POLITECNICO DI MILANO
Corso di Laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

RICERCA DI EQUILIBRI APPROSSIMATI PER I GIOCHI IN FORMA ESTESA BASATI SU SIMULAZIONE

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Relatore: Prof. Nicola Gatti
Correlatore: Prof. Marcello Restelli

Tesi di Laurea Specialistica di:
Marco Bellen, matricola 724784

Anno Accademico 2009-2010

a Papà e a Nonna Giò

Sommario

Durante lo studio qui documentato, ci si è addentrati all'interno di una branca dell'ingegneria dell'informazione molto specifica, la *teoria dei giochi*: in questa disciplina, l'attenzione è focalizzata sullo studio e sulla risoluzione di particolari situazioni, in cui più soggetti interagiscono tra loro al fine di raggiungere un obiettivo prefissato.

In particolare, lo scopo della tesi è quello di ideare e realizzare delle procedure per risolvere una determinata classe di giochi, meglio conosciuta con il termine *giochi in forma estesa basati su simulazione*: ad oggi, infatti, in letteratura non sono presenti degli algoritmi efficaci per risolvere tali modelli in un tempo ragionevole.

Durante lo sviluppo di questo lavoro, sono state realizzate delle procedure mirate alla localizzazione di equilibri in questa classe di giochi; in particolare, questi algoritmi sono stati progettati al fine di individuare due particolari tipi di equilibrio: gli *equilibri perfetti per sottogiochi* e gli *equilibri di Nash*. Questi metodi di ricerca sono stati sviluppati richiamando diverse tecniche di ottimizzazione quali il *simulated annealing*, il *cross entropy* e la *Lipschitz optimization*. Una volta conclusa la fase di implementazione, i due algoritmi realizzati – SPE–Approximation e NE–Approximation – sono stati valutati su un unico scenario, riportando tutti i risultati in tabelle riassuntive: attraverso l'analisi di quest'ultime, è emerso come gli algoritmi mirati ad individuare gli equilibri di Nash siano decisamente più veloci ed affidabili rispetto alla ricerca degli equilibri perfetti per sottogiochi mentre, tra gli algoritmi di ottimizzazione, il migliore è stato rintracciato nel cross entropy (sia in termini di tempo che in termini di precisione).

Ringraziamenti

Dopo sei anni, finalmente, sono arrivato al termine della mia carriera universitaria, ancora non riesco a crederci ma è così! La soddisfazione di esser riuscito a raggiungere un traguardo simile è immensa ma, come tutte le cose, sono stato aiutato e sostenuto da molte persone che hanno condiviso con me questi anni infernali.

Scrivere i ringraziamenti è sempre stata la parte più difficile per me: molte persone in tutti questi giorni mi sono state vicine, altre invece mi hanno dato grossi dispiaceri, ma comunque tutti quanti – nel bene o nel male – avete contribuito a farmi diventare ciò che sono. Cercherò di menzionarvi tutti quanti, ma non abbiatene a male se mi dimenticherò di qualcuno.

Per prima cosa voglio ringraziare Nicola e Marcello, che mi hanno fornito un validissimo aiuto e supporto per la realizzazione di questo lavoro: con me avete avuto una gran pazienza, e di questo ve ne sarò per sempre grato. Mi sembra doveroso anche "scusarmi" con il professor Matteucci che si è trovato alcune volte costretto, causa i nostri incontri, a traslocare dal suo ufficio e vagare per il dipartimento in cerca di tranquillità.

Un ringraziameno doveroso va ai miei genitori, Giulia e Riccardo, che mi hanno permesso di intraprendere questo ciclo di studi, e soprattutto mi hanno sempre sostenuto nei momenti di difficoltà, quando sembrava che tutto andasse storto (per non parlare poi di quando li ho costretti a correggermi la tesi).

Un saluto particolare va anche ai miei fratelli Claudio e Luca, e a tutti i miei familiari, soprattutto a Nonna Giò: non credo che questa tesi ti piacerà come quella della triennale, ma fidati che anche questo è stato un bel lavoro! Un saluto grande va anche a Nonna Carla e Nonno Aurelio, che adesso potranno orgogliosamente dire di essere i nonni di un Ingegnere!

Una menzione speciale va per i due "ciappolotti" Marco e Lele che, come nessun'altro hanno sofferto insieme a me per raggiungere la meta: Marco,

penso che quei due giorni di luglio passati a casa tua a studiare Robotica con 35 gradi non me li dimenticherò mai più; e la stessa cosa vale per le ore passate a studiare con te, Lele (e anche a giocare a carte sul treno, diciamocelo). Come dimenticarsi di quella volta che ti abbiamo passato gli appunti di RIM dal finestrino del treno????

Grazie a tutti i compagni dell'università che hanno sofferto insieme a me per questi lunghi anni: Cry, Frà, Vale, Piero, Luca, tutti i Marco che ho conosciuto (decisamente troppi!!!), Tommy e Stè.

Non posso non citare anche due figure indispensabili per la mia completa maturazione: la T–89 (a cui devo un 26) e la Scimmietta Scaramella!

Nonostante vi abbia già menzionati, un doveroso tributo va ai componenti dell'inimitabile Cooltraveling Team – Marco, Lele, Tommy e Stè – che mi hanno dimostrato che c'è speranza per noi, poveri ingegneri informatici, di trovare lavoro. Come al solito sarete impegnati nei vostri mille progetti e lavori, ma sappiate che siete stati veramente magici al concorso europeo: grandissimi!

Ovviamente un saluto ed un ringraziamento va anche a coloro che mi hanno tenuto compagnia nella triennale (oltre a quelli già citati), quindi non posso fare a meno di ringraziare con tutto il cuore Marco che mi ha alleggerito la sofferenza nei primi due anni di Polimi e, naturalmente, anche la Fede che, nonostante adesso sia una donna in carriera, mi ha fatto ridere e divertire per tre anni.

Un saluto va anche ai miei amici Benji, Confa, Luca, Andrea, Ciccio, Ele, Cinzia, Cozza, Sella, Just, Giò, l'inglesissimo Stè, Paolo, Erika, Mary, la mia compagna di viaggio Simo e Capu, che mi è venuta a tenere compagnia prima dell'ultimo terribile esame: grandissima!!

Spero di avervi salutato e ringraziato tutti quanti, ma siete troppi: non me ne vogliate male se ho dimenticato qualcuno....

Grazie ancora,
Ciao a tutti

Ing. Marco

Indice

Capitolo 1

Introduzione

1.1 Obiettivo del lavoro

Al giorno d'oggi l'informatica è una scienza ampiamente diffusa in tutti i settori lavorativi, trovando applicazione, in maniera più o meno esplicita, in qualsiasi aspetto della nostra vita. Come la maggior parte delle scienze ingegneristiche, questa è suddivisa in un elevato numero di specializzazioni.

A conclusione del mio percorso di studi ho scelto di approfondire e di concentrare la mia attenzione sul mondo dell'intelligenza artificiale, focalizzandomi in particolare sulla teoria dei giochi.

Questa branca dell'informatica studia ed analizza nella realtà le situazioni di conflitto e ricerca soluzioni competitive o cooperative tramite la costruzione di modelli adeguati; in altre parole, la teoria dei giochi studia come delle decisioni individuali possano influenzare e modificare le strategie di altri soggetti.

In particolare, facendo riferimento a situazioni in cui sono presenti interazioni tra due o più soggetti, ci si concentra sul modo in cui le scelte di un individuo (meglio noto come agente) possano condizionare i risultati conseguibili da parte di uno o più rivali, attraverso un complesso meccanismo di retroazione: è importante però tener presente che tali decisioni vengono effettuate al solo scopo di massimizzare il guadagno conseguibile da parte dell'individuo che le esegue.

Lo scopo di questo studio è la realizzazione di procedure (o algoritmi) di ricerca di equilibri in tale categoria di modelli. Ho quindi incentrato il

mio studio su una particolare sottoclasse – i giochi in forma estesa basati su simulazione – ideando ed implementando algoritmi per la ricerca della miglior strategia dei giocatori.

Per elaborare questo studio, mi sono avvalso di diversi testi tratti da conferenze ed articoli pubblicati su riviste specializzate.

1.2 Struttura della tesi

L'elaborato è strutturato in capitoli che dettagliatamente illustrano tutti gli "step" nei quali si è articolato questo lavoro di laurea.

Il secondo capitolo presenta una panoramica sui principali concetti, con l'introduzione della terminologia della teoria dei giochi e la presentazione dei principali metodi di risoluzione attualmente utilizzati dalla comunità scientifica. In questo capitolo viene anche presentata la classe di giochi su cui si è lavorato durante questo studio: i giochi in forma estesa basati su simulazione.

Nel capitolo 3 vengono presentati e sviluppati gli algoritmi di ottimizzazione impiegati, illustrando le procedure utilizzate per massimizzare le funzioni di utilità dei giocatori coinvolti. All'interno di questa sezione vengono presentati tre algoritmi, basati su simulated annealing (SA), cross entropy (CE) e Lipschitz optimization (LIP).

Nel capitolo successivo è sviluppata la parte centrale della tesi, in cui sono illustrati gli algoritmi di ricerca di un equilibrio all'interno della classe dei giochi in forma estesa basati su simulazione: qui sono descritte le due procedure di ricerca realizzate, mirate all'individuazione di equilibri perfetti per sottogiochi (SPE–Approximation) ed alla ricerca di equilibri di Nash (NE–Approximation).

Il capitolo 5 presenta il modello di simulazione su cui è stata effettuata la fase di testing, analizzando nel dettaglio i risultati conseguiti.

A conclusione dell'elaborato, l'ultimo capitolo traccia una panoramica riassuntiva su tutto il lavoro svolto, proponendo dei possibili sviluppi futuri ed ipotizzando dei perfezionamenti a ciò che è stato presentato in questa sede.

In allegato viene anche inserita l'appendice A, in cui vengono riportate le tabelle riepilogative dei risultati ottenuti nella fase di testing, corredate da grafici che evidenziano l'andamento di ogni singola grandezza considerata.

Capitolo 2

Teoria dei giochi

2.1 Cenni storici

Per meglio comprendere il lavoro qui esposto, è necessario introdurre e analizzare la grande branca dell'informatica a cui si fa riferimento, nota con il termine di **teoria dei giochi**.

La sua nascita viene fatta risalire al 1944, quando il matematico John Von Neumann e l'economista Oskar Morgenstern pubblicarono il testo *"Theory of Games and Economic Behavior"*. Nonostante questa publicazione sia considerata il testo di riferimento, in realtà vengono ripresi alcuni dei concetti già trattati pochi anni prima da altri autori, tra cui lo stesso Von Neumann.

Figura 2.1: *John Von Neumann*

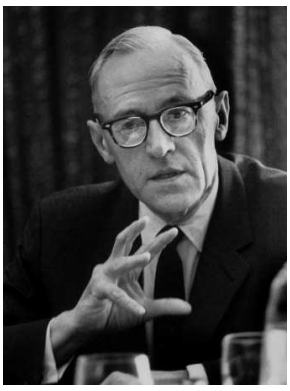

Figura 2.2: *Oskar Morgenstern*

In origine, i due studiosi pensarono alla teoria dei giochi come ad uno strumento per analizzare e catalogare il comportamente umano in situazioni

in cui le scelte operate conducono a tre soli possibili effetti: la vincita, la perdita o la spartizione di una risorsa in gioco. Questa visione estremamente riduttiva negli anni seguenti venne poi modificata, e tali concetti furono estesi ad un range più vasto di scenari.

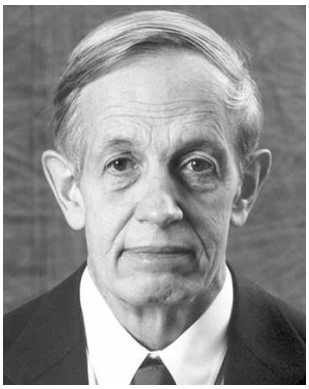

Figura 2.3: *John Nash*

Nonostante Von Neumann e Morgenstern abbiano dato vita alla teoria dei giochi, lo studioso di riferimento della materia è sicuramente il matematico statunitense John Forbes Nash che, nel 1949, durante il suo periodo di dottorato presso l'università americana di Princeton, studiò a fondo i giochi non–cooperativi (vedi *Sezione 2.2.1*) e formulò il teorema che porta il suo nome: il teorema di Nash (vedi *Teorema 1*).

Tale studio fu ufficialmente presentato alla comunità scientifica l'anno seguente, tramite la pubblicazione dell'articolo *"Equilibrium Points in n–person Games"* [1]. All'interno del teorema di Nash fu definito per la prima volta l'**equilibrio di Nash**, concetto cardine su cui ruota tutta la teoria dei giochi.

Questo nuovo concetto diede una vera svolta alla teoria dei giochi, come testimoniato dalla dichiarazione dell'economista statunitense Peter Ordeshook:

> *"Il concetto di "**equilibrio di Nash**" è sicuramente l'idea più importante nella teoria dei giochi, per quel che riguarda i giochi non–cooperativi. Se analizziamo le strategie di elezione dei candidati, le cause della guerra, la manipolazione degli ordini del giorno nelle legislature, o le azioni delle lobby, le previsioni circa gli eventi si riducono ad una ricerca o ad una descrizione degli equilibri. Detto in altri termini e banalizzando, le strategie di equilibrio sono tentativi di predizione circa il comportamento della gente."*
>
> Peter Ordeshook, economista

La teoria dei giochi può avere due ruoli diversi. Il primo (ruolo positivo) è quello di interpretare la realtà, ossia spiegare come mai, in certe situazioni di conflitto, i soggetti coinvolti adottano certe strategie e certe tattiche. Il secondo ruolo (prescrittivo) è invece quello di determinare quali situazioni di equilibrio possono (o non possono) verificarsi come risultato dell'interazione dei soggetti coivolti. In ogni caso, i concetti di soluzione che sono utilizzati nella teoria dei giochi intendono descrivere quelle tattiche (strategie) che i decisori, individualmente o congiuntamente, dovrebbero seguire come conseguenza delle ipotesi di razionalità assunte.

2.2 I giochi

Come già accennato, la teoria dei giochi è focalizzata sulla creazione e risoluzione di modelli estrapolati da situazioni quotidiane della vita. È così che l'interazione competitiva o cooperativa di due o più individui – che vengono definiti **agenti** – viene tradotta in modelli (detti anche *giochi*) allo scopo di analizzare per ciascun individuo tutti i possibili guadagni (*payoff*).

Con il termine **giocatore** si intende un qualsiasi agente che compia una scelta all'interno del gioco, o che venga influenzato dall'esito di una scelta effettuata da un altro.

In particolare, questa teoria si concentra su situazioni in cui le azioni di ogni singolo agente influenzano i risultati conseguibili dagli altri giocatori, mediante un meccanismo di retroazione.

Ogni gioco appartiene quindi – in base allo scenario che mira a rappresentare – a delle determinate categorie e deve soddisfare alle seguenti condizioni necessarie:

- Ogni giocatore deve essere a conoscenza delle regole del gioco a cui partecipa;
- Ogni giocatore deve essere consapevole delle conseguenze di ogni singola azione (almeno per quel che lo riguarda).

Formalmente un gioco G è definito dalla tupla $G = (N, A, u)$, dove N corrisponde all'insieme degli agenti partecipanti al gioco e, per ogni agente $i \in N$:

- $A = A_1 \times A_2 \times \ldots \times A_N$: insieme non vuoto delle azioni che possono essere eseguite dai vari giocatori, dove con A_i si intende l'insieme di mosse che può compiere l'agente i–esimo;
- $u = (u_1, u_2, \ldots, u_N)$: insieme delle funzioni di utilità degli agenti partecipanti al gioco, dove $u_i : A \to \mathbb{R}$ corrisponde alla funzione di utilità del giocatore i–esimo, ottenuta grazie alla strategia A scelta.

Per lo svolgimento del gioco, ogni agente intraprende un'azione (*mossa*) che, in base alla scelta operata, influirà sulla successiva mossa degli altri giocatori. Questa concatenazione di scelte crea quella che viene definita una

strategia: essa individua la mossa ammissibile del giocatore per ogni circostanza in cui è chiamato ad agire. In base all'insieme delle strategie adottate da tutti i giocatori, ognuno riceve un valore di utilità, detto **payoff**, secondo un'adeguata unità di misura che può essere positiva, negativa o nulla. Per convenzione, le strategie vengono indicate con la lettera greca sigma, e possiamo distinguere tra σ_i e $\boldsymbol{\sigma}$: con σ_i si indica l'azione (o la sequenza di azioni, a seconda della natura del gioco) che effettua il giocatore i–esimo, mentre con $\boldsymbol{\sigma}$ si intende il **profilo di strategie**, vale a dire l'insieme di tutte le strategie degli agenti coinvolti nel gioco (formalmente si indica come $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_n)$). Un profilo di strategie deve contenere un'unica strategia per ciascun giocatore partecipante. È di fondamentale importanza non confondere tra loro i concetti di mossa e di strategia: la prima è un'azione intrapresa da un agente in un certo momento del gioco (ad esempio, nel caso degli scacchi è il movimento di un pezzo in una nuova casella), mentre la seconda è una sequenza non vuota di mosse.

Per meglio chiarire il concetto di strategia, di seguito viene inserito un esempio di svolgimento di un gioco.

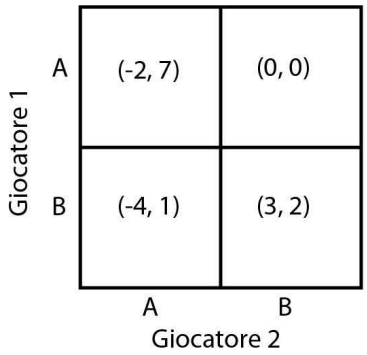

Figura 2.4: *Esempio di gioco*

Nel caso riportato in *Figura 2.4*, se

$$\begin{cases} \sigma_1 = \{B\} \\ \sigma_2 = \{A\} \end{cases}$$

allora si avrà

$$\begin{cases} u_1(\sigma_1, \sigma_2) = -4 \\ u_2(\sigma_1, \sigma_2) = 1 \end{cases}$$

2.2.1 Classificazione dei giochi

Nella teoria dei giochi esistono diversi criteri di suddivisione dei modelli: qui di seguito verranno esposte le principali categorie, in accordo con il testo [2]. Va però osservato che le classi qui elencate non sono in mutua esclusione: ogni gioco, cioè, può appartenere contemporaneamente a più classi.

Cooperativi VS Non–cooperativi

In questa categoria, i giochi vengono classificati sulla base delle relazioni che intercorrono tra i vari agenti coinvolti nel gioco, e si distinguono due classi:

- **Giochi cooperativi**, in cui i giocatori possono cooperare tra di loro, al fine di massimizzare la propria utilità. Molto spesso si assiste alla formazione di coalizioni in cui si racchiudono tutti i giocatori con il medesimo obiettivo. Un esempio di tali modelli è rappresentato dai giochi a squadre (*briscola chiamata* o *scopone scientifico*), o da quei giochi in cui l'obiettivo è massimizzare il guadagno complessivo;
- **Giochi non–cooperativi**, in cui ogni giocatore persegue un obiettivo differente dagli altri concorrenti, ed è vietato lo stipulamento di accordi vincolanti tra i vari agenti (tale classe di giochi è altresì detta *giochi competitivi*). Un esempio di tali giochi è rappresentato dagli *scacchi* o dalla *battaglia navale*.

Forma strategica VS Forma estesa

In questa categoria, i giochi vengono classificati sulla base della natura stessa del gioco, a seconda del numero di turni che lo compongono. In quest'ottica, possiamo distinguere tra:

- **Giochi in forma strategica**, in cui i giocatori possono effettuare una ed una sola mossa, e tutti devono compierla nel medesimo istante. Normalmente per rappresentare questi tipi di giochi viene utilizzata la **matrice di payoff** (vedi *Figura 2.5*): tale struttura è una matrice multidimensionale (possiede tante dimensioni quanti sono i giocatori partecipanti al gioco), in cui l'elemento della cella (i, j) – nel caso in cui si abbia solo due giocatori coinvolti – è dato dalla tupla $(u_1(i, j), u_2(i, j))$, dove con $u_k(i, j)$ si intende l'utilità che guadagnerebbe il giocatore k–esimo se il primo giocatore eseguisse l'azione i e il secondo

effettuasse invece la mossa j. Il formalismo con cui viene rappresentato un gioco in forma strategica non è diverso da quello presentato nella *Sezione 2.2* per un gioco generico. In questi modelli la strategia di ogni giocatore è costituita da una ed una sola azione. Un esempio di tali modelli è rappresentato dalla *morra cinese* o da *pari e dispari*;

- **Giochi in forma estesa**, in cui i giocatori effettuano le loro mosse in modo sequenziale. Per rappresentare questi tipi di giochi viene usata una **struttura ad albero**, dove ciascun livello simboleggia un turno.

  I nodi dell'albero si dividono in

  - **Nodi Non–Terminali**, che simboleggiano i vari turni del gioco. A ciascuno di essi è associato un giocatore: egli può scegliere quale mossa eseguire sulla base delle azioni disponibili a quel punto del gioco;
  - **Nodi Terminali**, che indicano gli stati finali del gioco. Ad ognuno di essi sono associati i valori di utilità che ciascun giocatore guadagnerà nel caso in cui venga selezionato il profilo di strategia corrispondente.

  In questi modelli si inizia a giocare da un unico nodo iniziale (la *radice*), per poi attraversare la struttura ad albero lungo un percorso determinato dai giocatori, sino a che un nodo terminale non viene raggiunto. Utilizzando tale struttura si possono modellizzare degli scenari molto diversi rispetto alla classe dei giochi in forma strategica: si può affermare che la potenza espressiva di questo modello è decisamente superiore.

  Formalmente, i giochi in forma estesa si indicano con la tupla $(N, A, V, T, \iota, \rho, \chi, u)$, dove:

  - N rappresenta l'insieme degli agenti partecipanti al gioco;
  - $A = (a_1, \ldots, a_N)$ rappresenta l'insieme delle azioni possibili;
  - V rappresenta l'insieme dei nodi non–terminali (detti anche *nodi decisionali*) dell'albero di gioco;
  - T rappresenta l'insieme dei nodi terminali dell'albero di gioco;
  - $\iota : V \to N$ rappresenta la funzione che, dato un nodo decisionale, restituisce l'agente che sta per effettuare la propria mossa;

- $\rho : V \to \wp(A)$ rappresenta la funzione che, dato un nodo decisionale v, restituisce le azioni a disposizione del giocatore che deve effettuare la propria mossa;
- $\chi : V \times A \to V \cup T$ rappresenta la funzione che, dati in ingresso un nodo decisionale e un'azione, associa ad essi il nodo (decisionale o terminale) con cui il gioco proseguirà;
- $u = (u_1, \ldots, u_N)$ rappresenta l'insieme delle funzioni di utilità degli agenti partecipanti al gioco, dove $u_i : T \to \mathbb{R}$ è la funzione di utilità del giocatore i–esimo.

Un esempio di tale categoria di giochi è rappresentato dagli *scacchi*, o dalla *scopa*.

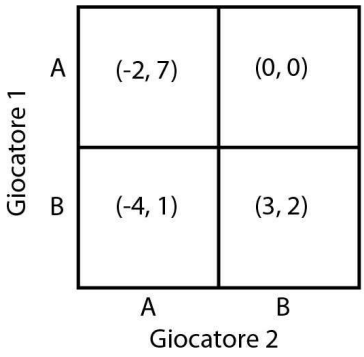

Figura 2.5: *Gioco in forma strategica*

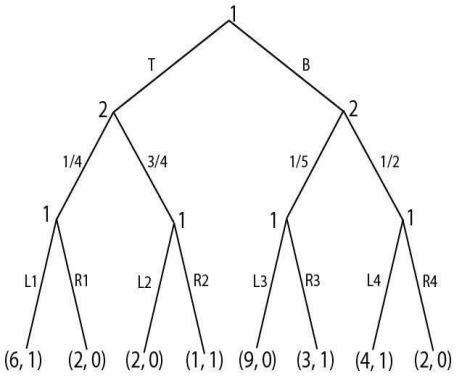

Figura 2.6: *Gioco in forma estesa*

Nell'esempio riportato in *Figura 2.6*, se

$$\begin{cases} \sigma_1 = \{T, L_2\} \\ \sigma_2 = \{\frac{3}{4}\} \end{cases}$$

allora si avrà

$$\begin{cases} u_1(\sigma_1, \sigma_2) = 2 \\ u_2(\sigma_1, \sigma_2) = 0 \end{cases}$$

Informazione perfetta VS Informazione imperfetta

In questa categoria, i giochi vengono classificati sulla base della conoscenza che possiedono i giocatori coinvolti: questo permette ai vari agenti di scegliere delle strategie diverse sulla base delle conoscenze in loro possesso. In questo senso possiamo distinguere tra:

- **Giochi a informazione perfetta**, in cui ogni giocatore conosce tutte le azioni eseguite dagli altri giocatori; nel caso di un gioco in forma estesa, questo significa che l'agente sa in che nodo dell'albero si trova in ogni momento (ad esempio nel caso degli *scacchi* o della *dama*);
- **Giochi a informazione imperfetta**, in cui ogni giocatore non ha nessuna informazione riguardo le azioni eseguite dagli altri agenti; nel caso di un gioco in forma estesa, questo significa che il giocatore non sa in che nodo dell'albero si trova (ad esempio nel caso della *battaglia navale*).

Finiti VS Non–finiti

In questa categoria, i giochi vengono classificati sulla base dell'insieme di azioni su cui i vari giocatori possono scegliere. In questo senso possiamo distinguere tra:

- **Giochi finiti**, in cui i giocatori possono scegliere la propria mossa su di un insieme finito di azioni (un esempio di tale tipologia di giochi è il *tris* o la *morra cinese*); questi modelli si possono rappresentare sia in forma strategica (vedi *Figura 2.5*) che in forma estesa (vedi *Figura 2.6*);
- **Giochi non–finiti**, in cui i giocatori possono scegliere la propria strategia su di un numero infinito di mosse. Tipicamente questa classe di giochi si rappresenta tramite modelli in forma estesa ma, con opportuni accorgimenti, è possibile anche ricorrere alle matrici di payoff. Qui di seguito vengono riportati due esempi di giochi non–finiti in forma strategica (vedi *Figura 2.7*) ed in forma estesa (vedi *Figura 2.8*).

Somma zero VS Somma non–zero

In questa categoria, i giochi vengono classificati sulla base dell'utilità che i vari giocatori ottengono, in base al profilo di strategia scelto. In questo senso possiamo distinguere tra:

- **Giochi a somma zero** dove, per tutti i profili di strategia possibili, la somma delle utilità ottenute dai giocatori è nulla. Più in generale possiamo allargare tale classe, introducendo i giochi a somma costante: in tali modelli la somma delle utilità non è zero, ma è pari ad una

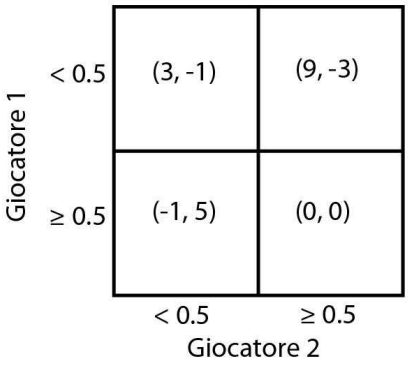

Figura 2.7: *Gioco non–finito in forma strategica*

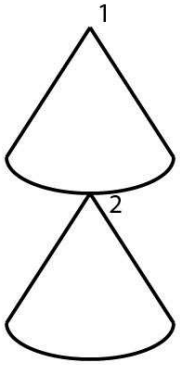

Figura 2.8: *Gioco non–finito in forma estesa*

costante. Un esempio di questi giochi è rintracciabile in *pari e dispari* (vedi *Figura 2.9*) o nella *morra cinese*;

- **Giochi a somma non–zero** in cui la somma delle utilità ottenute da tutti i giocatori non è pari ad un valore costante per tutti i profili di strategia. Un esempio di questi giochi è rappresentato dal *dilemma del prigioniero* (vedi *Figura 2.10*) o dalla *battaglia dei sessi*.

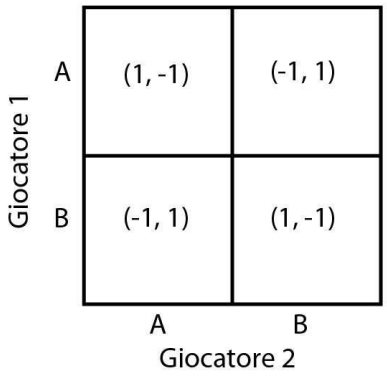

Figura 2.9: *Gioco a somma zero*

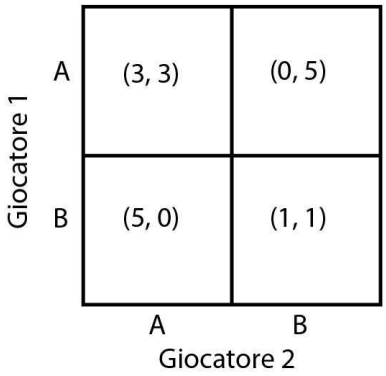

Figura 2.10: *Gioco a somma non–zero*

2.2.2 Giochi basati su simulazione

Durante questo lavoro, l'attenzione si è concentrata esclusivamente su una categoria di giochi non ancora illustrata, la categoria dei **giochi basati su simulazione**. Questa classe di modelli è stata introdotta per la prima volta nel 2008, grazie alla pubblicazione di [3] da parte del professor Michael

Wellman, e racchiude tutti quei modelli nei quali le funzioni di utilità degli agenti non sono analiticamente conosciute, ma vengono generate grazie ad un processo di simulazione effettuato da un **oracolo** (indicato con la notazione $\mathcal{O}$). Passando in input all'oracolo la strategia congiunta di tutti gli agenti, questo è in grado di restituire un vettore contenente le utilità dei giocatori partecipanti.

In particolare, in questa sede ci si focalizzerà sui giochi in forma estesa, vale a dire su tutti quei modelli rappresentabili mediante la struttura ad albero.

Considerando che la maggior parte dei processi di simulazione sono basati su variabili a valori reali, una categoria di studio molto interessante è sicuramente quella dei giochi continui, costituita da modelli in cui le azioni disponibili per ogni giocatore variano all'interno di uno spazio continuo: in questo modo, le mosse a disposizione dei giocatori equivalgono all'assegnamento di una o più variabili reali.

In questa sede, inoltre, sono stati presi in considerazione solamente i giochi ad informazione perfetta; in particolare lo studio si è concentrato sui **giochi a memoria perfetta**, vale a dire su quella categoria di giochi in cui ogni agente è in grado di richiamare tutte le azioni eseguite da tutti i giocatori che lo hanno preceduto, permettendo così ad ogni agente di mantenere lo storico di tutte le mosse effettuate.

I modelli basati su simulazione utilizzano lo stesso formalismo visto in precedenza per quelli in forma estesa (vedi *Sezione 2.2.1*).

Trattandosi di giochi basati su simulazione, nei modelli analizzati in questo studio non saranno note a priori le funzioni di utilità dei vari giocatori: formalmente, il componente u della tupla $(N, A, V, T, \iota, \rho, \chi, u)$ verrà sostituito dall'oracolo $\mathcal{O}$ il cui argomento è $t \in T$. Possiamo perciò dire che la tupla $(N, A, V, T, \iota, \rho, \chi, \bar{u})$, dove $\bar{u} = E[\mathcal{O}]$, indica il modello sottostante su cui si costruisce il gioco basato su simulazione in cui l'oracolo corrisponde a $\mathcal{O}$.

2.2.3 Strategie

Nella teoria dei giochi è possibile distinguere tra:

- **Strategia pura**, che fornisce una definizione completa del modo in cui un agente gioca una partita: in particolare, essa determina quale scelta eseguirà il giocatore in tutte le situazioni che potrebbe dover affrontare.

- **Strategia mista**, che fornisce la distribuzione di probabilità sull'insieme delle strategie pure che il giocatore ha a disposizione. Se un partecipante al gioco ha almeno due strategie pure, esistono infinite strategie miste a disposizione dello stesso. Essendo un valore probabilistico, una strategia mista è un intero compreso tra 0 e 1 che esprime la probabilità che il giocatore scelga una strategia pura rispetto ad un'altra. Da questa definizione è osservabile come una soluzione di tipo pura non sia altro che un caso particolare di strategia mista: è possibile infatti affermare che una soluzione pura consiste in una strategia mista in cui tutte le probabilità sono nulle, ad eccezione di una che varrà 1.

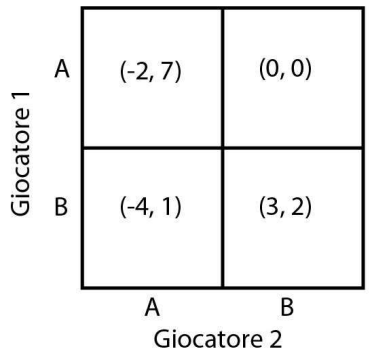

Figura 2.11: *Esempio di gioco con strategie miste*

Prendendo in analisi il gioco modellizzato in *Figura 2.11*, risulta che:

$$\sigma_1 = \begin{cases} A, \text{ con probabilità pari a } \frac{1}{8} \\ B, \text{ con probabilità pari a } \frac{7}{8} \end{cases} \qquad \sigma_2 = \begin{cases} A, \text{ con probabilità pari a } \frac{3}{5} \\ B, \text{ con probabilità pari a } \frac{2}{5} \end{cases}$$

Tali valori rappresentano la soluzione al modello nel caso di strategie miste.

Più formalmente, in un gioco in forma estesa, per strategia pura σ_i si intende un piano di azioni che specificano una ed una sola mossa per ogni nodo decisionale dell'agente i, mentre una strategia mista σ_i consiste in una randomizzazione delle strategie pure. In particolare, all'interno di un **gioco continuo** – ovvero in modelli in cui le azioni disponibili per ogni giocatore variano all'interno di uno spazio continuo – la definizione di un piano singolo è molto più complessa e, di conseguenza, non è conveniente ricorrere a questi tipi di strategia.

Un'alternativa più compatta è rappresentata dalla **strategia comportamentale**: essa definisce il comportamento di un agente ad ogni nodo, indipendentemente dalle scelte effettuate in precedenza. In sintesi, una strategia comportamentale σ_i assegna ad ogni nodo decisionale $v \in V$ una distribuzione di probabilità sulle azioni disponibili in v.

Nel caso di **giochi a memoria perfetta**, ovvero nei giochi in cui ogni agente ricorda le azioni eseguite da tutti i giocatori in precedenza, le rappresentazioni fino a qui illustrate si equivalgono. All'interno dei giochi continui, le strategie comportamentali sono spesso espresse sottoforma di funzioni che mappano le azioni disponibili, in base alla strategia tenuta durante i nodi precedenti.

2.2.4 Equilibrio di Nash

Come già accennato all'interno della *Sezione 2.1*, il teorema di Nash rappresenta il concetto fondamentale della teoria dei giochi, in quanto definisce il più efficace metodo di risoluzione di un gioco. A questo punto, è necessario introdurre una serie di concetti preliminari che contribuiscano alla comprensione del teorema.

Sia G un generico gioco caratterizzato da:

- un insieme N di giocatori che verranno indicati con $i = 1, \ldots, N$;
- un insieme $\boldsymbol{\sigma}$ di strategie costituito da N vettori, che indicheremo come $\sigma_i = (\sigma_{i,1}, \sigma_{i,2}, \ldots, \sigma_{i,M_i})$, dove con M_i si intende il numero di turni che il giocatore i–esimo può fare. Di conseguenza, si può affermare che il vettore σ_i contenga l'insieme delle strategie che il giocatore i–esimo ha a disposizione, cioè l'insieme delle azione che egli può compiere;
- un insieme u di funzioni $u_i = u_i(\sigma_1, \sigma_2, \ldots, \sigma_N)$ che associano ad ogni giocatore i il payoff u_i derivante dal profilo di strategie $(\sigma_1, \sigma_2, \ldots, \sigma_N)$.

Un **equilibrio di Nash** per un gioco è un profilo di strategie

$$\boldsymbol{\sigma}^* = ({\sigma_1}^*, {\sigma_2}^*, \ldots, {\sigma_N}^*)$$

tale per cui valga la seguente disequazione:

$$u_i({\sigma_1}^*, {\sigma_2}^*, \ldots, {\sigma_i}^*, \ldots, {\sigma_N}^*) \geq u_i({\sigma_1}^*, {\sigma_2}^*, \ldots, \sigma_i, \ldots, {\sigma_N}^*)$$

per ogni i e per ogni strategia σ_i scelta dal giocatore i–esimo. In altre parole, un equilibrio di Nash è un profilo di strategia $\boldsymbol{\sigma}^*$ tale per cui nessun giocatore è invogliato a cambiare la propria strategia σ_i perchè non ne trarrebbe alcun vantaggio, in quanto non potrebbe mai ottenere un guadagno superiore rispetto a quello derivante dall'equilibrio di Nash.

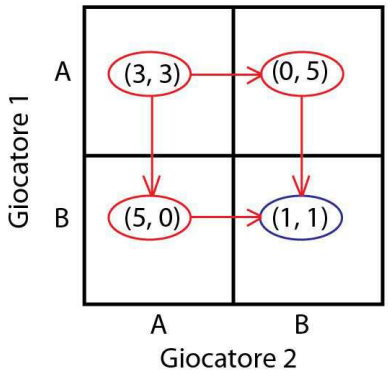

Figura 2.12: *Equilibrio di Nash per il dilemma del prigioniero*

Ad esempio, nel caso del *dilemma del prigioniero* modellizzato in *Figura 2.12*, esiste un unico equilibrio di Nash, ed è definito dal profilo di stategia $\boldsymbol{\sigma}^* = (B, B)$; infatti, analizzando tutti i profili di strategia, si verifica come nessuno di questi offra dei risultati migliori:

- Caso (A, A): fissata l'azione di Giocatore2, il payoff di Giocatore1 vale 3: se egli decidesse di giocare B il suo payoff varrebbe 5; dato che 5 è strettamente maggiore di 3, Giocatore1 viene invogliato a cambiare la sua strategia, di conseguenza questo non è un equilibrio di Nash. Si può osservare che, per le medesime ragioni, anche il Giocatore2 viene invogliato a cambiare tattica, ma questo controllo non è più necessario, avendo già scoperto che non siamo in un equilibrio di Nash;

- Caso (A, B): fissata l'azione di Giocatore2, il payoff di Giocatore1 vale 0: se egli decidesse di giocare B il suo payoff varrebbe 1; dato che 1 è strettamente maggiore di 0, Giocatore1 viene invogliato a cambiare la sua strategia, di conseguenza questo non è un equilibrio di Nash;

- Caso (B, A): fissata l'azione di Giocatore1, il payoff di Giocatore2 vale 0: se egli decidesse di giocare B il suo payoff varrebbe 1; dato che 1 è strettamente maggiore di 0, Giocatore2 viene invogliato a cambiare la sua strategia, di conseguenza questo non è un equilibrio di Nash;

- Caso (B, B):
  - Fissata l'azione di Giocatore1, il payoff di Giocatore2 vale 1: se egli decidesse di giocare A il suo payoff varrebbe 0; dato che 0 non è strettamente maggiore di 1, Giocatore2 non viene invogliato a cambiare la sua strategia;
  - Fissata l'azione di Giocatore2, il payoff di Giocatore1 vale 1: se egli decidesse di giocare A il suo payoff varrebbe 0; dato che 0 non è strettamente maggiore di 1, Giocatore1 non viene invogliato a cambiare la sua strategia.

  Visto che tutti i giocatori coinvolti non sono invogliati a modificare la propria strategia, questo è un equilibrio di Nash.

Un equilibrio di Nash rappresenta una situazione nella quale nessun agente ha interesse a cambiare strategia unilateralmente: esso simboleggia lo scenario in cui il gruppo si viene a trovare se ogni componente fa ciò che è meglio per sè, mira cioè a massimizzare il proprio guadagno a prescindere dalle scelte degli avversari. È importante però notare come un equilibrio di Nash non sia necessariamente la soluzione migliore per tutti i giocatori: nell'esempio illustrato, infatti, Giocatore1 preferirà giocare il profilo di strategia (A, A), che gli permetterà di guadagnare 3 anzichè 1; stesso discorso è valido anche per Giocatore2. Tale comportamento è spiegabile con il fatto che l'equilibrio di Nash è la soluzione migliore per tutti i giocatori nel caso in cui gli avversari non modifichino le proprie strategie: per raggiungere situazioni più convenienti, è indispensabile che una coalizione di giocatori modifichi la propria strategia.

Non tutti i giochi possiedono un equilibrio di Nash (vedi ad esempio il modello di *Figura 2.13*), mentre può accadere che un gioco possa averne più di uno come nel caso del modello in *Figura 2.14*, in cui gli equilibri di Nash sono (A, A) e (B, B).

L'esistenza degli equilibri di Nash all'interno dei giochi è proprio il punto cardine sul quale si basa il teorema di Nash, di cui ne viene riportato di seguito l'enunciato:

Teorema 1 (Teorema di Nash) *Ogni gioco finito ammette sempre almeno un equilibrio di Nash in strategie miste.*

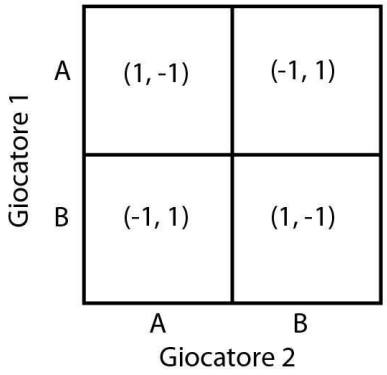

Figura 2.13: *Gioco senza equilibri di Nash*

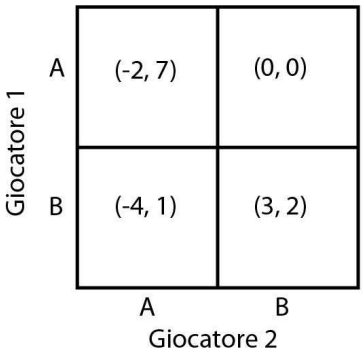

Figura 2.14: *Gioco con diversi equilibri di Nash*

Poichè la maggior parte dei giochi modellizzati soddisfano tale requisito, è spesso possibile prevedere il comportamento dei giocatori: essi giocheranno un equilibrio di Nash e, se esso è unico, il risultato sarà noto a priori.

Equilibri di Nash approssimati

Ma lo scopo di questo lavoro di tesi è di individuare strategie il più vicine possibile agli equilibri di Nash o agli equilibri perfetti per sottogiochi, e quindi è necessario introdurre altri concetti propedeutici alla comprensione del prosieguo del documento.

- **Equilibrio ε–close**: con questo termine si indica un profilo di strategia $\boldsymbol{\sigma}$ in cui la distanza tra ogni σ_i e σ_i^* è minore di un parametro ε, dove con il termine σ_i^* si intende la strategia ottima dell'agente i nell'equilibrio $\boldsymbol{\sigma}$. Questo concetto ci permette di ottenere una misura del grado di approssimazione della strategia trovata anche se, nella maggior parte dei casi, questa grandezza non è considerata sufficiente per valutare appieno la bontà di una soluzione;

- **Equilibrio ε–Nash**: con questo termine si indica un profilo di strategia $\boldsymbol{\sigma}$ dove nessun agente è in grado di migliorare la propria utilità di una quantità superiore a ε, mediante una **deviazione unilaterale**; si ottiene una deviazione unilaterale quando un solo giocatore modifica la propria strategia ($\sigma_i^{NEW} \neq \sigma_i^{OLD}$), mentre i restanti agenti non cambiano il loro comportamento ($\sigma_{-i}^{NEW} = \sigma_{-i}^{OLD}$). Tale concetto costituisce un sensibile miglioramento rispetto all'equilibrio ε–close;

- **Equilibrio ε–perfect**: con questo termine, infine, si indica un raffinamento dell'equilibrio ε–Nash: in questo caso si tiene conto anche di tutte le deviazioni dei giocatori in tutti i sottogiochi che formano la struttura ad albero. Di conseguenza, con il concetto di equilibrio ε–perfect si fa riferimento ad un profilo di strategia $\boldsymbol{\sigma}$ in cui non esiste alcun agente in grado di migliorare la propria utilità di una quantità superiore a ε, anche nel caso in cui tutti i giocatori modifichino la propria strategia.

2.2.5 Risoluzione di giochi in forma estesa

Per risolvere un modello all'interno della teoria dei giochi, è necessario individuare un profilo in cui le strategie dei vari giocatori siano in equilibrio tra di loro.

Poichè tutto il lavoro di ricerca esposto in questo documento verrà impostato sui giochi in forma estesa, di seguito vengono illustrati dei metodi di risoluzione che mirano ad individuare due tipi di equilibri ben distinti: gli equilibri perfetti per sottogiochi e gli equilibri di Nash.

Equilibri perfetti per sottogiochi (SPE)

Il metodo di risoluzione che ricerca questa categoria di equilibri si basa sul principio *divide et impera*, in quanto scompone l'albero di gioco in tutti i sottogiochi possibili, risolvendoli progressivamente e semplificando di volta in volta la struttura generale del modello. Un profilo di strategia è detto SPE se e soltanto se è un equilibrio di Nash per ogni sottogioco del modello originale.

Per un gioco in forma estesa basato su simulazione, per ogni nodo $v \in V$, un equilibrio perfetto per sottogiochi è definito formalmente come

$$\sigma^{SPE}(v) = \arg \max_{\sigma \in \rho(v)} u^{SPE}_{\iota(v)} \left(\chi(v, \sigma)\right), \tag{2.1}$$

in cui $u^{SPE}(v)$ corrisponde a

$$u^{SPE}(v) = \begin{cases} \mathcal{O}(v) & \text{se } v \text{ è un nodo terminale} \\ u^{SPE}\left(\chi\left(v, \sigma^{SPE}(v)\right)\right) & \text{altrimenti.} \end{cases}$$

Il metodo più comune per questo tipo di problemi è quello di agire per **induzione a ritroso**: partendo dal livello più basso del gioco – costituito dalle foglie – si risale progressivamente l'albero, valutando ad ogni nodo quale

sia l'azione migliore da eseguire tra quelle disponibili; una volta definita, si sostituisce il nodo analizzato con una foglia il cui payoff è pari a quello che si otterrebbe eseguendo la strategia trovata. Facendo questo ragionamento per tutti i nodi non–terminali, si risale l'albero fino ad ottenere una struttura con un solo livello.

È importante far notare come in tutti i giochi finiti in forma estesa ad informazione perfetta esista almeno un SPE [4], e tale accorgimento valga anche per tutti i giochi continui con funzioni di utilità limitate [5].

Qui viene riportato un esempio della ricerca di SPE all'interno del gioco in forma estesa illustrato in *Figura 2.6*:

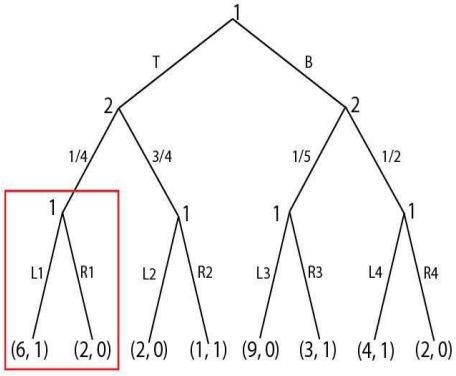

Figura 2.15: *SPE, L'albero iniziale*

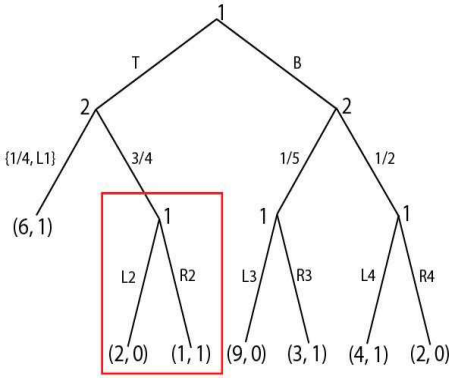

Figura 2.16: *SPE, Step 1*

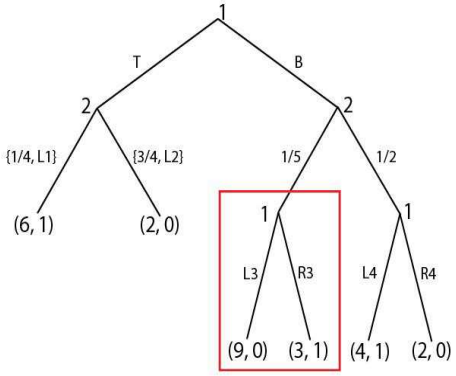

Figura 2.17: *SPE, Step 2*

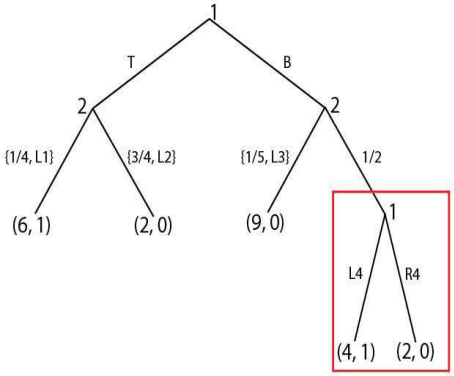

Figura 2.18: *SPE, Step 3*

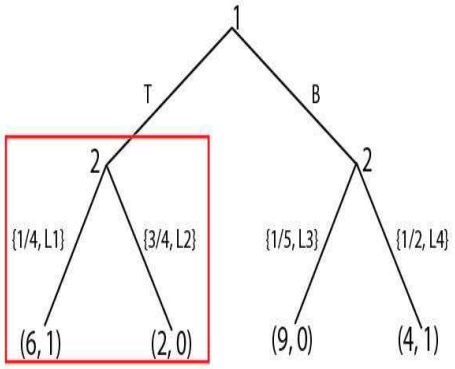

Figura 2.19: *SPE, Step 4*

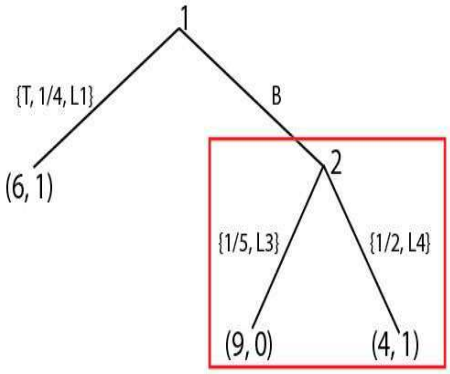

Figura 2.20: *SPE, Step 5*

Procedendo con la risoluzione, si trova che il profilo di strategia corrispondente ad un SPE è $(T, \frac{1}{4}, L_1)$, che restituisce come payoff il vettore $(6, 1)$.

Equilibri di Nash (NE)

Il secondo metodo di risoluzione analizzato in questa sede mira ad individuare gli equilibri di Nash (vedi *Sezione 1*) all'interno dello spazio delle strategie. Innanzitutto è necessario trasformare il gioco in forma strategica (deve cioè essere rappresentato tramite una matrice di payoff). Per fare questo, poichè si sta esaminando un modello in forma estesa, è necessario convertire la struttura ad albero nella corrispondente matrice.

Per definizione, la strategia di ogni agente deve poter prevedere una mossa a qualsiasi nodo decisionale, indipendentemente dal fatto che questo sia raggiungibile in base alle scelte effettuate ai precedenti nodi non–terminali. Per prima cosa perciò è necessario enumerare le strategie pure di ciascun giocatore: nel caso del modello rappresentato in *Figura 2.21* queste corrispondono a

$$\begin{cases} S_1 = \{(a, g); (a, h); (b, g); (b, h)\} \\ S_2 = \{(c, e); (c, f); (d, e); (d, f)\} \end{cases}$$

È stato dimostrato come, nei giochi in forma estesa, qualche equilibrio di Nash possa non essere ragionevole rispetto alla struttura sequenziale del gioco [4]: ad esempio (a, g) e (a, h) non sono attuabili, in quanto la scelta da parte del primo giocatore dell'azione a rende impossibile l'esecuzione delle

azioni g o h. Una volta definite tutte le strategie pure di ciascun giocatore, viene creata la matrice di payoff corrispondente nel seguente modo:

1. La matrice ha tante dimensioni quanti sono i giocatori partecipanti;
2. Le strategie selezionabili da ciascun giocatore corrispondono alle proprie strategie pure trovate in precedenza;
3. L'elemento (i, j) della matrice corrisponde al vettore di utilità ottenuto quando il primo giocatore sceglie la strategia pura i, e il secondo esegue invece j. Le utilità si individuano esplorando l'albero nel senso definito dalle strategie pure i e j.

Ad esempio, l'elemento $((b, g); (c, f))$ della matrice di payoff riportata in *Figura 2.21*, corrisponde alla situazione in cui il primo giocatore esegue l'azione b, l'avversario risponde con f, ed infine il primo giocatore chiude con la mossa g; esplorando l'albero, si nota come il vettore di utilità conseguibile da questo profilo di strategia vale $(2, 10)$, e si inserisce tale dato all'interno dell'elemento $((b, g); (c, f))$ della matrice in *Figura 2.22*.

È possibile affermare che tutti i giochi in forma estesa ad informazione perfetta ammetteranno sempre almeno un SPE, in quanto è stato dimostrato come l'idea di SPE consista in un raffinamento del concetto di NE [4]; tale risultato vale anche per i giochi continui, a patto che la funzione di utilità impiegata sia limitata [5].

Per chiarire meglio il concetto, qui di seguito viene riportato un esempio:

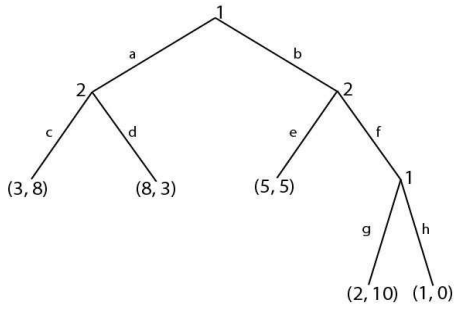

Figura 2.21: *NE, Struttura ad albero*

Giocatore 1 \ Giocatore 2	(c, e)	(c, f)	(d, e)	(d, f)
(a, g)	(3, 8)	(3, 8)	(8, 3)	(8, 3)
(a, h)	(3, 8)	(3, 8)	(8, 3)	(8, 3)
(b, g)	(5, 5)	(2, 10)	(5, 5)	(2, 10)
(b, h)	(5, 5)	(1, 0)	(5, 5)	(1, 0)

Figura 2.22: *NE, Matrice di Payoff corrispondente*

Partendo dalla struttura ad albero di *Figura 2.21*, una volta ottenuta la matrice di payoff corrispondente mediante il procedimento appena illustrato,

si definiscono gli equilibri di Nash con le regole viste nella *Sezione 1*; tale procedimento darà come risultato i seguenti NE:

$$\{(a,g)(c,f);(a,h)(c,f);(b,h)(c,e)\}$$

Dopo aver determinato gli equilibri di Nash all'interno della matrice di payoff è necessario, visto che il gioco di partenza era in forma estesa, rintracciare gli equilibri all'interno della struttura ad albero. In questo senso, a partire dal risultato trovato, si ottengono i seguenti equilibri di Nash per il modello dato: $\{(a,c);(a,c);(b,e)\}$.

È possibile osservare come la conversione appena effettuata produca più volte lo stesso NE e, per questo motivo, è necessario eliminare le soluzioni ridondanti: in conclusione, per il gioco riportato in *Figura 2.21*, esistono due diversi equilibri di Nash, che corrispondono ad $\{(a,c);(b,e)\}$.

Capitolo 3

Algoritmi di ottimizzazione

All'interno della categoria dei giochi in forma estesa, il processo di ricerca di equilibri SPE e NE in un modello è sempre affrontabile come la risoluzione di un problema di ottimizzazione ricorsivo. In questo senso, in ogni nodo v lo scopo è massimizzare la propria funzione di utilità (nel caso dei giochi basati su simulazione questa è sconosciuta agli agenti): tale funzione è strettamente legata alla risoluzione dei problemi di ottimizzazione definiti nei nodi del sottoalbero generato da v. Nel nostro caso, in cui vengono trattati giochi continui in forma estesa basati su simulazione, tale problema consiste nel dover risolvere una quantità indefinita di processi di ottimizzazione non vincolata su variabili continue.

Nella letteratura, gli studiosi hanno suddiviso i metodi di risoluzione dei problemi di ottimizzazione in due grosse categorie indipendenti, differenziando le tecniche utilizzate in:

- **Deterministiche**: in questi metodi di risoluzione, applicando più volte l'algoritmo allo stesso problema, la soluzione sarà sempre la medesima. Un esempio di metodi appartenenti a questa categoria sono il *real algebraic geometry* studiato in [6], e la *Lipschitz optimization* definito grazie a [7];

- **Non–deterministiche** (o *stocastiche*): a differenza del caso precedente, questi metodi di risoluzione hanno al loro interno una componente di casualità (o di probabilità) che ne definisce il non–determinismo. Se applicati iterativamente sullo stesso gioco, molto spesso forniscono soluzioni diverse. Questa categoria è molto più ampia rispetto alla precedente, e si basa su tecniche risolutive quali gli *evolutionary algo-*

rithms [8], il *simulated annealing* [9], il *cross entropy* [10] e il *particle swarm optimization* [11].

Ognuno degli algoritmi di ottimizzazione citati è costruito sulla base di una struttura comune: qui di seguito ne viene proposta una schematizzazione, illustrandone brevemente i passi principali.

Algorithm 1 Algoritmo di ottimizzazione

$i := 0$
$\{x_0^k\}_{1\leq k\leq N} = \text{SAMPLE\_INITIALIZATION}(D)$
repeat
  $i := i + 1$
  $\{x_i^k\}_{1\leq k\leq N} = \text{SAMPLE\_GENERATION}\left(D, \left\{x_{[0:i-1]}^k\right\}_{1\leq k\leq N}, \left\{y_{[0:i-1]}^k\right\}_{1\leq k\leq N}\right)$
  for $j = 1$ to N **do**
    $y_i^j = u(x_i^j)$
  end for
until $\text{TERMINATION\_CONDITION}\left(\left\{x_{[0:i]}^k\right\}_{1\leq k\leq N}, \left\{y_{[0:i]}^k\right\}_{1\leq k\leq N}\right)$
return $\arg\max_{x\in\{x_i^k\}_{1\leq k\leq N}} u(x), u(x)$

La procedura inizia generando una certa quantità di campioni nello spazio di ricerca D e valutandone, di volta in volta, l'utilità (questo dato dipende dal tipo di oracolo che viene utilizzato nel gioco). Ad ogni iterazione, vengono generate nuove soluzioni candidate e, sulla base della loro utilità ed eventualmente di altre informazioni aggiuntive, la ricerca viene diretta verso la porzione di spazio D più promettente.

Il processo di ricerca termina quando viene verificata una determinata condizione di terminazione (ad esempio, alcuni algoritmi di ottimizzazione terminano quando l'incremento ottenuto in una iterazione non è sufficientemente grande, oppure quando viene effettuato un numero di iterazioni troppo elevato).

In questo studio sono stati impiegati tre diversi algoritmi di ottimizzazione, uno di natura deterministica (Lipschitz optimization) e due di tipo non–deterministico (simulated annealing e cross entropy), così da poter analizzare e confrontare l'efficienza di entrambe le categorie presentate. Per semplicità espositiva, nelle prossime sezioni del capitolo verranno presentate le versioni monodimensionali degli algoritmi di ottimizzazione, concentrando cioè l'attenzione sui modelli a giocatore singolo; in realtà, in fase di implementazione il lavoro è stato sviluppato su modelli a due o tre giocatori.

3.1 Simulated annealing

Il **simulated annealing** è una procedura probabilistica, molto conosciuta in ambiente scientifico, mirata ad individuare i massimi e i minimi locali di una funzione (nel nostro caso della funzione di utilità del giocatore, u_i).

Il concetto di *annealing* deriva dalla scienza dei metalli: tale nome indica il processo di eliminazione di difetti reticolari dai cristalli, tramite una procedura di riscaldamento seguita da un lento raffreddamento. Nel caso della teoria dei giochi, gli algoritmi basati su simulated annealing tentano di emulare tale tecnica fisica, allo scopo di determinare il profilo di strategia migliore per il modello esaminato.

L'idea su cui si basa l'algoritmo consiste nell'effettuare diverse perlustrazioni dello spazio di ricerca, abbassando ad ogni iterazione la temperatura del sistema. Nonostante la tecnica di simulated annealing venga spesso impiegata in casi in cui lo spazio di ricerca è discreto, in questo lavoro è stato applicato per problemi continui di ottimizzazione globale [12].

Come accennato, il simulated annealing è un algoritmo non–deterministico, probabilistico: la componente di indeterminatezza deriva dall'introduzione della **distribuzione di Boltzmann**, rappresentata dalla formula:

$$p = \frac{e^{\frac{u_i(s_i \to \bar{s}_i, s_{-i})}{\tau}}}{\sum_{S \in D} e^{\frac{u_i(s_i \to S, s_{-i})}{\tau}}}, \tag{3.1}$$

Tale formula esprime la probabilità p che la strategia $\bar{s}$ del giocatore i–esimo venga selezionata. Le grandezze in gioco sono:

- $u_i(\cdot, \cdot)$ che rappresenta la funzione di utilità dell'agente i–esimo;
- s_i che rappresenta la strategia adottata dal giocatore i;
- s_{-i} che rappresenta la strategia congiunta che adottano tutti i giocatori diversi da i;
- $\bar{s}_i$ che rappresenta la strategia adottata da i di cui si vuole calcolare la probabilità di Boltzmann;
- τ che rappresenta la temperatura a cui è sottoposto il sistema;
- D che rappresenta lo spazio di ricerca su cui vengono definite le strategie dei giocatori.

3.1.1 SA

Alla procedura di simulated annealing implementata ed utilizzata in questo lavoro di tesi, di cui si riporta un estratto del codice nell'*Algoritmo 2*, vengono passati i seguenti parametri: il vettore delle temperature $\mathcal{T}$, il numero di campioni da generare ad ogni iterazione $\mathcal{N}$, il parametro di terminazione ε, la deviazione standard iniziale σ_{init}, la strategia degli avversari s_{-a} ed il parametro rappresentativo del progressivo decremento del valore della deviazione standard γ.

All'avvio della procedura, vengono generati in modo casuale un numero $\mathcal{N}_i$ di campioni da una distribuzione uniforme, salvandoli nel vettore di strategie x_i (dove con i si indica il numero dell'iterazione corrente). Successivamente si valuta l'utilità di tutti gli elementi del vettore, salvando ogni valore così generato all'interno di y_i: la funzione di utilità non è rilevante, in quanto è generata dall'oracolo utilizzato durante il gioco.

A questo punto, si calcola il modo in cui le diverse strategie si distribuiscono all'interno dello spazio, calcolando la probabilità di ogni singolo campione grazie alla *Formula 3.1*; i risultati verranno annotati all'interno di un altro vettore, definito come z_i. Da tali valori verrà poi calcolata la probabilità cumulativa e salvata nel vettore Z_i.

Per definire quale sia la strategia migliore, durante le varie iterazioni è stato opportuno implementare un ulteriore step: una volta definita la probabilità cumulativa, viene scelto un valore casuale, compreso tra 0 e 1, e si individua in Z_i il valore più vicino a tale numero. In questo modo, la strategia estratta sarà considerata come la best–response per l'iterazione corrente.

Una volta terminata la prima iterazione, si ripetono le fasi appena descritte, modificando la dstribuzione statistica dei punti lungo lo spazio di ricerca: invece di estrarre i punti in maniera uniforme, si è scelto di utilizzare come distribuzione di riferimento (la cui definizione è un punto critico per la convergenza del simulated annealing [13]) una gaussiana, con media pari al valore individuato dall'iterazione precedente, e varianza decrescente. Durante l'esecuzione del processo di ricerca, al fine di convergere verso una soluzione, il parametro di temperatura decresce secondo un determinato schema di raffreddamento, definito da γ [14].

L'algoritmo termina quando tra un'iterazione e la successiva non si hanno miglioramenti sostanziali dell'utilità generata, cioè quando il progresso è inferiore rispetto al parametro ε: tale criterio di terminazione è simile a quello

adottato in [15].

Algorithm 2 SA($\mathcal{T}$, $\mathcal{N}$, s_{-a}, σ_{init}, ε, γ)

$i := 1$
$\{x_i^k\}_{1 \leq k \leq \mathcal{N}_i} = rand()$
for $j = 1$ to $\mathcal{N}_i$ **do**
    $y_i^j := u_a(x_i^j, s_{-a})$
end for
for $j = 1$ to $\mathcal{N}_i$ **do**
    $z_i^j := \frac{e^{\frac{u_a(x_i^j, s_{-a})}{\tau_i}}}{\sum_{S \in x_i} e^{\frac{u_a(S, s_{-a})}{\tau_i}}}$
end for
$Z_i^1 := z_i^1$
for $j = 2$ to $\mathcal{N}_i$ **do**
    $Z_i^j := Z_i^{j-1} + z_i^j$
end for
$value := rand()$
for $j = 1$ to $\mathcal{N}_i$ **do**
    if $Z_i^j \geq value$ **then**
        $\mu_i := x_i^j$
        break
    end if
end for
$\sigma_i := \sigma_{init}$
repeat
    $i := i + 1$
    $\{x_i^k\}_{1 \leq k \leq \mathcal{N}_i} = gauss(\mu_{i-1}, \sigma_{i-1})$
    for $j = 1$ to $\mathcal{N}_i$ **do**
        $y_i^j := u_a(x_i^j, s_{-a})$
    end for
    for $j = 1$ to $\mathcal{N}_i$ **do**
        $z_i^j := \frac{e^{\frac{u_a(x_i^j, s_{-a})}{\tau_i}}}{\sum_{S \in x_i} e^{\frac{u_a(S, s_{-a})}{\tau_i}}}$
    end for
    $Z_i^1 := z_i^1$
    for $j = 2$ to $\mathcal{N}_i$ **do**
        $Z_i^j := Z_i^{j-1} + z_i^j$
    end for
    $value := rand()$
    for $j = 1$ to $\mathcal{N}_i$ **do**
        if $Z_i^j \geq value$ **then**
            $\mu_i := x_i^j$
            break
        end if
    end for
    $\sigma_{i+1} := \sigma_i * \gamma$
until $\bar{x}_i - |\bar{x}_i| * \varepsilon > x_i^{MIN}$
return μ_{i+1}, $u_a(\mu_(i+1), s_{-a})$

3.2 Cross entropy

Il **cross entropy** è un metodo di risoluzione dei problemi di ottimizzazione non–deterministico ideato per la prima volta da Reuven Rubinstein nel 2004, all'interno del documento [10]: tale tecnica si avvale di un approccio di tipo Montecarlo applicato a problemi di ottimizzazione combinatori e continui.

Il cross entropy è ben applicabile sia a problemi di ottimizzazione affetti da componenti di rumore (il *problema del commesso viaggiatore*, i *problemi di assegnamento quadratici*, l'*allineamento di sequenze di DNA*), sia a problemi di ottimizzazione globali continui con un elevato numero di massimi e minimi locali.

La struttura base del cross entropy si divide in due fasi:

- Generazione di un insieme di campioni casuali attraverso un meccanismo specifico per il tipo di problema;
- Modifica dei parametri del meccanismo basandosi sui dati prodotti, al fine di generare un insieme migliore di campioni nell'iterazione seguente. Procedendo in tal modo, si cerca di minimizzare la distanza di cross entropy prescelta (come ad esempio la *divergenza Kullback–Lieber* [16]) sulla base delle informazioni ottenute dai campioni più promettenti. In questo modo verranno generati nuovi campioni, situati nella regione più promettente dello spazio di ricerca.

3.2.1 CE

Nell'algoritmo implementato ed utilizzato in questo contesto, vengono passati in ingresso il vettore contenente la quantità di campioni da generare ad ogni iterazione $\mathcal{N}$, il profilo di strategia avversario s_{-a}, la percentuale dei campioni considerati promettenti γ ed il parametro di terminazione ε.

All'inizializzazione dell'algoritmo, come per il simulated annealing, vengono generati in modo casuale $\mathcal{N}_i$ campioni da valutare: tali punti, generati sullo spazio di ricerca a partire da una distribuzione uniforme, saranno poi salvati all'interno del vettore x_i (dove i indica il numero dell'iterazione che, in questo caso, sarà pari a zero). Completato tale vettore, viene calcolata l'utilità di ogni campione mediante l'oracolo, e i risultati vengono salvati nel vettore y_i; successivamente, questa struttura dati viene riorganizzata in ordine decrescente, al fine di mantenere i campioni più promettenti nelle prime posizioni di Y_i.

In seguito, viene calcolata la percentuale dei punti da considerare come elitari (sulla base del valore del parametro γ), creando così il nuovo vettore K_i, contenente i campioni più promettenti, e Z_i in cui saranno salvate le corrispondenti utilità.

Dopo aver così definito i campioni elitari, occorre specificare i parametri da passare all'iterazione seguente, cioè la media e la varianza della distribuzione gaussiana con cui ricercare i nuovi campioni nello spazio delle strategie D: come media viene utilizzato il valor medio del vettore K_i, mentre come varianza si utilizza la deviazione standard di Z_i. A questo punto si ripetono le fasi appena descritte, in cui i campioni verranno generati per mezzo di una gaussiana anzichè tramite una distribuzione uniforme.

L'algoritmo termina nel momento in cui la differenza tra le utilità di taglio di due iterazioni successive è minore del parametro di terminazione ε, vale a dire quando la differenza delle utilità dei peggior campioni elitari è molto piccola.

La procedura, a questo punto, restituirà il campione migliore tra quelli selezionati, vale a dire il primo valore presente all'interno del vettore K_i.

Qui di seguito viene riportata, tramite pseudocodice, l'implementazione dell'algoritmo di cross entropy utilizzato in questa sede.

Algorithm 3 CE(γ, $\mathcal{N}$, s_{-a}, ε)

$i := 1$
$\{x_i^k\}_{1 \leq k \leq \mathcal{N}_i} = rand()$
for $j = 1$ to $\mathcal{N}_i$ **do**
    $y_i^j = u_a(x_i^j, s_{-a})$
end for
$m := 1$
$Y_i = sort(descend, y_i)$
$u_i^{cut} := Y_i^{\lceil \gamma * size(Y_i) \rceil}$
for $j = 1$ to $\mathcal{N}_i$ **do**
    if $y_i^j \geq u_i^{cut}$ **then**
        $Z_i^m = y_i^j$
        $K_i^m = x_i^j$
        $m := m + 1$
    end if
end for
$\sigma_i = \sqrt{var(Z_i)}$
$\mu_i := mean(K_i)$
repeat
    $i := i + 1$
    $\{x_i^k\}_{1 \leq k \leq N} = gauss(\mu_{i-1}, \sigma_{i-1})$
    for $j = 1$ to $\mathcal{N}_i$ **do**
        $y_i^j = u_a(x_i^j, s_{-a})$
    end for
    $m := 1$
    $Y_i = sort(descend, y_i)$
    $u_i^{cut} := Y_i^{\lceil \gamma * size(Y_i) \rceil}$
    for $j = 1$ to $\mathcal{N}_i$ **do**
        if $y_i^j \geq u_i^{cut}$ **then**
            $Z_i^m = y_i^j$
            $K_i^m = x_i^j$
            $m := m + 1$
        end if
    end for
    $\sigma_i = \sqrt{var(Z_i)}$
    $\mu_i := mean(K_i)$
until $|u_i^{cut} - u_{i-1}^{cut}| > \varepsilon$
return $max(K_i)$, $u_a(max(K_i), s_{-a})$

3.3 Lipschitz optimization

Il **Lipschitz optimization** è un approccio di natura deterministica mirato alla risoluzione di problemi di ottimizzazione globale. L'ipotesi fondamentale su cui si basa la procedura implementata riguarda la funzione di utilità del giocatore $u_a(x, s_{-a})$: per poter funzionare correttamente, è necessario che per essa valga la condizione di **continuità di Lipschitz** sull'intervallo chiuso $[A, B]$ (nel nostro caso è stato considerato $[0, 1]$).

In analisi matematica, una **funzione Lipschitziana** è una funzione a variabili reali che ha una crescita limitata, nel senso che il rapporto tra la variazione di ordinata e la variazione di ascissa non può mai superare un valore prefissato, detto **costante di Lipschitz**. Il concetto di continuità di Lipschitz è una condizione più forte rispetto a quello di continuità in matematica, e prende il suo nome dallo studioso tedesco che l'ha definito per la prima volta, Rudolph Lipschitz.

Sia f una funzione tale per cui valga che $f : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}^m$: essa si dice Lipschitziana su Ω se $\exists K \geq 0$ tale per cui $||f(x) - f(y)|| \leq K||x - y||$, $\forall x, y \in \Omega$.

Questa condizione implica che:

- il rapporto incrementale è limitato;
- se una funzione è Lipschitziana è anche continua in Ω, ma non per questo è detto che sia anche derivabile.

L'idea che sta alla base dell'algoritmo consiste nel selezionare i campioni più promettenti sulla base di un limite superiore della funzione di utilità: la miglior strategia disponibile è quella che massimizza tale limite; questo valore viene generato durante il processo di ricerca dei campioni nello spazio.

Per comprendere al meglio l'*Algoritmo 4*, è necessario analizzare a fondo le formule che verranno impiegate, in particolare quelle per ricavare i nuovi campioni x_{NEW} e per calcolarne l'utilità massima prevista z_{NEW}. Per trovare un nuovo punto nello spazio di ricerca, è necessario individuare i due punti più vicini nella frontiera (x_L e x_R), e le loro utilità reali ($u_a(x_L, s_{-a})$ e $u_a(x_R, s_{-a})$)

I dati relativi ai nuovi campioni verranno così calcolati:

$$x_{NEW} = \frac{u_a(x_R, s_{-a}) - u_a(x_L, s_{-a}) + \alpha * (x_R + x_L)}{2 * \alpha}$$

$$z_{NEW} = \alpha * x_{NEW} + u_a(x_L, s_{-a}) - \alpha * x_L$$

dove con α si indica il **grado di Lipschitzianità** della funzione $u_a(x, s_{-a})$, indicante la massima inclinazione che la sua derivata può assumere. Con α elevato, la funzione sale (o scende) in modo molto repentino, mentre se α è piccolo, la funzione ha un gradiente molto meno ripido.

3.3.1 LIP

A differenza delle procedure illustrate nei paragrafi precedenti, questo algoritmo necessita di un numero di input inferiore e, nel nostro caso, questi si limitano al grado di Lipschitzianità α ed al parametro di terminazione ε.

In questo metodo vengono dichiarati ed usati un gran numero di strutture dati; di seguito se ne riporta un elenco completo:

- x: vettore contenente tutti i campioni generati durante il problema di ottimizzazione;
- y: vettore contenente le utilità reali degli elementi di x;
- z: vettore contenente le utilità massime degli elementi di x (è il vettore che racchiude gli upper–bound delle utilità);
- k: ordinamento crescente del vettore x;
- g: ordinamento decrescente del vettore z;
- X: campione più promettente presente nel vettore x, cioè quello che genererà i nuovi campioni;
- Y: utilità reale del punto X;
- Z: upper–bound sull'utilità del punto X.

Considerando il fatto che per generare nuovi campioni è necessario essere in possesso di almeno una coppia di punti, all'inizio l'*Algoritmo 4* utilizza i due estremi del dominio della funzione, vale a dire i campioni 0 e 1. Attraverso le formule appena presentate, viene inserito all'interno di x, z e y un nuovo elemento (rispettivamente il nuovo campione, l'upper–bound della sua utilità e la sua utilità reale). Ogni qualvolta che dei punti in x vengono usati per generare nuovi punti, i corrispettivi elementi in z vengono posti ad un valore negativo molto basso (in questo caso -1000): con questo accorgimento si

impedisce che l'algoritmo tenti di rigenerare un campione già calcolato in precedenza.

Successivamente, si ordinano i vettori x e z, salvando i risultati rispettivamente in k e g: questa fase serve per poter facilmente individuare il campione più promettente, e per riuscire a disporre in ordine crescente i punti dello spazio.

Una volta ordinati i vettori, viene selezionato il primo elemento di g (vale a dire il campione più promettente) e si individuano i due punti più vicini ad esso: con x_L si indica la strategia più prossima che si trova alla sinistra dell'elemento selezionato, con x_R di indica il campione più prossimo che si trova alla sua destra.

A partire da questi, si generano due nuovi elementi da salvare in x: da questo punto in poi si ripete iterativamente la procedura esaminata sopra, fino alla terminazione dell'algoritmo.

La procedura smette di funzionare quando lo scostamento tra l'upper–bound del campione selezionato e la sua utilità reale è minore del parametro ε prefissato.

È opportuno far notare come nello pseudocodice di seguito riportato vi sia una modifica di notazione: se per gli algoritmi precedenti il termine x_i indicava il vettore x all'iterazione i–esima, nel Lipschitz essa indica l'elemento i–esimo del vettore x, essendo presente un'unica iterazione.

Algorithm 4 LIP(α, ε)

$x_1 := 0$
$x_2 := 1$
$y_1 = u_a(x_1, s_{-a})$
$y_2 = u_a(x_2, s_{-a})$
$z_1 = -1000$
$z_2 = -1000$
$x_3 = \frac{y_2 - y_1 + \alpha * (x_2 + x_1)}{2 * \alpha}$
$z_3 = \alpha * x_3 + y_1 - \alpha * x_1$
$y_3 = u_a(x_3, s_{-a})$
$k = sort(ascend, x)$
$g = sort(descend, z)$
repeat
    for $j = 1$ to $size(z)$ **do**
        if $z_j = g_1$ **then**
            break
        end if
    end for
    $X := x_j$
    $Y := y_j$
    $Z := z_j$
    for $m = 1$ to $size(k)$ **do**
        if $k_m = X$ **then**
            $X_L := k_{m-1}$
            $X_R := k_{m+1}$
            break
        end if
    end for
    $Y_L := u_a(X_L, s_{-a})$
    $Y_R := u_a(X_R, s_{-a})$
    $x_{size(x)+1} = \frac{Y - Y_L + \alpha * (X + X_L)}{2 * \alpha}$
    $z_{size(x)} = \alpha * x_{size(x)} + Y_L - \alpha * X_L$
    $y_{size(x)} = u_a(x_{size(x)}, s_{-a})$
    $x_{size(x)+1} = \frac{Y_R - Y + \alpha * (X_R + X)}{2 * \alpha}$
    $z_{size(x)} = \alpha * x_{size(x)} + Y - \alpha * X$
    $y_{size(x)} = u_a(x_{size(x)}, s_{-a})$
    $z_j = -1000$
    $k = sort(ascend, x)$
    $g = sort(descend, z)$
until $Y > Z + \varepsilon$
return X, $u_a(X, s_{-a})$

Capitolo 4

Algoritmi di ricerca di un equilibrio approssimato

Una volta conclusa l'implementazione degli algoritmi di ottimizzazione, ci si è concentrati sulla progettazione e realizzazione delle procedure che costituiscono la parte centrale di questo lavoro di tesi: gli algoritmi di ricerca di un equilibrio approssimato.

L'approssimazione di un equilibrio in un gioco in forma estesa basato su simulazione non è un problema riconducibile agli algoritmi normalmente utilizzati per la risoluzione di giochi in forma strategica; in particolare, utilizzando le procedure presentate in [3], si otterrebbe come effetto la perdita della sequenzialità del gioco: se infatti volessimo utilizzare ugualmente tali algoritmi, sarebbe necessario rappresentare le strategie dei singoli giocatori come un piano di azioni, ottenendo l'effetto di modificare così la struttura del modello.

Per questo motivo, in questo studio, sono stati progettati degli algoritmi *ad–hoc* che potessero lavorare direttamente sulla struttura ad albero del gioco: in quest'ottica, sono stati utilizzati due diversi approcci per risolvere i problemi di approssimazione, basati sulla ricerca di SPE e di NE all'interno del gioco analizzato.

Le procedure qui illustrate sono riconducibili l'una all'altra, in quanto si può considerare il NE–Approximation come un'estensione del SPE–Approximation, in cui intervengono dei meccanismi di potatura e di semplificazione della struttura ad albero del gioco. In questa sede, verrà prima presentata la procedura di SPE–Approximation, per poi passare ad illustrare il funzionamento del NE–Approximation.

In entrambi i casi, verrà utilizzata una notazione leggermente diversa rispetto a quella fin qui presentata: in questo capitolo infatti, per indicare la strategia del giocatore $\iota(v)$ nel nodo v si userà il simbolismo $\sigma(v)$.

4.1 SPE–Approximation

La prima procedura realizzata in questa ottica è il **SPE–Approximation**, in cui la struttura ad albero del gioco viene visitata in tutti i suoi rami, valutando ogni possibile profilo di strategia realizzabile.

L'algoritmo viene inizialmente invocato sulla radice della struttura ad albero: per esplorare anche i livelli sottostanti, la procedura richiama sè stessa in modo ricorsivo, ottenendo così l'analisi e l'ottimizzazione di tutti i nodi sottostanti.

Da quanto detto, emerge chiaramente come questa tecnica di risoluzione lavori con una logica *bottom–up*, in quanto vengono prima calcolate le utilità nelle foglie e, successivamente, i valori ricavati vengono propagati verso l'alto, eseguendo un'esplorazione dell'albero in profondità.

Nel SPE–Approximation non intervengono operazioni di potatura, in quanto è obbligatorio visitare tutti possibili profili di strategia per poter trovare l'equilibrio desiderato, comportando così una complessità computazionale ed un tempo di esecuzione molto elevati.

4.1.1 Algoritmo SPE–Approximation

Quando viene invocato l'algoritmo di SPE–Approximation, di cui se ne riporta una traccia in pseudocodice, è necessario passare come parametro il nodo corrente v, vale a dire il turno in cui attualmente si trovano gli agenti coinvolti nel gioco.

È interessante notare come, di fatto, la struttura di questa procedura sia molto simile agli algoritmi presentati nel *Capitolo 3*: l'unica modifica significativa è relativa al calcolo delle utilità degli agenti.

Ogni iterazione inizia con la campionatura dello spazio di ricerca, in modo analogo agli algoritmi SA, CE e LIP visti in precedenza ma, al momento di valutare l'utilità dei singoli campioni, si introduce una piccola variazione:

- se il nodo corrente v è una foglia, la valutazione del payoff è uguale a quella usata negli *Algoritmi 2, 3 e 4*, viene cioè calcolata dall'oracolo $\mathcal{O}(v)$;

- se il nodo corrente v non è una foglia, l'utilità viene calcolata mediante la chiamata ricorsiva di SPE_Approximation, in cui verrà passato come parametro $\chi(v, \sigma^k(v))$, vale a dire il nodo successivo a v sulla base della strategia $\sigma^k(v)$.

Una volta giunti ai nodi terminali dell'albero di gioco (le foglie), tramite gli algoritmi di ottimizzazione si individua la best-response e le utilità generate da tale profilo di strategia; successivamente si propaga al nodo genitore i dati così ricavati: a sua volta il giocatore avvierà un'ottimizzazione nello spazio di ricerca, propagando i risultati al nodo sovrastante, procedendo in modo iterativo fino alla radice.

Nel frammento di pseudocodice riportato, compaiono numerosi simboli. Per favorirne la comprensione, è opportuno ricordare il significato di tali notazioni:

- $\boldsymbol{v}$ indica il nodo corrente in cui si trova il gioco, ovvero il turno in cui il giocatore deve fare la sua mossa;
- $\boldsymbol{\mathcal{O}(v)}$ indica il valore di utilità generata dall'oracolo $\mathcal{O}$ per il nodo v;
- $\boldsymbol{\sigma^k(v)}$ indica i campioni in cui è stato suddiviso lo spazio di ricerca al nodo v durante l'iterazione k–esima;
- $\boldsymbol{\rho(v)}$ indica le azioni a disposizione del giocatore nel nodo v;
- $\boldsymbol{u^k}$ indica il vettore di utilità ottenuto seguendo il k–esimo profilo di strategia;
- $\boldsymbol{\chi(v, \sigma^k(v))}$ indica il nodo successivo a v, nel caso in cui si segua il profilo di strategia $\sigma^k(v)$;
- $\boldsymbol{u_{\iota(v)}}$ indica l'utilità ottenibile dall'agente che gioca nel nodo v.

Inizialmente, l'*Algoritmo 5* viene invocato a partire dalla radice v: se tale nodo corrisponde ad un terminale, viene immediatamente restituito il valore di utilità calcolato dall'oracolo.

Se tale condizione non è verificata, la strategia ottimale viene calcolata grazie al seguente procedimento: inizialmente lo spazio di ricerca viene campionato in modo uniforme o gaussiano grazie alla funzione SAMPLE_INITIALIZATION. In seguito, per ogni campione generato $\sigma^k(v)$ viene valutata la

sua utilità invocando ricorsivamente la procedura SPE_Approximation, passando come parametro il nodo successivo a v.

Una volta terminata la valutazione di tutti i campioni, se la condizione di terminazione è soddisfatta, l'algoritmo restituisce l'utilità massima ottenuta dai vari campioni; se ciò non è verificato, vengono rigenerati i campioni sullo spazio di ricerca, e si ripete la procedura descritta. La valutazione della condizione di terminazione è garantita dall'invocazione della funzione termination_condition.

Le procedure SPE_Approximation, sample_initialization e termination_condition sono state implementate sulla base degli algoritmi di ottimizzazione non lineare visti nel *Capitolo 3*, ma sono del tutto sconosciute ai giocatori partecipanti: possiamo anzi dire che sono delle vere e proprie black–box per gli agenti.

Algorithm 5 SPE_Approximation(v)

if v is terminal **then**
  return $\mathcal{O}(v)$
else
  $\{\sigma^k(v)\} \leftarrow$ sample_initialization$(\rho(v))$
  while true **do**
    for each $\sigma^k(v)$ **do**
      $u^k =$ SPE_Approximation$(\chi(v, \sigma^k(v)))$
    end for
    if termination_condition $(\{\sigma^k(v)\}, \{u^k\})$ **then**
      return $\arg \max_{u \in \{u^k\}} u_{\iota(v)}$
    else
      $\{\sigma^k(v)\} \leftarrow$ samples_generation$(\rho(v), \{\sigma^k(v)\}, \{u^k\})$
    end if
  end while
end if

4.2 NE–Approximation

A differenza di quanto visto per il SPE–Approximation, la ricerca di equilibri di Nash approssimati è molto meno dispendiosa e più veloce, in quanto intervengono delle potature che permettono di scartare a priori determinate strategie, risparmiando così calcoli inutili e complicati: possiamo perciò sostenere che il concetto di equilibrio di Nash sia caratterizzato da una presenza di vincoli meno rigidi rispetto al concetto di equilibrio per sottogiochi perfetti. Nonostante ciò, l'algoritmo **NE–Approximation** si è rivelato di

fatto molto più macchinoso rispetto al SPE–Approximation: è stato infatti necessario ideare due differenti procedure che si invocassero a vicenda per poter implementare correttamente l'algoritmo di NE–Approximation.

A differenza di quanto fatto nel SPE–Approximation, in cui non è stato necessario definire tecniche diverse per il calcolo degli equilibri SPE, per individuare gli NE si sono dovuti definire due diversi approcci in base al nodo in esame:

- se questo si trova sul percorso di equilibrio (cioè il profilo di strategia che porta ad un NE), l'equilibrio viene calcolato tramite la stessa formula utilizzata per il SPE–Approximation, ovvero massimizzando le proprie utilità;

$$\sigma^{NE}(v) = \arg \max_{\sigma \in \rho(v)} u^{NE}_{\iota(v)} \left(\chi(v, \sigma)\right), \tag{4.1}$$

- se è posto invece su percorsi diversi da quello di equilibrio, gli agenti non cercheranno di massimizzare la propria utilità, ma invece tenderanno a minimizzare quella che guadagnerebbe il primo agente.

Differentemente da quanto visto all'interno di SPE–Approximation, nei problemi di NE–Approximation non è garantito che tutti i giocatori agiscano in maniera razionale: fuori dall'equilibrio, cioè nei nodi che non vengono raggiunti, le strategie possono essere libere. Questo comportamento è dovuto al fatto che, durante la ricerca degli equilibri di Nash, l'obiettivo di ogni giocatore è quello di trovare un profilo di strategie tale per cui ogni agente avversario non sia invogliato a modificare le proprie azioni: per far questo, è sufficiente minimizzare sistematicamente i guadagni dei giocatori avversari. Se infatti si cercasse di ricavare la minima utilità per i restanti agenti, questi non sarebbero tentati ad abbandonare la situazione di equilibrio trovata, garantendo così la validità dei NE trovati. Nella ricerca di equilibri perfetti per sottogiochi, ovvero nel SPE–Approximation, questo atteggiamento non risulta, in quanto ogni componente cerca solo e soltanto di guadagnare il massimo possibile, indipendentemente dalle scelte effettuate dagli avversari: non ha senso quindi cercare di minimizzare le utilità altrui perchè l'importante è ottenere il maggiore payoff possibile per sè stessi, e quindi la miglior strategia possibile consiste nel massimizzare il proprio guadagno.

4.2.1 Algoritmo NE–Approximation

La ricerca di equilibri di Nash approssimati è suddivisibile in due fasi distinte e, di conseguenza, devono essere implementate due funzioni differenti:

- NE_Finding in cui viene individuato un profilo di strategia $\boldsymbol{\sigma}$ e successivamente elevato a candidato NE;
- NE_Validation in cui viene verificata la correttezza del candidato NE trovato in precedenza. Se una strategia al di fuori dell'equilibrio si rivela migliore, allora quest'ultima verrà eletta a nuovo candidato NE, e si rilancerà NE_Validation per verificarne la bontà della soluzione.

Tali funzioni interagiscono tra di loro e, in particolare, l'algoritmo parte lanciando NE_Finding, che successivamente invocherà la funzione NE_Validation; una volta invocata quest'ultima, la procedura NE_Finding non verrà più coinvolta, lasciando che NE_Validation richiami sè stessa in maniera ricorsiva fino al termine della ricerca di NE approssimati per il gioco.

NE_Finding

La prima fase dell'algoritmo di NE–Approximation consiste nell'individuare il primo candidato NE, verificandone successivamente la correttezza mediante l'invocazione della procedura NE_Validation.

Nella descrizione che segue, viene introdotto per la prima volta il concetto di **nodo preterminale**: con questo termine si vuole indicare tutti quei nodi v in cui, una volta applicata una qualsiasi azione a v, viene raggiunto un nodo terminale.

Questa prima fase della procedura si occupa di individuare una strategia per ogni nodo appartenente al percorso di equilibrio, creando così un profilo di strategia $\boldsymbol{\sigma}$ completo che possa essere un NE candidato: questo compito è affidato alla funzione NE_Finding, a cui viene passato in ingresso il nodo corrente v di cui è necessario determinare la strategia.

All'interno dell'*Algoritmo 6*, ad ogni nodo non–preterminale v viene assegnata una strategia casuale $\sigma(v)$ e l'algoritmo viene ricorsivamente invocato sul nodo successivo che viene raggiunto applicando la strategia $\sigma(v)$ al nodo corrente v.

Nel caso in cui la condizione non sia verificata, cioè se il nodo v è preterminale, NE_Finding individua la miglior strategia per il giocatore $\iota(v)$ che

gioca in v ricorrendo alle tecniche di ottimizzazione presentate nel *Capitolo 3*.

È interessante osservare come la parte di ottimizzazione funzioni esattamente come le righe 6–15 dell'algoritmo di SPE–Approximation, a parte per il fatto che la valutazione dei campioni non è più eseguita tramite chiamate ricorsive della procedura, ma vengono calcolate invocando direttamente l'oracolo $\mathcal{O}(v)$.

Una volta conclusa la riga 20, viene definito un profilo di strategia $\boldsymbol{\sigma}$ completo che va dalla radice fino ad un nodo terminale della struttura ad albero del gioco considerato.

Una caratteristica fondamentale di tale procedura consiste nel fatto che durante l'esecuzione non sono stati visitati vari percorsi, ma il candidato NE è stato trovato calcolato attraverso un'unica strada.

Algorithm 6 NE_Finding (v)

```
 1: if v is non–preterminal then
 2:     σ(v) ← a random value uniformly distributed in ρ(v)
 3:     u ← NE_Finding(χ(v, σ(v)))
 4:     while true do
 5:         [û, σ̂(v)] ← NE_Validation(v, v)
 6:         if û_ι(v) ≤ u(v) then
 7:             return u
 8:         else
 9:             σ(v) ← σ̂(v)
10:             u ← NE_Finding(χ(v, σ(v)))
11:         end if
12:     end while
13: else
14:     {σ^k(v)} ← sample_initialization(ρ(v))
15:     while true do
16:         for each σ^k(v) do
17:             u^k = O(χ(v, σ^k(v)))
18:         end for
19:         if termination_condition({σ^k(v)}, {u^k}) then
20:             return arg max_{u∈{u^k}} u_ι(v)
21:         else
22:             {σ^k(v)} ← samples_generation(ρ(v), {σ^k(v)}, {u^k})
23:         end if
24:     end while
25: end if
```

NE_Validation

La seconda fase dell'algoritmo di NE–Approximation ha il compito di confermare la candidatura del profilo di strategia $\boldsymbol{\sigma}$ trovato al termine della

procedura di NE_Finding e, nel caso tale risultato non venisse validato, di trovare un nuovo candidato NE da verificare.

A differenza dell'*Algoritmo 6*, alla funzione NE_Validation vengono passati in ingresso il nodo corrente v ed il nodo dell'albero di cui si sta facendo la validazione della strategia v_0: questo input è fondamentale in quanto permette di distinguere tra giocatori massimizzatori e minimizzatori di utilità (se $v_0 = v$, allora l'agente $\iota(v)$ massimizzerà la propria utilità, altrimenti il giocatore cercherà di minimizzare l'utilità di $\iota(v_0)$).

Dato un nodo v, l'*Algoritmo 7* ricerca una strategia all'interno dei sottogiochi in cui l'agente $\iota(v)$ non potrà mai guadagnare di più deviando dalla sua strategia definita da $\boldsymbol{\sigma}$.

Visto che gli agenti tendono a minimizzare l'utilità del giocatore $\iota(v)$ che muove al nodo v, l'obiettivo di ogni iterazione è quello di individuare il **valore maxmin** per l'agente $\iota(v)$ da ogni sottogioco che compone l'albero: se il valore così trovato è strettamente maggiore dell'utilità ottenibile giocando la strategia $\boldsymbol{\sigma}$, allora $\boldsymbol{\sigma}$ non è effettivamente un NE; se invece questo non accade, significa che esiste almeno una strategia nel sottogioco per cui $\boldsymbol{\sigma}$ è ottima. Se in un nodo v viene trovato che il candidato NE non è confermato, la procedura assegna la strategia maxmin del sottogioco a $\sigma(v)$, invocando poi ricorsivamente NE_Validation per verificarne la validità.

Per essere efficace, è necessario che il processo di validazione venga eseguito su tutti i nodi che compongono il percorso di equilibrio trovato, verificando quindi per ogni turno la validità di $\boldsymbol{\sigma}$.

Il funzionamento dell'*Algoritmo 7* ricorda molto quello di SPE_Approximation, tranne per la particolarità che in NE_Validation compare anche la possibilità di minimizzare le utilità altrui.

Le tecniche di ottimizzazione sono sempre quelle presentate nell'*Algoritmo 5* con la differenza che, nel caso di minimizzazione, viene passato come parametro la funzione di utilità $\{-u^k_{\iota(v_0)}\}$.

Un metodo per migliorare l'efficienza e la velocità del problema di approssimazione di equilibri NE consiste nell'introdurre il concetto di **potatura**: con questo accorgimento si permette al calcolatore la possibilità di escludere a priori determinati percorsi, sulla base dei risultati parziali fin lì ottenuti. L'opera di pruning che verrà qui introdotta è molto simile alla **potatura alfa–beta**: chiamando v_0 il nodo la cui strategia è da validare, per poter affermare che un profilo di strategia $\boldsymbol{\sigma}$ non corrisponde ad un NE, non è necessario calcolare il valore maxmin di v_0, ma è sufficiente trovare una strategia

$\hat{\sigma}_{\iota(v_0)}$ tale per cui l'utilità del giocatore $\iota(v_0)$ sia maggiore di quella prodotta dalla strategia analizzata. Inoltre, in tutti i sottogiochi di v_0, non è strettamente necessario che gli avversari trovino la minima utilità del giocatore $\iota(v_0)$, ma è sufficiente che essi individuino un'azione tale per cui l'utilità di $\iota(v_0)$ non sia più grande di quella fornita dalla strategia da controllare.

Questa ottimizzazione è stata adottata in fase di implementazione, però non compare nello pseudocodice dell'*Algoritmo 7*, in quanto non è una componente essenziale al fine della sua comprensione.

Grazie a queste considerazioni, le condizioni di terminazione presenti alle righe 10 e 16 possono essere così interpretate:

- Riga 10: non appena un campione genera un guadagno all'agente $\iota(v_0)$ maggiore di quello fornito dalla strategia $\boldsymbol{\sigma}$, l'algoritmo restituisce tale valore ed il campione che l'ha generato;

- Riga 16: non appena un campione genera un guadagno all'agente $\iota(v_0)$ non superiore a quello fornito dalla strategia $\boldsymbol{\sigma}$, l'algoritmo restituisce tale campione, che entrerà a far parte del nuovo candidato NE.

Algorithm 7 NE_Validation (v, v_0)

if v is terminal **then**
  return $\mathcal{O}(v)$
else
  $\{\sigma^k(v)\} \leftarrow$ sample_initialization$(\rho(v))$
  while true **do**
    for each $\sigma^k(v)$ **do**
      $u^k =$ NE_Validation$\big(\chi(v, \sigma^k(v)), v_0\big)$
    end for
    if $\iota(v) = \iota(v_0)$ **then**
      if termination_condition$\Big(\{\sigma^k(v)\}, \{u^k_{\iota(v)}\}\Big)$ **then**
        return $[\arg \max_{u \in \{u^k\}} u_{\iota(v)}, \text{best } \sigma^k(v)]$
      else
        $\{\sigma^k(v)\} \leftarrow$ samples_generation $\Big(\rho(v), \{\sigma^k(v)\}, \{u^k_{\iota(v)}\}\Big)$
      end if
    else
      if termination_condition$\Big(\{\sigma^k(v)\}, \{-u^k_{\iota(v_0)}\}\Big)$ **then**
        return $[\arg \min_{u \in \{u^k\}} u_{\iota(v_0)}, \text{best } \sigma^k(v)]$
      else
        $\{\sigma^k(v)\} \leftarrow$ samples_generation$\Big(\rho(v), \{\sigma^k(v)\}, \{-u^k_{\iota(v_0)}\}\Big)$
      end if
    end if
  end while
end if

Capitolo 5

Testing e analisi dei risultati

Concluse le fasi di progettazione ed implementazione degli algoritmi, lo step successivo è consistito nell'applicare a dei giochi le procedure create, permettendo così di valutarne l'efficacia, la robustezza e la velocità. Nel presente capitolo verranno illustrati gli esperimenti eseguiti durante questa fase di testing, raccogliendo i risultati generati all'interno di tabelle; verrà poi proposta un'approfondita analisi critica di tali valori.

5.1 Giochi di contrattazione

Poichè questo lavoro di tesi mira a trovare applicazione all'interno della realtà, durante la fase di testing le procedure di approssimazione degli equilibri sono state applicate ad un modello che potesse ricalcare quanto più possibile la vita reale: in questo senso, tale scelta è ricaduta su una tipologia di modelli non ancora affrontata durante questo documento, i **giochi di contrattazione**. Tali giochi cercano di modellizzare tutte quelle situazioni in cui due o più individui contrattano tra di loro per l'acquisizione di una determinata risorsa: solitamente, esiste un unico soggetto che mette a disposizione una risorsa (il **venditore**), ed altri agenti (i **compratori**) che cercano di acquistarla, mediante un processo di negoziazione. Tale categoria di giochi ha da sempre esercitato una forte attrattiva verso la comunità scientifica, fin dalla nascita della teoria dei giochi: inizialmente nati per riprodurre un ben preciso scenario economico, questi modelli furono successivamente analizzati nel dettaglio da Nash in [1], [17], [18] e [19]. In modelli di questo genere possono partecipare più giocatori contemporaneamente, ma la cosa fondamentale è

che ci sia sempre e solo un venditore, mentre il numero di compratori può variare a seconda della situazioni: per semplicità, in questa sede è stato realizzato un gioco di contrattazione formato da un solo venditore e da un solo compratore.

La dinamica del processo di negoziazione utilizzato in questo studio è riassumibile nei seguenti passi:

1. Il venditore fa un'offerta di vendita, aprendo così di fatto le contrattazioni;

2. Il compratore può accettare l'offerta fattagli dal venditore, oppure può rilanciare effettuando una propria offerta;

3. Il venditore, allo stesso modo, può accettare la proposta del compratore oppure può a sua volta rilanciare. In questo caso la parola torna al compratore, e si ricomincia dal *Punto 2*;

4. L'asta termina quando uno dei due giocatori accetta la proposta fatta dall'avversario.

Per simulare al meglio una possibile situazione reale, visto che la maggior parte dei processi di contrattazione riguardano risorse ad alto rischio di deterioramento, nei giochi di contrattazione le utilità guadagnate dai giocatori sono strettamente legate ai **fattori di sconto**; tali grandezze influiscono direttamente sui payoff ottenuti, e simboleggiano la tendenza al deterioramento della risorsa: queste costanti sono direttamente proporzionali alla perdita di utilità che ogni turno di gioco comporta.

In *Figura 5.1* è riportata la forma della funzione di utilità tipica che il venditore cerca di massimizzare: come si può notare, il payoff inizialmente ha un andamento costante pari a 0, per poi subire una brusca impennata.

Il punto di massimo globale (cerchiato in rosso nella figura) è l'obiettivo del problema di approssimazione dell'equilibrio: esso simboleggia l'istante in cui al venditore conviene accettare l'ultima offerta pervenutagli; se questo non avviene, il suo payoff decresce in modo repentino, tendendo nuovamente a 0.

Un'altra particolarità di questa categoria di giochi consiste nella presenza di un limite alla durata della contrattazione: esattamente come per le negoziazioni reali, tutti i giocatori hanno un limite di tempo entro cui possono rilanciare alle offerte degli avversari; dopo tale tempo, gli agenti si trovano

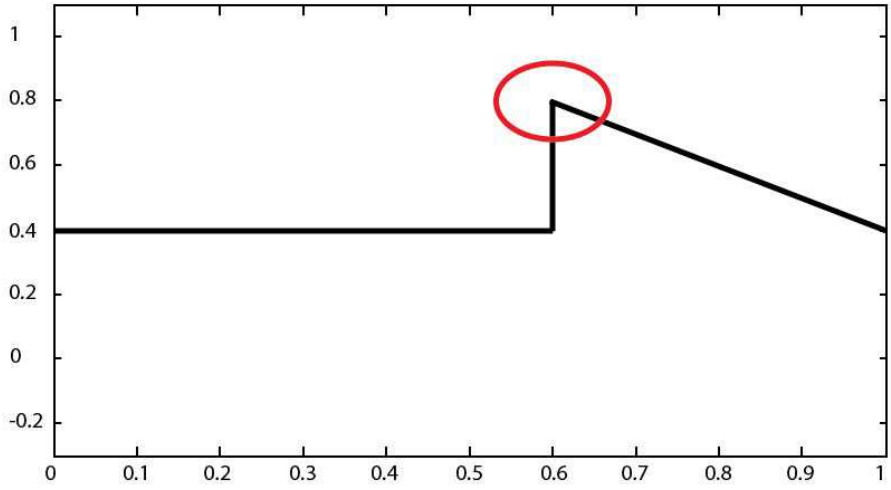

Figura 5.1: *Utilità del venditore nei giochi di contrattazione*

costretti ad accettare qualsiasi proposta, guadagnando di conseguenza un payoff decisamente più basso rispetto ai turni precedenti. Tale limite di tempo, indicato come $\bar{\boldsymbol{T}}$, viene calcolato come il minimo tra i tempi di ogni singolo giocatore coinvolto, e porta ad ottenere il peggior esito possibile a cui si possa giungere.

Nei giochi di contrattazione, ogni giocatore può eseguire al tempo t le seguenti strategie:

- se $t = 0$, cioè nel caso in cui sia il primo turno, il venditore può
  - abbandonare l'asta (**E**), generando un guadagno pari a

$$\begin{cases} u_{V,0} = 0 \\ u_{C,0} = 0 \end{cases}$$

  - presentare al compratore l'offerta x_0.
- se $0 < t \leq \bar{\boldsymbol{T}}$ ogni giocatore può
  - abbandonare l'asta (**E**), generando un guadagno pari a

$$\begin{cases} u_{V,t} = 0 \\ u_{C,t} = 0 \end{cases}$$

– accettare l'ultima offerta fatta (**A**), generando così un guadagno pari a

$$\begin{cases} u_{V,t} = (1 - x_{(t-1)}) * {\delta_V}^t \\ u_{C,t} = x_{(t-1)} * {\delta_C}^t \end{cases}$$

– presentare al compratore (o al venditore, in base a chi sta giocando) l'offerta x_t;

- se $t > \bar{\boldsymbol{T}}$ ogni giocatore può solamente accettare l'ultima proposta presentata (**A**), generando così un guadagno pari a

$$\begin{cases} u_{V,t} = -1 \\ u_{C,t} = -1 \end{cases}$$

5.1.1 Casi di studio

Durante questo studio, gli algoritmi sono stati testati su differenti parametrizzazioni, così da poter misurare la robustezza delle procedure realizzate: tutti i casi d'uso misurati propongono un modello composto da due giocatori (in cui il giocatore iniziale è il venditore) con $\varepsilon = 0.01$.

Di seguito vengono inseriti tutti i casi sottoposti al calcolatore durante questa fase di testing:

- Caso A) $T_V = 2$, $T_C = 4$; $\delta_V = 0.8$ e $\delta_C = 0.75$;
- Caso B) $T_V = 6$, $T_C = 2$; $\delta_V = 0.1$ e $\delta_C = 0.15$;
- Caso C) $T_V = 1$, $T_C = 2$; $\delta_V = 0.1$ e $\delta_C = 0.8$;
- Caso D) $T_V = 2$, $T_C = 8$; $\delta_V = 0.9$ e $\delta_C = 0.2$;

In *Figura 5.2* viene proposto uno schema esemplificativo dello scenario utilizzato durante questa fase, così da facilitare la comprensione del meccanismo di gioco da parte del lettore.

In questa immagine, si è convenuto di indicare ogni turno con la tupla (i, t), dove con i si intende il giocatore che deve giocare nel turno corrente, che può corrispondere a **V** (nel caso del venditore) o a **C** (nel caso del compratore), mentre con t viene indicato il tempo in cui la mossa deve essere eseguita.

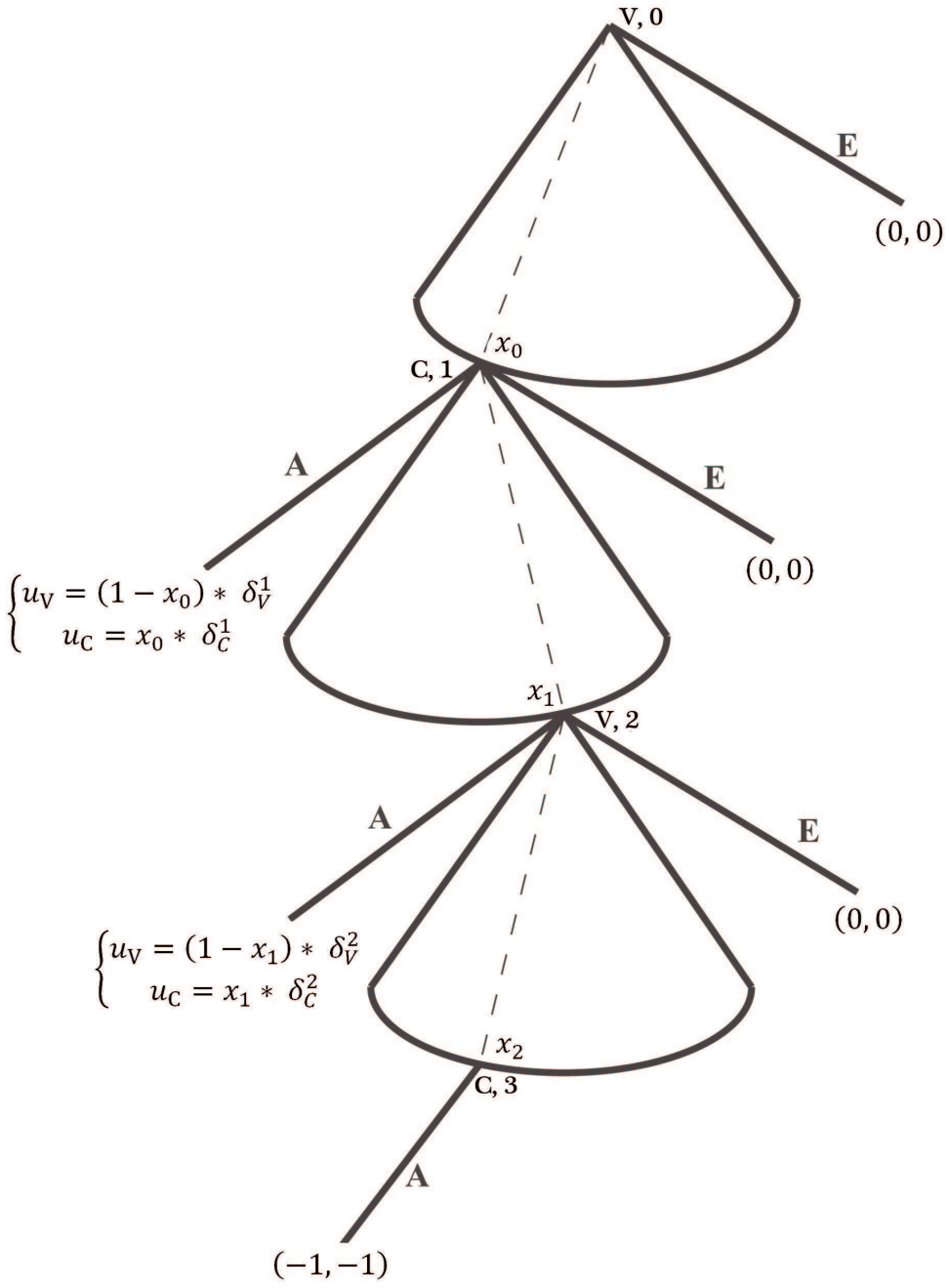

Figura 5.2: *Esempio di gioco di contrattazione*

5.2 Metodo di testing

Al fine di testare in maniera efficace gli algoritmi di ricerca di un equilibrio presentati in precedenza, le procedure SPE–Approximation e NE–

Approximation sono state sottoposte al medesimo modello, così da poter confrontare e paragonare tra di loro i diversi risultati. Ogni algoritmo è stato eseguito utilizzando tutte le procedure di ricerca della best–response realizzate durante questo lavoro di tesi: tale accorgimento è stato adottato per poter effettuare un doppio confronto, che comparasse tra di loro sia i metodi di ricerca di un equilibrio approssimato (SPE–Approximation e NE–Approximation), sia gli algoritmi di ottimizzazione (SA, CE e LIP).

L'operazione di testing si divide essenzialmente in due fasi: prima sono state effettuate le misurazioni per il problema di ricerca di un equilibrio SPE, utilizzando tutte le procedure di ricerca della best–response implementate in precedenza; una volta raccolti tutti i dati in tabelle riassuntive, tale procedura è stata replicata per la ricerca di equilibri NE, generando così altri risultati.

Una volta definiti i parametri del modello, è stato necessario focalizzarsi sul testing delle procedure di ottimizzazione, individuando i parametri che maggiormente influiscono sulle loro prestazioni e sulla loro precisione: è stato necessario differenziare tali parametri sulla base della tecnica di ottimizzazione adottata. Di seguito viene proposta una schematizzazione in cui, per ogni metodo di ricerca della best–response utlizzato, vengono riportate le variabili più significative:

- Simulated annealing: per questo metodo le variabili più interessanti sono state rintracciate nel numero di campioni che vengono generati ad ogni iterazione $\mathcal{N}$ e nel parametro con cui la deviazione standard si riduce ad ogni passo γ;

- Cross entropy: in questo caso i parametri più significativi sono stati individuati nel numero di campioni che vengono generati ad ogni iterazione della procedura $\mathcal{N}$ e nel numero di campioni promettenti γ;

- Lipschitz optimization: per tale tecnica è stato invece individuato solo un parametro realmente influente sull'esito della ricerca della best–response e la scelta è ricaduta sul grado di Lipschizianità α.

Per garantire la robustezza e la validità dei risultati conseguiti in questa fase, per entrambi gli algoritmi di ricerca della best–response di natura non–deterministica (simulated annealing e cross entropy) le varie misurazioni sono state ottenute dalla media di 30 esecuzioni.

Invece, nel caso del metodo Lipschitz optimization, tale accorgimento non è stato necessario per il carattere deterministico dell'algoritmo stesso: è infatti inutile lanciare più volte lo stesso problema, in quanto il risultato che ne deriverebbe sarebbe sempre il medesimo.

Per ogni esecuzione, sono state selezionate tre diverse grandezze per misurare le prestazioni in maniera efficace:

- e: indica lo scostamento della soluzione trovata rispetto al valore teorico che la procedura avrebbe dovuto restituire;
- t: indica il tempo, misurato in secondi, necessario affinchè la procedura restituisca l'equilibrio approssimato;
- s: indica il numero di campioni generati prima di giungere al risultato finale.

5.3 Risultati

Visto e considerato che in questa fase si è voluto effettuare un doppio confronto, valutando contemporaneamente tra di loro sia le procedure di best–response (simulated annealing, cross entropy e Lipschitz optimization) che quelle di approssimazione di un equilibrio (NE e SPE), per maggiore chiarezza verrà prima eseguita una comparazione tra i metodi di ricerca di un equilibrio approssimato (a parità di metodo di ottimizzazione) e quindi un confronto degli algoritmi di ricerca della best–response descritti all'interno del *Capitolo 3*. In questo modo si sono valutate prima le prestazioni e l'efficacia degli algoritmi di approssimazione di un equilibrio, e quindi l'analisi si è focalizzata sulle procedure di ottimizzazione.

In questo capitolo non verranno riportati i dati raccolti durante la fase di testing, ma ci si limiterà a riportare le conclusioni riguardanti l'efficacia e la velocità di esecuzione degli algoritmi implementati in questa sede. I valori numerici dei test effettuati sono comunque inseriti all'interno dell'*Appendice A*, e sono corredati da grafici al fine di evidenziare l'andamento delle tre grandezze analizzate: e, t e s.

5.3.1 Algoritmi di approssimazione

Dai dati raccolti all'interno delle tabelle contenute nell'*Appendice A*, risulta evidente come entrambe le procedure implementate siano delle valide

soluzioni per individuare un equilibrio approssimato all'interno dei giochi in forma estesa basati su simulazione. Analizzando i valori raccolti, emerge chiaramente come le strategie individuate durante la fase sperimentale convergano sempre verso l'equilibrio teorico: in tutte le parametrizzazioni misurate, infatti, l'errore medio varia sempre all'interno del range $0.00041 \div 0.44898$, vale a dire con una percentuale di inesattezza pari a $0.041\% \div 44,898\%$. A prima vista tale intervallo può apparire troppo elevato ma, andando ad analizzare le tabelle riassuntive, si nota come i valori di errore più grandi siano tutti registrati per parametrizzazioni molto sfavorevoli, come ad esempio in casi in cui il numero di campioni da prelevare ($\mathcal{N}$) è molto basso: in tali situazioni, infatti, il calcolatore campiona lo spazio di ricerca in modo molto grossolano, aumentando così la probabilità di non rilevare l'ottimo globale.

Tralasciando questo comportamento del tutto prevedibile, la rilevazione più interessante consiste nel fatto che, generalmente, entrambe le procedure di ricerca di un equilibrio nei giochi in forma estesa basati su simulazione (SPE–Approximation e NE–Approximation) convergono sempre verso la strategia ideale, il più delle volte in tempi accettabili.

Comparando i due metodi realizzati, si nota però come l'algoritmo NE–Approximation sia molto più veloce rispetto a SPE–Approximation, grazie al fatto che il numero di campioni valutati è di gran lunga inferiore: tale comportamento è dovuto alla presenza di alcuni meccanismi di potatura all'interno del codice di NE–Approximation, che bloccano l'esplorazione di porzioni inutili dello spazio di ricerca.

Grazie al meccanismo di potatura implementato, il calcolatore non è obbligato, a differenza di quanto accade in SPE–Approximation, a valutare tutti i possibili profili di strategia, ma è in grado di escludere a priori delle strade, sulla base dei risultati parziali ottenuti sino a quel momento. A causa di ciò, risulta evidente come il numero di campioni valutati sia decisamente inferiore rispetto al caso SPE–Approximation e, di conseguenza, anche il tempo di esecuzione evidenzia un sensibile abbassamento.

L'introduzione della potatura non ha comportato però un miglioramento anche a livello di precisione, in quanto gli algoritmi lavorano comunque sulla base del medesimo meccanismo: di conseguenza, non si è registrato un sensibile miglioramento dello scostamento misurato.

5.3.2 Algoritmi di ottimizzazione

In questo paragrafo verranno analizzati i dati non più nell'ottica di valutare le procedure di ricerca di un equilibrio approssimato, ma per paragonare tra loro i diversi algoritmi di ottimizzazione implementati in questa sede.

La procedura basata sul Lipschitz optimization, permette di approssimare con arbitraria precisione il massimo globale delle funzioni di utilità Lipschitziane solamente quando si è a conoscenza del grado di continuità della stessa: conoscendo questo dato, si riesce a determinare un upper–bound per la derivabilità della funzione analizzata.

Visto e considerato che il Lipschitz optimization è un metodo di natura deterministica (vedi *Capitolo 3*) con un numero di variabili piuttosto basso, non c'è ragione di effettuare più esecuzioni per ogni parametrizzazione e, di conseguenza, il numero di casi da considerare è decisamente inferiore rispetto alle altre procedure.

Se invece non fosse noto a priori il grado di continuità della funzione di utilità sotto esame, la garanzia di accuratezza di questo algoritmo sarebbe estremamente bassa, rendendo questo metodo inadatto ad ottimizzare la funzione di payoff data.

Nel caso in cui il grado di continuità fosse molto ampio, invece, la convergenza dell'algoritmo sarebbe comunque garantita, ma tale risultato sarebbe conseguito con tempi molto lunghi, in quanto la funzione di utilità analizzata tenderebbe a crescere in maniera molto repentina, rendendo difficile l'individuazione del massimo globale. Per meglio comprendere questo aspetto, osservando le tabelle presenti nell'*Appendice A*, è possibile notare come il tempo di esecuzione delle diverse parametrizzazioni aumenti molto rapidamente tanto che, già dopo $\alpha = 3$, l'esecuzione del problema di ricerca va in time–out (il calcolatore non riesce a giungere ad una soluzione entro i 10 minuti).

Se non si avessero informazioni a priori riguardo alla funzione di utilità, sarebbe necessario ricorrere a metodi non–deterministici (come il simulated annealing e il cross entropy), in cui la ricerca dei campioni è affidata al caso, o a determinate distribuzioni di probabilità. Entrambi gli algoritmi non–deterministici presentati nel *Capitolo 3* adottano un approccio semplice ed efficace per risolvere i problemi di ottimizzazione di funzioni continue, nonostante il simulated annealing sia una tecnica ideata principalmente per algoritmi di ricerca locale.

Studiando i dati del testing, è possibile notare come le procedure implementate in questa sede convergano, in modo più o meno rapido, verso l'ottimo della funzione analizzata: i risultati ottenuti sono strettamente legati alla parametrizzazione usata e, in entrambi i casi, un numero di campioni più elevato genera una precisione migliore.

Nel caso del simulated annealing, si nota come l'aumento del parametro della temperatura γ (e quindi la minor tendenza della temperatura ad abbassarsi) provochi un peggioramento della velocità e della precisione di convergenza dell'algoritmo. Tale comportamente si spiega facilmente pensando al significato fisico del parametro temperatura: più questo è elevato, maggiore sarà la difficoltà ad ottenere campioni prossimi all'ottimo e, di conseguenza, dovranno essere campionati un numero maggiore di punti all'interno dello spazio di ricerca.

Per quel che riguarda il cross entropy, invece, si può vedere come il numero di campioni considerati promettenti γ sia un parametro direttamente proporzionale alla precisione della soluzione, penalizzando però la velocità di ricerca dell'ottimo. Prendere in considerazione un numero maggiore di campioni promettenti ci permette di evitare di cadere in situazioni di massimi locali, che rappresentano il rischio più grosso nei problemi di ricerca della best–response: ciò dipende dal fatto che, tenendo in considerazione un numero sufficientemente grande di punti dello spazio di ricerca, il calcolatore ha una visione più ampia della funzione obiettivo. D'altro canto però, osservare un numero maggiore di campioni penalizza notevolmente la velocità di convergenza, in quanto sarà più difficile che la condizione di terminazione sia verificata. Come nel simulated annealing, anche per questa procedura il numero di campioni da generare ad ogni iterazione $\mathcal{N}$ è legato direttamente alla crescita della precisione e delle prestazioni: aumentando questo dato, infatti, la soluzione trovata si avvicina sempre più all'ottimo teorico causando però, come previsto, una forte penalizzazione in fatto di velocità di esecuzione.

È possibile affermare che, sotto certe condizioni riguardanti la distribuzione di riferimento e la funzione obiettivo, i metodi qui studiati convergono all'ottimo con probabilità prossima a 1 quando i campioni impiegati crescono all'infinito (per maggiori dettagli, fare riferimento a [13] e [20]).

Se si dovesse effettuare un confronto tra le due procedure non–deterministiche qui presentate, risulterebbe immediato notare come l'ottimizzazione cross entropy sia decisamente superiore al simulated annealing, sia in termini di precisione che in termini di velocità di esecuzione. Si può osservare,

infatti, che un'ottimizzazione basata sul simulated annealing sia molto più inaffidabile rispetto ad una eseguita con il cross entropy: nel caso documentato dal *Grafico A.43* e dal *Grafico A.44* si nota come l'equilibrio non venga approssimato in modo sufficientemente accurato, neanche nelle parametrizzazioni più favorevoli. Questo comportamento deriva dal fatto che la tecnica del simulated annealing sia un metodo nato per l'ottimizzazione locale, mentre il cross entropy sia stato ideato per problemi di ricerca di massimi e minimi globali.

Analizzando i grafici relativi al tempo impiegato ed al numero di campioni generati ad ogni esecuzione, è interessante notare come tutti gli algoritmi implementati in questo lavoro abbiano un comportamento molto simile tra loro. Studiando i grafici, si osserva come il loro andamento cresca molto velocemente, assumendo un andamento esponenziale: tale comportamento è dovuto al fatto che un incremento del valore di $\mathcal{N}$, per quanto piccolo ed insignificante, provoca un aumento sostanziale del numero di campioni generati, influenzando così anche il tempo impiegato dal calcolatore per giungere alla soluzione ottimale.

Concentrando l'attenzione sui grafici degli scostamenti, si nota invece come il comportamento delle procedure realizzate sia molto diverso tra loro:

- nel caso del Lipschitz optimization lo scostamento della soluzione trovata rispetto a quella ideale decresce in maniera lineare o esponenziale.

- nel caso del simulated annealing è impossibile definire l'andamento dell'errore, in quanto ogni caso analizzato produce risultati molto diversi dagli altri. Tale comportamento deriva dal fatto che nel simulated annealing la componente probabilistica è molto influente, rendendo difficile effettuare un'analisi qualitativa sull'efficienza del metodo;

- nel caso del cross entropy lo scostamento della soluzione trovata dall'equilibrio ideale decresce in maniera esponenziale raggiungendo, nel caso delle parametrizzazioni più favorevoli, un valore prossimo allo zero.

Capitolo 6

Conclusioni e sviluppi futuri

I giochi basati su simulazione, cioè i modelli in cui tutte le funzioni di utilità sono il risultato di un processo di simulazione, hanno da sempre ricevuto una grande attenzione da parte della comunità scientifica. Ad oggi, i giochi in forma estesa basati su simulazione non sono stati, però, soggetti di studi approfonditi: l'unico studio significativo è rintracciabile in [21]. In questo lavoro di tesi, si è tentato di estendere la classe dei modelli basati su simulazione, applicando tale classe ai giochi in forma estesa e ai giochi continui, in cui gli agenti possono scegliere la propria azione su un insieme continuo.

In questa sede sono state implementate due diverse procedure convergenti per calcolare ed individuare un'approssimazione di due diversi tipi di equilibri: gli equilibri perfetti per sottogiochi (SPE) e gli equilibri di Nash (NE).

Per raggiungere tale scopo, sono state realizzate differenti tecniche di ottimizzazione – basate su simulated annealing (prendendo spunto da [3]), cross entropy e Lipschitz optimization – successivamente applicate nel corso della fase di sperimentazione: durante il testing delle procedure realizzate, sono state eseguite diverse prove con molte parametrizzazioni, raccogliendo tutti i risultati in tabelle riassuntive.

Osservando i valori ricavati, è stato notato che l'individuazione di un SPE è computabile in giochi con un numero ridotto di livelli, mentre l'algoritmo di NE–Approximation è meno restrittivo, ed è applicabile su modelli con un numero superiore di livelli. In aggiunta, i dati in nostro possesso hanno rilevato che i metodi basati sull'ottimizzazione cross entropy sono molto più convenienti rispetto alle procedure basate su simulated annealing o su Lip-

schitz optimization: ciò è dovuto al fatto che il primo metodo è usato nelle ricerche dei massimi globali, il secondo viene impiegato per i massimi locali, e l'ultimo è sfruttato solo per modelli continui.

Come possibile ulteriore sviluppo, si potrebbe cercare di incrementare l'efficienza degli algoritmi qui implementati, in tutte quelle situazioni in cui sono disponibili tutte le informazioni riguardanti la struttura del gioco (come ad esempio nel caso dei giochi d'azione grafici); un eventuale altro sviluppo potrebbe consistere nel sottoporre alle procedure qui illustrate dei modelli in cui vengano coinvolti un numero sempre maggiore di giocatori, così da testare più approfonditamente il lavoro sviluppato. Infine, un'ulteriore implementazione potrebbe derivare dall'adattamento della procedura di Lipschitz optimization, come qui definita, a problemi con funzioni obiettivo discrete.

Bibliografia

[1] John Forbes Nash. Equilibrium points in n–person games. In *Proceedings of the National Academy of the USA*, pages 48–49, 1950.

[2] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory.* MIT Press, 1999.

[3] Y. Vorobeychik and M.P. Wellman. Stochastic search methods for nash equilibrium approximation in simulation–based games. In *AAMAS*, pages 1055–1062, 2008.

[4] D. Fudenberg and J. Tirole. *Game Theory.* The MIT Press, Cambridge, USA, 1991.

[5] C. Harris. Existence and characterization of perfect equilibrium in games of perfect information. *ECONOMETRICA*, 53:613–628, 1985.

[6] J. Bochnak, M. Coste, and M.F. Roy. *Real algebraic geometry.* Springer Verlag, 1998.

[7] B.O. Shubert. A sequential method seeking the global maximum of a function. *SIAM J NUMER ANAL*, 9(3):379–388, 1972.

[8] D.E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison–Wesley, 1989.

[9] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *SCIENCE*, 220(4598):671–680, 1983.

[10] R.Y. Rubinstein and D.P. Kroese. *The cross–entropy method: a unified approach to combinatorial optimization, Monte–Carlo simulation, and machine learning.* Springer–Verlag, 2004.

[11] J. Kennedy and R.C. Eberhart. Particle swarm optimization. In *ICNN*, volume 4, pages 1942–1948, 1995.

[12] M. Locatelli. Simulated annealing algorithms for continuous global optimization: convergence conditions. *J OPTIMIZ THEORY APP*, 104(1):121–133, 2000.

[13] A. Ghate and R.L. Smith. Adaptive search with stochastic acceptance probabilities for global optimization. *OPER RES LETT*, 36(3):285–290, 2008.

[14] I.O. Bohachevsky, M.E. Johnson, and M.L. Stein. Generalized simulated annealing for function optimization. *TECHNOMETRICS*, 28(3):209–217, 1986.

[15] D. Vanderbilt and S.G. Louie. A Monte Carlo simulated annealing approach to optimization over continuous variables. *J COMPUT PHYS*, 56(2):259–271, 1984.

[16] S. Kullback and R.A. Leibler. On information and sufficiency. *ANN MATH STAT*, pages 79–86, 1951.

[17] John Forbes Nash. Non–cooperative games. In *Annals of Mathematics*, pages 286–295, 1951.

[18] John Forbes Nash. The bargaining problem. In *ECONOMETRICA*, pages 155–162, 1950.

[19] John Forbes Nash. Two person cooperative games. In *ECONOMETRICA*, pages 128–140, 1953.

[20] L. Margolin. On the convergence of the cross–entropy method. *ANN OPER RES*, 134(1):201–214, 2005.

[21] Y. Vorobeychik, D.M. Reeves, and M.P. Wellman. Constrained automated mechanism design for infinite games of incomplete information. In *UAI*, 2007.

[22] Stefano Marino Nicola Gatti, Francesco Di Giunta. Alternating–offers bargaining with one–sided uncertain deadlines: an efficient algorithm. In *Artificial Intelligence*, volume 172, pages 1119–1157, May 2008.

[23] John Von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[24] Y. Shoham and K. Leyton-Brown. *Multiagent systems: algorithmic, game–theoretic, and logical foundations*. Cambridge University Press, 2008.

[25] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice–Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[26] Patrick R. Jordan, Yevgeniy Vorobeychik, and Michael P. Wellman. Searching for approximate equilibria in empirical games. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems – Volume 2*, AAMAS '08, pages 1063–1070, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[27] Ivan Marsa-Maestre, Miguel A. Lopez-Carmona, Juan R. Velasco, and Enrique de la Hoz. Effective bidding and deal identification for negotiations in highly nonlinear scenarios. In *AAMAS (2)'09*, pages 1057–1064, 2009.

[28] Ivan Marsa-Maestre, Miguel A. Lopez-Carmona, Juan R. Velasco, Takayuki Ito, Mark Klein, and Katsuhide Fujita. Balancing utility and deal probability for auction–based negotiations in highly nonlinear utility spaces. In *IJCAI'09*, pages 214–219, 2009.

[29] Takayuki Ito, Hiromitsu Hattori, and Mark Klein. Multi–issue negotiation protocol for agents: exploring nonlinear utility spaces. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 1347–1352, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[30] Nate Derbinsky and Nicholas A. Gorski. Exploring the space of computational memory models.

[31] Lihua Wu and Yuyun Wang. An introduction to simulated annealing algorithms for the computation ofeconomic equilibrium. *Comput. Econ.*, 12:151–169, October 1998.

[32] Andrea Lecchini-Visintini, John Lygeros, and Jan M. Maciejowski. Simulated annealing: Rigorous finite–time guarantees for optimization on continuous domains. In *NIPS*, 2007.

[33] D. Mitra, F. Romeo, and A. Sangiovanni-Vincentelli. Convergence and finite–time behavior of simulated annealing. In *ADV APPL PROBAB*, volume 18, pages 747–771, 1986.

[34] HP Benson, R. Horst, and PM Pardalos. Handbook of Global Optimization, 1995.

[35] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, et al. Equation of state calculations by fast computing machines. *J CHEM PHYS*, 21(6):1087, 1953.

[36] I. Marsa-Maestre, M.A. Lopez-Carmona., J.R. Velasco, and E. de la Hoz. Avoiding the prisoner's dilemma in auction–based negotiations for highly rugged utility spaces. In *AAMAS*, pages 425–432, 2010.

[37] F. Di Giunta and N. Gatti. Bargaining over multiple issues in finite horizon alternating–offers protocol. *ANN MATH ARTIF INTEL*, 47(3–4):251–271, 2006.

[38] D.E. Knuth and R.W. Moore. An analysis of alpha–beta pruning* 1. *ARTIF INTEL*, 6(4):293–326, 1975.

Appendice A

Risultati del testing

A.1 Caso A

A.1.1 Lipschitz optimization

α			
1	2	3	
0,22138	0,14342	0,07321	e
10,8857	89,8196	569,0171	t
138359	1127485	6692909	s

Tabella A.1: *Risultati Lipschitz optimization per SPE (A)*

α			
1	2	3	
0,15734	0,09836	0,06251	e
5,8374	53,1728	273,9184	t
59725	604627	3483090	s

Tabella A.2: *Risultati Lipschitz optimization per NE (A)*

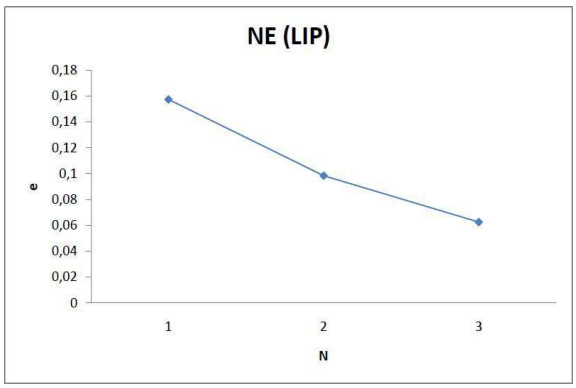

Figura A.1: *Errore NE–LIP (A)*

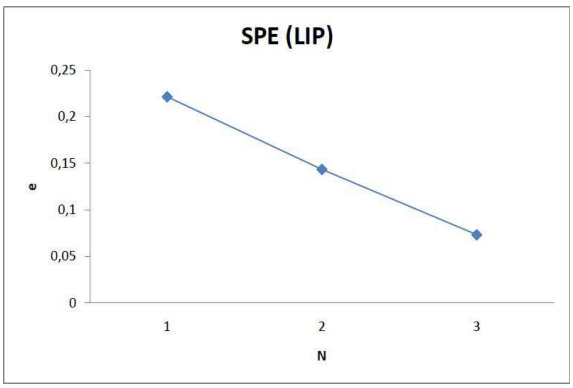

Figura A.2: *Errore SPE–LIP (A)*

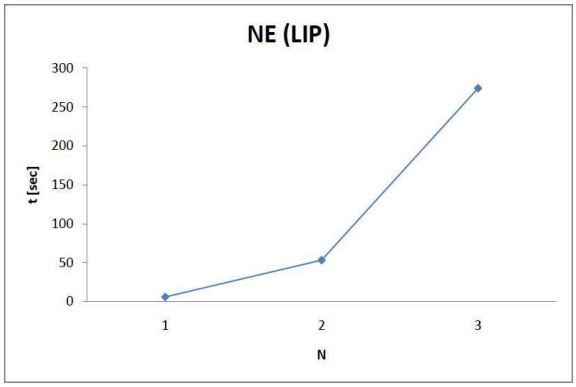

Figura A.3: *Durata NE–LIP (A)*

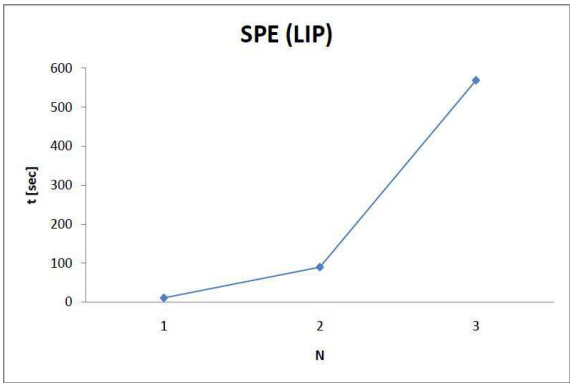

Figura A.4: *Durata SPE–LIP (A)*

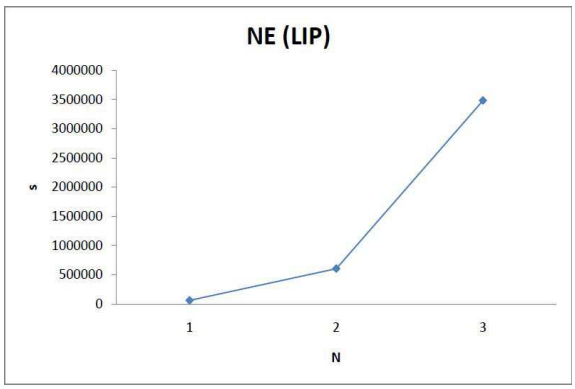

Figura A.5: *Campioni NE–LIP (A)*

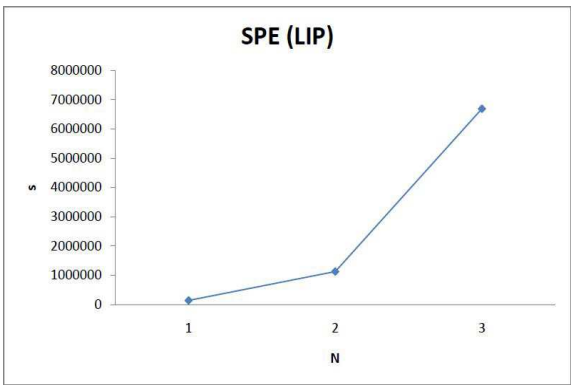

Figura A.6: *Campioni SPE–LIP (A)*

A.1.2 Simulated annealing

		γ					
		0,3	0,4	0,5	0,6	0,7	
N	10	0,21046	0,19312	0,18585	0,21560	0,26400	e
		12,2538	12,2999	12,2943	12,2781	12,3331	t
		249109	249297	249453	249204	250210	s
	20	0,14014	0,10187	0,14823	0,11036	0,22530	e
		97,2508	97,4243	97,0432	97,3415	97,3158	t
		2062844	2065496	2058486	2062596	2059530	s
	30	0,27083	0,32762	0,20656	0,21353	0,30134	e
		330,6471	331,9553	331,8999	331,2650	330,5719	t
		7033842	7066710	7067619	7055754	7046841	s
	40	0,22810	0,25716	0,17316	0,18736	0,24904	e
		795,9018	793,9436	796,3335	797,7401	801,9483	t
		16885000	16826516	16878900	16642594	16885172	s
	50	0,03654	0,02039	0,00789	0,00927	0,01764	e
		2005,7822	2000,9981	2017,0926	1998,2839	2000,0021	t
		33239550	33274952	33329637	33299872	33723641	s

Tabella A.3: *Risultati simulated annealing SPE (A)*

		γ					
		0,3	0,4	0,5	0,6	0,7	
N	10	0,18746	0,17898	0,17329	0,19720	0,21841	e
		4,8926	4,8172	4,0988	4,7203	4,8195	t
		45982	46028	46152	46682	46771	s
	20	0,14837	0,15902	0,16002	0,16970	0,18722	e
		31,7182	32,9288	31,9283	31,9925	30,8117	t
		403658	408512	411823	412988	415032	s
	30	0,20380	0,19722	0,19802	0,20081	0,23871	e
		97,8536	97,9471	96,0021	98,8461	97,8826	t
		872043	875003	876932	878923	880441	s
	40	0,17635	0,17603	0,17992	0,19102	0,20012	e
		238,9092	238,7164	237,9964	237,9015	238,0035	t
		1598203	1597625	1602617	1628374	1619934	s
	50	0,01384	0,01182	0,00519	0,00602	0,00927	e
		587,8273	591,8642	587,9043	587,2545	586,0273	t
		4183902	4191773	4206398	4213998	4248716	s

Tabella A.4: *Risultati simulated annealing NE (A)*

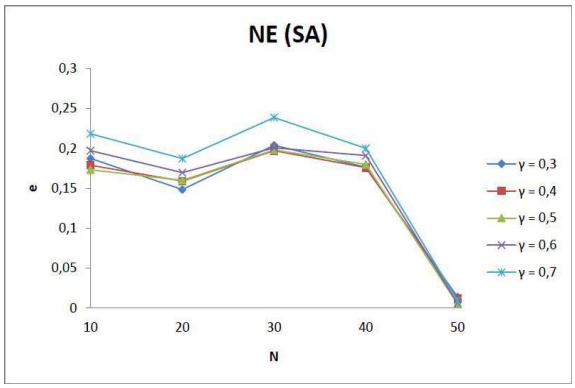

Figura A.7: *Errore NE–SA (A)*

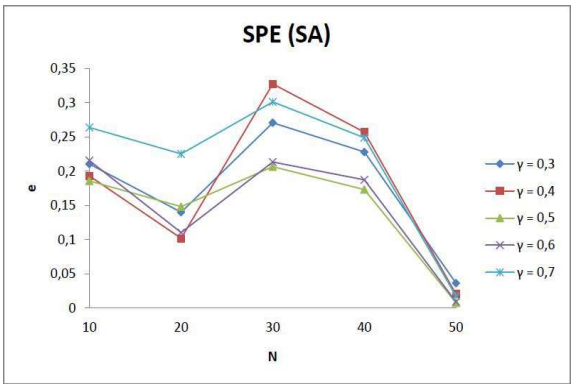

Figura A.8: *Errore SPE–SA (A)*

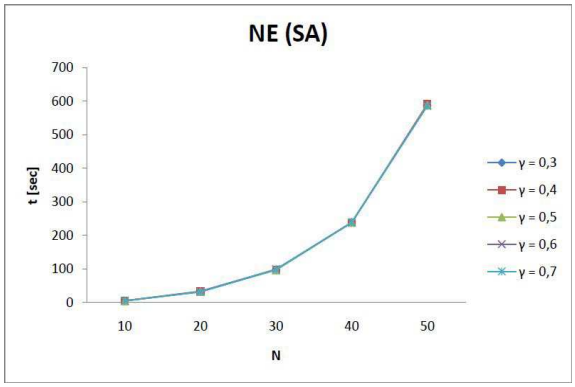

Figura A.9: *Durata NE–SA (A)*

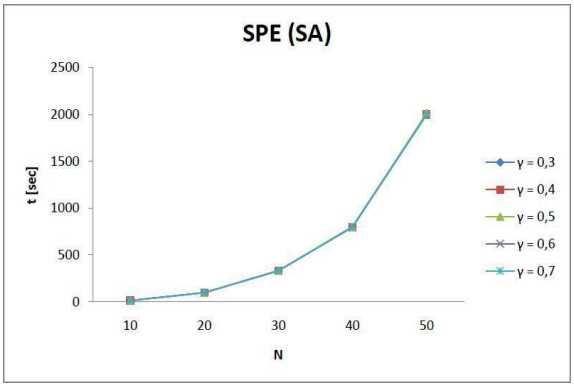

Figura A.10: *Durata SPE–SA (A)*

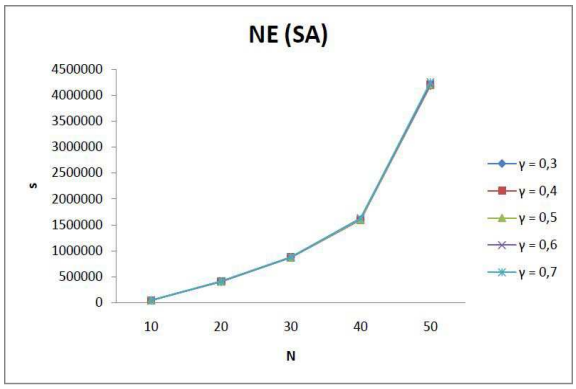

Figura A.11: *Campioni NE–SA (A)*

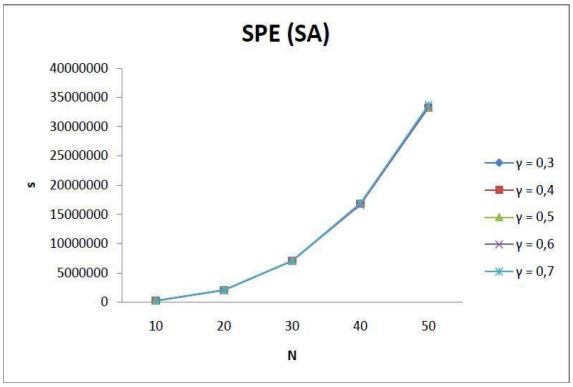

Figura A.12: *Campioni SPE–SA (A)*

A.1.3 Cross entropy

		γ					
		2	3	4	5	6	
N	10	0,14059	0,14312	0,19064	0,24468	0,10624	e
		1,1557	1,4605	1,7546	1,8150	3,0119	t
		30178	39126	46940	48575	80812	s
	20	0,02392	0,02406	0,01396	0,04653	0,01095	e
		4,5829	6,3191	8,7074	10,0738	11,4933	t
		139048	192604	265354	306276	349660	s
	30	0,01565	0,01264	0,01741	0,00746	0,00830	e
		13,8476	18,5617	23,3485	25,7340	30,5611	t
		440934	591741	745659	822522	974973	s
	40	0,01525	0,01163	0,02031	0,01758	0,00773	e
		33,5218	35,5333	42,2313	53,2719	57,4686	t
		1092980	1160688	1379076	1736796	1845660	s
	50	0,01169	0,01592	0,00382	0,00616	0,00284	e
		53,9420	68,9212	72,5592	93,5162	101,9582	t
		1760060	2254475	2500415	3044325	3290770	s

Tabella A.5: *Risultati cross entropy per SPE (A)*

		γ					
		2	3	4	5	6	
N	10	0,13948	0,14004	0,14293	0,14188	0,13949	e
		0,6284	0,9184	1,0284	1,4729	1,6872	t
		8582	12384	15038	15983	18936	s
	20	0,00893	0,00918	0,00962	0,01274	0,01383	e
		2,6366	3,5862	4,1985	5,3847	5,9752	t
		52938	61984	81938	90127	100234	s
	30	0,01038	0,01175	0,01199	0,01265	0,01176	e
		7,8472	9,4783	9,2847	11,3224	15,2975	t
		121998	132847	139847	149837	158029	s
	40	0,00726	0,00826	0,01832	0,01194	0,01318	e
		20,3765	24,2040	27,3264	30,2974	35,3874	t
		293812	309187	319287	330928	345276	s
	50	0,00264	0,00482	0,00586	0,00816	0,00917	e
		42,4665	47,3764	50,3875	57,2874	62,4876	t
		539287	549263	557263	569287	590273	s

Tabella A.6: *Risultati cross entropy per NE (A)*

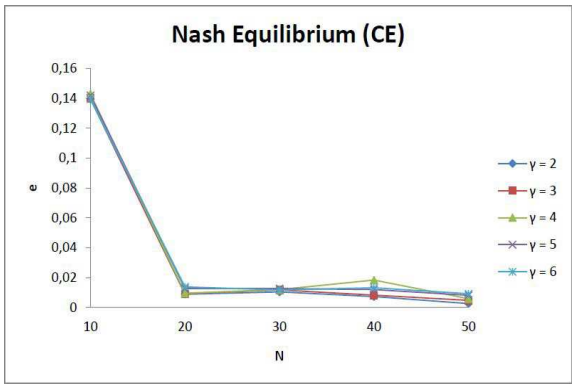

Figura A.13: *Errore NE–CE (A)*

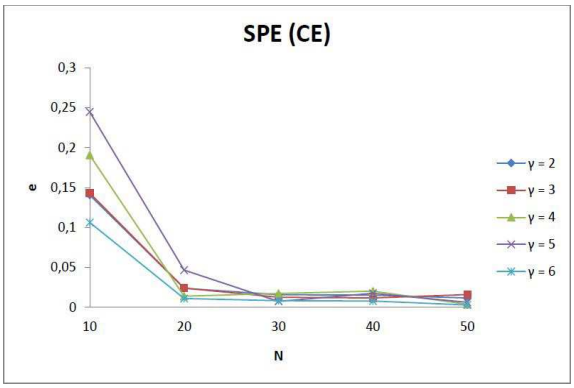

Figura A.14: *Errore SPE–CE (A)*

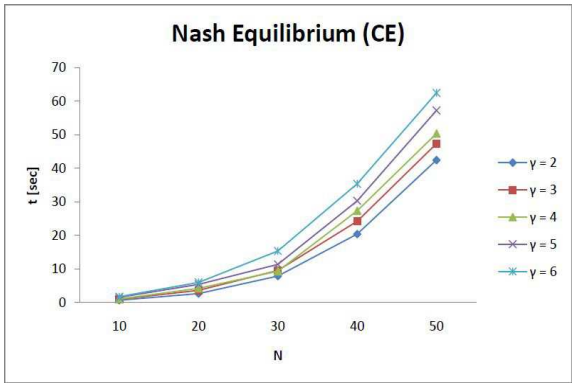

Figura A.15: *Durata NE–CE (A)*

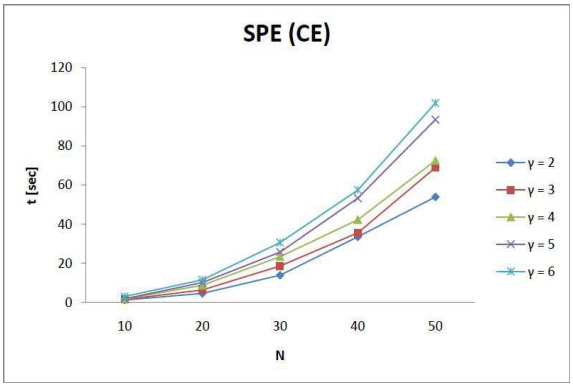

Figura A.16: *Durata SPE–CE (A)*

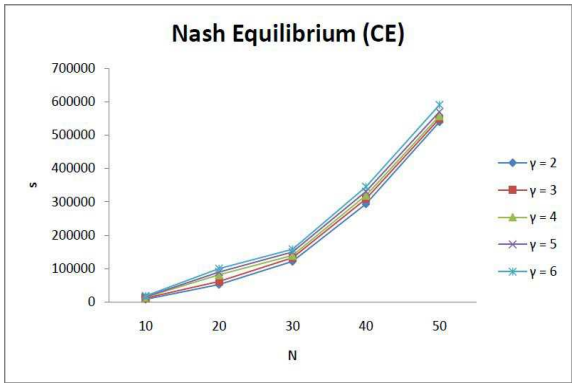

Figura A.17: *Campioni NE–CE (A)*

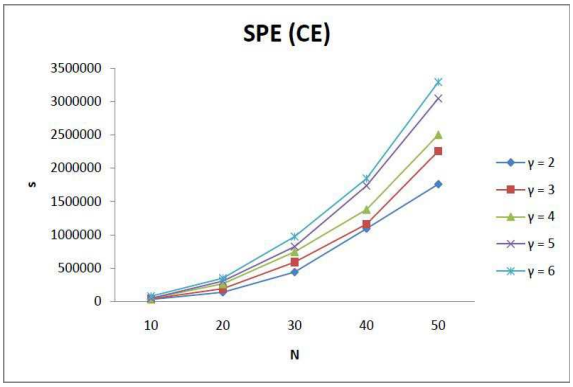

Figura A.18: *Campioni SPE–CE (A)*

A.2 Caso B

A.2.1 Lipschitz optimization

α		
1	2	
0,04910	0,00235	e
385,4531	2864,5098	t
5166853	36768127	s

Tabella A.7: *Risultati Lipschitz optimization per SPE (B)*

α		
1	2	
0,00837	0,00041	e
259,2639	2003,3841	t
2539871	12827400	s

Tabella A.8: *Risultati Lipschitz optimization per NE (B)*

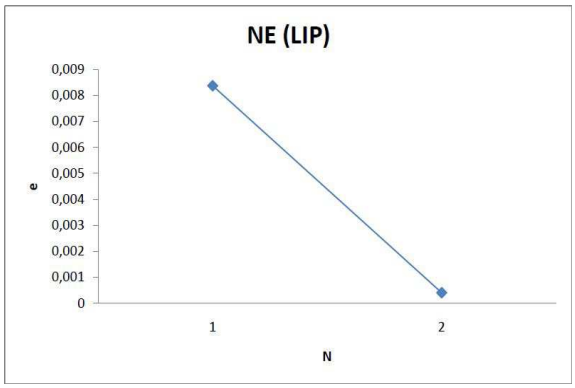

Figura A.19: *Errore NE–LIP (B)*

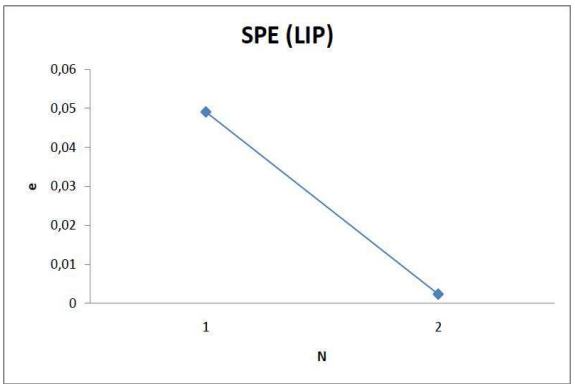

Figura A.20: *Errore SPE–LIP (B)*

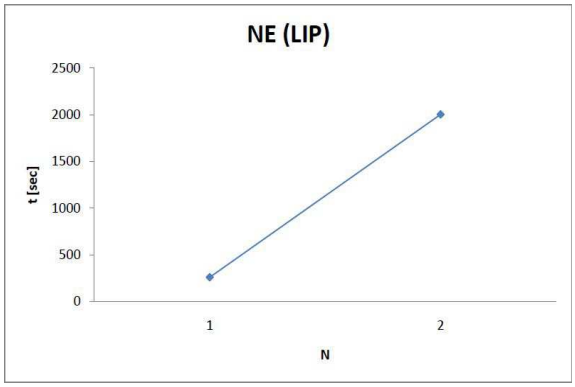

Figura A.21: *Durata NE–LIP (B)*

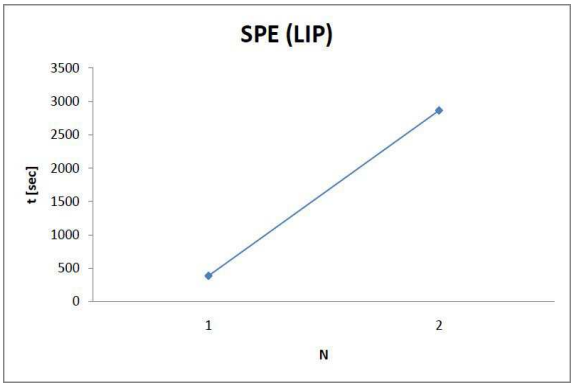

Figura A.22: *Durata SPE–LIP (B)*

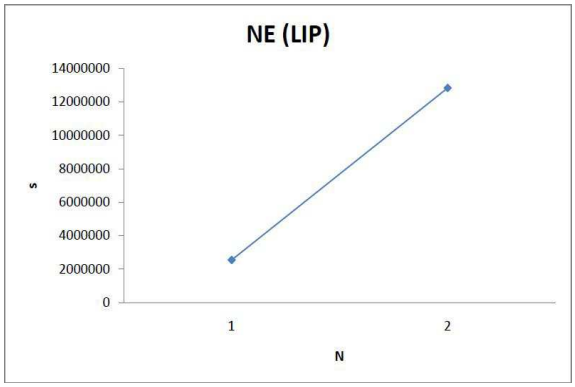

Figura A.23: *Campioni NE–LIP (B)*

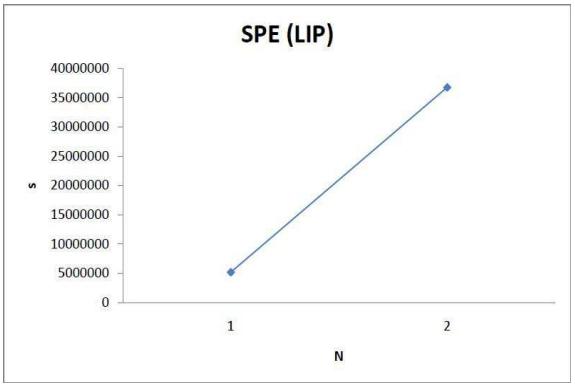

Figura A.24: *Campioni SPE–LIP (B)*

A.2.2 Simulated annealing

		γ					
		0,3	0,4	0,5	0,6	0,7	
N	10	0,03958	0,17630	0,11213	0,13170	0,09994	e
		0,9547	0,9308	0,9458	0,9909	0,9362	t
		39071	38461	39256	41035	38549	s
	20	0,10190	0,09324	0,09679	0,14198	0,10140	e
		7,1543	7,1111	7,0423	7,1892	7,0602	t
		327424	325200	322086	328604	322378	s
	30	0,10794	0,09218	0,14048	0,12939	0,17155	e
		25,0498	23,4261	24,3100	24,5802	23,8294	t
		1152627	1080699	1124430	1135413	1102473	s
	40	0,07061	0,13665	0,10502	0,10989	0,18378	e
		59,3594	58,4624	59,4836	55,9040	56,7217	t
		2711264	2670088	2714048	2553780	2590356	s
	50	0,05542	0,16347	0,09249	0,11453	0,11010	e
		118,4183	115,3286	113,7800	118,5393	116,3041	t
		5306980	5162875	5095030	5308250	5207250	s

Tabella A.9: *Risultati simulated annealing SPE (B)*

		γ					
		0,3	0,4	0,5	0,6	0,7	
N	10	0,05621	0,08264	0,05171	0,06154	0,04817	e
		0,3790	0,4037	0,6944	0,4652	0,5650	t
		18925	19005	20984	21719	20016	s
	20	0,05980	0,04890	0,04138	0,07164	0,05813	e
		3,3090	4,3903	2,9987	3,5985	3,0092	t
		169275	159327	170063	169438	176252	s
	30	0,06980	0,03098	0,06912	0,05872	0,07641	e
		13,3474	15,3498	14,8935	17,3474	16,3412	t
		570283	619251	579273	569263	593272	s
	40	0,03622	0,06083	0,05142	0,04811	0,08568	e
		31,9833	28,8784	29,4371	26,1231	32,3783	t
		1296638	1592028	1597316	1392651	1492891	s
	50	0,02218	0,08361	0,05985	0,04711	0,05593	e
		58,2178	59,2112	56,3987	60,2187	62,2187	t
		2710841	2619731	2409670	2834769	3026452	s

Tabella A.10: *Risultati simulated annealing NE (B)*

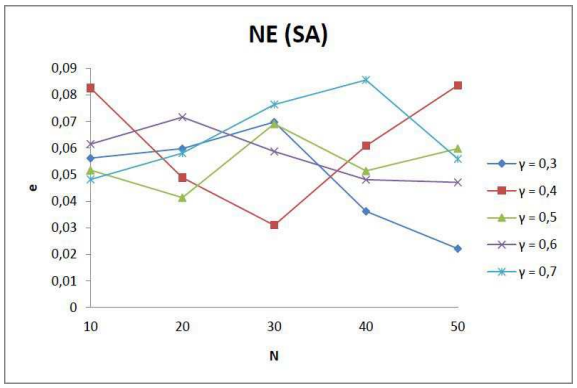

Figura A.25: *Errore NE–SA (B)*

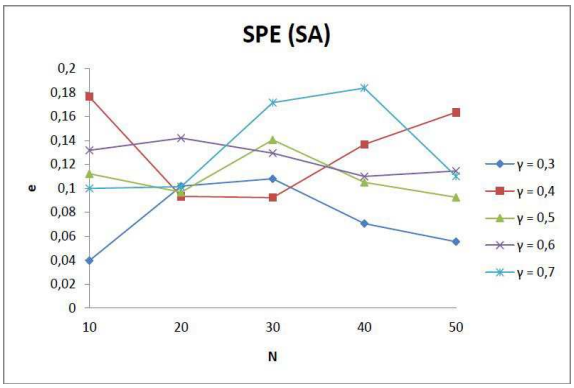

Figura A.26: *Errore SPE–SA (B)*

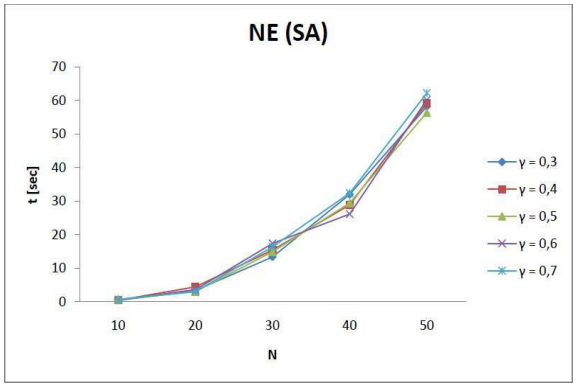

Figura A.27: *Durata NE–SA (B)*

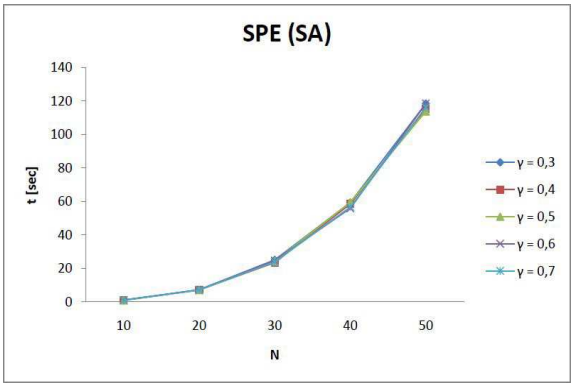

Figura A.28: *Durata SPE–SA (B)*

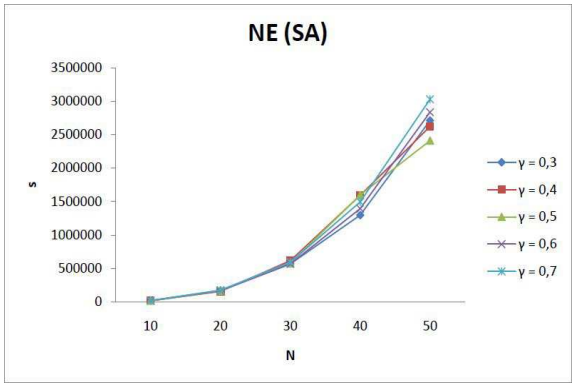

Figura A.29: *Campioni NE–SA (B)*

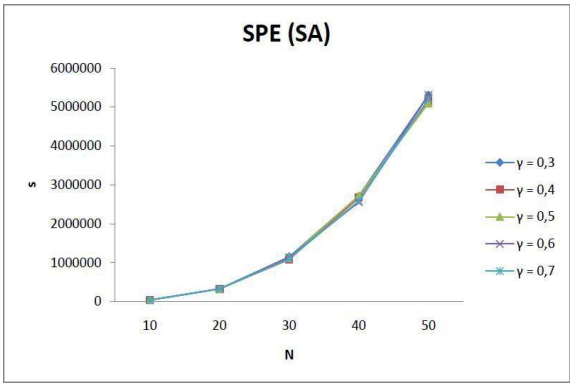

Figura A.30: *Campioni SPE–SA (B)*

A.2.3 Cross entropy

		γ					
		2	3	4	5	6	
N	10	0,05203	0,04486	0,10978	0,08290	0,11822	e
		0,3966	0,4409	0,4293	0,3879	0,3689	t
		10503	11641	11347	10212	9627	s
	20	0,03508	0,02861	0,03957	0,01565	0,02295	e
		2,5234	2,4970	3,1366	2,9573	2,9324	t
		72204	68922	91896	88532	88450	s
	30	0,02260	0,01535	0,01862	0,01351	0,01709	e
		7,2695	7,6321	7,6564	9,8877	9,8669	t
		230643	241626	241626	307524	307524	s
	40	0,01152	0,01201	0,01335	0,00999	0,01378	e
		16,3861	16,0714	17,1054	18,3957	20,0509	t
		518480	518480	544404	596252	648100	s
	50	0,01542	0,00565	0,00716	0,00952	0,00595	e
		31,1587	30,9879	30,9856	36,8793	32,1664	t
		1010100	1010100	1010100	1212120	1060605	s

Tabella A.11: *Risultati cross entropy per SPE (B)*

		γ					
		2	3	4	5	6	
N	10	0,01642	0,00928	0,01098	0,03810	0,05920	e
		0,2938	0,4349	0,5872	0,7983	0,8812	t
		3049	5869	8093	9384	9987	s
	20	0,00394	0,00430	0,00932	0,01978	0,02781	e
		1,2947	1,9483	2,0355	2,9485	2,7882	t
		25929	30947	42938	48711	49008	s
	30	0,00572	0,00600	0,00692	0,00601	0,00583	e
		4,6234	4,5929	5,0234	6,2013	7,6736	t
		69842	65298	71009	71284	75992	s
	40	0,00349	0,00498	0,00598	0,00608	0,00728	e
		9,2895	13,5735	14,6732	17,3894	18,3049	t
		157263	162004	169283	170093	179822	s
	50	0,00092	0,00198	0,00280	0,00493	0,00526	e
		24,5094	23,9343	27,3295	29,3843	34,3912	t
		289630	291877	301887	290485	319985	s

Tabella A.12: *Risultati cross entropy per NE (B)*

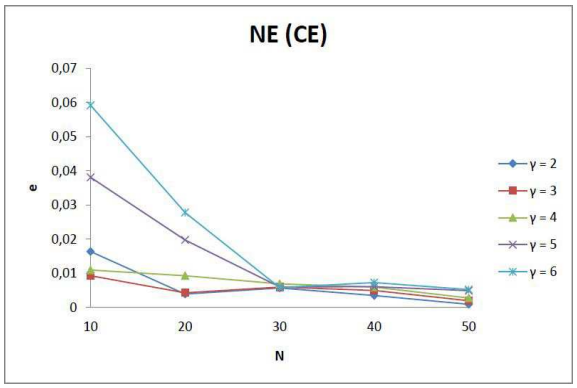

Figura A.31: *Errore NE–CE (B)*

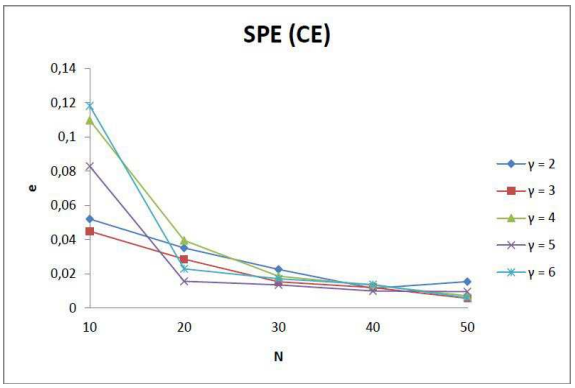

Figura A.32: *Errore SPE–CE (B)*

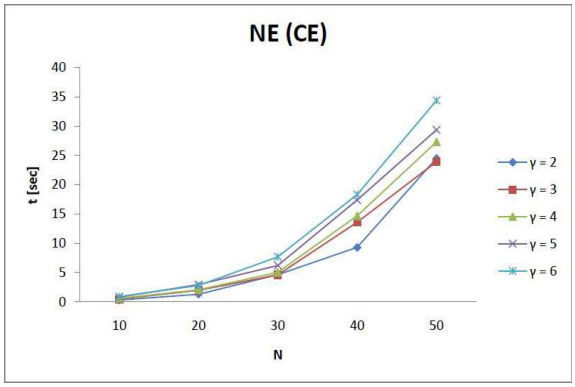

Figura A.33: *Durata NE–CE (B)*

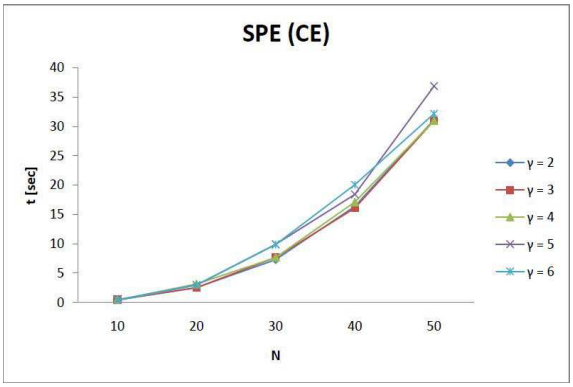

Figura A.34: *Durata SPE–CE (B)*

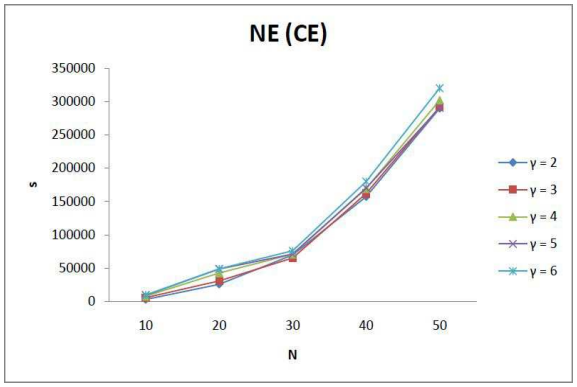

Figura A.35: *Campioni NE–CE (B)*

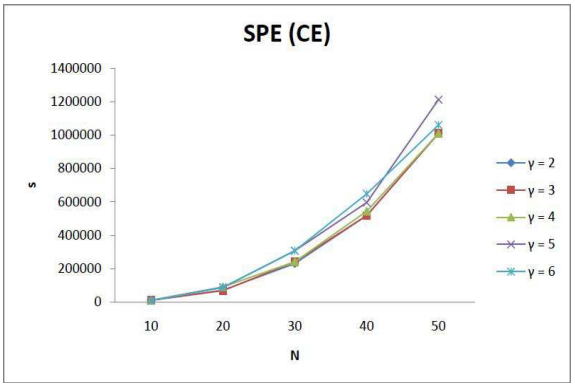

Figura A.36: *Campioni SPE–CE (B)*

A.3 Caso C

A.3.1 Lipschitz optimization

α			
1	2	3	
0,27629	0,18300	0,13140	e
46,3939	322,2327	2192,7394	t
611425	4087557	25913565	s

Tabella A.13: *Risultati Lipschitz optimization per SPE (C)*

α			
1	2	3	
0,12676	0,08263	0,04912	e
31,7502	182,9385	1204,2966	t
412977	2398122	15928100	s

Tabella A.14: *Risultati Lipschitz optimization per NE (C)*

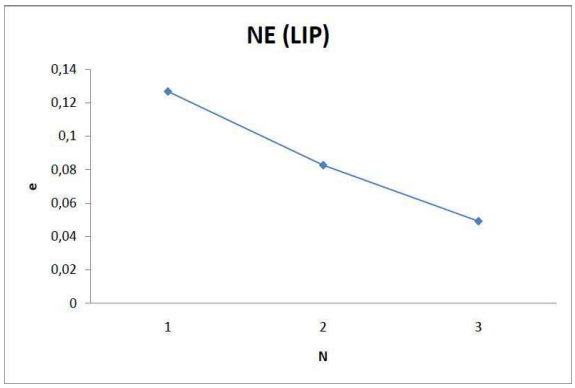

Figura A.37: *Errore NE–LIP (C)*

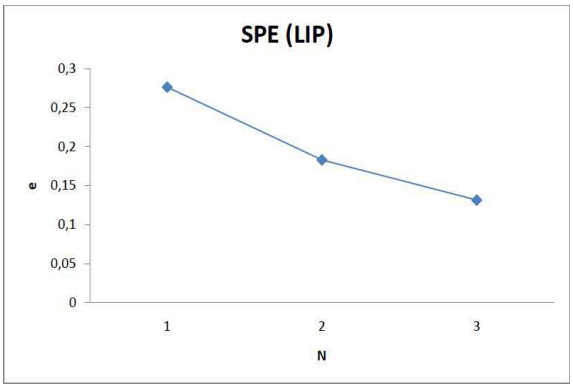

Figura A.38: *Errore SPE–LIP (C)*

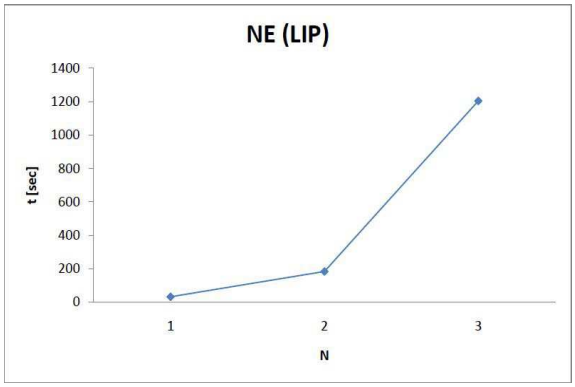

Figura A.39: *Durata NE–LIP (C)*

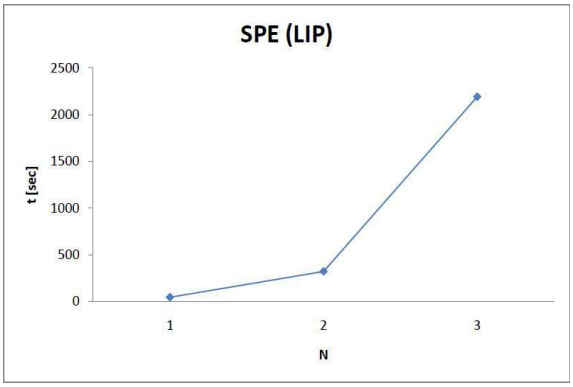

Figura A.40: *Durata SPE–LIP (C)*

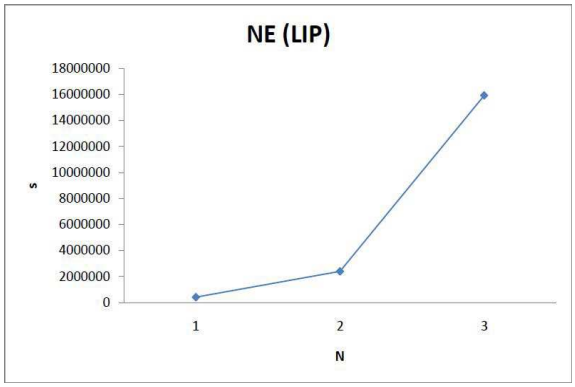

Figura A.41: *Campioni NE–LIP (C)*

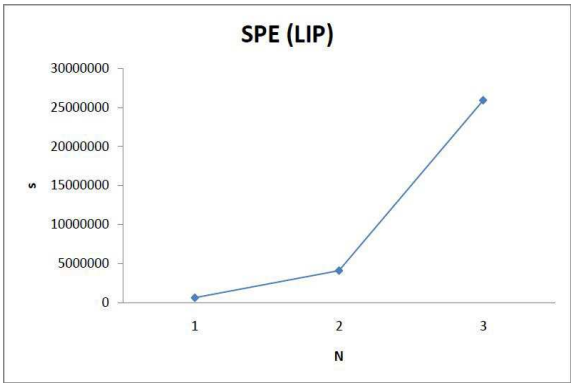

Figura A.42: *Campioni SPE–LIP (C)*

A.3.2 Simulated annealing

		γ					
		0,3	0,4	0,5	0,6	0,7	
N	10	0,21611	0,37098	0,20627	0,22543	0,22809	e
		0,9984	1,0063	0,9946	0,9916	1,0030	t
		37661	38248	37761	37584	38070	s
	20	0,11573	0,34653	0,31322	0,36138	0,29209	e
		7,2102	7,4541	7,3828	7,3736	7,3493	t
		299104	308732	306882	306126	304612	s
	30	0,13782	0,33887	0,34487	0,22786	0,24942	e
		24,8483	24,8110	24,6771	24,7570	24,6588	t
		1043385	1045248	1036593	1040685	1036965	s
	40	0,11151	0,22899	0,28104	0,25307	0,24864	e
		58,8382	59,2836	59,6485	59,7100	59,6394	t
		2458744	2478268	2492048	2494836	2490080	s
	50	0,35647	0,44898	0,31213	0,31389	0,18666	e
		118,0427	118,6488	118,7010	118,3114	117,2417	t
		4911030	4938575	4938329	4920980	4877875	s

Tabella A.15: *Risultati simulated annealing SPE (C)*

		γ					
		0,3	0,4	0,5	0,6	0,7	
N	10	0,11210	0,17909	0,13851	0,11237	0,14811	e
		0,4027	0,6849	0,4792	0,4822	0,5913	t
		18937	20966	19875	18030	19663	s
	20	0,06089	0,15903	0,15022	0,16298	0,17211	e
		3,6890	3,7392	3,9282	4,1871	3,0971	t
		148593	151029	155726	151299	154257	s
	30	0,05982	0,16329	0,18022	0,11291	0,14098	e
		13,2984	12,8722	14,9009	11,2191	12,3830	t
		539948	510093	529365	559327	569487	s
	40	0,05210	0,11269	0,15838	0,12643	0,13874	e
		28,2182	31,3988	30,3498	30,9882	32,4390	t
		1298374	1210043	1257730	1257730	1306291	s
	50	0,16932	0,20485	0,15821	0,16093	0,09216	e
		59,3985	60,4985	57,9860	60,3982	61,3903	t
		2498821	2510398	2572938	2538810	2569834	s

Tabella A.16: *Risultati simulated annealing NE (C)*

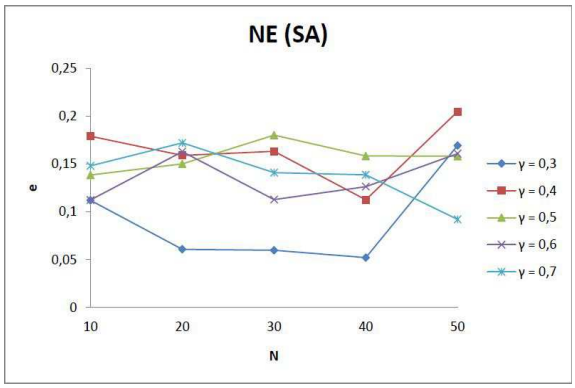

Figura A.43: *Errore NE–SA (C)*

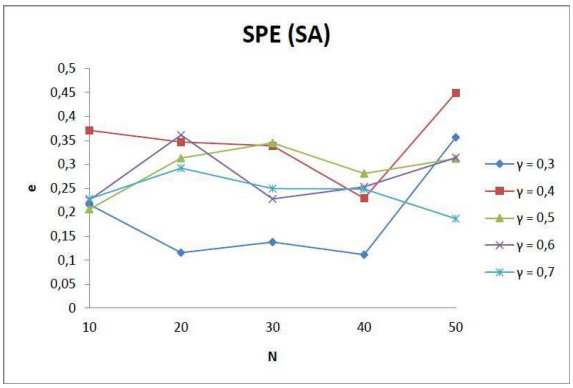

Figura A.44: *Errore SPE–SA (C)*

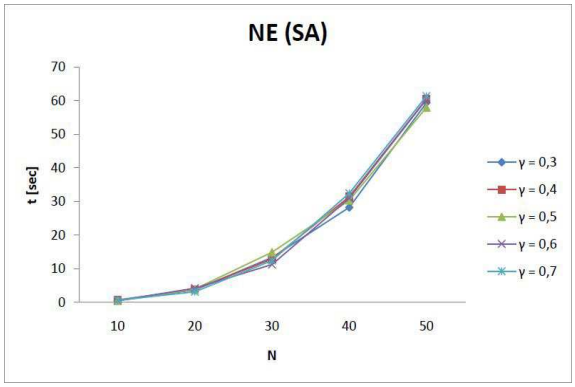

Figura A.45: *Durata NE–SA (C)*

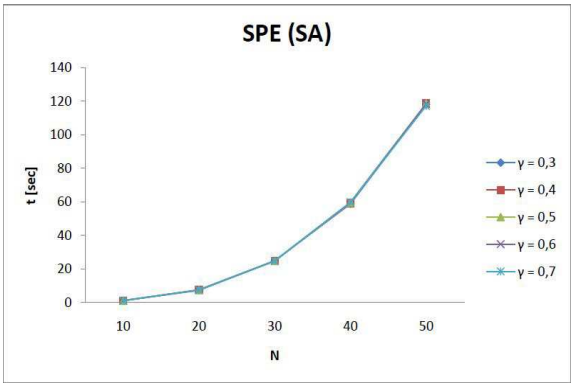

Figura A.46: *Durata SPE–SA (C)*

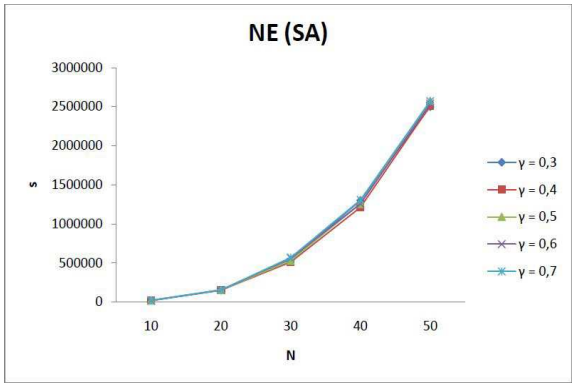

Figura A.47: *Campioni NE–SA (C)*

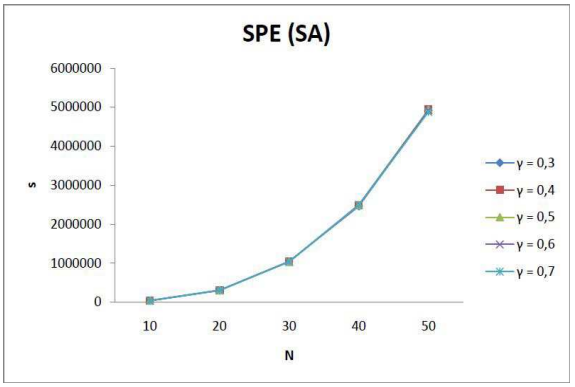

Figura A.48: *Campioni SPE–SA (C)*

A.3.3 Cross entropy

		γ					
		2	3	4	5	6	
N	10	0,12817	0,13240	0,03293	0,10839	0,09751	e
		0,3624	0,3228	0,4224	0,4305	0,5401	t
		9253	8662	11372	11560	13934	s
	20	0,03655	0,03459	0,06020	0,05086	0,02861	e
		2,9246	3,0103	2,3338	2,5410	2,5352	t
		89416	92610	70460	78086	76854	s
	30	0,03299	0,02540	0,02749	0,02849	0,02218	e
		9,0882	9,4015	9,2822	8,6296	8,5403	t
		291027	301452	296322	276186	273255	s
	40	0,00936	0,00587	0,01370	0,02112	0,01059	e
		21,2392	23,9240	19,8889	19,7250	19,5413	t
		696356	784808	652272	645464	639304	s
	50	0,01595	0,01424	0,01602	0,01988	0,02318	e
		39,7824	45,0617	35,7000	38,2048	37,6332	t
		1323200	1489850	1188340	1271665	1251460	s

Tabella A.17: *Risultati cross entropy per SPE (C)*

		γ					
		2	3	4	5	6	
N	10	0,06330	0,06321	0,05988	0,05029	0,04998	e
		0,0173	0,1865	0,2210	0,2194	0,2983	t
		5932	4938	5922	6928	6029	s
	20	0,01580	0,01510	0,01700	0,02179	0,01299	e
		1,3046	1,4596	1,5932	1,5967	1,2039	t
		44928	47981	36182	37198	32039	s
	30	0,01274	0,01198	0,01285	0,01597	0,01098	e
		5,3098	4,9731	3,9884	4,2894	4,9212	t
		153982	155988	147293	132957	139476	s
	40	0,00439	0,00230	0,00633	0,01139	0,00532	e
		12,4823	11,9348	13,2908	9,2983	10,2014	t
		393948	389135	342800	327162	300981	s
	50	0,00822	0,00716	0,00822	0,00937	0,00748	e
		24,3095	21,3905	18,3250	17,3940	20,3250	t
		659284	768103	529341	649920	682776	s

Tabella A.18: *Risultati cross entropy per NE (C)*

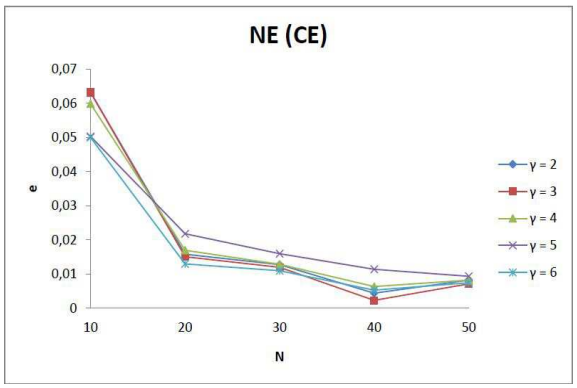

Figura A.49: *Errore NE–CE (C)*

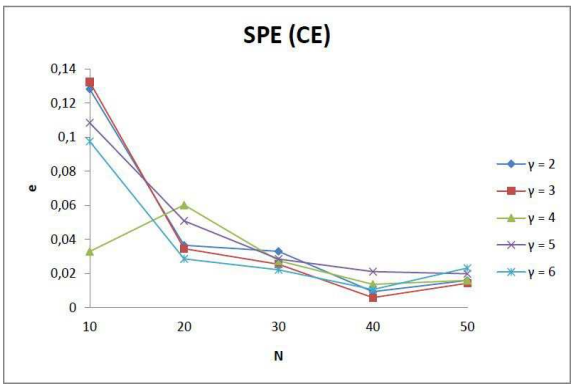

Figura A.50: *Errore SPE–CE (C)*

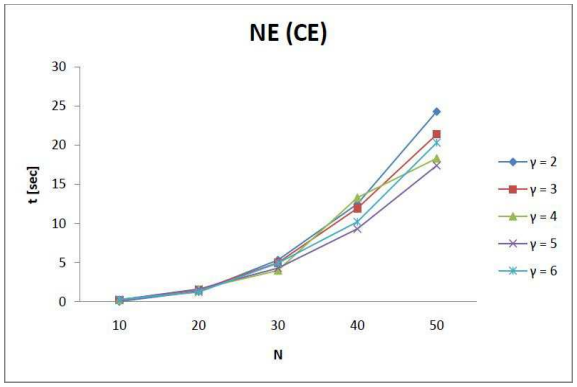

Figura A.51: *Durata NE–CE (C)*

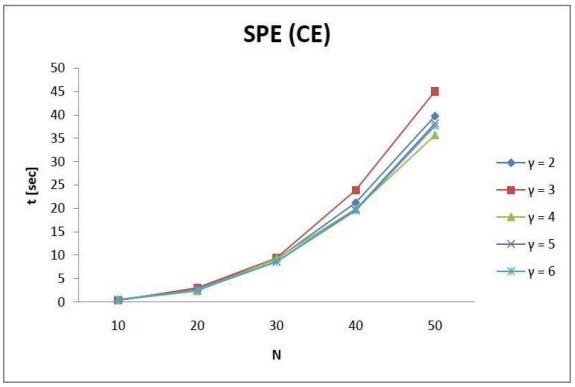

Figura A.52: *Durata SPE–CE (C)*

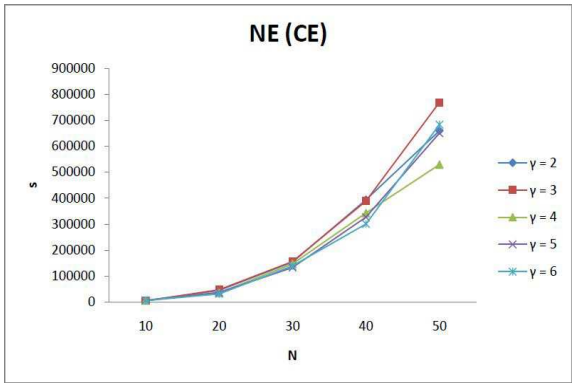

Figura A.53: *Campioni NE–CE (C)*

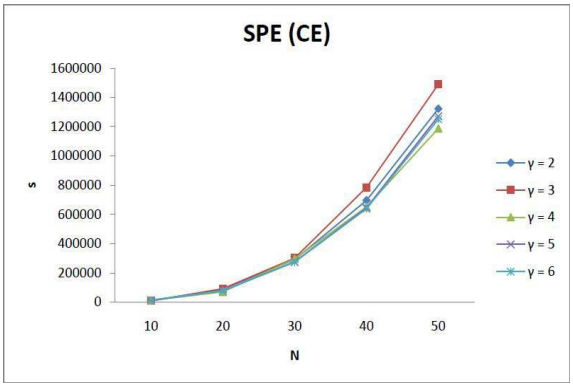

Figura A.54: *Campioni SPE–CE (C)*

A.4 Caso D

A.4.1 Lipschitz optimization

α			
1	2	3	
0,25624	0,10895	0,02910	e
33,4156	433,8846	2021,4721	t
431467	5436007	24855205	s

Tabella A.19: *Risultati Lipschitz optimization per SPE (D)*

α			
1	2	3	
0,11749	0,06293	0,00817	e
18,3947	235,9827	1104,9440	t
249772	2977361	15990223	s

Tabella A.20: *Risultati Lipschitz optimization per NE (D)*

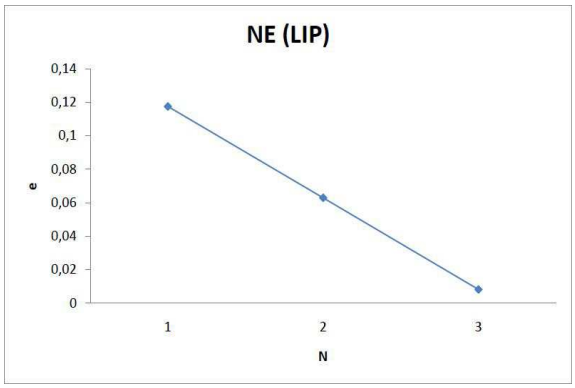

Figura A.55: *Errore NE–LIP (D)*

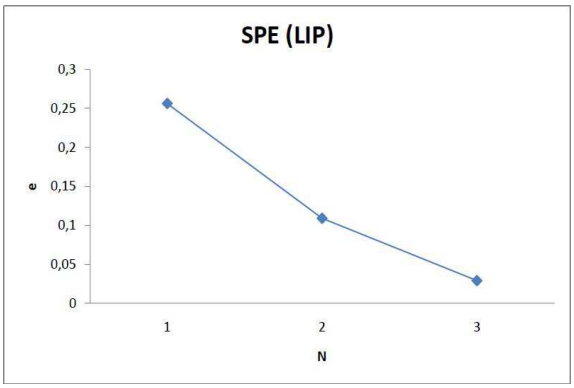

Figura A.56: *Errore SPE–LIP (D)*

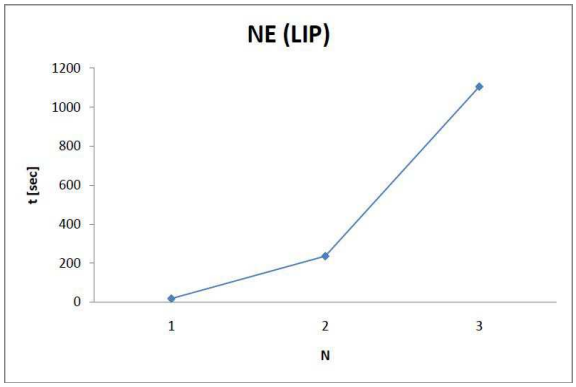

Figura A.57: *Durata NE–LIP (D)*

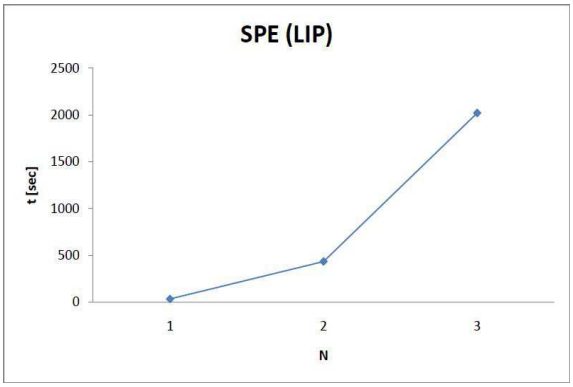

Figura A.58: *Durata SPE–LIP (D)*

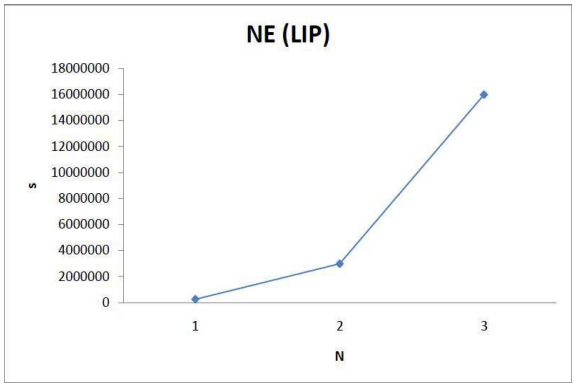

Figura A.59: *Campioni NE–LIP (D)*

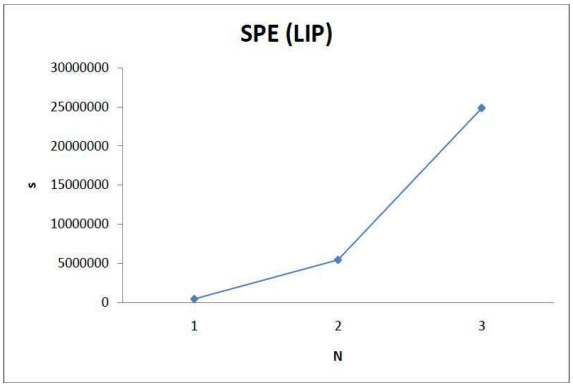

Figura A.60: *Campioni SPE–LIP (D)*

A.4.2 Simulated annealing

		γ					
		0,3	0,4	0,5	0,6	0,7	
N	10	0,02613	0,04531	0,04921	0,07598	0,04132	e
		1,0380	1,0780	1,1024	1,1204	1,0501	t
		39963	41542	42071	42934	40194	s
	20	0,08986	0,08645	0,06384	0,06173	0,06825	e
		8,1150	8,0328	8,1040	7,9018	7,8326	t
		344012	339388	342834	334552	332154	s
	30	0,09710	0,08203	0,09700	0,06407	0,06894	e
		27,0178	26,9811	26,8954	27,1537	27,0539	t
		1161033	1161312	1160103	1169775	1165869	s
	40	0,09021	0,09593	0,10168	0,08460	0,08258	e
		64,6102	64,7723	64,5806	64,8477	62,7999	t
		2761504	2767736	2762652	2769376	2686860	s
	50	0,11020	0,10739	0,10101	0,07701	0,09225	e
		128,7983	129,0052	128,6593	128,2162	128,8866	t
		5418740	5420525	5421545	5409815	5431745	s

Tabella A.21: *Risultati simulated annealing SPE (D)*

		γ					
		0,3	0,4	0,5	0,6	0,7	
N	10	0,01984	0,02781	0,02179	0,03103	0,01989	e
		0,5919	0,0672	0,0600	0,0611	0,0711	t
		18947	21009	22746	27253	26872	s
	20	0,04193	0,05808	0,04933	0,02202	0,03432	e
		4,1980	3,9931	5,4981	4,1902	3,5921	t
		186452	198019	151201	181268	168289	s
	30	0,04850	0,05322	0,04892	0,03907	0,03189	e
		14,9112	13,9803	13,1090	12,0950	14,4098	t
		570192	598261	582162	553102	572156	s
	40	0,04312	0,04102	0,05801	0,04112	0,03928	e
		30,4983	32,4034	34,3130	31,9873	38,3082	t
		1362852	1370815	1392709	1429742	1402927	s
	50	0,05238	0,05000	0,05091	0,03981	0,04020	e
		67,4908	72,2098	70,1981	63,3211	62,2191	t
		2893542	2790193	2859023	2810098	2873946	s

Tabella A.22: *Risultati simulated annealing NE (D)*

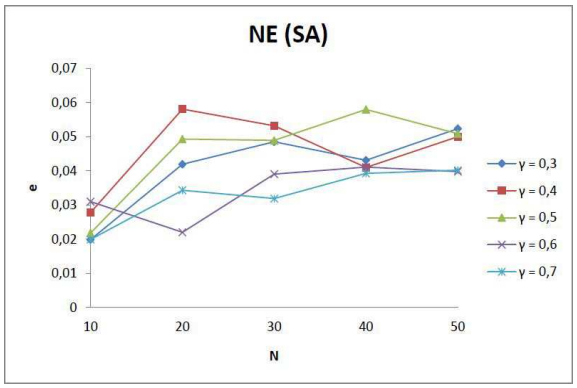

Figura A.61: *Errore NE–SA (D)*

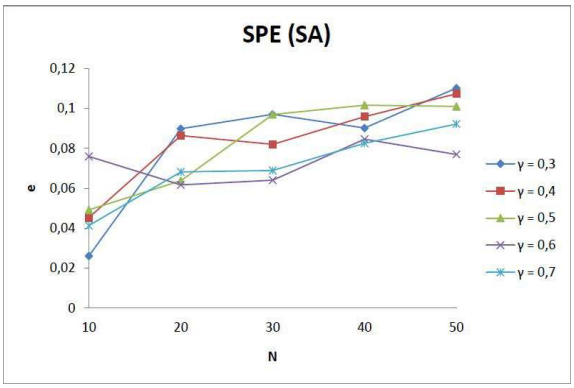

Figura A.62: *Errore SPE–SA (D)*

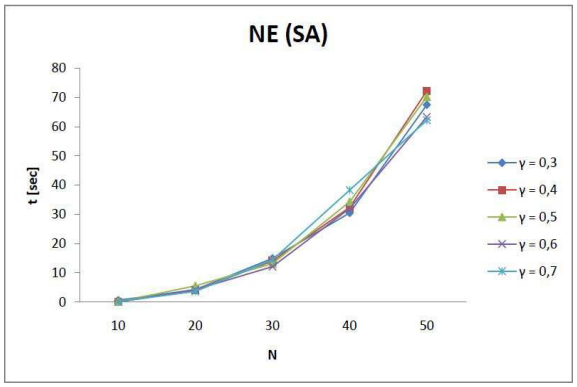

Figura A.63: *Durata NE–SA (D)*

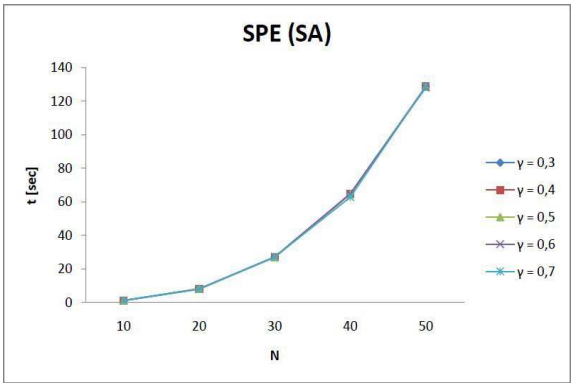

Figura A.64: *Durata SPE–SA (D)*

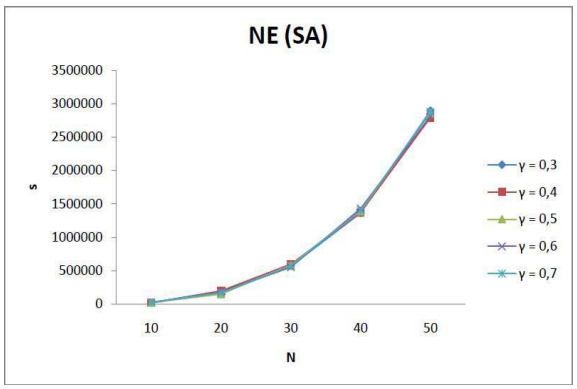

Figura A.65: *Campioni NE–SA (D)*

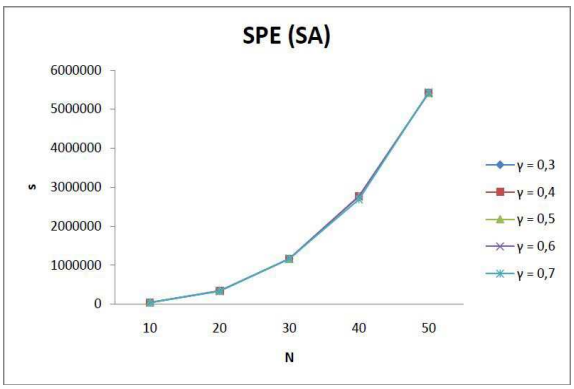

Figura A.66: *Campioni SPE–SA (D)*

A.4.3 Cross entropy

		γ					
		2	3	4	5	6	
N	10	0,13554	0,03478	0,05159	0,05501	0,14839	e
		0,5385	0,7981	0,8369	1,1071	1,0320	t
		13808	21595	22562	29906	27888	s
	20	0,02079	0,01517	0,01549	0,00275	0,04158	e
		3,1834	3,8597	4,7746	5,3609	5,2660	t
		98624	119710	147114	164918	162126	s
	30	0,02638	0,00505	0,01118	0,00862	0,01695	e
		9,8902	12,3331	13,0491	15,6092	16,4415	t
		318690	395571	418086	485814	511623	s
	40	0,00673	0,01320	0,01778	0,00510	0,00299	e
		22,8463	28,6551	28,7109	32,7372	32,0786	t
		725872	907340	907664	1037608	1013952	s
	50	0,01138	0,00679	0,00283	0,00348	0,00989	e
		42,2320	46,9875	54,8490	56,5079	64,3119	t
		1363635	1515150	1768180	1818180	2071210	s

Tabella A.23: *Risultati cross entropy per SPE (D)*

		γ					
		2	3	4	5	6	
N	10	0,06218	0,01393	0,02939	0,02391	0,07438	e
		0,2435	0,3858	0,4102	0,5802	0,6091	t
		7283	10273	11827	14998	15982	s
	20	0,01498	0,00713	0,00652	0,00193	0,01640	e
		1,8392	1,9348	2,3091	2,7980	2,9384	t
		58273	59982	71928	89726	84920	s
	30	0,01014	0,00698	0,00591	0,00383	0,00724	e
		5,3400	6,2092	6,3921	5,7213	8,3220	t
		172635	198825	220016	258371	296104	s
	40	0,00297	0,00649	0,00822	0,00239	0,00156	e
		11,3290	13,9320	14,9028	18,9343	17,1289	t
		398172	470012	418264	592712	572623	s
	50	0,00583	0,00239	0,00109	0,00158	0,00492	e
		24,9130	23,1980	28,3193	27,2140	31,9210	t
		693700	801633	892635	918009	1019264	s

Tabella A.24: *Risultati cross entropy per NE (D)*

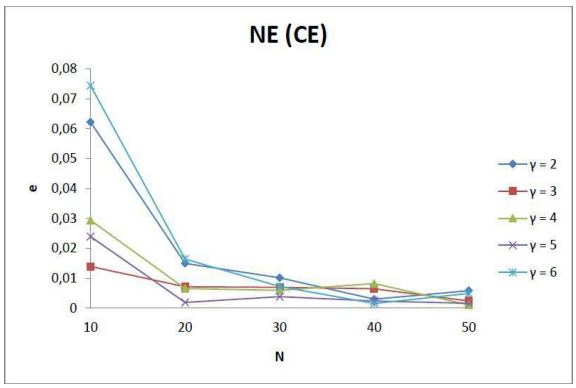

Figura A.67: *Errore NE–CE (D)*

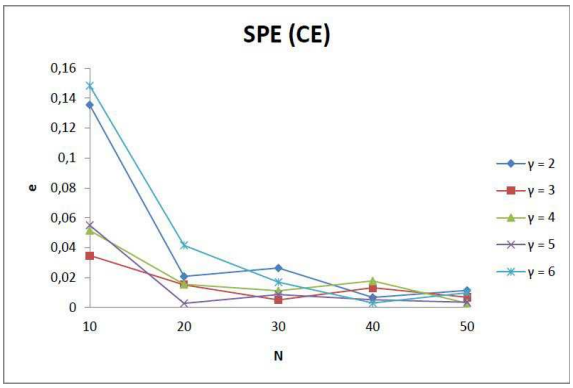

Figura A.68: *Errore SPE–CE (D)*

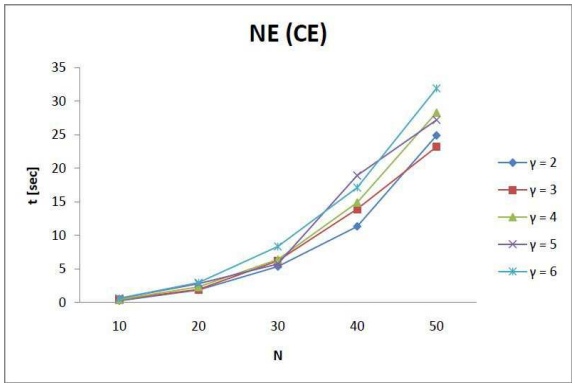

Figura A.69: *Durata NE–CE (D)*

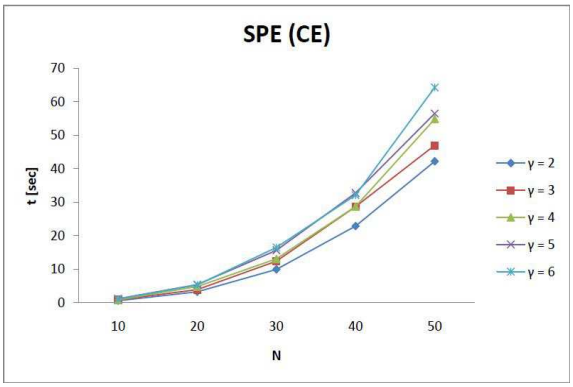

Figura A.70: *Durata SPE–CE (D)*

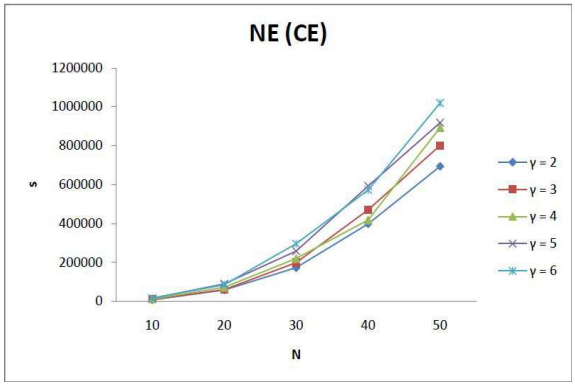

Figura A.71: *Campioni NE–CE (D)*

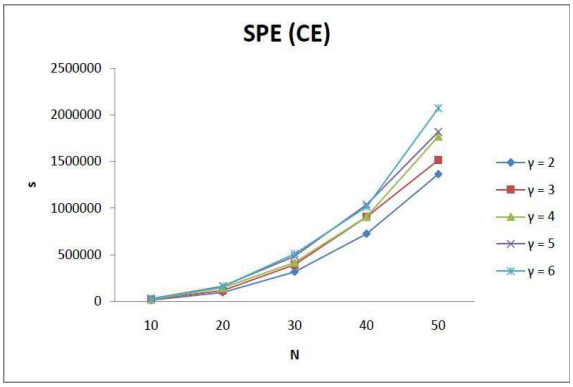

Figura A.72: *Campioni SPE–CE (D)*

Elenco delle figure

Elenco delle tabelle