Alma Mater Studiorum · Università di Bologna

Dipartimento di Informatica - Scienza e Ingegneria (DISI)
Corso di Laurea Magistrale in Informatica

Collaborative Robbies

Relazione di FSC

18 dicembre 2015

Giulio Biagini
0000705715
giulio.biagini@studio.unibo.it

Daniele Baschieri
0000688992
daniele.baschieri@studio.unibo.it

Indice

1 Introduzione **5**
- 1.1 Robby: The Soda-Can-Collecting Robot . . . 5
- 1.2 Entità Intelligenti . . . 6
- 1.3 Agenti Intelligenti . . . 6
- 1.4 Agenti Razionali . . . 7
- 1.5 Agenti Reattivi Semplici . . . 7
- 1.6 Agenti in Grado di Apprendere . . . 7

2 Obiettivi **9**
- 2.1 Entità Invisibili - Vista a Croce . . . 9
- 2.2 Entità Invisibili - Vista Quadrata . . . 10
- 2.3 Entità Visibili - Vista a Croce . . . 10

3 Progettazione **11**
- 3.1 Algoritmi Genetici: schema generale . . . 11
- 3.2 Collaborazione . . . 12
- 3.3 Funzione di Fitness . . . 12
- 3.4 Selezione dei Genitori . . . 13
- 3.5 Crossover . . . 13
- 3.6 Mutazione Genetica . . . 13

4 Implementazione **15**
- 4.1 ANSI C ed MPI . . . 15
- 4.2 Installazione . . . 15
- 4.3 Compilazione . . . 16
- 4.4 Esecuzione . . . 16
- 4.5 Architettura . . . 16

5 Risultati **17**
- 5.1 Vista a Croce Non Collaborativa . . . 17
  - 5.1.1 100 Coppie . . . 17

5.1.2 100 Coppie, da 1 a 5 Vecchie 18
5.1.3 100 Coppie, da 10 a 35 Vecchie 18
5.1.4 100 Coppie, da 40 a 90 Vecchie 18
5.1.5 200 Coppie, da 10 a 180 Vecchie 22
5.1.6 200 Coppie, da 70 a 120 Vecchie Non Mutate 26
5.2 Vista Quadrata Non Collaborativa 28
5.2.1 200 Coppie, da 70 a 120 Vecchie Non Mutate 28
5.3 Vista a Croce Collaborativa 30
5.3.1 200 Coppie, da 70 a 120 Vecchie Non Mutate 30
5.4 Viste Non Collaborative a Confronto 30
5.4.1 40 Azioni 30
5.4.2 30 Azioni 30

6 Conclusioni **31**

Capitolo 1

Introduzione

In questo capitolo andremo a descrivere *Robby: The Soda-Can-Collecting Robot* [1], il progetto a cui questo lavoro si ispira, riportando alcuni cenni teorici di Intelligenza Artificiale che useremo per descrivere in maniera più formale la struttura dell'intero progetto.

1.1 Robby: The Soda-Can-Collecting Robot

Robby è un robot il cui compito consiste nel muoversi in uno spazio sporco e cercare di ripulirlo. Lo spazio può essere visto in modo astratto come una griglia suddivisa in celle. Robby possiede solamente una vista parziale dello spazio nel quale si trova: è in grado di vedere cosa è presente nella cella a nord, in quella a sinistra, nella casella in cui si trova, in quella a destra ed in quella a sud rispetto alla propria posizione. Le celle possono essere "pulite" (vuote), "sporche" (nelle quali è presente una lattina da raccogliere) oppure rappresentare un ostacolo (un muro).
Robby attua un'azione in base alla vista che ha in quel momento dell'ambiente. Se, ad esempio, vede una lattina nella cella a nord, un muro nella cella a sinistra e niente nella cella da lui occupata, in quella a destra e nella cella a sud, questo non si muoverà negli spazi vuoti, né tantomeno sbatterà contro il muro muovendosi a sinistra, ma si muoverà di un passo verso l'alto, in direzione della lattina, per poi raccoglierla. Le azioni che Robby può effettuare sono: la mossa verso nord, verso sinistra, verso destra e verso sud, può decidere di rimanere fermo, raccogliere una lattina oppure effettuare una mossa casuale scelta fra le precedenti. Tutte le mosse hanno l'effetto immaginato a parte la raccolta della lattina. Questa toglierà la lattina dalla mappa nella cella in cui il robot si trova, se presente, altrimenti lascerà l'ambiente invariato.

Inizialmente Robby possiede un dna casuale, ovvero, ad ogni vista è associata un'azione casuale. Robby è però in grado di apprendere come muoversi correttamente nell'ambiente e come raccogliere lattine. La sua evoluzione è infatti guidata da un Algoritmo Genetico.

1.2 Entità Intelligenti

Lo scopo dell'*Intelligenza Artificiale* è quello di cercare di capire come le *entità intelligenti* funzionano ed, in particolare, quello di cercare di costruire, creare, entità intelligenti.
Negli anni sono state date varie definizioni di entità intelligenti: alcune fanno paragoni con gli esseri umani, altre usano il principio della razionalità, alcune definiscono queste entità in base a come pensano, altre a come agiscono. Nel nostro particolare caso, andremo a definire un'entità come intelligente se *agisce in modo razionale*, ovvero se "fa la cosa giusta".
Affinché **Robby**, l'entità intelligente che abbiamo il compito di creare, possa essere definita intelligente, dovrà quindi "fare la cosa giusta". Intuitivamente, siccome il compito del robot sarà quello di muoversi in uno spazio cercando di pulirlo, "fare la cosa giusta" significherà, ad esempio, non sbattere contro i muri durante il proprio movimento, raccogliere le lattine durante il proprio passaggio e non tentare di raccogliere lattine dove queste non sono presenti.

1.3 Agenti Intelligenti

Un *agente intelligente* è un'entità intelligente in grado di percepire l'ambiente tramite dei *sensori* e compiere delle azioni tramite degli *attuatori*.
Un agente ha delle *percezioni* dell'ambiente, ovvero l'insieme di tutti gli input percettivi che provengono dai propri sensori in un dato istante.
Il comportamento di un agente intelligente è descritto matematicamente da una *funzione agente*, ovvero l'insieme di tutte le sequenze percettive e da tutte le relative azioni. L'implementazione di una funzione agente prende il nome di *programma agente*.
Analizzando la definizione di agente intelligente, possiamo notare come **Robby** sia un'entità che appartenga a questa categoria: esso infatti è in grado di percepire l'ambiente tramite dei sensori che gli permettono di vedere il contenuto delle celle che lo circondano e possiede degli attuatori che gli consentono di muoversi e raccogliere lattine.
L'insieme di tutte le viste (percezioni) che Robby può avere dell'ambiente e le rispettive azioni sono descritte da una funzione agente. Il nostro compi-

to sarà quello di fornirne un'implementazione attraverso la scrittura di un programma agente.

1.4 Agenti Razionali

Il nostro obiettivo, però, non è semplicemente quello di creare agenti intelligenti, ovvero entità in grado di percepire l'ambiente e compiere azioni, ma, piuttosto, creare *agenti razionali*, ovvero entità sì intelligenti, ma che "facciano la cosa giusta". Questo significa che il programma agente, per ogni sequenza percettiva, produce un'azione che va a massimizzare una determinata *misura di prestazione*: se le azioni attuate dall'agente portano l'ambiente ad attraversare una sequenza di stati che può essere definita "desiderabile", allora la misura di prestazione è massimizzata.
Nel caso di **Robby**, la misura di prestazione da massimizzare sarà sia il numero di lattine che debbono essere raccolte in un dato ambiente, sia il numero di passi impiegato per raccoglierle.

1.5 Agenti Reattivi Semplici

Esistono varie tipologie di programmi agente in base alle tipologie di agenti di cui questi hanno il compito di descrivere il comportamento. I più semplici di tutti sono gli *Agenti Reattivi Semplici*, ovvero agenti che basano le proprie azioni solamente sulla percezione corrente. Questi agenti, dunque, analizzano tutti gli input percettivi che provengono dai sensori in un dato istante e computano quale azione compiere.
Robby è un agente reattivo semplice in quanto la scelta dell'azione da attuare è guidata solamente dalla vista che ha in quel momento dell'ambiente.

1.6 Agenti in Grado di Apprendere

Esistono particolari tipologie di agenti che possono essere programmati in modo che siano *in grado di apprendere*.
Un agente, infatti, può calcolare la scelta delle proprie azioni basandosi su conoscenze pregresse che ha dell'ambiente, che sono ad esempio state inserite a priori da un programmatore: in questo caso si dice che l'agente manca di autonomia. Al contrario un *agente autonomo* è in grado di apprendere per compensare la presenza di conoscenza parziale o erronea. Un agente in grado di apprendere ha il vantaggio di poter operare in ambienti all'inizio sconosciuti, diventando col tempo via via più competente.

Questa tipologia di agenti possiedono, oltre ad un *elemento esecutivo*, che gli permette di selezionare le azioni da compiere, anche un *elemento di apprendimento*, responsabile del miglioramento interno, il quale usa le informazioni provenienti dall'*elemento critico* riguardo le prestazioni correnti dell'agente e determina se e come modificare l'elemento esecutivo affinché in futuro si comporti meglio. Infine, le entità in grado di apprendere possiedono un *generatore di problemi*, il cui scopo è quello di suggerire azioni che portino ad esperienze nuove e significative dalle quali apprendere.
Nel caso di **Robby**, quello che vogliamo è un agente autonomo in grado di apprendere, ovvero un agente che non si basi su conoscenza pregressa da noi inserita. Per fare questo ci appoggeremo sugli *Algoritmi Genetici*.

Capitolo 2

Obiettivi

In questo capitolo sono descritti gli obiettivi di questo lavoro. In particolare, quello che ci interessa non è tanto re-implementare il lavoro proposto da Melanie Mitchell [1], quanto piuttosto realizzare un sistema che ci permetta di studiare come, inserendo più entità nell'ambiente, queste si influenzino a vicenda.
Lo scopo principale, dunque, è analizzare se può esservi **collaborazione spontanea** o meno fra le entità che si trovano a nella stessa mappa.

2.1 Entità Invisibili - Vista a Croce

Come prima cosa andremo a cercare di stabilire quali sono le performance di due agenti che si muovono sulla mappa senza vedersi, ovvero ignorando la posizione dell'altro robot. Sarà dunque ammesso trovarsi nella stessa cella e raccogliere la stessa lattina.
I due robot avranno vista a croce, ovvero come quella descritta nel capitolo introduttivo: potranno vedere la casella a nord, quella a sinistra, quella nella quale si trovano, quella a destra e quella a sud.
I risultati così ottenuti saranno usati come "caso base" per confrontare le performance in relazione all'utilizzo di viste che permettono di percepire una porzione più grande della mappa e viste che permettono di vedere anche l'altro robot che su di essa si muove.
Chiameremo la vista che non permette di vedere gli altri Robby impegnati sulla mappa **"non collaborativa"** e la vista che permette di vedere gli altri robot **"collaborativa"**.

2.2 Entità Invisibili - Vista Quadrata

In questo secondo test, così come nel caso precedente, le due entità non saranno in grado di vedersi ma avranno una percezione maggiore della mappa nella quale si troveranno. Potranno infatti vedere uno spazio di 3x3 celle dove il robot occupa la posizione centrale (seconda riga e seconda colonna).
Questa prova ci permetterà di capire se potendo vedere una porzione di mappa maggiore, i robot saranno avvantaggiati nella pulizia della stessa, ovvero, se disponendo di più risorse (sensori più potenti) saranno in grado di massimizzare maggiormente la misura di prestazione: sia essa la raccolta di un numero maggiore di lattine, sia essa la raccolta dello stesso numero di lattine ma con un numero minore di passi.

2.3 Entità Visibili - Vista a Croce

Con quest'ultima simulazione vedremo cosa succede dando la possibilità ai due robot di vedersi, usando, cioè, viste *collaborative*. In questo caso non sarà tollerata la compresenza nella medesima cella.
Vedremo così se i due robot impareranno a collaborare o se invece la presenza di uno ostacolerà l'altro.

Capitolo 3

Progettazione

In questo capitolo parleremo di come è stato implementato l'algrotimo genetico che ha permesso l'apprendimento di una strategia di pulizia alle coppie impegnate sulla mappa.

3.1 Algoritmi Genetici: schema generale

Gli *Algoritmi Genetici* sono algoritmi spesso usati in tutte quelle situazioni nelle quali risulta difficile progettare una strategia che permetta di arrivare ad una soluzione. Essi sono difatti usati per far sì che siano loro stessi ad evolvere il sistema in modo che esso arrivi autonomamente ad una soluzione. Uno dei campi nei quali sono maggiormente impiegati, infatti, è quello dei *Sistemi Complessi*.
Per far sì che **Robby** possa muoversi e pulire l'ambiente raccogliendo il maggior numero di lattine con il minor numero di mosse, non conoscendo a priori la posizione del robot stesso nella mappa, né delle lattine, non rimane che progettare un algoritmo genetico che permetta ai robot di apprendere una buona strategia.
La struttura utilizzata nel paper di riferimento [1] è la seguente:

- dapprima sono generati *POPULATION_SIZE* (200) individui con un dna casuale, ovvero, ad ogni possibile vista è associata una mossa scelta a caso fra quelle possibili;

- dopodiché, per *NUM_GENERATIONS* (500) generazioni:

  - viene calcolato il *valore di fitness* per ogni individuo assegnando un punteggio 10 nel caso in cui un robot raccolga una lattina, -1 se il robot tenta di raccogliere una lattina in una cella dove questa

non sia presente e -5 punti se il robot tenta di muoversi in una casella occupata da un ostacolo (muro). Questa operazione viene effettuata per *NUM_ACTIONS_PER_SESSIONS* passi e ripetuta per *SESSIONS_NUMBER* diverse sessioni. Ogni sessione prevede la generazione di una mappa nella quale le lattine sono posizionate casualmente nelle celle (ogni cella ha la medesima probabilità di ospitarne una), così come la posizione del robot è scelta casualmente. La mappa ha dimensione 10x10 con 10 lattine. Alla fine, ogni robot ottiene un valore di fitness mediato sul numero di sessioni. Il numero di azioni su ogni mappa è fissato a 200, così come il numero di sessioni;

- si procede poi ordinando gli individui in base al proprio valore di fitness;
- sono scelti due individui nella popolazione con una probabilità che varia linearmente in base all'ordinamento (l'individuo con il maggiore valore di fitness avrà probabilità più alta di essere scelto rispetto al secondo, e così via...) e viene applicato il *crossover*. Questo procedimento genererà due nuovi figli ed è ripetuto fino alla completa sostituzione della vecchia popolazione;
- una volta ottenuta la nuova generazione, ogni gene (azione corrispondente ad una vista) può essere mutato con una probabilità pari a *MUTATION_PROBABILITY*: 0.5%.

3.2 Collaborazione

Siccome il nostro obiettivo è quello di studiare la collaborazione fra più robot, nella mappa saranno presenti contemporaneamente 2 agenti. Questo farà sì che la grandezza POPULATION_SIZE non indichi il numero di robot nella popolazione, il numero di coppie. Dunque, il numero di entità sarà pari al doppio. Fisseremo, per iniziare, questo numero a 100. Dunque, 100 coppie e 200 individui.

3.3 Funzione di Fitness

La presenza di due robot nella mappa comporta anche la modifica di come la funzione di fitness valuta gli individui. Il valore di fitness non riguarderà un singolo agente, ma la coppia: se alla vista del primo robot corrisponde l'azione di muoversi verso sinistra e la cella rispettiva è occupata da un muro (-5 punti), mentre l'azione del secondo robot corrisponde alla raccolta della

lattina in una posizione dove essa non è presente (-1 punto), il valore di fitness sarà aggiornato sottraendo 6 punti.
Nel caso in cui i due robot abbiano la possibilità di vedersi, sarà assegnato un valore negativo di -5 punti anche nel caso le due entità cerchino di occupare la stessa cella, scontrandosi, così come avviene nel caso in cui un robot tenti di muoversi contro un muro.
Nel caso in cui le due entità non si vedono ed entrambi occupano la stessa cella, nessun punto è sottratto. Se entrambi, poi, occupano una stessa cella nella quale è presente una lattina e cercano entrambi di raccoglierla, il secondo robot non prenderà punteggio negativo (-1) in quanto la lattina è già stata raccolta dal primo Robby. Non si avrà però neanche l'incremento del valore di fitness di 20 punti (10 per l'azione di raccolta del primo robot e 10 per l'azione di raccolta del secondo). In un caso simile, il valore sarà incrementato di soli 10 punti in quanto entrambi hanno fatto l'azione corretta ma solo una lattina è stata raccolta.

3.4 Selezione dei Genitori

La tecnica da noi usata per generare la nuova popolazione non è esattamente quella descritta nel paper. Difatti, a seconda delle varie simulazioni, saranno copiati alcuni individui dalla vecchia alla nuova generazione senza passare tramite il crossover. Questa tecnica è anche nota con il nome di *élitarismo*.

3.5 Crossover

La tecnica di crossover da noi usata è quella descritta nel paper, ovvero: è scelto un punto di taglio casuale (random cut point) nel dna dei due genitori per generare i figli.
Avendo però delle coppie la strategia cambia lievemente: sono scelte due coppie come genitori dove: il primo Robby della prima coppia ed il primo Robby della seconda generano il primo figlio, mentre, il secondo Robby della prima coppia ed il secondo della seconda coppia generano il secondo figlio. Per ognuno dei due figli, il cut point nel dna dei genitori è scelto casualmente, dunque, la probabilità che sia lo stesso è molto bassa.

3.6 Mutazione Genetica

La mutazione genetica è fatta nella maniera standard, mutando i geni dei due robot generati in accordo con una percentuale fissata.

Il dna di ogni individuo sarà dunque analizzato gene per gene dove, di volta in volta sarà generato un numero casuale in base al quale sarà applicta o meno la mutazione. Nel caso in cui l'azione selezionata debba essere mutata, ne sarà scelta casualmente una che sia però diversa da quella attuale.

Capitolo 4

Implementazione

In questo capitolo andremo a presentare gli strumenti utilizzati nell'implementazione del codice. Spiegheremo anche come compilare ed eseguire lo stesso, riportando come sono stati ottenuti i risultati delle simulazioni presentati nel capitolo successivo.

4.1 ANSI C ed MPI

Il codice è stato scritto in **ANSI C** utilizzando la libreria **MPI**, un'interfaccia per il *Message Passing* open source. Grazie ad essa l'eseguibile può essere inviato a più computer che ne eseguono un'istanza. Ad ogni elaboratore è asseganto un id che può essere sfruttato in modo da dare alla macchina un comportamento specifico.
Nell'architettura da noi realizzata questi id sono usati per assegnare un *seed* ad ogni istanza del programma in esecuzione su ogni computer: si parte con seed 10 e si sommano gli id. È così che il l'istanza del programma eseguita dall'elaboratore con id 0 avrà seed 10, l'istanza del programma eseguita dal computer con id 1 avrà seed 11 e così via...
Le simulazioni da noi eseguite sono state lanciate con 11 elaboratori.

4.2 Installazione

Il programma è stato implementato e fatto eseguire su di un'architettura con sistema operativo *Debian stable 8.4*. Per compilarlo ed eseguirlo usando MPI, è stato necessario installare il pacchetto *mpich*.

4.3 Compilazione

La compilazione del sorgente avviene tramite il comando *mpicc* a cui abbiamo passato i flag *–ansi*, *-Wall*, *-pedantic* e *-lm* per il linking della libreria matematica.

4.4 Esecuzione

L'esecuzione è lanciata tramite il comando *mpirun* specificando con il flag *-np* il numero di processi da lanciare: 11 nel nostro caso. Questo dato è molto importante al fine di ripetere le simulazioni da noi effettuate, in quanto ogni processo esegue un'istanza del programma con un seed diverso dagli altri, ovvero: *SEED + id*, dove SEED è pari a 10.

4.5 Architettura

Il processo con id 0 agisce da master mentre gli altri 10 processi, con id da 1 a 10 appunto, agiscono da slave. Il master genera la popolazione iniziale e, per ogni generazione, la divide in parti uguali ed invia ad ogni slave una parte di coppie che devono essere valutate. Se ad esempio abbiamo una popolazione formata da 200 coppie, ogni slave dovrà valutarne 20. Ogni slave valuta le coppie a lui assegnate e risponde al master con i relativi valori di fitness. Una volta ottenuti tutti i valori di fitness, il master ordina le coppie in base a tali valori ed evolve la popolazione.
Così facendo abbiamo notevolmente incrementato le prestazioni del nostro programma. Esso infatti risulta essere distribuito ed esegue in parallelo 11 istanze, nonché 10 valutazioni di POPULATION_SIZE/10 coppie. La velocità di esecuzione è stata notevolmente aumentata.

Capitolo 5

Risultati

In questo capitolo saranno presentati i risultati ottenuti nelle varie simulazioni. Quando non meglio specificato, i parametri sono stati mantenuti come indicato nel capitolo relativo alla progettazione.
Visto il nostro assegnamento dei valori di fitness, il punteggio massimo è 100 punti, ottenuto non entrando mai nei muri, non scontrandosi mai con altri robot e raccogliendo tutte e 10 le lattine.

5.1 Vista a Croce Non Collaborativa

Con la vista a croce non collaborativa i robot non possono vedere il proprio compagno sulla mappa. Le celle che sono in grado di percepire sono quella a nord, quella a sinistra, quella nella quale si trovano, quella a destra e quella a sud.

5.1.1 100 Coppie

La prima simulazione è stata effettuata usando lo schema dell'algoritmo genetico descritto precedentemente, con un numero di coppie pari a 100, ovvero 200 robot. In 5000 generazioni il valore di fitness, dopo un primo incremento, si è sempre mantenuto attorno al valore 0. Questo ci ha fatto supporre che la strategia di evoluzione da noi adottata dovesse essere modificata. Anziché generare tutti i nuovi individui mediante l'utilizzo del crossover, abbiamo deciso di copiare alcuni degli individui vecchi nella nuova generazione, adottando una strategia élitaria.

5.1.2 100 Coppie, da 1 a 5 Vecchie

La Figura 5.1 mostra i valori di 6 simulazioni nelle quali abbiamo mantenuto da 0 (tutti gli individui generati tramite l'uso del crossover) a 5 coppie (le migliori) dalla vecchia alla nuova generazione. Come è possibile notare, più sono gli individui che vengono mantenuti da una generazione all'altra, più alto è il valore di fitness raggiunto.

5.1.3 100 Coppie, da 10 a 35 Vecchie

In Figura 5.2 abbiamo proseguito con il ragionamento, copiando nella vecchia generazione da 10 a 35 coppie, incrementando l'intervallo da 1 a 5. I valori di fitness si mantengono tutti fra i 90 ed i 95 punti.

5.1.4 100 Coppie, da 40 a 90 Vecchie

A questo punto ci siamo chiesti quando le performance sarebbero crollate ed abbiamo ulteriormente iterato il procedimento mantenendo da 40 a 90 coppie, procedendo per intervalli di 10. Come è mostarto in Figura 5.3, man mano che il numero di coppie vecchie passate nella nuova generazione aumenta, il valore di fitness inizia a scendere, fino a mostrare cali importanti con 80 e 90 coppie. Mantenendo nella nuova generazione 40, 50 e 60 coppie, comunque i valori di fitness rimangono fra i 90 ed i 95 punti.

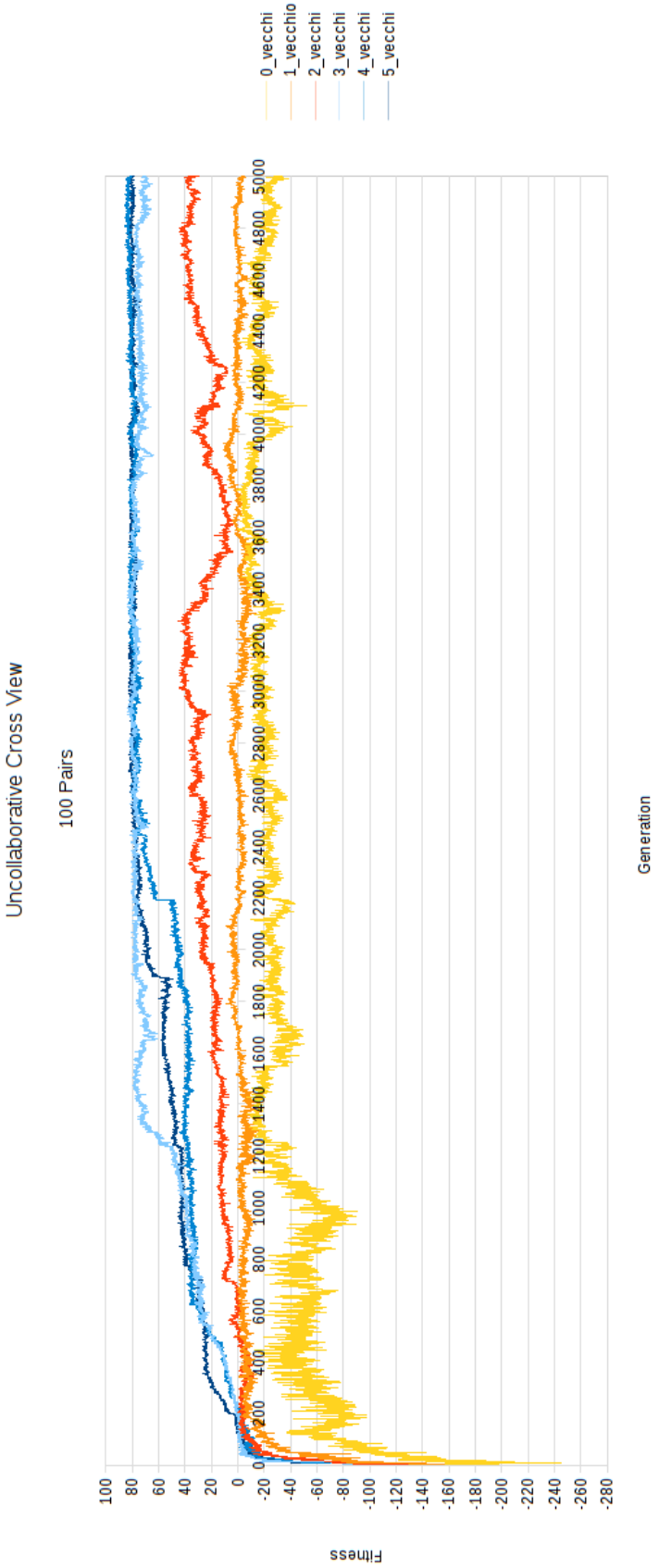

Figura 5.1: Vista a croce non collaborativa, 100 coppie

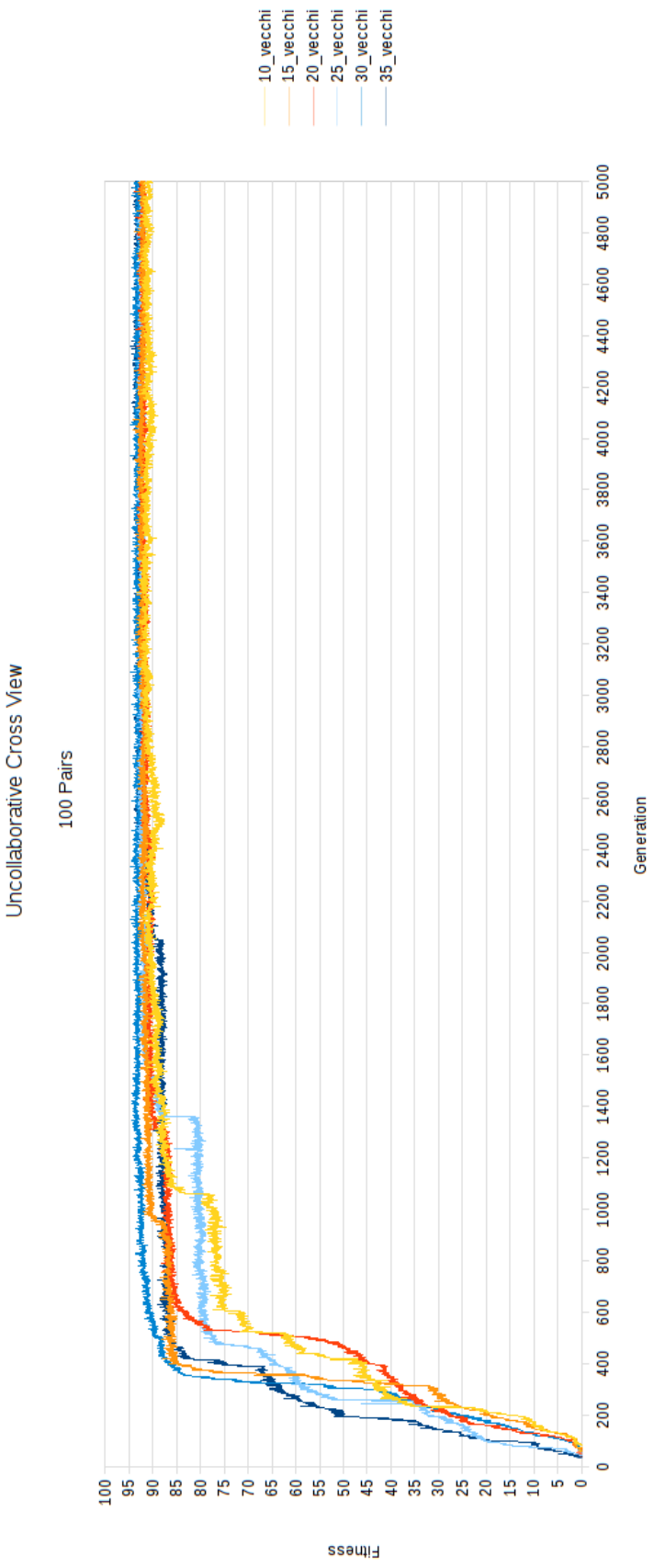

Figura 5.2: Vista a croce non collaborativa, 100 coppie

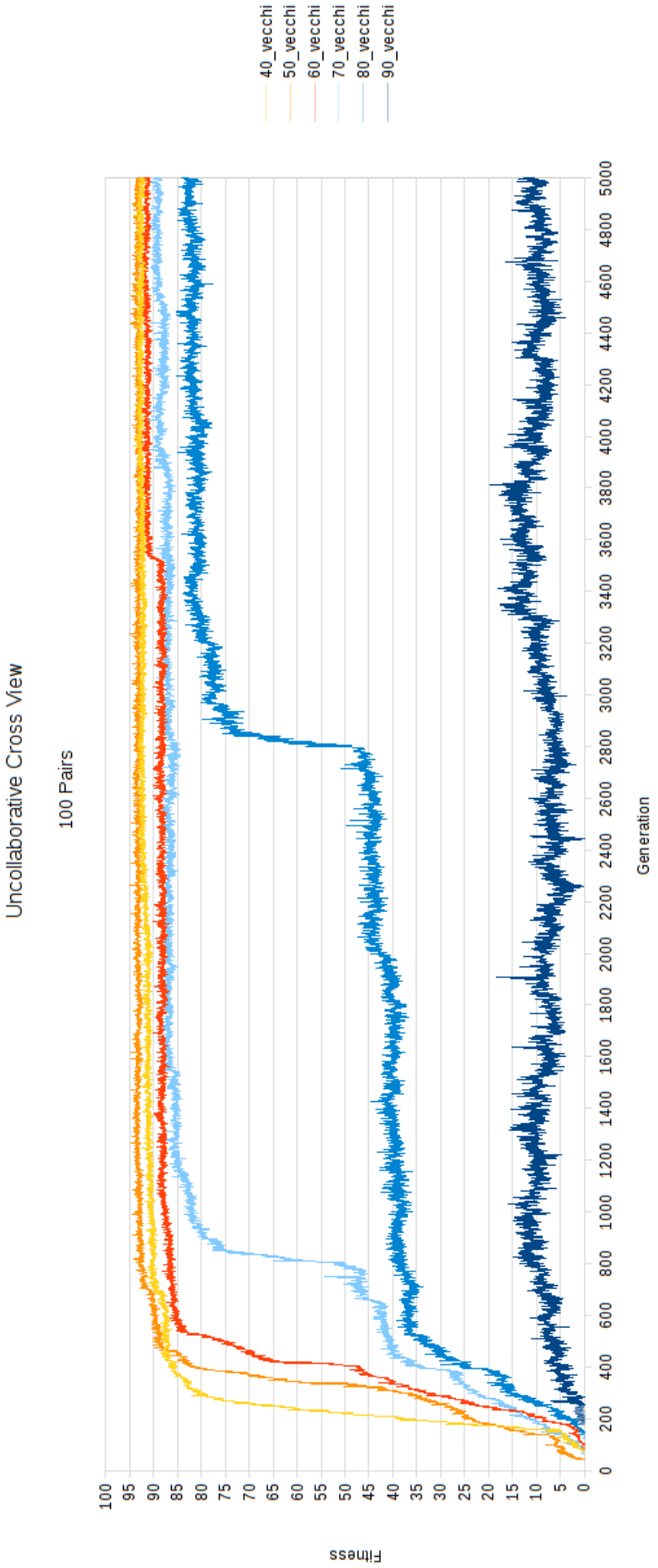

Figura 5.3: Vista a croce non collaborativa, 100 coppie

5.1.5 200 Coppie, da 10 a 180 Vecchie

Avendo notato come la modifica di un semplice parametro (il numero di coppie vecchie da mantenere nella nuova popolazione) possa influenzare in maniera significativa l'evoluzione, abbiamo deciso di ripetere lo stesso procedimento aumentando il numero di coppie che formano la popolazione da 100 a 200. Così facendo abbiamo sperato, avendo più varietà genetica all'inizio, di poter raggiungere valori di fitness più alti.
La Figura 5.4 mostra l'andamento dell'evoluzione tenendo da 10 a 60 coppie, la Figura 5.5 da 70 a 120 e, infine, la Figura 5.6 grafica l'andamento dei valori di fitness tenendo da 130 a 180 coppie vecchie nella nuova generazione.
Per quanto riguarda il raggiungimento di valori di fitness più alti, partire con una popolazione più grande (di dimensione doppia) non ha dato i risultati sperati. Se però si notano le velocità con cui le varie curve crescono verso i valori di fitness più grandi (il numero di generazioni che impiegano a superare, ad esempio, gli 80/85 punti di fitness), ci si rende conto come avere una popolazione formata da 200 coppie incrementi di molto le performance.
Questo esperimento ci ha poi permesso di identificare una costante, ovvero, che i valori di fitness più alti sono raggiunti mantenendo una percentuale di coppie vecchie da una generazione all'altra che va dal 10% al 60%. Con una popolazione di 100 coppie, infatti, i valori di fitness tra i 90 ed i 95 punti sono stati raggiunti mantenendo da 10 a 60 vecchie coppie nelle nuove generazioni, mentre, incrementando il numero di coppie a 200, gli stessi valori di fitness sono raggiunti mantenendo da 20 a 120 vecchie coppie nelle nuove generazioni.

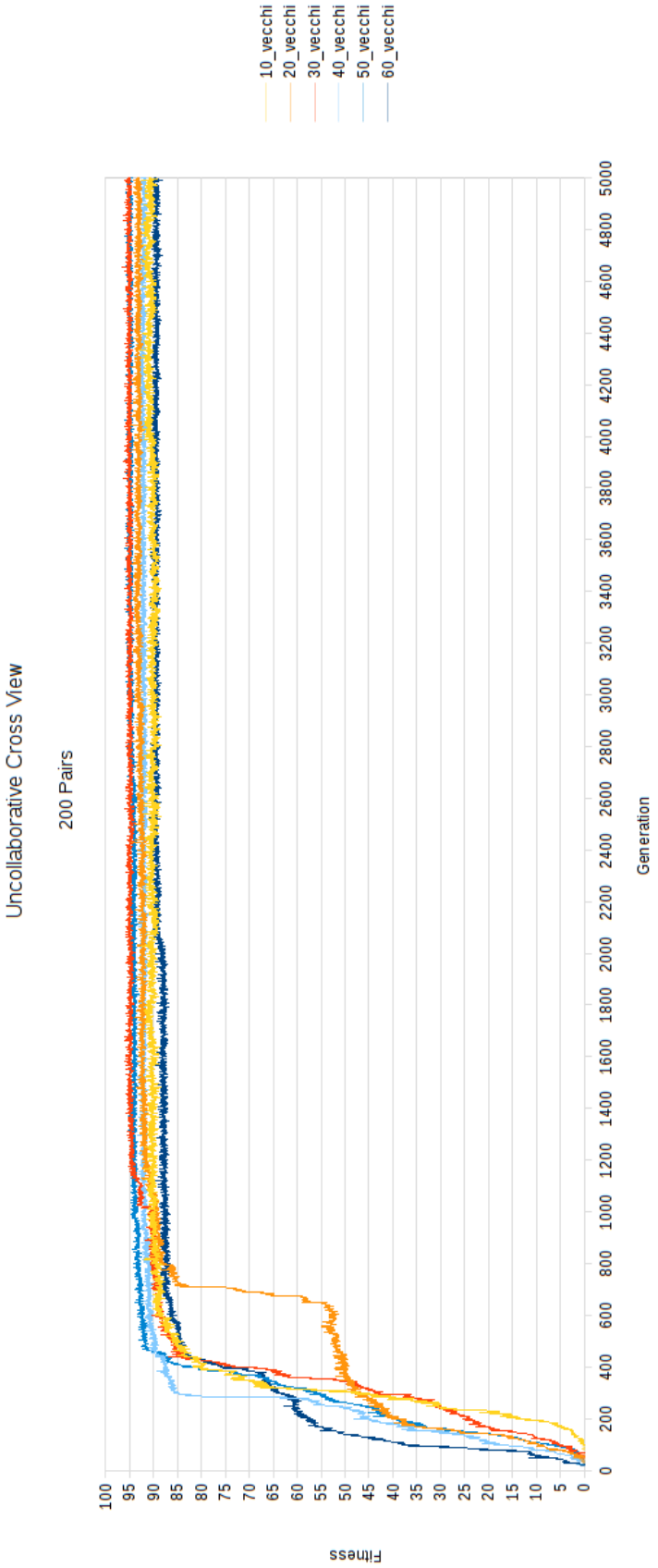

Figura 5.4: Vista a croce non collaborativa, 200 coppie

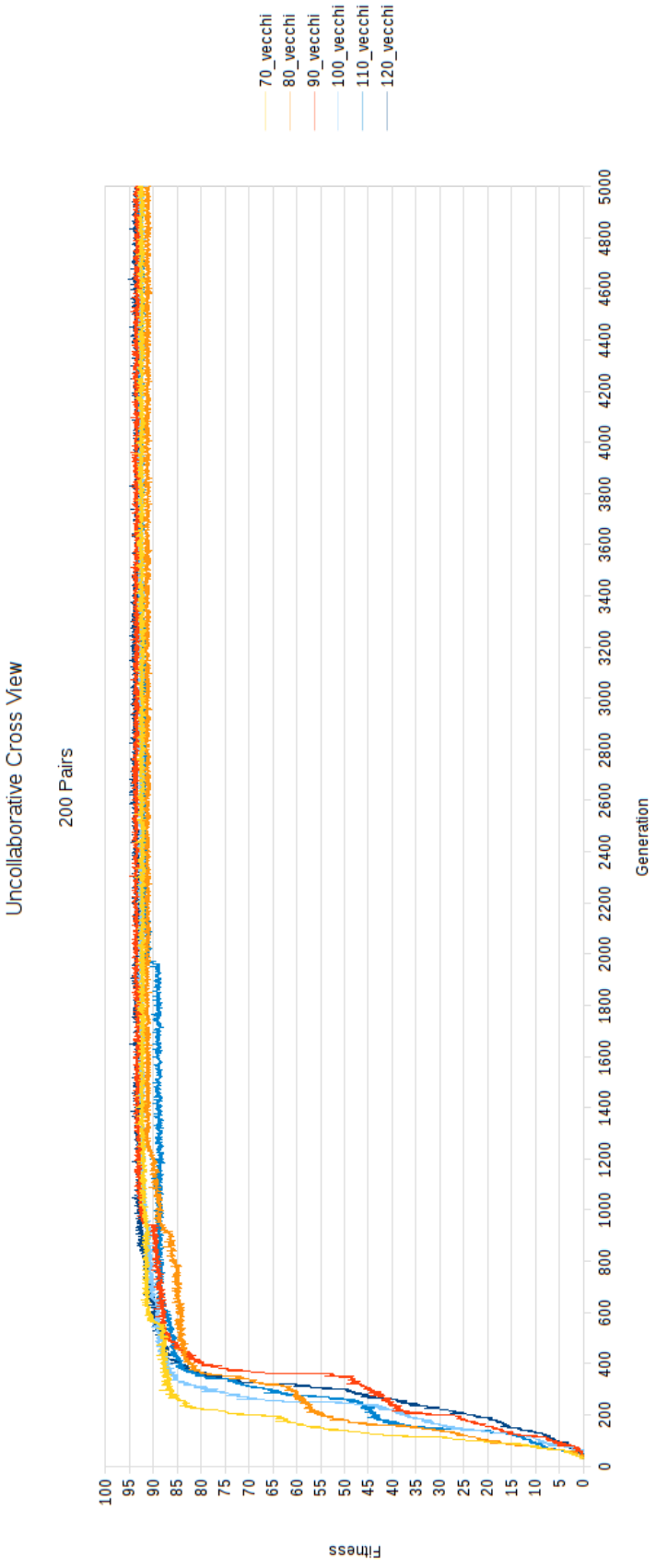

Figura 5.5: Vista a croce non collaborativa, 200 coppie

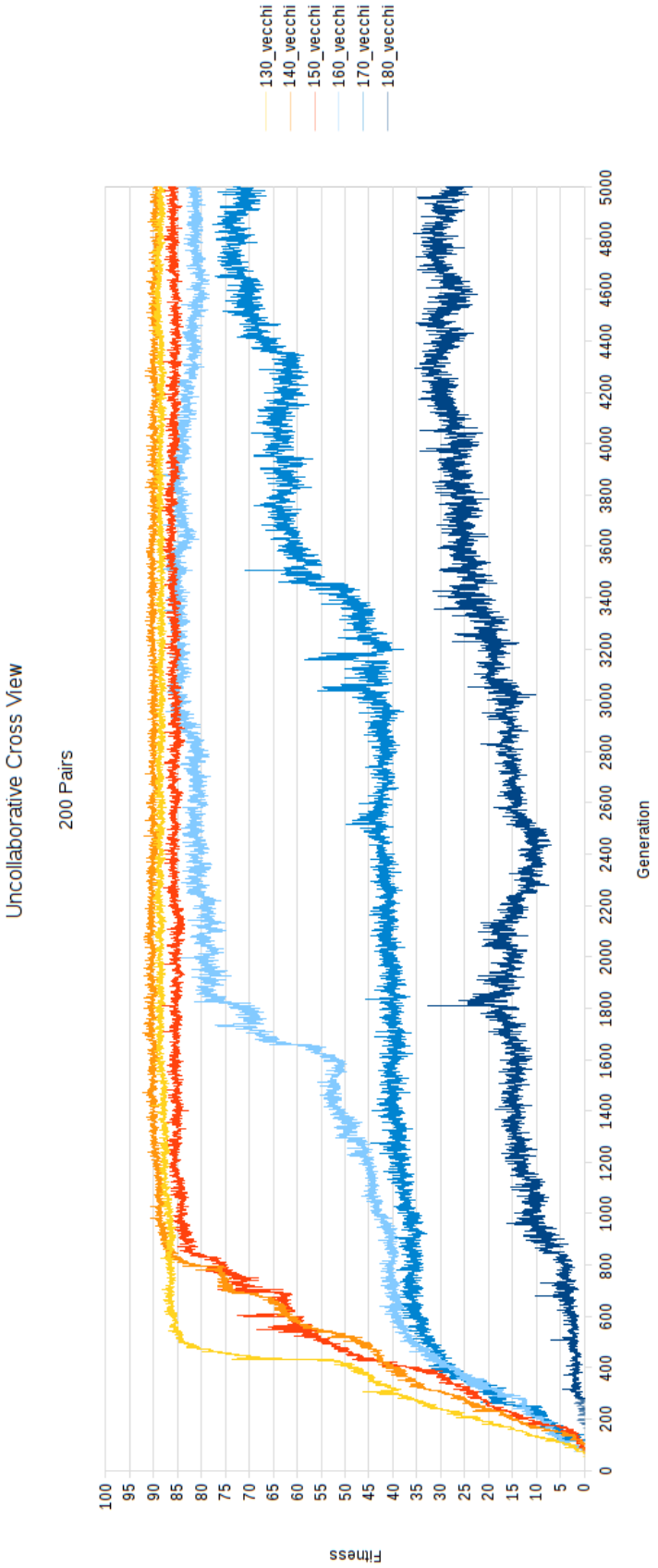

Figura 5.6: Vista a croce non collaborativa, 200 coppie

5.1.6 200 Coppie, da 70 a 120 Vecchie Non Mutate

Abbiamo poi effettuato un'ultima simulazione nella quale abbiamo ripreso il numero di coppie dalla vecchia alla nuova generazione che meglio si sono comportate (che hanno raggiunto i valori di fitness più elevati) con popolazione di dimensione 200 coppie, dove però le mutazioni genetiche sono state applicate solo ai nuovi individui generati e non più anche ai vecchi.
I risultati sono riportati in Figura 5.7. Confrontando questo grafico con quello di Figura 5.5, si nota come la velocità nella crescita sia paragonabile ma, non mutando le vecchie coppie, i valori di fitness siano tutti più elevati, non scendendo mai sotto i 95 punti fitness.

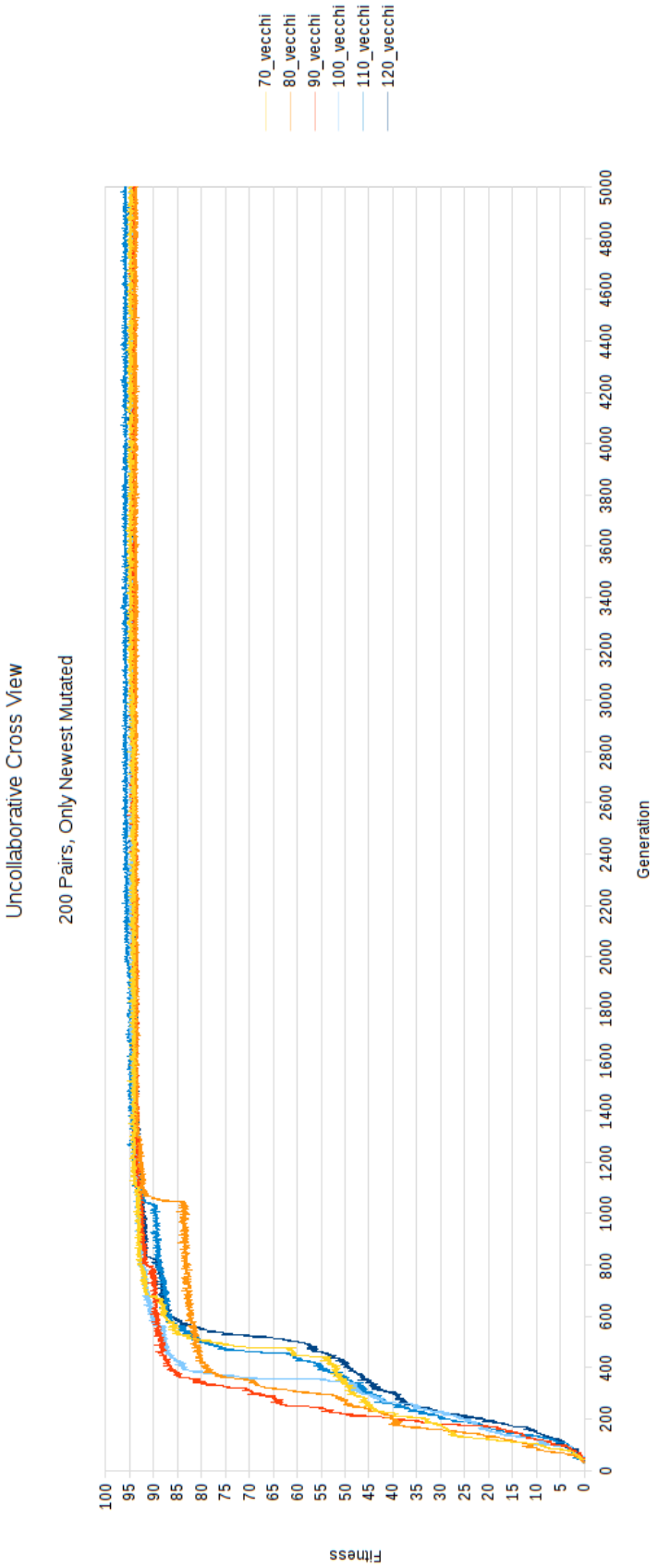

Figura 5.7: Vista a croce non collaborativa, 200 coppie, vecchie non mutate

5.2 Vista Quadrata Non Collaborativa

Questo tipo di vista, così come la precedente, non permette ai robot di percepire la presenza degli altri robot sulla mappa. A differenza di quella a croce, però, questa fa sì che i robot possano vedere un numero di celle maggiore. Sono infatti visibili anche le caselle in alto a sinistra, in alto a destra, in basso a sinistra ed in basso a destra, trasformando quella che prima era una croce in un quadrato di 3x3 celle. Come nel caso precedente, il robot occupa la posizione centrale (seconda riga e seconda colonna).

5.2.1 200 Coppie, da 70 a 120 Vecchie Non Mutate

Forti dei risultati ottenuti con le simulazioni precedenti, abbiamo evoluto una popolazione di 200 coppie tenendo, di volta in volta, da 70 a 120 coppie della vecchia generazione nella nuova, mutando solamente i nuovi individui generati con il crossover.
I risultati sono mostrati in Figura 5.8. Questo risultato ci ha particolarmente sorpresi in quanto, a rigor di logica, le entità impegnate nel ripulire la mappa, potendo vedere di più, avrebbero dovuto migliorare i risultati della vista a croce. Così non è stato, anche se i risultati precedentemente ottenuti sono stati uguagliati. Non ci rimane che concludere che una vista più grande non permette di migliorare le prestazioni che tengono conto del numero di lattine raccolte.

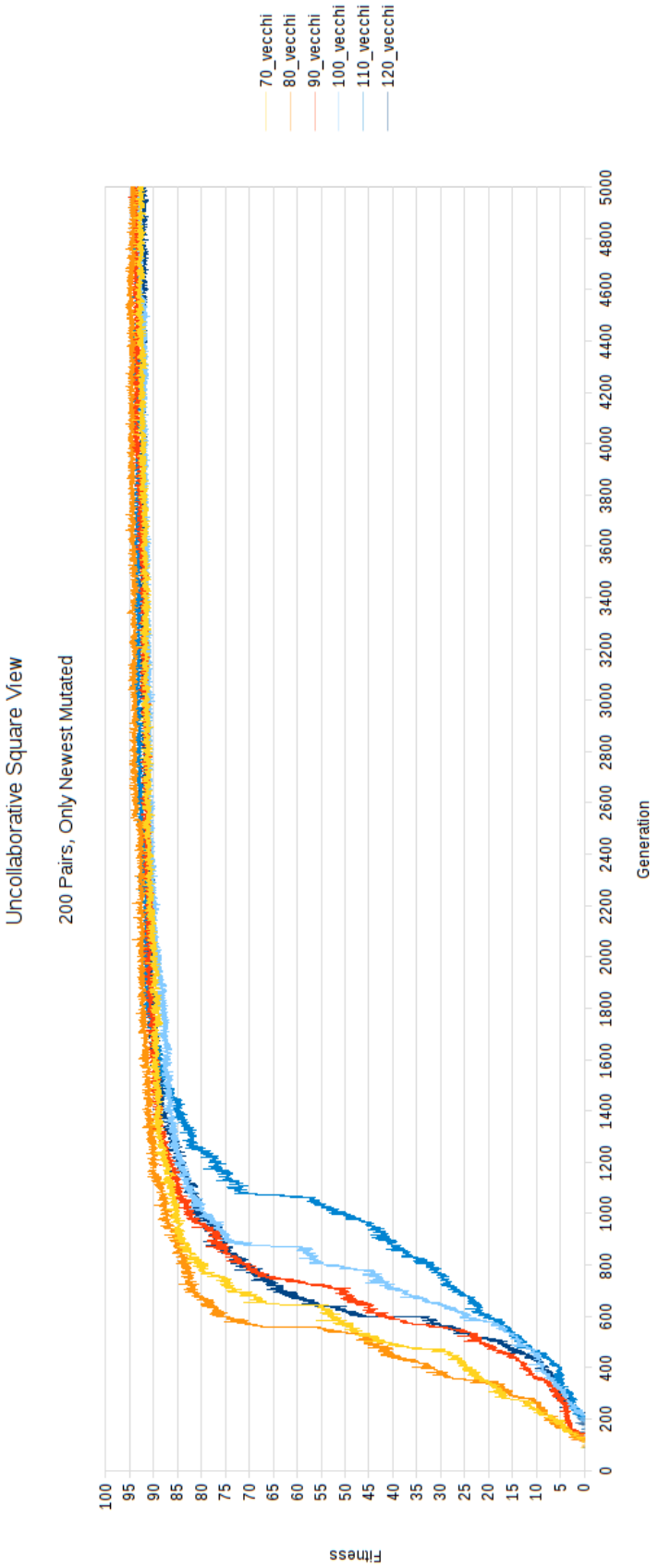

Figura 5.8: Vista quadrata non collaborativa, 200 coppie, vecchie non mutate

5.3 Vista a Croce Collaborativa

TODO

5.3.1 200 Coppie, da 70 a 120 Vecchie Non Mutate

TODO

5.4 Viste Non Collaborative a Confronto

Nelle sezioni precedenti abbiamo visto come evolvendo una popolazione formata da 200 coppie dove ne è mantenuto un certo numero dalla vecchia alla nuova generazione senza apportarvi modifiche, siano raggiunti in media i 95 punti fitness. Abbiamo anche visto come, dotare i robot di una vista più grande non permetta loro di ottenere un punteggio maggiore, cioè, raccogliere in media più lattine.
In questa sezione sono messe a confronto le due viste analizzando se, con un numero di passi minore, quella quadrata 3x3 permetta di raccogliere più lattine rispetto a quella a croce. Dunque, non se sono raccolte più lattine in generale, ma se sono raccolte più velocemente (con un numero di passi minore).
Per quanto riguarda le strategie evolutive, saranno mantenute quelle che hanno dato i migliori risultati, ovvero, usando 200 coppie e tenendone invariate dalla vecchia alla nuova generazione da 70 a 120. Il numero di sessioni usate per assegnare un valore di fitness medio alla coppia saranno sempre 200, ma verranno usate solo 30 e 40 azioni (per sessione) anziché 200.

5.4.1 40 Azioni

TODO

5.4.2 30 Azioni

TODO

Capitolo 6

Conclusioni

Bibliografia

[1] Melanie Mitchell,
"*Robby, The Soda-Can-Collecting Robot*".
http://web.cecs.pdx.edu/~mm/ArtificialIntelligenceFall2008/Homework/Homework6.pdf.