

Studio di algoritmi stocastici per la costruzione di quadrati magici

GABRIELE BOZZOLA

Università degli Studi di Milano
bozzola.gabriele@gmail.com

Luglio 2016

Sommario

In questo lavoro vengono presentati due algoritmi stocastici per costruire quadrati magici normali, che sono matrici di numeri naturali distinti che godono della proprietà che la somma degli elementi su ciascuna riga e colonna e sulle diagonali è sempre la stessa, detta costante magica. La prima implementazione è un algoritmo genetico realizzato utilizzando diverse funzioni di fitness e metodi di selezione, mentre la seconda consiste in un algoritmo evolutivo, basato sul lavoro di Xie e Kang [10]. Nessuna delle implementazioni dell'algoritmo genetico realizzate in questo lavoro ha portato alla costruzione di quadrati magici, a causa dell'impossibilità di formulare in modo sufficientemente adeguato il problema. L'algoritmo evolutivo, al contrario, si è mostrato efficace nella costruzione di quadrati magici. Si sono quindi confrontati i risultati ottenuti con quelli originali di Xie e Kang, trovando un disaccordo in merito a come scalano i tempi di esecuzione all'aumentare dell'ordine del quadrato. Tale discostamento è probabilmente dovuto ai limiti dell'implementazione in Mathematica e all'ignoranza su come i risultati originali sono stati effettivamente ottenuti. In ogni caso questo lavoro dimostra come sia possibile implementare un algoritmo stocastico per un complicato problema di ordinamento in cui non è possibile costruire una fitness adeguata. Infine, alcune possibili modalità per migliorare l'attuale implementazione sono discusse.

I. INTRODUZIONE

UN quadrato magico di ordine N è una matrice quadrata M di dimensioni $N \times N$ contenente numeri naturali distinti disposti in modo tale che la somma dei valori su ciascuna riga, colonna e diagonale sia sempre la stessa, detta *costante o numero magico*. Qualora i numeri che compaiono sono i primi N^2 allora il quadrato è detto *normale*, e la somma che devono le linee devono soddisfare è nota a priori in quanto esiste una relazione algebrica che lega l'ordine di un quadrato magico normale con la costante magica: per una matrice di ordine N è:

$$m = \frac{1}{2}N(N^2 + 1)$$

La dimostrazione di questa elementare proprietà è riportata in appendice. Si dimostra

inoltre che, ad eccezione di $N = 2$, è sempre possibile costruire almeno un quadrato magico per ogni ordine. Un esempio di quadrato magico normale è riportato in figura 1.

6	1	8	→ 15
7	5	3	→ 15
2	9	4	→ 15
↙ 15	↓ 15	↓ 15	↓ 15 ↘ 15

Figura 1: Esempio di quadrato magico 3×3 in cui sono riportati i valori delle somme di tutte le righe, le colonne e le diagonali. A meno di riflessioni non ne esistono di differenti.

I quadrati magici hanno una storia molto lunga, iniziata probabilmente in Cina nel settimo secolo prima di Cristo, quando ai quadrati magici erano attribuite proprietà mistiche e rituali. Dalla Cina passarono in India e successivamente in Europa, dove furono continuamente considerati come oggetti dotati di poteri magici, il che spiega il loro utilizzo in discipline come l'alchimia o l'astrologia. Il primo studio matematicamente ponderato dei quadrati magici fu condotto da Simon De la Loubère alla fine del Seicento, che fornì anche un metodo di costruzione per alcuni tipi di quadrati. Da quel momento numerosi matematici si dedicarono allo studio delle proprietà dei quadrati magici, tra cui Euler e Lagrange e Lucas, anche se rimangono aperti molti interrogativi [1]. Ad oggi sono in palio 8000 \$ e diverse bottiglie di champagne per chi risolve alcuni di questi enigmi [3].

I quadrati magici hanno inoltre alcune applicazioni tecnologiche, tra cui nella crittografia [8], nella steganografia [7] (la tecnica di occultare informazioni nelle immagini), ma anche in teoria dei grafi o dei giochi, e in molti altri campi tra cui la fisica [5].

Costruire quadrati magici non è un compito semplice, in quanto il loro numero è molto piccolo rispetto a tutte le possibilità e le equazioni che definiscono la *magicità* di un quadrato non sono abbastanza stringenti per utilizzare approcci di forza bruta. Anche se a oggi non esiste ancora una formula che permetta di calcolare il numero di quadrati magici di ordine N , si possono effettuare stime sono utilizzando metodi Monte Carlo e approcci con tecniche di meccanica statistica. I risultati di queste ricerche sul numero di quadrati magici sono riportate in tabella 1, dal quale risulta evidente che la *magicità* è una proprietà molto rara.

i. Approcci deterministici

Nonostante il numero di quadrati magici sia piccolo sono disponibili alcuni metodi di costruzione da molti anni, i più semplici dei qua-

Tabella 1: Stime del numero di quadrati magici distinti N_{ms} al variare dell'ordine N dei quadrati magici normali. N_{ns} è il numero di possibili quadrati magici normali distinti costruibili all'ordine N . In questo caso due quadrati sono ritenuti distinti se non è possibile trasformare l'uno nell'altro utilizzando riflessioni. I presenti valori sono dovuti a Trump [9] e sono ottenuti con metodi di Monte Carlo Backtracking con errori intorno a 1 %.

N	N_{ms}	N_{ns}	%
2	0	$\sim 10^1$	0
3	1	$\sim 10^5$	$\sim 10^{-5}$
4	880	$\sim 10^{12}$	$\sim 10^{-7}$
5	275 305 224	$\sim 10^{24}$	$\sim 10^{-18}$
6	$\sim 10^{19}$	$\sim 10^{41}$	$\sim 10^{-22}$
20	$\sim 10^{744}$	$\sim 10^{868}$	$\sim 10^{-124}$
35	$\sim 10^{2992}$	$\sim 10^{3252}$	$\sim 10^{-250}$
50	$\sim 10^{7000}$	$\sim 10^{7410}$	$\sim 10^{-410}$

li sono già stati elencati da Kraitichik [4] nel 1942.

Per quadrati dispari è disponibile il metodo Siamese¹ (anche noto come metodo di de la Loubère), il metodo "a losanghe" di Conway e il metodo Pheru. Per i quadrati di ordine pari esiste il metodo Medjig, mentre per quelli singolarmente pari, cioè di ordine $4n + 2$ con $n \in \mathbb{N}$ sono noti il metodo LUX di John Conway e il metodo Strachey.

Questi metodi sono di due tipologie: costruzioni passo a passo, oppure completamente. Nel primo caso si inizia posizionando uno o più numeri, e con passaggi ricorsivi (detti movimenti) si utilizzano i valori già posizionati nel quadrato per posizionarne uno successivo. Nel secondo caso si parte da una soluzione di ordine N per giungere ad una di $N + 1$ utilizzando apposite trasformazioni, come nel metodo Strachey.

ii. Approcci stocastici

Gli algoritmi elencati nella sezione precedente sono stati implementati in programmi di com-

¹Simon de la Loubère è stato ambasciatore francese in Siam, luogo dove ha appreso il metodo che porta il suo nome.

puter e risultano sufficientemente efficienti per la costruzione di quadrati magici. Tuttavia, tutti questi programmi hanno una limitazione: generano sempre il medesimo quadrato fissato l'ordine. Per questo motivo tali algoritmi risultano poco generalizzabili e quindi inadatti per studiare quadrati che, oltre ad essere magici, godono di ulteriori proprietà (ad esempio i quadrati bimagici, oppure i quadrati magici vincolati). A causa di ciò è conveniente provare ad implementare algoritmi in cui ci siano fattori di casualità che permettano di costruire sempre nuovi quadrati magici. In letteratura si trovano già alcuni risultati sulla costruzione di quadrati magici con questo tipo di metodologie, e i più efficienti algoritmi stocastici attualmente realizzati sono basati su tecniche note come *metodi iper-euristici* che permettono di realizzare un quadrato magico vincolato di dimensioni 2600×2600 in meno di un minuto [2]. Questi algoritmi sono generalmente molto complessi e articolati, mentre in questo lavoro sono state implementate due soluzioni meno sofisticate: alcuni algoritmi genetici e uno evolutivo. Questi sono stati realizzati in Mathematica 8, con la scopo principale di ottenere dei programmi che effettivamente sono in grado di costruire dei quadrati magici. Gli algoritmi che riescono a convergere sono quindi stati caratterizzati in termini delle leggi di scala in funzione della dimensione dell'input e sono stati confrontati con i risultati presenti in letteratura. Si è utilizzata la versione 8 di Mathematica principalmente perché permette di parallelizzare alcune funzioni in modo estremamente semplice, come ad esempio con il comando `ParallelTable[]` o il comando `ParallelMap[]`, che sono i corrispettivi parallelizzati di `Table[]` e `Map[]`, girando su quattro core questo ha permesso di ridurre di circa il 75% i tempi di esecuzione rispetto ad un'implementazione puramente sequenziale. Questi ed altri dettagli saranno discussi successivamente.

iii. Terminologia e notazione

In questo lavoro si utilizzerà il termine *linea* indicando in modo generico una riga o una colonna, laddove i termini riga e colonna designano i medesimi concetti generalmente utilizzati in algebra matriciale. Una linea si dice *magica* se la somma dei numeri che la compongono è il numero magico.

Si compie inoltre un innocuo abuso di linguaggio utilizzando la parola quadrato intendendo quella che matematicamente è una matrice quadrata.

Quando si riporta un comando di Mathematica lo si scrive omettendo tutti i suoi argomenti, a meno che non ve ne siano alcuni importanti da indicare. Questo permette di mantenere leggibile il testo, che altrimenti sarebbe occupato da molte righe di comandi di Mathematica.

Altra terminologia e notazione verrà introdotta pian piano.

II. ALGORITMI GENETICI

Gli algoritmi genetici sono metodi di ottimizzazione stocastici che implementano le modalità dell'evoluzione biologica. Questi algoritmi utilizzano una popolazione di possibili soluzioni del problema (chiamati *individui*) cercando di trovare quella che meglio approssima la vera soluzione del problema con il principio darwiniano di sopravvivenza del migliore. Tutti gli algoritmi genetici si basano su un ciclo di selezione - riproduzione - mutazione.

Selezione Inizialmente si selezionano i migliori individui della popolazione secondo un parametro che è chiamato *fitness*, la quale è una funzione dei *geni*, ovvero delle caratteristiche della soluzione, che è generalmente codificata come una stringa di numeri. Gli individui con fitness più alta hanno maggiore probabilità di essere selezionati per la riproduzione.

Riproduzione Dopo la selezione si producono figli a partire da coppie degli individui se-

lezionati precedentemente. I figli sono generati simulando quella che è la riproduzione sessuata in biologia cioè con lo scambio di geni dei genitori (questo processo è noto come *crossover*). Questa fase permette di costruire individui che hanno le caratteristiche migliori di entrambi i genitori in modo da avvicinarsi alla soluzione. Il crossover non avviene sempre, ma è eseguito con probabilità P_C , in modo che sia possibile che una coppia di genitori non si riproduca ma passi direttamente alla generazione successiva.

Mutazione Infine vengono applicate delle mutazioni sui figli che ne alterno casualmente qualche caratteristica. In questo modo ci si assicura di esplorare tutto lo spazio delle soluzioni e di non fossilizzarsi in un massimo locale della funzione di fitness. Come per i crossover, anche le mutazioni non avvengono sempre, ma si eseguono solo con probabilità P_M .

Una volta che si è completato il ciclo si verifica se si è trovata la soluzione del problema, altrimenti i figli della presente generazione diventano i genitori della successiva e il ciclo si ripete. Il workflow base di un algoritmo genetico è riportato in figura 2.

Il problema della costruzione di quadrati magici ha alcune delle caratteristiche adatte per essere affrontato con un algoritmo genetico:

- Lo spazio delle soluzioni è estremamente vasto, consistendo nelle permutazioni di N^2 elementi, sottoposte a $2N - 1$ vincoli (uno per ogni riga, per ogni colonna e per le due diagonali). Lo spazio delle soluzioni ha quindi $(N^2 - 2N - 1)!$ elementi (a titolo di esempio per $N = 9$ il numero di soluzioni possibili è più grande del numero di atomi presenti nell'universo osservabile).
- La complessità del problema è fortemente NP, quindi tecniche di *brute force* non sono attuabili.
- I quadrati magici possono essere codificati in modo diretto in individui dell'algoritmo genetico.

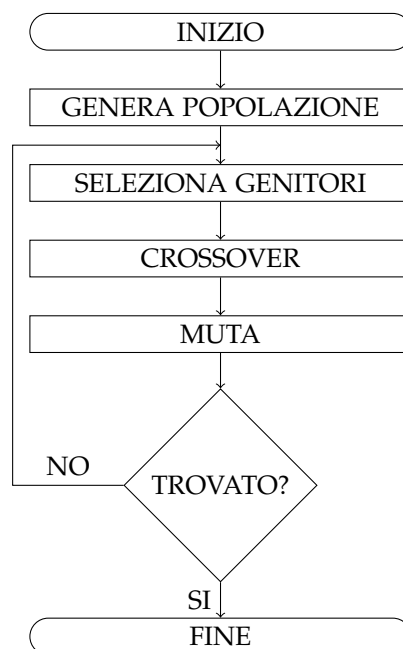


Figura 2: Diagramma di flusso di un algoritmo genetico. L'algoritmo può non convergere e rimanere bloccato nel loop.

- Il problema può essere formulato come un problema di ottimizzazione di una funzione di fitness.

Sono stati implementati diversi algoritmi genetici, i quali differiscono tra loro principalmente per la scelta della funzione di fitness e del metodo di selezione. Una volta scelti questi due parametri le variabili su cui è possibile giocare per ottenere algoritmi più performanti sono la dimensione della popolazione e le probabilità di mutazione e crossover.

i. Funzioni di fitness

Le funzioni di fitness che sono state implementate in questo lavoro sono:

- **totalSquared:** La fitness di un individuo è data dalla somma dei quadrati delle differenze di ogni linea dal numero magico.
- **totalAbs:** La fitness di un individuo è data dalla somma dei valori assoluti del-

le differenze di ogni linea dal numero magico.

- **correctLines:** La fitness di un individuo è data dall numero di linee magiche.

Ciascuna di queste funzioni di fitness ha una precisa espressione matematica in termini delle entrate della matrice e gode della proprietà che tutti quadrati magici di ordine N hanno la stessa fitness, che è zero nei primi due casi e $2N + 2$ nel terzo, e tutti i quadrati non magici hanno fitness diversa da questo valore. L'algoritmo genetico si propone perciò di minimizzare o massimizzare la fitness.

ii. Metodi di selezione e crossover

Per introdurre il principio biologico di sopravvivenza del più adatto sono stati implementati diversi meccanismi che selezionano gli individui da far riprodurre. Questi metodi di selezione sono tali da premiare le soluzioni con fitness migliore, che quindi si riprodurranno più facilmente, ma garantiscono anche sufficiente variabilità genetica, in modo che tutto lo spazio delle soluzioni sia esplorato. I metodi di selezione implementati sono:

- **fitnessProportionate:** Ciascun individuo ha probabilità di essere selezionato per riprodursi proporzionale alla sua fitness, o all'inverso della fitness per quelle funzioni per cui gli individui migliori sono quelli con fitness minore.
- **similarSquare:** Ciascun individuo ha una probabilità di essere selezionato per fare crossover proporzionale ad un fattore che dipende dalla fitness è dalla sua distanza dall'individuo migliore. In questo contesto per distanza si intende il numero di permutazioni necessarie a trasformare un quadrato in un altro. In un certo senso questo metodo è una implementazione di un sigma scaling [6].
- **fittestests:** I migliori $nFittestests$ individui si riproducono con eguale probabilità, dopodiché la popolazione è ripristinata alla dimensione iniziale aggiungendo il giusto numero di individui casualmente generati.

- **elitism:** Un dato numero di individui è selezionato per passare direttamente alla generazione successiva, i rimanenti si riproducono con uno dei metodi precedenti.

Una volta che si è prodotta la lista dei genitori si fanno accoppiare casualmente tra di loro con lo scambio di geni tramite crossover. Si sono utilizzate due tipologie di crossover:

- Ad un punto verticale od orizzontale: si seleziona una posizione all'intero del quadrato, i figli sono ottenuti incrociando le porzioni di quadrato che precedono e che succedono tale posizione tra i due genitori.
- Ad due punti verticale od orizzontale: si selezionano due posizioni all'interno del quadrato: i figli sono ottenuti scambiando tra i due genitori i numeri compresi tra le due posizioni.

La distinzione tra verticale e orizzontale consiste nel criterio con cui alcuni numeri sono considerati antecedenti o posteriori alla posizione di scambio. In figura 3 è mostrata questa distinzione. A tutti gli effetti, crossover verticale e orizzontale sono equivalenti a meno di una trasposizione.

Qualora sia in uso la fitness **correctLines** si utilizza invece un altro crossover: si selezionano una riga o una colonna da entrambi i quadrati iniziali: i figli sono uguali ai genitori a meno di queste due linee scambiate.

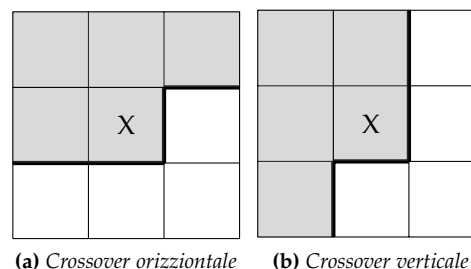


Figura 3: Crossover ad un punto. Assumendo che sia estratto il punto X come punto di taglio per il crossover, in un caso si considera come elementi precedenti quelli che sono

Siccome si vogliono costruire quadrati magici normali è necessario che tutti i numeri siano distinti, per questo motivo i crossover possono produrre degli individui non validi. Qualora questo succeda i quadrati vengono aggiustati eliminando i numeri doppi e sostituendoli con valori accettabili estratti casualmente.

iii. Implementazione in Mathematica

Sia gli algoritmi genetici che gli quelli evolutivi necessitano di una funzione che permetta di generare individui completamente casuali. Nei primi per costruire quadrati generici di ordine N si utilizza il metodo `generateSquare[order]` che procede in questo modo:

1. Si genera un numero casuale da 1 a N^2 e lo si aggiunge alla lista `used`. Questo è l'elemento m_{11} .
2. Si estraggono numeri casuali da 1 a N^2 finché non se ne trova uno che non è nella lista `used`. Questo è l'elemento m_{12} .
3. Si aggiunge il numero estratto alla lista `used`.
4. Si rieseguono tutte le operazioni di questo elenco a partire dalla seconda finché tutto il quadrato non è stato riempito.

Questo metodo non è efficiente, e può essere migliorato notevolmente utilizzando i comandi nativi `Intersection[]` e `RandomChoice[]`. Tuttavia non si è percorsa questa strada perché il secondo comando è disponibile solo nelle versioni più recenti di Mathematica e inizialmente si è lavorato su macchine con Mathematica 4.0 o 5.2. In ogni caso, compilando la funzione con `Compile[]`, l'impatto temporale di questi metodi è completamente trascurabile rispetto al resto del programma, e per questo motivo non vi è stata la necessità di aggiornarlo con un'implementazione più efficiente.

Oltre a costruire i quadrati, è fondamentale poter accedere a tutte le loro caratteristiche salienti (somme delle varie linee, deviazioni dal

valore magico, ...) in modo veloce, e per questo sono stati predisposti dei metodi che utilizzando `Table[Total[...]]` costruiscono liste che contengono tutte le informazioni di cui si ha bisogno.

In particolare una delle informazioni principali che si estraggono da un quadrato è il suo valore di fitness, e per far ciò è stata implementata la funzione `fitness[square, type]`. Questo metodo riceve come argomenti il quadrato di cui bisogna valutare la fitness e il tipo di fitness con cui bisogna effettuare il calcolo, a scelta tra quelli elencati precedentemente. Per questo motivo tutti i metodi del programma che necessitano del valore di fitness hanno un argomento `type` in cui si specifica la funzione di fitness da utilizzare. Per gestire le varie combinazioni di parametri con cui è possibile lanciare l'algoritmo genetico, nel caso della funzione di fitness, e in molti altri casi, si è utilizzato il costrutto `Switch[]` su parametri passati come argomenti alle funzioni.

Una volta implementate tutte le funzioni che agiscono su un singolo individuo, l'estensione a tutta la popolazione avviene immediatamente tramite la funzione `Map[]` cui cui è possibile valutare proprietà di tutta una popolazione in modo estremamente rapido e pulito.

Due metodi importanti che permettono di gestire tutte le varie funzioni di fitness implementate sono `targetFitness[type]` e `whoIsTheBest[pop, type]`. La prima restituisce il valore di fitness che i quadrati magici hanno se misurati con la fitness `type` (questo è zero se `type` è `totalSquared` o `totalAbs`, oppure $2N + 2$ se è `correctLines`). La seconda funzione, invece, restituisce l'individuo che più si avvicina ad essere un quadrato magico. Questo secondo metodo effettua questa selezione utilizzando il comando `Position[]`, con argomento `Min[fitnessPop]` o `Max[fitnessPop]` a seconda della fitness in uso.

Per gestire il processo di riproduzione si è implementato il metodo `reproduce[popolation, P_M]` che a sua volta chiama `divideInterval[popolation]`, `chooseParents[popolation]` e

`crossoverOne[parents]`. La prima di queste determina con quale probabilità gli individui verranno selezionati per andare incontro a crossover. Per fare questo si partiziona un intervallo da 0 a 1 in intervallini disgiunti ciascuno dei quali corrisponde ad un individuo della popolazione. Le reali dimensioni dell'intervallo dipendono dalla fitness dell'individuo e dal metodo selezionato. Per scegliere quindi quali genitori far riprodurre si estraggono due numeri reali casuali tra 0 e 1, i genitori associati agli intervallini in cui cadono i due punti saranno inseriti nella *mating pool*, ovvero nell'insieme di individui che si riproducono. I quadrati migliori avranno intervallino più lungo e quindi saranno estratti con maggiore probabilità. L'implementazione di `divideInterval[]` è piuttosto lineare, l'unico punto in cui bisogna fare attenzione è che se si vuole ottenere una lunghezza dell'intervallo che rispecchi la bontà dell'individuo, cioè individui migliori hanno intervalli più lunghi, nel caso si lavori con le fitness `totalSquared` e `totalAbs` bisogna utilizzare i reciproci dei valori di fitness, poiché valori più bassi corrispondono a individui migliori.

Per selezionare le coppie che fanno crossover si utilizza la funzione `chooseParents[mating_pool]` che accoppia i genitori prendendo il primo con l'ultimo, il secondo con il penultimo, e così via. Per questo motivo si lavora esclusivamente con popolazioni composte da un numero pari di individui. A questo punto il metodo per effettuare crossover, `crossoverOne[]`, si occupa di produrre due figli a partire da due individui che gli sono stati passati come argomento, assieme alla probabilità di crossover e al tipo di crossover da eseguire. Il metodo segue queste operazioni per le fitness `totalSquared` e `totalAbs`:

1. Viene estratto casualmente se il crossover è orizzontale o verticale. Se il crossover è verticale si esegue `Tranpose[]` sui quadrati e si procede come se il crossover fosse orizzontale.

2. I quadrati vengono "srotolati" diventando dei vettori di lunghezza N^2 . Per far ciò si usa il comando `Flatten[]`.
3. Se il crossover è ad un punto viene estratto un numero casuale da 1 a N^2 , se è a due punti ne vengono estratti due.
4. Utilizzando `Take[]` si producono due o tre sezioni di ciascun genitore.
5. Si generano i figli ricombinando le porzioni di quadrato prodotte nel punto 4 in modo da incrociarsi.
6. Si sottomettono i figli a `deleteDouble[square]` che è una funzione che elimina i doppietti sostituendoli con numeri accettabili, la cui implementazione ricalca quella utilizzata per generare da zero quadrati normali.

Nel caso si utilizzi la fitness `correctLines`, in questo caso il crossover consiste nel prendere una riga magica o una colonna magica da ciascun genitore e di scambiarle, questo nello spirito di preservare i buoni risultati già conseguiti. La procedura dettagliata con cui si compie questo crossover è:

1. Si estrae un numero casuale tra 0 e 1, questo determina se cercare di scambiare prima le righe o prima le colonne.
2. Si generano le liste contenenti le righe e le colonne non magiche.
3. Se il numero estratto è 0 e vi sono almeno una riga non magica in entrambi i quadrati si procede scambiando due righe a caso. Se non vi sono, si cerca di scambiare due colonne magiche, se ci sono. Al contrario se il numero estratto è 1 si cerca prima di scambiare le colonne, e se non vi sono colonne magiche si scambiano le righe. Se non è possibile fare nessuno scambio si producono due figli uguali ai genitori.

E' necessaria questa implementazione barocca perché se si fosse evitata si sarebbe data priorità allo scambio di righe piuttosto che a quello di colonne in quanto l'algoritmo controlla

prima se vi sono righe magiche, e se vi sono effettua lo scambio. Così facendo in media l'algoritmo avrebbe effettuato più scambi di righe che di colonne, e non c'è alcun motivo perché ciò debba accadere, in questo modo invece si garantisce simmetria tra le righe e le colonne.

Un altro dei metodi fondamentali degli algoritmi genetici è quello responsabile delle mutazioni. Per produrne una viene generato un numero casuale intero tra 1 e 5 che determina il tipo di mutazione che sarà applicata. Quelle implementate sono nell'ordine: scambia una coppia di numeri, scambia due colonne, scambia due righe, permuta una riga e permuta una colonna. Una volta selezionata la mutazione si estrae un numero casuale reale tra 0 e 1, se questo è minore della probabilità di mutazione allora questa viene applicata. In particolare, per le ultime due tipologie di mutazioni si sostituisce la linea con un elemento causale dall'insieme generato da `Partitions[line]`, mentre l'implementazione delle altre è molto diretta e si basa su `Replace[]`. Anche le mutazioni sono state implementate per un singolo individuo e successivamente estese a tutta la popolazione con `Map[]`.

Nel complesso i metodi per produrre una nuova popolazione a partire dalla precedente sono stati raccolti nella funzione `reproduce[population]`, che restituisce una lista di figli prodotti dalla popolazione attuale applicando tutte le funzioni sui vari individui dopo aver effettuato i controlli necessari per gestire le varie opzioni, come ad esempio l'elitismo. Per preservare un numero e di quadrati da una generazione alla successiva si procede in questo modo:

1. Si riordina la popolazione dall'individuo migliore a quello peggiore utilizzando la fitness e stando attenti che la definizione di migliore o peggiore dipende dalla funzione di fitness in uso.
2. Si memorizzano i primi e migliori individui in una lista a parte e li si eliminano dalla popolazione corrente.
3. Si applica `reproduce[]` alla restante popolazione.

4. Si uniscono la lista dei figli prodotti con `reproduce[]` e la lista degli e individui salvati con il comando `Join[]`.

Per evitare di avere individui doppi in una popolazione si usa la funzione `purge[]` che elimina i quadrati doppi rimpiazzandoli con quadrati generati casualmente da zero.

L'algoritmo genetico nel suo complesso è stato implementato in `run[]`, il quale consiste in un ciclo `While[]` che implementa il loop in figura 2 e memorizza statistiche in una lista, che verrà consegnata come output. Questa lista contiene la fitness minima, quella massima e quella media. Il tempo di esecuzione e il numero di generazioni è stampato a video in caso di convergenza o se si giunge al numero di generazioni limite. Per gestire le varie possibili configurazioni con cui si desidera lanciare l'algoritmo genetico il comando supporta delle opzioni, ovvero degli argomenti che vengono passati con `->`. Per far ciò si è utilizzato la funzione `OptionsPattern[]` inizializzata con dei valori di default in modo che anche se non viene passato nessuna opzione a `run[]` questo venga eseguito lo stesso. In ogni caso `run[]` richiede come minimo l'ordine dei quadrati e le probabilità di mutazione e crossover da passare come argomenti. Una delle opzioni che è possibile definire consiste nel numero massimo di generazioni che l'algoritmo fa e serve a evitare di far girare il programma indefinitamente.

Siccome gli algoritmi genetici si sono rivelati inefficaci non si è proceduto ripulendo e raffinando il codice, che quindi presenta delle implementazioni poco ottimizzate.

III. ALGORITMO EVOLUTIVO

Gli algoritmi genetici si sono rivelati inadatti a risolvere il problema della costruzione dei quadrati magici, come verrà approfondito successivamente. Per superarne i problemi si è mostrato conveniente:

1. Eliminare la fase di crossover e aumentare il numero di mutazioni effettuate sul singolo individuo.

2. Effettuare controlli sistematici quando l'algoritmo comincia ad essere in condizioni di stallo.

Un algoritmo evolutivo è quindi un algoritmo genetico in cui non si effettuano crossover, ma si utilizzano esclusivamente mutazioni, rese tuttavia più sofisticate per sopperire alla mancanza dei crossover, i quali in generale permettono di giungere più velocemente alla soluzione.

L'algoritmo evolutivo implementato in questo lavoro è basato sull'idea di Xie e Kang [10] che introduce alcune operazioni sistematiche da effettuare sui quadrati qualora siano soddisfatte alcune condizioni.

i. Algoritmo di Xie-Kang

L'algoritmo di Xie-Kang oltre ad implementare i miglioramenti esposti all'inizio di questa sezione aggiunge un ulteriore contributo fondamentale: *la congettura della costruzione a due fasi*.

Per enunciare questa congettura è necessario definire una nuova tipologia di quadrati: i quadrati semimagici. Una matrice composta da numeri naturali differenti $N \times N$ è detta *quadrato semimagico* se è un quadrato magico a meno delle diagonali, ovvero se la somma dei valori su tutte le righe e su tutte le colonne è uguale al numero magico. Un quadrato semimagico è normale se le sue entrate sono tutti i numeri da 1 a N^2 . In seguito ci si riferirà a quadrati semimagici intendendo quadrati semimagici normali.

Congettura della costruzione a due fasi La congettura afferma che un quadrato semimagico è sempre completabile ad un quadrato magico utilizzando un numero finito di permutazioni di righe e di colonne oppure di rettificazioni locali (saranno definite in seguito).

Non è disponibile una dimostrazione per questa affermazione, che è per questo motivo definita *congettura*.

i.1 Codifica degli individui

A differenza degli algoritmi genetici, nell'algoritmo evolutivo l'individuo non è direttamente il quadrato, bensì è una coppia di due matrici (M, Σ) . Il primo quadrato, M , è quello composto dagli interi da 1 a N^2 che si cerca di rendere magico, mentre il secondo, Σ , contiene delle informazioni utili per le mutazioni. Gli elementi σ_{ij} di Σ determinano il range di valori che l'elemento $m_{ij} \in M$ può assumere a causa di una mutazione. Σ è una matrice dinamica, che subisce essa stessa mutazioni come M e che evolve tenendo in considerazione quanto M si discosta dall'essere un quadrato magico in termini di fitness e di linee non magiche.

La funzione di fitness f adottata è differente se il quadrato è semimagico oppure se possiede ancora linee non magiche:

- Per quadrati non semimagici:

$$f(M) = \sum_{i=1}^N \left| \sum_{j=1}^N a_{ij} - m \right| + \sum_{i=1}^N \left| \sum_{j=1}^N a_{ji} - m \right|$$

- Per quadrati semimagici:

$$f(M) = - \left| \sum_{j=1}^N a_{jj} - m \right| - \left| \sum_{j=1}^N a_{j, N-j+1} - m \right|$$

Si introduce ulteriore notazione: sia $\text{col}(k)$ la somma di tutti gli elementi sulla colonna k e $\text{row}(h)$ la somma di tutti gli elementi sulla riga h . Con questa nomenclatura se una riga s è magica allora vale che $\text{row}(s) = m$ e analogamente per le colonne. Similmente sino a dg1 la somma dei termini sulla diagonale principale, e dg2 la somma dei termini sull'antidiagonale, cioè:

$$\text{dg1} = \sum_{k=1}^N a_{kk} \quad \text{dg2} = \sum_{k=1}^N a_{k, -k}$$

Dove con $-k$ si intende l'indice $N - k + 1$, che è il k -esimo elemento della matrice leggendo da destra a sinistra. Con queste notazioni la funzione di fitness può essere comodamente riscritta come:

$$f(M) = \begin{cases} \sum_{i=1}^N (\text{row}(i) + \text{col}(i)) & \text{semimagico} \\ -(\text{dg1} + \text{dg2}) & \text{altrimenti} \end{cases}$$

Si è scelto di utilizzare una fitness negativa per i quadrati semimagici in modo che ogni quadrato semimagico sia evolutivamente preferito a tutti i quadrati che hanno ancora linee non magiche. Questo accorgimento permette di trovare sempre l'individuo migliore di una popolazione anche se questa è composta da quadrati semimagici e non cercando l'individuo che ha fitness negativa più vicina a zero.

ii. Mutazioni

Siccome negli algoritmi evolutivi non vi è crossover, per esplorare lo spazio delle soluzioni si utilizzano esclusivamente mutazioni. Inoltre, non è necessario lavorare con una popolazione composta da numero elevato di individui, ma se ne utilizza uno solo, il quale produce un numero fissato di figli che subiscono mutazioni diverse.

Sono stati utilizzati due tipi di mutazioni: le mutazioni puntuali, che coinvolgono solo due numeri, e le mutazioni lineari, le quali invece coinvolgono tutti i valori di una linea.

Sia $\text{randint}(a, b)$ una funzione che estrae un numero intero nell'intervallo (a, b) con probabilità uniforme. Questa funzione è fornita da Mathematica come $\text{Random}[\text{Integer}]$ e $\text{RandomInteger}[]$. Siano inoltre n_{row} e n_{col} rispettivamente il numero di righe e di colonne non magiche del quadrato.

ii.1 Mutazioni puntuali

Si definiscono alcuni insiemi di mutazione:

- S_1 : Numeri la cui riga e colonna non è magica.
- S_{2r} : Numeri in righe non magiche.
- S_{2c} : Numeri in colonne non magiche.
- $S_2 = S_{2r} \cup S_{2c}$

Risulta inoltre che:

$$S_1 = S_{2r} \cap S_{2c}$$

A meno che uno di questi insiemi non sia vuoto, ogni individuo è mutato con una delle seguenti tipologie di mutazione selezionata in modo casuale:

Mutazione da S_1 a S_2 Ogni elemento dell'insieme S_1 è sottoposto a mutazione con probabilità $1/(n_{row}n_{col})$. Se un elemento x aventi indici i e j è selezionato per essere mutato, allora si calcola il valore:

$$n = x + \text{randint}(-\sigma_{ij}, \sigma_{ij})$$

Il quale viene aggiustato se produce risultati non accettabili:

$$\begin{cases} n = \text{randint}(1, N) & \text{se } n < 1 \\ n = N^2 - \text{randint}(0, N) & \text{se } n > N^2 \end{cases}$$

Si cerca quindi in S_2 l'elemento che più si avvicina a n , cioè il numero t in S_2 tale che sia minimizzato il valore di

$$\min_{t \in S_2} |n - t|$$

Gli elementi n e t sono quindi scambiati. Una volta avvenuto lo scambio si effettua una mutazione anche sul valore di σ_{ij} . Il nuovo valore z di σ_{ij} è dato da:

$$z = \sigma_{ij} + \text{randint}(-1, 1)$$

Anche in questo caso per evitare di avere dei valori insensati si pone:

$$z = \text{randint}(1, \sigma_t) \quad \text{se } z < 1 \quad \text{o} \quad z > \sigma_t$$

Dove σ_t definisce in qualche modo un *parametro di salute* del quadrato: σ_t è definito così:

$$\sigma_t = \begin{cases} |f(M)| / (n_{row} + n_{col}) & \text{se } n_{row}n_{col} \neq 0 \\ |f(M)| / n_{diag} & \text{se } n_{row}n_{col} = 0 \end{cases}$$

Quando σ_t è piccolo significa che il quadrato è vicino ad essere magico, e quindi bisogna fare piccole mutazioni, mentre al contrario se σ_t è grande allora significa che la fitness è alta e sono necessarie grandi mutazioni per rendere il quadrato magico.

Mutazione da S_2 a S_2 Ogni elemento x di S_2 è sottoposto a mutazione con probabilità che dipende determinata dall'appartenenza agli insiemi S_{2r} e S_{2c} :

$$P_M(x) = \begin{cases} 1/n_{row} & \text{se } x \in S_{2r} \\ 1/n_{col} & \text{se } x \in S_{2c} \\ 1/(n_{row}n_{col}) & \text{se } x \in S_{2r} \cap S_{2c} \end{cases}$$

Se x è sottoposto a mutazione si eseguono le medesime operazioni descritte per la mutazione da S_1 a S_2 .

Mutazione da S_2 a M Questa mutazione è uguale alla precedente, a meno del fatto che gli elementi con cui si scambiano quelli di S_2 sono in tutta la matrice M .

ii.2 Mutazioni lineari

Una volta che l'algoritmo trova almeno un quadrato avente tutte le righe e le colonne magiche per evitare di perdere questi risultati vengono utilizzate mutazioni lineari al posto di quelle puntuali. Queste consistono nello scambio di due righe o di due colonne scelte casualmente, in questo modo sicuramente l'individuo continua ad avere tutte le righe e tutte le colonne magiche.

Per ogni individuo si eseguono un numero casuale tra 1 e N mutazioni lineari con probabilità unitaria. Si nota tuttavia che è possibile che le due linee che vengono estratte per essere scambiate coincidano, e ciò corrisponde a non aver effettuato alcuna mutazione.

iii. Metodi di selezione

Una volta che tutta la popolazione è andata incontro a mutazione bisogna selezionare quale sarà il nuovo genitore. Vengono utilizzati due metodi di selezione (*evolution strategies*) differenti per incentivare o meno la variabilità degli individui. Questi sono metodi standard negli algoritmi evolutivi:

- $(\mu, \lambda) - ES$: La nuova generazione di genitori è formata dai migliori figli della precedente.
- $(\mu + \lambda) - ES$: La nuova generazione è formata dagli individui migliori tra i genitori e i figli della generazione precedente.

In queste diciture μ indica i genitori della generazione precedente, mentre λ i figli. L'utilizzo della seconda strategia evolutiva garantisce che non vengono persi individui buoni ma modificati dalle mutazioni, mentre l'utilizzo del

primo permette di esplorare più velocemente lo spazio delle soluzioni.

Nell'algoritmo implementato in questo lavoro sono state utilizzate entrambe le strategie evolutive a seconda di quale sia più utile nel momento della selezione.

iv. Rettificazioni locali

Senza intervenire direttamente in modo sistematico sulla costruzione dei quadrati, anche l'algoritmo evolutivo, come quello genetico, non è molto più efficiente di una ricerca casuale nello spazio delle soluzioni, e per questo motivo non converge quando lo spazio delle soluzioni è troppo grosso, e fallisce già con quadrati di piccole dimensioni. Per ovviare a questo problema Xie e Kang hanno proposto di utilizzare delle *rettificazioni locali*: quando gli individui sono abbastanza buoni vengono passati in rassegna per cercare se vi sono degli scambi che diminuiscono il numero di righe sbagliate oppure che rendono le diagonali magiche. Si definiscono *rettificazioni locali lineari* quelle che hanno come obiettivo la diminuzione delle righe non magiche, mentre *rettificazioni locali diagonali* quelle che intendono aggiustare le diagonali.

iv.1 Rettificazioni locali lineari

Sono state implementate due tipologie di rettificazioni locali lineari: lo scambio di una coppia, e lo scambio di due coppie tali che soddisfino alcune condizioni. Queste condizioni garantiscono che lo scambio renda magica una o più linee senza rovinare le altre. Non si fanno analisi a più di due coppie perché sarebbe un compito computazionalmente troppo oneroso. Ricordando che $\text{col}(k)$ è la somma degli elementi sulla k -esima colonna e $\text{row}(h)$ è la somma degli elementi sulla h -esima riga:

Scambio di una coppia Due numeri a_{ks} e a_{ls} sono scambiati alla riga k e l e alla colonna s se sono soddisfatte:

- $\text{row}(k) - m = a_{ks} - a_{ls}$
- $m - \text{row}(l) = a_{ks} - a_{ls}$

Un esempio di questa rettificazione è riportato in appendice.

Scambio di due coppie I numeri a_{ks} e a_{lt} sono scambiati con i numeri a_{ls} e a_{kt} (il primo con il primo e il secondo con il secondo), corrispondenti alle righe k e l e alle colonne s e t se sono soddisfatte:

- $\text{row}(k) - m = a_{ks} + a_{kt} - a_{lt} - a_{ls}$
- $m - \text{row}(l) = a_{ks} + a_{kt} - a_{lt} - a_{ls}$

Le condizioni duali sono applicate anche alle colonne con le ovvie sostituzioni.

iv.2 Rettificazioni locali diagonali

Una volta che si è realizzato un quadrato semimagico per ottenere il quadrato magico è necessario un pesante intervento sistematico. I metodi implementati sono scambi di coppie di elementi, oppure di intere linee. Ricordando che $dg1$ e $dg2$ sono rispettivamente la somma degli elementi sulla diagonale e sull'antidiagonale gli scambi implementati sono:

Scambio puntuale Se sono soddisfatte le condizioni:

- $a_{ii} + a_{ij} = a_{ji} + a_{jj}$
- $(a_{ii} + a_{jj}) - (a_{ij} + a_{ji}) = dg1 - m$

allora a_{ii} è scambiato con a_{ji} e a_{ji} con a_{jj} . In modo simmetrico per l'antidiagonale.

Scambio lineare Se sono soddisfatte le condizioni:

- $(a_{ii} + a_{jj}) - (a_{ij} + a_{ji}) = dg1 - m$
- $(a_{i,-i} + a_{j,-j}) - (a_{i,-j} + a_{j,-i}) = dg2 - m$

allora la riga i è scambiata con la riga j . Simmetricamente con colonne.

v. Implementazione in Mathematica

La struttura base dell'implementazione in Mathematica dell'algoritmo evolutivo ricalca quella utilizzata per l'algoritmo genetico con

la fondamentale differenza che ora gli individui non sono più semplicemente il quadrato, ma sono coppie di matrici, di cui una è il quadrato che si vuole rendere magico, mentre l'altra contiene informazioni per effettuare le mutazioni. In questo algoritmo per manipolare direttamente quadrati è necessario un livello di profondità aggiuntivo rispetto agli algoritmi genetici, e per questo motivo la quasi totalità delle funzioni è stata riscritta per integrarsi naturalmente a questa codifica differente. Anche in questo caso la maggior parte delle funzioni è implementata per un singolo individuo, e viene estesa a tutta la popolazione con il comando `Map[]`.

A differenza degli algoritmi genetici, in questo algoritmo vi sono pochi parametri liberi, quindi l'implementazione è molto più diretta e senza la pesante infrastruttura degli `Switch[]`. Ad esempio esiste un solo tipo di fitness, che automaticamente distingue se il quadrato è semimagico oppure no, e utilizza metodi di fitness differenti nei due casi, come illustrato precedentemente.

Per trovare il miglior individuo di una popolazione la funzione implementata si chiama `fittestChild[]` e funziona in questo modo: se nella popolazione tutti gli individui hanno fitness positiva si prende quello che ha il valore minimo, se anche uno solo ha fitness negativa si prende l'individuo il cui reciproco della fitness è minore. In questo modo si seleziona sempre il quadrato più vicino ad essere magico. Per selezionare tali individui si utilizza `Position[Min[]]` che restituisce la posizione dell'individuo che nella popolazione realizza il minimo di fitness (o del reciproco).

Le mutazioni di questo algoritmo sono molto più sofisticate rispetto a quelle dell'algoritmo genetico e richiedono innanzitutto che si generino gli insiemi di mutazione S_1 e S_2 . Per far ciò è stata predisposta una funzione apposita denominata `generateMutationSets[]`. Questo metodo costruisce S_{2r} e S_{2c} passando in rassegna il quadrato e verificando se l'elemento in considerazione è soddisfa le condizioni per essere in uno dei due insiemi, e a partire da questi due insieme genera anche S_1

e S_2 per intersezione e unione di S_{2r} e S_{2c} . Un secondo metodo ausiliario necessario per eseguire le mutazione è `sigmat[individual]` il quale calcola il valore σ_t per il quadrato.

A questo punto è possibile mutare un quadrato con il comando `mutateInd[]`, che assieme a `rectifyLinesInd[]` è il cuore dell'algoritmo di Xie e Kang. `mutateInd[]` controlla inizialmente se il quadrato è semimagico o meno. Nel primo caso si estrae un numero casuale q compreso tra 1 e N e per q volte il quadrato è sottoposto alle seguenti operazioni:

1. Si seleziona se mutate righe o colonne estraendo un numero intero tra 0 e 1.
2. Si estraggono due numeri da 1 a N che corrispondono agli indici delle linee da scambiare.
3. Si effettua lo scambio.

Qualora invece il quadrato non sia semimagico i passi da seguire sono differenti:

1. Si generano gli insiemi di mutazione e si calcolano le probabilità di mutazione.
2. Si controlla se S_1 è vuoto, in questo caso la prima mutazione è inibita. Non è possibile che S_2 sia vuoto perché il quadrato non è semimagico.
3. Si seleziona una mutazione in modo casuale.
4. Si percorre tutto il quadrato, quando si trova un elemento che appartiene all'insieme di mutazione in considerazione si eseguono le operazioni riportate nel sottoelenco. Per scoprire se un elemento appartiene a un insieme di mutazione si utilizza il comando `Count[]` e si verifica se il risultato è zero o meno.
 - (a) Si estrae un numero reale tra 0 e 1, se questo è inferiore alla probabilità di mutazione si procede con il punto successivo di questo elenco.
 - (b) Si effettua la mutazione sul quadrato come descritto nell'apposita sezione

utilizzando i comandi `Min[]` per trovare con chi scambiare e `Replace[]` per effettuare lo scambio.

- (c) Si muta la matrice Σ .
- (d) Si calcolano i nuovi insiemi di mutazioni e si aggiornano i valori delle linee incorrette del nuovo individuo.

Il secondo punto critico dell'implementazione di questo algoritmo sono i metodi di rettificazione. Per implementare queste funzioni è stato necessario operare a basso livello, cioè manipolando direttamente gli indici, in quanto nessuna funzione nativa di Mathematica si è mostrata adatta allo scopo. Questi metodi devono effettuare controlli su una o due coppie di elementi del quadrato, per selezionare tali coppie si procede in questo modo:

1. Vi sono due cicli innestati sugli indici i e j , questi fissano la posizione del primo elemento nel quadrato.
2. All'intero di questi cicli ve ne sono altri due sugli indici k e l in modo che siano presi in considerazione solo gli elementi che si trovano dopo l'elemento in posizione ij .
3. Si effettuano i controlli esposti precedentemente in merito alle rettificazioni locali sulle coppie che si possono formare con questi quattro indici.
4. Se le condizioni sono verificate si fanno gli scambi con la funzione `Replace[]`.

In questo modo si continua a percorrere il quadrato finché tutte le possibilità non sono state esplorate. Inoltre, nel caso delle rettificazioni ad una coppia è possibile evitare un ciclo perché le gli elementi da confrontare sono solo quelli sulla medesima riga o colonna.

Infine, per implementare i metodi di selezione (λ, μ) e $(\lambda + \mu)$ si utilizza il metodo `fittestChild[population]` sulla lista dei figli nel primo caso, o sulla lista dei figli a cui è stato aggiunto il genitore con `Append[]` nel secondo.

L'intero algoritmo è esposto nel diagramma di flusso riportato in figura 7, nel quale sono illustrati i criteri utilizzati per decidere quando effettuare rettificazioni o meno. Il metodo che esegue l'intero algoritmo si chiama `xiekang[order]`, e utilizza i comandi di Mathematica 8 per parallelizzare le operazioni: con `ParallelMap[]` si distribuiscono le varie funzioni da eseguire sui singoli individui, quali mutazioni e rettificazioni tra i 4 core a disposizione della macchina utilizzata. Oltre a far ciò `xiekang[]` raccoglie statistiche riguardo all'esecuzione dell'algoritmo memorizzando tempi di esecuzione, numero di generazioni, e fitness minima, questi dati costituiscono l'output del programma.

Una funzione `workout[]` esegue dieci volte `xiekang[]` per differenti ordini in modo da produrre la statistica in merito all'algoritmo esposta nella sezione successiva.

IV. RISULTATI

i. Algoritmi genetici

Gli algoritmi genetici implementati non sono stati in grado di produrre i risultati sperati: nessuna combinazione di fitness, criterio di selezione e di crossover è riuscita a produrre quadrati di dimensioni superiori al 4×4 , e nella maggior parte dei casi nemmeno di dimensioni superiori al 3×3 . I risultati ottenuti sono riportati nel dettaglio in tabella 4a in appendice. Nemmeno giocando sugli ulteriori gradi di libertà a disposizione, cioè cambiando la dimensione della popolazione, e le probabilità di mutazione e di crossover, si riesce ad ottenere un'implementazione sempre convergente di un algoritmo genetico.

Oltre a non essere convergenti, gli algoritmi genetici si sono rivelati computazionalmente inefficienti, in quanto le popolazioni con cui si lavora sono composte da molti quadrati su cui bisogna operare massicciamente ad ogni generazione.

Uno dei motivi principale del fallimento di questa tecnica è che il crossover, che normalmente serve a raggiungere la soluzione, in que-

sto caso risulta dannoso. Ciò accade perché la prescrizione che il quadrato abbia tutti numeri diversi impedisce al crossover di produrre un individuo migliore a partire dai quadrati genitori. A causa delle ricostruzioni del quadrato dopo la riproduzione, questi algoritmi genetici con crossover sono quasi equivalenti alla ricerca casuale. L'idea di base di un algoritmo genetico per cui gli individui possano scambiarsi dei geni buoni non si applica in questo caso, in cui una linea che è magica per un quadrato potrebbe non esserlo per un altro se contiene degli elementi doppi e deve quindi essere corretta. Per questo motivo la riproduzione tramite crossover non fa avvicinare alla soluzione, ma al contrario mantiene la popolazione ad una certa distanza dai quadrati magici, a causa del fatto che tutti i quadrati che fanno crossover sono sempre parzialmente generati a caso. Tuttavia anche ponendo $P_C = 0$ gli algoritmi genetici non convergono in quanto i loro metodi di mutazione sono ben più primitivi rispetto a quelli implementati nell'algoritmo evolutivo.

Il motivo di fondo per cui gli algoritmi genetici non convergono è che non si può trovare una funzione di fitness che individui un percorso verso l'individuo perfetto. E' possibile costruire un quadrato che è perfetto a meno di due sole permutazioni ma che ha fino a otto linee non magiche, e viceversa è possibile costruire un quadrato che ha tutte le linee magiche a meno di due, eppure che sia molto distante in termini di permutazioni da un quadrato magico.

ii. Algoritmo evolutivo

Al contrario degli algoritmi genetici, l'algoritmo evolutivo ha prodotto con successo un discreto numero di quadrati magici. Sono stati costruiti quadrati di ordine dal 10 a 40, e in nessun caso l'algoritmo si è trovato in una condizione metastabile e non è giunto a convergenza (per un totale di più di cento esecuzioni). Curiosamente, invece, per quadrati di dimensioni piccole (da 4×4 a 6×6) può capitare che il programma rimanga in una condi-

zione metastabile e quindi non sia in grado di convergere.

Si sono raccolte delle statistiche per caratterizzare come varia il tempo di esecuzione del programma al variare della dimensione dell'input e per confrontare i dati raccolti con quelli di Xie e Kang.

Nelle tabelle 2a e 2b sono riportati i dati raccolti, mentre nella figura 4 sono rappresentati i tempi di esecuzione al variare dell'ordine del quadrato con un possibile andamento interpolante.

Confrontando i dati ottenuti con quelli di Xie e Kang si nota che l'algoritmo implementato in questo lavoro si comporta decisamente peggio in termini di legge di scala rispetto all'originale. Si nota tuttavia che il numero di generazioni necessarie per raggiungere il quadrato magico è sempre notevolmente minore (circa inferiore del 50%) rispetto a quelle richieste di Xie e Kang: questo è dovuto al fatto che in questa implementazione ogni generazione è composta da una popolazione di 25 individui, mentre nell'originale di Xie e Kang sono solo 10. Si è adoperata questa scelta perché si è verificato che per i quadrati 10×10 la convergenza avviene circa il 20% più velocemente. E' tuttavia possibile che per ordini superiori questa scelta sia dannosa e abbia causato questo bias nei tempi.

La dipendenza della dimensione della popolazione inoltre incide anche per un ulteriore motivo: le mutazioni e le rettificazioni sono eseguite in parallelo su tutti i core a disposizione. Nella macchina usata venivano in effetti istanziati quattro thread, e per questo motivo le popolazioni che non sono potenze di 4 non sono ottimali.

Inoltre si nota che Xie e Kang hanno trovato che la legge di scala è quasi perfettamente come N^4 (con i dati a disposizione il corretto esponente di una power law interpolante è 4.04), il quale è il limite inferiore ottenuto da considerazioni teoriche nella implementazione realizzata in questo lavoro.

I dati raccolti si accordano con una legge di potenza avente per esponente un valore compreso tra 5 e 6. Nel caso di una popolazione

di 25 elementi questo, e con i dati a disposizione questo esponente vale ~ 5.34 con un coefficiente di determinazione $R^2 = 0.989$, mentre nel caso della popolazione da 10 individui con esponente ~ 5.04 e coefficiente di determinazione $R^2 = 0.997$.

Si nota inoltre che rimane una proporzione tra il numero di generazioni necessarie per trovare il quadrato magico nel caso a 25 e a 10 individui: tale rapporto decresce con l'ordine del quadrato e varia tra il 50% e il 20%. Questo mostra come non sia proporzionale al rapporto tra gli individui dei due casi, che è esattamente il 40%.

Anche se si è lontani dal risultato di Xie e Kang, questo è un miglioramento sostanziale rispetto alla ricerca *brute-force*.

I fit sono realizzati con il metodo dei minimi quadrati.

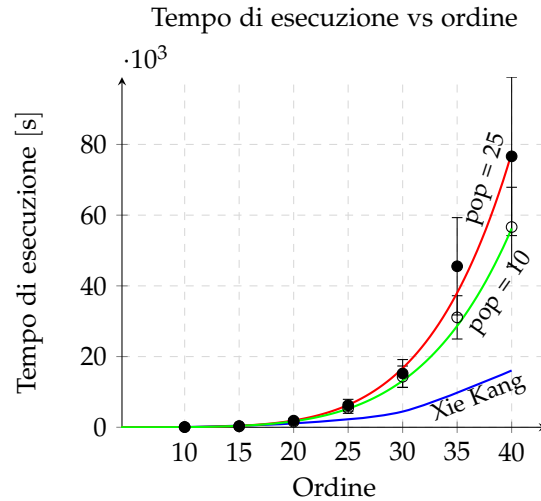


Figura 4: Tempi di computazione medi su dieci esecuzioni dell'algoritmo evolutivo per quadrati di ordine dal 10 al 30. La curva rossa descrive una possibile legge di scala: è una legge di potenza αx^β con esponente β compreso tra 5 e 6. Le barre di errore sono le deviazioni standard delle distribuzioni dei dieci risultati effettuati per ogni ordine. In blu sono i dati di Xie e Kang, rinormalizzati sul tempo di computazione. I fit sono realizzati con il metodo dei minimi quadrati.

V. CONCLUSIONI

i. Confronto con i risultati di Xie e Kang

L'attuale implementazione è quindi adatta per la costruzioni di quadrati di ordine relativamente piccolo.

Anche questa implementazione individua tre fasi nella ricerca del quadrato magico:

1. Durante la prima fase la fitness decresce vertiginosamente mentre il numero di linee non magiche rimase costante: nessuna linea ha la somma corretta.
2. Quando la fitness è sufficientemente bassa (inferiore a $50 \times N$) cominciano ad essere utilizzati i meccanismi di rettificazione locale lineare: questo aggiusta molte linee, ma non abbassa sensibilmente la funzione di fitness.
3. Infine si riesce a produrre un quadrato semimagico, in questa fase si utilizzano le permutazioni lineari e le rettificazioni per esplorare tutto lo spazio delle soluzioni, come mostra il grafico 6a, in cui le brusche oscillazioni di fitness indicano che molte soluzioni differenti sono state tentata.

L'indagine sui tempi di esecuzione delle varie subroutine dell'algoritmo mostra come la quasi totalità del tempo sia spesa nel cercare di rettificare le linee del quadrato, in particolare nella rettificazione a due coppie. Ci si attendeva questo comportamento in quanto l'implementazione ha richiesto l'utilizzo di quattro cicli innestati per controllare tutte le coppie di elementi del quadrato. Questo significa che per un quadro $N \times N$ in questa fase vengono eseguiti circa N^4 , che

Le rettificazioni ad una coppia invece effettuano $2(N-1)N^2 \sim N^3$ controlli ogni volta, e quindi per quadrati grandi influiscono sensibilmente meno sul tempo di elaborazione. Per questo non si sono implementate rettificazioni che coinvolgono un numero maggiore di due di coppie.

N	n_{tent}	n_{ok}	τ	τ/τ_0	n_{gen}
3	10	10	0.12 s	0	3
10	10	10	55 s	1	178
15	10	10	5.75 min	6	536
20	10	10	31.2 min	33	1085
25	10	10	1.73 h	112	1893
30	10	10	4.23 h	275	3007
35	10	10	12.38 h	824	4359
40	10	10	25.79 h	1681	7087

(a) Risultati con popolazione di 25 figli.

N	n_{tent}	n_{ok}	τ	τ/τ_0	n_{gen}
3	10	10	0.22 s	0	4
10	10	10	69 s	1	377
15	10	10	4.37 min	3	1109
20	10	10	26.3 min	22	2852
25	10	10	1.51 h	78	4855
30	10	10	3.97 h	206	7949
35	10	10	8.63 h	447	20725
40	10	10	15.73 h	820	46532

(b) Risultati con popolazione di 10 figli.

Tabella 2: Risultati ottenuti dall'algoritmo evolutivo: N è l'ordine del quadrato, n_{tent} il numero di esecuzioni dell'algoritmo, n_{ok} il numero di quadrati costruiti con successo entro 60 000 generazioni, τ il tempo medio di costruzione, τ/τ_0 è il rapporto tra il tempo medio con il tempo medio per la costruzione del quadrato con $N = 10$, n_{gen} è il numero medio di generazioni.

Tabella 3: Tempi di esecuzioni medi delle varie routine dell'algoritmo per quadrato di ordine 20. Siccome `rectifyLines` ha complessità maggiore degli altri metodi, quindi aumentando l'ordine tende a occupare tutto il tempo di elaborazione. Il restante tempo è speso nell'eseguire operazioni per produrre le statistiche. L'ultima riga specifica come si suddivide il tempo di esecuzione della routine `rectifyLines` nelle due subroutine per la rettificazione a una coppia o a due coppie. Questi dati sono nel caso della popolazione di 25 elementi, ma i risultati non sono sostanzialmente differenti se si considera la popolazione da 10 individui.

Routine		Tempo speso (%)
<code>selectFittest</code>		~ 0.01
<code>mutate</code>		~ 0.7
<code>rectifyDiagonals</code>		~ 7.7
<code>rectifyLines</code>		~ 91
/ \		/ \
<code>onePair</code>	<code>twoPairs</code>	~ 13 ~ 87

Bisogna inoltre precisare che il lavoro originale di Xie e Kang omette numerosi dettagli riguardo l'implementazione del loro algoritmo, limitandosi ad una descrizione sufficientemente accurata solo in merito alle mutazioni e alle rettificazioni. In particolare non è noto il linguaggio di programmazione con cui è stato realizzato l'algoritmo originale, né nessun altro dettaglio tecnico, come il grado di parallelizzazione dell'implementazione, per questo motivo il confronto non è perfettamente.

ii. Limiti degli algoritmi stocastici e possibili sviluppi futuri

In questo lavoro sono stati implementati e analizzati due possibili algoritmi stocastici per la costruzioni di quadrati magici, uno genetico e uno evolutivo. Il primo di questi due si è rivelato non essere in grado di costruire quadrati magici di ordini superiori a 3 in quanto non è possibile definire una funzione di fitness con la quale i quadrati non magici si avvicinino progressivamente a un soluzione magica. Al contrario, l'attuale implementazione

dell'algoritmo evolutivo riesce sempre a produrre quadrati magici se l'ordine è superiore a 7. Il prezzo da pagare per questo successo è un tempo di elaborazione piuttosto elevato, in particolare per costruire quadrati di grandi dimensioni, come si nota dalla tabella 2a. Per questo motivo è impensabile utilizzare questo algoritmo evolutivo per costruire quadrati di ordine superiore al 50. Si è tuttavia trovato che i tempi non scalano come nel lavoro originale di Xie e Kang, e quindi è lecito pensare che alcuni metodi sia ottimizzabili per rendere l'algoritmo più vicino all'originale, di cui tuttavia non si conosce alcun dettaglio tecnico riguardo l'implementazione.

Quando si esegue questa correzione tutto il quadrato viene passato in rassegna, per far ciò si utilizzano cicli innestati: uno che fissa il primo elemento, e il secondo che fa scorrere l'intero quadrato in cerca di numeri che soddisfino le condizioni richieste. Per questo motivo in generale il numero di cicli innestati è quattro, due per gli indici del primo elemento, e due per gli indici del secondo elemento, per questo motivo la complessità della rettificazione è sicuramente maggiore a $\mathcal{O}(N^4)$.

Si è studiato come viene speso il tempo di elaborazione all'interno dell'algoritmo evolutivo e si è trovato che la quasi totalità di questo è spesa nel cercare di rettificare i quadrati. Il motivo di questo comportamento è dovuto al fatto che le rettificazioni devono passare in rassegna l'intero quadrato più volte alla ricerca di determinate condizioni, e quindi l'implementazione contiene numerosi cicli innestati, che la rendono molto pesante dal punto di vista del tempo di esecuzione. Inoltre, così come è stato stati implementati, questi metodi non può essere parallelizzata in maniera diretta. Questo perché la matrice su cui si lavora si modifica durante l'esecuzione del metodo se si vengono a trovare condizioni favorevoli, e per questo non è possibile suddividere i controlli da effettuare tra più unità di elaborazione. Tuttavia è possibile modificare la funzione di rettificazione in modo da renderla parallelizzabile con poco sforzo: al posto di cercare di fare

quanti più correzioni possibili se ne fa uno solo, e quando questo avviene il metodo termina la sua esecuzione. In questo modo sono sicuramente necessarie più generazioni per giungere a un quadrato magico, ma diventa possibile implementare la funzione in modo altamente parallelo: ad ogni thread è dato un sottoinsieme di tutti i controlli che bisogna effettuare, quando uno di questi risulta positivo la routine termina e si effettua lo scambio, in questo modo si evita che se in un thread la condizione è verificata e si effettua uno scambio gli altri thread lavorino con il quadrato nello stato precedente. Questa parallelizzazione porterebbe notevoli vantaggi in termini di tempo di computazione della singola generazione, ma non si può dire a priori se nel complesso sia una miglioria o un peggioramento.

Un secondo modo per ridurre sensibilmente il tempo di elaborazione mantenendo le attuali implementazioni può essere quello di compilare tutti i metodi del programma con la funzione `Compile[]`. Questo aumenta sicuramente di molto l'efficienza del programma in quanto tutte le manipolazioni che è necessario fare sono puramente numeriche. Non è stato fatto in questo lavoro perché `Compile[]` non si comporta bene rispetto alla funzione `Return[]`: `Compile[]` percorre sempre tutta la funzione ignorando eventuali `Return[]` e restituisce sempre l'ultimo risultato (quello che precede la virgola). A causa di ciò alcuni metodi implementati andrebbero sistemati per essere compilati. Nella versione utilizzata di Mathematica eseguire:

```
Compile[{{x}}, Return[x]; 0.0]
```

produce una funzione che restituisce sempre il valore 0.0 e non x .

Il fatto che sia possibile in linea teorica compilare l'intero algoritmo di Xie e Kang con Mathematica mostra come effettivamente non sia necessario tale linguaggio ai fini dell'implementazione. Il motivo principale per cui Mathematica non è essenziale ai fini dell'implementazione dell'algoritmo è che nessuna manipolazione simbolica è strettamente necessaria. Sebbene Mathematica non sia necessario,

alcune funzioni native, come `Map[]`, `Table[]`, `Replace[]` sono state utilizzate numerose volte e si sono rivelate strumenti di grande aiuto per la semplificazione del codice. In particolare, la gestione delle liste di Mathematica ha permesso di scrivere in maniera concisa ed elegante tutti i metodi che si applicano alla intere popolazioni intere. Tuttavia, il cuore dell'algoritmo, cioè i metodi di rettificazione, richiedono che si percorra l'intero quadrato più volte lavorando esplicitamente con gli indici, e questo costituisce un grosso limite per Mathematica, il quale non gestisce in modo sufficientemente le risorse per questi cicli in maniera altrettanto efficiente rispetto a linguaggi compilati come il C.

Un aspetto che bisognerebbe indagare ulteriormente è la dipendenza del tempo di esecuzione della dimensione della popolazione, che è sostanzialmente l'unico parametro libero dell'algoritmo. Tuttavia, eseguire dieci costruzioni di quadrati dal 3×3 al 40×40 richiede circa due settimane di computazione, e per questo motivo questa correlazione non è stata studiata nei dettagli. Si è comunque mostrato che questo parametro ha un impatto non trascurabile e per questo si potrebbe rendere ulteriormente l'algoritmo più veloce trovando il parametro ottimale.

A differenza degli algoritmi deterministici elencati all'inizio di questo lavoro, l'algoritmo evolutivo può essere generalizzabile fin tanto che sono generalizzabili le condizioni da imporre sulle rettificazioni. Alcune possibili estensioni che è possibile fare in maniera abbastanza diretta sono quelle in cui la costante magica è fissata, nel caso questa sia lasciata libera (ad esempio per la ricerca di quadrati bimagici non vincolati) non è possibile adattare le funzioni di rettificazione e quindi questo algoritmo non è più adattabile.

VI. APPENDICE

i. Calcolo del numero magico per un quadrato magico normale

Dato un quadrato magico normale (cioè che contiene tutti i numeri da 1 a N) allora ciascuna linea e le diagonali hanno per somma il valore magico m , questo significa che sommando tutti i numeri del quadrato si ottiene Nm che è la somma di tutte le somme di ogni riga. Quindi:

$$Nm = \sum_{n=1}^{N^2} n$$

Ma la somma al membro destro può essere valutata esplicitamente:

$$\sum_{n=1}^{N^2} n = \frac{1}{2}N^2(N^2 + 1)$$

Da cui utilizzando la relazione precedente:

$$m = \frac{1}{2}N(N^2 + 1)$$

ii. Esempio di rettificazione lineare

Si riporta qui un esempio di rettificazione locale lineare: con una permutazione si passa da un quadrato con una sola riga magica, ad uno con tutte le righe magiche.

1	5	6	1	5	9
4	3	8	4	3	8
2	7	9	2	7	6
(a) Prima			(b) Dopo		

Figura 5: Esempio di rettificazione lineare. Dopo la rettificazione la prima è l'ultima riga diventano magiche.

N	F,C	n_{tent}	n_{ok}
3	tS, fP, E	10	7
3	tS, fP	10	4
3	tA, fP, E	10	8
3	cL, F, E	10	6
3	tS, sS, E	10	1
4	tS, fP, E	10	1
4	tS, F, E	10	0
4	tA, fP	10	0
4	tA, sS	10	0
4	cL, fP	10	0
10	cL, fP, E	10	0
10	tS, fP, E	10	0
10	tS, F, E	10	0

(a) Numero di successi ottenuti dagli algoritmi genetici. Le abbreviazioni utilizzate sono riportate nella tabella sottostante (4b).

Sigla	Metodo
fP	fitnessProportionate
sS	similarSquare
F	fittests
E	elitism
tS	totalSquared
tA	totalAbs
cL	correctLines

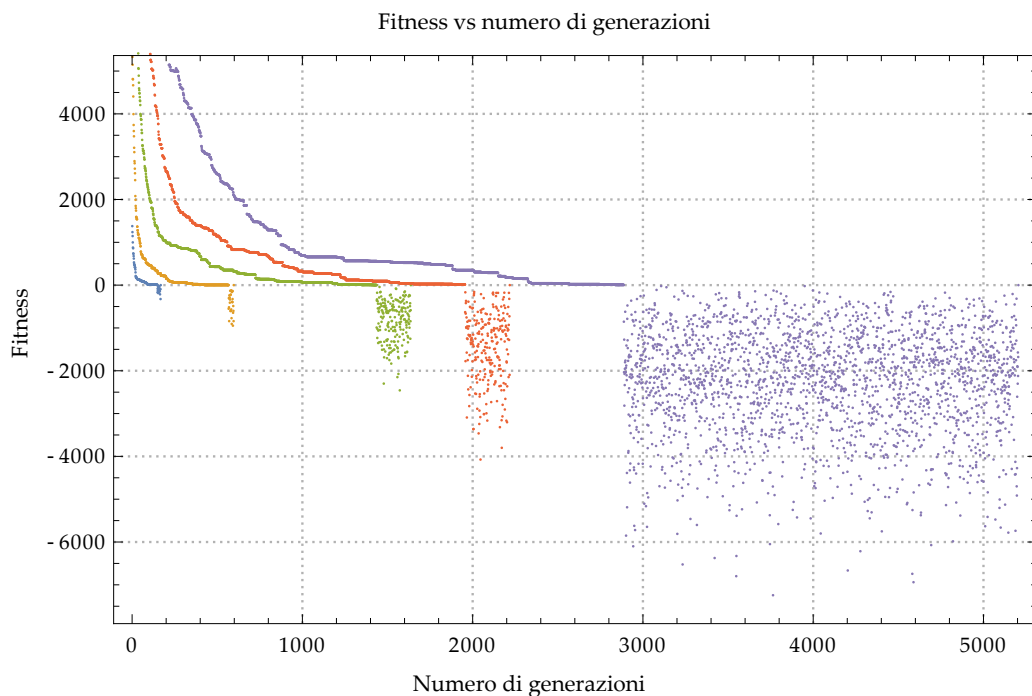
(b) Abbreviazioni utilizzate nella tabella sopra riportata (4a).

Tabella 4: Risultati ottenuti degli algoritmi genetici: N è l'ordine del quadrato, F, C la funzione di fitness e il criterio di selezione, n_{tent} il numero di esecuzioni dell'algoritmo, n_{ok} il numero di quadrati costruiti con successo entro 10 000 generazioni

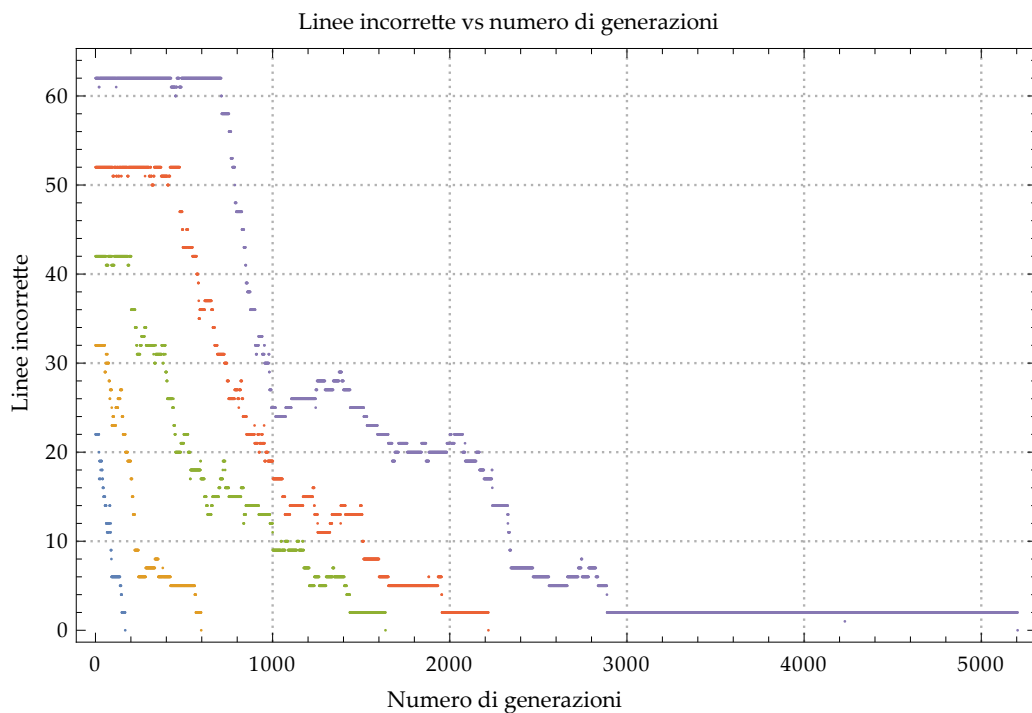
iii. Risultati degli algoritmi genetici

RIFERIMENTI BIBLIOGRAFICI

- [1] Abe, G. (1994). Unsolved Problems on Magic Squares. *Discrete Mathematics* (1993) 127 (13).
- [2] Ahmed, K. e Ender Ö. (2014). Constructing Constrained-Version of Magic Squares Using Selection Hyper-heuristics. *The Computer Journal* (2014) 57 (3).
- [3] Boyer, C. <http://www.multimagie.com>.
- [4] Kraitchik, M. (1942). Mathematical Recreations. Norton, New York.
- [5] Loly, P. (2003). Scientific Studies of Magic Squares. <http://home.cc.umanitoba.ca/~loly/IHPST.pdf>.
- [6] Mitchell, M. (1996). An Introduction to Genetic Algorithms. MIT Press, Cambridge.
- [7] Saha, B. e Bhattacharya, S (2015). An Approach To Hiding Image Into Video Using Magic Square *International Conference on Computer Science and Engineering, 2012*.
- [8] Tomba, I. e Shibiraj, N (2014). Successful Implementation of the Hill and Magic Square Ciphers: A New Direction. *International Journal of Advanced Computer Technology*.
- [9] Trump, W. (2015). <http://www.trump.de/magic-squares/estimates/index.html>.
- [10] Xie, T. e Kang, L. (2003). An Evolutionary Algorithm for Magic Squares. *The 2003 Congress on Evolutionary Computation, 2003*.



(a) *Fitness vs generazione al variare dell'ordine dei quadrati. Dove la fitness è negativa il quadrato è semimagico e si fanno rettificazioni locali diagonali. In questa fase non è rappresentato il nuovo genitore, ma il figlio migliore. Questo mostra come sia esplorato lo spazio delle soluzioni.*



(b) *Linee non magiche vs generazione al variare dell'ordine dei quadrati.*

Figura 6: Statistiche al variare dell'ordine del quadrato: blu 10, giallo 15, verde 20, rosso 25, viola 30.

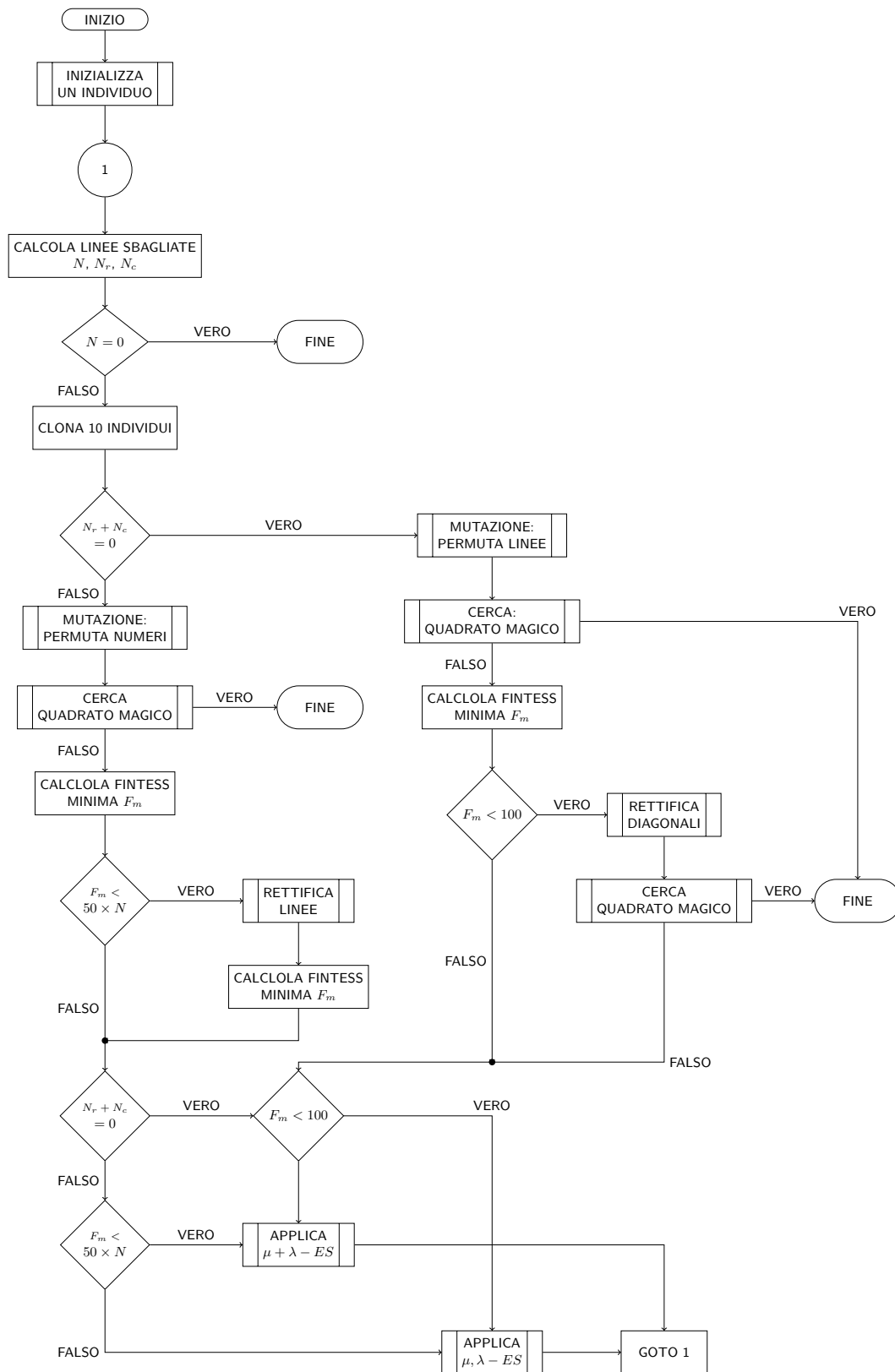


Figura 7: Diagramma di flusso dell'algoritmo di Xie e Kang.