Università degli Studi di Catania
Dipartimento di Matematica e Informatica
Corso di Laurea Magistrale in Informatica

Cancemi Damiano

Un Algoritmo Immunologico Ibrido per il Problema del Weighted Feedback Vertex Set

---

Relazione finale

Relatore
Prof. Pavone Mario Francesco

Anno Accademico 2016/2017

Indice

Introduzione

Negli ultimi anni i problemi legati al Feedback Vertex Set sono stati oggetto di un crescente interesse. Questi hanno trovato applicazioni in molti campi, inclusa la prevenzione di deadlock nei sistemi operativi, la verifica dei programmi, nelle inferenze bayesiane e nella verifica del soddisfacimento dei vincoli dei problemi.

Per questo motivo, sono stati creati diversi algoritmi approssimati per risolvere questi tipi di problemi che fanno parte della classe di problemi NP-completi, ovvero che non possono essere risolti in tempo polinomiale.

Gli algoritmi immunologici, grazie alle loro caratteristiche chiave come il learning e la memoria sono ampiamente utilizzati per la risoluzione di problemi NP-completi, ottenendo risultati prossimi alla soluzione ottima cercata.

Questo lavoro di tesi sperimentale è incentrato sulla creazione di un algoritmo immunologico ibrido, basato sul principio della selezione clonale, per la risoluzione del problema Weighted Feedback Vertex Set (WFVS).

L'algoritmo creato è stato testato su differenti istanze del problema,

ognuna con caratteristiche diverse (numero di nodi, numero di archi, peso dei nodi). I vari risultati sperimentali mostrano come l'algoritmo immunologico proposto sia efficiente in termini di accuratezza ed in grado di risolvere approssimativamente soluzioni anche su istanze di grandi dimensioni.

Capitolo 1

Definizioni e notazioni

Dato un grafo non diretto $G=(V,E)$, un Feedback Vertex Set (FVS) $F \subseteq V$ di *G* è un sottoinsieme di vertici tale che ogni ciclo nel grafo *G* contiene alcuni vertici in *F*. In altre parole un FVS di un grafo *G* è un sottoinsieme di vertici i quali rimossi rendono il grafo *G* aciclico.

Il problema Feedback Vertex Set consiste nel trovare *F* di cardinalità minima. Se *G* è un grafo con una funzione peso non-negativa nei vertici, allora parliamo della versione pesata del problema chiamata Weighted Feedback Vertex Set (WFVS) che consiste nel trovare un FVS di *G* di minimo peso.

Questo problema trova applicazioni in svariati campi. Per esempio, nel contesto dei sistemi operativi, il Feedback Vertex Set svolge un ruolo importante nella prevenzione/rimozione dei deadlock quando i processi richiedono ciclicamente risorse bloccanti. In questo contesto il minumum feeedback vertex set corrisponde al numero minimo di processi che bisogna interrompere. Un problema simile sorge nel contesto del design dei circuiti, dove questi ultimi sono rappresentati

da grafi i cui cicli possono generare quella che viene chiamata “race-condition”, un fenomeno che si presenta nei sistemi concorrenti e avviene quando, in un sistema basato su processi multipli, il risultato finale dell'esecuzione dei processi dipende dalla temporizzazione o dalla sequenza con cui vengono eseguiti. Un'altra applicazione riguarda l'utilizzo nelle telecomunicazioni, dove per monitorare una rete è utile trovare il minor numero di vertici di controllo.

Un'altra applicazione riguarda lo studio dei “monopoli” nella sincronizzazione dei sistemi distribuiti dove le connessioni dei network sono rappresentati da “grid graph” e “toroidal graph”.

Per questi motivi il problema del Feedback Vertex Set è stato largamente studiato in questi anni ed è uno tra i primi problemi dimostrati essere NP-completi, i più difficili problemi nella classe NP ("problemi non deterministici in tempo polinomiale"). Tuttavia è stato dimostrato che questo problema è risolvibile in tempo polinomiale per alcune particolari classi di grafi come: *permutation graphs*, *convex bipartite graphs*, *k-diamond graphs* [2].

1.1. Definizioni

Sia $G=(V,E,w)$ un grafo non diretto con una funzione pesi sui vertici, dove V è l'insieme degli *n* vertici, E è l'insieme degli *m* archi, e *w* è una funzione peso positiva associata ad ogni vertice.

Dato un sottoinsieme $X \subseteq V$ di vertici, definiamo $\bar{X}=V \setminus X$ e $W(X)$ come la somma dei pesi dei suoi elementi, ovvero $W(X)=\sum_{v \in X} w(v)$ (Figura 2).

Denotiamo con $G[X]$ il sottografo di *G* indotto dall'insieme dei vertici $X \subseteq V$; formalmente definiamo $G[X]=(X, E_{[X]}, w)$ dove $E[X]=\{(u,v) \in E : u,v \in X\}$.

Un albero è un grafo aciclico e connesso, mentre una foresta è un grafo aciclico dove ogni componente connessa è un albero. Dato un insieme di vertici $X \subseteq V$, il grafo residuo di G, generato da X, è il sottografo $G[\bar{X}]$. L'insieme X è un Feedback Vertex Set (FVS) di G se il grafo residuo $G[\bar{X}]$ è una foresta.

Nel grafo mostrato nella Figura 1a, l'insieme dei vertici $X=\{6,\ 16\}$ è un FVS di G poiché $G[\bar{X}]$ è una foresta (Fig. 1b)

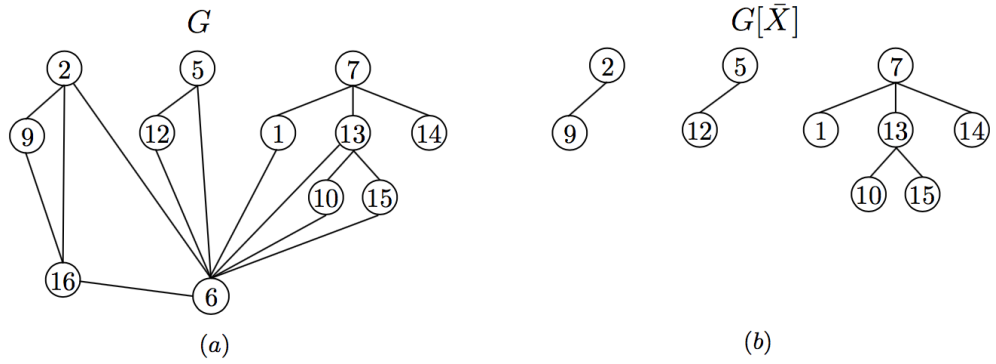

Figura 1

Denotiamo con $F(G)$ e con $F'(G)$ ogni FVS e un minimun weighted FVS (soluzione ottimale) di G, rispettivamente.

Un vertice $v \in F(G)$ è ridondante se $F(G) \backslash \{v\}$ è già un FVS di G. $F(G)$ è un FVS minimale se non contiene vertici ridondanti, in altre parole non può esserci un FVS $F'(G)$ tale che $|F'(G)| < |F(G)|$ e $F'(G) \subset F(G)$.

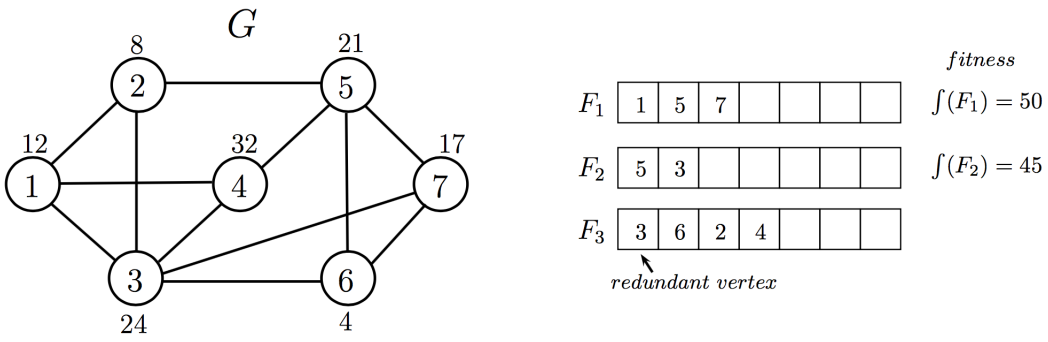

Figura 2

Nella figura 2, vengono mostrati un grafo G e tre FVS relativi. Qui F_1 e F_2 sono due FVS di G e la loro fitness, calcolata come la somma dei pesi dei nodi che compongono la soluzione, viene mostrata sulla destra. L'ultimo FVS F_3 invece non rappresenta una soluzione poiché, rimuovendo il vertice 3 da F_3 , otteniamo un FVS di G con fitness (peso totale) minore.

Capitolo 2

Algoritmi immunologici

Prima di trattare gli algoritmi immunologici è utile descrivere brevemente il funzionamento del sistema immunitario degli esseri viventi.

Gli esseri umani, similmente agli altri organismi viventi, sono costantemente esposti ad un'ampia gamma di microrganismi come batteri, virus, parassiti e altre molecole dannose (chiamate antigeni) che possono danneggiare il corpo umano.

Il sistema immunitario è un sistema di difesa molto complesso composto da cellule diverse (linfociti B e T) che impediscono agli oggetti estranei di danneggiare il corpo. La cellula T è un tipo speciale di globulo bianco che è di importanza fondamentale per il sistema immunitario. Sulla superficie presenta i cosiddetti recettori delle cellule T con la quale può rilevare antigeni. Normalmente, i recettori di una cellula T non corrispondono alle sostanze proprie del corpo.

Poiché nel corso degli anni il corpo umano è continuamente attaccato da alcuni antigeni, il sistema immunitario non ha solo l'obiettivo di

distruggere questi antigeni, ma anche di memorizzarli per poterli riconoscere successivamente. Il sistema immunitario ha quindi l'importante capacità nel riconoscimento dei pattern differenziando le cellule esterne dalle cellule del proprio corpo.

Il metodo del Sistema immunitario artificiale (in inglese Artificial Immune System da cui l'acronimo AIS) è un tipo di algoritmo di ottimizzazione utilizzato in molteplici applicazioni reali nell'ambito dell'informatica e dell'ingegneria.

Questa tipologia di algoritmi randomizzati, ispirati ai principi e ai processi del sistema immunitario degli esseri viventi. Nascono dal "paradigma" della evoluzione biologica della specie e sfruttano le caratteristiche della memoria e apprendimento (learning) per risolvere problemi complessi. Sono legati all'intelligenza artificiale, al machine learning, alla computazione bioispirata e naturale.

Lo scopo dell'utilizzo di questa classe di algoritmi per la risoluzioni di problemi nasce dall'identificazione di alcune proprietà del sistema immunitario che sono attraenti da un punto di vista computazionale. Queste includono:

- l'auto-organizzazione di un grande numero di cellule

immunitarie e il funzionamento distribuito del sistema immunitario in tutto il corpo;

- il pattern recognition e l'individuazione di anomalie per consentire al sistema immunitario di riconoscere gli agenti patogeni;
- ottimizzazione e memoria, cioè ricordare tutte le entità estranee già riconosciute ed eliminate, permettendo così una veloce e più efficace risposta immunitaria.

Questi algoritmi fanno parte della classe degli algoritmi evolutivi che usano una fase di "local search" per intensificare la fase di ricerca della soluzione. È proprio il concetto di evoluzione, legato alle soluzioni candidate del problema, che ne permette la diversificazione tramite operatori di clonazione, di mutazione e di invecchiamento.

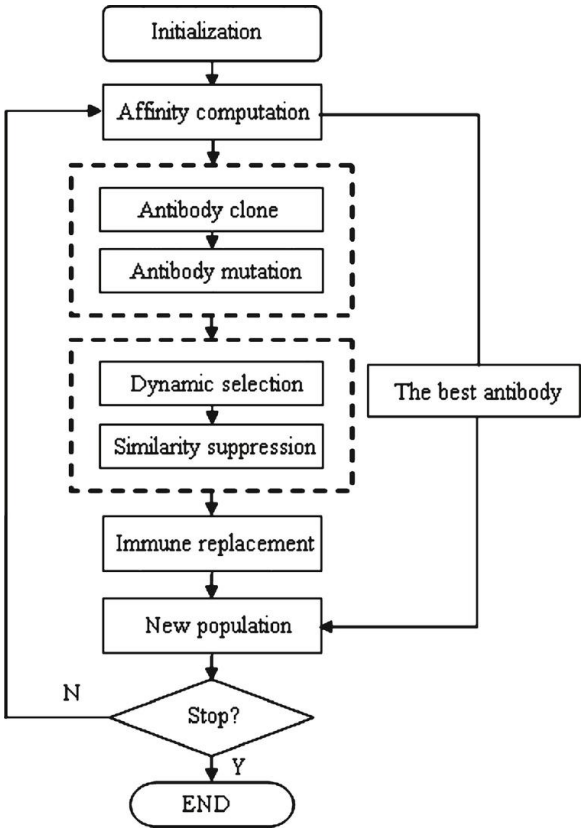

Figura 3: *flowchart degli algoritmi immunologici*

Le diverse tecniche esistenti per l'implementazione degli algoritmi immunologici, si suddividono in:

- *Clonal Selection Algorithm*: una classe di algoritmi ispirati dalla teoria della seleziona clonale che spiega come i linfocidi B e T migliorano la loro risposta immunitaria agli antigeni. Questi algoritmi si concentrano sugli attributi della teoria darwiniana in cui la selezione è ispirata dall'affinità delle interazioni antigeni-anticorpi. Qui la riproduzione è ispirata dalla divisione cellulare e

la diversificazione dall'ipermutazione somatica. Gli algoritmi di selezione clonale sono applicati maggiormente a domini di ottimizzazione e al pattern recognition.

- *Negative Selection Algorithm*: sono algoritmi ispirati ai processi di selezione positiva e negativa che si verificano durante la maturazione delle cellule T. La selezione negativa si riferisce all'identificazione e all'eliminazione (apoptosi) delle cellule. Vale a dire che le cellule T possono selezionare e attaccare i loro stessi tessuti. Questa classe di algoritmi viene usata tipicamente per problemi di anomaly detection, classificazione e di pattern recognition dove lo spazio del problema è modellato solamente dalle conoscenze disponibili.
- *Immune Network Algorithms*: sono algoritmi ispirati alla teoria proposta da Niels Kaj Jerne che descrive la regolazione del sistema immunitario con anticorpi anti-idiotipici (anticorpi che scelgono altri anticorpi). Questa classe di algoritmi si concentra sulle strutture delle reti dove i nodi rappresentano generalmente gli anticorpi. Questi algoritmi vengono utilizzati per clustering, per la visualizzazione dei dati, in problemi di

ottimizzazione e per la condivisione di proprietà in reti neurali artificiali.

L'algoritmo sviluppato in questo lavoro di tesi sperimentale si basa sul principio della "selezione clonale" descritto precedentemente. Gli algoritmi basati su tale principio rientrano in una particolare classe chiamata *Clonal Selection Algorithms* (CSA).

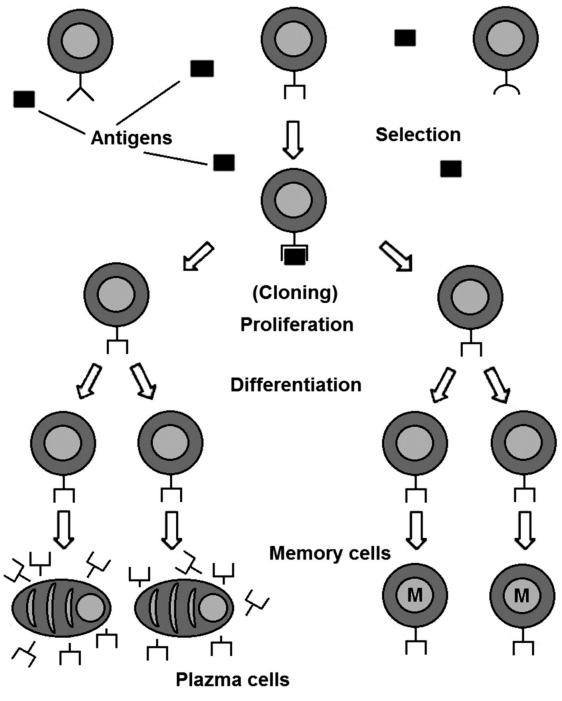

Figura 4: *principio della selezione clonale*

Le principali caratteristiche di questi algoritmi sono l'operatore di cloning (clonazione), l'operatore di hypermutation (ipermutazione) e l'operatore di aging (invecchiamento).

Il primo si occupa di duplicare ogni soluzione candidata all'interno della popolazione al fine di poter esplorare il suo "vicinato" nello spazio di ricerca. L'operatore di ipermutazione si occupa di mutare ogni soluzione candidata utilizzando una legge inversamente proporzionale sulla sua funzione obiettivo (nel nostro caso il peso totale di un FVS). Infine l'operatore di aging rimuove tutte le vecchie soluzioni candidate dalla popolazione corrente per introdurre diversità e quindi evitare minimi locali durante il processo di evoluzione.

Gli operatori verranno descritti in dettaglio nelle sezioni a seguire.

2.1. Clonal Selection Algorithms

Nei sistemi immunitari artificiali questa classe di algoritmi trae ispirazione dalla teoria della selezione clonale, ampiamente utilizzata per lo sviluppo di Sistemi Immunitari Artificiali (AIS) utilizzati in applicazioni per l'ottimizzazione computazionale e di riconoscimento dei pattern.

La teoria della selezione clonale propone che gli antigeni scelgano dei linfocidi (cellule B e T) a cui legarsi. Quando un linfocida viene scelto e si lega ad un antigene, la cellula prolifera producendo migliaia di copie di se stessa sotto forma di plasma e cellule di memoria.

La caratteristica importante di questa teoria è che la duplicazione della cellula è sottoposta a piccoli errori di copia (modifiche al genoma chiamate ipermutazioni somatiche) che modificano la forma dei recettori espressi e le successive capacità di riconoscimento del determinante antigenico di entrambi gli anticorpi

Le componenti principali di questi algoritmi sono quindi l'operatore di clonazione e l'operatore di ipermutazione (hypermutation). Il primo innesca la crescita di una nuova popolazione di cellule B (le soluzioni candidate) centrate su un valore di affinità maggiore, mentre l'ultima può essere considerata come una procedura di ricerca locale che porta ad una maturazione più veloce durante la fase di apprendimento.

Capitolo 3

Algoritmo immunologico ibrido per il problema del Weighted Feedback Vertex Set

In questa sezione viene introdotto l'algoritmo immunologico ibrido proposto che si basa, come detto precedentemente, sul principio della selezione clonale appartiene alla classe degli algoritmi evolutivi. Esso è stato sviluppato in Python ed è stata utilizzata la libreria NetworkX.

L'algoritmo prende in input una serie di parametri principali come la dimensione della popolazione, il numero di volte in cui l'operatore di clonazione dovrà clonare la popolazione, l'età massima per le soluzioni candidate, il mutation shape utilizzato dall'operatore di hypermutation e il parametro T_{max} che descrive il numero massimo di valutazioni della funzione fitness che l'algoritmo dovrà eseguire e quindi il criterio di arresto di quest'ultimo.

Inizialmente viene creato un set di soluzioni iniziali per il problema

WFVS, chiamato popolazione tramite una semplice permutazione dei vertici del grafo in input. Queste soluzioni vengono successivamente utilizzate per generare una nuova popolazione. In particolare ad ogni iterazione, la popolazione corrente viene clonata tramite l'operatore di clonazione e successivamente viene invocato l'operatore di mutazione sulla popolazione clonata. Ciò permette una migliore esplorazione delle soluzioni e la diversificazione di queste ultime.

Tutto ciò viene ripetuto finché non viene soddisfatto un criterio di arresto. Gli elementi che compongono l'algoritmo vengono descritti in dettaglio a seguire.

Come vedremo successivamente, l'insieme dei vertici generato dagli operatori di clonazione e di mutazione, può contenere vertici ridondanti. Per prevenire ciò, ogni volta che la popolazione viene clonata o modificata da questi operatori, viene applicato un controllo di ridondanza.

3.1. Inizializzazione della popolazione

Dato un grafo $G=(V,E,w)$ non diretto, la procedura *initialize-population(d)* genera la popolazione iniziale *P* di cardinalità *d* al tempo *t=0*.

Ogni elemento $p \in P$ della popolazione viene ottenuto a partire da una permutazione dei vertici *V* in *G*. In questo modo gli elementi della popolazione generati, contengono qualsiasi tipo di vertice indipendentemente dalle sue caratteristiche (peso, grado, etc.). Ciò garantisce che tutti i vertici del grafo possono partecipare al processo evolutivo dell'algoritmo, quindi la popolazione creata risulterà variegata. Ciò permette una migliore esplorazione dello spazio delle soluzioni.

Le caratteristiche chiave di ogni soluzione candidata *p* sono:

- l'età, legata al numero di cicli evolutivi per la quale l'elemento *p* dovrà rimanere all'interno della popolazione. Determina la durata della vita all'interno della popolazione: quando un elemento della popolazione raggiunge l'età massima, questo viene eliminato dalla popolazione, "muore". Questa strategia riduce la convergenza prematura dell'algoritmo e garantisce

un'alta diversificazione della popolazione.

- la fitness di *p*, corrisponde alla nostra funzione obiettivo ed è calcolata come la somma dei pesi dei primi *k* vertici di *p*, i quali rimossi dal grafo lo rendono privo di cicli.

A partire dagli elementi iniziali "grezzi" della popolazione, vengono costruite le soluzioni candidate tramite la procedura *make-solution(P)*. In particolare per ogni elemento $p \in P$, seguendo l'ordine della permutazione di p, si rimuovono i vertici di $p\{v_1, \ldots, v_k\}$ finché il grafo risultante da $G \setminus p\{v_1, \ldots, v_k\}$ non è aciclico.

Successivamente segue la fase di raffinamento delle soluzioni (local search), in quanto come accennato precedentemente, queste potrebbero essere FVS di *G*, ma potrebbero non essere dei minimun weighted FVS.

La procedura *remove-redundant-vertex(P)* si occupa del raffinamento della soluzione. Per ogni $p \in P$ prova a rimuoverne i primi k vertici seguendo un ordine decrescente della funzione peso *w(v).* Un vertice $v_i \in p$ viene rimosso da *p* se e solo se il grafo $G \setminus \{v_1, \ldots, v_k\} \cup v_i$ non contiene cicli.

Terminata la fase di inizializzazione della popolazione, ne viene calcolata la fitness per ogni elemento, utilizzando la funzione descritta in dettaglio a seguire.

3.2. Calcolo della fitness

La procedura fitness $f(p)$ viene calcolata per ogni soluzione candidata $p \in P$ tramite la procedura *compute-fitness(P)*.

La fitness di *p* è definita come la somma dei pesi $w(v)$ dei primi k vertici in *p*, i quali rimossi dal grafo lo rendono aciclico.

Formalmente: $f(p) = \sum_{i=1}^{k} w(v_i)$.

3.3. Cloning operator

Il cloning operator svolge un ruolo chiave nell'algoritmo in quanto ne influenza l'efficacia. Questo operatore permette la creazione di nuove soluzione candidate utilizzando gli elementi della popolazione già esistenti. In particolare vengono duplicati tutti gli elementi della popolazione corrente un numero *dup* di volte. Ciò produce una popolazione intermedia denominata con P^{clo} di dimensione $d \times dup$, dove sia *d* che *dup* sono parametri forniti in input dall'utente.

Agli elementi duplicati (cloni) viene assegnata un'età random nell'intervallo $[0, 2/3 \cdot \tau_B]$, che garantisce ad ogni clone un certo numero minimo di cicli evolutivi.

L'introduzione di mutazioni randomiche alla popolazione clonata può produrre individui con elevate affinità che saranno poi selezionati per formare la progenie migliorata.

3.4. Hypermutation operator

L'Hypermutation è un operatore che altera uno o più vertici all'interno di una soluzione candidata per introdurre perturbazioni e quindi fornire diversificazione nella nuova popolazione clonata. L'operatore viene applicato ad ogni soluzione candidata della popolazione $P^{(clo)}$.

Sebbene vi siano differenti modi per implementare questo operatore, in questo lavoro di ricerca è stata utilizzata una strategia "inversamente proporzionale" dove ogni soluzione candidata viene sottoposta ad M mutazioni senza utilizzare esplicitamente una probabilità di mutazione. Il numero di mutazioni M è determinato da una legge inversamente proporzionale alla fitness: migliore è la funzione obiettivo della soluzione candidata (nel nostro caso la fitness),

minore sarà il numero di mutazioni eseguite.

Sia data $\alpha = \mathrm{e}^{-\rho \hat{f}(x)}$ dove α rappresenta il mutation rate, ρ determina lo shape della mutation rates (fornito in input dall'utente) e $\hat{f}(x)$ il valore della funzione obiettivo (la fitness dell'elemento) normalizzato fra $[0,1]$, il numero di mutazioni M è dato da:

$$M = \lfloor (\alpha \times l) + 1 \rfloor$$

dove l è la lunghezza della soluzione candidata, ovvero il numero di vertici contenuti nella soluzione p.

Tramite questa equazione almeno una mutazione viene garantita per ogni soluzione; questo accade esattamente quando la soluzione rappresentata da *p* è realmente vicina alla soluzione ottima.

Una volta che la funzione obiettivo viene normalizzata nel range [0,1], le migliori soluzioni sono quelle che hanno questo valore vicino ad 1, mentre le peggiori saranno vicine allo 0.

L'operatore di mutazione quindi, per ogni $p \in P$ sostituisce *M* vertici (della permutazione iniziale) con quelli della foresta $G \setminus p\{v_1, \ldots, v_k\}$. Dopodiché ricostruisce le relative soluzioni eseguendo nuovamente le procedure *make-solution($P^{(hyp)}$)* e *remove-redundant-vertex($P^{(hyp)}$)* sulla popolazione mutata.

3.5. Aging operator

L'aging operator ha lo scopo di eliminare tutte le vecchie soluzioni candidate dalla popolazione $P^{(t)}$ e $P^{(hyp)}$. L'obiettivo principale di questo operatore è quello di produrre un'elevata diversità nella popolazione corrente ed evitare quindi la convergenza prematura dell'algoritmo.

Ogni soluzione candidata può rimanere all'interno della popolazione per un numero fissato di generazioni in accordo al parametro τ_B definito dall'utente. Quindi τ_B indica il massimo numero di generazioni permesse (l'età massima); quando una soluzione candidata p raggiunge l'età τ_B+1, questa viene scartata dalla popolazione corrente indipendentemente dal suo valore di fitness. Questo operatore viene chiamato "***static aging operator***".

L'algoritmo utilizzato invece utilizza una variante chiamata "***elitist aging operator***": quando viene generata una nuova popolazione, il meccanismo di selezione conserva la migliore soluzione trovata sino a quel momento indipendentemente dalla sua età (anche se è maggiore di τ_B).

3.6. $(\mu+\lambda)$-selection

Dopo l'esecuzione dell'aging operator, le migliori soluzioni candidate sopravvissute vengono selezionate per generare la nuova popolazione $P^{(t+1)}$, di d elementi scelti dalle popolazioni $P_a^{(t)}$ e $P_a^{(hyp)}$.

Solamente se $d_1 < d$ candidati sono sopravvissuti allora l'operatore $(\mu+\lambda)$-selection seleziona randomicamente $d-d_1$ soluzioni fra quelli scartati dall'aging operator). In particolare i candidati vengono ripescati dall'insieme:

$$((P^{(t)} \setminus P_a^{(t)}) \cup (P^{(hyp)} \setminus P_a^{(hyp)}))$$

L'operatore $(\mu+\lambda)$-selection, con $\mu = d$ e $\lambda = N_c$, riduce la popolazione creata dagli operatori di cloning e hypermutation di dimensione $\lambda \geq \mu$, ad una nuova popolazione di dimensione $\mu = d$.

L'operatore di selezione identifica quindi i migliori d elementi fra la nuova popolazione e la vecchia popolazione contenente le soluzioni candidate scartate, questo garantisce monotonicità nella dinamica dell'evoluzione.

3.7. L'algoritmo immunologico ibrido

In questa sezione viene descritto in dettaglio l'algoritmo creato, anche tramite pseudo-codice, basato sugli elementi e sugli operatori descritti nelle precedenti sezioni.

L'algoritmo combina le capacità di esplorazione degli algoritmi immunologici con un'efficiente procedura di local search e con un meccanismo di diversificazione delle soluzioni basato sugli operatori di cloning e hypermutation.

Il primo passo dell'algoritmo prevede la creazione della popolazione iniziale al tempo $t=0$ ottenuta tramite permutazioni dei vertici del grafo iniziale G.

Il ciclo while (linee 7-15) gestisce il meccanismo di diversificazione della popolazione. Questo viene ripetuto finché il valore della variabile *FFE*, che indica il numero di valutazioni della funzione fitness, non supera il valore T_{max} fornito in input. Poiché l'algoritmo potrebbe rimanere "intrappolato" in un minimo locale, lontano dalla soluzione ottimale, viene applicato uno schema di diversificazione che genera una nuova popolazione a partire dalla popolazione corrente tramite gli operatori descritti precedentemente. In questo modo l'algoritmo può

migliorare le soluzioni candidate esplorando nuove regioni dello spazio delle soluzioni. Ovviamente, al crescere del parametro T_{max} la probabilità di ottenere soluzioni migliori aumenta in quanto ciò garantisce un numero maggiore di cicli evolutivi.

Ad ogni iterazione, viene generata una popolazione che differisce dalla popolazione precedente. A tal fine gli operatori di cloning, hypermutation e local search vengono eseguiti sequenzialmente ad ogni ciclo evolutivo. Conseguentemente alla modifica della popolazione, viene eseguita la procedura *compute-fitness()* che ricalcola il valore attuale della fitness delle soluzioni canditate presenti all'interno della popolazione.

Dopo aver incrementato l'età di ogni soluzione candidata tramite la procedura di aging, l'operatore $(\mu+\lambda)$-selection seleziona le migliori soluzioni candidate che formeranno la nuova popolazione utilizzata per il successivo ciclo evolutivo.

L'algoritmo riparte quindi dalla nuova popolazione generata.

Pseudo-codice dell'algoritmo immunologico ibrido

Input: $G(V, E, w)$, d, dup, τ_B, T_{max}, [?]

Output: approximate minimal feedback vertex set di G

1 $t \leftarrow 0$

2 $FFE \leftarrow 0$

3 $N_c \leftarrow d * dup$

4 $P^{(t)} \leftarrow$ initialize-population(d)

5 compute-fitness($P^{(t)}$)

6 $FFE \leftarrow FFE + d$

7 **while** $FFE <= T_{max}$ **do**

8 $P^{(clo)} \leftarrow$ cloning($P^{(t)}$, dup)

9 $P^{(hyp)} \leftarrow$ hypermutation($P^{(clo)}$, [?])

10 $P^{(ls)} \leftarrow$ local_search($P^{(hyp)}$)

11 compute-fitness($P^{(ls)}$)

12 $FFE \leftarrow FFE + N_c$

13 ($P_a^{(t)}$, $P_a^{(ls)}$) $\leftarrow$ aging($P^{(t)}$, $P^{(ls)}$, τ_B)

14 $P^{(t+1)} \leftarrow (\mu+\lambda)$-selection($P_a^{(t)}$, $P_a^{(ls)}$)

15 t $\leftarrow$ t + 1

16 **end**

Capitolo 4

Risultati e test effettuati

L'algoritmo immunologico ibrido è stato sviluppato in Python ed eseguito in un processore 2.5 GHz Intel Core i7.

I test sono eseguiti sull'insieme di istanze di riferimento proposto in [6]. Le istanze rappresentano diverse topologie di grafi come *random graphs*, *grid graphs*, *toroidal graphs* e *hypercube graphs*. Ogni istanza è caratterizzata da un numero differente di vertici, archi e range del valore dei pesi assegnato ai vertici. Le varie istanze sono composte da 25, 50 e 75 vertici e il peso dei loro vertici è dato da un valore random scelto fra i range 10-25, 10-50 e 10-75.

La combinazione di questi parametri ci permette di verificare la robustezza delle soluzioni prodotte dall'algoritmo sopracitato.

I parametri utilizzati per effettuare i test mostrati a seguire sono i seguenti: $d=100,\ dup=2,\ \tau_B=20,\ \rho=0.5,\ T_{max}=200000.$

Nelle Tabella 1, vengono mostrati i risultati dell'algoritmo eseguito su differenti istanze di *random graphs*. Le colonne riportano per ogni istanza il numero di vertici (*n*), il numero di archi (*m*), il limite inferiore e

superiore dei pesi dei vertici (Low e Up), il valore della soluzione ottima per l'istanza corrente (Opt). L'ultima colonna (Value) riporta il valore della soluzione trovata dall'algoritmo immunologico (in grassetto quando uguale alla soluzione ottima).

I valori riportati nella colonna Value, sono ottenuti dalla media dei risultati ottimi trovati dall'algoritmo, eseguito su cinque istanze aventi le stesse caratteristiche ma con differenti pesi nei nodi.

TABELLA 1. Random graphs

Id	*n*	*m*	Low	Up	Opt	Value
S_R1	25	33	10	25	63.08	**63.08**
S_R2	25	33	10	50	99.08	**99.08**
S_R3	25	33	10	75	125.2	**125.2**
S_R4	25	69	10	25	157.6	**157.6**
S_R5	25	69	10	50	272.2	**272.2**
S_R6	25	69	10	75	409.4	**409.4**
S_R7	25	204	10	25	273.4	**273.4**
S_R8	25	204	10	50	507.0	**507.0**
S_R9	25	204	10	75	785.8	**785.8**
S_R10	50	85	10	25	174.6	176.0
S_R11	50	85	10	50	280.8	282.4
S_R12	50	85	10	75	348.0	351.6
S_R13	50	232	10	25	386.2	392.2
S_R14	50	232	10	50	708.6	740.6
S_R15	50	232	10	75	951.6	985.4

Le Tabelle 2 e 3 a seguire, riportano i risultati su istanze *squared*, *not squared grid graphs* e *toroidal graphs*, rispettivamente. Qui le colonne *x* e *y* rappresentano il numero di righe e di colonne della griglia.

TABELLA 2. Squared grid graphs						
Id	*n*	*m*	Low	Up	Opt	Value
S_SG1	5	5	10	25	114.0	**114.0**
S_SG2	5	5	10	50	199.8	**199.8**
S_SG3	5	5	10	75	312.4	**312.4**
S_SG4	7	7	10	25	252.0	260.6
S_SG5	7	7	10	50	437.0	450.4
S_SG6	7	7	10	75	713.6	787.2

TABELLA 3. Not squared grid graphs						
Id	*n*	*m*	Low	Up	Opt	Value
S_NG1	8	3	10	25	96.8	**96.8**
S_NG2	8	3	10	50	157.4	**157.4**
S_NG3	8	3	10	75	220.0	**220.0**
S_NG4	9	6	10	25	295.6	304.3
S_NG5	9	6	10	50	488.6	535.2
S_NG6	9	6	10	75	755.0	846.8

TABELLA 4. Toroidal graphs						
Id	*n*	*m*	Low	Up	Opt	Value
S_T1	5	5	10	25	101.4	**101.4**
S_T2	5	5	10	50	124.4	**124.4**
S_T3	5	5	10	75	157.8	**157.8**
S_T4	7	7	10	25	195.4	210.0
S_T5	7	7	10	50	234.2	259.4
S_T6	7	7	10	75	269.6	312.4

I test effettuati mostrano come l'algoritmo sia efficace in termine di

accuratezza. Riesce infatti a raggiungere quasi sempre la soluzione ottima o ad approssimarla notevolmente. Inoltre si presenta robusto in quanto non è influenzato né dall'intervallo dei pesi assegnati, né al numero di nodi, né dalle tipologie di grafi utilizzati.

Conclusioni

Tramite questo lavoro di tesi sperimentale è stato presentato un approccio ibrido efficace per il problema di WFVS.

L'algoritmo proposto prende spunto dal principio della selezione clonale ed è caratterizzato da due aspetti principali:

I. il primo è l'operatore di cloning che permette la duplicazione della popolazione per la creazione di nuove soluzione candidate utilizzando gli elementi della popolazione già esistenti, e
II. l'operatore di ipermutazione il cui compito è quello di mutare le soluzioni candidate al fine di ottenere una popolazione variegata e quindi l'esplorazione dello spazio delle soluzioni.

I risultati illustrati nel Capitolo 4, mostrano come l'algoritmo immunologico ibrido proposto, testato su differenti istanze, riesca a raggiungere o approssimi molti valori corrispondenti alla soluzione ottima delle istanze.

Questi risultati certificano la robustezza dell'algoritmo presentato la cui efficacia non sembra essere influenzata né dall'intervallo dei pesi assegnati, né dai nodi o dalle classi di grafi utilizzati.

Le prestazioni dell'algoritmo sono influenzate dal valore assegnato al

parametro T_{max} che ne determina il numero massimo di chiamate alla funzione fitness.

Ovviamente, più alto sarà il valore T_{max}, maggiori saranno le probabilità di ottenere soluzioni migliori e al contempo maggiore sarà il tempo di esecuzione richiesto per la risoluzione del problema.

Una miglioria dell'algoritmo in termini di prestazioni computazionali prevede il calcolo del valore di T_{max} sulla base delle caratteristiche del problema da affrontare come la densità del grafo, il numero dei nodi e il numero degli archi.

Poiché questi classi di algoritmi immunologici ibridi sono idonei per l'implementazione parallela, questa potrebbe essere una possibile direzione per un lavoro futuro che ne migliorerebbe certamente i tempi di esecuzione necessari.

Bibliografia

[1] Pavone, M., Narzisi, G., & Nicosia, G. (2012). Clonal selection: an immunological algorithm for global optimization over continuous spaces. Journal of Global Optimization, 1-40.

[2] Carrabs, F., Cerrone, C., & Cerulli, R. (2014). A memetic algorithm for the weighted feedback vertex set problem. Networks, 64(4), 339-356.

[3] Cutello, V., Nicosia, G., Romeo, M., & Oliveto, P. S. (2007, April). On the convergence of immune algorithms. In Foundations of Computational Intelligence, 2007. FOCI 2007. IEEE Symposium on (pp. 409-415). IEEE.

[4] Focardi, R., Luccio, F. L., & Peleg, D. (2000). Feedback vertex set in hypercubes. Information Processing Letters, 76(1-2), 1-5.

[5] Bafna, V., Berman, P., & Fujito, T. (1999). A 2-approximation algorithm for the undirected feedback vertex set problem. SIAM Journal on Discrete Mathematics, 12(3), 289-297.

[6] Carrabs, F., Cerulli, R., Gentili, M., & Parlato, G. (2011, January). A Tabu Search Heuristic Based on k-Diamonds for the Weighted

Feedback Vertex Set Problem. In INOC (pp. 589-602).

[7] Vazirani, V. V. (2003). Feedback Vertex Set. In Approximation Algorithms (pp. 54-60). Springer Berlin Heidelberg.

[8] Cutello, V., Nicosia, G., & Pavone, M. (2004, September). Exploring the capability of immune algorithms: A characterization of hypermutation operators. In ICARIS (Vol. 4, pp. 263-276).

[9] Cutello, V., Narzisi, G., Nicosia, G., & Pavone, M. (2005, August). Clonal selection algorithms: a comparative case study using effective mutation potentials. In ICARIS (pp. 13-28).

[10] Bafna, V., Berman, P., & Fujito, T. (1999). A 2-approximation algorithm for the undirected feedback vertex set problem. SIAM Journal on Discrete Mathematics, 12(3), 289-297.

[11] Bar-Yehuda, R., Geiger, D., Naor, J., & Roth, R. M. (1998). Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and Bayesian inference. SIAM journal on computing, 27(4), 942-959.

[12] Takaoka, A., Tayu, S., & Ueno, S. (2012, December). On minimum feedback vertex sets in graphs. In Networking and Computing

(ICNC), 2012 Third International Conference on (pp. 429-434). IEEE.

[13] Pardalos, P. M., Qian, T., & Resende, M. G. (1998). A greedy randomized adaptive search procedure for the feedback vertex set problem. Journal of Combinatorial Optimization, 2(4), 399-412.

[14] Liang, Y. D. (1994). On the feedback vertex set problem in permutation graphs. Information Processing Letters, 52(3), 123-129.

[15] Focardi, R., Luccio, F. L., & Peleg, D. (2000). Feedback vertex set in hypercubes. Information Processing Letters, 76(1-2), 1-5.

[16] Festa, P., Pardalos, P. M., & Resende, M. G. (1999). Feedback set problems. In Handbook of combinatorial optimization (pp. 209-258). Springer US.

[17] Even, G., Naor, J., Schieber, B., & Zosin, L. (2000). Approximating minimum subset feedback sets in undirected graphs with applications. SIAM Journal on Discrete Mathematics, 13(2), 255-267.

[18] Al-Enezi, J. R., Abbod, M. F., & Alsharhan, S. (2010). Artificial Immune Systems-models, algorithms and applications.