hAIcker

SILVANA DE MARTINO, SIMON CARBONE

ACM Reference Format:
Silvana De Martino, Simon Carbone. 2025. hAIcker. 1, 1 (January 2025), 10 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Contents

Author's address: Silvana De Martino, Simon Carbone.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM XXXX-XXXX/2025/1-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUZIONE

Al giorno d'oggi il mondo dell'intelligenza artificiale sta facendo passi da gigante, diventando di utilizzo rilevante per i più disparati scopi. Purtroppo, uno strumento dalle così vaste capacità, nelle mani dei più disparati tipi di persone, può generare danni di diverse entità, dalla più semplice (ma non meno dannosa) diffusione di informazioni errate, a forme più gravi come l'accesso ad informazioni riservate tramite l'impiego di prompt strutturati ad hoc. L'AI sta iniziando a diventare mezzo anche per lo sviluppo di attacchi hacker ai danni di sistemi software di diversa natura e, di contro risposta, le aziende stanno iniziando a utilizzare la stessa a fini di difendesa dei propri sistemi.

In questo testo si esamina quindi la possibilità di affrontare questa questione in termini di algoritmi genetici. Questi, partendo da un numero dato di risorse e di punti del software soggetti a vulnerabilità, permettono di definire la migliore strategia di distribuzione delle stesse risorse. Infatti, tramite delle strategie di selezione della prima generazione, il giusto tuning di parametri come crossing e mutation, l'output ottenuto può rivelarsi ottimale ai fini della determinazione dei costi e dei rischi, quindi rivelarsi utile fin dalle prime fasi dello sviluppo di un software.

L'analisi che viene fatta per il resto del documento trova le sue basi nell'astrazione del problema in questione, riportandolo ad un livello in cui tutto è riducibile ad un impiego di risorse difensivo. Si vedrà come ponderare l'impiego di risorse in base alla vulnerabilità dei punti del sistema e, in base agli attacchi, quale potrebbe essere la soluzione più promettente sulla base dei valori restituiti dalle varie generazioni che si produrranno. Lo scopo finale si propone di essere quindi quello di comprendere in che modo l'AI può assistere nella scelta della distribuzione ed impiego delle risorse, aiutando i sviluppatori fin dalle prime fasi di un progetto di System Engineering a capire quale punto del loro sistema richiederà un maggior numero di risorse e quindi di costi.

2 DESCRIZIONE DEL PROBLEMA

2.1 Obiettivi

L'obiettivo del progetto è quello di realizzare un agente intelligente che sia in grado di:

- Definire le migliori azioni difensive da adottare in maniera preventiva rispetto alle possibili azioni portate avanti da un attaccante, permettendo di valutare in base ad una stima della vulnerabilità e delle risorse a disposizione;
- Selezionare degli algoritmi di crossover e mutation validi ai fini di ottenere valori promettenti dalle funzioni di fitness;
- Determinare la modalità di selezione della prima generazione ai fini di ottimizzare i tempi di convergenza dell'algoritmo;
- Identificare funzioni di fitness che tengano in considerazione quanti più aspetti posibili senza però rendere la computazione eccessivamente complicata;

2.2 Analisi del problema

In un sistema software, la protezione delle sue componenti è una questione critica, poiché la distribuzione delle risorse di difesa può influenzare direttamente la sicurezza e le prestazioni globali del sistema. Ogni parte del sistema ha una diversa importanza relativa e una diversa vulnerabilità che devono essere prese in considerazione per allocare le risorse in modo ottimale. Tuttavia, l'allocazione non può essere arbitraria, poiché è necessario rispettare vincoli sulle risorse disponibili e bilanciare le priorità tra sicurezza e costi.

Gli elementi principali del problema includono:

(1) La necessità di rappresentare il sistema software in modo che sia possibile analizzare e ottimizzare la distribuzione delle risorse di difesa.

(2) La definizione di un metodo per quantificare l'importanza relativa e la vulnerabilità delle varie parti del sistema.
(3) La gestione delle risorse disponibili, vincolate tra un minimo e un massimo, per garantire flessibilità senza superare i limiti imposti.
(4) La necessità di bilanciare due obiettivi principali:
   - Minimizzare i danni potenziali alle componenti più vulnerabili del sistema.
   - Minimizzare i costi complessivi legati all'uso delle risorse.

Inoltre, il problema presenta una componente di complessità aggiuntiva poiché le risorse allocabili devono tenere conto di una serie di vincoli e priorità che variano in funzione dello stato del sistema e delle sue esigenze. Questo rende necessaria una strategia di ottimizzazione in grado di adattarsi a queste dinamiche complesse, garantendo al contempo la robustezza e l'efficienza del sistema.

2.3 Specifica PEAS

Definizione generica di cosa siano i PEAS

(1) **Performance** : sono definite dal valore dei danni relativi inflitti. Per danni relativi si intende la somma del valore dei danni calcolato in relazione al ranking della zona, valore che si otterrà tramite l'impiego di funzioni matematiche apposite.
(2) **Enviroment** : l'ambiente in cui opera l'agente è lo stato iniziale di un sistema software in fase di sviluppo, quindi un campo ancora teorico. Dal punto di vista pratico l'ambiente consisterà di una hashmap composta da una chiave, il ranking e quindi la vulnerabilità della zona in questione, e da un valore numerico variabile, il numero di risorse impiegate per quel determinato punto.
(3) **Actuators**: azioni difensive del sistema, si intendono in questo caso tutti quegli elementi che permettono di effettuare una miglior difesa del sistema, come firewall, password, crittografia etc. Dal punto di vista pratico dell'algoritmo agisce assegnando ad ogni punto del sistema che corrisponde ad un numero dato di risorse.
(4) **Sensors** : l'agente percepisce l'ambiente prendendo lo stato attuale e valutando il posizionamento delle risorse.

2.4 Caratteristiche d'ambiente

Breve descrizione su cosa indicano effettivamente le caratteristiche d'ambiente all'interno di un contesto definito strettamente intorno all'AI Genetica.

(1) **Completamente osservabile** : l'agente sa in qualunque momento quali sono i ranking delle varie parti del sistema e il numero di risorse massimo e minimo a sua disposizione.
(2) **Discreto** : Il numero di percezioni dell'agente è limitato
(3) **Stocastico** : l'evoluzione è influenzata da elementi casuali generati dagli algoritmi di crossover e mutation.
(4) **Parzialmente osservabile** :
(5) **Singolo agente** : per quanto vengano percepiti degli attacchi, l'ambiente si focalizza unicamente su software e sul suo comportamento al fine di minimizzare il danno subito.
(6) **Sequenziale** :la soluzione finale è data dal raffinamento sequenziale delle fasi di definizione dell'algoritmo

3 ALTRI ALGORITMI

3.1 Algoritmi di ricerca non informata ed informata locale

Algoritmi di ricerca non informata, quali ad esempio la ricerca in ampiezza (Breadth-First Search) e la ricerca in profondità (Depth-First Search), sono stati scartati a causa di una loro principale

caratteristica: la staticità. Il problema che stiamo andando a discutere in questo documento, infatti, definisce una problematicità in continua evoluzione e cambiamento, che richiede necessariamente un continuo adattamento del codice al presentarsi di nuove difficoltà e nuove priorità da parte di stakeholder o implementatori.

Un'altra caratteristica che ha portato a evitare la scelta dell'impiego di questa tipologia di algoritmi, è la loro complessità, che risulta essere molto alta, sia in termini di tempo che di spazio (ad esempio, il tempo di convergenza in problemi di ottimizzazione complessi o la gestione di grandi popolazioni). Nonostante si sappia che, con le giuste accortezze e utilizzando le opportune variazioni, questo problema possa essere efficacemente aggirato, si è comunque preferito evitare di incorrere in questi tipi di difficoltà. Immaginando, infatti, un contesto di utilizzo reale, le complessità di alcuni di questi algoritmi avrebbe reso il loro utilizzo poco efficace e avrebbero potuto portare a un loro totale mancato impiego, venendo meno al compito principale di questo progetto. Tale compito è quello di ricercare una modalità di utilizzo degli algoritmi che possa facilitare l'operato degli ingegneri del software, soprattutto nelle prime fasi, quando le uniche informazioni disponibili sono i costi e gli obiettivi del sistema, fornendo loro anche valori significativi da poter mostrare ai clienti per avvalorare o svalutare una determinata proposta in termini di ingegnerizzazione del sistema.

In un contesto così volubile, com'è quello dell'interazione con un cliente estraneo alla materia, la possibilità di modificare e personalizzare un algoritmo del genere è quindi fondamentale. Gli algoritmi in questione non avrebbero permesso questo dinamismo, non solo in termini di complessità, come si è voluto dimostrare poco sopra, ma anche in termini di personalizzazione. La ricerca non informata, infatti, non prevede la presenza di funzioni obiettivo che possano indicare un punto di arrivo, ma si basa sulla conoscenza statica di un unico fattore da trovare.

Lo stesso discorso si può sostenere per la ricerca informata, che prevede l'uso di una singola funzione obiettivo, anch'essa poco flessibile in un contesto in cui gli obiettivi possono essere descritti da più funzioni. Questi obiettivi possono essere caratterizzati da aspetti estremamente differenti, come il calcolo della sensibilità di una determinata zona del nostro sistema e i costi massimi e minimi da poter impiegare.

Di conseguenza, sia la ricerca non informata sia quella informata presentano limitazioni che le rendono inadatte a rispondere efficacemente alle esigenze di un sistema in continua evoluzione.

3.2 Teoria dei giochi/Ricerca con avversari

La ricerca con avversari è stata, durante il periodo di ideazione del progetto, la scelta che sembrava essere più valida. La possibilità di tradurre un simile problema da un contesto reale ad un contesto di gioco, permettendo così un'astrazione che ne favorisse lo studio e l'impiego successivo, sembrava essere la scelta migliore.

L'idea principale era quella di utilizzare l'algoritmo minimax e andare poi ad analizzare il suo comportamento, comparandolo con i risultati e le performance ottenute a seguito di una potatura Alpha-Beta. La teoria dei giochi trova infatti applicazione in scenari come quello proposto, ritrovando i due attori in un difensore, dal lato del mini, il cui obiettivo principale sarebbe stato quello di andare a minimizzare il valore dell'attacco inflitto, e in un attaccante, un hacker, dal lato del max e che avrebbe avuto come obiettivo quello di massimizzare i danni inflitti. I due punti di vista sarebbero stati considerati con uguale attenzione e si sarebbe andati a snocciolare il problema partendo dalla visuale dell'attaccante come spesso accade in contesti di cybersecurity.

L'impiego della teoria dei giochi risultava quindi particolarmente interessante, poiché consentiva di rappresentare la realtà in esame utilizzando un modello familiare e ben consolidato, tipico dei giochi strategici.

Cosa quindi ha portato alla decisione di non utilizzare questa tecnica? La rigidità della teoria dei giochi e la nostra poca competenza sul campo. Per quanto sembrasse semplice infatti valutare la questione come un gioco non si era tenuto in considerazione la necessità del problema di considerare più di un singolo obiettivo, questo complicava eccessivamente lo sviluppo in questo senso del problema in esame, rischiando di rendere ostico lo sviluppo dell'analisi. La teora dei giochi si rivela infatti ideale al fine di descrivere situazioni competitive con obiettivi chiari ed un'unica soluzione, mentre un sistema di sicurezza è caratterizzato da obiettivi multipli e vulnerabilità distribuite in maniera non uniforme, senza contare tutti gli aspetti all'apparenza marginali, come i costi e le priorità date dal cliente, che però nello sviluppo di un sistema possono fare la differenza. In sostanza questa teoria non si sarebbe prestata bene al problema in questi termini, data il poco dinamismo che lo caratterizza, quindi la soluzione auspicabile sarebbe potuta essere solo quella di andare a semplificare eccessivamente il problema rispetto a come lo si era inizialmente immaginato. L'altra soluzione, quella poi definitivamente intrapresa, era quella di eliminare la figura dell'hacker dalla visione generale dell'analisi, mantenendo solo il software con tutte le sue parti da analizzare e valutare.

Per concludere su questo aspetto possiamo dire che la teoria dei giochi quindi, per quanto si presti bene alla considerazione di due figure e ambiti reali come questo, non avrebbe permesso di focalizzarsi in maniera opportuna su tutti gli aspetti desiderati e considerati centrali al fine della risoluzione del problema. Difatti, ci proponiamo di fornire uno strumento agile in fase di valutazione iniziale del sistema, impiegabile per fornire valori attendibili che permettano una più semplice interazione con lo stakeholder.

4 SOLUZIONE DEL PROBLEMA

4.1 Algoritmo genetico

1. Scegliere il numero della popolazione ideale e motivarlo (magari sarebbe meglio fisso definendo il valore tra 1 e 15, meglio fare una stima delle parti dei vari sistemi, come ad esempio il sistema delle banche per capire quale potrebbe essere un numero ideale di celle per considerare tutte le varie parti di un software che potrebbero, numericamente parlando, entrare in gioco in un discorso simile) 2. criterio di inizializzazione, su quali euristiche ci si basa per creare le prima generazione

4.2 Codifica degli individui

Un individuo, prima di poter essere analizzato algoritmicamente, deve essere codificato, e cioè rappresentato in una struttura dati utilizzabile per le varie operazioni che saranno necessarie ai fini dell'ottenimento della risposta sperata. Per questo problema si è scelto di optare per un individuo estremamente semplice che però potesse contenere i valori necessari ad estrarre tutti gli altri con semplici calcoli matematici e ciò un array di elementi di sistema. Tale array contiene

- **Ranking** : il valore della cella i-esima corrisponde al ranking i-esimo del elemento del sistema. Il ranking non fa riferimento ad un elemento fisso, infatti non viene esplicitato il nome dello stesso, ma si lascia agli utilizzatori la definizione dello stesso (magari tramite apposita documentazione) in base alla realtà in cui lo si sta adoperando.
- **Risorse allocate** : ogni cella conterrà un valore intero positivo, che, in base a vincoli specifici, non possono scendere al di sotto delle 20 unità. Per quanto riguarda l'eccesso, non si è voluto imporre un limite in quanto, se dovessero essere eccedute le unità rispetto al numero totale di risorse allora l'algoritmo avrebbe fallito, lasciando alcune sezioni sprovviste di metodi di difesa adeguati.

- **Risorse allocabili** : la cella 0 non è considerabile all'interno delle operazioni che verranno successivamente spiegate, in quanto restituirebbe un ranking di valore 0 che, invece di impattare in modo significativo, andrebbe ad annullare il valore proprio del primo elemento, e quindi quello più pesante all'interno della distribuzione delle risorse. Per questo la cella 0 conterrà il numero di risorse allocabili da parte del sistema e non sarà intaccata dai cambiamenti evolutivi dell'algoritmo.

Inoltre, nonostante non facciano parte della codifica in termini di strutture dati del sistema, sembra conveniente inserire in questa parte la definizione delle seguenti componenti fondamentali del sistema:

- **Danni potenziali**: i danni non sono inflitti da un esterno, come sarebbe potuto essere per la teoria dei giochi, ma saranno calcolati sulla base di quanto danno sarebbe potenzialmente possibile infliggere ad una determinata parte del sistema, tenendo in consideranzione risorse allocate e ranking
- **Vulnerabilità** : La vulnerabilità è un concetto molto vicino al ranking, che però tiene conto della vulnerabilità totale del sistema, quindi di quanto di per sè un sistema sia "a rischio" o delicato. Il ranking tra due sistemi, che siano uno bancario ed uno relativo alla visualizzazione di ricette, avranno sicuramente un elemento che corrisponderà all'elemento di ranking 1, ma per il primo sistema la vulnerabilità sarà sicuramente più alta che per il secondo.

4.3 Funzione di fitness

Le funzioni di fitness sono fondamentali, ma prima di scendere nei dettagli c'è bisogno nel nostro contesto di fare un passo indietro. Come detto all'inizio della trattazione del problema, si è deciso di lasciare al lato computazionale la stragrande maggioranza della complessità del problema. Ora quindi ci prenderemo un piccolo spazio discorsivo per introdurre alcune semplici funzioni fondamentali.

un primo aspetto che deve essere definito prima di poter passare alla formalizzazione degli obiettivi in una funzione di fitness, è la vulnerabilità del sistema. Questo aspetto viene spiegato nel dettaglio al paragrafo 4.2.
La funzione per il calcolo della vulnerabilità della singola cella senza che si siano ancora impiegate le risorse è la seguente:

$$V_i = \lambda \cdot \frac{n}{r_i}$$

V_i rappresenta la vulnerabilità della i-esima cella, λ è una costante che determina il peso del ranking, r_i è il ranking della cella i e n è il numero totale di celle nel sistema.

La funzione che segue invece rappresenta il variare della vulnerabilità a seguito dell'impiego di risorse:

$$V_i = \lambda \cdot \frac{n}{r_i \cdot \sqrt{a_i}}$$

il valore delle risorse sotto radice indica il decrescere dell'impatto delle risorse sulla vulnerabilità del sistema stesso.

L'altra formula riguarda l'aspetto portante di questo progetto: i danni potenziali. Di questi si è parlato in alcuni paragrafi precedenti senza però esplicitare come sarebbero stati recuperati a partire dai dati che si è deciso di includere nel sistema. La formula definita a questo scopo è la seguente:

$$D_i = r_i \cdot \frac{a_i}{V_i}$$

Il danno potenziale viene calcolato moltiplicando le risorse allocate alla cella per il ranking della stessa e dividendo il risultato per la vulnerabilità della cella. Questa relazione evidenzia come il danno sia influenzato dall'importanza relativa della cella (espressa dal ranking) e dalla vulnerabilità calcolata, fornendo una misura bilanciata dell'impatto delle risorse assegnate.

Il ragionamento si basa sul fatto che, all'aumentare del ranking, il danno potenziale cresce in proporzione all'importanza della cella, ma una maggiore allocazione di risorse a_i riduce V_i, abbattendo il danno complessivo. In particolare, aumentando i valori di ranking o diminuendo le risorse allocate, il danno calcolato tende a incrementare. I danni totali sono infine calcolati come la somma di tutti i danni potenziali.

$$\sum_{i=1}^{n} D_i$$

Una volta definiti questi valori è possibile definire una funzione di fitness che permetta di accomunare due obiettivi principali:

(1) Minimizzare i danni potenziali
(2) Ridurre la vulnerabilità del sistema senza eccedere nel valore delle risorse oltre il valore impattante

La funzione di fitness relativa agli obiettivi qui indicati è la seguente:

$$\text{Fitness} = \alpha \cdot D + \beta \cdot V$$

Non è però l'unica funzione di fitness che si è ritenuto necessario definire nel nostro contesto. Infatti si sono individuati altri obiettivi ragionevoli per la realtà d'interesse che si sta andando a considerare. Tali ulteriori obiettivi sono :

- Minimizzazione dei costi, i costi sono rappresentati come le risorse impiegate nel sistema rispetto alle risorse rese disponibili;
- Distribuire in maniera pesata le risorse in maniera da dare priorità alle zone più sensibili.

Si può osservare con semplicità che il primo punto è facilmente deducibile a seguito della sottrazione tra il valore delle risorse totali, che ricordiamo essere definito nella cella 0 dell'array, e il valore di tutte le risorse allocate in tutto l'array.

4.4 Selezione degli individui

A seguito di un'attenta valutazione di vantaggi e svantaggi, combinado anche le scelte che seguiranno di crossover e mutation al fine di compensare la scelta presa, si è deciso di adottare un algoritmo di selezione di tipo K-Way Tournament. L'algoritmo, a seguito della scelta di un numero M, che verrà posto per iniziare a 10 ma il cui valore reale potrà essere soggetto a cambiamenti in fase di osservazione delle performance, riducerà il rischio di introdurre soluzioni non ammissibili. Il K-Way Tournament è infatti un algoritmo di selezione che seleziona le generazioni successive sulla base di tornei, di cui unicamente i vincitori vanno avanti nel processo evolutivo partecipando alla formazione della seguente generazione. Tale vittoria è determinata dal valore della funzione di fitness.

L'algoritmo è stato selezionato per il principale vantaggio di fornire una solida base di partenza per gli individui. Lo svantaggio principale che però questo algoritmo porta con se è quello legato alla mancanza di diversificazione degli individui. Questo tipo di svantaggio, seppur può sembrare essere banale, porta in realtà a problemi di subottimalità dell'algoritmo, che potrebbe finire per non valutare soluzioni molto valide al fine di risolvere i problemi proposti. Difatti, un individuo che a primo impatto può risultare essere ottimale, non è detto che lo sia sempre e soprattuo non è

detto che lo sia sempre per tutte le funzioni di fitness che si punta ad ottimizzare. Un individuo meno promettente a primo impatto potrebbe celare un'ottimizzazione di una delle funzioni di fitness che un individuo apparentemente meno valido invece non comporta. A questo limite è chiaramente sensibile la soluzione proposta, in quanto vengono valutate due funzioni di fitness di cui, l'ottimizzazione della seconda non è direttamente correlato all'ottimizzazione della prima, semmai lo sono inversamente, e questo potrebbe comportare che un individuo che apparentemente è estremamente valido, in realtà ottimizza solo la prima funzione di fitness senza tener conto della seconda. Nonostante ciò si è scelto comunque di adoperare questa scelta, andando a mitigare questa lacuna dell'algoritmo in questione lavorando con gli algoritmi di crossover e mutation, che quindi andranno a diminuire l'impatto della limitazione del K-Way Tournament. Inoltre, verrà fatto in fase di valutazione delle performance un tuning relativo al valore di M, assicurandosi che quello dato in base al numero della popolazione fornito sia quanto migliore possibile, assicurandosi di ottimizzare per quanto possibile il sistema.

Parlando più nel dettaglio di cosa ha portato alla selezione dell'algoritmo del K-Way Tournament, possiamo dire che la motivazione principale, se non l'unica, è che il campo di possibile applicazione reale dell'algoritmo necessita di quanta più affidabilità possibile. I vincoli che già sono presenti e quelli che potrebbero essere nel tempo richiesti sono estremamente stringenti e il mantenimento di una base solida di individui è cruciale per il successo dell'algoritmo a lungo termine. L'affidabilità delle soluzioni è pressocchè il punto focale dell'algoritmo stesso, anche se questo significa sacrificare la diversità in parte. Infatti, il campo della cybersecurity si applica a situazioni reali molto delicate, di cui l'esempio più immediato è chiaramente quello bancario, e in tali contesti l'errore non può essere contemplato, se lo è deve essere minimo o impattare aree poco o per niente sensibili. Il controllo sulla popolazione degli individui che partecipa alla generazione della popolazione successiva è quindi di fondamentale importanza e permette di avere un controllo stretto sulla qualità delle soluzioni che si andranno potenzialmente a generare.

Si è quindi fatto null'altro che un trade-off tra i vantaggi e gli svantaggi dell'algoritmo in essere.

4.5 Algoritmo di crossover

L'algoritmo scelto per il crossover è il *K-point Crossover*. In questo approccio, vengono individuati K punti casuali all'interno del cromosoma e, successivamente, i segmenti compresi tra questi punti vengono alternati tra i due genitori per creare i figli.

Inizialmente, si era considerato l'adozione di un *Two-point Crossover*, ma questo approccio non riusciva a soddisfare adeguatamente la necessità di introdurre diversità nel processo evolutivo, che risultava fortemente limitata dall'algoritmo di selezione scelto. Il *K-point Crossover*, al contrario, offre una maggiore flessibilità: il numero di tagli può essere adattato alle specifiche necessità del problema. Un numero maggiore di punti di crossover è indicato quando è necessario introdurre una maggiore diversità nelle soluzioni, mentre un numero moderato di punti è più adatto per problemi che richiedono di mantenere il rispetto dei vincoli, pur introducendo un minimo livello di diversità.

Nel nostro caso, un numero moderato di K è stato scelto per bilanciare la qualità delle soluzioni con la necessità di esplorare nuove combinazioni. Questo approccio consente di garantire la qualità senza appesantire il processo con calcoli complessi per la validazione delle soluzioni. Ad ogni passaggio, il crossover introduce un fattore di diversità che permette di esplorare soluzioni alternative, potenzialmente più ottimali, rispetto a quelle inizialmente identificate come le migliori.

Il valore di K per l'algoritmo in questione è stato fissato inizialmente a 4, ma si prevede di testare anche un valore di $K = 3$ per valutare l'effetto di una leggera variazione sul comportamento dell'algoritmo.

In conclusione, l'adozione del *K-point Crossover* si è rivelata la scelta migliore, poiché, con un numero adeguato di punti di taglio, l'interazione tra i segmenti dei genitori è sufficientemente

alta da introdurre variabilità nelle soluzioni, ma non così elevata da compromettere l'integrità dei blocchi genetici ottimali. Questo approccio permette di ottenere un buon compromesso tra la diversità e la stabilità delle soluzioni, favorendo l'esplorazione senza compromettere i vincoli critici. Concludiamo dando il valore del crossover rate che è di 0.85, senza troppa logica dietro, semplicemente è stato selezionato basandosi sui valori geenralmente impiegati in questo tipo di algoritmi.

4.6 Mutation

La mutazione è un'operazione, in questo contesto, che serve per introdurre varietà nella popolazione. Nel nostro contesto è quindi di particolare importanza, sia nelle parti iniziali per garantire una maggiore esplorazione delle varie possibili soluzioni, ma anche in fasi più avanzate del progetto per evitare che le scelte prese in precedenza, in particolare quella legata all'algoritmo di selezione, portino la soluzione a convergere troppo velocemente. La convergenza prematura è di fatto un problema che può portare a diverse conseguenze tra cui l'ottenimento di una soluzione che, seppur buona, non è quella ottimale, oppure la sospensione dei miglioramenti. Per andare a sopperire quindi a questa problematicità l'algoritmo di mutazione che si è deciso di adottare è l Adaptive. L'algoritmo di mutazione Adaptive ha la caratteristica di mutare nel corso del processo la probabilità di mutazione. Questo permette di adattare dinamicamente il livello di esplorazione dello spazio delle soluzioni possibili, evitando blocchi prematuri e convergenza troppo rapida dell'algoritmo. In oltre, la realtà d'interesse dell'algoritmo è notoriamente in continuo sviluppo ed evoluzione, in un sistema potrebbero esserci delle modifiche che potrebbero portare all'aggiunta di altre componenti del sistema ed ad un eventuale aumento delle celle dell'array, questo cambiamento non costituirebbe un problema in quanto l'algoritmo di mutazione sarebbe in grado di adattarsi a questo cambiamento senza necessitare di ulteriori verifiche, o per lo meno che non siano time-consuming. Parametri fondamentali per il corretto funzionamento di questo tipo di mutazione sono:

(1) Soglia di fitness: va definita una soglia di fitness che indichi di quanto, tra una generazione e l'altra, deve migliorare il valore. Nel nostro caso la soglia di miglioramento minima che di generazione in generazione ci deve essere per non definire un aumento della mutation rate è del 2
(2) Tasso di mutazione adattivo: ogni volta che la condizione di soprà si presenterà il mutation rate aumenterà di 2, con una semplice somma. Questo perchè andando a moltiplicarlo la crescita rischirebbe di essere troppo veloce e le mutazioni di diventare troppo frequenti. Questo stesso valore verrà sottratto quando ci sarà un miglioramento più netto tra i valori della funzione di fitness precedente rispetto a quelli della successiva.

Questi due parametri che si è deciso di inserire servono a migliorare la definizione della funzione e l'andamento dell'algoritmo, pur non essendo standard si è preferito considerarle ai fini di ottimizzare i risultati ottenuti.

Per quanto riguarda infine il rate iniziale del mutation questo sarà di 0.02, che è un valore indicativo generalmente utilizzato come valore iniziale negli algoritmi genetici, consentirà una leggera mutazione iniziale che poi si andrà ad adattare nel corso del tempo.

5 IMPLEMENTAZIONE

6 ANALISI DELLE PERFORMANCES

6.1 How to write Mathematics

LaTeX is great at typesetting mathematics. Let $X_1, X_2, \ldots, X_n$ be a sequence of independent and identically distributed random variables with $\mathrm{E}[X_i] = \mu$ and $\mathrm{Var}[X_i] = \sigma^2 < \infty$, and let

$$S_n = \frac{X_1 + X_2 + \cdots + X_n}{n} = \frac{1}{n}\sum_{i}^{n} X_i$$

denote their mean. Then as n approaches infinity, the random variables $\sqrt{n}(S_n - \mu)$ converge in distribution to a normal $\mathcal{N}(0, \sigma^2)$.

6.2 How to add Citations and a References List

Per il nostro progetto abbiamo deciso di rappresentare il sistema software come un array di interi, in cui ogni cella corrisponde a un componente del sistema stesso. Ogni cella è caratterizzata da due elementi principali. Il primo è un ranking fisso, che ne rappresenta l'importanza relativa e la vulnerabilità. Questo ranking è fondamentale per determinare quanto sia critica quella specifica parte del sistema. Il secondo elemento è un valore numerico variabile, che rappresenta il numero di risorse allocabili a quel componente.

È importante evidenziare il ruolo particolare della cella zero. Questa non contribuisce direttamente alla difesa del sistema, ma viene utilizzata come registro per tracciare le risorse allocate complessivamente. Ciò permette di monitorare e regolare l'allocazione in modo più flessibile.

Ora, per quanto riguarda i vincoli del sistema: ogni componente può ricevere tra un minimo di 20 risorse e un massimo di 100 risorse. Questo intervallo è stato scelto per garantire una certa elasticità all'algoritmo. L'obiettivo è consentire all'algoritmo di ottimizzare l'allocazione, tenendo conto sia della sicurezza del sistema che dei costi complessivi.

Per l'ottimizzazione, ci siamo posti due criteri fondamentali:

Minimizzare i danni relativi. I danni sono calcolati come una funzione del ranking, della vulnerabilità della cella e delle risorse allocate. Minimizzare i costi complessivi, cercando di ridurre il numero di risorse utilizzate senza però compromettere la sicurezza globale del sistema. In sintesi, i valori presenti nelle celle del nostro array saranno utilizzati per costruire delle funzioni che calcolano i danni potenziali in ogni punto del sistema. Questo calcolo tiene conto dell'importanza specifica delle parti, grazie al ranking associato a ciascuna cella. La cella zero, pur essendo vuota per impostazione predefinita, può occasionalmente contenere il valore delle risorse complessive. Questo valore non viene considerato durante il crossover, ma può subire modifiche nella fase di mutazione per favorire l'ottimizzazione.

You can simply upload a `.bib` file containing your BibTeX entries, created with a tool such as JabRef. You can then cite entries from it, like this: [Gre93]. Just remember to specify a bibliography style, as well as the filename of the `.bib`. You can find a video tutorial here to learn more about BibTeX.

If you have an upgraded account, you can also import your Mendeley or Zotero library directly as a `.bib` file, via the upload menu in the file-tree.

REFERENCES

[Gre93] George D. Greenwade. The Comprehensive Tex Archive Network (CTAN). *TUGBoat*, 14(3):342–351, 1993.