiettivi

Negli ultimi anni l'attenzione della ricerca riguardo l'ingegneria del software si è concentrata soprattutto sull'importanza di testare i sistemi software sia dopo che durante la loro produzione. Si tratta di un discorso estremamente ampio, in quanto il processo che guida lo sviluppo di un sistema software è senza dubbio uno dei più complessi nell'ambito dell'informatica; si tratta infatti di un processo che comprende valutazioni di natura economica, sociale, tecnica, inoltre richiede che vi sia, da parte di tutti coloro che partecipano ad una tale attività, una grande predisposizione alla collaborazione ed al lavoro in team. Proprio in questo contesto

si inserisce la fase di testing di un sistema software, la cui corretta integrazione nel processo di sviluppo è cruciale per poter individuare malfunzionamenti nel sistema, aumentandone dunque sicurezza, affidabilità e qualità, nonché diminuendone i tempi ed i costi manutenzione. Tutto ciò si traduce anche in un aumento della fiducia che utenti e clienti porranno nei confronti del prodotto e dell'azienda produttrice.

Oltre alle motivazioni appena descritte, altre cause che spiegano la crescente attenzione riguardo i processi di testing sono da ricercarsi nell'adozione, sempre maggiore negli ultimi anni, di sviluppi detti "Agili" per il software. L'adozione di questo tipo di metodologie di sviluppo pone un grande accento sull'utilizzo dei test. Infatti, se i metodi di sviluppo più classici prevedevano un approccio rigoroso e temporalmente ben cadenzato della fase di testing, i metodi agili prevedono che il testing pervada l'intera fase di sviluppo dei sistemi software, di fatto favorendo il più delle volte un approccio di implementazione di tipo test-driven. Questo tipo di approccio detto "Test-Driven Development" prevede infatti prima lo sviluppo di test che descrivano accuratamente il comportamento desiderato dal software da sviluppare e solo successivamente lo sviluppo del codice relativo che superi i test già prodotti.

Il test di regressione, tra i tanti test usati nell'ingegneria del software, è l'oggetto di studio del presente lavoro di tesi. Il suo scopo è identificare l'emergenza di errori ogni volta che siano state apportate delle modifiche al codice del software in produzione, eseguendo una suite di test già pronta (una spiegazione più accurata sarà eseguita nella Sezione 2.1). Il problema di tale approccio consiste proprio nella dimensione della suite di test da eseguire; infatti, per sistemi troppo complessi, risulta troppo dispendioso eseguire per ogni piccola modifica l'intera suite.

Proprio a partire da questa premessa prende le mosse il presente lavoro di tesi, il cui obiettivo è riuscire a sfruttare la potenza dei sistemi quantistici per rispondere al problema sopra citato dei test di regressione. La soluzione proposta in questo lavoro consiste nell'esecuzione dell'algoritmo di quantum annealing per risolvere il problema di selezione dei casi di test della suite utilizzata per la fase di test di regressione.

Per poter utilizzare il quantum annealing rispetto al problema di test case selection

è però necessario riuscire a mappare tale problema sotto forma di un problema di ottimizzazione quadratico a variabili binarie (QUBO).

Nello specifico, la soluzione proposta da questo lavoro di tesi consiste in 3 fasi fondamentali: lettura della suite di test, creazione del problema di ottimizzazione, risoluzione tramite quantum annealing.

Nella fase di lettura, data una suite di test, vengono estrapolati per ogni caso di test: costo di esecuzione, storico dei fallimenti rilevati, copertura delle istruzioni presenti nel codice (*statement coverage*).

Nella fase di creazione del problema, le informazioni estrapolate sono manipolate in modo da ottenere il corrispondente problema QUBO.

Infine, sul problema ottenuto viene eseguito l'algoritmo di quantum annealing per mostrarne le soluzioni e codificarne la migliore.

1.3 Risultati Ottenuti

Il progetto proposto ha portato alla creazione di una vera e propria pipeline di esecuzione che, a partire da una suite di test, arriva alla creazione di una suite di test finale che sia conforme agli obiettivi ed ai vincoli del problema di ottimizzazione formalizzato.

Il contesto dello studio corrisponde all'utilizzo di quattro programmi industriali open-source GNU messi a disposizione dalla "software-artifact infrastructure repository" (SIR): grep, gzip, flex e sed. La SIR mette a disposizione tutto il necessario per poter estrapolare le informazioni utili al fine di risolvere il problema di test case selection tramite quantum annealing.

La creazione del problema QUBO e l'esecuzione del quantum annealing sono stati possibili grazie all'utilizzo del modulo dwave per Python. Si tratta di un modulo messo a disposizione dalla compagnia D-Wave che è la principale protagonista della progettazione di computer che sfrutta proprio l'algoritmo di quantum annealing. Inoltre, la scelta si è resa obbligata dal fatto che D-Wave è l'unica compagnia a mettere a disposizione sistemi pubblici abbastanza potenti per il problema da risolvere.

1.4 Struttura della Tesi

Gli argomenti di cui si compone questo lavoro di tesi sono elencati e descritti di seguito:

1. **Capitolo 2:** in questo capitolo vengono descritti nel dettaglio il regression testing, i principi fondamentali dell'informatica quantistica, le soluzioni attualmente utilizzate per risolvere il problema di test case selection e il processo di quantum annealing.

2. **Capitolo 3:** in questo capitolo vengono descritti il sistema progettato e le fasi che ne hanno permesso lo sviluppo.

3. **Capitolo 4:** in questo capitolo vengono descritte le possibili minacce alla validità delle valutazioni fatte sul sistema progettato.

4. **Capitolo 5:** in questo capitolo ci si concentra sulle potenziali fonti di errore o di minacce che potrebbero compromettere la validità dei risultati ottenuti durante lo studio.

CAPITOLO 2

Background e Stato dell'Arte

In questo capitolo vengono descritti lo stato dell'arte e gli studi già presenti in letteratura da cui prende le mosse questo lavoro di tesi.

2.1 Selezione dei Casi di Test di Regressione: Una Panoramica degli Approcci Più Comuni

In questa sezione viene descritto il compito della fase di test di regressione. Successivamente, dopo la trattazione nei dettagli del problema di selezione dei casi di test, vengono descritte approfonditamente le strategie attualmente utilizzate per la risoluzione di tale problema.

2.1.1 Regression Testing e Affidabilità del Software

Uno dei concetti più importanti che vengono affrontati in ingegneria del software è senza dubbio quello dell'*affidabilità di un sistema software*. L'affidabilità viene definita da John D. Musa [1] nel modo seguente:

1. Affidabilità di un Sistema Software

L'affidabilità del software è la probabilità che esso operi senza errori per un periodo di tempo specificato in un ambiente specificato.

Tra le pratiche che più di tutte vanno incontro all'esigenza degli sviluppatori di ottenere dei sistemi software che siano il più affidabile possibile vi è sicuramente quella di *testing*. Il testing è un processo atto a verificare e valutare il funzionamento di un sistema software rispetto a delle aspettative, il cui obiettivo primario consiste quindi nel rilevare malfunzionamenti che potrebbero rivelarsi sotto precise condizioni di utilizzo del software. Ci sono diversi tipi di test a cui è possibile sottoporre un software; essi si differenziano per gli obiettivi e per le strategie che adottano. In questo lavoro di tesi ci si concentrerà sul *regression testing*. Lo scopo del regression testing non è altro che quello di rieseguire una suite di test su un sistema software in via di sviluppo, non appena siano apportate delle modifiche al codice sorgente [2]. Quando parliamo di Regression Testing non possiamo non specificare il concetto di "*regressione*". Per regressione si intende un errore o un malfunzionamento che emerge in seguito a modifiche, anche qualora le funzionalità testate fossero in precedenza correttamente funzionanti. Dunque, l'obiettivo che si pone il regression testing è proprio quello di rilevare regressioni, scongiurando il pericolo di degradazione del software dovuto a cambiamenti più o meno frequenti.

La pratica di regression testing può essere potenzialmente eseguita durante qualsiasi fase del ciclo di vita del software; ad esempio può essere inclusa nelle pipeline di continuous integration e continuous deployment, diventa perfettamente adattabile a contesti "*Agile*" o ad ambiti che fanno uso di un approccio "*DevOps*", permettendo quindi che il software in via di sviluppo resti altamente affidabile anche in contesti di sviluppo rapido. Tutto ciò consente di avviare il processo di regression testing ogni volta che viene effettuata una modifica al software, andando incontro quindi alle esigenze di affidabilità del software. Dunque, il regression testing consente di mantenere alta la qualità del codice che si sta sviluppando, man mano che la sua complessità aumenta nel tempo; inoltre, la rapida identificazione di difetti nel codice riduce di fatto i tempi di sviluppo del sistema, in quanto la risoluzione di tali difetti

viene eseguita immediatamente e quindi anche i tempi di rilascio del software al cliente vengono ridotti.

Anche la pratica del regression testing presenta però delle problematiche piuttosto importanti che bisogna affrontare. Più nel dettaglio, va considerato che rieseguire il test dell'intero sistema software lanciandone ogni caso di test di cui si ha disponibilità, oltre ad essere potenzialmente molto dispendioso, potrebbe rivelarsi addirittura inattuabile, soprattutto per sistemi grandi e se la riesecuzione delle test suite intere avviene per ogni singola modifica [3][4]. I tempi di esecuzione delle test suite sono chiaramente variabili e ci si potrebbe trovare a dover eseguire test suite che richiedono l'impiego di ore o giorni per essere interamente eseguite [5]; è chiaro che il problema peggiora con la crescita del sistema stesso da testare.

Una delle strategie impiegate per risolvere il suddetto problema consiste nel selezionare un sottoinsieme (possibilmente minimo) di casi di test dalla test suite originale che possa verificare in maniera efficace il software in esame, cercando però di ridimensionare il più possibile il costo totale di esecuzione dei test. Questa strategia è detta "*Test Case Selection*" e rientra nelle pratiche di ottimizzazione dei casi di test.

2.1.2 Ottimizzazione di Test Suite

La sezione di "Ottimizzazione di Test Suite" si occupa di descrivere nel dettaglio gli approcci più usati per la riduzione dei costi di esecuzione di suite di test, soffermandosi in particolare sull'utilizzo di tali tecniche per l'ottimizzazione di suite di test dedicate ai test di regressione.

Gli approcci descritti riguardano: *test suite minimization* (TSM), *test case selection*, *test case prioritization* (TCP). Riguardo l'utilizzo pratico degli approcci appena elencati è possibile consultare il lavoro di Yoo ed Harman[6].

Nelle sezioni a seguire verranno descritti nei dettagli gli approcci di ottimizzazione delle test suite; in particolare ci si soffermerà sull'approccio di test case selection che sarà poi affrontato in questo lavoro.

Test Case Minimization

L'obiettivo del problema di minimization delle suite di test consiste nel ridurre il più possibile le dimensioni delle suite di test, eliminando da esse tutti i casi di test che risultano ridondanti. La ridondanza è rilevata tramite l'utilizzo di precisi criteri[7], tra i quali: code coverage, branch coverage, data flow, dynamic program invariant e call stack[8].

In effetti, si tratta di un problema di tipo NP-completo in quanto riducibile al più classico problema del *minimal hitting set* e per questo motivo gli algoritmi che lo risolvono sono costretti ad usare determinate euristiche[9][8][10]. È possibile citare per esempio il lavoro di Harrold et al.[8] che, per risolvere il problema del minimal hitting set, sfrutta un algoritmo greedy; oppure ancora il lavoro di Offut et al.[10] che, sebbene tenti sempre un approccio greedy, lo esegue su differenti criteri di ordinamento dei casi di test, piuttosto che sull'ordinamento originario prefissato. Attualmente non sembra esserci alcun approccio greedy che predomini sugli altri[10]. I lavori appena citati utilizzano un approccio greedy basato su criteri di copertura strutturali esclusivamente a livello di codice; sono quindi state proposte alternative che utilizzino criteri di copertura differenti. Nel lavoro di Marré e Bertolino[11] viene proposta una riformulazione del problema di test suite minimization al fine di trovare un insieme di coperturta minimo del grafo *decision-to-decision*. Un'altra proposta ancora è invece quella presentata dal lavoro di McMaster e Memon[12], che consiste nel risolvere il problema di minimization tramite una tecnica basata sulla call-stack coverage. Infine, il lavoro di Black et al.[13] considera un modello a due criteri, ossia: copertura del codice e storico degli errori rilevati da ciascun caso di test. Tali obiettivi, combinati tra loro tramite l'uso di una somma pesata, danno luogo ad un problema di programmazione lineare di ottimizzazione, riducendo di fatto il problema da due ad un unico obiettivo.

Test Case Prioritization

Il problema di prioritizzazione della test suite consiste nel riordinare i casi di test presenti all'interno di una test suite di partenza, in modo da massimizzare de-

terminate proprietà[14]. Il risultato ideale di un tale processo consiste nell'ottenere un ordinamento dei casi di test che permetta di massimizzare la frequenza di rilevamento degli errori. Purtroppo quest'ultimo è un dato che può essere conosciuto solo dopo aver ottenuto l'ordinamento e quindi in seguito all'esecuzione della suite risultante. Di conseguenza, gli approcci che eseguono la test suite prioritization si basano su delle euristiche correlate alla metrica di frequenza di rilevamento degli errori, come la copertura del codice[15][16][14], la copertura delle interazioni[17], la copertura basata su cluster[18] e la copertura dei requisiti[19]. Una volta scelto un criterio, l'ordinamento viene eseguito con l'aiuto di un algoritmo greedy, poiché l'ordine di scelta dei vari casi di test rispecchia anche l'ordine secondo cui essi vanno eseguiti.

Spesso affrontare i TCP significa formalizzarli come problemi costituiti da modelli a due criteri (in genere i criteri utilizzati riguardano la copertura ed il costo dei test) che vanno risolti tramite algoritmi greedy; tale modello può essere però convertito in un approccio basato su un singolo obiettivo da massimizzare, applicando una somma pesata ai criteri scelti per l'ordinamento[15][20]. Per citare alcuni studi in merito al problema di prioritizzazione di suite di test possiamo considerare il lavoro di Rothelmel et al.[14], il quale effettua un confronto tra tecniche di prioritizzazione basate su precisi criteri e una tecnica di prioritizzazione completamente casuale, per dimostrare empiricamente la valenza effettiva della prima categoria di tecniche allo scopo di rilevare fallimenti il prima possibile; risultati simili sono stati raccolti, tramite l'utilizzo del framework JUnit per i test di unità Java, dal lavoro di Do et al.[21]. Infine, ad ampliare le analisi riguardo i criteri utilizzabili per il TCP, come per esempio il criterio di copertura degli statement, vi è il lavoro di Elbaum et al.[22].

Test Case Selection

Il problema di selezione di casi di test consiste nel selezionare un sottoinsieme di casi di test, partendo da una test suite iniziale (insieme originale di casi di test); lo scopo del sottoinsieme risultante potrebbe essere, come nel caso del regression testing, quello di verificare che il comportamento delle porzioni di codice non modificate del programma che si sta sviluppando non sia stato degradato da un'eventuale modifica

al codice e che quindi non presenti regressioni[23]. Per riuscire ad identificare quali sono le porzioni di codice che non sono state raggiunte direttamente dalle modifiche, possono essere utilizzate diverse tecniche di analisi come: l'*Integer Programming*, l'esecuzione simbolica[24], l'analisi del data flow[25], le tecniche basate sul grafo delle dipendenze[26] e gli approcci basati sul flow graph[23]. Dunque, le differenze che intercorrono tra i vari approcci elencati per identificare le porzioni di codice non modificate, riguardano le modalità tramite le quali tali tecniche definiscono, cercano ed identificano delle modifiche nel programma che si sta sviluppando[6]. Una volta rilevate le porzioni di codice che siamo interessati a ritestare ed una volta scelto un algoritmo di ottimizzazione (per esempio quello greedy), è possibile eseguire il processo di selezione dei casi di test a partire dalla suite di test originale, in modo da ottenere un sottoinsieme della suite di partenza, i cui test massimizzino un criterio (come lo statement coverage) scelto, riducendo il costo totale della fase di regression testing.

Esistono delle forti correlazioni tra i problemi di TSM, TCP e test case selection, come già mostrato dagli studi di Yoo ed Harman[27][28]. In effetti, sia nel problema di minimizzazione che in quello di selezione, viene eseguita una scrematura di casi test da una suite di test iniziale, per fare in modo che si soddisfino al meglio possibile dei precisi criteri di test, come la copertura del codice[29]. Inoltre, risultano alquanto evidenti anche i legami tra la prioritizzazione dei casi di test e la loro selezione, in quanto l'obiettivo vero e proprio dell'ordinamento da applicare alla test suite consiste nella riduzione del costo totale della fase, nel caso di studio, di regression testing. Un esempio di combinazione tra il problema di prioritizzazione e quello di selezione è presente nel lavoro di Srivastava e Thiagarajan[30], nel quale, dopo aver rilevato le porzioni di codice non direttamente modificate tramite il confronto del codice binario del software pre e post modifica, si esegue un algoritmo greedy di ordinamento che però considera solo la copertura del codice non modificato.

2.1.3 Ottimizzazione di Test Suite Basata sulla Ricerca

Un approccio ampiamente utilizzato in letteratura per risolvere il problema di ottimizzazione delle test suite consiste nel riformulare i problemi di test suite mi-

nimization, test case prioritization e test case selection come un unico problema multi obiettivo. In questo modo, applicando algoritmi per loro natura adatti a questo tipo di problemi, come i Multi-Objective Genetic Algorithms (MOGAs), possiamo ottenere un sottoinsieme *ottimi di Pareto* di casi di test, partendo dalla suite iniziale.

NB: ►***La formulazione basata sulla ricerca che viene di seguito enunciata si riferisce al problema di selezione di casi di test utilizzato in questo lavoro di tesi, il quale non richiede un ordinamento dei casi di test; quindi, si rende necessaria una riformulazione del problema nel caso in cui si voglia includere anche la prioritizzazione dei casi di test, per esempio considerando delle*** $\mathbf{n-ple}$ ***piuttosto che degli insiemi***◄

2. Problema Multi Obiettivo di Selezione dei Casi di Test

Sia $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$ una suite di test e sia $F = \{f_1, f_2, ..., f_n\}$ un insieme di funzioni obiettivo: selezionare un sottoinsieme $\Gamma' \subseteq \Gamma$ tale che Γ' sia l'insieme ottimo di Pareto rispetto alle funzioni obiettivo in F.

Si consideri che l'insieme F delle funzioni obiettivo contiene le definizioni matematiche dei criteri di test che devono essere soddisfatti durante il processo di ricerca. Inoltre, l'ottimalità della soluzione viene misurata tramite i concetti di *ottimalità di Pareto* e *dominanza di Pareto*.

Si esprimono di seguito tutte le definizioni utili al fine di comprendere i concetti più importanti riguardanti valutazioni basate su ottimalità di Pareto e dominanza di Pareto.

3. Ottimo di Pareto

Una soluzione X è detta *Ottimo di Pareto* se e solo se è *non dominata* da nessun'altra soluzione all'interno dello spazio di ricerca.

4. Dominanza di Pareto

Una soluzione Y domina una soluzione X se e solo se:

1. Y è almeno buona quanto X per ogni funzione obiettivo;
2. esiste almeno una funzione obiettivo tale che Y è strettamente migliore di X per quella funzione.

5. Insieme Ottimo di Pareto

L'insieme ottimo di Pareto è costituito da tutte le soluzioni che non sono dominate da nessun'altra soluzione.

6. Fronte di Pareto

L'insieme di tutti i vettori obiettivo, ossia tutti i vettori che contengono i valori delle funzioni obiettivo.

Il ruolo che ricopre il fronte di Pareto è centrale nell'ingegneria del software, in quanto è possibile usufruirne per prendere decisioni consapevoli, eseguendo trade-off tra i vari obiettivi che si vogliono perseguire. Per esempio, un ingegnere del software può essere interessato principalmente a soluzioni che riducano il costo di esecuzione della suite di test risultante, anche a scapito della copertura del codice totale.

Gli algoritmi attualmente più utilizzati per risolvere il problema di test case selection si dividono in due famiglie: MOGA e greedy. In particolare, i MOGA che più interessano questo lavoro di tesi sono: vNSGA-II e DIV-GA.

I Multi-Objective Genetic Algorithm

Per poter comprendere il funzionamento e le proprietà degli algoritmi NSGA-II e vNSGA-II attualmente utilizzati per risolvere il problema di test case selection, è necessario comprendere con precisione la famiglia di algoritmi a cui appartengono, ossia quella degli *algoritmi genetici multi obiettivo*.

I MOGA rappresentano una classe estremamente potente di algoritmi che risolvono problemi a più obiettivi di ottimizzazione ispirandosi al processo di selezione naturale. Tali algoritmi partono da una "*popolazione*" iniziale e prodotta casualmente di soluzioni possibili al problema da risolvere, dove individui diversi della popolazione rappresentano soluzioni diverse. Quindi, l'algoritmo procede per cicli detti "*generazioni*", durante i quali seleziona gli individui che ritiene migliori sulla base degli obiettivi da raggiungere, combinandoli fra loro e costruendo così la generazione successiva di individui (soluzioni). L'idea è dunque quella di migliorare sempre di più gli individui man mano che si avanza con le generazioni, tenendo di volta in volta traccia degli individui migliori che incontra durante la sua esecuzione e costruendo in questo modo il fronte di Pareto da restituire al termine dell'esecuzione.

Dunque, il vantaggio principale di usare MOGA piuttosto che strategie classiche come quelle greedy sta nella loro capacità intrinseca di operare su più soluzioni contemporaneamente.

I MOGA vengono spesso suddivisi in MOGA "*elitisti*" e MOGA "*non elitisti*", sulla base dell'utilizzo o meno di strategie di elitismo, ossia strategie che consistono nel preservare soluzioni di alta qualità durante l'avanzare delle generazioni. Si descrivono di seguito le differenze tra i due tipi di MOGA:

1. **MOGA Elitisti:** Nel caso dei MOGA elitisti, in ogni momento dell'esecuzione le soluzioni presenti nel fronte di Pareto attuale sono conservate e trasferite nella generazione successiva. In questo modo si assicurano anche alle generazioni successive le soluzioni migliori utilizzate dalle generazioni passate, in modo da non perdere il fronte già scoperto. I MOGA elitisti assicurano un'alta qualità delle soluzioni, inoltre mantengono stabilità ed efficacia dell'algoritmo.

2. **MOGA Non Elitisti:** nel caso dei MOGA non elitisti, le soluzioni migliori delle generazioni precedenti non vengono necessariamente tramandate nelle successive. Quindi, non c'è garanzia che il fronte di Pareto già scoperto sia effettivamente mantenuto. Il vantaggio di usare una strategia non elitista sta nel fatto che questo tipo di MOGA riesce ad ottenere generazioni con livelli di

diversità maggiore tra gli individui, anche se potrebbe rendere più difficile la convergenza verso una soluzione ottima.

Gli Algoritmi Greedy e vNSGA-II

Nei lavori di Yoo ed Harman[27][28], sono considerate delle formulazioni che utilizzano tre obiettivi contrastanti: la copertura del codice, il costo di esecuzione e lo storico dei rilevamenti di errori. Nel loro lavoro, essi esaminano i risultati di diversi algoritmi utilizzati per la ricerca dei sottoinsiemi ottimi di Pareto della suite di test: *algoritmi greedy "additivi"* ed una variante dell'algoritmo genetico multi obiettivo NSGA-II[31] (chiamata vNSGA-II)[32].

NSGA-II è un algoritmo di ricerca genetico il cui output non consiste in una soluzione singola, bensì nell'intera frontiera di Pareto costruita dall'algoritmo. NSGA-II, per raggiungere il proprio obiettivo, sfrutta un meccanismo detto di *crowding distance*; la crowding distance misura quanto un individuo è distante rispetto al resto della popolazione, in modo da riuscire a costruire, selezionando individui il più possibile diversi fra loro, un fronte di Pareto abbastanza ampio. Inoltre NSGA-II è basato sull'elitismo, ossia esegue un ordinamento "non dominato" ad ogni generazione, in modo da preservare gli individui che sono nel fronte di Pareto attuale nelle generazioni successive.

Sulla base di NSGA-II è stata implementata da Yoo ed Harman una variante detta vNSGA-II. Le modifiche apportate a NSGA-II per la generazione di questa variante hanno interessato due aspetti fondamentali dell'algoritmo originale. Per prima cosa, per ottenere la costruzione di un fronte di Pareto il più ampio possibile, l'algoritmo utilizza un gruppo di sottopopolazioni separate le una dalle altre; quando poi viene eseguito il *pairwise tournament selection* sugli individui che formano una coppia non dominata, ognuna delle sottopopolazioni preferirà obiettivi differenti ed in questo modo il fronte sarà allargato in ogni direzione. In secondo luogo, vNSGA-II presenta un elitismo "rinforzato", ossia, rispetto a NSGA-II mantiene un registro del fronte di Pareto *migliore fino ad ora* separato dalle sottopopolazioni.

Oltre a vNSGA-II, nei lavori di Yoo ed Harman, è stata utilizzata anche una strategia greedy. Per poter lanciare una strategia greedy al problema di selezione dei casi di test, è stato necessario utilizzare un approccio a somma pesata, in modo da far confluire i tre differenti obiettivi in un'unica funzione obiettivo da ottimizzare. In particolare, nella versione a tre obiettivi è stata utilizzata una somma pesata della copertura di codice per unità di tempo e copertura dei fallimenti per unità di tempo (sempre con l'obiettivo di minimizzare il costo totale della suite risultante). Più nello specifico, considerando M obiettivi diversi, f_i con $i = 1, 2, ..., M$, l'approccio a somma pesata calcola l'obiettivo singolo f' come:

$$f' = \sum_{i=1}^{M}(w_i \cdot f_i), \sum_{i=1}^{M} w_i = 1 \tag{2.1.1}$$

Si consideri inoltre che gli obiettivi di copertura di codice e di errori sono stati combinati tramite dei coefficienti di 0.5 e 0.5, in modo da dar loro pari importanza.

In seguito all'implementazioni di vNSGA-II e della strategia greedy, sono stati eseguiti dei paragoni per valutarne la validità ed eventualmente la dominanza di una strategia sull'altra. I paragoni empirici effettuati tra i risultati ottenuti tramite i MOGA e quelli ottenuti con algoritmi greedy non hanno rivelato un chiaro vincitore; non sempre infatti i MOGA si sono resi capaci di superare le performance delle soluzioni greedy[27]; per altro, neanche soluzioni ibride hanno mostrato cambiamenti effettivi[28]. Inoltre, sebbene gli algoritmi greedy performino molto bene per formulazioni ad obiettivo singolo, non sono sempre Pareto-efficienti nel paradigma multi obiettivo, il che motiva quindi tecniche metaeuristiche[27][28].

DIV-GA

Proprio in questo contesto va inserito un nuovo MOGA, detto *"Diversity based Genetic Algorithm"* (DIV-GA), proposto da Panichella, Oliveto, Di Penta e De Lucia[33]. Lo sviluppo di DIV-GA prende le mosse proprio dall'incapacità degli approcci MOGA di superare definitivamente gli algoritmi greedy. In particolare, si ritiene che il problema dietro le performance non sempre eccellenti dei MOGA sia dovuto

al fenomento del *genetic drift*, che corrisponde ad una perdita di diversità tra le generazioni che si susseguono nei cicli di esecuzione dei MOGA[2][34]. Ciò causa una convergenza prematura verso una regione sub-ottima. In questo modo gli algoritmi genetici tendono a generare alcuni gruppi di soluzioni (nicchie), tutte vicine fra loro nello spazio di ricerca, lasciando dunque inesplorato tutto il resto dello spazio. Il problema del mantenimento di diversità all'interno delle generazioni degli algoritmi genetici è in generale riconosciuto come uno dei problemi principali di questa classe di algoritmi. Inoltre ha un impatto ancora maggiore quando ci si ritrova ad affrontare problemi multi obiettivo[35]. L'importanza di mantenere diversità è stata dimostrata da molti contributi scientifici di ricerca, il cui lavoro si è concentrato sul risolvere tale problema per problemi numerici[36][32][37][38][39].

Una classificazione completa dei meccanismi in grado di mantenere diversità proposti in letteratura per prevenire il genetic drift nell'ambito di problemi numerici possono essere trovati in un sondaggio recente, registrato da Črepinšek et al.[2]. Secondo questo sondaggio, i meccanismi di mantenimento di diversità applicati a problemi di ottimizzazione multi obiettivo cercano di aumentare la diversità tra le soluzioni fornite dallo *spazio dei fenotipi* (nel caso del problema della selezione dei casi di test, raggiungere un alto livello di diversità fenotipica significa che sottoinsiemi diversi della test suite di partenza, ossia risultati diversi, hanno punteggi delle funzioni obiettivo diversi), ossia lo spazio delle soluzioni decodificate tramite le funzioni di fitness, intervenendo su componenti diversi dell'algoritmo genetico; spesso si modifica la funzione di fitness, oppure si cerca di promuovere la diversità durante il processo di selezione. Esempi di meccanismi largamente utilizzati con lo scopo di preservare la diversità sono: *fitness sharing*[34][40], *crowding distance*[41], *restricted tournament selection*[42], *rank scaling selection*[43].

L'obiettivo di DIV-GA è quello di proporre un algoritmo che riesca a migliorare le performance di vNSGA-II e delle soluzioni greedy, tramite la promozione di diversità tra le soluzioni (ossia i sottoinsiemi della suite di partenza) nello *spazio del genotipo*, ossia nello spazio delle stringhe genetiche che codificano le soluzioni, e non in quello del fenotipo. Per fare ciò, DIV-GA propone due nuovi operatori genetici. Prima di

tutto implementa un algoritmo generativo per la costruzione di una popolazione iniziale diversificata, basato sul *design ortogonale*; inoltre implementa un meccanismo di *esplorazione ortogonale* dello spazio di ricerca che, attraverso una decomposizione a valori singolari (DVS), preserva la diversità durante l'intero processo evolutivo. Poiché questi due meccanismi agiscono sullo spazio del genotipo, essi possono essere applicati per ogni problema di ottimizzazione di suite di test, indipendentemente dal numero di funzioni obiettivo o di criteri che vengono considerati.

La generazione di una popolazione iniziale tramite il metodo del design ortogonale gioca un ruolo essenziale riguardo le performance degli algoritmi genetici. Infatti, partire da una popolazione iniziale che sia ben distribuita e ben diversificata rende l'esplorazione dello spazio di ricerca molto più efficace, facilitando la convergenza verso l'ottimo globale. Partire da una popolazione iniziale poco diversificata riduce drasticamente l'ottimalità e le performance dell'algoritmo.

Una soluzione genetica al problema di selezione di casi di test consiste in una lista di valori binari $X = \{x_1, x_2, ..., x_n\}$, dove x_i è un gene. Dunque, il problema di generare una popolazione iniziale ben distribuita per un algoritmo genetico è equivalente al problema di trovare un campione (possibilmente piccolo) rappresentativo di tutte le possibili combinazioni dei geni (i casi di test) per un certo esperimento.

DIV-GA implementa il design ortogonale tramite l'utilizzo di matrici di Hadamard, utilizzate per la generazione di *liste ortogonali* che rappresentano i singoli individui della popolazione iniziale.

La generazione di una popolazione iniziale ben diversificata non assicura però che, durante il processo evolutivo, la diversità degli individui possa essere compromessa al punto da portare l'algoritmo ad esplorare sempre le stesse zone dello spazio di ricerca. Il concetto base su cui si fonda l'operato degli algoritmi genetici consiste nell'evolvere le popolazioni verso le regioni che presentano fitness migliori, in quanto da ogni generazione viene sfruttato un *operatore di selezione* il cui scopo è proprio quello di selezionare gli individui migliori da far sopravvivere. Nel caso multi obiettivo, l'operatore di selezione sceglie per la riproduzione gli individui non dominati da altre soluzioni della popolazione attuale. Dunque, man mano che

l'algoritmo crea le generazioni, gli individui migliori tenderanno a convergere verso regioni ottime di Pareto locali o globali; ciò significa che vi è il rischio di rimanere intrappolati in ottimi locali, anche nel caso di selezioni completamente casuali.

Del Lucia et al.[36] hanno dimostrato che è possibile osservare in che direzione gli individui migliori stanno evolvendo dopo due generazioni consecutive. Tramite una decomposizione a valori singolari è possibile stimare questi movimenti, detti *direzioni evolutive*; in questo modo è possibile iniettare (quando necessario) diversità nella popolazione per spingerne l'evoluzione verso regioni inesplorate/ortogonali. L'iniezione consiste nella sostituzione degli individui peggiori della popolazione con degli individui nuovi che siano ortogonali agli individui migliori della popolazione attuale.

DIV-GA adatta quest'osservazione al problema della selezione dei casi di test.

In seguito a studi empirici sulla validità di DIV-GA rispetto alle soluzioni MOGA ed agli algoritmi genetici, è apparso evidente che DIV-GA riesca a superare le performance sia degli altri MOGA visti che delle strategie greedy. Inoltre, si è visto che DIV-GA presenta anche una superiorità in termini di qualità delle soluzioni costruite. Purtroppo però, vNSGA-II, DIV-GA e gli algoritmi greedy risolvono il problema di selezione dei casi di test in tempi spesso eccessivi, problematica che si aggrava con l'aumentare delle dimensioni del codice da testare. Le capacità della computazione quantistica, soprattutto nell'ambito di problemi di ottimizzazione, promettono però di risolvere il problema in tempi praticamente costanti. Poiché la produzione di software a livello industriale prevede spesso sistemi di dimensioni estremamente elevate, risulta necessario affidarsi a soluzioni quantistiche e l'obiettivo di questo lavoro di tesi consiste nel proporre come soluzione il quantum annealing.

2.2 Informazione Quantistica e Computazione Quantistica

In questa sezione viene presentata una panoramica generale delle implicazioni dell'avvento della computazione quantistica. Successivamente vengono descritti i fon-

damenti su cui si basa l'informatica quantistica attraverso l'analisi delle potenzialità e dunque dei relativi limiti.

2.2.1 Il passaggio al mondo quantistico

L'avvento dei computer quantistici rappresenta una svolta rivoluzionaria nella storia della computazione. Per comprendere appieno questo progresso, è fondamentale esaminare i contributi di visionari come Alan Turing, John von Neumann e Richard Feynman.

Alan Turing è noto per aver sviluppato il concetto di macchina di Turing, ossia un modello astratto che è in grado di descrivere in maniera formale ciò che può realmente eseguire un calcolatore. Il modello classico di macchina di Turing prevede semplicemente un dispositivo con a disposizione un nastro di lunghezza infinita, suddiviso in celle, oltre che una testina che è in grado di leggere e scrivere simboli sul nastro stesso seguendo un insieme di regole predefinite. A partire da questo modello ideale, che è riuscito a dimostrare che molte operazioni di calcolo possono essere eseguite da un semplice dispositivo meccanico, si è riusciti ad aprire la strada alla costruzione di computer reali.

John von Neumann ha contribuito ulteriormente all'architettura dei computer, impostando un'architettura costituita da memorie centrali, unità di elaborazione e programmi memorizzati. Questo modello, noto con il nome di "Architettura di Von Neumann", rappresenta l'architettura su cui si basano tutti i calcolatori moderni ed è caratterizzato dalla capacità di eseguire istruzioni in sequenza da programmi presenti in memoria.

Il contributo di Richard Feynman è particolarmente rilevante nell'ambito della computazione quantistica. Feynman ha suggerito che i sistemi quantistici possono essere simulati più efficientemente da un computer quantistico rispetto alla loro controparte classica. Inoltre, ha introdotto il concetto di *simulatore quantistico universale*, ossia un dispositivo che è in grado di simulare un qualsiasi sistema quantistico in maniera del tutto accurata. Questo principio è fondamentale nella progettazione dei computer quantistici e sottolinea la potenziale superiorità di tali dispositivi nel risolvere problemi legati alla fisica quantistica, alla chimica e ad altre discipline.

L'emergere della computazione quantistica sta portando pian piano alla nascita di una disciplina innovativa e affascinante: l'*ingegneria del software quantistica*. Questa disciplina è plasmata dalla necessità di sviluppare in un futuro non troppo lontano sistemi software su larga scala in grado di sfruttare al meglio la potenza dei computer quantistici. A differenza della sua controparte classica, l'ingegneria del software quantistica richiede una comprensione profonda della natura quantistica della computazione. Gli ingegneri del software quantistico devono creare algoritmi e applicazioni in grado di sfruttare i qubit, i blocchi di costruzione fondamentali dei computer quantistici, per risolvere problemi complessi in settori come la crittografia, l'apprendimento automatico e molti altri.

Gli sviluppi della ricerca nell'ambito dell'informazione quantistica stanno portando il mondo verso quella che è definita come l'"*era dei quanti*", ovvero il momento in cui si raggiungerà una vera e propria *supremazia* delle macchine quantistiche nei confronti di quelle classiche. Proprio in tale contesto si inserisce questo lavoro di tesi, il cui obiettivo è nello specifico quello di contribuire alla ricerca nell'ambito dell'ingegneria del software quantistica.

2.2.2 Fondamenti dell'Informazione Quantistica

Per poter comprendere appieno come come sfruttare le capacità della computazione quantistica per il problema di test case selection, è necessario comprendere quali siano gli aspetti che ne pongono le basi.

I Qubit

I bit classici, ossia i così detti mattoni fondamentali della computazione classica, hanno un solo parametro tramite il quale possono essere manipolati: ogni bit può essere infatti inizializzato a 0 oppure ad 1. Questa caratteristica fondamentale dei bit rende la loro logica matematica piuttosto semplice. I possibili valori di un bit classico sono mostrati dalla Figura 2.1.

I qubit rappresentano invece il mattone fondamentale della computazione quantistica ed anche nel loro caso, in seguito ad un'azione di lettura del valore di un qubit,

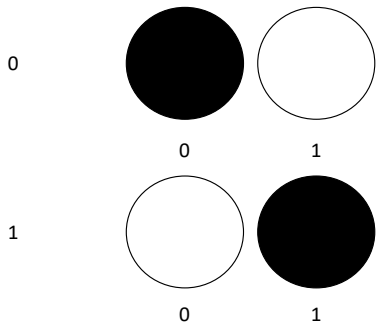

Figura 2.1: Possibili valori di un bit classico.

il risultato sarà sempre uno tra 0 ed 1. La differenza con i bit sta nel fatto che, sebbene *dopo la lettura* i valori possibili siano gli stessi, *prima della lettura* un qubit può esistere in uno stato detto di *sovrapposizione*.

Possiamo definire un qubit in questo modo:

7. Qubit

Un qubit rappresenta l'unità fondamentale di informazione quantistica e può esistere in uno stato equivalente alla combinazione lineare di due stati grazie al fenomeno quantistico della sovrapposizione. Inoltre, un qubit può essere correlato ad un altro grazie al fenomeno quantistico dell'entanglement, che impone la dipendenza dello stato di un qubit dallo stato dell'altro anche a distanza.

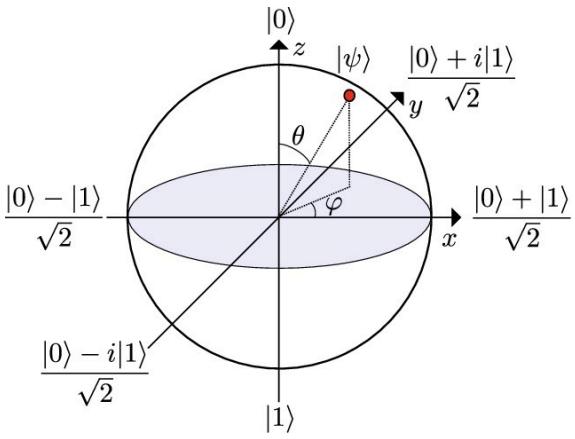

Figura 2.2: Rappresentazione degli stati assumibili da un qubit trami la sfera di Bloch

La sovrapposizione

Esattamente come i classici bit hanno uno stato, il quale può essere 0 o 1, anche i bit quantistici ne hanno uno. In particolare, due stati possibili nei quali possono esistere i qubit sono gli stati quantistici $|0\rangle$ e $|1\rangle$, ossia gli stati corrispondenti ai classici 0 ed 1. La differenza però sta nel fatto che un qubit può esistere anche in uno stato "*intermedio*" corrispondente alla combinazione lineare degli stati $|0\rangle$ e $|1\rangle$. Quando un qubit è in uno stato di questo tipo, esso è detto essere in *sovrapposizione*.

In virtù di questa proprietà dei qubit, lo stato di un qubit viene espresso nel modo seguente:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2.2.1}$$

I coefficienti α e β sono coefficienti che indicano quanto il qubit è in uno stato di base "0" e quanto è in uno stato di base "1". In altre parole, α rappresenta l'ampiezza di

probabilità di trovare il qubit nello stato di base "0", mentre β rappresenta l'ampiezza di probabilità di trovare il qubit nello stato di base "1".

Solo in seguito ad una misurazione il qubit "*collassa*" irreversibilmente ad un valore effettivo pari a 0 o ad 1.

L'Entanglement

L'entanglement è un fenomeno quantistico fondamentale che porta a correlazioni non classiche tra due o più particelle, indipendentemente dalla loro separazione spaziale. Questo concetto ha costituito una delle pietre miliari della meccanica quantistica ed ha aperto la strada a nuove intuizioni sulla natura del mondo quantistico.

Per ottenere un entanglement tra due particelle, ossia per correlarne gli stati, è necessario che esse vengano prodotte simultaneamente con un'interazione fisica.

Per comprendere l'entanglement, consideriamo l'esempio degli spin di due particelle come gli elettroni. Gli spin sono proprietà intrinseche delle particelle subatomiche e possono assumere valori che convenzionalmente sono descritti come positivo e negativo. Tuttavia, quando due particelle sono entangled, la misura dello spin di una particella istantaneamente determinerà lo spin dell'altra, indipendentemente dalla distanza tra di loro. Ad esempio, se misuriamo lo spin di una particella e otteniamo uno stato positivo, sappiamo istantaneamente che l'altra particella è nello stato negativo, anche se è molto lontana. Questa correlazione quantistica è ciò che rende l'entanglement così misterioso e sorprendente.

L'entanglement da solo però non ci permette di sapere con certezza quale sarà l'informazione che verrà trasmessa nel momento in cui eseguiremo la misurazione del qubit. Quindi, anche se le particelle sono entangled e sembra che una particella "comunichi" con l'altra istantaneamente, non è possibile sfruttare questa correlazione per esempio per trasmettere segnali o informazioni a velocità superluminali. Questo è dovuto al fatto che non si può controllare quale stato verrà misurato in una particella entangled. Le misure degli spin saranno casuali e non possono essere utilizzate per la trasmissione di dati o messaggi. Per ciò che concerne la comunicazione, va considerato che il passaggio di un'informazione, per quanto possibile, risulta sempre nella distruzione dell'informazione originale.

L'entanglement è stato sfruttato in applicazioni come la crittografia quantistica, che garantisce la sicurezza delle comunicazioni quantistiche. Inoltre, la manipolazione di stati entangled consente di svolgere calcoli più rapidamente di quanto sia possibile con computer classici. L'entanglement rimane uno dei fenomeni più affascinanti e misteriosi della meccanica quantistica, con implicazioni profonde per la nostra comprensione del mondo quantistico.

2.2.3 I limiti dell'informatica quantistica

La computazione quantistica è una disciplina attualmente in rapida crescita ed ha ampiamente dimostrato le sue enormi capacità. Per fare alcuni esempi dei campi in cui le potenzialità di tale disciplina vengono maggiormente esaltate possiamo elencare:

1. **Fattorizzazione di numeri:** gli algoritmi quantistici, come ad esempio quello di Shor, hanno una capacità impressionante di fattorizzazione di numeri anche molto grandi, in quanto riescono in tale intento con tempi estremamenti minori rispetto a quelli impiegati dagli algoritmi classici. Si tratta di una capacità che attualmente preoccupa i ricercatori in quanto è potenzialmente in grado di minacciare l'attuale infrastruttura di sicurezza software; proprio per questo si è a lavoro su alternative post-quantistiche.

2. **Simulazione quantistica:** i computer quantistici riescono a simulare in modo estremamente efficiente sistemi quantistici, il che porta ad implicazioni molto importanti in campi quali la produzione di farmaci o la comprensione di processi chimici e fisici.

3. **Ricerca ed ottimizzazione:** gli algoritmi quantistici si sono dimostrati anche incredibilmente efficaci in problemi di ricerca di soluzioni ottimali in spazi complessi, come l'algoritmo di Grover. Le applicazioni possibili di tale proprietà spaziano dall'ottimizzazione all'intelligenza artificiale, dalla crittografia alla logistica e così via.

Tuttavia, ci sono anche molti limiti che l'informatica quantistica presenta anche rispetto alle soluzioni classiche.

1. **Gestione degli errori:** la gestione degli errori quantistici è molto più complessa rispetto a quella utilizzata nel caso classico. Infatti, la coerenza delle informazioni nel caso quantistico è soggetta a fenomeni molto più vasti e complessi.

2. **Costi implementativi:** sebbene si tratti probabilmente di un problema solo temporaneo, non si può non considerare che attualmente la produzione di software e soprattutto di macchine quantistiche sia molto costosa. Ciò è dovuto anche alle caratteristiche non solo tecnologiche ma anche fisiche (come la temperatura) che consentono il corretto funzionamento di tali sistemi.

3. **Specifici domini applicativi:** sebbene possa sembrare controintuitivo, non tutti i problemi sono adatti ai computer quantistici; anzi, alcuni di essi sono risolvibili con prestazioni simili se non migliori con metodi classici.

4. **Disponibilità e scalabilità:** anche questo, sebbene sia un problema temporaneo, non può essere ignorato. Attualmente infatti, i computer quantistici sono ancora limitati in termini di scalabilità e disponibilità, poiché forniscono la computazione su un numero di qubit ancora troppo basso rispetto al loro potenziale massimo.

Dunque, sebbene l'informatica quantistica prometta delle rivoluzioni di grande portata in diversi settori, sono ancora molte le sfide che vanno affrontate per raggiungere tali obiettivi. Il futuro di questa disciplina dipende ancora dai progressi nella ricerca e questo progetto di tesi si propone di offrire il suo contributo.

2.3 Ottimizzazione Quantistica tramite Quantum Annealing

I sistemi quantistici risultano essere particolarmente adatti a risolvere problemi di ottimizzazione di natura combinatoriale. Dunque, questo lavoro di tesi si propone di

convertire il problema di test case selection in un problema di ottimizzazione risolvibile in maniera efficiente tramite l'utilizzo di un algoritmo quantistico. In particolare, l'algoritmo quantistico scelto è il *quantum annealing*, un approccio quantistico che secificherà per ogni caso di test il costo di esecuzione, il livello di statement coverage e la capacità di rilevare errori, per poi ricercare lo stato ad energia minima che rappresenti la soluzione migliore per l'esecuzione di una fase di test di regressione che sia il meno costosa ed allo stesso tempo il più efficace possibile.

In questa sezione verrà introdotto il concetto di ottimizzazione quantistica e ne sarà quindi proposta una possibile rappresentazione matematica. Infine, verrà descritto il processo di Quantum Annealing

2.3.1 Ottimizzazione Quantistica

Come già mostrato nella sezione precedente, la caratteristica fondamentale di un qubit è quella di poter esistere in uno stato che sia zero o uno con una certa probabilità (stato detto di "sovrapposizione") e che il valore effettivo del qubit è reso noto solo successivamente alla misurazione; in seguito alla misurazione il qubit collassa in uno stato pari a zero o uno e non può più ritornare in sovrapposizione. Ebbene, la filosofia che governa la computazione quantistica consiste proprio nell'esplorare e cercare soluzioni ottime in uno spazio probabilistico[44].

La maggior parte degli algoritmi utilizzati per risolvere problemi di ottimizzazione nell'ambito quantistico sono basati sull'algoritmo di Grover[45]; questo algoritmo esegue una ricerca a partire da uno spazio degli stati sconosciuto, basandosi sulla codifica dei requisiti della soluzione ottenuti tramite un oracolo. Tali oracoli[46][47] sono come delle black box che fanno sì che tali algoritmi raggiungano una complessità lineare.

Oltre a queste soluzioni, esistono alternative altrettanto valide. Per esempio, vi sono soluzioni basate sulle porte quantistiche come quelle messe a disposizione da Qiskit[48], che implementa il *quantum approximate optimization algorithm* (QAOA)[49].

La Scelta del Quantum Annealing

Per la realizzazione di questo lavoro di tesi è stata utilizzata un'altra alternativa ancora: l'ambiente D-Wave (https://dwavesys.com). Si tratta di un ambiente in grado di risolvere problemi NP-hard combinatoriali tramite un approccio detto di "quantum annealing" di ottimizzazione quantistica adiabatica[50][51]. L'approccio D-Wave si basa sulla definizione del problema di ottimizzazione da risolvere sotto forma di una hamiltoniana, che rappresenti contemporaneamente obiettivi e vincoli del sistema quantistico; il computer quantistico interpreta quindi il problema dal punto di vista energetico, ossia ha il compito di trovare la soluzione che risulti nel livello di energia minimo per il sistema, cioè la soluzione che minimizzi il valore dell'hamiltoniana che codifica il problema da risolvere.

La principale motivazione che ha condotto all'utilizzo del quantum annealing piuttosto che all'uso di un'altra tecnica quantistica è da ricercarsi nella natura dei diversi approcci sviluppati dalla computazione quantistica. Più nel dettaglio, la computazione quantistica può essere divisa in due approcci generali: quello basato sul modello dei gate quantistici e quello di quantum annealing. Il primo approccio, sviluppato da algoritmi come per esempio il QAOA di IBM, affronta il problema suddividendolo in una sequenza di operazioni primitive (dette gate) applicate ai singoli qubit e che portano a risultati di misurazione "digitali" ben definiti per certi stati di input, similmente all'approccio classico. Il secondo approccio, ossia quello di quantum annealing, traduce il problema da risolvere in un sistema quantistico di qubit di cui trovare, tramite una transizione graduale da uno stato quantistico all'altro, la configurazione ad energia minima, ossia la configurazione che minimizza una certa funzione obiettivo.

Entrambi gli approcci sono adatti a risolvere problemi di ottimizzazione combinatoriali e dunque la scelta dell'uno piuttosto che dell'altro dipende dalla natura del problema da risolvere. In generale, l'approccio di quantum annealing è utilizzato in contesti in cui è necessario rintracciare configurazioni che minimizzino una certa funzione obiettivo (come nei problemi di scheduling, ottimizzazione di percorsi e così via) ed in cui è accettabile anche una soluzione approssimata, ossia non ottimale ma comunque vicina all'ottimo. L'approccio basato sui gate è invece preferito per la

risoluzione di problemi per i quali sono stati già sviluppati degli algoritmi quantistici (per esempio per implementare l'algoritmo di Shor per la fattorizzazione o quello di Grover per la ricerca non strutturata) e nei casi in cui è fondamentale l'efficacia e dunque la precisione delle soluzioni finali. Sebbene quest'ultime considerazioni mantengano aperto il dibattito riguardo alla tecnica più adeguata per la risoluzione del problema di selezione dei casi di test, il prossimo punto spiegerà come mai la strada D-Wave sia stata praticamente obbligata.

In effetti, lo sviluppo pratico di computer quantistici basati su gate risulta ancora estremamente limitato, in quanto essi possono lavorare solo su un numero piuttosto ristretto di variabili. Tali limiti, sebbene ancora presenti, sono molto più attenuati negli ambienti D-Wave, che permetto l'esecuzione del quantum annealing su problemi di dimensioni molto maggiori. Lo stesso problema di selezione dei casi di test presenta nel caso del programma "grep" ben 806 variabili, ossia 806 qubit; attualmente non esiste alcuna macchina che implementi l'approccio basato sui gate che riesca a manipolare tanti qubit insieme nonché interazioni fra essi. Dunque, questa importante osservazione esclude definitivamente la posibilità di scegliere un algoritmo basato su gate quantistici.

La formulazione però degli obiettivi e dei vincoli del problema come un problema hamiltoniano risolvibile da un computer quantistico D-Wave tramite quantum annealing consiste nel definite i coefficienti quadratici e lineari di un *binary quadratic model* (BQM) che, nel caso di macchine quantistiche, sono specificati come un problema di tipo *quadratic uncostrained binary optimization* (QUBO). Prima di comprendere come funziona il quantum annealing, dobbiamo quindi comprendere in cosa consiste un problema QUBO.

Rappresentazione QUBO

I problemi QUBO utilizzano variabili binarie, per la precisione variabili che possono assumere valori nel dominio $0, 1$.

Un problema QUBO può essere definito utilizzando una matrice Q triangolare superiore di dimensione NxN di pesi reali ed utilizzando un vettore x di variabili binarie, con lo scopo di minimizzare la funzione:

$$f(x) = \sum_i (Q_{ii} \cdot x_i) + \sum_{i<j} (Q_{ij} \cdot x_i \cdot x_j) \tag{2.3.1}$$

dove i termini diagonali Q_{ii} sono i coefficienti lineari, mentre i termini diversi da zero e presenti sopra la diagonale Q_{ij} sono i coefficienti quadratici.

Modellando in questo modo il problema, abbiamo ottenuto una matrice quadrata triangonale superiore che codifica sulla diagonale gli obiettivi del problema, mentre presenta dei valori al di sopra della diagonale che codificano i vincoli del problema da risolvere. A questo punto, il problema così specificato è inviato al risolutore del computer quantistico adiabatico, il cui scopo è quello di trovare lo stato ad energia minima che, nel caso specifico del problema di test case selection, consiste nella combinazione di variabili, ossia di casi di test, che vanno selezionati per trovare una soluzione ottima al problema.

2.3.2 Il processo di Annealing

Il concetto principale del quantum annealing consiste nello specificare il problema da risolvere tramite l'utilizzo di qubit in uno stato di sovrapposizione e, seguendo il processo di annealing, collassarli ad uno stato classico (ossia zero o uno), rappresentando così la soluzione al problema che presenta un'energia minima. Si parte dunque da uno stato iniziale ad alta energia, in cui tutti i qubit sono in superposizione, per poi raggiungere uno stato finale ad energia minima, in cui tutti i qubit sono stati collassati.

È doverosa però un'importante precisazione. Il fatto che la funzione obiettivo spinga l'algoritmo verso una configurazione del sistema quantistico ad energia minima, non significa che il quantum annealing sia alla ricerca di soluzioni di risparmio energetico. La funzione hamiltoniana codifica i 3 obiettivi che sono: minimizzazione del costo di esecuzione, massimizzazione della copertura dei fallimenti e massimizzazione della copertura degli statement della suite. A partire da tale formulazione, l'algoritmo di quantum annealing semplicemente interpreta il problema da risolvere, ossia in questo caso quello di selezione dei casi di test, dal punto di vista energetico. Cioè, codifica ogni caso di test come un singolo qubit e dunque la suite di

test iniziale come un sistema quantistico corrispondente ad una sequenza di qubit tutti in superposizione (stato di energia iniziale massimo); da qui, cerca di trovare lo stato ad energia minima, ossia la configurazione di qubit che, collassati in bit, corrispoda al valore minimo della funzione obiettivo hamiltoniana, la quale indica il valore energetico (ossia la bontà di una soluzione) che una configurazione del sistema quantistico assume. In poche parole, il valore energetico di una configurazione del sistema quantistico indica la sua bontà come soluzione.

Il processo di quantum annealing si compone di più fasi e può essere visualizzato grazie alla Figura 2.3, che rappresenta un diagramma energetico per un sistema composto da un solo qubit. Si tratta di un diagramma che cambia con l'avanzare del tempo. Inizialmente, il processo parte con un minimo unico, ossia vi è una sola curva di minimo. Man mano che il processo avanza, le possibilità energetiche che il sistema può raggiungere vengono divise fra loro generando uno stato con due curve di minimo equivalenti. Una volta conclusosi il processo di annealing, solo una delle due curve corrisponderà allo stato ad energia minima, generando così una curva più profonda che corrisponderà alla soluzione più probabile.

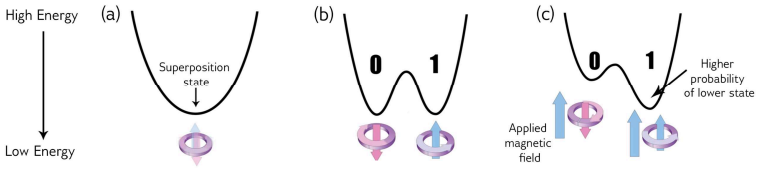

Figura 2.3: Il processo di quantum annealing

CAPITOLO 3

Metodologia di Ricerca

Questo capitolo presenta una descrizione dettagliata di tutti i procedimenti che hanno composto lo svolgimento del presente lavoro di tesi. Il lavoro svolto si è composto di più fasi, ognuna delle quali sarà analizzata con precisione nel presente capitolo e di cui si fornisce qui una sommaria descrizione.

La prima fase è stata sicuramente la fase di definizione formale del problema da risolvere, durante la quale si è definita a livello matematico l'hamiltoniana che rappresenta il problema di test case selection, tramite i vincoli da rispettare e gli obiettivi da raggiungere.

Successivamente, per poter eseguire una sperimentazione della soluzione proposta su casi reali, si è passati all'organizzazione ed alla manipolazione di tutti i dati utili di alcuni programmi della "software-artifact infrastructure repository" (SIR).

I dati così ottenuti sono stati poi utilizzati nella fase successiva per la codifica, eseguita tramite linguaggio Python sottoforma di matrice triangolare superiore del problema formalizzato inizialmente solo come hamiltoniana. Da qui, la matrice ottenuta è stata inviata al risolutore di D-Wave per l'esecuzione di 30 sperimentazioni per ciascuno dei programmi della SIR oggetto di studio.

Infine, sono state acquisite, durante l'esecuzione degli esperimenti, le metriche di ipervolume e di tempo di esecuzione del processore quantistico, con lo scopo di eseguire confronti con le tecniche già utilizzate in questo ambito, descitte nella Sezione 2.1, e con la tecnica di simulated annealing.

3.1 Fase di Formalizzazione del Problema di Selezione dei Casi di Test

Si comincia con la descrizione delle scelte che hanno portato alla formulazione dell'hamiltoniana da risolvere tramite quantum annealing.

3.1.1 Definizione dell'Hamiltoniana

La selezione dei casi di test è stata descritta come il problema di trovare un sottoinsieme della suite di partenza che rappresenti un ottimo di Pareto rispetto a delle specifiche funzioni di fitness. In particolare, tali funzioni di fitness corrispondono ai seguenti tre criteri contrastranti: copertura degli statement, costo di esecuzione e storico del rilevamento di fallimenti.

Così come nel caso, visto nella Sezione 2.2, delle strategie greedy, anche per il quantum annealing bisogna optare per una formulazione del problema che concerna un'unica funzione da ottimizzare. Ciò è dovuto al fatto che, come si è visto, i sistemi D-Wave di quantum annealing sono in grado di risolvere problemi espressi in termini di funzioni hamiltoniane.

Il Problema della Copertura Minima

Il primo passo verso la formulazione dell'hamiltoniana obiettivo consiste nell'osservare che il problema che si cerca di risolvere non è che una variante del classico problema di copertura minima.

Possiamo definire in questo modo il problema di copertura minima: dati un insieme S ed un insieme T i cui elementi t_i sono sottoinsiemi di S, si desidera trovare il numero minimo di elementi t_i che coprano S, ossia la cui unione sia S. La collezione finale dei sottoinsiemi di S è detta *copertura minima*, indicata come T'; tale copertura potrebbe essere anche *sub-ottima*.

Nel caso specifico, selezionare i casi di test da una suite di test in modo da non perdere la copertura di statement della suite di partenza ma minimizzando il numero di casi di test da eseguire, equivale proprio a risolvere un problema di copertura minima. In tal caso, S rappresenta l'insieme di tutti gli statement del programma da

testare, mentre T rappresenta i casi di test della suite di partenza sotto forma degli statement che coprono (ossia sottoinsiemi di S).

Il problema di selezione dei casi di test di regressione, risolto dalle strategie discusse nella Sezione 2.1, presenta però una variazione rispetto al problema originale di copertura minima. Infatti, l'obiettivo principale è più preciso rispetto alla semplice minimizzazione del numero di casi di test della suite finale; l'obiettivo principale consiste nel costruire una suite di test che minimizzi il costo di esecuzione e contemporaneamente massimizzi la probabilità di rilevare fallimenti nel codice, cercando, appunto, di non diminuire il valore finale della copertura degli statement rispetto alla suite di partenza, ossia cercando di massimizzare la copertura degli statement; infatti, il massimo valore di statement coverage raggiungibile dalla suite finale corrisponde alla statement coverage della suite originale.

Partendo dunque dalla concezione del problema appena espressa, appare evidente che dunque l'input iniziale dell'algoritmo non sia altro che la suite di test di partenza. Per ciascun caso di test in essa contenuta, siamo a conoscenza del costo di esecuzione, dello storico riguardo il rilevamento di fallimenti nel passato e degli statement che esegue.

Prima di descrivere più nel dettaglio in che modo vada formulata l'hamiltoniana finale, si noti una caratteristica del problema in esame. Infatti, ogni caso di test esegue determinati statement del programma da testare, i quali potrebbero essere eseguiti in comune con altri casi di test della suite iniziale. Quindi, affinché la suite da costruire massimizzi il valore finale della copertura degli statement, è sufficiente che per ogni statement sia considerato solo uno dei casi di test che lo esegue. Ciò è dovuto al fatto che la valutazione della copertura del codice della fase di test non dipende dalla frequenza con cui uno statement è stato eseguito, bensì solo dalla sua avvenuta esecuzione (anche singola).

3.1.2 Il Problema QUBO da Risolvere

Si veda dunque il processo di costruzione del problema QUBO definitivo.

Tabella 3.1: Definizioni Importanti

#	Definizione
i	indice univoco per l'identificazione di un caso di test
k	indice univoco che rappresenta uno statement
T_k	lista di tutti i casi di test che eseguono lo statement numero k
x_i	variabile binaria che specifica se nella soluzione dell'algoritmo l'i-esimo caso di test è stato inserito
$cost(\tau_i)$	costo di esecuzione dell'i-esimo test
e_i	variabile binaria che indica se l'i-esimo caso di test ha rilevato errori in passato o meno
α	fattore di peso il cui scopo consiste nel modulare il rapporto di contrasto tra gli obiettivi del problema
P	coefficiente di penalità il cui scopo è quello di regolare l'importanza dei vincoli nell'equazione finale

Definizioni

Per poter comprendere appieno come modellare il problema da risolvere, sono date delle definizioni che specificano il significato di variabili e funzioni necessarie per la formulazione dell'algoritmo. La Tabella 3.1 racchiude tutte le informazioni.

QUBO per la Selezione dei Casi di Test

Per poter risolvere il problema, è necessario modellarlo come un problema QUBO, le cui caratteristiche sono state descritte nella Sezione 2.3.1. Il modello QUBO, anche conosciuto come *Unconstrained Binary Quadratic Programming* (UBQP), rappresenterà contemporaneamente gli obiettivi ed i vincoli del problema di test case selection, in modo da poter essere risolto dal risolutore del computer quantistico adiabatico di D-Wave, il cui scopo consisterà nel rilevare lo stato ad energia minima equivalente

alla combinazione delle variabili (ossia dei casi di test) del problema QUBO che dovranno essere selezionate nella costruzione della suite di test finale.

Tutti i problemi che seguono lo schema QUBO sono specificati tramite un'hamiltoniana, che descrive sotto forma di somma gli obiettivi ed i vincoli che devono essere rispettati dalla soluzione. L'hamiltoniana sarà espressa come un *Binary Quadratic Model* (BQM) e dunque codificata come una matrice BQM che è proprio l'oggetto da inviare al risolutore adiabatico.

Gli algoritmi più classici utilizzati per i problemi di ottimizzazione corrispondono ai modelli lineari, costituiti da una funzione da minimizzare (o massimizzare), da una serie di vincoli e diverse variabili. Il problema di copertura minima è esso stesso risolvibile tramite un modello lineare. Seguendo il lavoro di Glover, F., Kochenberger, G., Hennig, R. et al.[52], è possibile riformulare un problema di programmazione lineare in un problema QUBO.

Il primo passo è definire gli obiettivi principali del problema di selezione dei casi di test. In particolare, tali obiettivi rappresenteranno la parte lineare del problema; in altre parole, essi saranno espressi in termini delle singole variabili.

Per questo motivo, gli obiettivi corrispondono a: i) minimizzare il costo della suite di test, ii) massimizzare la capacità della suite di rilevare errori durante l'esecuzione.

Il primo obiettivo è scritto come espressione BQM nel seguente modo:

$$\alpha \sum_{i=1}^{|\Gamma|} [x_i \cdot cost(\tau_i)] \tag{3.1.1}$$

Riguardo al secondo obiettivo, è doverosa una precisazione. Infatti, si è scelto di trasformare il secondo obiettivo in un problema da minimizzare per semplificazione. Lo si traduce di seguito in una espressione BQM:

$$-(1-\alpha) \sum_{i=1}^{|\Gamma|} (e_i \cdot x_i) \tag{3.1.2}$$

Si noti la presenza del coefficiente α all'interno delle due funzioni obiettivo. Si tratta di un fattore di peso ($0 < \alpha < 1$) il cui scopo è quello di permettere eventualmente di stabilire la preferenza verso un obiettivo piuttosto che l'altro. Si tratta di un approccio a somma pesata[53], ossia quando $\alpha = 0.5$, l'importanza dei due obiettivi è la stessa. Nel momento in cui α assume altri valori, uno dei due obiettivi avrà maggiore rilevanza nella ricerca del risultato finale.

Dunque, la funzione finale da minimizzare consta di due parti, è detta cioè *composita* ed è così formulata:

$$H = \alpha \sum_{i=1}^{|\Gamma|} [x_i \cdot cost(\tau_i)] - (1 - \alpha) \sum_{i=1}^{|\Gamma|} (e_i \cdot x_i) \tag{3.1.3}$$

Arrivati a questo punto, è necessario definire i vincoli del problema di programmazione lineare che si sta definendo. Nel caso specifico, i vincoli rappresentano la necessità di ottenere una suite di test finale che mantenga lo stesso livello di copertura degli statement della suite iniziale. Per ottenere tale risultato, è sufficiente che almeno un caso di test, tra tutti quelli nella suite originale che eseguono quello statement, sia selezionato.

Per maggiore chiarezza, si propone un semplice esempio con la Tabella 3.2. Tramite la tabella appare evidente che, per coprire tutti gli statement $1, 2, 3, 4$, non è necessario selezionare tutti i casi di test τ_1, τ_2, τ_3, in quanto basta selezionare τ_1, τ_2, cioè τ_3 risulta essere ridondante dal punto di vista della statement coverage.

Dunque, per poter convertire questa proprietà del problema in un vincolo che fosse applicabile al problema lineare in formulazione, si è deciso di porre l'attenzione sugli statement e non sui casi di test. L'obiettivo del vincolo deve essere assicurarsi che ogni statement (eseguito dalla suite originale) abbia almeno un caso di test della suite finale che lo esegua. Se questo vincolo viene rispettato, si è certi che il livello di statement coverage della suite risultante sia equivalente a quello della suite originale (ossia il massimo possibile).

Poiché le variabili del problema sono binarie, il modo più semplice per esprimere il vincolo descritto è il seguente::

Tabella 3.2: Esempio di casi di test e statement

Caso di Test	Statement
τ_1	1,2
τ_2	3,4
τ_3	1,2,3

$$\sum_{i \in T_k} (x_i) \geq 1 \tag{3.1.4}$$

Il vincolo così espresso dichiara che almeno un caso di test che esegue lo statement k deve essere selezionato nella costruzione della suite finale. La lista dei casi di test che eseguono il k-esimo statement è T_k.

Poiché un programma è fatto di più statement e si è interessati soprattutto agli statement già eseguiti dalla suite originale, applichiamo il vincolo per ogni statement da coprire. I vincoli ottenuti sono espressi dunque così:

$$\sum_{k} (\sum_{i \in T_k} (x_i - 1)^2) \tag{3.1.5}$$

Il problema lineare è ora completo. Per poterlo convertire in un problema QUBO bisogna costruire l'espressione hamiltoniana includendo in essa, oltre agli obiettivi, i vincoli appena descritti; ciò al fine di ottenere un problema che sia apputo "*unconstrained*". Per fare ciò è necessario aggiungere una costante di penalizzazione (P)[52], il cui compito è quello di regolare l'importanza del vincolo all'interno dell'espressione hamiltoniana per combinare la funzione obiettivo con la funzione di vincolo.

Il vincolo P può essere considerato in modo empirico come equivalente al limite superiore più 1 della funzione obiettivo del problema, in modo che tale penalità sia coerente con il dominio applicativo e che dunque sia influente nella ricerca[54].

L'equazione QUBO che si ottiene è dunque:

$$H = \alpha \sum_{i=1}^{|\Gamma|}[x_i \cdot cost(\tau_i)] - (1-\alpha)\sum_{i=1}^{|\Gamma|}(e_i \cdot x_i) + P \cdot \sum_k (\sum_{i \in T_k} (x_i - 1)^2) \tag{3.1.6}$$

La parte dell'equazione che rappresenta il vincolo, peso incluso, può essere semplificata. Ecco i passaggi:

$$P \cdot \sum_k (\sum_{i \in T_k} (x_i - 1)^2) = P \cdot \sum_k (\sum_{i,j \in T_k} (x_i^2 + 1^2 + 2x_i x_j - 2x_j)) \tag{3.1.7}$$

Rimuovendo i quadrati e le costanti in quanto ininfluenti:

$$P \cdot \sum_k (\sum_{i,j \in T_k} (-x_i + 2x_i x_j)) \tag{3.1.8}$$

Infine, semplificando ulteriormente si ottiene:

$$\sum_k (\sum_{i,j \in T_k} (-Px_i + 2Px_i x_j)) \tag{3.1.9}$$

Dunque, l'espressione finale BQM (nel caso specifico QUBO) dell'hamiltoniana è la seguente:

$$H = \alpha \sum_{i=1}^{|\Gamma|}[x_i \cdot cost(\tau_i)] - (1-\alpha)\sum_{i=1}^{|\Gamma|}(e_i \cdot x_i) + \sum_k (\sum_{i,j \in T_k} (-Px_i + 2Px_i x_j)) \tag{3.1.10}$$

Sebbene la formulazione a livello matematico del problema sia completata, prima di poter passare alla traduzione in codice è necessario superare un altro passaggio. Risulta infatti necessario raccogliere tutte le informazioni di costo, storico degli errori e copertura del codice dai programmi della SIR scelti per la sperimentazione della strategia proposta in questo lavoro di tesi.

3.2 Fase di Raccolta Dati dai Programmi della SIR: Metodologia Utilizzata

Per poter eseguire delle sperimentazioni esaustive e realistiche della soluzione quantistica al problema di selezione dei casi di test, è necessario munirsi dei dati necessari. Come già specificato, la SIR è una repository che mette a disposizione programmi industriali open-source, in modo da raccogliere le informazioni necessarie tramite l'utilizzo di specifici tool.

In questo lavoro di tesi, per la risoluzione del problema sono stati considerati tre criteri differenti, utilizzati anche in lavori precedenti[13][33][55][28]. Tali criteri sono: *statement coverage*, *costo di esecuzione dei casi di test*, *storico dei fallimenti rilevati*.

Statement Coverage

Il criterio di statement coverage per il problema QUBO precedentemente descritto è piuttosto cruciale. L'informazione che si è interessati a risolvere consiste nel descrivere, per ogni statement raggiunto dalla fase di regression testing del programma in esame, quali sono i casi di test della suite di partenza che lo hanno eseguito.

Per poter ottenere questa informazione è necessario l'utilizzo dello strumento *gcov* del compilatore C *GNU (gcc)*. In particolare, gcov è in grado di tenere traccia, per ogni caso di test, degli statement che esegue.

Execution cost

Si è scelto di calcolare il costo di esecuzione dei singoli casi di test non sulla base dei loro tempi di esecuzione; tale misurazione sarebbe infatti troppo legata a fattori esterni al problema e soprattutto alla macchina sulla quale si stanno eseguendo i test stessi (hardware, sistemi operativi ecc...). Dunque, il calcolo del costo di esecuzione dei casi di test viene eseguito tramite il conteggio delle istruzioni elementari effettivamente eseguite, in coerenza con i precedenti lavori[33][28][27].

Anche in questo caso, si usa gcov per il calcolo della frequenza di esecuzione di ogni blocco di base (ossia sequenza di statement) che compone ogni linea di codice.

Per blocco di base si intende una sezione lineare del codice non ramificata e con un solo punto di ingresso ed un solo punto di uscita. Si è preferito utilizzare il conteggio dei blocchi piuttosto che delle righe stesse, in quanto una riga potrebbe contenere più ramificazioni o chiamate a funzione.

Dunque, la funzione più volte menzionata $cost(\tau_i)$ esegue esattamente la valutazone appena descritta.

Past Fault History

La SIR mette a disposizione delle versioni dei programmi che contengono al loro interno dei fault *innestati*, specificando, tramite l'uso della così detta *fault matrix*, se un determinato caso di test è in grado di rilevare gli errori o meno. Questa informazione può quindi essere tradotta in un valore binario da associare ai corrispondenti casi di test.

Una volta formulato matematicamente il problema e raccolti tutti i dati per le sperimentazioni della soluzione proposta, la fase successiva consiste nella codifica delle procedure che portano alla generazione, tramite risolutore quantistico, della soluzione desiderata.

3.3 Fase di Implementazione

La fase di implementazione, sviluppata tramite linguaggio Python, consiste nella progettazione e poi codifica di tutti i procedimenti che, a partire dai dati ottenuti durante la fase precedente, portano alla risoluzione del problema tramite l'esecuzione del risolutore quantistico di D-Wave, oltre che all'analisi empirica dei risultati ottenuti.

Si consideri che l'implementazione è finalizzata ad eseguire delle sperimentazioni sui 4 programmi della SIR: flex, gzip, grep e sed. Dunque, ciascuna procedura che sarà mostrata di seguito è da considerarsi eseguita per ognuno dei programmi sotto esame.

L'intero codice sorgente del progetto è presente nella seguente repository GitHub: https://github.com/AntonioTrovato/Quantum-Regression-Test-Case-Selection.

3.3.1 Lettura dei Valori Lineari e Quadratici del Problema QUBO

Per poter arrivare alla codifica completa del problema QUBO definito precedentemente, è necessario ottenere e manipolare adeguatamente i dati ottenuti dalla SIR.

Per quanto riguarda la funzione obiettivo, per conoscere lo storico dei fallimenti rilevati dai casi di test, viene eseguita una lettura dei file contenenti le *fault-matrix* di ogni caso di test. In questo modo, è possibile stabilire quali casi di test hanno effettivamente rilevato errori in passato e quali non li hanno rilevati. La Figura 3.1 rappresenta il corrispondente codice.

Il tool gcov fornisce un file json contenente tutte le informazioni necessarie al calcolo della frequenza di esecuzione dei casi di test ed all'identificazione, per ogni statement del programma sotto esame, dei casi di test che li eseguono. Dunque, i due criteri di costo e copertura dei casi di test vengono calcolati in un'unica procedura, che è mostrata nelle Figure 3.2 e 3.3.

```python
#let's make a function to read the fault matrices
#IMPORTANT: all the fault-matrix files must be renamed as "fault-matrix".txt and must be written using the
 same standard used by the files of this project (i-th line->e0e1e2e3..., where ej is 0 if the i-th test
 found a bug when launched on the j-th version)
def get_fault_list(program_name:str):
    """This function opens the fault-matrix file of a sir programs and make a list of binary values for each
      test case of that program to indicate whether a test case found or not a bug in at leat 1 of the
     available versions of the program"""
    #open the fault-matrix file of the desired SIR program
    program_file = open("SIR_Programs/"+program_name+"/fault-matrix.txt")
    lines = program_file.readlines()

    #we need a list which elements represent test cases, the i-th element is 1 if the i-th test case
    #discovered a fault in the past, 0 otherwise
    faults_test_by_test = list()

    i = 0
    for line in lines:
        if "1" in line:
            faults_test_by_test.append(1)
        else:
            faults_test_by_test.append(0)
        i += 1

    program_file.close()

    return faults_test_by_test
```

Figura 3.1: Funzione Python per la lettura delle fault-matrix

3.3.2 Codifica del Problema QUBO di Selezione dei Casi di Test

Per codificare il problema QUBO da risolvere, viene generata la matrice triangolare da inviare al risolutore quantistico. Si tratta di una matrice triangolare inferiore, in quanto sulla diagonale appaiono i termini lineari del problema, mentre al di sotto della diagonale appaiono i termini quadratici.

Nella Figura 3.4 è mostrato il codice per la generazione della matrice.

3.3.3 Esecuzione del Risolutore Quantistico

La matrice finale ottenuta va inviata ad un risolutore quantistico messo a disposizione da D-Wave. Si può accedere ad un tale risolutore solo tramite l'utilizzo di una chiave privata e gratuitamente solo per un periodo limitato di tempo.

Nel caso di questo lavoro di tesi, sono state eseguite 30 sperimentazioni indipendenti per ogni programma in esame, allo scopo di tener conto della natura casuale della computazione quantistica.

```
#the next function is able to research into the json coverage information file of each test case
#of each sir program to gather information about the single test cases costs and coverage
def cost_and_coverage_information_gathering(program_name:str):
    """The aim of this function is to obtain a dictionary that for each test case of a program indicates its
      cost, and a dictionary that, for each code line of the program to test, makes a list of all the test
     cases that run that line"""
    test_case_execution_cost = 0

    execution_cost_test_by_test = dict()
    executed_lines_test_by_test = dict()

    for test_case in range(sir_programs_tests_number[program_name]):
        #to open the correct file, we must remember that the folders and the json files are
        #numbered from 1 and not from 0
        if program_name == "gzip":
            json_name = "allfile"
        else:
            json_name = program_name
        test_case_json = open("SIR_Programs/"+program_name+"/json_"+program_name+"/t"+str(test_case+1)
         +"/"+json_name+str(test_case+1)+".gcov.json")

        #read the JSON object as a dictionary
        json_data = json.load(test_case_json)
```

Figura 3.2: Prima parte calcolo costi e statement coverage

La procedura di esecuzione del risolutore quantistico è mostrata nelle Figure 3.5 e 3.6.

3.3.4 Metriche per Valutazione dei Risultati

L'Ipervolume

Per poter eseguire dei confronti tra la soluzione proposta in questo lavoro di tesi e le soluzioni MOGA e greedy di cui si è discusso nella Sezione 2.1, è necessaria una metrica che sia in grado di valutare l'efficacia delle suite di test; ciò permetterebbe di paragonare l'efficacia di suite differenti, ottenute come risultati dell'esecuzione di algoritmi differenti. Un metodo piuttosto semplice per valutare l'efficacia di una suite di test e che consideri gli obiettivi del problema di selezione dei casi di test, consiste nel calcolare la percentuale di fallimenti che sono rilevati dalla suite di test in esame, dato il suo costo di esecuzione.

Nel lavoro di Panichella, Oliveto, Di Penta e De Lucia[33], il confronto tra le

```
        #for programs tested above more than one file, the initial row of a file will start from
        #the final row of the preceding file
        i = 0
        for file in json_data["files"]:
            line_count_start = i
            for line in file["lines"]:
                #if a line is executed, we want to remember FOR THAT LINE which are the tests
                #that executed it, and we want to increment the execution cost
                if line["unexecuted_block"] == False:
                    #the test suite exec cost = sum of the exec freq. of each executed basic block
                    #by each test case
                    #test_suite_execution_cost += line["count"]
                    test_case_execution_cost += line["count"]

                    if (line_count_start + line["line_number"]) not in executed_lines_test_by_test:
                        executed_lines_test_by_test[line_count_start + line["line_number"]] = [test_case]
                    else:
                        executed_lines_test_by_test[line_count_start + line["line_number"]].append(test_case)
                    executed_lines_test_by_test[line_count_start + line["line_number"]].sort()
                i = line["line_number"]

        #saving the total amount of execution cost for this test case and resetting for the next
        execution_cost_test_by_test[test_case] = test_case_execution_cost
        test_case_execution_cost = 0

        test_case_json.close()

    return execution_cost_test_by_test, executed_lines_test_by_test
```

Figura 3.3: Seconda parte calcolo costi e statement coverage

tecniche MOGA e greedy viene eseguito utilizzando la metrica dell'*ipervolume*; si è scelto dunque di considerare la stessa metrica anche per l'algoritmo di quantum annealing, in modo da poter aggiungere quast'ultimo ai confronti.

La metrica dell'ipervolume viene utilizzata per poter quantificare l'efficacia di un fronte di Pareto, in quanto è in grado di misurare il volume che è racchiuso tra un fronte di Pareto $P = \{p_1, ..., p_h\}$ (ossia l'insieme delle soluzioni generate dall'esecuzione di un algoritmo di ottimizzazione), rispetto ad un fronte ideale R[56][57]. Ciò significa che l'ipervolume è in grado di valutare la lontananza tra P ed il fronte ideale R. Quindi, minore è il valore dell'ipervolume, maggiore sarà la qualità delle soluzioni contenute in P[57]. L'ipervolume viene calcolato sulla base delle funzioni obiettivo che definiscono il problema da risolvere che, nel caso di questo lavoro di tesi, sono: il costo di esecuzione e la percentuale di fallimenti rilevati da ogni soluzione (ossia da ogni suite di test generata da un algoritmo di ottimizzazione). Dunque, il fronte ideale R consiste in un insieme di soluzioni ideali, ossia un insieme di suite di test che sono in grado di rilevare tutti i fallimenti per ciascun valore di

```python
def create_adjvector_bqm(sir_program, alpha, P):
    """This function is the one that has to encode the QUBO problem that the D-Wave Hybrid Sampler will have
      to solve. The QUBO problem specifies the optimization to solve and a quadratic binary unconstrained
     problem"""
    qubo = dimod.AdjVectorBQM(dimod.BINARY)

    #linear coefficients, that are the diagonal of the matrix encoding the QUBO
    for i in range(sir_programs_tests_number[sir_program]):
        cost = alpha * test_cases_costs[sir_program][i] - (1 - alpha) * faults_dictionary[sir_program][i]
        qubo.set_linear(i, cost)

    #quadratic coefficient, that are the lower part of the QUBO matrix
    for k in coverage[sir_program].keys():
        test_cases = coverage[sir_program][k]
        for i in test_cases:
            for j in test_cases:
                if i < j:
                    qubo.set_linear(i,qubo.linear[i]-P)
                    qubo.set_linear(j,qubo.linear[j]-P)
                    try:
                        qubo.set_quadratic(i, j,qubo.quadratic[i,j] + 2 * P)
                    except:
                        qubo.set_quadratic(i, j, 2 * P)

    return qubo

```

Figura 3.4: Generazione della matrice che codifica obiettivi e vincoli del problema

costo di esecuzione. In questo modo, l'indicatore di ipervolume pesato diventa un indicatore bi-dimensionale, come mostrato dalla Figura 3.7.

Poiché il quantum annealing, così come le soluzioni greedy, non prevede la costruzione di un fronte di Pareto, in quanto non risolve un problema multi obiettivo, bensì unisce ogni funzione obiettivo in un'unica funzione, per ogni soluzione costruita dal quantum annealing, ossia per ogni suite di test risultante dall'esecuzione di tale algoritmo, vengono calcolati separatamente, come mostrato dalla Figura 3.8, il costo di esecuzione e la percentuale di fallimenti che è in grado di rilevare. Così facendo, è possibile eseguire confronti con una soluzione ideale nel modo che viene descritto di seguito.

Consideriamo $P = \{p_1, ..., p_h\}$ come un insieme di soluzioni, ossia un insieme di suite di test. Sia dunque $f(p_i)$ la percentuale di fallimenti rilevati dalla soluzione $p_i \in P$ e sia $cost(p_i)$ il corrispondente costo di esecuzione. Sia ora $R = \{r_1, ..., r_h\}$ l'ideale insieme corrispondente di soluzioni, ossia l'insieme di suite di test in grado di rilevare tutti i fallimenti a diversi livelli di costo di esecuzione; in particolare, si

```python
#I want to run the sampler 30 times to obtain different results for each sir program
for sir_program in sir_programs:
    response = None
    for _ in range(tot_run):
        #for each iteration get the result
        response = sampler.sample_qubo(qubos_dictionary[sir_program].to_numpy_matrix())
        sample_dictionary = {i: response.samples()[0][i] for i in range(len(response.samples()[0]))}
        responses_dictionary[sir_program].append(sample_dictionary)
        run_times_dictionary[sir_program].append(response.info["run_time"])
    #compute I_CE and save the I_CE points to make the final plots
    I_H = 0
    for solution in range(len(responses_dictionary[sir_program])):
        if solution == 0:
            solution_cost = get_solution_cost(solution,responses_dictionary[sir_program],sir_program)
            I_H += solution_cost
            total_solution_faults = 0
            for test_case, var_x in responses_dictionary[sir_program][solution].items():
                if var_x:
                    total_solution_faults += faults_dictionary[sir_program][test_case]
            I_CE_points[sir_program].append((solution_cost,total_solution_faults))
        else:
            solution_cost = get_solution_cost(solution,responses_dictionary[sir_program],sir_program)
            I_H += (get_solution_cost(solution+1,responses_dictionary[sir_program],sir_program)
             -solution_cost)*(1-get_total_fault_coverage(solution,responses_dictionary[sir_program],
             sir_program))
            total_solution_faults = 0
            for test_case, var_x in responses_dictionary[sir_program][solution].items():
                if var_x:
                    total_solution_faults += faults_dictionary[sir_program][test_case]
            I_CE_points[sir_program].append((solution_cost,total_solution_faults))
```

Figura 3.5: Prima parte dell'esecuzione del sampler

impone che $cost(p_i) = cost(r_i)$ e che $f(r_i) = 1$ per ogni r_i. Il calcolo dell'ipervolume contenuto tra questi punti si esegue come la somma dei rettangoli di larghezza $[cost(p_{i+1}) - cost(p_i)]$ e di altezza $[f(r_i) - f(p_i)]$, dunque si ha che:

$$I_H(P) = cost(p_1) + \sum_{i=1}^{h} [cost(p_{i+1} - cost(p_i)] \cdot [1 - f(p_i)] \tag{3.3.1}$$

Come detto in precedenza, la metrica dell'ipervolume è una funzione inversa, minore è il suo valore e migliore è l'efficacia media delle suite di test costruite da un particolare algoritmo di ottimizzazione. Dunque, a partire da $I_H(P)$ possiamo esprimere la metrica dell'ipervolume come la percentuale dell'area (ipervolume) al di sotto del fronte ideale R nel modo seguente:

$$I_{CE}(P) = \frac{I_H(P)}{cost(p_h)} \tag{3.3.2}$$

```
37      if I_H<0: I_H *= -1
38      I_CE_dictionary[sir_program] = I_H/get_solution_cost(len(responses_dictionary[sir_program])-1,
         responses_dictionary[sir_program],sir_program)
39      #print the time information of the last run of the sir program
40      run_time = response.info["run_time"]/1000000
41
42      #Show the results
43      print("INFORMATION FOR " + sir_program + ":")
44      print("I_CE: " + str(I_CE_dictionary[sir_program]))
45      print("TOTAL TIME: ", run_time, "seconds\n")
```

Figura 3.6: Seconda parte dell'esecuzione del sampler

dove $cost(p_h)$ corrisponde al costo dell'ultimo punto di P.

La metrica dell'ipervolume viene calcolata durante l'esecuzione del sampler quantistico, come mostrato nelle Figure 3.5 e 3.6.

Il Tempo di Esecuzione

Per poter mostrare le incredibili prestazioni in termini di performance degli algoritmi quantistici rispetto a quelli classici, è stata raccolta anche un'altra metrica: il tempo di esecuzione.

Il tempo di esecuzione dell'algoritmo di quantum annealing considera nello specifico il tempo di esecuzione del processore quantistico del sampler messo a disposizione da D-Wave per ciascun programma usato nella sperimentazione. Nel caso di questo progetto di tesi, il sampler utilizzato è stato un D-Wave Hybrid Binary Quadratic Model (Version2).

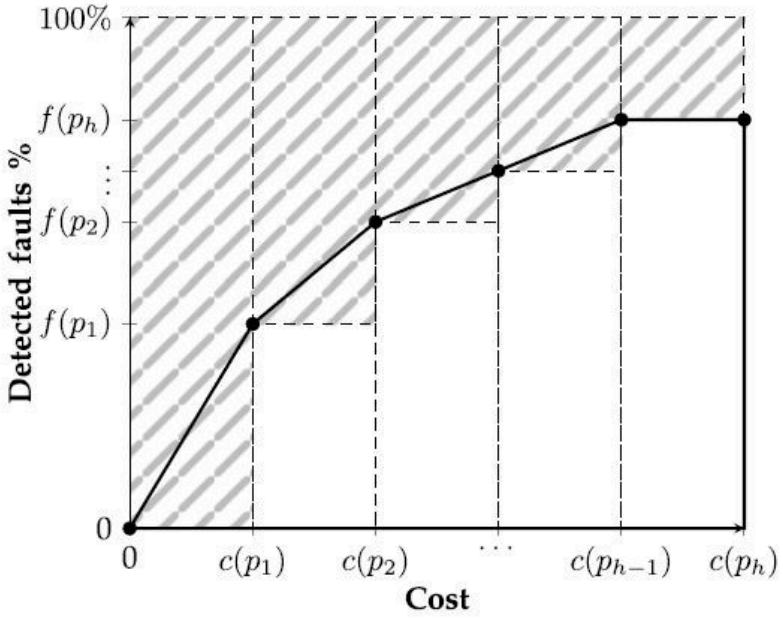

Figura 3.7: Metrica dell'ipervolume basato su costo e capacità di rilevamento dei fallimenti delle suite di test

Per comprendere la variabilità e soprattutto l'affidabilità della metrica di tempo di esecuzione del processore quantistico, ne è stato calcolato l'intervallo di confidenza tramite il metodo del bootstrapping. Il calcolo dell'intervallo di confidenza è particolarmente utile nel caso quantistico, in quanto i tempi di esecuzioni di macchine di questo tipo sono soggetti a fluttuazioni più o meno evidenti a causa di rumore o dell'hardware utilizzato per l'esecuzione.

Nella Figura 3.9 è mostrato il codice che implementa la procedura di calcolo dell'intervallo di confidenza del tempo di esecuzione del processore quantistico tramite boostrapping.

```
def get_solution_cost(solution_number,solutions_dictionary_list,sir_program):
    """This function returns the total cost (in terms of execution cost) of a single solution given by the
     D-Wave machine"""
    solution_cost = 0
    try:
        for test_case, var_x in solutions_dictionary_list[solution_number].items():
            solution_cost += test_cases_costs[sir_program][test_case] * var_x
    except:
        #index out of range
        return 0
    return solution_cost

def get_total_fault_coverage(solution_numer,solutions_dictionary_list,sir_program):
    """This function returns the percentage of faults covered by a single solution given by the D-Wave
     machine"""
    total_faults = 0
    total_solution_faults = 0
    for fault_value in faults_dictionary[sir_program]:
        total_faults += fault_value
    for test_case, var_x in solutions_dictionary_list[solution_numer].items():
        if var_x:
            total_solution_faults += faults_dictionary[sir_program][test_case]
    return total_solution_faults/total_faults
```

Figura 3.8: Calcolo del costo di esecuzione e della percentuale di fallimenti rilevati per ciascuna suite di test

```python
def bootstrap_confidence_interval(data, num_samples, alpha=0.95):
    """This function determines the statistical range within we would expect the mean value of execution
     times to fall; it relies on the bootstrapping strategy, which allows the calculation of the confidence
     interval by repeatedly sampling (with replacement) from the existing data to obtain an estimate of the
     confidence interval."""
    sample_means = []
    for _ in range(num_samples):
        bootstrap_sample = [random.choice(data) for _ in range(len(data))]
        sample_mean = np.mean(bootstrap_sample)
        sample_means.append(sample_mean)

    lower_percentile = (1 - alpha) / 2 * 100
    upper_percentile = (alpha + (1 - alpha) / 2) * 100
    lower_bound = np.percentile(sample_means, lower_percentile)
    upper_bound = np.percentile(sample_means, upper_percentile)

    return lower_bound, upper_bound
```

Figura 3.9: Calcolo dell'intervallo di confidenza tramite bootstrapping

CAPITOLO 4

Analisi dei Dati e Risultati

In questo capitolo si descrivono nel dettaglio i risultati ottenuti durante le sperimentazioni, traendo le conseguenti conclusioni.

4.1 Efficacia delle Soluzioni

4.1.1 La Qualità delle Soluzioni del Quantum Annealing

La Tabella 4.1 riporta i valori medi della metrica di ipervolume (I_{CE}), calcolata come descritto nella Sezione 3.3.4, relativi agli insiemi di soluzioni costruiti dai quattro algoritmi che si stanno confrontando: vNSGA-II, DIV-GA, greedy e quantum annealing. I valori che sono riportati corrispondono alla media dei valori di I_{CE} ottenuti in seguito a 30 esecuzioni indipendenti di ciascun algoritmo.

Per 3 programmi su 4, i valori della metrica di ipervolume ottenuti dall'algoritmo di quantum annealing sono maggiori rispetto a quasi tutti quelli ottenuti dagli altri algoritmi. Solo nel caso del programma "gzip" l'algoritmo di quantum annealing riesce ad ottenere un valore della metrica minore rispetto agli altri, eguagliando l'efficacia dall'algoritmo DIV-GA.

Dunque, da questi risultati si evince che le suite di test costruite come soluzioni dal quantum annealing sono in media meno efficaci rispetto alle suite costruite dagli

Tabella 4.1: Valori Medi dell'Ipervolume

Programma	Quantum Annealing	DIV-GA	vNSGA-II	Greedy
flex	0.73	**0.05**	0.20	0.15
grep	0.51	**0.27**	0.28	0.51
gzip	**0.51**	**0.51**	0.54	0.52
sed	0.52	**0.10**	**0.10**	0.20

altri algoritmi. Ossia, le suite di test costruite dagli algoritmi classici sono in grado di rilevare più fallimenti con costi di esecuzione minori, rispetto alle suite ottenute dall'esecuzione del quantum annealing.

In seguito ai confronti eseguiti, appare evidente che DIV-GA, rispetto a tutti gli altri algoritmi analizzati, riesca a trovare delle soluzioni che siano migliori dal punto di vista dell'efficacia. Non c'è invece un chiaro vincitore tra vNSGA-II e la strategia greedy, in quanto in due programmi vNSGA-II registra valori migliori della strategia greedy, mentre negli altri due il confronto presenta il risultato opposto.

4.1.2 Il Problema Della Strategia di Annealing

Sulla base delle analisi descritte nella Sezione precedente, si è concluso che le soluzioni prodotte dall'algoritmo di quantum annealing non raggiungono gli stessi livelli qualitativi di quelle prodotte dagli altri algoritmi presi in esame.

Resta dunque da capire se il problema del quantum annealing si debba ricercare nella sua natura quantistica o nella natura stessa delle strategie di annealing. A tal fine, si rende necessario effettuare, sempre nei termini della metrica dell'ipervolume, un confronto tra il quantum annealing e la sua controparte classica, ossia il "*Simulated Annealing*".

Il Simulated Annealing

Il simulated annealing è una tecnica di ottimizzazione meta-euristica, introdotta da Kirkpatrick et al.[58] per risolvere il problema del commesso viaggiatore.

Il nome dell'algoritmo non è casuale, esso si ispira infatti all'*annealing*, ossia un noto processo usato in metallurgia. L'annealing prevede che un metallo venga portato rapidamente ad una temperatura estremamente elevata, per poi essere fatto gradualmente raffreddare. Quando il metallo si trova a livelli di temperatura alti, i suoi atomi si muovono molto rapidamente, quindi, con la riduzione della temperatura, la loro energia cinetica diminuisce; al termine del processo di annealing, gli atomi si ritroveranno in uno stato più ordinato di quello di partenza, facendo sì che il metallo sia più duttile e facile da lavorare.

Il successo del simulated annealing è dovuto alla sua notevole capacità di evitare minimi locali, riuscendo quindi a convergere più facilmente verso un ottimo globale; inoltre, è piuttosto semplice da implementare.

L'ALGORITMO Come si diceva, l'algoritmo si avvia a partire da una soluzione candidata iniziale, ossia una stringa di bit che indica quale caso di test è considerato come incluso nella suite di test "attuale", che viene poi man mano migliorata tramite delle leggere perturbazioni casuali, le quali sono accettate solo con una certa probabilità. Una perturbazione potrebbe generare una soluzione peggiore rispetto a quella precedente, per questo la probabilità di accettare una soluzione del genere inizialmente è alta ma diminuisce gradualmente con l'avanzare del tempo (ossia con l'aumentare delle iterazioni dell'algoritmo).

LA FUNZIONE OBIETTIVO Prima di passare all'implementazione dell'algoritmo, è necessario definire un obiettivo da raggiungere, ossia la funzione che si vuole minimizzare (o massimizzare); per il lavoro di tesi, la funzione obiettivo, detta in questo caso "*funzione energetica*", corrisponde all'hamiltoniana definita nel Capitolo precedente.

LA FUNZIONE DI PERTURBAZIONE La funzione di perturbazione corrisponde alla strategia utilizzata per la generazione di una nuova soluzione candidata a partire da una soluzione candidata "attuale". Una tale perturbazione dovrebbe consentire di generare soluzioni che siano vicine ma non troppo simili alla soluzione attuale. Ciò è dovuto al fatto che generare soluzioni troppo simili non permetterebbe una ricerca esaustiva, soprattutto quando il numero di iterazioni dell'algoritmo non è molto elevato.

La strategia di perturbazione di una soluzione è estremamente legata alle carat-

teristiche del problema che si sta cercando di risolvere; non esiste una strategia di perturbazione adeguata ad ogni situazione, essa viene scelta empiricamente sulla base di ripetute sperimentazioni. Nel caso di questo lavoro di tesi, a partire dalla stringa di bit che rappresenta la soluzione corrente, di 1/3 di tali bit viene eseguito il flip.

IL CRITERIO DI ACCETTAZIONE Come si diceva, nel caso in cui la perturbazione produca una nuova soluzione che sia peggiore di quella precedente, essa viene accettata solo con una certa probabilità. Il criterio di accettazione definisce proprio le modalità con cui una soluzione viene rifiutata o accettata.

I fattori da cui dipendono le valutazioni del criterio di accettazione sono diversi, uno fra questi è il coefficiente di "*variazione di energia*" ΔE, il quale rappresenta la differenza di energia tra la nuova soluzione e quella precedente.

Il secondo fattore è quello di temperatura T, il quale dovrà partire da un valore elevato per poi scendere fino ad una soglia prestabilita. Si tratta di valori che vengono stabiliti in seguito a diverse sperimentazioni, in quanto dipendenti dai singoli casi. In questo lavoro di tesi, si è deciso di impostare il valore iniziale di T a 80000 e quello soglia a 10; ciò rende possibile l'esecuzione di un numero di iterazioni esaustivo per la ricerca in atto.

Il terzo fattore da cui dipende il criterio di accettazione è il fattore α, detto di "*cooldown*" e compreso tra 0 ed 1. Si tratta di un fattore che determina in che modo la temperatura di un sistema cambia nel tempo. Infatti, partendo da una temperatura elevata e applicandovi il fattore α, con l'avanzare delle iterazioni il valore della temperatura diminuirà gradualmente. Ciò significa che man mano l'algoritmo diventerà sempre più selettivo, diminuendo le chance di accettare nuove soluzioni peggiori di quelle correnti. In letteratura si tende a dare un valore di α compreso tra 0.8 e 0.99, ma anche qui si tratta di valutazioni troppo dipendenti dai singoli problemi. Nel caso di questo lavoro di tesi, in seguito a diverse sperimentazioni, è stato scelto un valore di α pari a 0.87.

L'ultimo fattore da considerare è t, il quale rappresenta semplicemente il numero di iterazioni svolte fino ad un certo momento. Tale parametro viene tracciato in quanto, con l'avanzare delle iterazioni, l'effetto del fattore α va gradualmente aumentato.

Dunque, il criterio di accettazione funziona nel modo seguente: se la nuova soluzione candidata ha un valore energetico minore rispetto a quella precedente, viene sempre accettata (diventando la soluzione candidata attuale); altrimenti, se la nuova soluzione candidata ha un valore energetico superiore rispetto alla soluzone candidata precedente, la nuova è accettata con una probabilità pari a: $\exp(\frac{-\Delta E}{T \cdot \alpha^t})$.

Implementazione del Simulated Annealing

Nel presente lavoro di tesi il simulated annealing è stato implementato in Python e si riportano di seguito le procedure che lo compongono.

La prima procedura valuta il valore energetico di una nuova soluzione candidata generata. Tale soluzione sarà la prima inizialmente, successivamente corrisponderà a quelle ottenute tramite il processo di perturbazione. La procedura completa è mostrata nella Figura 4.1.

```
def compute_solution_energy(solution,sir_program):
    """This function computes the energy of a solution"""
    solution_cost = 0
    for test_case, test_case_cost in enumerate(test_cases_costs[sir_program]):
        solution_cost += (test_case_cost * solution[test_case])
    solution_cost *= alpha

    detected_faults = 0
    for test_case, fault_value in enumerate(faults_dictionary[sir_program]):
        if sir_program == "grep" and test_case == 806:
            break
        detected_faults += (fault_value * solution[test_case])
    detected_faults *= (1 - alpha)

    statement_coverage = 0
    for k in coverage[sir_program].keys():
        test_cases = coverage[sir_program][k]
        for i in test_cases:
            for j in test_cases:
                if i < j:
                    statement_coverage += ((solution[i] * (-1 * penalties_dictionary[sir_program]))
                    + (2 * penalties_dictionary[sir_program] * solution[i] * solution[j]))

    return solution_cost - detected_faults + statement_coverage
```

Figura 4.1: Risoluzione dell'hamiltoniana sulla base della soluzione candidata

La seconda procedura implementa la strategia di perturbazione. Come si è già detto, viene eseguito il flip di 1/3 dei bit della soluzione candidata corrente. La

procedura è mostrata dalla Figura 4.2.

```
def generate_new_solution(sir_program,solution):
    """This function generate a new candidate solution near to the actual solution by flipping 1/3 of the
     bits of the solution string representing a test suite"""
    for _ in range(sir_programs_tests_number[sir_program]//3):
        random_test_case = random.randint(0,sir_programs_tests_number[sir_program]-1)
        if solution[random_test_case] == 1:
            solution[random_test_case] = 0
        else:
            solution[random_test_case] = 1

    return solution
```

Figura 4.2: Generazione di una nuova soluzione tramite perturbazione

Prima di implementare l'algoritmo di simulated annealing vero e proprio, manca ancora una procedura che implementi la strategia di accettazione di una nuova soluzione candidata. Nella Figura 4.3 è mostrata tale procedura.

L'algoritmo di simulated annealing è implementato dalla procedura mostrata nella Figura 4.4.

Una volta implementate tutte le procedure necessarie all'esecuzione corretta dell'algoritmo di simulated annealing, non resta che eseguire ripetute sperimentazioni di tale strategia e, per poter eseguire gli adeguati confronti con la sua controparte quantistica, tener traccia dei valori di ipervolume finali, suddivisi per ciascuno dei 4 programmi in esame. L'implementazione di quest'ultima fase è mostrata nella Figura 4.5.

```python
def is_soluction_accepted(temperature,delta_energy):
    """This function determines whether a new solution is accepted or rejected"""
    if delta_energy < 0:
        return True
    else:
        r = random.random()
        if r < math.exp((-delta_energy)/temperature):
            return True
        else:
            return False
```

Figura 4.3: Strategia di accettazione di una nuova soluzione candidata

La Qualità delle Soluzioni del Simulated Annealing

La Tabella 4.2 riporta i valori medi della metrica di ipervolume (I_{CE}) relativi agli insiemi di soluzioni ottenuti in seguito all'esecuzione di sperimentazioni dell'algoritmo di simulated annealing.

Tabella 4.2: Valori Medi dell'Ipervolume del Simulated Annealing

Programma	Simulated Annealing
flex	0.80
grep	0.54
gzip	**0.50**
sed	0.48

Esattamente come nel caso del quantum annealing, anche per il simulated annealing i valori della metrica dell'ipervolume, per (gli stessi) 3 programmi su 4, sono

```
def simulated_annealing(sir_program,annealing_alpha):
    """This function implements the simulated annealing algorithm for the test case selection problem"""
    iteration = 1
    temperature = 80000
    solution = {key: 1 for key in range(sir_programs_tests_number[sir_program])}
    energy = compute_solution_energy(solution,sir_program)

    while temperature > 10:
        new_solution = generate_new_solution(sir_program,solution)
        new_energy = compute_solution_energy(new_solution,sir_program)
        energy_delta = energy - new_energy
        if is_soluction_accepted(temperature,energy_delta):
            solution = new_solution
            energy = new_energy
        temperature *= (annealing_alpha ** iteration)
        iteration += 1

    return solution
```

Figura 4.4: L'algoritmo di simulated annealing

maggiori rispetto a quelli ottenuti dagli altri algoritmi. Nel caso di "gzip", come per il quantum annealing, si è ottenuto un valore della metrica migliore rispetto agli altri ma comunque simile a quello ottenuto dalla sua controparte quantistica.

Questi risultati mostrano che in effetti il problema dietro il peggioramento della qualità delle soluzioni ottenute dal quantum annealing potrebbe non essere dovuto alla sua natura quantistica. In effetti, il problema sembra da doversi ricercare nelle caratteristiche della strategia di annealing stessa, che non pare intrinsecamente adatta al problema di test case selection.

Conclusioni Riguardo la Strategia di Annealing

Sia il quantum annealing che il simulated annealing sono strategie estremamente potenti e flessibili. Nonostante ciò, hanno entrambe mostrato di non riuscire ad ottenere dei risultati che abbiano dei livelli di qualità validi quanto quelli ottenuti dagli algoritmi MOGA e greedy per il probma di test case selection.

Si deve concludere che dunque la strategia di annealing, per ciò che riguarda

```
for sir_program in sir_programs:
    for _ in range(tot_run):
        solutions_dictionary[sir_program].append(simulated_annealing(sir_program,0.87))

    I_H = 0

    for solution in range(len(solutions_dictionary[sir_program])):
        if solution == 0:
            solution_cost = get_solution_cost(solution,solutions_dictionary[sir_program],sir_program)
            I_H += solution_cost
            total_solution_faults = 0
            for test_case, var_x in solutions_dictionary[sir_program][solution].items():
                if var_x:
                    total_solution_faults += faults_dictionary[sir_program][test_case]
        else:
            solution_cost = get_solution_cost(solution,solutions_dictionary[sir_program],sir_program)
            I_H += (get_solution_cost(solution+1,solutions_dictionary[sir_program],sir_program)
             -solution_cost)*(1-get_total_fault_coverage(solution,solutions_dictionary[sir_program],
             sir_program))
            total_solution_faults = 0
            for test_case, var_x in solutions_dictionary[sir_program][solution].items():
                if var_x:
                    total_solution_faults += faults_dictionary[sir_program][test_case]
    if I_H<0: I_H *= -1
    I_CE_simulated_annealing_dictionary[sir_program] = I_H/get_solution_cost(len
     (solutions_dictionary[sir_program])-1,solutions_dictionary[sir_program],sir_program)

    #Show the results
    print("INFORMATION FOR " + sir_program + ":")
    print("I_CE: " + str(I_CE_simulated_annealing_dictionary[sir_program]))
```

Figura 4.5: Esecuzione di sperimentazioni del simulated annealing per cianscun programma in esame e calcolo della metrica dell'ipervolume

l'aspetto dell'efficacia delle soluzioni, non sia la strategia ottimale per il problema di selezione dei casi di test. In effetti, non tutti i problemi sono adatti ad essere risolti con il quantum o il simulated annealing e purtroppo è difficile riconoscerli a priori.

Una motivazione che potrebbe spiegare i risultati registrati, consiste nell'eccessiva complessità dello spazio di ricerca del problema così come è stato formulato (Capitolo 3.1.2). In effetti, le interconnessioni che sono presenti tra i diversi casi di test (variabili del problema) di una test suite riguardo il vincolo di copertura degli statement sono molto fitte; questo comporta sia un problema per il simulated annealing, che quindi richiede una ricerca intensiva e dunque rivela una difficoltà a convergere verso una soluzione ottimale, sia un problema per il quantum annealing, in quanto i problemi ad "alta precisione" soffrono molto la trasformazione in modelli QUBO a causa del rumore ancora difficilmente gestito dalle macchine quantistiche.

C'è da dire che i problemi decisionali soffrono anche dell'incertezza del fatto che non esista una soluzione fattibile, oppure che la soluzione semplicemente non

sia stata osservata dall'annealer. Inoltre, spesso i problemi che sembrano mostrare buoni risultati con il quantum annealing presentano vincoli in cui il numero e la cui connettività dei qubit richiesti per il problema non aumentano fortemente con le dimensioni del problema, al contrario di come avviene per questo lavoro di tesi.

4.2 Le Notevoli Performance del Quantum Annealing

Un importante limite delle tecniche MOGA e greedy per la risoluzione del problema di selezione dei casi di test, consiste negli eccessivi tempi di esecuzione. Talvolta, la ricerca della soluzione può durare anche diversi minuti, soprattutto nel caso delle strategie MOGA, come nel caso di DIV-GA[33]. Infatti, sebbene i risultati ottenuti mostrino che DIV-GA sia superiore in termini di qualità delle soluzioni costruite rispetto a vNSGA-II ed alle tecniche greedy, è anche da considerare che, per raggiungere gli obiettivi di diversità che propone, aggiunge la computazione di SVD, che è di per sé abbastanza dispendiosa, nel ciclo principale dell'algoritmo genetico.

Al contrario delle soluzioni classiche per la risoluzione di problemi di ottimizzazione combinatoriali, gli algoritmi di ottimizzazione adiabatici, grazie alla loro natura quantistica che introduce il concetto di sovrapposizione, riescono a raggiungere risultati simili al processamento su thread multipli con un tempo lineare o addirittura costante (a seconda dell'algoritmo implementato).

Sono state eseguite delle verifiche per accertarsi che le previsioni circa la dominanza in termini di performance del quantum annealing sugli altri algoritmi esaminati fosse reale. In effetti, i risultati ottenuti confermano tali aspettative, mostrando dei valori di performance del quantum annealing estremamente sorprendenti.

Non si può dire lo stesso del simulated annealing che, come previsto, presenta delle performance molto basse per ciò che riguarda i tempi di esecuzione. Infatti, il simulated annealing è un tipo di algoritmo le cui performance sono molto influenzate dalla grandezza del problema da risolvere, che nel caso di questo lavoro di tesi è di grandi dimensioni. Inoltre, l'algoritmo per funzionare correttamente necessita del tuning di molti parametri e risulta quindi difficile trovarne la combinazione di valori che impatti al meglio dal punto di vista delle performance.

Nella Tabella 4.3 vengono mostrati i tempi medi di esecuzione di ciascun algoritmo esaminato e, come previsto, il quantum annealing esegue in un tempo praticamente costante. Si ricorda che le esecuzioni sono state effettuate su macchine diverse, infatti gli algoritmi MOGA e greedy (i cui tempi sono stati presi dal lavoro di Panichella et al.[33]) sono stati eseguiti su una macchina dotata di un processore Intel Core i7 da 2.40GHz e di una RAM di 8GB; al contrario, il quantum annealing è stato eseguito sul sampler D-Wave Hybrid Binary Quadratic Model (Version2). Infine, il simulated annealing è stato testato su una macchina dotata di un processore Intel Core i5 da 2.30GHz e di una RAM da 8GB.

Tabella 4.3: Tempi di esecuzione medi per gli algoritmi

Programma	Quantum Annealing	DIV-GA	vNSGA-II	Greedy	Simulated Annealing
flex	2.9s	2min 51sec	5min 40sec	1min 7sec	15min 31sec
grep	2.9s	3min 45sec	5min 57sec	2min 4sec	15min 28sec
gzip	2.9s	33s	1min 22sec	4s	25s
sed	2.9s	1min 33sec	2min 25sec	14s	4min 10sec

Per avere maggiore chiarezza e maggiore sicurezza riguardo ai risultati ottenuti dal quantum annealing sui tempi di esecuzione, è stato anche calcolato un intervallo di confidenza dei tempi di esecuzione della macchina quantistica tramite il metodo del bootstrapping. I risultati di questa analisi, che confermano le osservazioni precedenti, sono mostrati nella Figura 4.6.

```
Intervallo di confidenza per flex: (2.9937, 2.9966)
Intervallo di confidenza per grep: (2.9939, 2.9965)
Intervallo di confidenza per gzip: (2.9908, 2.9937)
Intervallo di confidenza per sed: (2.9890, 2.9924)
```

Figura 4.6: Intervalli di confidenza, divisi per programma, del tempo di esecuzione della macchina quantistica

CAPITOLO 5

Threats to Validity

In questo capitolo si discutono tutti gli aspetti che potrebbero minacciare la validità dello studio effettuato. Le minacce da anlizzare possono essere suddivise in minacce alla validità: di *costrutto*, *interna*, *esterna* e della *conclusione*.

5.1 Validità di Costrutto

Le minacce alla validità del costrutto riguardano le relazioni tra la teoria e le osservazioni. Nel caso di questo lavoro di tesi, la principale minaccia in questo senso riguarda la correttezza delle misure utilizzate come criteri per la selezione dei test: copertura, storico dei fallimenti e costo di esecuzione.

Per riuscire a limitare questo pericolo le informazioni di copertura del codice sono state collezionate utilizzando degli strumenti di profilazione e di compilazione open-source (ossia GNU gcc e gcov). Per quel che riguarda le informazioni di costo di esecuzione dei casi di test, le misurazioni hanno riguardato il conteggio del numero di blocchi elementari di codice che ci si aspetta che i casi di test eseguano; infine, lo storico dei fallimenti rilevati dai casi di test sono stati estratti dal dataset della SIR.

5.2 Validità Interna

Le minacce alla validità interna del progetto riguardano tutti i fattori che avrebbero potuto influenzare i risultati del lavoro di tesi e che sarebbero dovuti essere maggiormente considerati.

Uno di questi fattori è senza dubbio la natura casuale degli algoritmi quantistici. Proprio per questo motivo, le sperimentazioni sono state ripetute 30 volte per ciascun programma sotto esame, per poi considerare un intervallo di confidenza riguardo al tempo di esecuzione utilizzato poi per il confronto con le altre tecniche di risoluzione del problema di selezione dei casi di test.

Anche il *tuning* del parametro di penalità P è un fattore che potrebbe minare la validità interna di questo lavoro. Per questo, nella scelta del valore da affidare a P, si è utilizzato un metodo diffuso in letteratura[54].

Altri parametri il cui tuning potrebbe minare la validità interna di questo lavoro corrispondono a quelli utilizzati per l'implementazione del simulated annealing. Ebbene, il parametro α è stato scelto, oltre che sulla base di ripetuti test, utilizzando un metodo già diffuso in letteratura. Per quanto riguarda gli altri parametri, sono stati tutti validati in seguito a sperimentazoni ripetute.

5.3 Validità Esterna

Le minacce alla validità esterna corrispondono a problemi relativi alla generalizzazione dei risultati ottenuti e riguardano l'insieme dei programmi utilizzati per la sperimentazione.

Per mitigare i risultati ottenuti in questo lavoro di tesi, sono stati considerati 4 programmi estratti dalla SIR utilizzati anche in precedenti lavori sul regression testing[9][7][27][28][55]. Inoltre, potrebbero esserci altre soluzioni algoritmiche che non sono state considerate nei confronti con la soluzione proposta in questo lavoro; ciò in quanto non sono conosciuti, oltre a quelli già presentati, algoritmi particolarmente efficaci per risolvere il problema di selezione dei casi di test[28]. Questo lavoro di tesi ha paragonato le performance del quantum annealing rispetto alle soluzioni più utilizzate al momento come appunto vNSGA-II.

5.4 Validità della Conclusione

Per mitigare le minacce alle conclusioni finali, le deduzioni, fatte in seguito ai risultati ottenuti durante le sperimentazioni, sono supportate dal calcolo della metrica dell'ipervolume e dell'intervallo di confidenza dei tempi di esecuzione della macchina quantistica.

CAPITOLO 6

Conclusioni

Questo lavoro di tesi ha sviluppato ed analizzato un approccio quantistico al problema di selezione dei casi di test di regressione, con lo scopo di proporre un'alternativa valida alle soluzioni algoritmiche MOGA e greedy attualmente utilizzate. Nel dettaglio, in seguito a valutazioni relative al numero di qubit necessari per la codifica del problema da risolvere, l'algoritmo proposto in questo lavoro è quello di quantum annealing. Si tratta di un approccio che mira a trovare il minimo globale della funzione che rappresenta gli obiettivi ed i vincoli del problema di ottimizzazione.

In seguito ad una serie di esperimenti condotti su 4 programmi differenti estratti dalla SIR, si è potuto concludere che la strategia quantistica, nonostante domini senza alcun dubbio le tecniche classiche in termini di performance, costruisca soluzioni che in media sono meno efficaci di quelle ottenute dalle classiche. Sono state necessarie dunque ulteriori analisi per comprendere se la causa dell'inferiore efficacia delle soluzioni ottenute dalla strategia quantistica rispetto a quelle ottenute dalle altre fosse dovuta alla natura quantistica del quantum annealing oppure alla natura del processo stesso di annealing.

A tal fine, è stata implementata la controparte classica del quantum annealing,

ossia il simulated annealing. Quest'ultimo ha però presentato dei risultati paragonabili a quelli del quantum annealing, allontanando dunque l'attenzione dalla natura quantistica del quantum annealing.

La causa che con maggior probabilità rispetto alle altre sottende a queste osservazioni è da ricercarsi nell'eccessiva complessità dello spazio di ricerca, che presenta interconnessioni troppo dense tra le variabili del problema.

In tal senso, volendo continuare a percorrere la strada del quantum annealing, una possibile soluzione consiste nel rilassamento del problema. Per esempio, si consideri il vincolo di copertura degli statement dei programmi da testare. Ebbene, tale vincolo considera la copertura degli statement al livello dei singoli blocchi eseguiti dai casi di test; lo si potrebbe invece convertire in una terza funzione obiettivo da minimizzare che consideri la copertura degli statement globale, ossia solamente al livello dell'intero programma da testare. In questo modo si otterrebbe un problema lineare e sicuramente molto più adatto ad un processo di annealing. Oppure ancora, volendo sfruttare la capacità di calcolo del quantum annealing e dunque mantenendo un problema quadratico, si potrebbe pensare di rilassare il problema ponendo una soglia al numero massimo di casi di test da considerare come esecutori di un preciso statement, diminuendo dunque la quantità di interconnessioni tra le variabili (ossia i casi di test) del problema.

In effetti, la soluzione di quantum annealing così come è stata proposta in questo lavoro di tesi, apre una questione inerente al trade-off tra efficienza ed efficacia. Infatti, se da un lato il quantum annealing riesce a risolvere il problema di test case selection in un tempo praticamente costante (indifferentemente dalla grandezza del programma da testare), dall'altro costruisce soluzioni di efficacia inferiore rispetto a quelle costruite dalle tecniche MOGA e greedy. Ovvio è che la prevalenza dell'efficienza sull'efficacia (e viceversa) dipende dalla natura del problema che si affronta. In particolare, nel contesto della selezione dei casi di test per la fase di regressione, la propensione per una tecnica piuttosto che per un'altra dipende molto dalle esigenze e dai vincoli del progetto a cui si sta lavorando.

Il quantum annealing, o in generale soluzioni più efficienti, è da preferirsi ad esempio in situazioni in cui i tempi sono particolarmente limitati; per esempio, volen-

do inserire il regression testing in una pipeline di continuous integration, risulterebbe scomodo eseguire una fase di selezione dei casi di test della durata di diversi minuti. Altri casi in cui il quantum annealing risulterebbe vantaggioso corrispondono a scenari in cui si devono gestire in maniera scalabile ed efficace suite di test molto grandi, oppure in presenza di risorse computazionali esigue che andrebbero gestite in maniera ottimale da algoritmi per l'appunto più prestanti che precisi.

Vi sono però anche molti contesti in cui risulterebbe preferibile se non neceserraria l'efficacia piuttosto che l'efficienza. Ad esempio, nel caso in cui si fosse interessati a garantire un'alta qualità delle suite di regressione, ovviamente la scelta non potrebbe cadere sul quantum annealing. Oppure ancora, esistono molti contesti in cui per garantire sicurezza e robustezza dei sistemi software sono necessarie estrema precisione e correttezza dei casi di test; ciò quindi farebbe propendere per il valore dell'efficacia, rispetto a quello dell'efficienza. Infine, quando la differenza dei tempi di esecuzione tra algoritmi efficaci ed efficienti non è così estrema, il che accade in genere per problemi di piccole dimensioni, allora anche qui potrebbe essere più saggio puntare su tecniche efficaci.

In alternativa al quantum annealing (ma rimanendo sempre nell'ambito quantistico), una possibile strada da seguire è quella che considera strategie ibride classico-quantistiche, alcune delle quali sono state già analizzate dalla ricerca. Si tratta di soluzioni che cercano di ottenere i vantaggi da entrambi i tipi di algoritmi, massimizzando efficacia e performance. In effetti, si potrebbe considerare la possibilità di implementare algoritmi genetici ibridi, i quali sfruttano l'approccio quantistico in genere nelle fasi di crossover e mutazione, ottenendo quindi un notevole aumento delle capacità di diversificazione della popolazione e dunque dell'efficacia della ricerca; tutto ciò con dei tempi di esecuzione praticamente lineari, se non costanti, tipici degli algoritmi quantistici.

Il processo che ha portato alla traduzione del problema di selezione dei casi di test dal caso classico al caso quantistico, ha sottolineato inoltre la possibilità di considerare anche molti altri problemi estremamente attuali nell'ambito dell'ingegneria del software adatti ad una tale conversione. Uno fra tutti, rimanendo sempre nell'ambito

del testing, potrebbe essere il problema di prioritizzazione delle suite di test. Oppure si potrebbero considerare problemi come la pianificazione delle risorse, l'assegnazione dei compiti, la pianificazione dei test, dei rilasci o delle manutenzioni. Insomma, il campo dell'ingegneria del software presenta moltissimi problemi le cui soluzioni potrebbero essere migliorate tramite una conversione da un approccio classico ad un approccio quantistico. Proprio per questo, molta energia sarà spesa dalla ricerca per sviluppi futuri in merito all'applicazione di algoritmi di ottimizzazione quantistica nell'ambito dell'ingegneria del software.

Bibliografia

[1] J. D. Musa, "Developing more reliable software faster e cheaper," *IEEE International Conference on Engineering of Complex Computer Systems*, 2004. (Citato a pagina 7)

[2] M. Črepinšek e S.-H. Liu e M. Mernik, "Exploration e exploitation in evolutionary algorithms: A survey." *ACM Comput. Surv.,vol. 45, no. 3, pp. 35:1–35:33*, 2003. (Citato alle pagine 8 e 18)

[3] V. G. e T. Varma, "A replicated survey of software testing practices in the canadian province of alberta: What has changed from 2004 to 2009?" *J. Syst. Softw., vol. 83, no. 11, pp. 2251–2262*, 2010. (Citato a pagina 9)

[4] G. R. e M. J. Harrold, "Empirical studies of a safe regression test selection technique," *IEEE Trans. Softw. Eng., vol. 24, no. 6, pp. 401–419*, 1998. (Citato a pagina 9)

[5] P. Runeson, "A survey of unit testing practices," *IEEE Softw., vol. 23, no. 4, pp. 22–29*, 2006. (Citato a pagina 9)

[6] S. Y. e M. Harman, "Regression testing minimization, selection e prioritization: A survey," *Softw. Test. Verif. Rel., vol. 22, no. 2, pp. 67–120*, 2012. (Citato alle pagine 9 e 12)

[7] G. R. e M. J. Harrold e J. von Ronne e C. Hong, "Empirical studies of test-suite reduction," *J. Softw. Testing, Verification, Rel., vol. 12, pp. 219–249*, 2002. (Citato alle pagine 10 e 66)

[8] M. J. H. e R. Gupta e M. L. Soffa, "A methodology for controlling the size of a test suite," *ACM Trans. Softw. Eng. Methodol., vol. 2, pp. 270–285*, 1993. (Citato a pagina 10)

[9] T. Y. C. e M. F. Lau, "Dividing strategies for the optimization of a test suite," *Inf. Process. Lett., vol. 60, no. 3, pp. 135–141*, 1996. (Citato alle pagine 10 e 66)

[10] A. J. O. e J. Pan e J. M. Voas, "Procedures for reducing the size of coverage-based test sets," *Proc. 12th Int. Conf. Testing Comput. Softw, pp. 111-123*, 1995. (Citato a pagina 10)

[11] M. M. e A. Bertolino, "Using spanning sets for coverage testing," *IEEE Trans. Softw. Eng., vol. 29, no. 11, pp. 974–984*, 2003. (Citato a pagina 10)

[12] S. M. e A. M. Memon, "Call stack coverage for test suite reduction," *Proc. IEEE Int. Conf. Softw. Maintenance, pp. 539-548*, 2005. (Citato a pagina 10)

[13] J. B. e E. Melachrinoudis e e D. Kaeli, "Bi-criteria models for all-uses test suite reduction," *Proc. 26th Int. Conf. Softw. Eng., pp. 106–115*, 2004. (Citato alle pagine 10 e 41)

[14] G. R. e R. Untch e C. Chu e M. Harrold, "Prioritizing test cases for regression testing," *IEEE Trans. Softw. Eng., vol. 27, no. 10, pp. 929–948*, 2001. (Citato a pagina 11)

[15] S. E. e A. Malishevsky e G. Rothermel, "Incorporating varying test costs e fault severities into test case prioritization," *Proc. 23rd Int. Conf. Softw. Eng., pp. 329–338*, 2001. (Citato a pagina 11)

[16] S. E. e A. G. Malishevsky e G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE Trans. Softw. Eng., vol. 28, no. 2, pp. 159–182*, 2002. (Citato a pagina 11)

[17] R. C. B. e C. J. Colbourn e M. B. Cohen, "A framework of greedy methods for constructing interaction test suites," *Proc. Int. Conf. Softw. Eng., pp. 146–155*, 2005. (Citato a pagina 11)

[18] M. C. e M. Dwyer e J. Shi, "Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach," *IEEE Trans. Softw. Eng., vol. 34, no. 5, pp. 633–650*, 2008. (Citato a pagina 11)

[19] H. S. e L. Williams e J. Osborne, "System test case prioritization of new e regression test cases," *Proc. Int. Symp. Empir. Softw. Eng., pp. 60–63*, 2005. (Citato a pagina 11)

[20] A. G. M. e J. R. Ruthruff e G. Rothermel e S. Elbaum, "Cost-cognizant test case prioritization," *Dept. Comput. Sci. Eng., Univ. Nebraska-Lincoln, Lincoln, Nebraska, USA, Tech. Rep. TR-UNL-CSE-2006-0004*, 2006. (Citato a pagina 11)

[21] H. D. e G. Rothermel e A. Kinneer, "Empirical studies of test case prioritization in a junit testing environment," *15th Int. Symp. Softw. Reliability Eng., pp. 113–124*, 2004. (Citato a pagina 11)

[22] S. E. e A. G. Malishevsky e G. Rothermel, "Prioritizing test cases for regression testing," *Proc. ACM SIGSOFT Int. Symp. Softw. Testing Anal., pp. 102–112*, 2000. (Citato a pagina 11)

[23] G. R. e M. J. Harrold, "Analyzing regression test selection techniques," *" IEEE Trans. Softw. Eng., vol. 22, no. 8, pp. 529–551*, 1996. (Citato a pagina 12)

[24] S. S. Y. e Z. Kishimoto, "A method for revalidating modified programs in the maintenance phase," *Proc. Int. Comput. Softw. Appl. Conf.*, 1987. (Citato a pagina 12)

[25] G. R. e M. Harrold, "A safe, efficient algorithm for regression test selection," *Proc. Conf. Softw. Maintenance, pp. 358–367*, 1993. (Citato a pagina 12)

[26] S. B. e S. Horwitz, "Incremental program testing using program dependence graphs," *Proc. 20th ACM SIGPLAN-SIGACT Symp. Principles Program. Language, pp. 384–396*, 1993. (Citato a pagina 12)

[27] S. Y. e M. Harman, "Pareto efficient multi-objective test case selection," *Proc. ACM/SIGSOFT Int. Symp. Softw. Testing Anal. pp. 140–150*, 2007. (Citato alle pagine 12, 16, 17, 42 e 66)

[28] ——, "Using hybrid algorithm for pareto efficient multi-objective test suite minimisation," *J. Syst. Softw., vol. 83, no. 4, pp. 689–701*, 2010. (Citato alle pagine 12, 16, 17, 41, 42 e 66)

[29] M. Harman, "Making the case for morto: Multi objective regression test optimization," *Proc. ICST Workshops, pp. 111–114*, 2011. (Citato a pagina 12)

[30] A. S. e J. Thiagarajan, "Effectively prioritizing tests in development environment," *Proc. ACM SIGSOFT Int. Symp. Softw. Testing Anal., pp. 97–106*, 2002. (Citato a pagina 12)

[31] K. D. e S. Agrawal e A. Pratab e T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii," *Proceedings of the Parallel Problem Solving from Nature VI Conference, pages 849–858, Paris, France.*, 2000. (Citato a pagina 16)

[32] K. D. e A. Pratap e S. Agarwal e T. Meyarivan, "A fast elitist multi-objective genetic algorithm: Nsga-ii," *IEEE Trans. Evol. Comput., vol. 6, no. 2, pp. 182–197*, 2002. (Citato alle pagine 16 e 18)

[33] A. P. e R. Oliveto e M. Di Penta e A. De Lucia, "Improving multiobjective test case selection by injecting diversity in genetic algorithms," *IEEE Trans. Softw. Eng., vol. 41, no. 4, pp. 358–383*, 2015. (Citato alle pagine 17, 41, 42, 45, 62 e 63)

[34] J. H. Holle, "Adaptation in natural e artificial systems." *Ann Arbor, MI, USA: Univ. Michigan Press*, 1975. (Citato a pagina 18)

[35] A. E. E. e C. A. Schippers, "On evolutionary exploration e exploitation," *Fundam. Inform., vol. 35, no. 1–4, pp. 35–50*, 1998. (Citato a pagina 18)

[36] A. D. L. e M. Di Penta e R. Oliveto e A. Panichella, "Estimating the evolution direction of populations to improve genetic algorithms," *Proc. 14th Int. Conf. Genetic Evol. Comput. Conf., pp. 617–624*, 2012. (Citato alle pagine 18 e 20)

[37] T. H. e K. Kobayashi e M. Nishioka e M. Miki, "Diversity maintenance mechanism for multi-objective genetic algorithms using clustering e network inversion," *Proc. 10th Int. Conf. Parallel Problem Solving Nature: PPSN X, pp. 722–732*, 2008. (Citato a pagina 18)

[38] Q. Z. e Y.-W. Leung, "An orthogonal genetic algorithm for multimedia multicast routing," *IEEE Trans. Evol. Comput., vol. 3, no. 1, pp. 53–62*, 1999. (Citato a pagina 18)

[39] J. Z. e G. Dai e L. Mo, "A cluster-based orthogonal multiobjective genetic algorithm," *Comput. Intell. Intell. Syst., vol. 51, pp. 45–55*, 2009. (Citato a pagina 18)

[40] D. E. Goldberg, "Genetic algorithms in search, optimization e machine learning," *1st ed. Reading, MA, USA: Addison-Wesley*, 1989. (Citato a pagina 18)

[41] S. W. Mahfoud, "Niching methods for genetic algorithms," *Illinois Genetic Algorithms Laboratory, Univ. Illinois at Urbana-Champaign Champaign, IL, USA, Tech. Rep. No. 9500*, 1995. (Citato a pagina 18)

[42] G. Harik, "Finding multimodal solutions using restricted tournament selection," *Proc. 6th Int. Conf. Genetic Algorithms, pp. 24–31*, 1995. (Citato a pagina 18)

[43] H. H. e A. Arcuri e L. Brie, "Empirical investigation of the effects of test suite properties on similarity-based test case selection," *Proc. 4th IEEE Int. Conf. Softw. Testing, Verification Validation, pp. 327–336*, 2011. (Citato a pagina 18)

[44] L. G. e S. Imre, "A survey on quantum computing technology," *Computer Science Review, vol. 31, pp. 51–71*, 2019. (Citato a pagina 28)

[45] L. K. Grover, "Quantum mechanics helps in searching for a needle in a haystack," *Physical Review Letters, vol. 79, no. 2, pp. 325–32*, 1997. (Citato a pagina 28)

[46] R. Sutor, "Dancing with qubits." *Packt Publishing Birmingham, UK*, 2019. (Citato a pagina 28)

[47] E. R. J. e N. Harrigan e M. Gimeno-Segovia, "Programming quantum computers: essential algorithms e code samples." *O'Reilly Media*, 2019. (Citato a pagina 28)

[48] A. A. e A. Corcoles e L. Bello e Y. Ben-Haim M. e Bozzo-Rey e S. Bravyi e N. Bronn e L. Capelluto e A. C. Vazquez e J. Ceroni e R. Chen e A. Frisch e J. Gambetta e S. Garion e L. Gil e S. D. L. P. Gonzalez e F. Harkins e T. Imamichi e H. Kang e A. h. Karamlou e R. Loredo e D. McKay e A. Mezzacapo e Z. Minev e R. Movassagh e G. Nannicini e P. Nation e A. Phan e M. Pistoia e A. Rattew e J. Schaefer e J. Shabani e J. Smolin e J. Stenger e K. Temme e M. Tod e S. Wood e e J. Wootton, "Learn quantum computation using qiskit," *[Online]. http://community.qiskit.org/textbook*, 2020. (Citato a pagina 28)

[49] E. F. e J. Goldstone e S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014. (Citato a pagina 28)

[50] E. F. e J. Goldstone e S. Gutmann e J. Lapan e A. Lundgren e D. Preda, "A quantum adiabatic evolution algorithm applied to reom instances of an np-complete problem," *Science, vol. 292, no. 5516, pp. 472–475*, 2001. (Citato a pagina 29)

[51] A. D. e B. K. Chakrabart, "Colloquium: Quantum annealing e analog quantum computation," *Reviews of Modern Physics, vol. 80, no. 3, p. 1061*, 2008. (Citato a pagina 29)

[52] F. G. e Gary Kochenberger e Rick Hennig e Yu Du, "Quantum bridge analytics i: A tutorial on formulating and using qubo models," *Ann Oper Res 314, 141–183*, 2022. (Citato alle pagine 37 e 39)

[53] R. E. Steuer, "Multi-criteria optimization: Theory, computation, and application," *John Wiley, New York*, 1986. (Citato a pagina 38)

[54] M. Ayodele, "Penalty weights in qubo formulations: Permutation problems," 2022. (Citato alle pagine 40 e 66)

[55] S. Y. e M. Harman e S. Ur, "Highly scalable multi objective test suite minimisation using graphics cards," *2011*, Proc. 3rd Int. Conf. Search Based Softw. Eng., pp. 219–236. (Citato alle pagine 41 e 66)

[56] A. A. e J. Bader e D. Brockhoff e E. Zitzler, "Theory of the hypervolume indicator: Optimal m-distributions and the choice of the reference point," *Proc. 10th*

ACM SIGEVO Workshop Found. Genetic Algorithms, pp. 87–102, 2009. (Citato a pagina 46)

[57] E. Z. e D. Brockhoff e L. Thiele, "The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration," *Proc. 4th Int. Conf. Evol. Multi-Criterion Optim., 2007, pp. 862–876*, 2007. (Citato a pagina 46)

[58] S. K. e C. D. Gelatt e Jr. e M. P. Vecchi, "Optimization by simulated annealing," *Science, Volume 220, Number 4598*, 1983. (Citato a pagina 54)

Questa tesi ha contribuito a piantare un albero tramite il progetto Treedom.

`https://www.treedom.net/it/user/sesalab/event/sesa-random-forest`