# Studi esplorativi sul problema del commesso viaggiatore tramite algoritmo genetico

Carlo Zollo

14 giugno 2022

# Indice

# Introduzione

## Problema del commesso viaggiatore

Il problema del commesso viaggiatore (TSP, acronimo dall'inglese *Travelling Salesman Problem*) è rappresentativo della classe di problemi di ottimizzazione combinatoria. In questo lavoro lo schematizziamo così: un reticolo quadrato di lato $L$ ha $N$ siti[1] occupati (da qui, equivalentemente, città); la condizione è di partire da uno di questi e attraversare tutti gli altri esattamente una volta, poi tornare a quello iniziale. La richiesta è di scegliere il percorso che minimizza la distanza totale. Delle diverse interpretazioni, useremo quella simmetrica[2] e con distanza euclidea. Uno dei modi naturali per descrivere un percorso vaildo è assegnargli una stringa contenente l'etichetta dei siti da attraversare, nell'ordine in cui vengono attraversati1.

Risolvere esattamente il TSP è molto costoso in termini di risorse computazionali[3], ma ha diverse applicazioni; per questo sono stati proposti diversi algoritmi, euristici ed esatti. Uno degli approcci ideali è tramite algoritmi genetici, che hanno il vantaggio di riuscire a sondare *bene* i molti minimi locali presenti in questa classe di problemi.

## Algoritmi genetici

Gli algoritmi genetici (GA) sono una classe di algoritmi stocastici basati sul principio di evoluzione: in natura gli individui più adatti hanno maggiori chance di riprodursi, quindi ogni generazione dovrebbe essere in media *migliore* di quella precedente. Lo schema base di questi algoritmi è il seguente:

- Generazione di $N_{pop}$ individui e calcolo della loro *fitness*;

[1] etichettati, uno per ogni città

[2] la distanza tra due città è la stessa sia che si parta da una che dall'altra.

[3] una stima grezza dell'andamento temporale si può avere contando tutti i possibili percorsi, $\approx N!$.

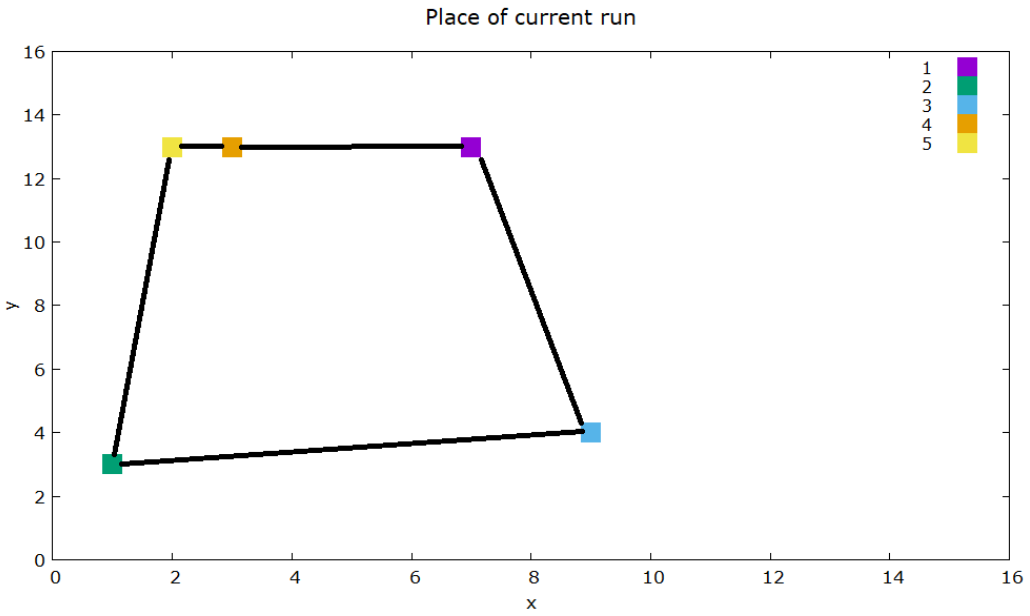


Figura 1: Reticolo quadrato di lato $L = 15$ contenente $N = 5$ siti occupati, o città. Il percorso più breve è quello in linea nera e viene schematizzato con $\langle 1\,3\,2\,5\,4\rangle$, o una qualsiasi permutazione che preservi l'ordine di visita.

- Un operatore di selezione agisce sulla popolazione, restituendo due individui in base alla loro *fitness*;
- Su questi agisce un operatore di incrocio, che restituisce due individui *figlio*, con una certa probabilità $P_C$;
- Sulla prole agisce un operatore di mutazione secondo una certa probabilità $P_M$;
- Quando l'operatore di selezione avrà agito $N_{pop}/2$ volte, la generazione iniziale verrà sostituita dai figli, che inizieranno un altro ciclo.
- L'algoritmo termina una volta soddisfatta una certa condizione, o dopo un numero massimo di generazioni $N_{gen}$.

Nella presentazione dello schema è stata introdotta un termine di fondamentale importanza, *fitness*: intendiamo una certa proprietà di un individuo che ne quantifica il grado di adattamento all'ambiente. Tra la terminologia tipica della biologia troviamo un nome più familiare: operatore; selezione, incrocio (o *crossover*), e mutazione sono il nucleo del processo[1], ciò che

distingue un algoritmo genetico da un altro. In questo lavoro ci concentriamo in particolare sul secondo, fondamentale per questa specifica classe di problemi.

## Approccio al problema

L'idea del lavoro si basa sull'articolo[2]: gli autori propongono un nuovo operatore di incrocio, CX2, confrontandone le soluzioni ottenute con quelle restituite dai più utilizzati PMX e OX. Questo progetto vuole replicare alcuni dei risultati ottenuti dagli autori, utilizzando alcune configurazioni dei TSP dalla libreria[3]; inoltre, come parte originale[4], vogliamo verificare la bontà del GA, modificando i parametri e la strategia interni, con diverse configurazioni generate casualmente e al variare di $N$. I risultati di questa seconda parte sono ottenuti confrontando le soluzioni del GA con quelle calcolate *brute-force*.

[4]oltre all'implementazione dell'intero algoritmo

# Implementazione algoritmo

## Adattamento dello schema base al TSP

Adattiamo la terminologia base degli algoritmi genetici al TSP. Gli individui saranno delle sequenze $\{S_p\}_p$ del tipo $S_p = \langle p_1\, p_2\, \ldots\, p_N \rangle$ con $p_i \in \mathbb{N}^+\ \forall\, i \in [1, N]$; sono valide se non contengono due volte lo stesso numero (nel codice chiamato impropriamente *bit*). Vogliamo minimizzare la distanza totale, allora scegliamo come *fitness*:

$$F(S_p) = \frac{L^2}{\sum_{i=1}^{N-1} d_{i,i+1}^{(S_p)} + d_{N,1}^{(S_p)}}, \tag{1}$$

dove $d_{i,j}^{(S_p)}$ indica la distanza della città $p_i$ da $p_j$ e il fattore $L^2$ è scelto[5] in modo da lavorare con numeri vicini all'unità. Siccome in questo lavoro il ruolo fondamentale è dell'operatore di *crossover*, abbiamo implementato dei modelli semplici per gli altri due; come operatore di selezione si è scelto la cosiddetta *roulette wheel*: a ogni sequenza si associa come probabilità la sua *fitness* divisa per la somma delle *fitness* di tutta la popolazione. Come operatore di mutazione abbiamo scelto la trasposizione (per la classe dei TSP) del cosiddetto *one-bit flip*: nella sequenza vengono randomicamente scelti due bit e scambiati di posto. Gli operatori di *crossover* sono PMX, *Partially Mixed crossover operator*, OX *Order crossover operator*, e CX2, *Cycle crossover operator 2*; per brevità di esposizione[6] aggiungiamo solo che l'idea di base dei primi due è di preservare delle porzioni, scelte randomicamente, delle sequenze *genitore*, mentre il CX2 è stato proposto per limitare la possibilità di avere sequenze *figlio* uguali a quelle *genitore*. Modifica fondamentale rispetto al GA proposto dagli autori, è l'implementazione della cosiddetta *elitism selection strategy*: una piccola percentuale, il parametro `elitism`, dei migliori individui di una generazione non viene sostituita dai peggiori individui della

[5] senza perdita di generalità.

[6] Maggiori informazioni nell'articolo[2]

successiva. Abbiamo deciso di agire in questo modo in seguito ai test preliminari per $N \gtrapprox 10$, che mostravano una crescita *piccola* della *fitness* media della popolazione.

## Struttura del codice

Per le simulazioni abbiamo usato il codice `GA_for_TSP.f90`[7], che descriviamo brevemente.

- Genera una configurazione casuale (o ne legge una) del TSP e da qui costruisce una matrice contenente le distanze tra le città[8];
- inizia la *run* vera e propria: $N_{sim}$ simulazioni (è importante sondare l'*evoluzione* a partire da diversi genitori) di $N_{gen}$ generazioni che si susseguono a partire da una generata casualmente. L'ordine è quello descritto nello schema base dei GA, e ogni simulazione è ripetuta per i 3 operatori di *crossover*;
- il codice tiene traccia della *fitness* media e della sua dispersione per ogni generazione, e soprattutto *fitness* e composizione della cosiddetta sequenza d'*elite*: l'individuo meglio adattato nel corso di tutta l'*evoluzione*, che sarà il risultato dell'algoritmo;
- Per finire, l'approccio *brute-force*: vengono calcolate tutte le permutazioni possibili delle $N$ città e valutate le *fitness* di ogni sequenza così prodotta.

[7] Consultabile alla repository `https://github.com/DeWitt46/HW_Fisica_Computazionale`.

[8] una sola volta, non è necessario calcolare le distanze ogni volta che serve la *fitness*

# Risultati e conclusioni

## Confronto tra operatori di incrocio

Vogliamo confrontare i risultati ottenuti dai 3 diversi operatori di *crossover*. Per tutti i problemi in questa sezione abbiamo fissato i parametri $P_C = 0.95$, $P_M = 0.1$, $N_{sim} = 30$, `elitism` $= 15\%$; Tab.1 mostra gli altri e le configurazioni nelle quali sono applicati[9]. Nella Tab.2 mostriamo i risultati ottenuti

| configurazione TSP | $N_{pop}$ | $N_{gen}$ |
|---|---|---|
| SEVEN ($N = 7$) | 30 | 10 |
| P_01 ($N = 15$) | 200 | 40 |
| GR_17 ($N = 17$) | 200 | 40 |
| FRI_26 ($N = 26$) | 400 | 40 |
| ATT_48 ($N = 48$) | 1200 | 34 |

Tabella 1: Parametri dell'algoritmo genetico utilizzati per i 3 operatori di *crossover* nelle diverse configurazioni del TSP.

dai confronti: il valore ottimale è preso dalla libreria, per *optimum* intendiamo il numero di volte che il miglior risultato si è presentato sul totale delle simulazioni; infine, la media è ottenuta dai migliori individui di ogni simulazione. Con l'implementazione della *elitism selection strategy* notiamo innanzitutto un generale incremento nelle prestazioni dell'algoritmo rispetto all'articolo[2], ma anche un disaccordo nel comportamento dell'operatore proposto da Hussain et al.; nel loro lavoro mostrano un lieve vantaggio nell'uso di CX2 per un numero contenuto di città, mentre per $N \gtrapprox 30$ c'è generale accordo con gli altri operatori, escluso due configurazioni per le quali il vantaggio è consistente. Guardando Tab.2 CX2 è in linea con PMX ed OX per $N \lessapprox 20$, ma per configurazioni con un numero maggiore di città si presenta un netto vantaggio nell'uso di OX2, che riesce a trovare anche il risultato

[9] disponibili nella libreria [3], mentre SEVEN è in [2].

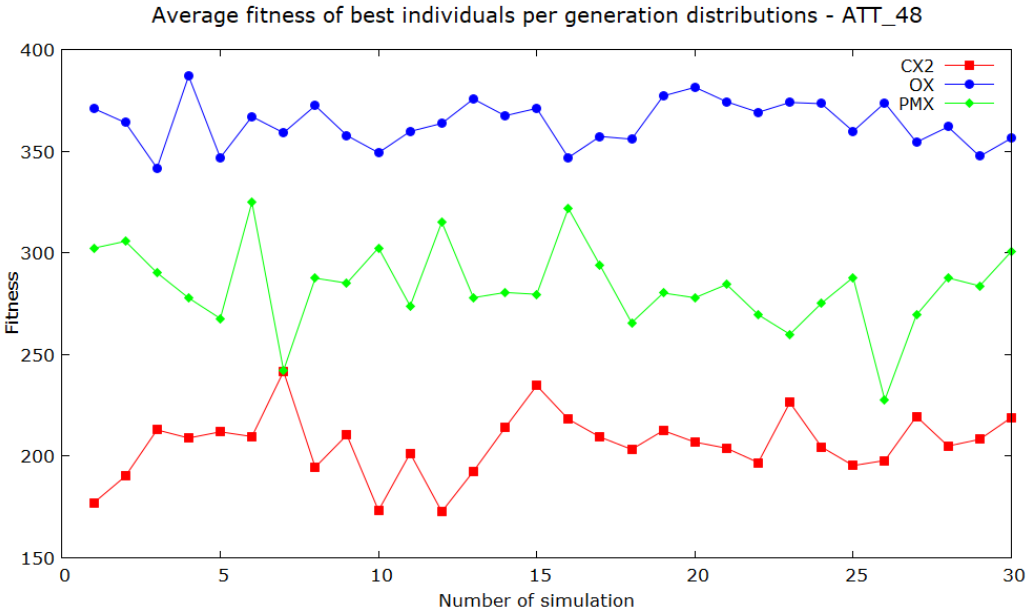


Figura 2: Distribuzioni della *fitness* media dei migliori individui tra tutte le generazioni, per ogni simulazione, a confronto per i 3 operatori. OX ha stabilmente una *fitness* media maggiore.

ottimale per la configurazione FRI_26. Una possibile motivazione per questo contrasto è da ricercare anche nella scelta dei parametri: date le nostre limitate risorse computazionali $N_{pop} \leq 40$[10].

Aggiungiamo qui che è risultato vantaggioso, per ATT_48, abbassare il numero degli individui per permettere di aumentare di molto $N_{gen}$ visto che, come ci si aspetta dalla classe dei GA, è in generale difficile trovare un plateau nella distribuzione della *fitness*3,4.

# Attendibilità generale dell'algoritmo

Vogliamo ora confrontare la bontà dell'algoritmo su diverse configurazioni randomiche, il cui valore ottimale viene calcolato *brute-force* verificando la *fitness* di tutte le possibili permutazioni delle città. Per ogni $N \leq 11$ generiamo 10 problemi: siamo interessati all'*optimum* e alla *fitness* media, in più vogliamo stimare il vantaggio in termini di risorse computazionali dell'algoritmo genetico. Siccome $N$ è *piccolo*, utilizziamo CX2 con i seguenti parametri: $P_C = 0.95$, $P_M = 0.1$, $N_{sim} = 30$, $N_{pop} = 30$, $N_{gen} = 30$.

[10] in termini di risorse incide di più rispetto a $N_{gen}$.

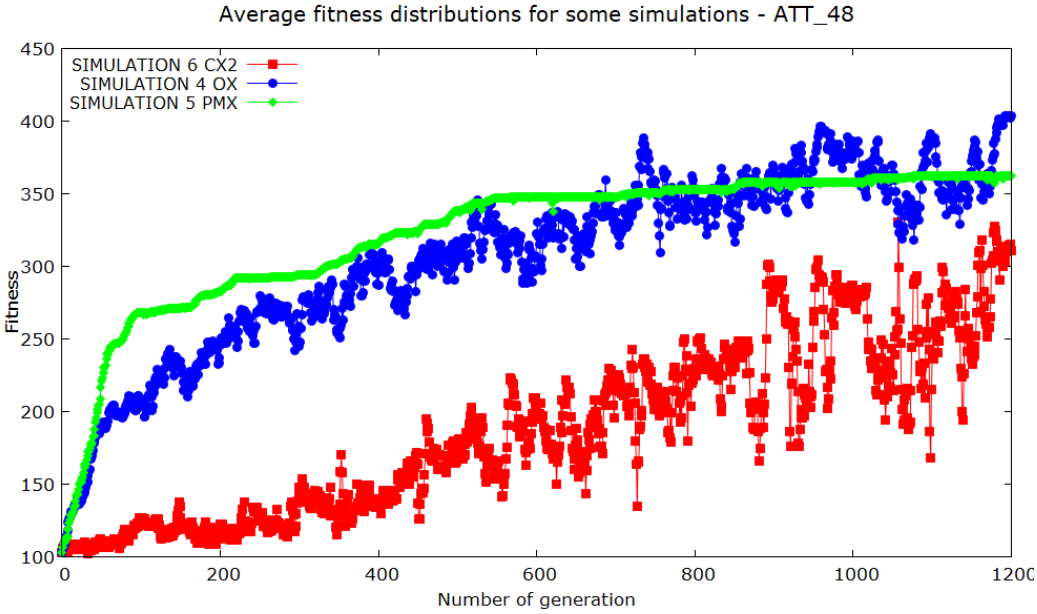


Figura 3: Distribuzioni della *fitness* media di tutti gli individui al variare della generazione, per la simulazione migliore, a confronto per i 3 operatori. Il comportamento medio di PMX è in linea con OX, ma quest'ultimo ha una evoluzione genuinamente con maggiore possiblità di *diversità*, che gli consente di avere individui d'*elite* con migliore adattamento.

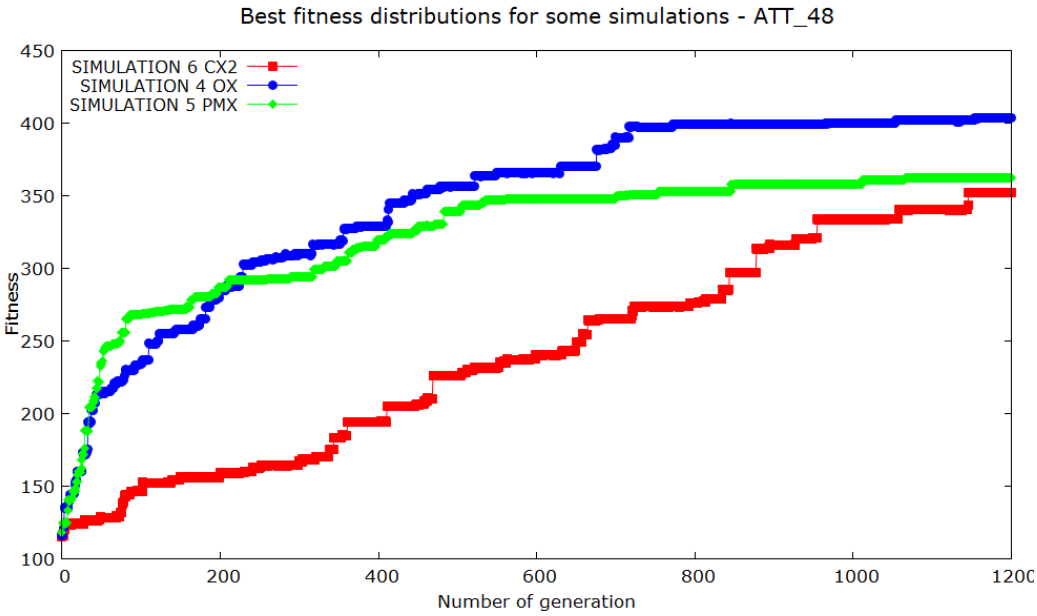


Figura 4: Distribuzioni della *fitness* del miglior individuo al variare della generazione, per la simulazione migliore, a confronto per i 3 operatori.

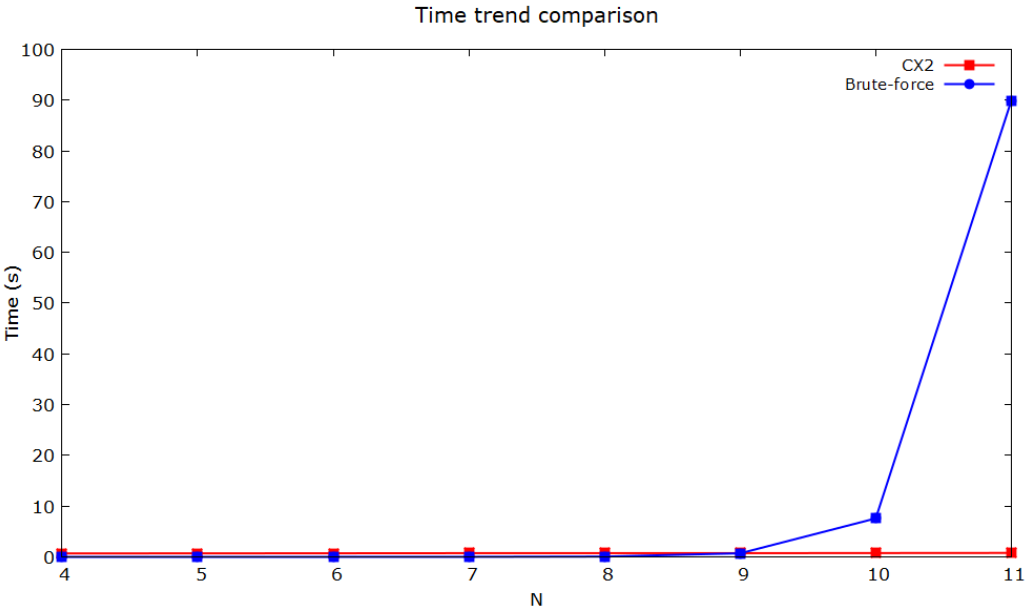


Figura 5: Andamento temporale, mediato su 10 configurazioni random, degli algoritmi genetico e *brute-force* a confronto.

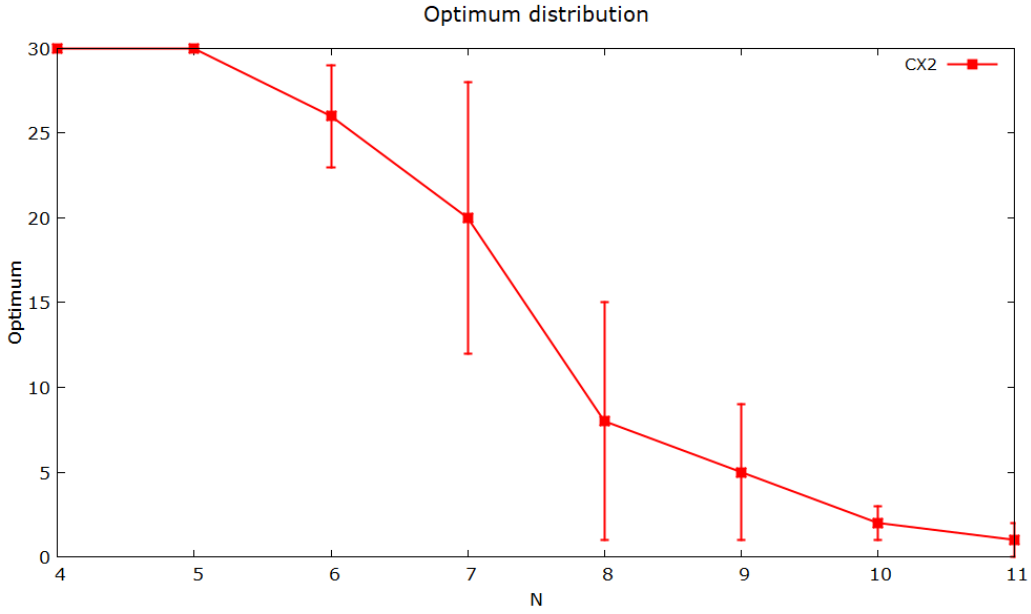


Figura 6: Distribuzione dell'*optimum*, mediato su 10 configurazioni random, dell'algoritmo genetico.

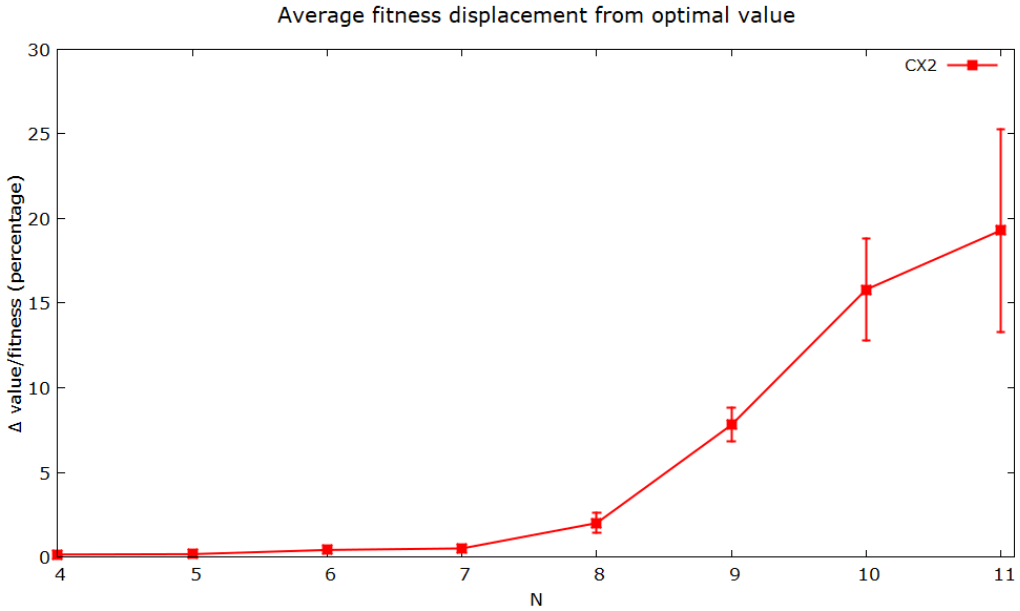


Figura 7: Andamento, mediato su 10 configurazioni, della differenza percentuale del valor medio della *fitness* dal valore ottimale.

| configurazione TSP | Valore ottimale | Risultato | PMX | OX | CX2 |
|---|---|---|---|---|---|
| SEVEN ($N = 7$) | 159 | Migliore | 159 | 159 | 159 |
| | | Optimum | 14/30 | 20/30 | 23/30 |
| | | Media | 160.2 | 160 | 160.4 |
| P_01 ($N = 15$) | 291 | Migliore | 291 | 291 | 291 |
| | | Optimum | 2/30 | 23/30 | 10/30 |
| | | Media | 317.4 | 296.7 | 340.7 |
| GR_17 ($N = 17$) | 2085 | Migliore | 2085 | 2085 | 2085 |
| | | Optimum | 2/30 | 2/30 | 3/30 |
| | | Media | 2175.9 | 2112.6 | 2179.8 |
| FRI_26 ($N = 26$) | 937 | Migliore | 1020 | 937 | 970 |
| | | Optimum | 1/30 | 2/30 | 2/30 |
| | | Media | 1161.5 | 1011.2 | 1312.8 |
| ATT_48 ($N = 48$) | 33523 | Migliore | 42105 | 36068 | 45070 |
| | | Optimum | 1/30 | 1/30 | 1/30 |
| | | Media | 49536.8 | 37647.1 | 52313.4 |

Tabella 2: Risultati dell'algoritmo genetico per 3 operatori di *crossover* a confronto su diverse configurazioni del TSP.

Fig.5 mostra l'andamento temporale: come ci si aspettava, il metodo *brute-force* scala come $N!$, mentre il costo di GA dipende, in prima approssimazione, solo da $N_{pop}$ e $N_{gen}$ ed è infatti costante. L'accuratezza generale dell'algoritmo genetico è in 6, 7; mostriamo solo l'andamento della differenza della *fitness* media dal valore ottimale e non della migliore perché per tutti i problemi[11] il GA ha trovato la sequenza migliore.

# Conclusioni

Come ci si aspettava, i GA sono efficaci nella ricerca del minimo per la classe dei TSP; per quanto l'algoritmo implementato sia stato in grado di trovare il valore ottimale di configurazioni anche a $N$ *non basso*, il lavoro non è esaustivo. In particolare, si può indagare sulla possibilità di un diverso operatore di selezione in modo da avere maggiore pressione nella scelta delle

[11]escluse due configurazioni per $N = 11$, dove infatti la dispersione è maggiore. È comunque possibile risolverle con il GA aumentando $N_{gen}$ a 50.

sequenze genitore; un'idea potrebbe essere l'implementazione di pesi (che tengano conto della dispersione della generazione) nella *roulette wheel*.

# Bibliografia

[1] Fu C., Zhang L., Wang X., et al. (2018). *Solving TSP problem with improved genetic algorithm.* AIP Conference Proceedings 1967, 040057: https://aip.scitation.org/doi/abs/10.1063/1.5039131

[2] Hussain A., Muhammad Y. S., Nauman Sajid M., Hussain, I., Mohamd Shoukry A., Gani S. (2017). *Genetic algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator.* Computational Intelligence and Neuroscience: https://doi.org/10.1155/2017/7430125

[3] TSP test data: https://people.sc.fsu.edu/~jburkardt/datasets/tsp/tsp.html