POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Matematica

Tesi di Laurea Magistrale

Ranking tramite crowdsourcing: metodo dei minimi quadrati su grafi

Relatore
prof. Alberto Tarable
firma del relatore

. . . . . .
. . . . . .

Candidato
Francesco Bosio
firma del candidato

. . . . . .

Anno Accademico 2020-2021

A mio zio Guido,
una presenza preziosa
nella mia vita

Sommario

La tesi studia algoritmi di ranking di oggetti a partire da valutazioni rumorose che vengono prodotte da lavoratori su coppie di oggetti. In particolare, viene descritto e studiato nel dettaglio un algoritmo di ranking basato sul metodo dei minimi quadrati, nelle due versioni pesata e non pesata. All'algoritmo è associato un grafo, in cui i nodi sono gli oggetti e gli archi sono le coppie di oggetti che vengono valutate dai lavoratori. La tesi si propone di ottimizzare la struttura del grafo a partire da un budget totale di valutazioni. Viene studiato anche un approccio in cui le valutazioni sono divise in due stadi.

Ringraziamenti

Prima di procedere con la trattazione, vorrei dedicare qualche riga a tutti coloro che mi sono stati vicini in questo percorso di crescita personale e professionale. Un sentito grazie al mio relatore Alberto Tarable per la sua umanità, disponibilità, competenza e tempestività nel rispondere ad ogni mia richiesta. Grazie per la professionalità con la quale ha seguito il mio lavoro e per avermi fornito il materiale utile alla stesura dell'elaborato. Senza il totale supporto dei miei genitori, non sarei mai potuto arrivare fin qui. Grazie per esserci stati sempre. Ringrazio i miei compagni di studio di Cavallermaggiore per essermi stati accanto in questo periodo intenso e per gioire, insieme a me, dei traguardi raggiunti.

Grazie a tutti, senza di voi non ce l'avrei mai fatta.

Indice

Elenco delle figure

Ci sono giorni in cui esco ad osservare uno spaccapietre che martella la sua pietra, forse anche cento volte, senza che questa mostri la più piccola crepa.
Eppure, al centunesimo colpo, questa si spacca sempre in due e io so che non è stato l'ultimo colpo a farlo, ma tutto ciò che c'è stato prima.

[*Anonimo*]

Capitolo 1

Introduzione

L'obiettivo di questo capitolo è introdurre brevemente il problema che è oggetto di questa tesi e l'approccio utilizzato per risolverlo. In particolare introdurremo, in modo generale, i sistemi di crowdsourcing, ambito nel quale si colloca il nostro studio, e proporremo alcuni esempi di applicazioni tratti dalla realtà. Vedremo inoltre quali sono i principali contributi di questa tesi e come è strutturata.

1.1 Descrizione del problema ed introduzione ai sistemi di crowdsourcing

Il problema che questa tesi si pone di risolvere è, dato un insieme di N oggetti, quello di produrre, su essi, una classifica sulla base di una determinata caratteristica. Per assolvere questo task supponiamo di aver a disposizione dei lavoratori ai quali viene fornita una coppia di oggetti e viene richiesto di esprimere una preferenza. Sulla base di tali valutazioni si andrà a stimare, usando un approccio ai minimi quadrati (che sarà descritto nel dettaglio nel seguito), un valore, che chiameremo qualità, che sarà tanto più alto quanto più la caratteristica sulla base della quale vogliamo produrre le classifiche viene rilevata dai lavoratori. Una volta stimate le qualità di tutti gli oggetti, produrremo, in modo banale, la classifica richiesta sugli oggetti.

Tale approccio si colloca dunque nel contesto più ampio dei sistemi di crowdsourcing. Il crowdsourcing è una tipologia di attività online partecipativa nella quale una persona, o un ente qualsiasi, propone ad un gruppo di individui, mediante un annuncio aperto e flessibile, la realizzazione libera e volontaria di un compito specifico. Il compito che viene chiesto di essere assolto da parte degli individui può essere di complessità e modularità variabile ed essi possono partecipare apportando lavoro, denaro, conoscenze e/o esperienza. La realizzazione di tale compito implica sempre un beneficio per ambe le parti. L'utente otterrà infatti, come ricompensa per la sua partecipazione, il soddisfacimento di una concreta necessità, economica, di riconoscimento sociale, di autostima, o di sviluppo di capacità personali. Il crowdsourcer, d'altro canto, otterrà e utilizzerà a proprio beneficio il contributo offerto dall'utente, la cui forma dipenderà dal tipo di attività realizzata. Spesso il crowdsourcer non si rivolge all'intera massa di utenti disponibili, ma solo ad

alcuni individui che vengono selezionati, in quanto ritenuti interessanti, competenti o significativi.

Applicando sistemi di crowdsourcing, rispetto ai tradizionali metodi di problem solving, è possibile riscontrare alcuni benefici. I problemi che ci si pone di risolvere vengono infatti studiati ad un costo relativamente basso e, solitamente, vengono risolti in breve tempo. Il minor costo necessario è dovuto principalmente al fatto che il pagamento degli utenti è basato sul risultato o, in molti casi, non è proprio previsto. Tali utenti, infatti, sono spesso stimolati a svolgere il loro compito da una pura ricerca di personale soddisfazione intellettuale e non da un premio in denaro. L'organizzazione committente può inoltre avvalersi, selezionando in modo opportuno i potenziali utenti a cui assegnare il compito, di un numero di esperti maggiore rispetto a quelli già presenti all'interno dell'organizzazione stessa e questo può chiaramente portare a molti benefici in termini di prestazioni. Ascoltando la massa di utenti, inoltre, le organizzazioni hanno la possibilità di conoscere direttamente i desideri dei consumatori e questo aspetto è particolarmente utile nel nostro contesto applicativo. Può essere infatti spesso interessante che le classifiche vengano prodotte sulla base dei gusti reali di una porzione rappresentativa di utenti: questo ci permetterà di sfruttarle in modo più efficace a favore dell'intera massa. Non ultimo, in molti contesti applicativi è importante osservare come, grazie al contributo che offre, la comunità sviluppa un senso di appartenenza all'organizzazione e questo porta ad una conseguente fidelizzazione degli utenti.

Le più grandi critiche mosse ai sistemi di crowdsourcing sono invece di non produrre sempre ottimi risultati a livello qualitativo e di essere puramente utilizzati al fine di sfruttare i lavoratori con un costo minimo o addirittura nullo.

In base al task che si richiede di svolgere ai lavoratori, è possibile distinguere quattro diverse strategie di crowdsourcing:

- crowdfunding (finanziamento collettivo);
- crowdcreation (creazione collettiva);
- crowd wisdom (saggezza della folla);
- crowdvoting (votazione collettiva).

Poiché il compito che assegniamo ai nostri utenti è quello di esprimere un giudizio segnalando una preferenza su una coppia di oggetti, il nostro contesto più specifico è quello del crowdvoting. Una volta raccolte le informazioni parziali che ci vengono fornite da ciascun utente della "folla", noi, che rappresentiamo il crowdsourcer, le aggreghiamo con lo scopo di ottenere una qualche conoscenza. In questo senso il nostro studio si colloca anche in un contesto di crowd wisdom. Nel nostro caso specifico il crowdsourcer è infatti rappresentato da noi stessi e gli utenti (ai quali ci riferiremo da ora in avanti come lavoratori) saranno verosimilmente fruitori di un servizio o consumatori di un determinato prodotto. Noi decidiamo dunque di scegliere coppie di oggetti (che potranno essere libri, film, hotel, ristoranti o altro) e di delegare ai lavoratori il compito di esprimere preferenza fra i due rispetto ad una determinata caratteristica e secondo il loro gusto. Raccoglieremo infine i giudizi e li elaboreremo con modalità descritte nel dettaglio nei capitoli successivi e con lo scopo finale di ottenere una classifica.

Nel seguito presenteremo qualche esempio applicativo di sistemi di crowdsourcing che vengono attualmente implementati e sfruttati nella vita reale, anche se sussistono alcune differenze sostanziali rispetto al nostro caso specifico.

Amazon Mechanical Turk

Uno degli esempi più noti di sistemi di crowdsourcing è rappresentato da Amazon Mechanical Turk (MTurk), un sito web che permette alle aziende (dette Requesters) di assumere in modalità remota crowdworkers per eseguire determinati compiti che i computer non sono attualmente in grado di svolgere. Il nome Mechanical Turk è stato ispirato da "Il Turco", un'automa giocatore di scacchi del XVIII secolo realizzato da Wolfgang von Kempelen che girò l'Europa, battendo sia Napoleone Bonaparte che Benjamin Franklin. Fu rivelato in seguito che tale "macchina" non era affatto un'automa, ma che, in realtà, era un maestro di scacchi umano nascosto nell'armadio sotto la scacchiera che controllava i movimenti di un manichino umanoide. Allo stesso modo, il servizio online Mechanical Turk utilizza il lavoro umano a distanza nascondendolo dietro un'interfaccia digitale per aiutare i datori di lavoro ad eseguire compiti attualmente complessi o intrattabili per i computer. MTurk è gestito da Amazon Web Services, ed è di proprietà di Amazon. I datori di lavoro pubblicano lavori specifici che vengono denominati Human Intelligence Tasks (HITs), che possono essere molto vari e diversi fra loro. Fra questi troviamo, ad esempio: identificare contenuti specifici in un'immagine o un video, scrivere descrizioni di prodotti o rispondere a domande. I lavoratori, denominati in questo ambito Turkers o crowdworkers, possono visualizzare i lavori esistenti e svolgerli in cambio di una ricompensa economica stabilita dal datore di lavoro.

MTurk è stato lanciato pubblicamente il 2 novembre 2005. Dopo il suo lancio, la base di utenti di Mechanical Turk è cresciuta rapidamente. All'inizio-metà novembre 2005, erano già presenti decine di migliaia di lavori, tutti caricati nel sistema da Amazon stessa per alcuni dei suoi compiti interni che richiedevano intelligenza umana. Da allora i tipi di HIT si sono ampliati per includere la trascrizione, la valutazione, il tagging delle immagini, i sondaggi e la scrittura. Nel marzo 2007, secondo quanto riferito, si contavano più di 100000 lavoratori in oltre 100 paesi. Il numero è aumentato a oltre 500000 lavoratori registrati da oltre 190 paesi nel gennaio 2011. Nel 2018, una ricerca ha però dimostrato che, mentre in qualsiasi momento erano disponibili oltre 100000 lavoratori sulla piattaforma, solo circa 2000 stavano lavorando attivamente.

Nello specifico, esempi di lavori che possono essere commissionati e svolti su MTurk sono:

- Elaborazione di foto/video: questa tipologia di compiti risulta essere molto adatta all'intelligenza umana. I task in questione consistono nel richiedere ai lavoratori di etichettare eventuali oggetti trovati in un'immagine, selezionare l'immagine più rilevante in un gruppo di immagini, schermare i contenuti inappropriati e classificare gli oggetti nelle immagini satellitari. Compiti di questo tipo sono fondamentali, ad esempio, per poter essere in grado di eseguire il training di algoritmi supervisionati di Machine Learning , i quali, per svolgere task che riguardano immagini e video (quali, ad esempio, l'object detection), necessitano di una quantità importante di dati

commentati dall'uomo. Spesso, dunque, ricercatori di Machine Learning assumono lavoratori attraverso Mechanical Turk per produrre data set adatti al loro scopo;

- Raccolta di informazioni: la varietà e il grande numero di personale che MTurk mette a disposizione permette di raccogliere una quantità di informazioni di cui difficilmente si potrebbe entrare in possesso altrimenti. Mechanical Turk permette ad un qualsiasi ente o richiedente di accumulare un gran numero di risposte che gli vengono fornite riguardo vari tipi di indagini.

- Elaborazione dei dati: le aziende utilizzano MTurk per richiedere a lavoratori umani di svolgere compiti di elaborazione dei dati che sono più complesse e che dunque non possono essere svolte in modo efficace dai computer. Fra questi troviamo, ad esempio: l'editing e la trascrizione di podcast, la traduzione o la corrispondenza dei risultati dei motori di ricerca.

Esistono ulteriori task pubblicati frequentemente su MTurk, fra i quali troviamo la scrittura di commenti, descrizioni e voci di blog e siti web, la ricerca di elementi e di dati ed altro ancora.

Sistemi di raccomandazione (IMDb)

Ulteriori esempi di sistemi di crowdsourcing sono rappresentati dai sistemi di raccomandazione, cioè software di filtraggio di contenuti in grado di produrre raccomandazioni (anche personalizzate) per l'utente così da aiutarlo nelle sue scelte. Viene utilizzato per diversi prodotti, come libri, musica, film, video, notizie e social media. Noi vogliamo studiare più nel dettaglio il caso dei film ed, in particolare, il caso di IMDb.

IMDb (acronimo di Internet Movie Database) è un database online che contiene informazioni relative a film, programmi televisivi, home video, videogiochi e contenuti in streaming online, inclusi cast, personale di produzione e biografie personali, riassunti della trama, curiosità, valutazioni e recensioni di fan e critici. Originariamente un sito web gestito dai fan, il database è ora di proprietà e gestito da IMDb.com, Inc, una filiale di Amazon. A fine 2020 il data base di IMDb conteneva circa 7,5 milioni di titoli (compresi gli episodi) e 10,4 milioni di personaggi nel suo database, così come 83 milioni di utenti registrati.

Le pagine relative ai film ed agli attori di IMDb sono accessibili a tutti gli utenti di Internet, ma è necessario un processo di registrazione per poter apportare un contributo attivo nell'aggiornare le informazioni del sito.

In pieno accordo con quanto abbiamo visto valere per i sistemi di crowdsourcing, la maggior parte dei dati nel database è fornita da collaboratori volontari. Il sito permette agli utenti registrati di inviare nuovo materiale e di apportare modifiche alle voci esistenti. Gli utenti con una comprovata esperienza nell'invio di dati ricevono un'approvazione istantanea per aggiunte o correzioni al cast, ai crediti e ad altri dati demografici di prodotti e personaggi dei media. Tuttavia, l'immagine, il nome, il nome del personaggio, i riassunti della trama e le modifiche al titolo sono presumibilmente controllati prima della pubblicazione, e di solito impiegano tra le 24 e le 72 ore per apparire. Tutti gli utenti registrati possono scegliere il proprio nome del sito, e la maggior parte opera in modo

anonimo. Hanno una pagina di profilo che mostra da quanto tempo un utente registrato è membro, così come le valutazioni personali dei film (se l'utente decide di mostrarle) e, dal 2015, vengono aggiunti "badge" che rappresentano quanti contributi un particolare utente registrato ha presentato.

Il fatto più interessante, ai fini del nostro studio, è che gli utenti sono anche invitati a valutare i film su una scala da 1 a 10 e che tali voti vengono aggregati mediante una media ponderata che può essere visualizzata accanto a ciascun titolo. Tale aggregazione non consiste solo nel pesare le valutazioni, ma anche nel filtrarle. IMDb dichiara che i filtri sono usati per evitare il ballot stuffing. Il metodo utilizzato non è descritto nel dettaglio per evitare tentativi di aggirarlo. Quello che è possibile osservare è che talvolta si ha una grande differenza tra la media ponderata e la media aritmetica.

IMDb propone diverse tipologie possibili di classifiche. La più rilevante di esse è la Top 250 di IMDb, cioè una lista dei 250 film più alti in classifica, basata sulle valutazioni degli utenti registrati al sito utilizzando i metodi descritti. Oltre ad altre ponderazioni, i film Top 250 sono anche basati su una formula di valutazione ponderata nota in scienza attuariale come formula di credibilità. Il nome di questa formula nasce dal fatto che una statistica è considerata tanto più affidabile quanto è più alto il numero di singoli frammenti di informazioni che vengono prodotte da utenti idonei. Anche se la formula attuale non è divulgata, IMDb originariamente usava la seguente formula per calcolare il loro rating ponderato:

$$W = \frac{R \cdot v + c \cdot m}{v + m},$$

dove: W rappresenta la valutazione ponderata, R la valutazione media ottenuta dal film, espressa come numero da 1 a 10, v il numero di voti che vengono sfruttati per il film, m il minimo numero di voti richiesto affinché il film possa essere elencato nella Top 250 (attualmente 25.000) ed infine c il voto medio sull'intero insieme di film (attualmente 7.0).

Affinità e differenze dei sistemi presentati rispetto al nostro caso di studio

Come accennato, il nostro caso di studio specifico è un chiaro esempio di sistema di crowdsourcing. Si delega infatti ad utenti di un servizio o, più in generale, a lavoratori il compito di scegliere, una volta fornita loro una coppia di oggetti, quale è il loro preferito. Quindi tutte le valutazioni vengono raccolte ed elaborate da un sistema centrale, che aggrega, con modalità descritte nel dettaglio nei capitoli successivi, le informazioni parziali provenienti dagli utenti al fine di produrre una classifica che sia la più accurata possibile. Si possono notare dunque delle affinità con gli altri due sistemi di crowdsourcing presentati in precedenza. I compiti assegnati ai lavoratori nel nostro contesto possono, ed esempio, essere visti come un caso particolare di task proposto su Amazon Mechanical Turk. Se, infatti, il nostro scopo è di produrre una classifica basata sui gusti e le opinioni degli utenti, possiamo presentare loro coppie di oggetti, chiedere di esprimere una preferenza, raccogliere le informazioni prodotte e, sulla base di esse, generare una classifica. Per quanto riguarda IMDb, osserviamo come esso abbia come finalità, oltre a quella di fornire informazioni su film e, in generale, sul mondo del cinema, anche quella di generare una classifica dei migliori film sulla base dei gusti degli utenti. Tale obiettivo è del tutto simile a quello che vogliamo perseguire attraverso il nostro studio. Osserviamo però come

sussistano differenze sostanziali rispetto al nostro caso specifico: noi non chiediamo, ad esempio, agli utenti di valutare un singolo oggetto, ma, come detto, forniamo loro coppie di essi e chiediamo di esprimere una preferenza.
Inoltre, come vedremo nel dettaglio in seguito, a differenza di quanto avviene su IMDb e su Amazon Mechanical Turk, noi, per semplicità, assumiamo di avere a disposizione lavoratori indistinguibili fra loro e tali che producono valutazioni fra loro indipendenti.

1.1.1 Struttura generale della tesi e contributo originale

In seguito a questo capitolo introduttivo, nel successivo, dopo aver introdotto più nel dettaglio il problema in questione, viene presentata l'idea e gli elementi principali del modello utilizzato per risolverlo. Sempre nel capitolo 2 viene anche presentata la ricerca bibliografica ed i lavori affini a quello proposto in questa tesi. Il capitolo 3 è invece dedicato ad un trattazione teorica dell'algoritmo utilizzato per ricavare, mediante una stima ai minimi quadrati, le qualità degli oggetti. In particolare viene presentato, oltre al classico LS, anche una versione pesata dell'algoritmo detta WLS. Sempre in questo capitolo viene poi presentato uno studio asintotico, in termini analitici, di tale stimatore ai minimi quadrati e viene studiato l'algoritmo al variare di alcune strutture particolari per il grafo. Nel capitolo 4 vengono invece presentate le prestazioni degli algoritmi LS e WLS descritti nel capitolo 3, al quale ci riferiremo come algoritmo a singolo stadio, ricavate mediante simulazione quando abbiamo un numero finito di oggetti. Studieremo e confronteremo, in particolare, le prestazioni dell'algoritmo al variare della struttura scelta per il grafo. A tal fine presenteremo alcune nuove, al fine della nostra trattazione, tipologie di grafi, come quello a catena chiusa o multi-corona.

Il vero contributo originale della tesi è però discusso nel capitolo 5, dove viene presentato una variante dell'algoritmo discusso nel capitolo 3, alla quale ci riferiremo come approccio a due stadi. Presentiamo inoltre, in questo capitolo, le prestazioni dell'algoritmo ricavate mediante simulazione e vedremo come la versione non pesata soffra di un problema intrinseco, che verrà risolto dalla versione pesata dell'algoritmo a due stadi.

In appendice, infine, è possibile trovare i codici MATLAB utilizzati per le simulazioni descritte nel capitolo 4 e 5.

Capitolo 2

Descrizione del metodo utilizzato

2.1 Introduzione al problema e idea del modello utilizzato

Il problema che ci poniamo di risolvere in questa trattazione è, come detto, quello di proporre un algoritmo che permetta, dato un insieme di N oggetti, di formulare su di essi un ranking. La maggior parte degli algoritmi proposti per questo task stimano l'ordinamento degli oggetti a partire da un insieme di valutazioni o di confronti che vengono effettuati fra gli oggetti in questione. Presentiamo in questo capitolo solo le idee fondamentali del nostro approccio, riprese da Christoforou et al. [2021], e dell'algoritmo che ne deriva. Vedremo, in particolare, come le nostre classifiche saranno generate sulla base di valutazioni che ci vengono fornite da lavoratori. Chiaramente, siccome il comportamento dei lavoratori non è deterministico, occorrerà, come vedremo, adottare un modello probabilistico per descrivere il fenomeno in modo opportuno. L'idea è dunque quella di realizzare un algoritmo che, a partire da un insieme di valutazioni rumorose degli oggetti prodotte dai lavoratori, permetta di stimare nel modo più preciso ed affidabile possibile l'ordinamento reale degli oggetti. In particolare ci poniamo nella situazione in cui le valutazioni dei lavoratori consistono nell'identificazione del preferito fra coppie di oggetti. Per descrivere il risultato di tali confronti binari sono stati proposti in letteratura diversi modelli stocastici che permettono di rappresentare il comportamento dei lavoratori. A questi modelli appartengono ad esempio quelli di Bradley-Terry-Luce (BTL) e di Thurstone (che vengono descritti nel dettaglio in Plackett [1975], Luce [2012], Thurstone [1927] e Bradley and Terry [1952]).

Tali algoritmi hanno molte applicazioni pratiche, quali, ad esempio, produrre una classifica delle pagine web quando viene effettuata una ricerca su un motore di ricerca, oltre alla produzione di classifiche su hotel, ristoranti, giochi online o altro in base alle valutazioni che vengono forniti dai clienti (Richardson et al. [2007], Herbrich et al. [2006]).

L'algoritmo proposto in Christoforou et al. [2021] e che andremo a descrivere si basa

sulle seguenti assunzioni:

- gli oggetti da confrontare per produrre una classifica sono dotati di una qualità intrinseca, la quale non ci è però nota;
- la probabilità che un oggetto i venga preferito ad un oggetto j dipende unicamente dalle qualità q_i e q_j dei due oggetti.

Tali assunzioni, per come è posto il problema, appaiono ragionevoli. L'idea dell'algoritmo sarà, come vedremo, quella di ricostruire una stima delle qualità degli oggetti a partire dal confronto fra coppie di essi utilizzando un approccio ai minimi quadrati. Vedremo inoltre come sarà possibile stabilire un parallelismo fra il processo di stima delle qualità ed il ritorno cumulativo medio di un cammino casuale su un grafo pesato (vedremo come è possibile costruirlo in modo opportuno). Solo successivamente, sulla base di tale stime, verrà prodotto il ranking degli oggetti.

2.1.1 Descrizione del modello usato

Supponiamo di avere $\mathcal{Q} \subset \mathbf{R}$ insieme compatto ed N oggetti sui quali occorre produrre una classifica. Supponiamo, inoltre, che ogni oggetto i sia dotato di una qualità intrinseca $q_i \in \mathcal{Q}$, che non ci è però nota. Osserviamo come tali qualità inducano una graduatoria corretta r sugli oggetti, in modo che, dati due oggetti i e j, i precede j se e solo se $q(i) > q(j)$. Al fine di stimare le qualità intrinseche degli oggetti che vogliamo ordinare sfruttiamo, come anticipato, le valutazioni fornite dai lavoratori, i quali confrontano coppie di oggetti ed esprimono il loro giudizio selezionando l'oggetto che preferiscono fra i due. Questa procedura contiene implicitamente della stocasticità imputabile al giudizio dei lavoratori che, come già osservato in precedenza, non è deterministico. In particolare, le risposte dei lavoratori possono essere modellate come una collezione di variabili aleatorie binarie, la cui distribuzione dipende unicamente, come già ipotizzato in precedenza, dalle qualità intrinseche (non note) dei due oggetti che vengono confrontati. Tale randomicità nel processo di valutazione implica che la posizione stimata nella graduatoria per l'oggetto i, che denoteremo con $\hat{r}(i)$, non sempre coincide con quella reale $r(i)$. La bontà e l'affidabilità di $\hat{r}(i)$ dipende da come è organizzato e dunque modellato il processo di valutazione. In particolare dipende:

- dal comportamento dei lavoratori;
- dalla scelta dell'insieme di coppie di oggetti che decidiamo di confrontare;
- dal numero di lavoratori che abbiamo a disposizione per esprimere un giudizio e dalla scelta di come distribuire tali lavoratori sulle varie coppie di oggetti da confrontare;
- dall'algoritmo che utilizziamo per stimare il ranking degli oggetti a partire dalle risposte fornite dai lavoratori.

Modello di comportamento dei lavoratori

Per semplicità, nel seguito, assumeremo che tutti i lavoratori siano indistinguibili (e dunque che si comportino allo stesso modo) e che il loro modello di comportamento, che abbiamo visto essere stocastico, sia noto dal sistema. In particolare un lavoratore che confronta due oggetti i e j preferirà l'oggetto i all'oggetto j con una probabilità data da:

$$p_{i,j} = F(q_i - q_j) \tag{2.1}$$

dove la funzione F viene scelta differenziabile, strettamente crescente nei suoi argomenti, e tale che $F(0) = \frac{1}{2}$. Il fatto che la funzione sia strettamente crescente implica inoltre che essa sia invertibile. Osserviamo come le caratteristiche descritte e richieste per la F siano quelle necessarie per modellare la probabilità che l'oggetto i sia preferito a j, in quanto F strettamente crescente implica che tanto maggiore q_i è di q_j e tanto più è probabile che i venga preferito a j. Inoltre se $q_i = q_j$ allora, poiché $F(0) = \frac{1}{2}$, la probabilità che un qualsiasi lavoratore scelga i è uguale a quella che scelga j. Inoltre assumiamo che $F'(q) > 0$ per ogni $q \in \bar{\mathcal{Q}}$, dove $\bar{\mathcal{Q}} = \{q_i - q_j | q_i, q_j \in \mathcal{Q}\}$. Osserviamo inoltre che, poiché $p_{i,j}$ esprime una probabilità, vale che: $p_{i,j} = 1 - p_{j,i}$.

Dunque, scelti a caso un lavoratore ed una coppia di oggetti (i, j) che devono essere confrontati, la risposta del lavoratore può essere modellata come una variabile aleatoria binaria $w_{i,j} \in \{0,1\}$, i cui valori sono dati da:

$$\mathbf{P}(w_{i,j} = 1) = p_{i,j}$$

$$\mathbf{P}(w_{i,j} = 0) = 1 - p_{i,j} = p_{j,i}$$

Il modello (2.1) è molto generale ed in particolare è possibile, a partire da tale modello, costruire i due seguenti modelli più particolari:

- il modello di Thurstone (Thurstone [1927]), dove la scelta sulla preferenza fra i due oggetti viene effettuata sulla base delle qualità così come vengono percepite da parte dei lavoratori. Ricordiamo infatti come le qualità intrinseche reali non sono note al sistema ed in particolare sono sconosciute anche ai lavoratori. Definiamo allora le qualità percepite dai lavoratori come:

  $$\tilde{q_i} = q_i + n_i, \qquad \tilde{q_j} = q_j + n_j$$

  dove n_i ed n_j sono variabili aleatorie con media zero che rappresentano il rumore nella stima delle qualità che avvertono i lavoratori. Osserviamo che, definita $\eta_{i,j} = n_i - n_j$:

  $$p_{i,j} = F(q_i - q_j) = \mathbf{P}(\tilde{q_i} < \tilde{q_j}) = \mathbf{P}(q_j + n_j < q_i + n_i) = \mathbf{P}(\eta_{i,j} < q_i - q_j)$$

  Vale dunque che la funzione F nella configurazione generale (2.1) è in questo contesto una funzione di distribuzione cumulativa di una variabile aleatoria $\eta_{i,j}$ che ha valore atteso nullo.

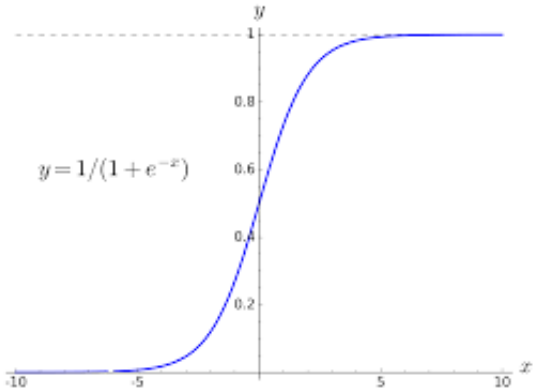


Figura 2.1. Grafico della funzione sigmoidea

- il modello di Bradley-Terry-Luce (BTL, Plackett [1975], Luce [2012]) dove viene operata una scelta di una funzione sigmoidea per rappresentare la F generale in (2.1). Vale allora che:

$$p_{i,j} = F(q_i - q_j) = \frac{e^{q_i - q_j}}{1 + e^{q_i - q_j}} = \frac{1}{1 + e^{-(q_i - q_j)}}$$

  Tale scelta è coerente con le caratteristiche che abbiamo visto essere opportune per la funzione F in (2.1) in quanto è derivabile, strettamente crescente (e dunque ha derivata prima sempre strettamente maggiore di 0), e vale che: $F(0) = \frac{1}{2}$. Tale funzione mappa inoltre $\mathbf{R}$ in (0,1) e, come era auspicabile, satura a 0 per valori di q_j molto più grandi rispetto a quelli di q_i e viceversa, per valori di q_i molto più grandi rispetto a quelli di q_j, satura ad 1.

Descrizione del grafo indotto dalle valutazione espresse dai lavoratori

Sia ora $\mathcal{V} = \{1, ..., N\}$ l'insieme di oggetti su cui occorre produrre una classifica. Osserviamo come una qualsiasi scelta di un insieme di coppie di oggetti, denotato con $\xi \subseteq \mathcal{V} \times \mathcal{V}$, che devono essere confrontate da un certo numero di lavoratori induce un grafo indiretto $\mathcal{G} = (\mathcal{V}, \xi)$, i cui vertici sono gli oggetti da ordinare e le coppie di oggetti che vengono effettivamente confrontate per effettuare l'ordinamento costituiscono gli archi. Osserviamo inoltre che, come ci viene suggerito dall'intuizione, è possibile inferire un ranking completo degli N oggetti solo nel caso in cui $\mathcal{G}$ sia connesso, in quanto, in caso contrario, averemo un oggetto o un gruppo di oggetti che non è connesso, mediante un cammino qualsiasi, con tutti gli altri. Per questo motivo, nel seguito della nostra trattazione, assumeremo di lavorare sempre in contesti dove il grafo $\mathcal{G}$, che viene indotto dagli oggetti e dai confronti che vengono effettuati su di esso, sia connesso. Anche nei capitoli 3 e 4, dove il nostro scopo sarà quello di costruire grafi che permettano all'algoritmo di realizzare il miglior ordinamento possibile sugli oggetti dati, terremo presente questo fatto.

Accenno alla misurazione delle prestazioni dell'algoritmo

Come accennato, ad ogni coppia $(i, j) \in \xi$ viene assegnato un certo numero di valutazioni W prodotte dai lavoratori e, in generale, all'aumentare di W la stima del ranking prodotto sugli oggetti migliora. Se però prendiamo in esame la complessità totale C dell'algoritmo che andiamo a proporre, osserviamo come questa sia proporzionale al numero totale di valutazioni di cui si usufruisce, che, supponendo di utilizzare lo stesso numero di valutazioni per tutti i confronti da effettuare, è dato da:

$$C = |\xi|W$$

Osserviamo dunque come un buon algoritmo di ranking debba risolvere un problema di trade-off fra la complessità C dell'algoritmo e l'affidabilità del ranking che andiamo a proporre. Si cercherà dunque di ricavare una classifica che sia sufficientemente corretta utilizzando il numero minimo di comparazione fra coppie. Affronteremo questo problema più nel dettaglio nei capitoli 4 e 5, dove fisseremo il numero massimo di valutazioni che i lavoratori possono produrre (e dunque fisseremo la complessità dell'algoritmo), e cercheremo di generare, in questo contesto, il miglior ordinamento possibile. A tale scopo interverremo sulla struttura del grafo e dunque sulla scelta delle coppie sulle quali effettuare i confronti.

Per misurare l'affidabilità e la bontà di un ordinamento prodotto mediante il nostro algoritmo diciamo che esso è ϵ-quality approximately correct (o ϵ-quality ranking) se $\hat{r}(i) \prec \hat{r}(j)$, ogni volta che vale che $q_i \geq q_j + \epsilon$. Inoltre un algoritmo di ranking si dice (ϵ, δ)-PAC se garantisce di trovare un ϵ-quality ranking con una probabilità maggiore di $1 - \delta$. In questo lavoro, come avviene in Christoforou et al. [2021], Szörényi et al. [2015], Falahatgar et al. [2017] e Falahatgar et al. [2018] la bontà e l'affidabilità delle classifiche prodotte viene misurata in questo modo.

2.2 Contributo della tesi e lavori correlati

Il principale contributo di questa tesi, che verrà discusso nel dettaglio nei capitoli 4 e 5, sta nella ricerca di una "ricetta" pratica ed ottimale per la costruzione del grafo per ottenere, fissata una certa complessità (cioè un certo numero di valutazioni che i lavoratori possono produrre sulle coppie di oggetti), il miglior ranking possibile, dato un numero relativamente basso di oggetti. Costruire tale grafo significa di fatto progettare l'esperimento data una complessità fissata, ovvero selezionare le coppie di oggetti da confrontare e decidere il numero di valutazioni da assegnare a ciascun confronto nel modo migliore possibile. Nel capitolo 5, in particolare, verrà discusso un algoritmo a due stadi che permetterà di ottenere, a parità di complessità C, classifiche più accurate.

Il modello e l'algoritmo di base dal quale si è preso spunto per lo studio successivo (e che verrà discusso nel dettaglio nel secondo capitolo) è preso da Christoforou et al. [2021]. In questo articolo in particolare si studiano i limiti fondamentali degli algoritmi di ranking basati su confronti a coppie in condizioni nelle quali il numero di oggetti che si prende in esame è alto e in cui si ha a disposizione un gran numero di valutazioni. I risultati principali di questo articolo sono un completamento ed un'estensione dei risultati

precedenti che riguardano lo studio della complessità minima di algoritmi di ranking sotto diversi modelli di preferenza non parametrici, recentemente derivati in Falahatgar et al. [2017] ed in Falahatgar et al. [2018]. In Falahatgar et al. [2017] ed in Falahatgar et al. [2018] viene infatti mostrato come l'efficienza di questi algoritmi sia fortemente determinata da come viene strutturato il modello di comportamento dei lavoratori, cioè il modello di scelta della preferenza fra coppie di oggetti.

Quando si considerano i modelli parametrici, come vedremo, la stima di una classifica è essenzialmente legata alla stima delle qualità. Gli autori in Shah et al. [2015] e Hendrickx et al. [2019] forniscono una caratterizzazione della distanza attesa tra qualità stimate e quelle vere in norma 2, indicate in seguito come errore quadrato medio (MSE), in relazione alle proprietà di un grafo fissato $\mathcal{G}$. Il recentissimo articolo Hendrickx et al. [2019], solo per il modello di Bradley-Terry-Luce (BTL), introduce un algoritmo LS e fornisce limiti superiori e inferiori per una variante del MSE e le relative probabilità di coda ottenibili da tale algoritmo, caratterizzandolo in termini di robustezza del grafo.

Lavori interessanti sono anche Jang et al. [2016], Chen and Suh [2015], Negahban et al. [2012], Negahban et al. [2017], Hirani et al. [2010], Cucuringu [2016], Shah and Wainwright [2017] e d'Aspremont et al. [2019] dove viene introdotto un approccio ai minimi quadrati per questo task, ma non vengono fornite garanzie sulle prestazioni dal punto di vista teorico. In particolare, Cucuringu [2016] propone Sync-Rank, un algoritmo di programmazione semidefinita basato sulla sincronizzazione angolare. In Jang et al. [2016], Chen and Suh [2015], invece, viene presentato un algoritmo iterativo che emula un cammino casuale pesato su un grafo $\mathcal{G}$ e le sue prestazioni sono analizzate sotto il modello BTL. Gli articoli precedenti forniscono limiti sul MSE ottenibile (cioè la distanza nella norma 2 tra la qualità vera e quella stimata dall'algoritmo) e le corrispondenti probabilità di coda (cioè la probabilità che la distanza tra le qualità vere e stimate superi una data soglia). Un confronto diretto tra le prestazioni degli algoritmi proposti in Hendrickx et al. [2019], Negahban et al. [2012], Negahban et al. [2017] è riportato in Hendrickx et al. [2019] dove l'approccio LS è dimostrato essere, in generale, asintoticamente più efficiente. Sotto il modello BTL Jang et al. [2016] e Chen and Suh [2015] propongono e analizzano algoritmi in grado di identificare, dato un insieme di oggetti, quelli che sono nella top-k all'interno della classifica. Un altro articolo interessante da questo punto di vista è Shah and Wainwright [2017], il quale propone un algoritmo con un approccio simile a quello di questa trattazione che raggiunge i limiti teorici dell'informazione a meno di una costante moltiplicativa. Tale algoritmo è robusto in quanto la sua ottimalità è mantenuta senza che si debbano imporre condizioni sulla matrice di probabilità di confronto a coppie, in contrasto con altri lavori, come il nostro, che si applicano solo al modello parametrico BTL ed è efficiente a livello computazionale. Viene affrontato il problema del ranking esatto, e, per il problema del ranking top-k, i risultati sono estesi per ottenere garanzie per il ranking approssimativo. Infine d'Aspremont et al. [2019] descrive un algoritmo di ranking basato sulla decomposizione in valori singolari, assumendo che i lavoratori restituiscano stime non quantificate e rumorose delle differenze fra le qualità.

Per quanto concerne gli algoritmi di ranking online in Szörényi et al. [2015], per il modello BTL, viene descritto un algoritmo online, ispirato ad una versione a budget finito del quick sort, in grado di ottenere una classifica (ϵ, δ)-PAC con $O(\frac{N}{\epsilon^2} \log N \log(\frac{N}{\delta}))$ confronti. In Heckel et al. [2019] si dimostra che, per gli algoritmi di classificazione online,

modelli parametrici aiutano a ridurre la complessità solo di fattori logaritmici.

In Christoforou et al. [2021], articolo dal quale, come detto, prendiamo il modello e l'algoritmo, a differenza di Hendrickx et al. [2019], introduciamo un modello di preferenza parametrico piuttosto generale secondo il quale le probabilità di preferenza sono determinate da una funzione monotona e liscia arbitraria che ha come argomento la differenza fra le qualità degli oggetti. In questo scenario, in Christoforou et al. [2021] viene mostrato come gli algoritmi non adattivi per un ordinamento ottimale possono essere definiti senza la necessità di introdurre alcuna restrizione al parametro δ. In particolare, a differenza di quanto viene fatto in Shah et al. [2015] e Hendrickx et al. [2019], in Christoforou et al. [2021] si lavora nel contesto PAC ed in particolare viene mostrato come gli algoritmi proposti siano (ϵ, δ)-PAC, a condizione che $O(\frac{N}{\epsilon^2} \log \frac{N}{\delta})$ valutazioni siano assegnate in modo casuale in un solo stadio.

Il nostro approccio al task descritto, basata su quella proposta in Christoforou et al. [2021], si basa sulla ricostruzione delle qualità degli oggetti a partire dalle differenze di qualità fra coppie di essi, adottando un approccio LS simile a quello che viene proposto in Hendrickx et al. [2019]. Va tuttavia osservato come in Hendrickx et al. [2019] l'analisi proposta può essere applicata solo nel caso in cui vengono eseguiti in totale $\Omega(N \log^2 \frac{N}{\delta})$ e quindi non è utile per rispondere all'esigenza di dimostrare l'esistenza di algoritmi ottimali nel senso dell'ordine.

È inoltre interessante osservare come in Christoforou et al. [2021] venga stabilito un parallelismo tra la stima della qualità e la ricompensa cumulativa delle passeggiate casuali su grafi e che, come contributo originale, viene introdotto un algoritmo LS pesato (WLS) con prestazioni molto vicine all'algoritmo ML, che è però più complesso. Utilizzeremo tale algoritmo nel nostro studio e ne studieremo le prestazioni in casi particolari. In Christoforou et al. [2021] viene inoltre dimostrato, mediante simulazione, che le prestazioni dei nostri algoritmi sono estremamente buone anche in scenari non asintotici.

Un altro lavoro interessante è Zhai [2010], dove vengono presentati diversi approcci ed algoritmi non numerici per stimare un ranking basandosi sui confronti fra coppie di oggetti. In questo lavoro si considera inoltre il problema di avere oggetti in qualche misura non coerenti fra loro e dunque non inseribili all'interno di un'unica classifica. L'autore si preoccupa dunque di produrre solamente classifiche su oggetti coerenti fra loro.

In Gutin and Yeo [1996] il problema descritto viene invece visto e trasformato nel problema equivalente del sequenziamento del lavoro di una singola macchina per minimizzare il tempo totale di completamento del lavoro soggetto a vincoli di precedenza su "catene parallele". Questo fornisce un algoritmo per generare classifiche ottimali ed un algoritmo polinomiale per ricavare il rango medio di ogni vertice.

Capitolo 3

Stima ai minimi quadrati delle qualità degli oggetti

In questo capitolo presentiamo un semplice algoritmo di stima lineare, basato sul criterio dei minimi quadrati, che è possibile applicare su grafi connessi $\mathcal{G} = (\mathcal{V}, \xi)$.

3.1 Descrizione dell'algoritmo

Come punto di partenza definiamo la distanza fra due oggetti i e j come:

$$d_{i,j} = q_i - q_j.$$

Sia $W_{i,j}$ l'insieme delle W risposte binarie fornite dai lavoratori immaginari per confrontare la coppia di oggetti (i, j). Chiamiamo, a questo punto, $K_{i,j}$ il numero di volte che l'oggetto i è preferito dai lavoratori rispetto all'oggetto j. Dunque, per come è stato definito, $K_{i,j}$ è una variabile aleatoria con distribuzione binomiale: $K_{i,j} \sim Bin(W, p_{i,j})$, dove $p_{i,j} = F(q_i - q_j) = F(d_{i,j})$. Dunque, come risultato delle valutazioni, otteniamo una stima $\hat{d}_{i,j}$ della distanza fra le due osservazioni. Vale infatti che, definito $y_{i,j} = \hat{p}_{i,j} - p_{i,j}$ l'errore che viene commesso sulla stima di $p_{i,j}$:

$$F(\hat{d}_{i,j}) = \hat{p}_{i,j} \Rightarrow \hat{d}_{i,j} = F^{-1}(\hat{p}_{i,j}) = F^{-1}(y_{i,j} + p_{i,j}),$$

dove la stima di $p_{i,j}$ viene fornita da $\hat{p}_{i,j} = \frac{K_{i,j}}{W}$. Osserviamo come $y_{i,j}$ sia una variabile aleatoria tale che:

- $\mathbf{E}[y_{i,j}] = \mathbf{E}[\hat{p}_{i,j} - p_{i,j}] = \mathbf{E}[\frac{K_{i,j}}{W}] - \mathbf{E}[p_{i,j}] = \frac{1}{W}\mathbf{E}[K_{i,j}] - p_{i,j} = \frac{W}{W}p_{i,j} - p_{i,j} = 0$;
- $Var(y_{i,j}) = \mathbf{E}[y_{i,j}^2] - (\mathbf{E}[y_{i,j}])^2 = \mathbf{E}[(\hat{p}_{i,j} - p_{i,j})^2] = \mathbf{E}[\hat{p}_{i,j}^2] + \mathbf{E}[p_{i,j}^2] - 2\mathbf{E}[p_{i,j}\hat{p}_{i,j}] =$

  $= \frac{1}{W}\mathbf{E}[K_{i,j}^2] + p_{i,j}^2 - 2p_{i,j}\mathbf{E}[\hat{p}_{i,j}] = \frac{1}{W^2}(Var(K_{i,j}) + (\mathbf{E}[K_{i,j}])^2) + p_{i,j}^2 - 2p_{i,j}^2 =$

  $= \frac{1}{W^2}(Wp_{i,j}(1 - p_{i,j}) + W^2p_{i,j}^2) - p_{i,j}^2 = \frac{p_{i,j}(1-p_{i,j})}{W}$.

Dunque, come conseguenza di questo fatto, anche la stima di $d_{i,j}$ è data da:

$$\hat{d}_{i,j} = d_{i,j} + z_{i,j},$$

dove $z_{i,j}$ rappresenta l'errore sulla stima di $d_{i,j}$ ed è indotto dalla presenza di $y_{i,j}$ nella stima di $p_{i,j}$.

Date dunque $\{\hat{d}_{i,j}, (i,j) \in \xi\}$ insieme di stime rumorose di $\{d_{i,j}, (i,j) \in \xi\}$, vogliamo produrre una stima $\hat{\mathbf{q}} = [\hat{q}_1, ..., \hat{q}_N]^T$ di $\mathbf{q} = [q_1, ..., q_N]^T$. Per farlo scegliamo di impostare, e poi risolvere, il seguente problema di ottimizzazione ai minimi quadrati:

$$\hat{\mathbf{q}} = \arg\min_{\mathbf{x}} \sum_{(i,j)\in\xi} \omega_{i,j}(x_i - x_j - \hat{d}_{i,j})^2,$$

dove gli $\omega_{i,j}, \forall (i,j) \in \xi$ rappresentano i pesi arbitrari e positivi che risultano essere molto utili al fine di migliorare le prestazioni dell'algoritmo e che verranno discussi nel dettaglio in seguito. Risolvendo tale problema di ottimizzazione cerchiamo, infatti, di minimizzare la somma su tutti gli $(i,j) \in \xi$ di $(\hat{q}_i - \hat{q}_j - \hat{d}_{i,j})^2$, che è tanto più bassa quanto la stima che viene prodotta è accurata.

Per trovare una soluzione a tale problema di ottimizzazione imponiamo la condizione del primo ordine:

$$\frac{\partial}{\partial x_i} \sum_{(i,j)\in\xi} \omega_{i,j}(x_i - x_j - \hat{d}_{i,j})^2 = 2 \sum_{j\in\mathcal{N}_i} \omega_{i,j}(x_i - x_j - \hat{d}_{i,j}) =$$
$$2(\sum_{j\in\mathcal{N}_i} \omega_{i,j})x_i - 2 \sum_{j\in\mathcal{N}_i} \omega_{i,j}(x_j + \hat{d}_{i,j}) = 0, \forall i = 1, ..., N$$

Qui $\mathcal{N}_i = \{j \in \mathcal{V}/(i,j) \in \xi \vee (j,i) \in \xi\}$ rappresenta l'insieme dei vicini del nodo i. Definiamo ora il grado generalizzato $\rho_i = \sum_{j\in\mathcal{N}_i} \omega_{i,j}$ del nodo i (cioè la somma dei pesi relativi agli archi che connettono i ai suoi vicini) e vale che:

$$x_i = \sum_{j\in\mathcal{N}_i} \omega_{i,j} \frac{x_j + \hat{d}_{i,j}}{\rho_i}, \quad \forall i = 1, ..., N$$

Osserviamo inoltre come:

$$\frac{\partial}{\partial x_i} 2 \sum_{(i,j)\in\xi} \omega_{i,j}(x_i - x_j - \hat{d}_{i,j}) = 2 \sum_{j\in\mathcal{N}_i} \omega_{i,j} > 0, \quad \forall i = 1, ..., N$$

Allora vale che la soluzione del problema di ottimizzazione posto è data dalla soluzione del seguente sistema lineare:

$$\hat{q}_i = \sum_{j\in\mathcal{N}_i} \omega_{i,j} \frac{\hat{q}_j + \hat{d}_{i,j}}{\rho_i}, \quad \forall i = 1, ..., N \tag{3.1}$$

Abbiamo dunque ricavato un sistema lineare $N \times N$ e, per risolverlo, occorre riscriverlo in forma matriciale. A tal scopo definiamo la matrice $\tilde{\mathbf{H}}$ associata al grafo $\mathcal{G}$, le cui componenti sono definite come:

$$[\tilde{\mathbf{H}}]_{i,j} = \begin{cases} \frac{\omega_{i,j}}{\rho_i}, & (i,j) \in \xi \\ 0, & \text{altrove} \end{cases}$$

Definiamo ora le matrici:

- $\mathbf{Z}$ tale che $\mathbf{Z}_{i,j} = \{z_{i,j}\}$;
- $\mathbf{D}$ tale che $\mathbf{D}_{i,j} = \{d_{i,j}\}$, la matrice (antisimmetrica) dei valori reali delle differenze fra le qualità degli oggetti;
- $\hat{\mathbf{D}}$ tale che $\hat{\mathbf{D}}_{i,j} = \{\hat{d}_{i,j}\}$, la matrice (antisimmetrica) dei valori stimati delle differenze fra le qualità degli oggetti.

Il sistema lineare (3.1) può essere riscritto come:

$$\hat{q}_i = \sum_{j \in \mathcal{N}_i} \frac{\omega_{i,j}}{\rho_i} \hat{q}_i + \sum_{j \in \mathcal{N}_i} \frac{\omega_{i,j}}{\rho_i} \hat{d}_{i,j}, \qquad i = 1, ..., N$$

e dunque può essere riscritto nella forma matriciale:

$$\hat{\mathbf{q}} = \tilde{\mathbf{H}}\hat{\mathbf{q}} + (\tilde{\mathbf{H}} \odot \hat{\mathbf{D}})\mathbf{1} \tag{3.2}$$

dove $\mathbf{1} = [1, ...,1]^T$ è un vettore colonna di dimensione N e $\odot$ indica l'operazione di prodotto componente per componente per matrici della stessa dimensione. Indicata ora con $\mathbf{I}$ la matrice identità $N \times N$, definiamo la matrice $\tilde{\mathbf{M}} = \mathbf{I} - \tilde{\mathbf{H}}$ e riscriviamo il sistema lineare (3.2) come:

$$\tilde{\mathbf{M}}\hat{\mathbf{q}} = (\tilde{\mathbf{H}} \odot \hat{\mathbf{D}})\mathbf{1}$$

Osserviamo che vale: $\hat{\mathbf{D}} = \mathbf{D} + \mathbf{Z}$, dunque è possibile scrivere:

$$\tilde{\mathbf{M}}\hat{\mathbf{q}} = (\tilde{\mathbf{H}} \odot \hat{\mathbf{D}})\mathbf{1} = (\tilde{\mathbf{H}} \odot \mathbf{D})\mathbf{1} + (\tilde{\mathbf{H}} \odot \mathbf{Z})\mathbf{1} \tag{3.3}$$

Ricordiamo di aver assunto, al fine di poter ricavare una classifica completa degli oggetti assegnati, di considerare solamente grafi $\mathcal{G}$ connessi. Dunque, per come è stato definita, $\tilde{\mathbf{M}}$ ha lo stesso rango di $\tilde{\mathbf{H}}$ e, avendo il grafo una sola componente connessa, vale che: $\text{rank}(\tilde{\mathbf{M}}) = N - 1$. Dunque $\tilde{\mathbf{M}}$ è singolare ed è possibile verificare facilmente che: $\tilde{\mathbf{M}}\mathbf{1} = \mathbf{0}$. Infatti è immediato verificare che, per come è definita $\tilde{\mathbf{D}}$, la somma degli elementi extra-diagonali di $\tilde{\mathbf{M}}$ è uguale a -1:

$$\sum_{j \neq i} -\frac{\omega_{i,j}}{\rho_i} = -\frac{1}{\rho_i} \sum_{j \neq i} \omega_{i,j} = -\frac{\rho_i}{\rho_i} = -1, \qquad i = 1, ..., N$$

Se per ogni riga sommo tale quantità con l'elemento diagonale (che vale chiaramente 1 in ogni riga) ottengo $\mathbf{0}$ (vettore verticale di tutti 0 di dimensione N). Questo fatto implica che l'operatore lineare associato alla matrice $\tilde{\mathbf{M}}$ su $\mathbf{R}^N$ non è iniettivo ed in particolare che se $\hat{\mathbf{q}}'$ è soluzione del sistema lineare, allora anche $\hat{\mathbf{q}}'' = \hat{\mathbf{q}}' + \alpha\mathbf{1}$ è una soluzione per ogni $\alpha \in \mathbf{R}$. Osserviamo come, al fine di proporre un ranking degli oggetti, il valore assunto dal parametro α è irrilevante, in quanto ogni soluzione della forma: $\hat{\mathbf{q}}'' = \hat{\mathbf{q}}' + \alpha\mathbf{1}$ induce lo stesso ordinamento nel ranking degli oggetti. Possiamo allora decidere di fissare, ad esempio, la qualità dell'oggetto N a 0 (scelta arbitraria) e prendere dunque $q_N = 0$ come quantità di riferimento. Al fine di tenere in conto di tale scelta definiamo le nuove matrici:

- $\mathbf{H}$ tale che:

$$[\mathbf{H}]_{i,j} = \begin{cases} [\tilde{\mathbf{H}}]_{i,j}, & i < N, \forall j \\ 0, & i = N, \forall j \end{cases}$$

- $\mathbf{M} = \mathbf{I} - \mathbf{H}$.

Sostituendo tali matrici in (3.3) abbiamo che:

$$\mathbf{M}\hat{\mathbf{q}} = (\mathbf{H} \odot \hat{\mathbf{D}})\mathbf{1} = (\mathbf{H} \odot \mathbf{D})\mathbf{1} + (\mathbf{H} \odot \mathbf{Z})\mathbf{1}$$

Poiché ora $\mathbf{M}$ ha rango pieno (ed è dunque invertibile) è possibile risolvere per $\mathbf{q}$ il sistema precedente ed ottenere:

$$\hat{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{H} \odot \hat{\mathbf{D}})\mathbf{1} \tag{3.4}$$

Osserviamo inoltre come: $\mathbf{q} = \mathbf{M}^{-1}(\mathbf{H} \odot \mathbf{D})\mathbf{1}$, dunque vale che:

$$\hat{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{H} \odot \hat{\mathbf{D}})\mathbf{1} = \mathbf{M}^{-1}(\mathbf{H} \odot \mathbf{D})\mathbf{1} + \mathbf{M}^{-1}(\mathbf{H} \odot \mathbf{Z})\mathbf{1} = \mathbf{q} + \mathbf{M}^{-1}(\mathbf{H} \odot \mathbf{Z})\mathbf{1}$$

3.1.1 Scelta ed ottimizzazione dei pesi

Per quanto riguarda i pesi $\omega_{i,j}$ possiamo considerare due possibili scelte. La prima consiste nel porre $\omega_{i,j} = 1, \forall (i,j) \in \xi$ e chiamiamo l'algoritmo corrispondente LS non pesato. La seconda scelta permette di modellare il fatto che le stime $\hat{d}_{i,j}$ non hanno tutte la stessa affidabilità e l'algoritmo che si costruisce in questi casi si chiama LS pesato (o WLS). Abbiamo infatti visto valere che:

$$\hat{d}_{i,j} = F^{-1}(\hat{p}_{i,j})$$

e dunque vale che, sviluppando al primo ordine tale equazione:

$$z_{i,j} = \hat{d}_{i,j} - d_{i,j} = F^{-1}(\hat{p}_{i,j}) - d_{i,j} = F^{-1}(p_{i,j}) + \left.\frac{dF^{-1}(p)}{dp}\right|_{p=p_{i,j}} y_{i,j} + \\ + O(y_{i,j}^2) - d_{i,j} = \left.\frac{dF^{-1}(p)}{dp}\right|_{p=p_{i,j}} y_{i,j} + O(y_{i,j}^2)$$

Dunque, se assumiamo questa approssimazione al primo ordine, $z_{i,j}$ è una variabile aleatoria tale che:

- $\mathbf{E}[z_{i,j}] = \left.\frac{dF^{-1}(p)}{dp}\right|_{p=p_{i,j}} \mathbf{E}[y_{i,j}] = 0$;
- $Var(z_{i,j}) = \left(\left.\frac{dF^{-1}(p)}{dp}\right|_{p=p_{i,j}}\right)^2 Var(y_{i,j}) = \left(\left.\frac{dF^{-1}(p)}{dp}\right|_{p=p_{i,j}}\right)^2 \frac{p_{i,j}(1-p_{i,j})}{W}$

Dati allora i valori di q_j, per ogni $j \in \mathcal{N}_i$ il valore ottimale per i pesi per $W \to \infty$ nel sistema lineare (3.1) devono necessariamente essere proporzionali a $\sigma_{i,j}^{-2} = \frac{1}{Var(z_{i,j})}$. Dunque, nell'algoritmo WLS setteremo $\omega_{i,j} = \hat{\sigma}_{i,j}^{-2}$, dove:

$$\hat{\sigma}_{i,j}^2 = \left(\left.\frac{dF^{-1}(p)}{dp}\right|_{p=\tilde{p}_{i,j}}\right)^2 \frac{\tilde{p}_{i,j}(1-\tilde{p}_{i,j})}{W},$$

dove:

$$\tilde{p}_{i,j} = \max\{\min\{\hat{p}_{i,j}, 1-\epsilon\}, \epsilon\}.$$

Qui ϵ è un piccolo parametro tale che $\left.\frac{dF^{-1}(p)}{dp}\right|_{p=\epsilon}$ esiste finito. Osserviamo che, in questo caso, vale che: $0 < \omega_{i,j} < \infty$.

3.2 Analisi asintotica dello stimatore ai minimi quadrati

In questa sezione vediamo alcuni risultati teorici che si ottengono utilizzando lo stimatore LS non pesato. Facciamo questa scelta per semplicità, ma i risultati che si ottengono possono essere estesi anche al caso pesato. Il teorema che andremo a proporre e ad analizzare fornisce una condizione per la convergenza asintotica delle qualità stimate ai valori reali. L'obiettivo sarà allora quello di trovare il minimo numero di confronti sotto i quali l'approccio ai minimi quadrati soddisfi le condizioni (ϵ, δ)-PAC e, per farlo, occorre valutare $\mathbf{P}(\sup_i |\hat{q}_i - q_i| > \epsilon)$, per $\epsilon > 0$. Il teorema che andremo a proporre fornirà una condizione sufficiente per la convergenza a zero dell'errore assoluto e, dunque, per l'esistenza di un algoritmo di ranking (ϵ, δ)-PAC.

3.2.1 Considerazioni sulla struttura del grafo

Al fine di enunciare il teorema al quale siamo interessati è necessario fare alcune considerazioni riguardo la struttura del grafo $\mathcal{G}$. Vogliamo cioè caratterizzare classi di grafi per le quali sarà possibile garantire l'esistenza di un algoritmo di ranking (ϵ, δ)-PAC. Per farlo osserviamo che il sistema (3.4):

$$\hat{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{H} \odot \hat{\mathbf{D}})\mathbf{1}$$

calcola le qualità dell'oggetto i come il valore medio delle stime delle differenze delle qualità lungo tutti i cammini che collegano il nodo i al nodo di riferimento N. Cioè $\hat{q}_i$ può essere visto come la media delle ricompense totali guadagnate durante un cammino casuale standard che parte dal nodo i e si ferma non appena raggiunge il nodo N, dove $\hat{d}_{i,j}$ rappresenta la ricompensa elementare associata all'arco (i, j) del grafo.

Studiamo ora più nel dettaglio il caso in cui $\mathcal{G}$ è indiretto, cioè la tipologia di grafo che viene indotta in modo naturale dal nostro algoritmo. Dal punto di vista di un singolo nodo la soluzione di (3.1):

$$\hat{q}_i = \sum_{j \in \mathcal{N}_i} \omega_{i,j} \frac{\hat{q}_j + \hat{d}_{i,j}}{\rho_i}, i = 1, ..., N$$

per un grafo indiretto può essere ottenuta, come accennato, definendo un problema equivalente per un grafo aciclico che ora andiamo a definire in modo opportuno. Consideriamo un grafo $\mathcal{G} = (\mathcal{V}, \xi)$ con N nodi e sia $\mathbf{T}$ la matrice $(N-1) \times (N-1)$ ottenuta a partire da $\mathbf{H}$ rimuovendo l'ultima riga e l'ultima colonna (cioè quelle corrispondenti al nodo di riferimento N). Consideriamo un dato nodo i, con $i = 1, ..., N$ ed osserviamo come $[(\mathbf{I} - \mathbf{T})^{-1}]_{i,j}$ corrisponda al numero medio di volte che il nodo j è visitato dal cammino casuale che parte in i prima che esso termini nel nodo N. Definiamo:

$$\theta_{j,i} = \begin{cases} \frac{[(\mathbf{I}-\mathbf{T})^{-1}]_{i,j}}{\rho_j}, & j < N \\ 0, & j = N \end{cases}$$

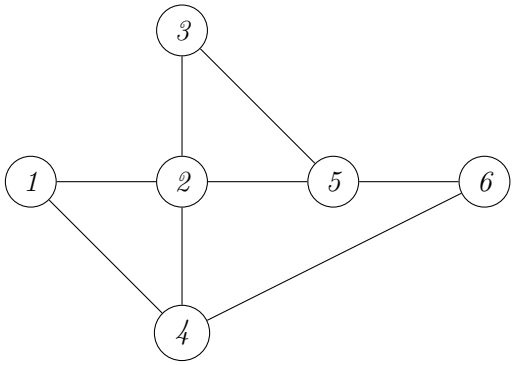
il numero medio di volte che ogni arco adiacente al nodo j è attraversato nella direzione da j a uno dei suoi vicini nel cammino casuale standard definito su $\mathcal{G}$. Possiamo a questo punto definire un grafo diretto aciclico ed un cammino casuale su di esso in modo che sia, dal punto di vista del nodo i, stocasticamente equivalente al cammino casuale standard su $\mathcal{G}$. Sia allora $\vec{\mathcal{G}}_i = (\mathcal{V}, \vec{\xi}_i)$ un grafo diretto e aciclico, dove $(j,l) \in \vec{\xi}_i$ se e solo se $(j,l) \in \xi$ e $\theta_{j,i} > \theta_{l,i}$. Definiamo ancora il cammino casuale sul grafo $\vec{\mathcal{G}}_i$ per il quale, supponendo di essere nel nodo j, la probabilità di dirigerci verso il nodo l scegliendo l'arco uscente (j,l), con $l \in \mathcal{N}_{ji}^+$ è data da:

$$\eta_{j\to l,i} = \frac{\theta_{j,i} - \theta_{l,i}}{\sum_{l' \in \mathcal{N}_{ji}^-} (\theta_{j,i} - \theta_{l',i})},$$

dove $\mathcal{N}_{ji}^+$ è l'insieme dei vicini uscenti di j in $\vec{\mathcal{G}}_i$.

È possibile osservare un'analogia fra il grafo $\vec{\mathcal{G}}_i$ ed il circuito elettrico dove i nodi del grafo rappresentano i nodi del circuito e gli archi rappresentano resistenze di 1 Ω che collegano i nodi del circuito. A questo punto assumiamo di collegare al nodo sorgente i un generatore che inietta una corrente di 1 A nel nodo e di cortocircuitare a terra il nodo di riferimento N. In tale contesto il valore di probabilità $\eta_{j\to l,i}$ associato all'arco (j,l) rappresenta il valore di corrente normalizzato sull'arco. Vale dunque che il valore di corrente sull'arco (j,l) del grafo $\vec{\mathcal{G}}_i$ è dato da $\theta_{j,i} - \theta_{l,i}$, dove $\theta_{j,i}$ rappresenta la tensione al nodo j-esimo nella rete indotta da G_i.

Esempio 3.2.1 *Lo scopo di questo esempio è di mostrare come sia possibile, a partire dal grafo $\mathcal{G} = (\mathcal{V}, \xi)$ seguente:*



ricavare, in un caso molto semplice, il grafo $\vec{\mathcal{G}}_i = (\mathcal{V}, \vec{\xi}_i)$, per $i = 1,2,3,4,5$. Il grafo di partenza, descritto in termini analitici, è dato da:

$$\mathcal{G} = (\mathcal{V}, \xi) = (\{1, ...,6\}, \{(1,2), (1,4), (2,3), (2,4), (2,5), (3,5), (4,6), (5,6)\})$$

dove le coppie (i,j) rappresentano archi indiretti e dunque corrispondono contemporaneamente agli edges (i,j) e (j,i). Abbiamo visto valere:

$$[\boldsymbol{H}]_{i,j} = \begin{cases} [\tilde{\boldsymbol{H}}]_{i,j}, & i < N, \forall j \\ 0, & i = N, \forall j \end{cases}$$

dove:

$$[\tilde{\boldsymbol{H}}]_{i,j} = \begin{cases} \frac{\omega_{i,j}}{\rho_i}, & (i,j) \in \xi \\ 0, & \textit{altrove} \end{cases}$$

Poiché, come detto, stiamo utilizzando l'algoritmo LS non pesato nel quale $\omega_{i,j} = 1, \forall (i,j) \in \xi$ *e siccome i valori dei gradi dei nodi sono dati da:* $\rho_1 = 2$, $\rho_2 = 4$, $\rho_3 = 2$, $\rho_4 = 3$, $\rho_5 = 3$, $\rho_6 = 2$, *è allora immediato ricavare la seguente matrice:*

$$\tilde{\boldsymbol{H}} = \begin{bmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 \\ 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} \\ 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{3} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix}$$

Eliminando l'ultima riga e l'ultima colonna è banale ottenere la matrice $\mathbf{T}$ *e, sottraendo all'identità tale matrice, otteniamo:*

$$\mathbf{I} - \mathbf{T} = \begin{bmatrix} 1 & -\frac{1}{2} & 0 & -\frac{1}{2} & 0 & 0 \\ -\frac{1}{4} & 1 & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & 0 \\ 0 & -\frac{1}{2} & 1 & 0 & -\frac{1}{2} & 0 \\ -\frac{1}{3} & -\frac{1}{3} & 0 & 1 & 0 & -\frac{1}{3} \\ 0 & -\frac{1}{3} & -\frac{1}{3} & 0 & 1 & -\frac{1}{3} \\ 0 & 0 & 0 & -\frac{1}{2} & -\frac{1}{2} & 1 \end{bmatrix}$$

Invertendo, per via numerica, questa matrice si ottiene:

$$(\mathbf{I} - \mathbf{T})^{-1} = \begin{bmatrix} 2.1026 & 2.2564 & 0.8205 & 1.6154 & 0.7692 \\ 1.1282 & 2.8205 & 1.0256 & 1.2692 & 0.9615 \\ 0.7179 & 1.7949 & 1.7436 & 0.8077 & 0.8846 \\ 1.0769 & 1.6923 & 0.6154 & 1.9615 & 0.5769 \\ 0.6154 & 1.5385 & 0.9231 & 0.6923 & 1.6154 \end{bmatrix}$$

Preso ad esempio il valore 2.2564 (che corrisponde alla componente (1,2)) abbiamo visto come esso rappresenti il numero medio di volte che il nodo 2 è visitato dal cammino casuale che parte in 1, prima che esso termini nel nodo N. *A questo punto, dato che abbiamo a disposizione sia la matrice* $(\mathbf{I} - \mathbf{T})^{-1}$ *che i gradi di tutti i nodi del grafo, possiamo facilmente ricavare la seguente matrice:*

$$\theta = \begin{bmatrix} 1.0513 & 0.5641 & 0.3590 & 0.5385 & 0.3077 \\ 0.5641 & 0.7051 & 0.4487 & 0.4231 & 0.3846 \\ 0.4103 & 0.5128 & 0.8718 & 0.3077 & 0.4615 \\ 0.5385 & 0.4231 & 0.2692 & 0.6538 & 0.2308 \\ 0.2564 & 0.3205 & 0.2949 & 0.1923 & 0.5385 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Anche in questo caso abbiamo visto come ciascuna componente del grafo abbia un significato ben preciso: il valore 0.5641, ad esempio, rappresenta il numero medio di volte che ogni arco adiacente al nodo 2 è attraversato nella direzione da 2 ad uno dei suoi vicini nel cammino casuale standard definito su $\mathcal{G}$.

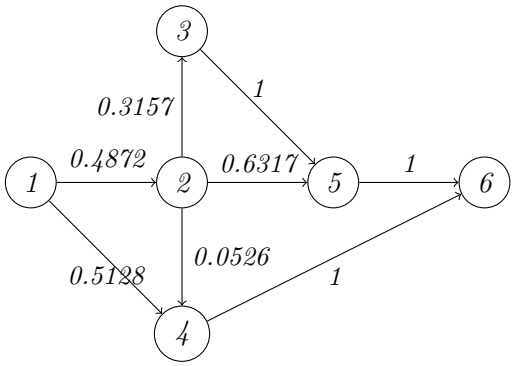
A questo punto è possibile ricavare il grafo diretto ed aciclico $\vec{\mathcal{G}}_i = (\mathcal{V}, \vec{\xi}_i)$*, dove* $(j,l) \in \vec{\xi}_i$ *se e solo se* $(j,l) \in \xi$ *e* $\theta_{j,i} > \theta_{l,i}$*. Abbiamo visto che l'insieme degli archi di partenza è dato da:*

$$\xi = \{(1,2),(1,4),(2,3),(2,4),(2,5),(3,5),(4,6),(5,6)\}$$

Al fine di ricavare $\vec{\mathcal{G}}_1$ *è necessario ricavare* $\vec{\xi}_1$*, cioè l'insieme degli archi diretti appartenenti al grafo originale* $\mathcal{G}$ *e tali che* $\theta_{j,1} > \theta_{l,1}$*. Le coppie dell'insieme che andiamo ora a definire definiscono gli archi diretti che verranno associati ai nodi del grafo di partenza per ottenere* $\vec{\mathcal{G}}_1$*. Per quanto detto, vale allora che:*

$$\vec{\xi}_1 = \{(1,2),(1,4),(2,3),(2,4),(2,5),(3,5),(4,6),(5,6)\}$$

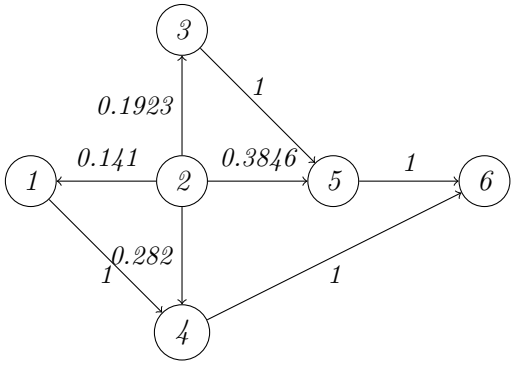
Il grafo diretto ed aciclico $\vec{\mathcal{G}}_1$ *è dunque:*



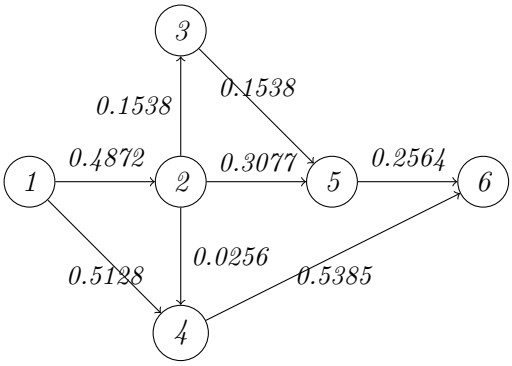
È possibile, in modo analogo, ottenere il grafo $\vec{\mathcal{G}}_2$*, che si ricava associando ai vertici del grafo di partenza il seguente insieme di archi diretti:*

$$\vec{\xi}_2 = \{(1,4),(2,1),(2,3),(2,4),(2,5),(3,5),(4,6),(5,6)\}$$

Il grafo ottenuto è il seguente:



In modo analogo è possibile ricavare anche tutti gli altri grafi: $\vec{\mathcal{G}}_i$ *per* $i = 3,4,5$*.*
Preso ora in esame il grafo $\vec{\mathcal{G}}_1$ *osserviamo come il circuito elettrico associato al grafo presenti i seguenti valori di corrente:*



Osserviamo allora come, ad esempio, solo una piccola quantità di corrente fluisca dal nodo 2 al nodo 4 (in quanto la differenza di potenziale fra i due nodi è piccola) e come questo fatto si rifletta anche, in modo proporzionale, sulla probabilità del cammino che passa per l'arco (2,4) *del grafo* $\vec{\mathcal{G}}_1$.
Per completezza di trattazione, assumiamo di avere a disposizione le quantità $\hat{d}_{1,2}$, $\hat{d}_{1,4}$, $\hat{d}_{2,3}$, $\hat{d}_{2,4}$, $\hat{d}_{2,5}$, $\hat{d}_{3,5}$, $\hat{d}_{4,6}$ *e* $\hat{d}_{5,6}$ *e vediamo come, a partire dai cammini esistenti dal nodo 1 al nodo 6 su* $\vec{\mathcal{G}}_1$, *sia possibile ricavare* $\hat{q}_1$. *Si osserva come, partendo dal nodo 1, sia possibile giungere al nodo di riferimento 6 mediante 4 differenti cammini:*

- *Cammino* $1 \to 2 \to 3 \to 5 \to 6$, *che viene percorso con probabilità 0.1538;*
- *Cammino* $1 \to 2 \to 5 \to 6$, *che viene percorso con probabilità 0.3078;*
- *Cammino* $1 \to 2 \to 4 \to 6$, *che viene percorso con probabilità 0.0256;*
- *Cammino* $1 \to 4 \to 6$, *che viene percorso con probabilità 0.5128.*

Vale allora che, poiché abbiamo imposto $\hat{q}_6 = 0$, *la qualità stimata per l'oggetto 1 sarà data da:*

$$\begin{aligned}\hat{q}_1 =&(\hat{d}_{1,2} + \hat{d}_{2,3} + \hat{d}_{3,5} + \hat{d}_{5,6})0.1538 + (\hat{d}_{1,2} + \hat{d}_{2,5} + \hat{d}_{5,6})0.3078+ \\ &+ (\hat{d}_{1,2} + \hat{d}_{2,4} + \hat{d}_{4,6})0.0256 + (\hat{d}_{1,4} + \hat{d}_{4,6})0.5128\end{aligned}$$

Prima di enunciare il teorema seguente, osserviamo come questo sfrutti la nozione di prossimità dei nodi, proprietà che ora andiamo a definire.

Definizione 3.2.1 *Data una famiglia di grafi* $\{\mathcal{G}_N\}_N$, *diciamo che un nodo* i *è prossimale al nodo di riferimento* N, *con parametri* (τ, h), *se un cammino casuale che ha origine in* i *raggiunge il nodo di riferimento* N *entro* h *salti con una probabilità che asintoticamente, al crescere di* N, *è limitata dal basso da* τ.

A questo punto è possibile enunciare il teorema seguente.

Teorema 3.2.1 *Consideriamo l'algoritmo di stima ai minimi quadrati descritto applicato su un grafo* $\mathcal{G}$ *connesso. Per ogni* $\epsilon > 0$, *al crescere di* N, *vale che:* $\mathbf{P}(\sup_i |\hat{q}_i - q_i| > \epsilon) < \delta$ *se vale che:*

(i) Il numero totale di archi di $\mathcal{G}$ *è* $O(N)$*, e* $W > \beta(\epsilon,\delta)\log N$ *per qualche* $\beta(\epsilon,\delta) = O(\frac{1}{\epsilon^2}\frac{\log(\frac{N}{\delta})}{\log N})$.

e , data una famiglia di grafi connessi $\{\mathcal{G}_N\}_{N\in\mathbf{N}}$ *con diametro limitato per ogni nodo i, con* $i = 1, ..., N-1$*, una delle seguenti condizioni è soddisfatta:*

(i') tutti i cammini su $\vec{\mathcal{G}}_i$ *che hanno origine in i e terminano nel nodo di riferimento N hanno lunghezza limitata;*

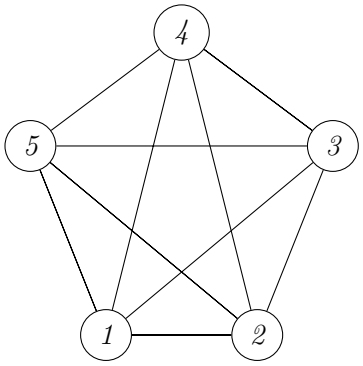
(ii') in $\vec{\mathcal{G}}_i$ *una frazione (strettamente positiva) dei vicini entranti di un nodo qualsiasi è prossimale.*

Di seguito proponiamo lo studio effettuato per tre esempi di famiglie di grafi che soddisfano le condizioni enunciate in questo teorema e che sono dunque adatte per poter stimare le qualità degli oggetti utilizzando l'approccio ai minimi quadrati.

Grafi completi

Consideriamo la famiglia dei grafi completi su N nodi (cioè $\mathcal{G}_N = \mathcal{K}_N$). Data la simmetria del grafo è immediato osservare come, dopo un'adeguata permutazione dei nodi, $(\vec{\mathcal{G}}_N)_i = (\vec{\mathcal{G}}_N)_1, \forall i = 1, ..., N$. Osserviamo inoltre come in $(\vec{\mathcal{G}}_N)_1$, $\theta_{j_1,1} = \theta_{j_2,1}$ per $j_1, j_2 = 2, ..., N-1$. È possibile verificare che gli unici archi (diretti) a sopravvivere in $(\vec{\mathcal{G}}_N)_1$ sono quelli uscenti dal nodo 1 o entranti nel nodo N.

Esempio 3.2.2 *Per chiarire meglio quanto avviene nel caso generale studiamo più nel dettaglio il caso particolare in cui* $N = 5$*, cioè nel caso in cui abbiamo il grafo completo* $\mathcal{G}_5 = \mathcal{K}_5$*:*



e vogliamo ricavare il grafo $(\vec{\mathcal{G}}_5)_1$*. Per come è definita* $\mathbf{H}$*, vale che:*

$$\mathbf{T} = \frac{1}{4}\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

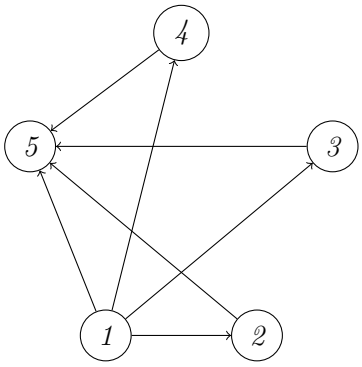
È allora possibile calcolare la matrice:

$$(\mathbf{I}-\mathbf{T})^{-1}=\frac{1}{5}\begin{bmatrix}8&4&4&4\\4&8&4&4\\4&4&8&4\\4&4&4&8\end{bmatrix}$$

dalla quale è immediato ricavare la matrice:

$$\theta=\frac{1}{5}\begin{bmatrix}2&1&1&1\\1&2&1&1\\1&1&2&1\\1&1&1&2\\0&0&0&0\end{bmatrix}$$

Mediante questa matrice è allora possibile ricavare il grafo $(\vec{\mathcal{G}}_5)_1$*:*



Poiché il cammino di lunghezza massima dal nodo 1 al nodo N in $(\vec{\mathcal{G}}_N)_1$ è 2, questa famiglia di grafi rispetta la condizione (i') del teorema 3.2.1 e la stima di q_i è data da:

$$\hat{q}_i=\frac{2}{N}\hat{d}_{i,N}+\frac{1}{N}\sum_{j=1,j\neq i}^{N-1}(\hat{d}_{i,j}+\hat{d}_{j,N})$$

Grafi a poli

Dati N' e Δ due numeri positivi qualsiasi, costruiamo ora una nuova famiglia di grafi $\{\mathcal{G}_N\}_{N\geq N'}$, del tipo mostrato in figura 3.1..

In tale famiglia di grafi i nodi $N-N'+1,...,N$ sono poli ed hanno grado potenzialmente illimitato. I nodi polo sono definiti in questo modo in quanto ad ognuno di essi è associato un sottografo, che non possiede particolari caratteristiche, ma che deve essere connesso. I nodi rimanenti (che non sono poli) hanno grado massimo Δ e sono divisi in N' sottoinsiemi $\mathcal{S}_1,...,\mathcal{S}_{N'}$. Tutti i nodi nel sottoinsieme $\mathcal{S}_j$, per $j=1,...,N'$, sono vicini del nodo polo $N-j+1$ e tutti gli altri suoi vicini appartengono a $\mathcal{S}_j$.

È allora immediato verificare come il diametro di questa famiglia di grafi sia limitato e, in particolare, corrisponde a $N'-1$. Consideriamo ora un nodo $i\in\mathcal{S}_j$: poiché ogni

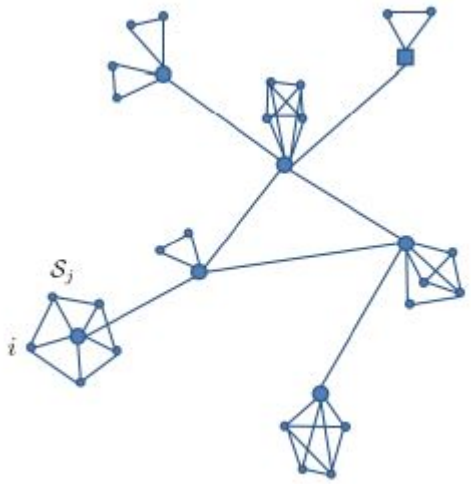


Figura 3.1. Esempio di grafo a poli con N=32 nodi, N'=7 poli e grado massimo dei nodi che non sono poli Δ=4

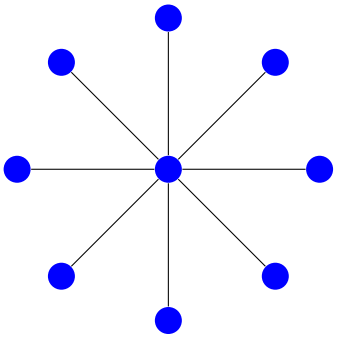
cammino che raggiunge il nodo di riferimento N deve passare attraverso i nodi polo è facile osservare come, in $(\vec{G}_N)_i$, il nodo i è connesso solo a $\mathcal{S}_j \cup \{N - N' + 1, ..., N\}$. In ogni caso il cammino minimo con bias su $(\vec{\mathcal{G}}_N)_i$ lascia $\mathcal{S}_j$ (raggiungendo il nodo polo $N - j + 1$) e non entra più in esso. È allora possibile dividere in due parti il cammino random con bias: la prima parte sul sottografo su $(\vec{\mathcal{G}}_N)_i$ indotto da $\mathcal{S}_j$, dove il nodo polo $N - j + 1$ funge da nodo di riferimento ed il secondo sul nodo di riferimento. È dunque possibile dedurre i seguenti fatti:

- Nella prima parte del cammino casuale, poiché il nodo centrale $N - j + 1$ è il nodo di riferimento, la probabilità di raggiungerlo in un passo da un altro nodo qualsiasi di $\mathcal{S}_j$ è maggiore di $\frac{1}{\Delta}$. Allora la probabilità di raggiungerlo in D' passi è limitata dall'alto da $\tau = 1 - (1 - \frac{1}{\Delta})^{D'}$.
- La seconda parte del cammino casuale termina al massimo in $N' - 1$ passi

Allora, ogni nodo è prossimale con parametri $(\tau, D' + N' - 1)$ e la condizione (ii') del teorema 3.2.1 è soddisfatta.

Grafi a stella

Come ultimo esempio consideriamo i grafi a stella ed osserviamo come questi rappresentino un caso particolare dei grafi descritti nell'esempio precedente. Un esempio di grafo di questo tipo, dove abbiamo un nodo centrale e 8 nodi ad esso collegato, è quello riportato nella figura seguente:



Su di essi è chiaramente soddisfatta la condizione (i') del teorema 3.2.1. In tale contesto il nodo polo, ovvero il centro della stella, viene identificato con il nodo N e le qualità di tutti gli altri oggetti sono stimate attraverso il confronto diretto con tale centro. A causa della particolare struttura del grafo osserviamo come la classifica sugli oggetti possa essere ricavata in modo diretto sulla base dei $\hat{p}_{i,N}$, senza che ci sia la necessità di invertire la funzione $F(\cdot)$, in quanto gli oggetti possono essere ordinati seguendo le seguenti regole:

- $\hat{r}(i) \prec \hat{r}(j) \iff \hat{p}_{i,N} > \hat{p}_{j,N}$;
- $\hat{r}(i) \prec \hat{r}(N) \iff \hat{p}_{i,N} > \frac{1}{2}$.

Di conseguenza la scelta di utilizzare un grafo a stella risulta essere molto utile in contesti in cui non è noto al sistema il preciso modello di comportamento dei lavoratori.

Capitolo 4

Ranking di un numero finito di oggetti: approccio a singolo stadio

In questo capitolo ci poniamo l'obiettivo di studiare nel dettaglio il problema nel quale ci viene assegnato un numero finito di oggetti sui quali vogliamo produrre una classifica ed abbiamo a disposizione un determinato numero di valutazioni, che possono essere distribuite sulle coppie di oggetti. Occorre dunque scegliere, in tale contesto, le coppie di oggetti che vogliamo mettere a confronto e come distribuire le valutazioni su tali coppie al fine di ottenere una classifica che sia la più accurata possibile. Se pensiamo agli oggetti assegnati come a nodi di un grafo, come detto, operare una scelta di questo tipo significa disegnare gli archi che rappresentano i confronti fra le coppie di oggetti e, dunque, di fatto, disegnare il grafo $\mathcal{G} = (\mathcal{V}, \xi)$. In particolare, la prima parte di questo capitolo sarà dedicato allo studio, mediante simulazione, delle prestazioni dell'algoritmo descritto nel capitolo 3, sia nella sua versione non pesata che in quella pesata, su un numero fissato di oggetti. Gli algoritmi verranno studiati al variare del numero di valutazioni che si hanno a disposizione e della scelta della struttura dei confronti fra le coppie di oggetti. Ci riferiamo a tale metodo come approccio a singolo stadio.

Nel capito successivo verranno invece discussi i risultati che si ottengono utilizzando un approccio a due stadi. Tale metodo consiste, come vedremo nel dettaglio in seguito, nel suddividere il pool totale di valutazioni a disposizione in due insiemi: uno che viene assegnato al primo stadio ed uno al secondo. L'idea di tale approccio è di ricavare un ordinamento, seppure meno accurato, dal primo stadio e contemporaneamente nel memorizzare le valutazioni che sono state riservate ed espresse in questa fase. Poi, al secondo stadio, viene sfruttato l'ordinamento ottenuto al passo precedente al fine di concentrare le valutazioni sulle coppie di oggetti che ricoprono posizioni vicine fra loro nella classifica prodotta al primo stadio. Tali coppie di oggetti, infatti, avranno verosimilmente qualità abbastanza simili fra loro e dunque è plausibile che al primo stadio possano essere stati scambiati di posto. A questo punto, utilizzando contemporaneamente le valutazioni espresse ai due stadi, viene prodotto un ranking che, come vedremo, è più accurato, a

parità di numero totale di valutazioni, rispetto al caso in cui tutte le osservazioni sono assegnate ad un singolo stadio.

Modello scelto per i lavoratori

In questo capitolo, così come nel prossimo, nelle simulazioni abbiamo deciso di considerare i lavoratori indistinguibili e, dunque, tali che hanno tutti lo stesso comportamento. Si è scelto, in particolare, di utilizzare, per descrivere e simulare il loro comportamento, il modello di Thurstone (Thurstone [1927]), dove la scelta sulla preferenza fra i due oggetti viene effettuata sulla base delle qualità così come vengono percepite da parte dei lavoratori. Ricordiamo infatti come le qualità intrinseche reali non sono note al sistema e, in particolare, sono sconosciute anche ai lavoratori. Le qualità percepite dai lavoratori sono dunque definite come:

$$\tilde{q_i} = q_i + n_i, \qquad \tilde{q_j} = q_j + n_j$$

dove n_i ed n_j sono variabili aleatorie gaussiane a media nulla e rappresentano il rumore nella stima delle qualità che avvertono i lavoratori. Nelle simulazioni effettuate abbiamo scelto come valore di deviazione standard $\sigma = 0.4$.

Misurazione delle prestazioni degli algoritmi scelti

Al fine di misurare le prestazioni degli algoritmi che andremo a studiare in questo capitolo e nel successivo, non essendo percorribile altra via, procederemo mediante simulazione ed utilizzeremo due misure di accuratezza: la probabilità di errore (P_e) e la distorsione media (D), che ora andiamo a discutere. Al fine di ricavare tali quantità viene fissato il numero massimo di errori che decidiamo di tollerare ($E_{\max}$) ed un numero massimo di istanze ($R_{\max}$), che fissa di fatto un limite superiore per il numero di volte che l'algoritmo viene simulato. A questo punto, dopo che vengono campionate in modo casuale le qualità degli oggetti, viene applicato l'algoritmo e, ad ogni iterazione, viene prodotta una classifica. Questo procedimento viene iterato fino a quando non viene raggiunto il numero massimo di errori ammessi $E_{\max}$ o l'algoritmo non viene iterato $R_{\max}$ volte. Viene conteggiato un errore ogni volta che l'algoritmo produce una classifica che, nel contesto (ϵ, δ)-PAC, viene considerata errata. In particolare, ogni volta che viene prodotta una classifica, viene calcolato il vettore $\mathbf{\Delta}$ di dimensione N tale che la componente i-esima rappresenta la differenza, in valore assoluto, fra la qualità reale dell'oggetto che è in posizione i e la qualità reale dell'oggetto che viene stimato essere in posizione i. A questo punto viene calcolato il massimo su questo vettore e, se tale quantità supera la tolleranza ϵ, allora il conteggio degli errori viene incrementato di un'unità. Ad ogni iterazione viene inoltre calcolata la distorsione della classifica che viene prodotta che, se consideriamo la k-esima iterazione, è data da:

$$D_k = \frac{\sum_{i=1}^{N} \mathbf{\Delta}_i}{N}$$

Esistono, come detto, due regole di stop a causa delle quali il ciclo viene interrotto: una viene attivata se si sono raggiunte $R_{\max}$ iterazioni e l'altra se il conteggio degli errori raggiunge $E_{\max}$. A questo punto, supponendo che l'algoritmo sia stato simulato s volte

prima che una delle due condizioni di stop si attivi, vengono calcolate le due seguenti misure di prestazione:

- La probabilità di errore, che esprime la probabilità di produrre una classifica errata nel contesto (ϵ, δ)-PAC (dove vengono cioè tollerati scambi di posizione fra oggetti che hanno qualità non molto differenti fra loro):

$$P_e = \frac{\#errori}{s};$$

- La distorsione media:

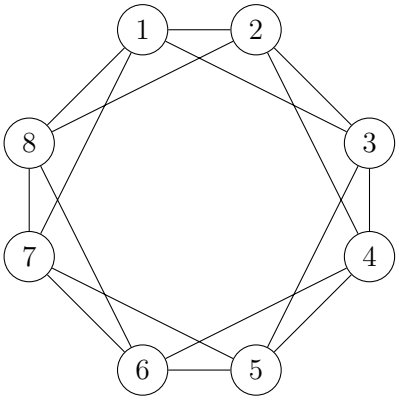
$$D = \frac{\sum_{i=1}^{s} D_i}{s}.$$

4.1 Studio degli algoritmi utilizzando un grafo a catena chiusa

In questa sezione presenteremo lo studio, mediante simulazione, delle prestazioni, in termini di P_e e di D, dell'algoritmo descritto nel capitolo 3 sia nella sua versione pesata (WLS) che in quella non pesata (LS) quando scegliamo di utilizzare il grafo a catena chiusa. Descriviamo le caratteristiche di tale grafo nella sottosezione seguente.

Struttura del grafo

Una delle decisioni che si sono rese necessarie, fissati gli oggetti sui quali occorre produrre una classifica, è stata quella relativa alla scelta delle coppie di oggetti da confrontare, cioè, di fatto, alla struttura del grafo che andiamo a considerare. Si è allora deciso, per la versione a singolo stadio dell'algoritmo, di costruire ed utilizzare, in prima battuta, un grafo indiretto regolare a catena chiusa con grado pari. Al fine di costruire un grafo di questo tipo, con N nodi e di grado d, è sufficiente, per ogni nodo i del grafo, generare un arco indiretto che lo metta in relazione con i nodi $(i+1) \mod N$, $(i+2) \mod N$,..., $\left(i+\frac{d}{2}\right) \mod N$. Dati, ad esempio, $N = 8$ oggetti e grado $d = 4$ il grafo descritto sarà:



La scelta di utilizzare un grafo regolare è motivata dal fatto che, a priori, non esiste alcun motivo per privilegiare un nodo rispetto ad un altro e dunque assegnare a tutti i nodi del grafo lo stesso grado sembra essere la decisione più sensata. Osserviamo, inoltre, come, posto che ciascun nodo del grafo regolare abbia quanto meno grado 2, siamo certi di ottenere un grafo connesso, condizione che abbiamo visto essere necessaria al fine di ottenere classifiche complete sugli oggetti assegnati. Nonostante la struttura del grafo descritto sia deterministica, osserviamo inoltre come, poiché gli oggetti ci vengono forniti dopo che sono stati permutati in modo casule, i confronti fra le coppie di oggetti indotti da tale struttura del grafo sono casuali.

4.1.1 Studio dell'algoritmo LS

In questa sezione presentiamo lo studio delle prestazioni dell'algoritmo nella sua versione non pesata (LS) nel caso in cui ci vengono forniti 20 oggetti sui quali si vuole costruire una classifica. In particolare studiamo le prestazioni dell'algoritmo a singolo stadio dove la scelta della struttura del grafo è ricaduta su quello regolare a catena chiusa descritto in precedenza. In tale contesto abbiamo applicato l'algoritmo facendo variare prima il grado di ogni nodo del grafo regolare scelto e poi il numero totale di valutazioni che abbiamo a disposizione. Studieremo, inoltre, le prestazioni dell'algoritmo al variare della tolleranza che assumiamo di avere nel giudizio della correttezza e della bontà delle classifiche prodotte. Nel seguito della sezione presenteremo e discuteremo infatti il comportamento della probabilità di errore e della distorsione media per l'algoritmo LS che si ottengono operando due particolari scelte di questo parametro.

Caso ϵ=0.04

In questo caso, nel quale fissiamo il parametro di tolleranza dell'errore ϵ a 0.04, scegliamo di considerare errate classifiche dove esiste almeno una coppia di oggetti le cui qualità reali differiscono fra loro per un valore superiore o uguale a 0.04, che viene scambiata di ordine. In caso contrario la classifica viene considerata corretta. Osserviamo come, essendo le qualità distribuite in [0,1] ed avendo noi supposto di avere N=20 oggetti, nel caso ideale in cui le qualità degli oggetti fossero equispaziate nell'intervallo [0,1], avremmo che le qualità di due oggetti che sono in posizioni fra loro consecutive nella classifica reale differirebbero in valore assoluto per un valore superiore a 0.05. Dunque, supponendo di essere in un caso ideale di questo tipo, ogni classifica che scambia di posizione una qualsiasi coppia di oggetti viene considerata errata e solo le classifiche realmente ed interamente esatte si considerano corrette. Con una scelta di questo tipo, anche nel caso più generale dove le qualità sono campionate in modo casuale da un'uniforme in [0,1], siamo sicuramente molto severi nel considerare corrette o errate le classifiche che vengono prodotte.

Le figure 4.1. e 4.2. presentano il comportamento della probabilità di errore al variare di alcuni parametri. In particolare nella figura 4.1. viene mostrato l'andamento della P_e al variare del grado del grafo regolare e fissato il numero di valutazioni totali. La scelta operata è stata quella di testare l'algoritmo con tutti i grafi a catena chiusi regolari di grado pari che è possibile costruire con 20 oggetti. Dunque il grado assumerà i valori 2, 4,..., 18. Nella figura 4.2., invece, viene mostrato il comportamento della P_e al variare del

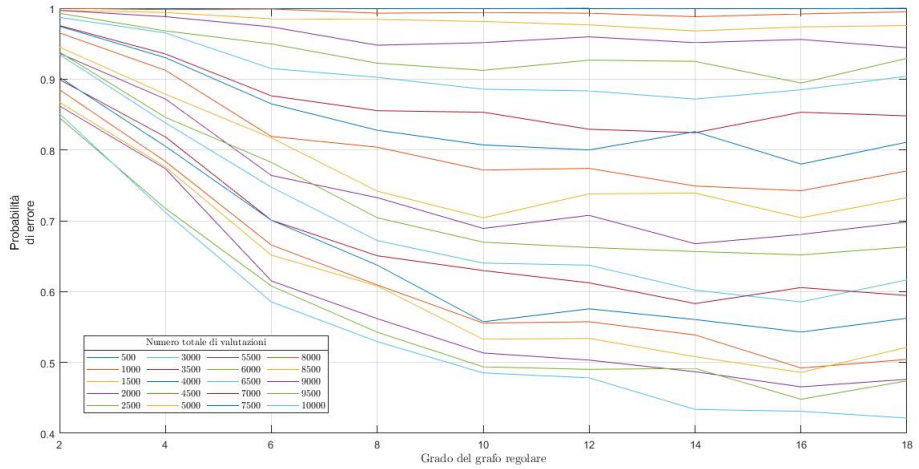


Figura 4.1. P_e al variare del grado del grafo regolare e fissato il numero di valutazioni totali nel caso $N = 20$ ed $\epsilon = 0.04$

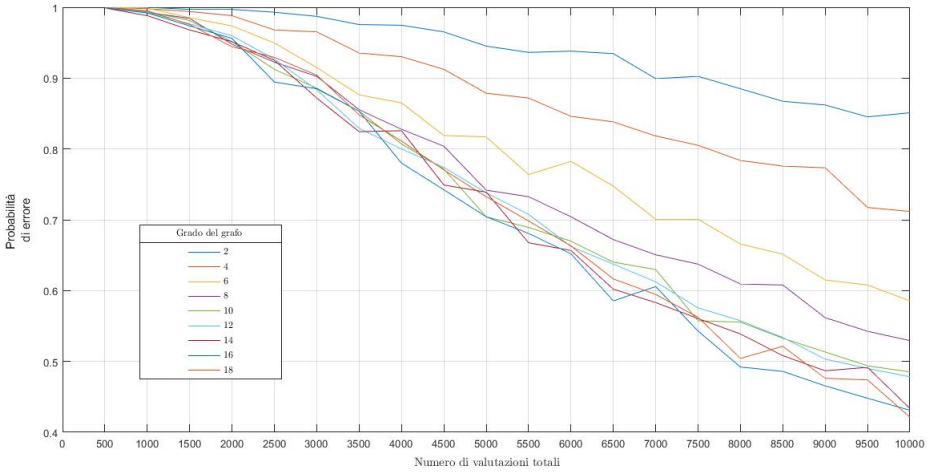


Figura 4.2. P_e al variare del numero totale di valutazioni e fissato il grado del grafo regolare nel caso $N = 20$ ed $\epsilon = 0.04$

numero di valutazioni totali e fissato il grado del grafo regolare. Abbiamo deciso di settare il numero minimo di valutazioni con le quali testare l'algoritmo a 500 e poi di aumentare progressivamente tale quantità di 500 alla volta fino a quando non vengono raggiunte le 10000 valutazioni. Come detto, abbiamo assunto di dover produrre una classifica su 20 oggetti ($N = 20$) e, per farlo, abbiamo implementato l'algoritmo nella sua versione non pesata (LS) con $\epsilon = 0.04$. In particolare, abbiamo poi impostato il numero massimo

di errori tollerati $E_{\max}$ a 1000 e l'eventuale numero massimo di iterazioni totali $R_{\max}$ a 1000000. Le qualità reali degli oggetti, sulla base delle quali viene prodotta la classifica corretta, sono campionate in modo casuale in [0,1].

Nella figura 4.1., dunque, ogni curva del grafico mostra le prestazioni, in termini di P_e, al variare del grado del grafo e fissato il numero di valutazioni totali da assegnare alle coppie di oggetti che decidiamo di mettere a confronto. È interessante osservare come, indipendentemente dal numero totale di valutazioni utilizzate, le prestazioni dell'algoritmo migliorano sensibilmente al crescere del grado del grafo fino a quando non viene raggiunto il grado 8-10. A partire da tale valore si osserva infatti una saturazione del miglioramento delle prestazioni. Tale comportamento può essere giustificato dal fatto che, fino a quando il grado del grafo in questione è basso, l'algoritmo trova giovamento nel distribuire maggiormente le valutazioni a disposizione su un numero maggiore di archi rispetto che a concentrarle su un numero ristretto di essi. Quando però viene raggiunta la soglia di 8-10 per il grado del grafo, cioè da quando esso risulta essere sufficientemente connesso, si osserva come, sempre indipendentemente dal numero totale di valutazioni, non si tragga più giovamento nel distribuire le valutazioni su un numero maggiore di archi. Questo perché, verosimilmente, avremo troppe poche valutazioni per ogni arco e questo porta chiaramente a stimare qualità per gli oggetti che non sono sufficientemente affidabili. I risultati ottenuti mostrano infatti come associare all'algoritmo un grafo con grado 10 o con grado 18 sia sostanzialmente indifferente al fine di ottenere un miglioramento in termini di P_e. Osserviamo inoltre come, al crescere del numero di valutazioni totali utilizzate, si riscontri un generale miglioramento delle prestazioni.

Osservando la figura 4.2. è possibile confermare le considerazioni fatte sul grafico precedente. Qui ogni curva rappresenta la P_e al variare del numero totale di valutazioni utilizzate e fissato il grado del grafo regolare. In questo grafico risulta ancora più evidente come l'algoritmo migliori significativamente al crescere del grado del grafo fino a quando esso ha grado 8-10, in quanto le curve corrispondenti sono effettivamente distanziate fra loro. Le curve relative all'algoritmo applicato su grafi di grado maggiore di tale soglia, invece, hanno un comportamento molto simile e questo è indice del fatto che presentano, a parità di valutazioni utilizzate, prestazioni simili fra loro. Osserviamo, inoltre, come il comportamento generale di tutte le curve sia decrescente e come questo confermi quanto evidenziato a partire dal grafico precedente, ovvero che le prestazioni dell'algoritmo migliorano sempre all'aumentare del numero di valutazioni a disposizione.

Considerazioni simili possono essere fatte osservando le figure 4.3. e 4.4., le quali riportano l'andamento della distorsione media rispettivamente al variare del grado del grafo regolare e fissato il numero di valutazioni totali ed al variare del numero totale di valutazioni e fissato il grado del grafo regolare. L'esperimento dal quale abbiamo ottenuto questi grafici è stato effettuato mantenendo costanti tutti i parametri utilizzati nell'esperimento che ci ha permesso di ricavare i grafici riportati nelle figure 4.1. e 4.2.. I grafici in questione accentuano le caratteristiche e le osservazioni che sono emerse studiando il comportamento della probabilità di errore. Il comportamento della distorsione media mostrato nella figura 4.3. mostra infatti un miglioramento significativo delle prestazioni al crescere del grado del grafo solo fino a quando questo rimane basso (inferiore a 6-8) e successivamente, per il fenomeno descritto in precedenza, non si ha più un miglioramento significativo in

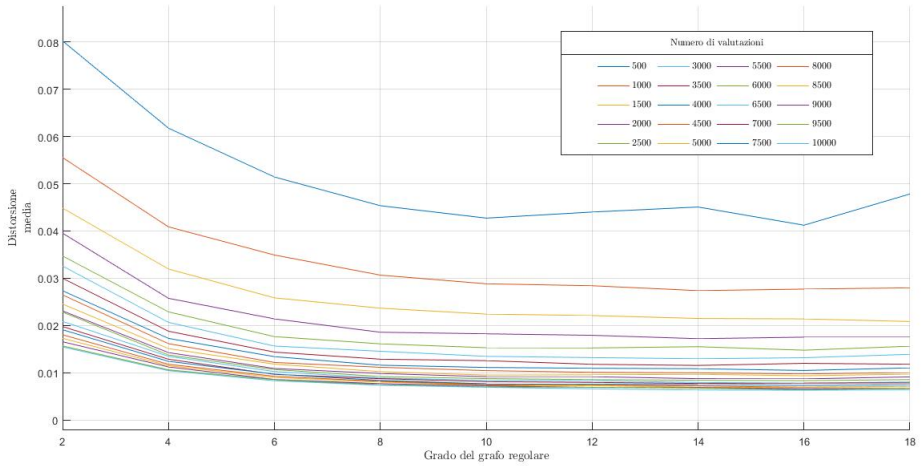


Figura 4.3. D al variare del grado del grafo regolare e fissato il numero di valutazioni totali nel caso $N = 20$ ed $\epsilon = 0.04$

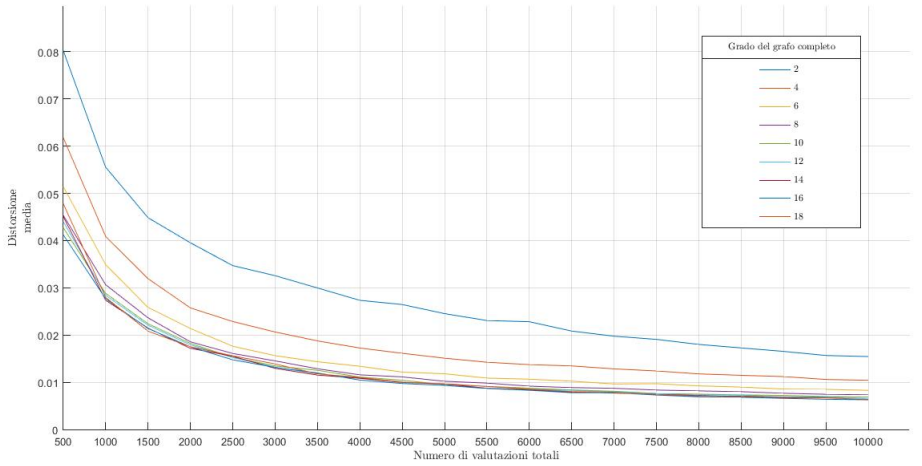


Figura 4.4. D al variare del numero totale di valutazioni e fissato il grado del grafo regolare nel caso $N = 20$ ed $\epsilon = 0.04$

termini di D. Anche nella figura 4.4. è possibile notare tale fenomeno: tutte le curve relative alle prestazioni dell'algoritmo applicato utilizzando un grafo di grado superiore a 6-8 sono infatti sovrapposte fra loro ed hanno un comportamento del tutto simile in termini di distorsione media. Si riscontra, inoltre, un generale miglioramento delle prestazioni dell'algoritmo al crescere del numero di valutazioni utilizzate per produrre la classifica. Da tale grafico si evince come tale miglioramento sia però decisamente più netto quando

il numero di valutazioni a disposizione è basso e come sia presente un fenomeno di saturazione che suggerirebbe, al fine di ridurre il valore della distorsione media, che non è troppo utile aggiungere ulteriori valutazioni quando si raggiunge un numero pari a 5000-6000 di esse. Occorre, infatti, ricordare come, al crescere del numero di valutazioni coinvolte, la complessità dell'algoritmo cresca e come, una volta raggiunte le 5000-6000 valutazioni, il valore ottenuto di D sembri suggerire che aggiungere ulteriori valutazioni accrescerebbe solamente il costo computazionale (e, in caso di applicazione reale, di raccolta delle valutazioni) senza che si abbia un effettivo miglioramento in termini di prestazioni.

Ricordiamo, inoltre, come il parametro ϵ di tolleranza dell'errore fosse settato in questa sezione a 0.04 e come, di conseguenza, fossimo qui molto severi nella valutazione della correttezza delle classifiche prodotte. Nella sezione successiva ripeteremo l'analisi effettuata con un parametro di tolleranza che ci permetterà di essere più morbidi e permissivi in questo senso.

Caso ϵ=0.08

In questa sezione, come anticipato, riproponiamo l'analisi effettuata nella sezione precedente fissando il parametro di tolleranza dell'errore a 0.08. Consideriamo allora corrette classifiche che eventualmente scambiano di posizione coppie di oggetti con qualità che differiscono fra loro, in valore assoluto, per valori inferiori di 0.08. Nel caso ideale in cui le qualità degli oggetti fossero distribuite in modo uniforme in [0,1], permetteremo dunque al più scambi di posizione fra oggetti che, nella classifica corretta, occupano posizioni fra loro consecutive. Siamo dunque meno rigidi nel considerare errate le classifiche che produciamo.

Nessun altro parametro è stato modificato, rispetto alla sezione precedente, al fine di produrre le figure 4.5. e 4.6..

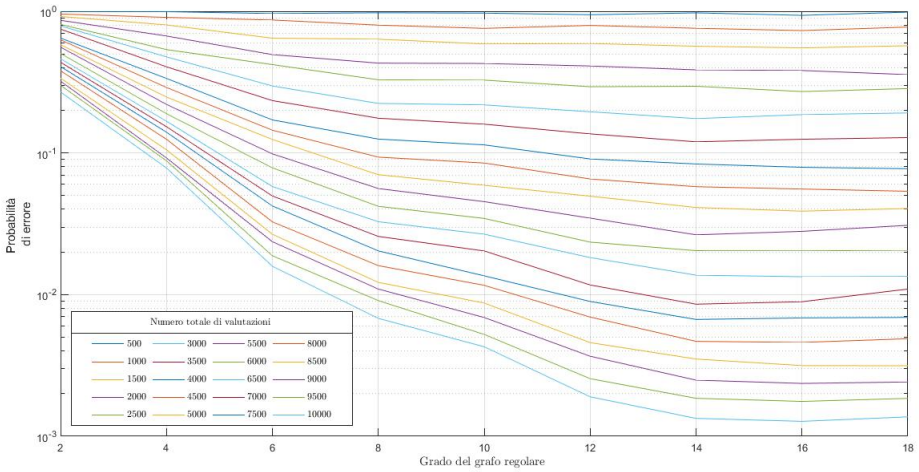


Figura 4.5. P_e al variare del grado del grafo regolare e fissato il numero di valutazioni totali nel caso $N = 20$ ed $\epsilon = 0.08$

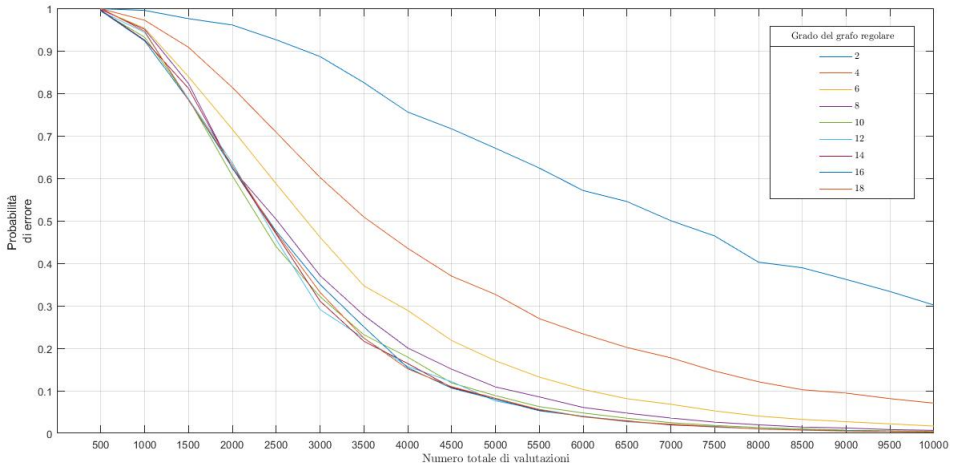


Figura 4.6. P_e al variare del numero totale di valutazioni e fissato il grado del grafo regolare nel caso $N = 20$ ed $\epsilon = 0.08$

Analogamente a quanto presentato nella sezione precedente, la figura 4.5. mostra il comportamento della probabilità di errore al variare del grado del grafo regolare e fissato il numero di valutazioni totali, mentre la figura 4.6. presenta la probabilità di errore al variare del numero totale di valutazioni e fissato il grado del grafo regolare. Gli intervalli scelti per selezionare i parametri relativi al grado del grafo ed al numero di valutazioni totali sono i medesimi scelti nella sezione precedente. Dalle due figure osserviamo come, quando viene utilizzato un numero sufficientemente alto di valutazioni ed un grafo sufficientemente connesso (cioè dove ogni nodo ha grado maggiore o uguale a 8-10), si ottengono prestazioni, in termini di P_e, che, tollerando errori del tipo descritto in precedenza, sembrano essere piuttosto soddisfacenti. Dalla figura 4.6., in particolare, è possibile osservare i due fenomeni che sono già stati evidenziati nella sezione precedente. Fintanto che viene sfruttato un numero basso di valutazioni, si osserva infatti che, una volta che viene raggiunto il grado 8-10, non si trae un beneficio nell'utilizzare grafi che abbaino un grado superiore. Una volta superato tale valore di soglia, a causa del fenomeno che è stato descritto nella sezione precedente, non è più conveniente aumentare il grado del grafo che scegliamo di associare all'algoritmo. Osserviamo inoltre come, anche in questo caso, sia possibile riscontrare un beneficio imputabile all'impiego di un numero via via crescente di valutazioni, ma anche come tale miglioramento sia significativamente più netto quando le valutazioni sfruttate sono relativamente poche (ad esempio inferiori a 5000). Osserviamo, infatti, un fenomeno di saturazione della P_e per valori eccessivamente alti del numero di valutazioni utilizzate.

4.1.2 Studio dell'algoritmo WLS

In questa sezione riproporremo lo studio presentato nella sezione precedente, ma scegliendo di implementare l'algoritmo nella sua versione pesata (WLS) nel caso in cui ci vengono

forniti 20 oggetti sui quali si vuole costruire una classifica. Anche in questo caso studieremo le prestazioni dell'algoritmo in termini di probabilità di errore e di distorsione media per le due scelte operate in precedenza per il parametro ϵ di tolleranza dell'errore.

Caso ϵ=0.04

In questa sezione valuteremo le prestazioni dell'algoritmo WLS nel caso in cui, come detto, siamo molto severi nel considerare errate le classifiche prodotte.

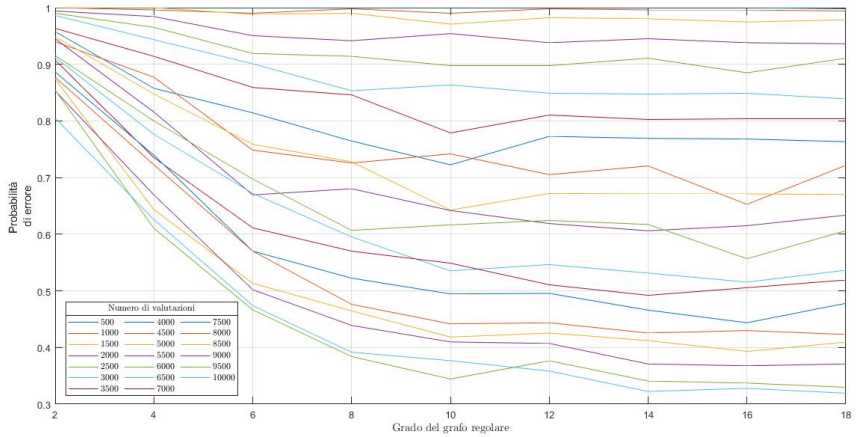


Figura 4.7. P_e al variare del grado del grafo regolare e fissato il numero di valutazioni totali nel caso $N = 20$ ed $\epsilon = 0.04$

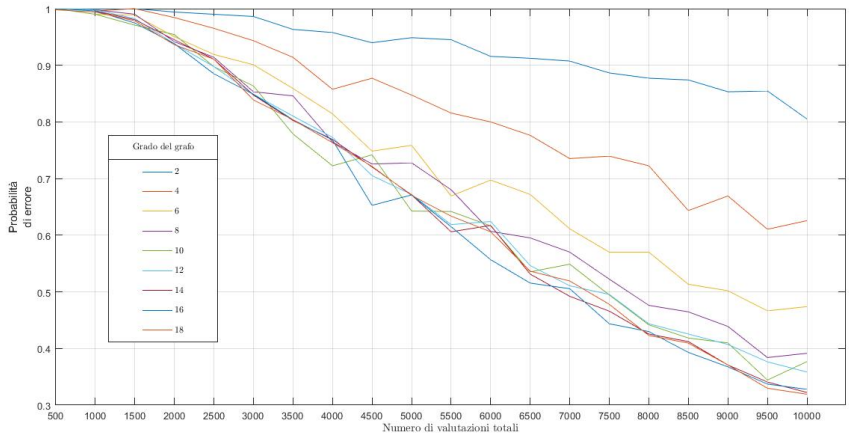


Figura 4.8. P_e al variare del numero totale di valutazioni e fissato il grado del grafo regolare nel caso $N = 20$ ed $\epsilon = 0.04$

Dai grafici 4.7. e 4.8. si evincono le stesse considerazioni fatte per l'algoritmo nella versione non pesata. Tali grafici riportano il comportamento della probabilità di errore rispettivamente in funzione del grado del grafo regolare (fissato il numero totale di valutazioni) ed in funzione del numero totale di valutazioni (fissato il grado del grafo). Dunque, anche per la versione pesata dell'algoritmo, vale quanto sottolineato per l'algoritmo nella sua versione non pesata. Va osservato, però, come l'aggiunta dei pesi, discussi nell'apposita sezione del capitolo 3, comporti un significativo e generale miglioramento delle prestazioni dell'algoritmo in termini di probabilità di errore.

Le stesse considerazioni possono essere fatte osservando le figure 4.9. e 4.10., le quali sono prodotte ripetendo il medesimo esperimento effettuato per ricavale le figure 4.7. e 4.8., ma che, al posto della P_e, riportano il comportamento della D.

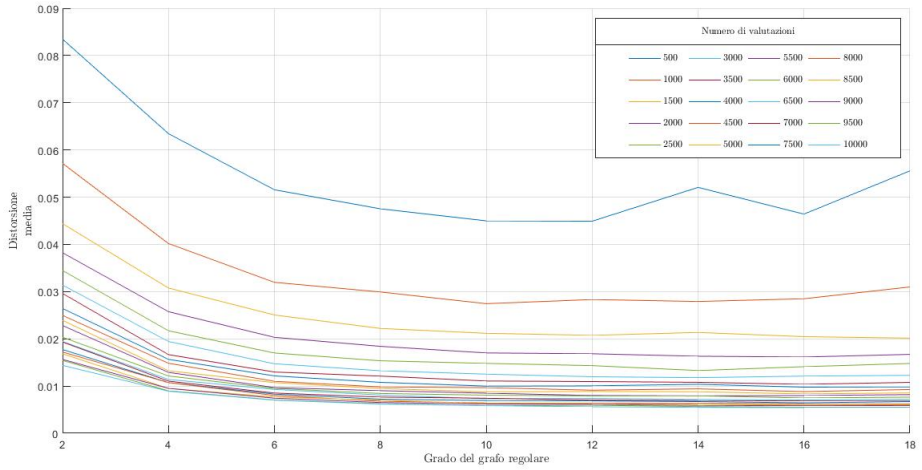


Figura 4.9. D al variare del grado del grafo regolare e fissato il numero di valutazioni totali nel caso $N = 20$ ed $\epsilon = 0.04$

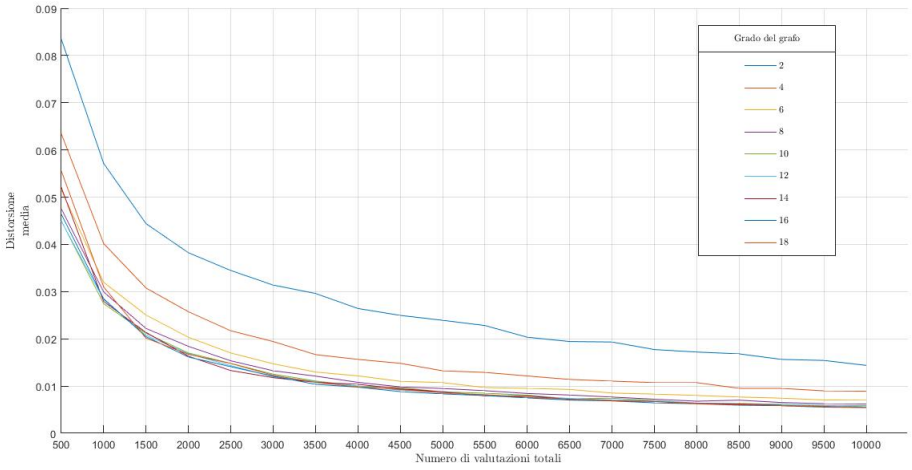


Figura 4.10. D al variare del numero totale di valutazioni e fissato il grado del grafo regolare nel caso $N = 20$ ed $\epsilon = 0.04$

Caso ϵ=0.08

In questa sezione riportiamo, infine, il comportamento della probabilità di errore nel caso in cui siamo più tolleranti nel considerare errate le classifiche che generiamo, cioè nel caso in cui fissiamo il parametro ϵ a 0.08. Otteniamo dunque, mediante simulazione, le figure 4.11. e 4.12., dove viene riportata appunto la P_e rispettivamente al variare del grado del grafo completo, fissato il numero di valutazioni totali che vengono utilizzate ed al variare del numero totale di valutazioni, fissato il grado del grafo.

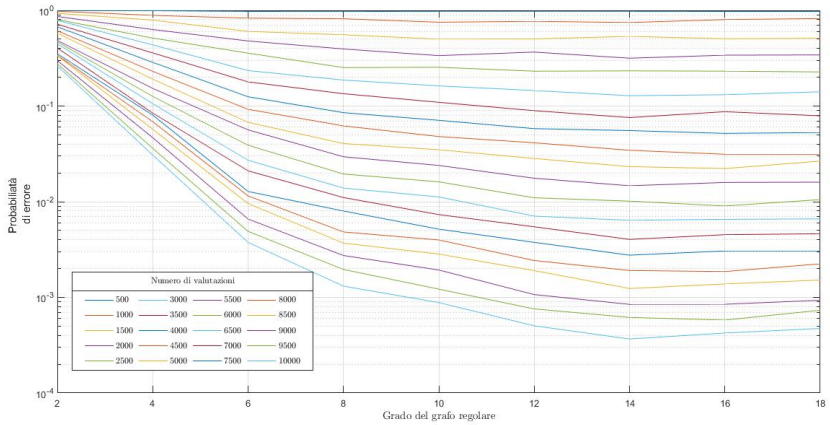


Figura 4.11. P_e al variare del grado del grafo regolare e fissato il numero di valutazioni totali nel caso $N = 20$ ed $\epsilon = 0.08$

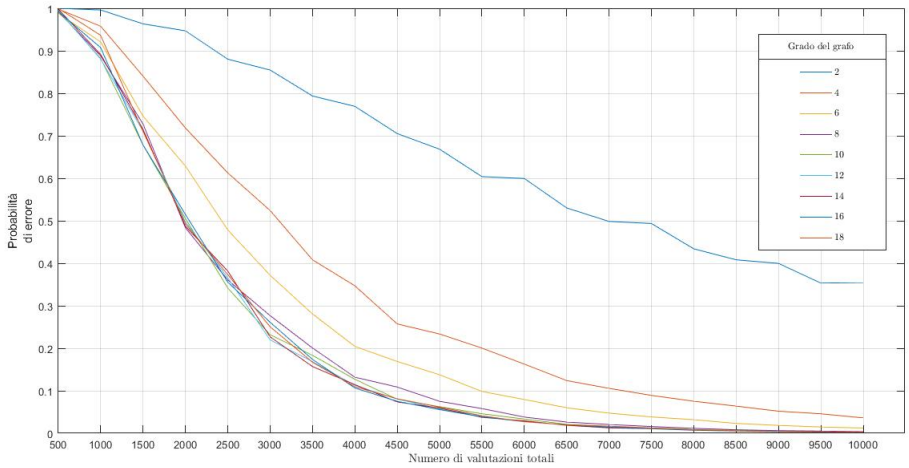


Figura 4.12. P_e al variare del numero totale di valutazioni e fissato il grado del grafo regolare nel caso $N = 20$ ed $\epsilon = 0.08$

Analizzando tali grafici è possibile evincere le medesime osservazioni già sottolineate nella sezione precedente. È infatti possibile rilevare un miglioramento in termini di P_e che, come osservato in precedenza, è sicuramente riconducibile alla scelta di utilizzare i pesi.

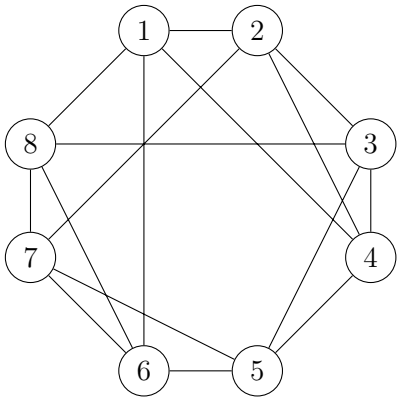
4.2 Studio degli algoritmi utilizzando un grafo regolare, connesso e pseudocasuale

In questa sezione riproponiamo lo studio effettuato in precedenza, ma utilizzando, questa volta, un grafo regolare, connesso e pseudocasuale. Le caratteristiche di questo grafo e la motivazione della sua scelta sono riportate nella seguente sottosezione.

Struttura del grafo

Le caratteristiche del grafo regolare a catena chiusa utilizzato in precedenza sono desiderabili nel nostro contesto, ma, se osserviamo più attentamente la struttura di tale grafo, ci accorgiamo che è presente un'evidente correlazione fra le coppie di oggetti che decidiamo di mettere a confronto. Se prendiamo infatti un grafo con un qualsiasi numero di nodi e tale che abbia almeno grado 4 ci accorgiamo immediatamente che, quando il nodo i viene confrontato con il nodo $i + 1$ ed il nodo $i + 1$ con il nodo $i + 2$, allora sicuramente anche i nodi i ed $i + 2$ sono oggetto di confronto. Questo fatto potrebbe non essere ottimale in quanto, note le differenze fra le qualità, seppure rumorose, fra i nodi i e $i + 1$ e fra i nodi $i + 1$ e $i + 2$, è allora possibile inferire anche una stima per la differenza fra le qualità dei nodi i ed $i + 2$. Anche se utilizzare un certo numero di valutazioni per stimare la differenza fra le qualità di i ed $i + 2$ può essere di aiuto al fine di ridurre il rumore nelle

suddette stime, come vedremo, sembra essere più conveniente sfruttare tali valutazioni per effettuare confronti fra altre coppie di oggetti. Un fenomeno di correlazione di questo tipo è presente se il grado del grafo a catena chiusa regolare è superiore a 2 e diventa tanto più evidente e pesante al crescere del grado dei nodi del grafo. Al fine di evitare, almeno in parte, questo fenomeno si è allora scelto di costruire un grafo tale che possiede le stesse proprietà utili del grafo precedente, ma scegliendo gli archi che collegano i nodi del grafo, per la maggior parte, in modo casuale. Infatti, al fine di ottenere con certezza un grafo connesso, abbiamo imposto che ciascun nodo fosse collegato al successivo e che l'ultimo fosse collegato al primo. A partire da tale grafo a catena chiusa di grado 2 abbiamo aggiunto ad ogni nodo un uguale numero di archi indiretti facendo in modo che i confronti indotti dai suddetti archi fossero casuali. Un esempio di grafo di questo tipo, prendendo il grafo con 8 nodi e di grado 4, è:



In questo modo il grafo costruito è sicuramente connesso e regolare ed il fenomeno di correlazione fra i confronti descritto in precedenza viene, in qualche modo, ridotto. Va infatti osservato come il fenomeno di correlazione sia presente ogni volta che nel grafo sono presenti cicli e come la correlazione sia tanto più forte quanto più i cicli presenti sono corti. Per questo motivo il grafo a catena chiusa non è ottimale: esso è infatti caratterizzato da un gran numero di triangoli, i quali chiaramente inducono un fenomeno di correlazione eccessivamente forte. Osserviamo come tale fenomeno, che (se presente con un certo grado di moderazione) può anche portare a benefici in termini di riduzione del rumore nelle stime delle differenze fra le qualità degli oggetti, è anche presente, in misura minore, nel grafo casuale appena descritto. Questo tipo di scelta per il grafo, come vedremo, permetterà di ottenere un miglioramento delle classifiche prodotte rispetto al caso in cui all'algoritmo viene associato un il grafo regolare a catena chiusa.

4.2.1 Studio dell'algoritmo LS

In modo analogo a quanto fatto in precedenza, in questa sezione presentiamo lo studio delle prestazioni dell'algoritmo nella sua versione non pesata (LS) nel caso in cui ci vengono forniti 20 oggetti sui quali si vuole costruire una classifica. Mostriamo, in particolare, le prestazioni dell'algoritmo a singolo stadio dove il grafo scelto è quello regolare, connesso

e pseudocasuale appena descritto. Presentiamo dunque, anche in questo caso, il comportamento dell'algoritmo al variare sia del numero totale di valutazioni a disposizione che del grado di ogni nodo del grafo regolare scelto. Esporremo, inoltre, le prestazioni dell'algoritmo, in termini di P_e e di D, per i valori 0.04 e 0.08 del parametro di tolleranza dell'errore ϵ.

Caso ϵ=0.04

Come illustrato nel dettaglio in precedenza, in questa sezione ci poniamo nel caso in cui siamo molto severi nel considerare corrette o sbagliate le classifiche che andiamo a produrre.

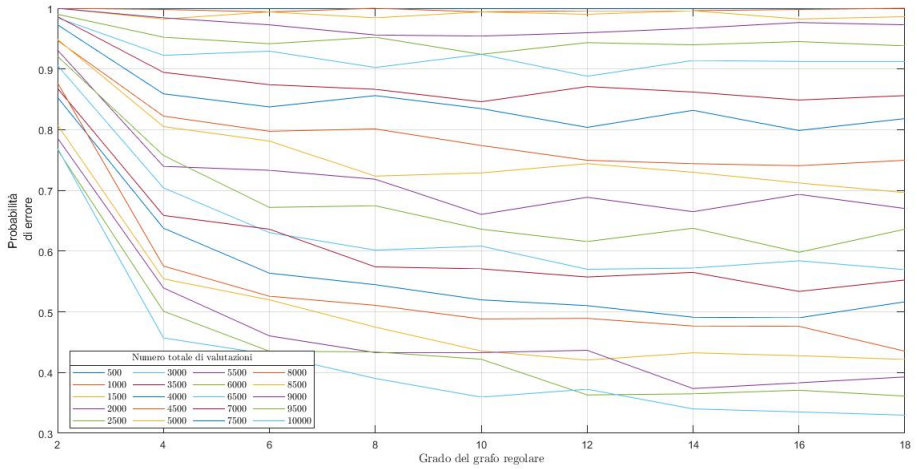


Figura 4.13. P_e al variare del grado del grafo regolare e fissato il numero di valutazioni totali nel caso $N = 20$ ed $\epsilon = 0.04$

Le figure 4.13. e 4.14. presentano il comportamento della probabilità di errore al variare, rispettivamente, del grado del grafo regolare e fissato il numero di valutazioni totali ed al variare del numero di valutazioni totali e fissato il grado del grafo regolare. È interessante effettuare una comparazione fra tali grafici e quelli presentati nelle figure 4.1. e 4.2., i quali sono prodotti sotto le medesime condizioni, ma associando all'algoritmo un grafo a catena chiusa. Per la figura 4.1., in particolare, avevamo osservato come, indipendentemente dal numero totale di valutazioni utilizzate, le prestazioni dell'algoritmo migliorassero sensibilmente al crescere del grado del grafo fino a quando non viene raggiunto il grado 8-10 e come, a partire da tale valore, ci sia una saturazione del miglioramento delle prestazioni. Il comportamento mostrato nella figura 4.13. è invece in qualche modo differente: si evince infatti un miglioramento molto più netto delle prestazioni dell'algoritmo quando il grado del grafo utilizzato è molto basso (ad esempio 4 e 6) e la saturazione del miglioramento della P_e è presente già a a partire da tali valori. Questo fatto è diretta conseguenza di quanto osservato sulla struttura del grafo scelto, ovvero del

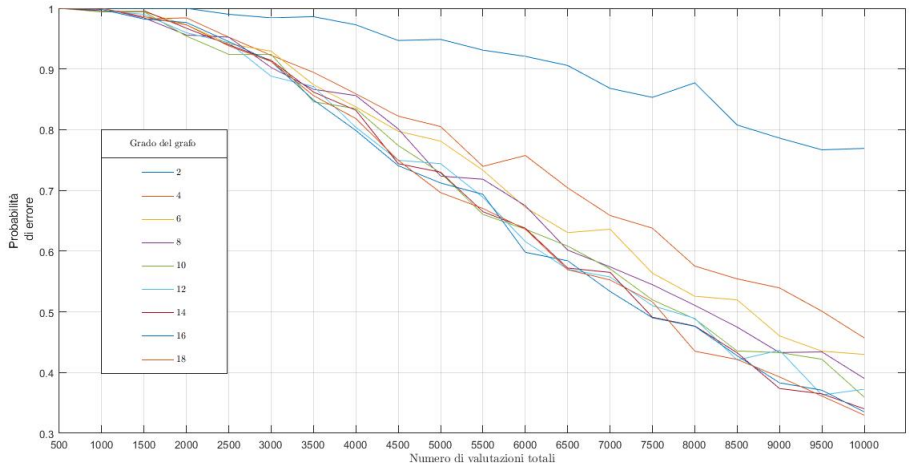


Figura 4.14. P_e al variare del numero totale di valutazioni e fissato il grado del grafo regolare nel caso $N = 20$ ed $\epsilon = 0.04$

fatto che, attraverso tale scelta, si sta attenuando il fenomeno di correlazione nella scelta delle coppie oggetti che decidiamo di mettere a confronto, che prima era sicuramente eccessivo. Risulta quindi evidente, ponendoci nella situazione in cui disponiamo di un pool di valutazioni da assegnare alle coppie di oggetti e per questa scelta dell'algoritmo, come scegliere a caso, a patto che il grafo indotto sia connesso, le coppie di oggetti da mettere a confronto sia conveniente a rispetto scegliere quelle indotte dal grafo regolare a catena chiusa. Osserviamo, inoltre, come, al crescere del numero totale di valutazioni utilizzate, si riscontri, anche in questo caso, un generale miglioramento delle prestazioni. È infatti possibile osservare come, poiché il numero di valutazioni che assegniamo all'algoritmo non è eccessivo, la descrescita della probabilità di errore sembri essere lineare al crescere del numero di valutazioni. Non si osserva dunque un fenomeno di saturazione del miglioramento delle prestazioni all'aumentare del numero di valutazioni di cui è possibile usufruire.

Osservando la figura 4.14. è possibile confermare le considerazioni che sono state fatte sul grafico precedente. In questo grafico è ancora più evidente come l'algoritmo migliori significativamente al crescere del grado del grafo quando esso passa da avere grado 2 a 4 e poi ancora, in modo più marginale, a 6, in quanto le curve corrispondenti sono effettivamente distanziate fra loro e non si sovrappongono. Le curve relative all'algoritmo applicato su grafi di grado maggiore di 6, invece, hanno un comportamento fra loro molto simile e questo è indice del fatto che presentano, a parità di valutazioni utilizzate, prestazioni simili fra loro. Osserviamo, inoltre, come il comportamento generale di tutte le curve sia decrescente e come questo confermi quanto evidenziato a partire dal grafico precedente, ovvero che le prestazioni dell'algoritmo migliorano sempre all'aumentare del numero di valutazioni a disposizione.

Considerazioni simili possono essere fatte osservando le figure 4.15. e 4.16., le quali

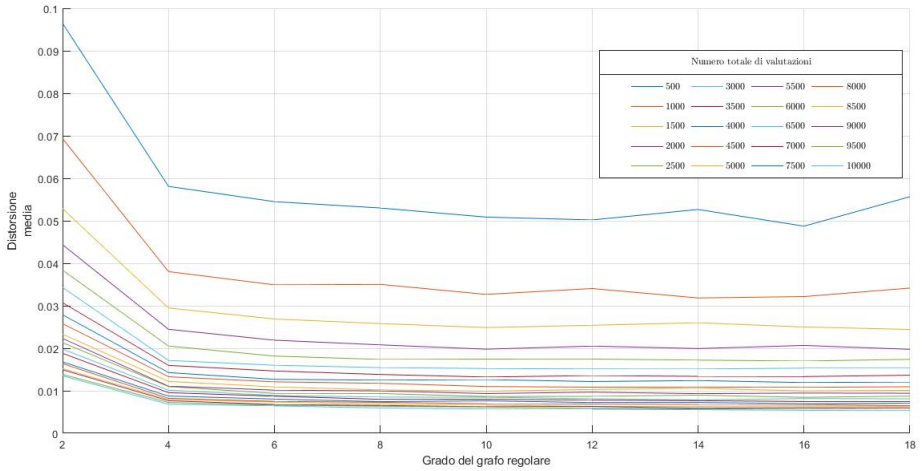


Figura 4.15. D al variare del grado del grafo regolare e fissato il numero di valutazioni totali nel caso $N = 20$ ed $\epsilon = 0.04$

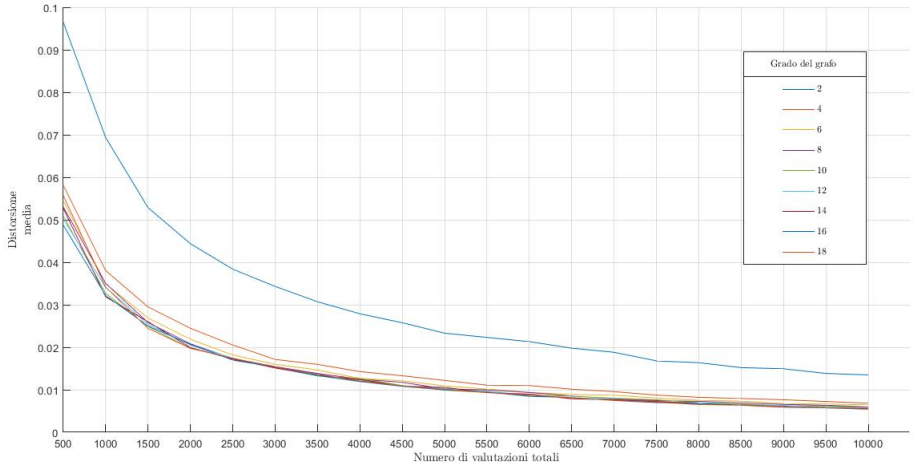


Figura 4.16. D al variare del numero totale di valutazioni e fissato il grado del grafo regolare nel caso $N = 20$ ed $\epsilon = 0.04$

riportano l'andamento della distorsione media rispettivamente al variare del grado del grafo regolare e fissato il numero di valutazioni totali ed al variare del numero totale di valutazioni e fissato il grado del grafo regolare. L'esperimento dal quale abbiamo ottenuto questi grafici è stato effettuato mantenendo costanti tutti i parametri utilizzati nell'esperimento che ci ha permesso di ricavare i grafici proposti nelle figure 4.13. e 4.14.. I grafici in questione accentuano le caratteristiche e le osservazioni che sono emerse

studiando il comportamento della probabilità di errore. Anche qui si osserva, infatti, un miglioramento delle prestazioni rispetto a quelle riportate dalle figure 4.9 e 4.10., in modo particolare, indipendentemente dal numero di valutazioni totali impiegate, quando il grafo utilizzato ha grado basso.

Nella sezione seguente viene nuovamente presentata l'analisi effettuata, ma fissando ϵ a 0.08.

Caso ϵ=0.08

L'obiettivo di questa sezione è di riproporre lo studio effettuato nella sezione precedente, ma nel caso in cui consideriamo corrette classifiche che scambiano di posizione coppie di oggetti con qualità che differiscono fra loro, in valore assoluto, per meno di 0.08. Siamo dunque più permissivi nel considerare corrette le classifiche che produciamo. Nessun altro parametro è stato modificato, rispetto alla sezione precedente, al fine di produrre i grafici presentati nelle figure 4.17. e 4.18..

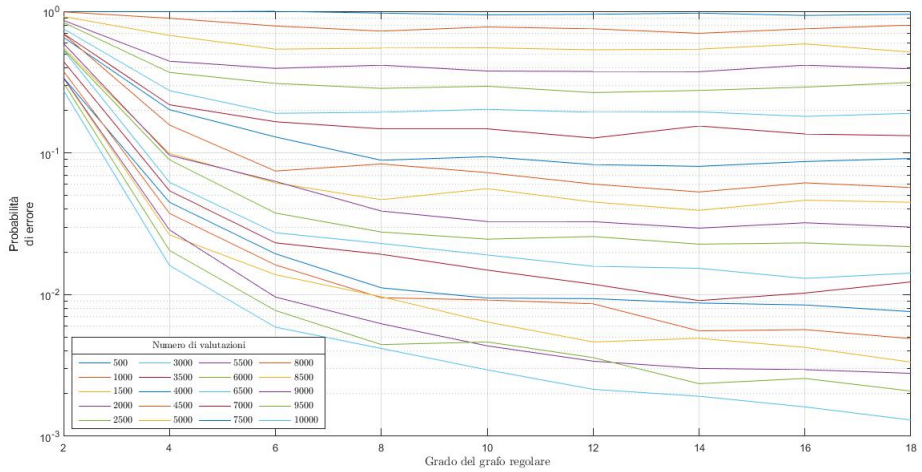


Figura 4.17. P_e al variare del grado del grafo regolare e fissato il numero di valutazioni totali nel caso $N = 20$ ed $\epsilon = 0.08$

In modo analogo a quanto fatto in precedenza, vogliamo effettuare un confronto fra questi grafici e quelli presentati nelle figure 4.5. e 4.6., i quali mostrano il comportamento della probabilità di errore nel caso in cui, nella simulazione, veniva utilizzato il grafo a catena chiusa. Ricordiamo che, come osservato in precedenza, a causa dei frequenti triangoli che si vengono a formare, un grafo di questo tipo induce una forte correlazione fra le coppie di oggetti che mettiamo a confronto. Effettuando allora un confronto di questo tipo ci accorgiamo come, anche in questo caso, si ottenga un generale e significativo miglioramento delle prestazioni, che risulta però essere decisamente più netto quando le valutazioni in gioco sono poche e quando il grado del grafo è ancora relativamente basso. Questo fatto è giustificabile dal fatto che, quando si ha a disposizione un numero basso di

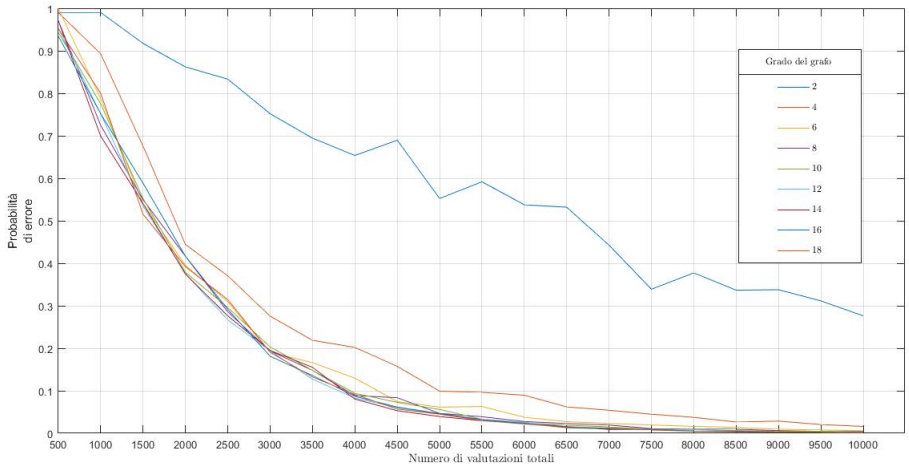


Figura 4.18. P_e al variare del numero totale di valutazioni e fissato il grado del grafo regolare nel caso $N = 20$ ed $\epsilon = 0.08$

valutazioni, il fenomeno di correlazione che, quando è presente in modo troppo marcato, tende a non sfruttare a pieno le risorse a disposizione, è più evidente. Quando invece il numero di valutazioni in gioco è alto questo fenomeno è chiaramente meno marcato in quanto abbiamo a disposizione un numero maggiore di risorse ed eventualmente, come si può osservare dai grafici, è anche possibile che l'algoritmo tragga benefici dall'assegnare valutazioni ad archi correlati, in quanto questo permette di ridurre il rumore presente nelle valutazioni.

Osserviamo inoltre che, come evidenziato per il caso di $\epsilon = 0.04$, anche qui è possibile rilevare, in particolare nella figura 4.18., come tutte le curve che mostrano la P_e al variare del numero di valutazioni e fissato il grado del grafo regolare abbiano un comportamento molto simile da quando viene utilizzato un grafo con grado superiore a 4. Questo significa che, di fatto, la scelta del grado del grafo, a parità del numero di valutazioni utilizzate e dunque della complessità scelta per il modello, è ininfluente in termini di probabilità di errore, a patto che il grado sia maggiore di 4.

4.2.2 Studio dell'algoritmo WLS

In questa sezione viene riproposto lo studio presentato nella sezione precedente, ma scegliendo di implementare l'algoritmo nella sua versione pesata (WLS). Anche in questo caso presenteremo, dunque, le prestazioni dell'algoritmo in termini di P_e e di D per le medesime scelte del parametro ϵ di tolleranza dell'errore operate nella sezione precedente.

Caso ϵ=0.04

L'obiettivo di questa sezione è di valutare le prestazioni dell'algoritmo WLS nel caso in cui, come detto, siamo molto severi nel giudicare la correttezza delle classifiche che andiamo a produrre.

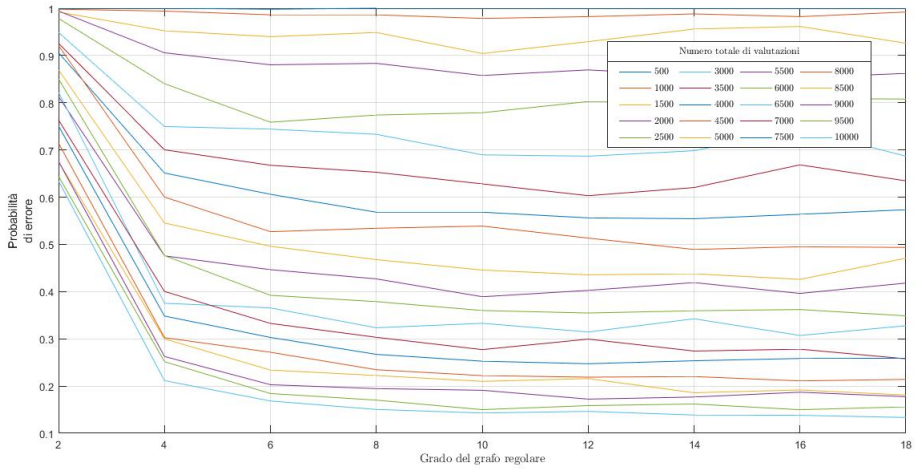


Figura 4.19. P_e al variare del grado del grafo regolare e fissato il numero di valutazioni totali nel caso $N = 20$ ed $\epsilon = 0.04$

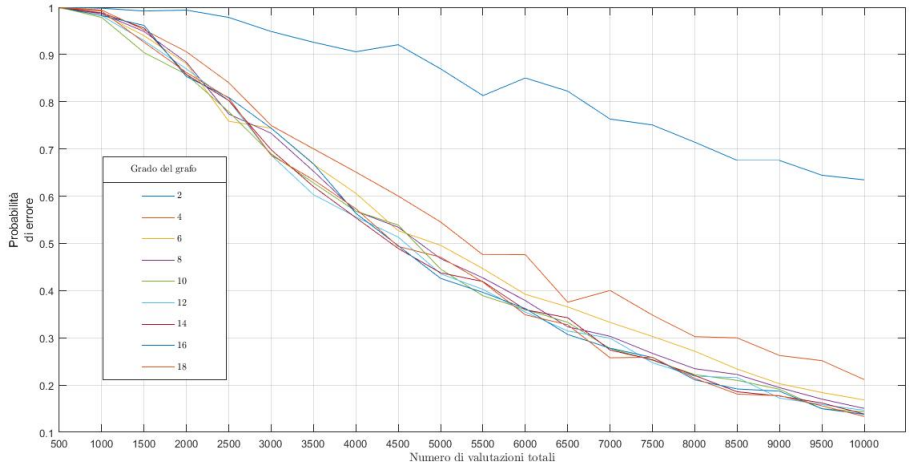


Figura 4.20. P_e al variare del numero totale di valutazioni e fissato il grado del grafo regolare nel caso $N = 20$ ed $\epsilon = 0.04$

Dai grafici riportati nelle figure 4.19. e 4.20. si evincono le stesse considerazioni fatte prestando quelli mostrati figure 4.13. e 4.14., i quali mostrano il comportamento della probabilità di errore per l'algoritmo nella versione non pesata quando $\epsilon = 0.04$. Tali grafici riportano infatti la P_e rispettivamente in funzione del grado del grafo regolare e fissato il numero totale di valutazioni ed in funzione del numero totale di valutazioni e fissato il grado del grafo. Dunque, quando scegliamo di utilizzare un grafo regolare, connesso e casuale, quanto detto per la versione non pesata dell'algoritmo vale anche per l'algoritmo nella sua versione pesata. Osserviamo infatti, anche in questa situazione, un netto miglioramento delle prestazioni, in modo particolare quando il grado del grafo associato all'algoritmo è basso. Questo risultato, esattamente come nel caso precedente, si spiega osservando che, attraverso la scelta di un grafo di questo tipo, i confronti indotti sugli oggetti soffrono in modo minore della correlazione e dunque il pool di valutazioni che si ha a disposizione viene sfruttato in modo più efficace. Va osservato però come l'aggiunta dei pesi comporti un significativo e generale miglioramento delle prestazioni dell'algoritmo in termini di probabilità di errore.

Le stesse considerazioni possono essere fatte osservando i grafici riportati nelle figure 4.15. e 4.16., i quali sono prodotti ripetendo il medesimo esperimento effettuato per ricavare quelli presentati nelle figure 4.13. e 4.14., ma che, al posto della P_e, riportano il comportamento della D.

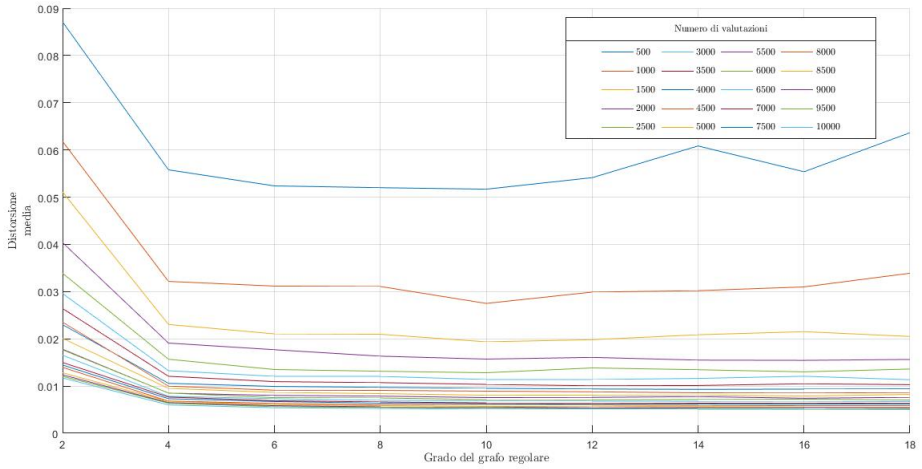


Figura 4.21. D al variare del grado del grafo regolare e fissato il numero di valutazioni totali nel caso $N = 20$ ed $\epsilon = 0.04$

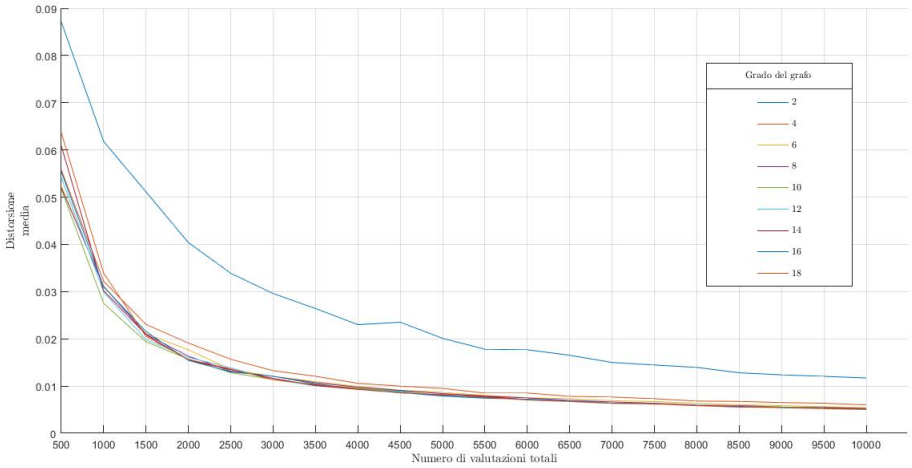


Figura 4.22. D al variare del numero totale di valutazioni e fissato il grado del grafo regolare nel caso $N = 20$ ed $\epsilon = 0.04$

Caso ϵ=0.08

In questa sezione riportiamo, infine, il comportamento della probabilità di errore nel caso in cui siamo più tolleranti nel considerare errate le classifiche che andiamo a produrre. A tal fine proponiamo le figure 4.23. e 4.24., dove, al solito, viene riportata la P_e rispettivamente al variare del grado del grafo completo e fissato il numero di valutazioni totali che vengono utilizzate ed al variare del numero totale di valutazioni, fissato il grado del grafo.

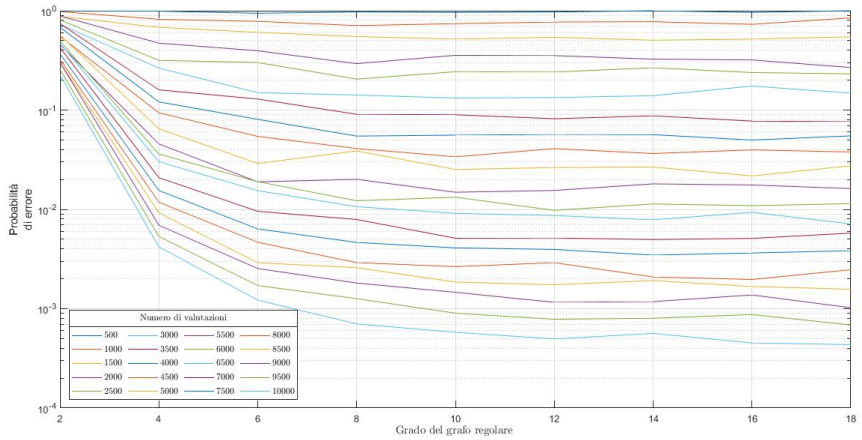


Figura 4.23. P_e al variare del grado del grafo regolare e fissato il numero di valutazioni totali nel caso $N = 20$ ed $\epsilon = 0.08$

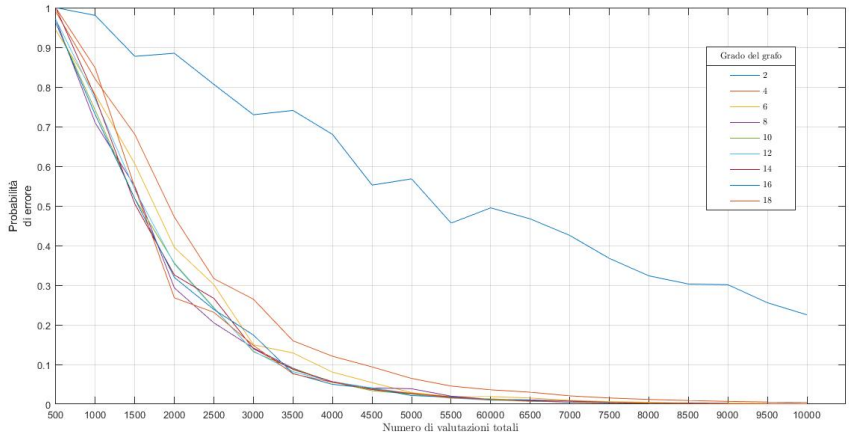


Figura 4.24. P_e al variare del numero totale di valutazioni e fissato il grado del grafo regolare nel caso $N = 20$ ed $\epsilon = 0.08$

Effettuando un confronto fra tali grafici e quelli presentati nel caso analogo dove, per la simulazione, si era scelto di utilizzare il grafo regolare a catena chiusa (figure 4.11. e 4.12.), è possibile evincere osservazioni del tutto analoghe a quelle riportate nelle sezioni precedenti.

Concludendo, sulla base del comportamento della P_e e della D ottenuti mediante simulazione, quando abbiamo la facoltà di distribuire a piacere le valutazioni possiamo affermare che, indipendentemente dal numero totale di risorse a disposizione, dalla scelta dell'algoritmo (LS o WLS), dal tipo di errori che decidiamo di tollerare sulle classifiche che produciamo e del grado del grafo regolare, è preferibile scegliere le coppie da mettere a confronto in modo casuale rispetto a selezionare quelle indotte dal grafo a catena chiusa. Questo fatto è giustificabile, come avevamo supposto, dal fatto che la struttura del grafo casuale induce una minore correlazione fra le coppie di oggetti che decidiamo di mettere a confronto. Questa scelta permette, dunque, di sfruttare con più efficacia il pool totale di valutazioni di cui disponiamo.

4.3 Confronto delle prestazioni ottenute utilizzando grafi di tipo diverso con grado 4 e 6

Mettendo a confronto lo studio effettuato nelle sezioni 4.1 e 4.2 è possibile osservare una generale e diffusa differenza nelle prestazioni dell'algoritmo a singolo stadio, sia nella sua versione pesata che non, imputabile alla struttura scelta per il grafo utilizzato nelle simulazioni. Si osserva, in particolare, come tale differenza sia però maggiormente marcata nel caso in cui il grado del grafo considerato è basso. Per questa ragione, in questa sezione, ci poniamo l'obiettivo di indagare più nel dettaglio l'impatto che la scelta della struttura del grafo regolare ha sulle prestazioni dell'algoritmo LS e WLS a singolo stadio, fissando

il grado di tale grafo a 4 ed a 6. Studieremo, in particolare, le prestazioni di tali algoritmi operando le seguenti scelte per la struttura del grafo:

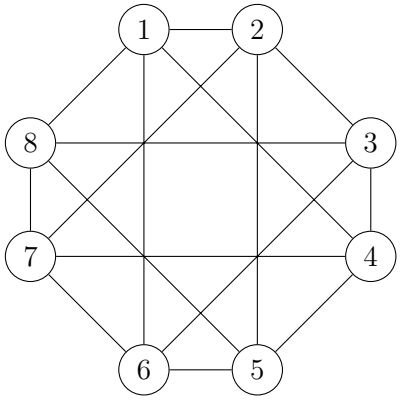
- Grafo a catena chiusa (descritto nella sezione 4.1);
- Grafo regolare, connesso e random (descritto nella sezione 4.2);
- Grafo multi-corona, che ora ci accingiamo a presentare.

Valuteremo, come effettuato nelle sezioni precedenti, le prestazioni dell'algoritmo mediante simulazione in termini di P_e e di D e fissando il parametro ϵ di tolleranza sugli errori nelle classifiche generate a 0.04 e 0.08.

Struttura e caratteristiche del grafo multi-corona

Questa sezione nasce con l'obiettivo di illustrare la struttura di una nuova tipologia di grafo regolare e connesso, al quale ci riferiremo come grafo multi-corona. L'osservazione che dà vita al grafo in questione è che la correlazione fra le coppie di oggetti che decidiamo di confrontare (indotta dalla struttura del grafo considerato) è tanto più marcata quanto i cicli presenti nel grafo stesso sono di piccole dimensioni. Per la costruzione del grafo a doppia corona di grado 4 abbiamo dunque deciso di sovrapporre al grafo a catena chiusa di grado 2 (al quale da ora in avanti ci riferiremo con il nome di grafo a corona) un ulteriore grafo a corona, il quale non è più ottenuto effettuando salti di un'unità per collegare, mediante un cammino chiuso, tutti i nodi, ma, per farlo, compie salti di 7 nodi. In questo modo il nodo 1 sarà collegato con il nodo 8, il nodo 8 con il nodo 15, il 15 con il 2 e così via. La scelta di fissare a 7 il passo dei salti è arbitraria e può essere modificata, a patto che esso sia un numero primo con il numero di nodi del grafo (20): questo permette, attraverso un unico ciclo, di collegare fra loro tutti i nodi prima di ritornare al nodo di partenza.

Di seguito riportiamo, a titolo esemplificativo, un grafo a doppia corona nel caso in cui si hanno 8 nodi, grado 4 e scegliamo un passo di 3:



Se osserviamo da un'altra angolazione quanto fatto per costruire il grafo a doppia corona in questione, ci accorgiamo che, dopo aver generato il classico grafo a corona,

abbiamo effettuato una permutazione dei nodi ed abbiamo generato, su tale grafo con i nodi permutati, un nuovo grafo a corona classico. Abbiamo infine sovrapposto i due grafi appena descritti per ottenere il nuovo grafo a doppia corona. La permutazione dei nodi utilizzata nel nostro caso specifico è: (1, 8, 15, 2, 9, 16, 3, 10, 17, 4, 11, 18, 5, 12, 19, 6, 13, 20, 7, 14). Iterando un procedimento di questo tipo sarà possibile, mediante la scelta di altre permutazioni dei nodi, generare grafi di grado superiore. Nel nostro caso specifico, infatti, per ottenere il grafo a tripla corona di grado 6, abbiamo scelto la permutazione indotta dalla scelta di 3 come numero di nodi da saltare a partire dal primo. La permutazione così generata è: (1, 4, 7, 10, 13, 16, 19, 2, 5, 8, 11, 14, 17, 20, 3, 6, 9, 12, 15, 18).

L'idea che ci ha portato a considerare un grafo di questo tipo, come detto, è che i confronti indotti sulle coppie di oggetti sono poco correlati fra loro. Nelle prossime sezioni studieremo le prestazioni che l'algoritmo a singolo stadio ottiene utilizzando un grafo di questo tipo. Si vuole indagare, in particolare, se, come è lecito aspettarsi, utilizzando un grafo multi-corona, si riscontrano benefici rispetto al caso in cui all'algoritmo viene associato un grafo del tipo descritto nelle sezioni 4.1 e 4.2, in modo particolare quando il grado scelto per i grafi è basso.

4.3.1 Caso di algoritmo LS e di grafo con grado 4

Iniziamo dunque il nostro studio presentando le prestazioni, in termini di P_e e di D, ottenute mediante l'impiego dei grafi citati, nel caso in cui il grado del grafo viene fissato a 4. Nel seguito studieremo l'algoritmo nella sua versione non pesata quando il parametro ϵ di tolleranza sugli errori è fissato a 0.04 e 0.08.

Caso $\epsilon = 0.04$

In questa sezione vogliamo studiare l'accuratezza delle classifiche prodotte nel caso in cui vengono tollerati solamente scambi di posizione fra coppie di oggetti le cui qualità reali differiscono, in valore assoluto, per un valore inferiore a 0.04. Le figure 4.25. e 4.26. riportano i grafici che mostrano il comportamento rispettivamente della P_e e della D al variare del numero totale di valutazioni e fissato il grado dei grafi utilizzati a 4. Tali grafici mostrano, in particolare, le prestazioni ottenute mediante l'utilizzo di un grafo a catena chiusa, un grafo random ed uno a doppia corona.

Tali grafici evidenziano un significativo miglioramento ottenuto mediante l'utilizzo di un grafo random o a doppia corona, rispetto ad uno a catena chiusa. Un comportamento di questo tipo delle misure di prestazioni è giustificabile dal fatto che, come detto, il grafo a catena chiusa induce una maggiore correlazione nelle coppie di oggetti che decidiamo di mettere a confronto. È inoltre interessante osservare come utilizzando il grafo a doppia corona che, come detto, induce una bassa correlazione fra le coppie di oggetti che scegliamo di confrontare, si ottengono, anche se in modo marginale, migliori prestazioni in termini di P_e, indipendentemente dalla quantità di valutazioni totali utilizzate, rispetto al caso in cui decidiamo di utilizzare il grafo random. Questo perché, quando consideriamo quest'ultima tipologia di grafi, fra quelli che vengono generati durante la simulazione,

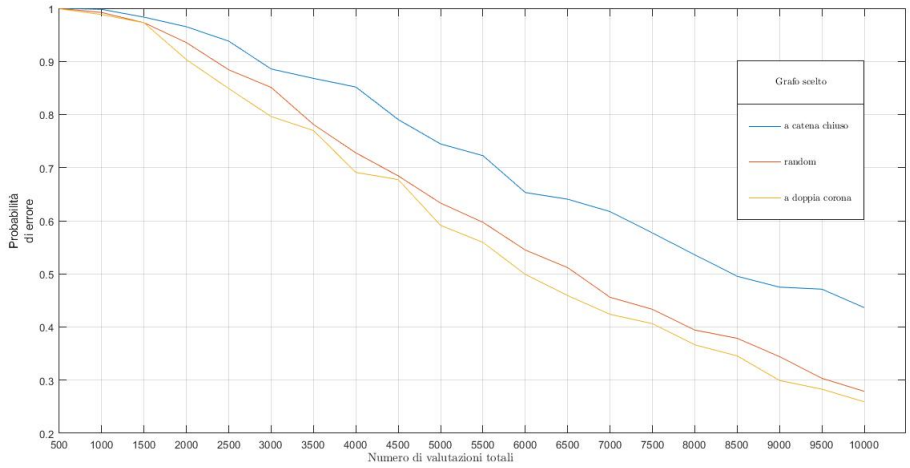


Figura 4.25. Confronto della P_e utilizzando grafi di tipo diverso al variare del numero di valutazioni totali e fissato il grado del grafo a 4 nel caso $N = 20$ ed $\epsilon = 0.04$

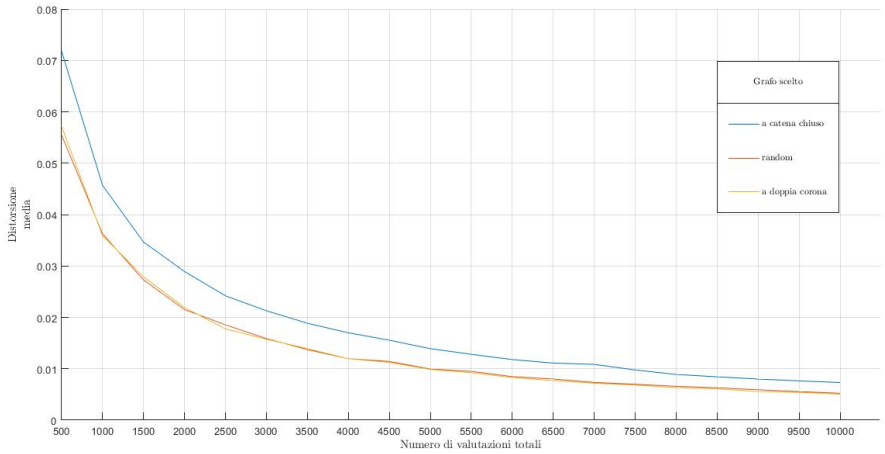


Figura 4.26. Confronto della D utilizzando grafi di tipo diverso al variare del numero di valutazioni totali e fissato il grado del grafo a 4 nel caso $N = 20$ ed $\epsilon = 0.04$

qualcuno sarà buono ed altri meno in termini di correlazione indotta fra le coppie che mettiamo a confronto.

Caso $\epsilon = 0.08$

Le figure 4.27. e 4.28. riportano nuovamente il comportamento della P_e e della D al variare del numero totale di valutazioni utilizzate e della struttura del grafo scelto, fissato

il grado a 4. La scelta per il parametro di tolleranza sugli errori ϵ è qui di 0.08 e questo fatto permette, come già osservato in precedenza, di essere più tolleranti nel giudicare corrette le classifiche generate.

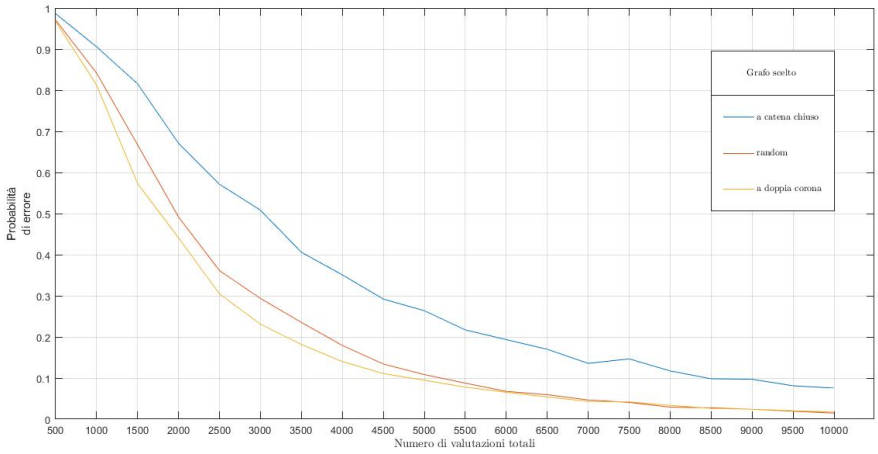


Figura 4.27. Confronto della P_e utilizzando grafi di tipo diverso al variare del numero di valutazioni totali e fissato il grado del grafo a 4 nel caso $N = 20$ ed $\epsilon = 0.08$

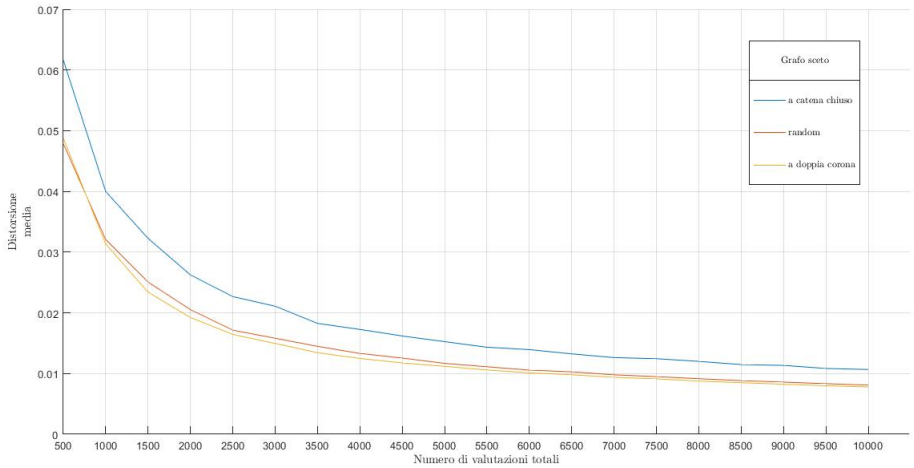


Figura 4.28. Confronto della D utilizzando grafi di tipo diverso al variare del numero di valutazioni totali e fissato il grado del grafo a 4 nel caso $N = 20$ ed $\epsilon = 0.08$

Osservando tali grafici è possibile evincere le medesime conclusioni che sono state evidenziate nella sezione precedente, dove il parametro di tolleranza degli errori ϵ era stato fissato a 0.04.

4.3.2 Caso di algoritmo LS e di grafo con grado 6

L'obiettivo di questa sezione è di riproporre lo studio effettuato nella sezione 4.3.1 sull'algoritmo LS, ma nel caso in cui il grado del grafo è fissato a 6 (per il grafo multi-corona viene dunque scelto il grafo a tripla corona). In particolare, analogamente a quanto fatto in precedenza presenteremo le prestazioni dell'algoritmo in termini di P_e e di D e per i valori di $\epsilon = 0.04$ ed $\epsilon = 0.08$.

Caso $\epsilon = 0.04$

I grafici riportati nelle figure 4.29. e 4.30. riportano il comportamento della P_e e della D al variare del numero di valutazioni totali utilizzate e della struttura scelta per il grafo, quando il grado è fissato a 6. Qui il parametro di tolleranza degli errori sulle classifiche prodotte ϵ è fissato a 0.04.

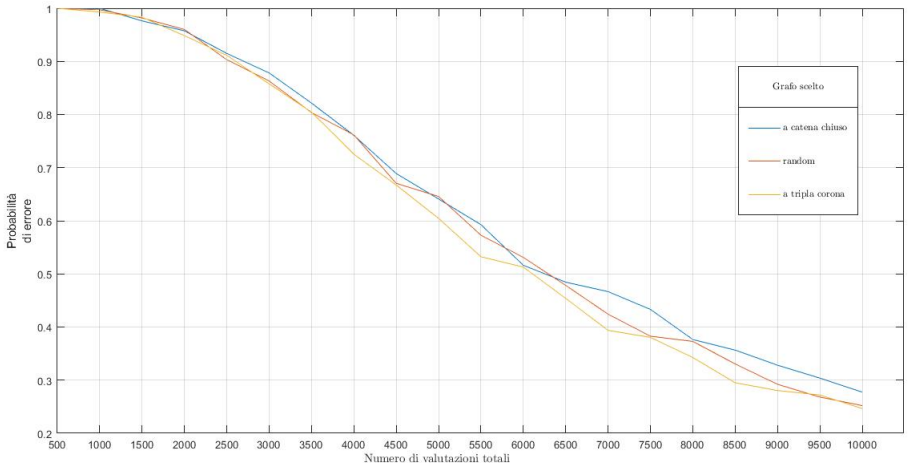


Figura 4.29. Confronto della P_e utilizzando grafi di tipo diverso al variare del numero di valutazioni totali e fissato il grado del grafo a 6 nel caso $N = 20$ ed $\epsilon = 0.04$

Attraverso il confronto fra tali grafici e quelli presentati nelle figure 4.25. e 4.26., prodotti sotto le medesime condizioni, ma utilizzando grafi di grado 4, è possibile evidenziare un generale miglioramento delle prestazioni dell'algoritmo LS che si ottengono utilizzando un grafo di grado 6 rispetto ad uno di grado 4. Questo fatto, già rilevato nelle sezioni precedenti quando nelle simulazioni venivano utilizzati grafi a catena chiusa e random (e che ora osserviamo valere anche per i grafi multi-corona), è imputabile al fatto che l'algoritmo trova giovamento, quando il grado del grafo utilizzato è ancora così basso, nel distribuire il pool totale di valutazioni a disposizione su un numero maggiore di archi. Si preferisce dunque confrontare un numero maggiore di coppie di oggetti con un numero minore di valutazioni invece che confrontare, con un numero maggiore di valutazioni, un numero più ristretto di coppie.

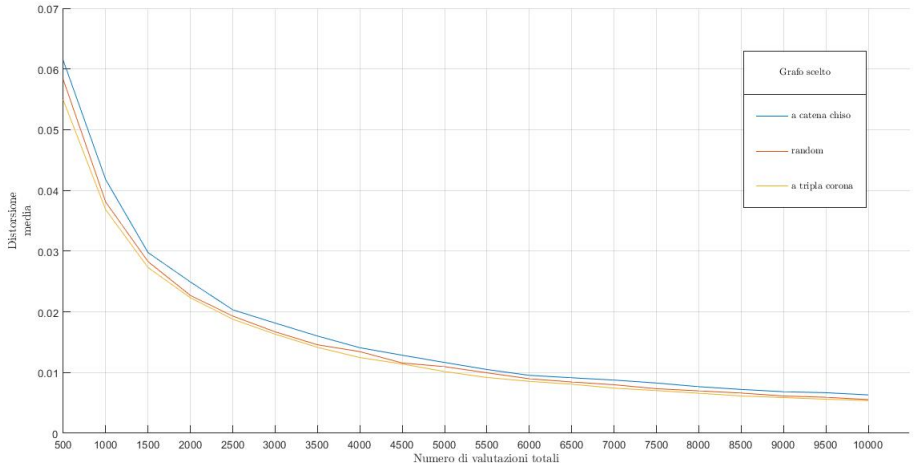


Figura 4.30. Confronto della D utilizzando grafi di tipo diverso al variare del numero di valutazioni totali e fissato il grado del grafo a 8 nel caso $N = 20$ ed $\epsilon = 0.04$

Osservando le figure 4.29. e 4.30. appare inoltre evidente come, aumentando anche solo di 2 il grado dei grafi utilizzati, la differenza nelle prestazioni dell'algoritmo LS, evidenziata quando venivano utilizzati nelle simulazioni grafi di grado 4, ottenuta mettendo a confronto le prestazioni ricavate mediante l'impiego del grafo a catena chiusa rispetto alle altre tipologie di grafo, è di molto ridotta. Anche in questo caso sembra comunque essere conveniente, anche se il miglioramento ottenuto è molto basso, scegliere, rispetto agli altri, il grafo multi-corona.

Caso $\epsilon = 0.08$

Analogamente alla sezione precedente i grafici riportati nelle figure 4.31. e 4.32. mostrano il comportamento della P_e e della D al variare del numero di valutazioni utilizzate e, fissato a 6 il grado, della scelta della struttura del grafo. Studiamo qui, in particolare, le prestazioni dell'algoritmo LS nel caso $\epsilon = 0.08$.

Osservando tali figure è possibile evidenziare le medesime osservazioni fatte per le figure 4.29. e 4.30., le quali sono state generate sotto le medesime condizioni, ma presentano le prestazioni dell'algoritmo quando siamo più fiscali nel considerare corrette o errate le classifiche che andiamo a produrre. Otteniamo, mediante questa scelta, come già osservato nelle sezioni precedenti, curve maggiormente stabili, che permettono di cogliere in modo più netto il comportamento delle nostre misure di prestazione.

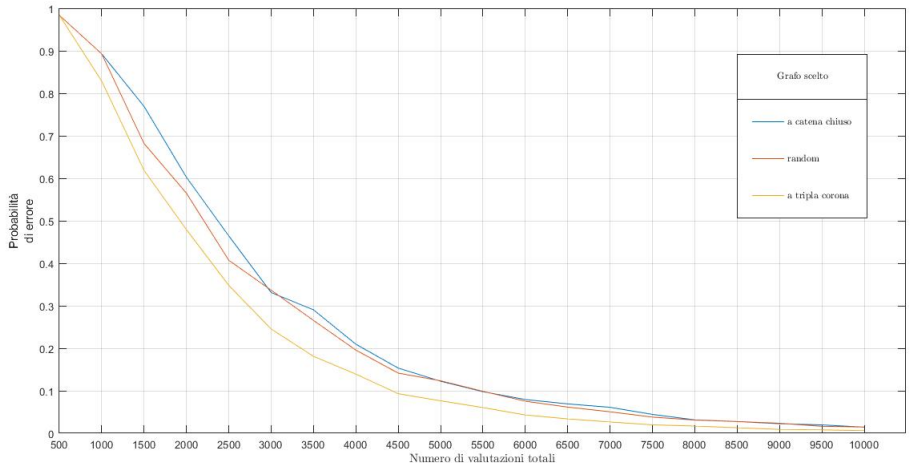


Figura 4.31. Confronto della P_e utilizzando grafi di tipo diverso al variare del numero di valutazioni totali e fissato il grado del grafo a 6 nel caso $N = 20$ ed $\epsilon = 0.08$

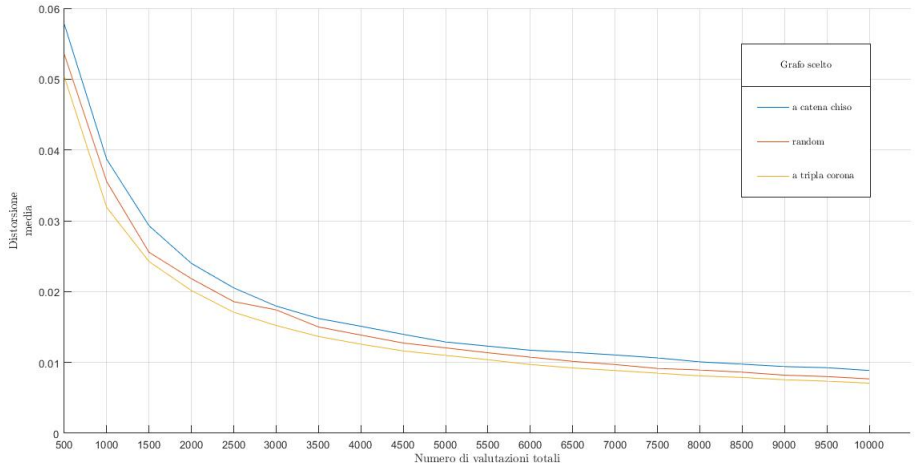


Figura 4.32. Confronto della D utilizzando grafi di tipo diverso al variare del numero di valutazioni totali e fissato il grado del grafo a 6 nel caso $N = 20$ ed $\epsilon = 0.08$

4.3.3 Caso di algoritmo WLS e di grafo con grado 4

In questa sezione riproponiamo lo studio effettuato nella sezione 4.3.1 per l'algoritmo LS, ma scegliendo questa volta la versione pesata dell'algoritmo. Studieremo dunque le prestazioni, in termini di P_e e di D, al variare della struttura scelta per il grafo, quando ne fissiamo il grado a 4. Nelle due sottosezioni seguenti, in particolare, presenteremo lo

studio per ϵ fissato a 0.04 ed a 0.08.

Caso $\epsilon = 0.04$

Le figure 4.33. e 4.34. mostrano la P_e e la D al variare del numero totale di valutazioni di cui si usufruisce e della struttura del grafo, fissatone il grado a 4. Al fine di produrre tali grafici, in particolare, il parametro di tolleranza sugli errori delle classifiche prodotte è stato fissato ad $\epsilon = 0.04$.

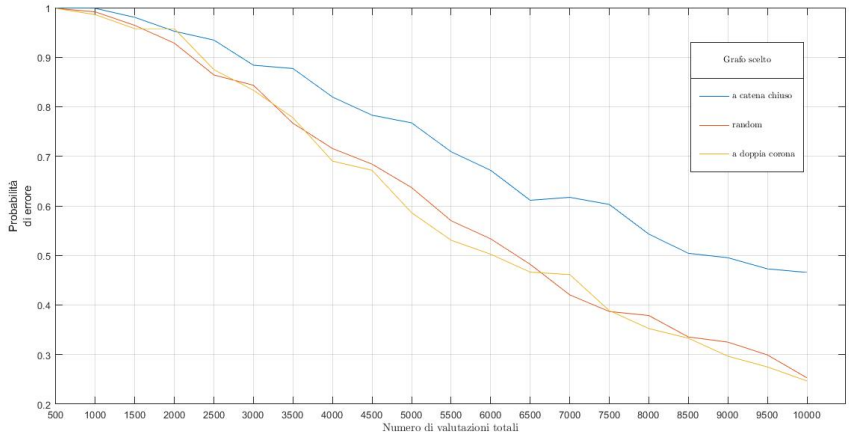


Figura 4.33. Confronto della P_e utilizzando grafi di tipo diverso al variare del numero di valutazioni totali e fissato il grado del grafo a 4 nel caso $N = 20$ ed $\epsilon = 0.04$

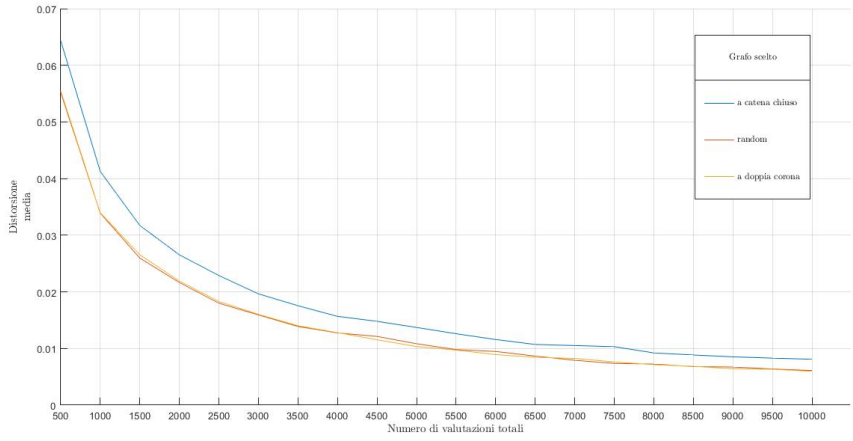


Figura 4.34. Confronto della D utilizzando grafi di tipo diverso al variare del numero di valutazioni totali e fissato il grado del grafo a 4 nel caso $N = 20$ ed $\epsilon = 0.04$

Mettendo a confronto tali grafici con quelli presentati nelle figure 4.25. e 4.26. è possibile osservare come non sia presente, per questa scelta del grado dei grafi, un significativo miglioramento delle prestazioni apportato dall'aggiunta dei pesi. È inoltre possibile rilevare come, anche in questo caso, utilizzando un grafo a catena chiusa si ottengono prestazioni peggiori rispetto al caso in cui si sceglie di associare all'algoritmo un grafo random o a doppia catena. Si osserva inoltre come, in termini di P_e e di D, sia indifferente scegliere di utilizzare uno degli ultimi due grafi appena citati.

Caso $\epsilon = 0.08$

Riportiamo poi, in questa sezione, le figure 4.35. e 4.36. che, analogamente a quanto fatto nella sezione precedente, riportano la P_e e la D al variare del numero totale di valutazioni utilizzate e della struttura scelta per il grafo, il cui grado è fissato, anche in questo caso, a 4. Qui, a differenza della sezione precedente, il parametro di tolleranza dell'errore ϵ è fissato a 0.08.

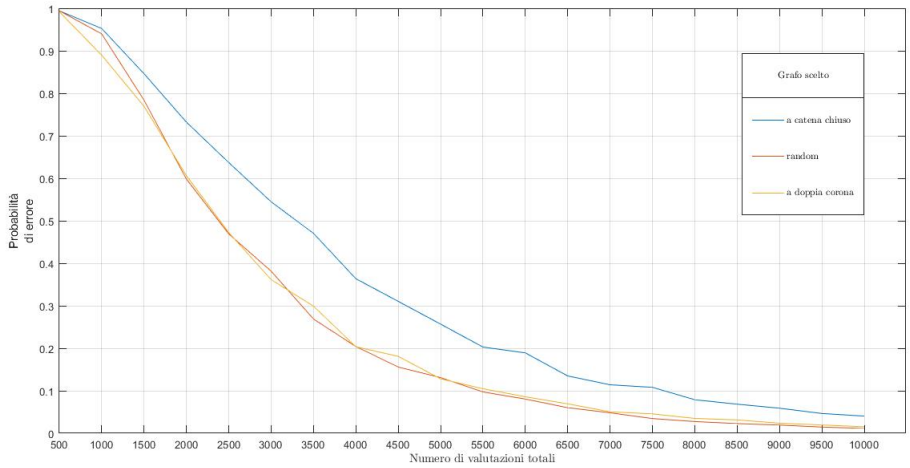


Figura 4.35. Confronto della P_e utilizzando grafi di tipo diverso al variare del numero di valutazioni totali e fissato il grado del grafo a 4 nel caso $N = 20$ ed $\epsilon = 0.08$

Osservando tali figure è possibile mettere in risalto le medesime considerazioni fatte per i grafici presentati nelle figure 4.33. e 4.34..

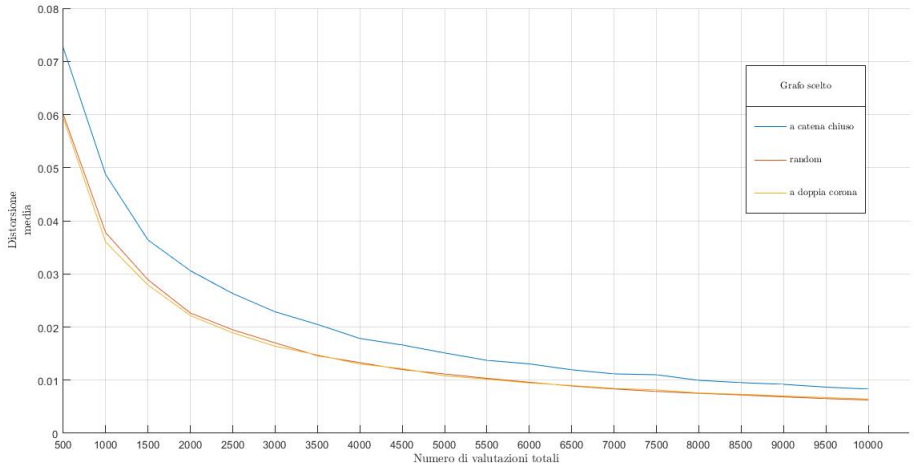


Figura 4.36. Confronto della D utilizzando grafi di tipo diverso al variare del numero di valutazioni totali e fissato il grado del grafo a 4 nel caso $N = 20$ ed $\epsilon = 0.08$

4.3.4 Caso di algoritmo WLS e di grafo con grado 6

Studiamo, infine, le prestazioni dell'algoritmo, nella sua versione pesata, sempre al variare della struttura del grafo scelto e nel caso in cui il grado è fissato a 6. Esattamente come nelle sezioni precedenti, in particolare, presenteremo il comportamento della P_e e della D fissati i valori di ϵ a 0.04 e 0.08.

Caso $\epsilon = 0.04$

Le figure 4.37. e 4.38. riportano rispettivamente il comportamento della P_e e della D al variare del numero di valutazioni totali utilizzate e della struttura scelta per il grafo, quando esso ha grado 6 ed ϵ vale 0.04.

A differenza di quanto riportato nella sezione 4.3.2, dove sono state presentate, sotto le medesime condizioni, le prestazioni della versione non pesata dell'algoritmo, qui osserviamo un miglioramento nell'utilizzo di un grafo random o a doppia corona rispetto ad uno a catena chiusa. Il comportamento del tutto simile, sia in termini di P_e che di D, delle curve relative al grafo random ed a quello a doppia corona suggerisce che la scelta dell'una o dell'altra struttura per il grafo è indifferente.

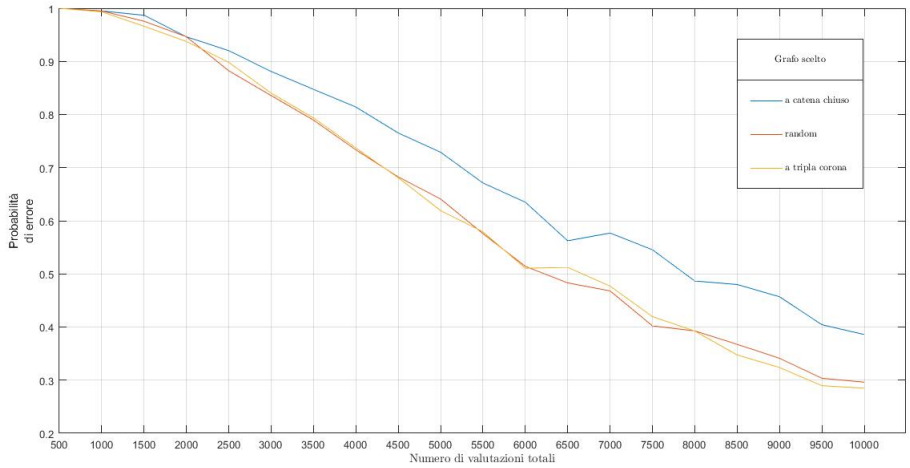


Figura 4.37. Confronto della P_e utilizzando grafi di tipo diverso al variare del numero di valutazioni totali e fissato il grado del grafo a 6 nel caso $N = 20$ ed $\epsilon = 0.04$

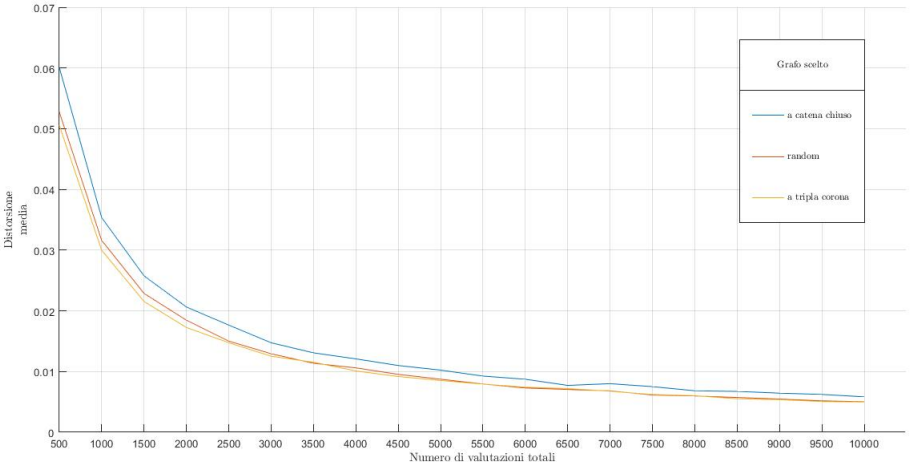


Figura 4.38. Confronto della D utilizzando grafi di tipo diverso al variare del numero di valutazioni totali e fissato il grado del grafo a 6 nel caso $N = 20$ ed $\epsilon = 0.04$

Caso $\epsilon = 0.08$

Riportiamo, infine, nelle figure 4.39. e 4.40., il comportamento della P_e e della D ricavato simulando la versione pesata dell'algoritmo al variare del numero totale di valutazioni e della struttura del grafo, nel caso in cui il grado è fissato a 6 ed $\epsilon = 0.08$.

Analizzando tali figure è possibile ricavare le medesime osservazioni sottolineate per i

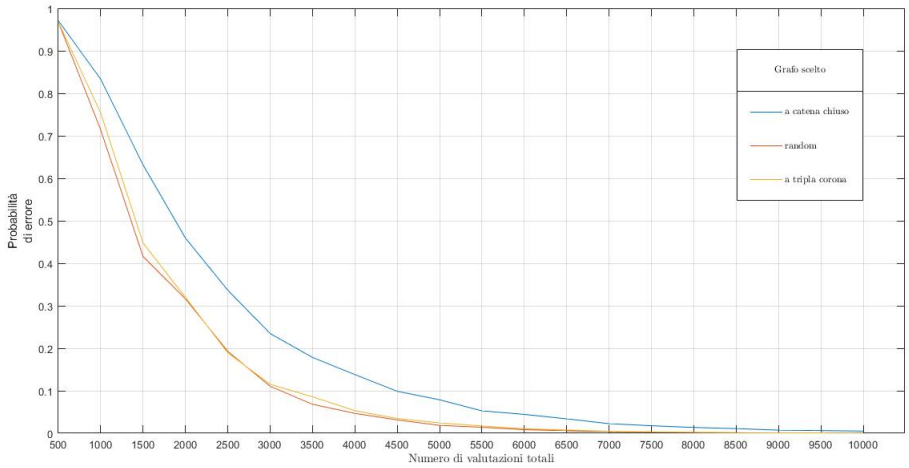


Figura 4.39. Confronto della P_e utilizzando grafi di tipo diverso al variare del numero di valutazioni totali e fissato il grado del grafo a 6 nel caso $N = 20$ ed $\epsilon = 0.08$

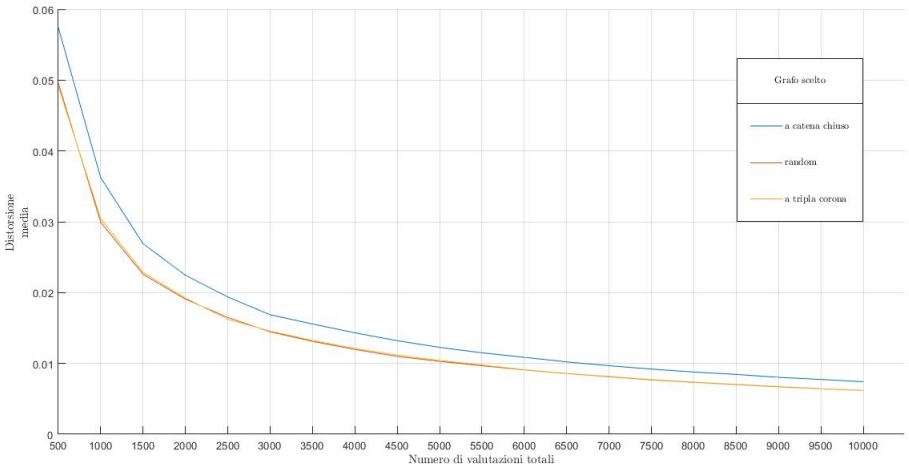


Figura 4.40. Confronto della D utilizzando grafi di tipo diverso al variare del numero di valutazioni totali e fissato il grado del grafo a 6 nel caso $N = 20$ ed $\epsilon = 0.08$

grafici mostrati nelle le figure 4.27. e 4.28., ovvero quelli relativi alla sezione precedente, dove il valore di tolleranza dell'errore ϵ era fissato a 0.04.

Concludendo osserviamo come, nel caso in cui siamo forzati (per qualche vincolo pratico) a mantenere basso il grado del grafo regolare che decidiamo di utilizzare per indurre i

confronti fra coppie di oggetti, sia allora conveniente scegliere un grafo multi-corona. Attraverso le simulazioni precedenti abbiamo infatti dimostrato come tale struttura del grafo permetta di eguagliare o migliorare la riduzione nella correlazione fra le coppie di oggetti che scegliamo di mettere a confronto che viene ottenuta mediante un grafo random. Tale tipologia di grafo permette infatti di ottenere risultati generalmente migliori rispetto a quelli ottenuti mediante l'impiego di un grafo a catena chiusa ed i risultati sono affetti da una minore variabilità rispetto a quelli che vengono generati utilizzando un grafo random. Attraverso quest'ultima scelta, infatti, poiché consideriamo grafi di piccole dimensioni, c'è la possibilità di cadere in casi sfortunati, dove il grafo random generato induce una forte correlazione fra le coppie di oggetti che mettiamo a confronto, e questo produrrà risultati non ottimali.

Va però osservato come il beneficio apportato dalla scelta della struttura del grafo da utilizzare per l'algoritmo a singolo stadio (sia nella sua versione pesata che non) è marginale. I veri miglioramenti, infatti, si ottengono o aumentando il grado del grafo (almeno fino a 10) o aumentando il numero di risorse, ovvero di valutazioni, a disposizione.

4.4 Confronto fra l'algoritmo LS e WLS

A supporto di quanto osservato nelle sezioni precedenti, dove si era rilevato un generale miglioramento nell'utilizzo dell'algoritmo nella sua versione pesata rispetto a quello LS, presentiamo i risultati delle simulazioni degli algoritmi LS e WLS ottenuti utilizzando il grafo con struttura casuale. Si è scelto, in particolare, di mettere a confronto le prestazioni dei due algoritmi quando il grado del grafo è basso (4) e quando esso è alto (16), fissando il parametro ϵ di tolleranza sugli errori nelle classifiche prodotte a 0.08. I risultati della simulazione sono riportati nella figura 4.41., dove viene mostrato il comportamento della probabilità di errore al variare del numero totale di valutazioni di cui si usufruisce.

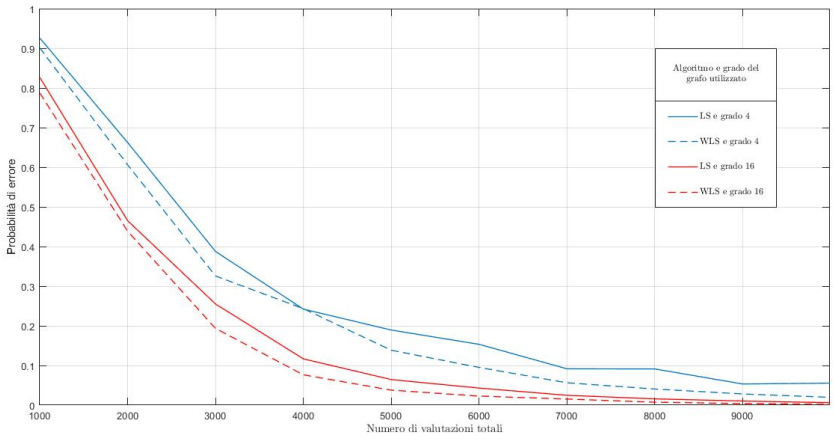


Figura 4.41. Confronto fra l'algoritmo LS e WLS in termini di P_e utilizzando un grafo casuale di grado 4 e 16, nel caso $N = 20$ ed $\epsilon = 0.08$

In tale grafico è possibile osservare, in modo esplicito, il miglioramento che si ottiene, in termini di probabilità di errore, mediante l'utilizzo dei pesi.

Capitolo 5

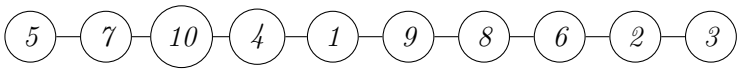
Ranking di un numero finito di oggetti: approccio a due stadi

In questo capitolo, come accennato in precedenza, l'obiettivo è di presentare e studiare una variante dell'algoritmo discusso nella capitolo 3. Al fine di migliorare, in termini di accuratezza, le prestazioni dell'algoritmo presentate nel quarto capitolo la nostra idea è, infatti, di creare ed utilizzare una versione a due stadi dell'algoritmo. A tale scopo fissiamo a priori il numero totale di valutazioni W e partizioniamo tale pool in due sottoinsiemi: il primo, W_1, assegnato al primo stadio ed il secondo, W_2, al secondo. Il nostro approccio consiste dunque nel prendere le risorse, e cioè le valutazioni, assegnate al primo stadio e, sulla base di esse, estrarre e salvare le valutazioni che vengono espresse sugli archi e la classifica che viene generata. Ciò che in realtà viene registrato, al termine di questa fase, non è l'intero insieme di valutazioni espresse per ogni arco, ma un termine che aggrega tali singole informazioni in un unico valore associato ad ogni arco del grafo (cioè ad ogni coppia di oggetti che decidiamo di confrontare al primo stadio). Tale termine è dunque un vettore $\mathbf{p^{(1)}}$ di dimensione pari al numero di archi del grafo dove ogni componente $p_{i,j}^{(1)}$ è relativa ad un determinato arco (i, j) e rappresenta la probabilità che, sulla base delle valutazioni che ci vengono fornite, il primo nodo i dell'arco abbia una qualità maggiore rispetto secondo j. Osserviamo come la classifica prodotta e salvata a questo stadio sarà più o meno accurata a seconda del valore di W_1, ovvero della quantità di risorse che decidiamo di assegnare al primo stadio. La scelta del grafo utilizzato, il quale induce i confronti fra le coppie di oggetti, è ricaduta in questa prima fase sul grafo a catena chiusa che è stato descritto nella relativa sezione del capitolo precedente dedicato all'approccio a singolo stadio.

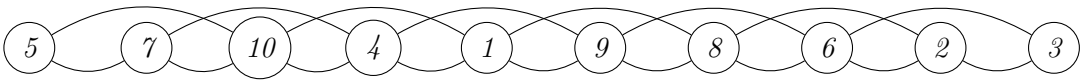
A questo punto le informazioni acquisite al primo stadio, cioè il vettore $\mathbf{p^{(1)}}$ delle probabilità assegnate agli archi e la classifica grezza, vengono fornite come input al secondo stadio dell'algoritmo. Il fine di tale stadio è di prendere la classifica prodotta alla fase precedente e cercare di migliorarla quanto più possibile. È infatti verosimile supporre che la classifica generata al primo stadio possa eventualmente commettere errori scambiando di posto oggetti che, nella classifica reale, occupano posizioni relativamente vicine fra loro. Insiemi di oggetti di questo tipo avranno infatti qualità simili fra loro. La nostra idea

sarà dunque quella di mettere a confronto, al secondo stadio, coppie di oggetti che sono disposti vicini nella classifica generata al primo stadio. A tal fine prendiamo l'ordinamento prodotto al primo stadio e, sulla base di esso, generiamo ed utilizziamo al secondo stadio un grafo a catena aperta, che connette, attraverso i suoi archi, gli oggetti che sono disposti vicini nella classifica prodotta alla fase precedente. Non ha più senso, infatti, utilizzare un grafo a catena chiusa in quanto, verosimilmente, non sarà più conveniente confrontare coppie di oggetti collocati in posizioni così lontane nella classifica prodotta allo stadio precedente. Siamo già infatti sufficientemente confidenti del fatto che tali oggetti abbiano qualità molto differenti fra loro. Per comprendere meglio quanto detto studiamo il seguente esempio pratico.

Esempio 5.0.1 *Consideriamo, a titolo esemplificativo, il caso in cui occorre produrre una classifica su 10 oggetti e dove la classifica prodotta al primo stadio è ad esempio: 5, 7, 10, 4, 1, 9, 8, 6, 2, 3. Se consideriamo allora il grafo a catena aperta con grado 2, il grafo utilizzato al secondo stadio sarà:*



Se invece, ponendoci esattamente nella situazione descritta in precedenza, scegliessimo di utilizzare un grafo a catena aperta di grado 4, quello che si otterrebbe è il grafo:



Osserviamo che i grafi così costruiti non sono regolari, cioè non tutti i nodi hanno lo stesso grado. In particolare, mentre tutti i nodi situati nella parte centrale hanno lo stesso grado, i nodi che occupano le due estremità hanno chiaramente grado minore. Quando parleremo dunque di grado del grafo, per grafi di questo tipo, ci riferiremo al grado dei nodi centrali. Osserviamo inoltre come, dato un grafo di questa tipologia dotato, in generale, di N nodi e di grado d, questo avrà un numero di archi pari a:

$$\#\text{ archi} = \left(N - \frac{d}{2}\right)\frac{d}{2} + \sum_{i=1}^{\frac{d}{2}-1} i = \left(N - \frac{d}{2}\right)\frac{d}{2} + \frac{\frac{d}{2}\left(\frac{d}{2}-1\right)}{2} = \frac{Nd}{2} - \frac{d^2}{8} - \frac{d}{4}$$

Come detto, la scelta di utilizzare, al secondo stadio, un grafo di questo tipo permette di confrontare le coppie di oggetti indotte dagli archi indiretti del grafo e dunque, in questo modo, stiamo raggiungendo l'obiettivo che ci siamo prefissati a questo stadio. A questo punto è possibile sfruttare contemporaneamente le valutazioni espresse al primo ed al secondo stadio e, sulla base di esse, produrre la classifica finale. In particolare, per sfruttare le informazioni provenienti dai due stadi, abbiamo considerato il vettore delle probabilità prodotte al primo stadio $\mathbf{p}^{(1)}$ ed al secondo stadio $\mathbf{p}^{(2)}$. Per ogni arco, nel caso in cui esso appartenga solo ad uno dei grafi ai due stadi, abbiamo utilizzato il valore corrispondente del relativo vettore delle probabilità per il calcolo della matrice $\mathbf{D}$. Quando, invece, l'arco (i, j) è presente in entrambi i grafi, abbiamo ricavato una probabilità finale

dato dalla media pesata delle probabilità associate all'arco nei due stadi, utilizzando come pesi il numero di valutazioni per arco utilizzate ai due stadi. Vale infatti che:

$$p_{i,j}^{(\text{fin})} = \frac{k_1 p_{i,j}^{(1)} + k_2 p_{i,j}^{(2)}}{k_1 + k_2}$$

dove k_1 e k_2 rappresentano appunto il numero di valutazioni assegnate rispettivamente al primo ed al secondo stadio per ogni arco e $p_{i,j}^{(1)}$ e $p_{i,j}^{(2)}$ le probabilità associate all'arco (i, j) al primo ed al secondo stadio. In questo modo, almeno per gli archi presenti ad entrambi gli stadi, si è tenuto conto del fatto che le due probabilità non sono state ricavate sulla base dello stesso numero di valutazioni e che, di conseguenza, esse non sono ugualmente affidabili.

Di tale approccio studieremo, in prima battuta, la versione non pesata e vedremo come questo metodo porta con sé un problema intrinseco nel modo in cui le valutazioni espresse ai due stadi vengono combinate. Vedremo dunque come tale problema intrinseco dell'algoritmo sarà evidenziato dalle prestazioni della probabilità di errore e della distorsione media e come esso potrà essere superato attraverso l'utilizzo della versione pesata dell'algoritmo.

Il modello utilizzato per descrivere il comportamento dei lavoratori nelle simulazioni successive è esattamente quello descritto nell'apposita sottosezione del quarto capitolo.

Le prestazioni dell'algoritmo sono studiate, nelle sezioni successive, nel caso in cui occorre produrre una classifica su N=20 oggetti e fissiamo il numero totale di valutazioni che abbiamo a disposizione a W=8000. Dunque, a differenza di quanto effettuato nel caso di approccio a singolo stadio, qui, al variare dei parametri che regolano la struttura dei grafi (e cioè del grado dei grafi ai due stadi), viene impiegato sempre lo stesso numero totale di risorse totali. In questo modo, fissata la complessità dell'algoritmo, ovvero il totale di valutazioni che abbiamo a disposizione, sarà interessante studiare, fissato il grado del grafo ai due stadi (e cioè a parità di condizioni), il comportamento dell'algoritmo al variare delle risorse assegnate al primo piuttosto che al secondo stadio. Esattamente come fatto nello studio della versione a singolo stadio dell'algoritmo, anche qui valuteremo le prestazioni dell'algoritmo mediante simulazione e basandoci sulla probabilità di errore e sulla distorsione media. Studieremo, in particolare, sia la versione non pesata che quella pesata dell'algoritmo nel caso in cui il parametro di tolleranza dell'errore viene settato ad ϵ=0.04 e ϵ=0.08.

5.0.1 Studio dell'algoritmo LS

In questa sezione, come accennato, presenteremo l'algoritmo non pesato nella sua versione a due stadi, nel caso in cui occorre produrre una classifica su 20 oggetti e si ha a disposizione un pool totale di 8000 valutazioni. I parametri utilizzati per lo studio del modello sono 3: il numero di valutazioni assegnate al primo stadio, il grado del grafo regolare a catena chiusa utilizzato al primo stadio ed il grado del grafo a catena aperta al secondo stadio. Al fine di studiare il comportamento dell'algoritmo al variare dei 3 parametri, nei grafici che verranno riportati nel seguito, si è scelto di fissare il grado del grafo al secondo stadio. In questo modo è possibile riportare su un grafico bidimensionale il comportamento delle

prestazioni dell'algoritmo al variare del numero di valutazioni assegnate al primo stadio e del grado del grafo utilizzato al primo stadio. Nelle due sottosezioni seguenti studieremo, come detto, il caso ϵ=0.04 e ϵ=0.08.

Caso ϵ=0.04

I grafici riportati nelle figure 5.1., 5.2. e 5.3. presentano il comportamento della probabilità di errore al variare del numero di valutazioni assegnate al primo stadio e mantenendo fissato il grado del grafo al primo stadio, rispettivamente nel caso in cui il grado del grafo a catena aperta al secondo stadio è 2, 4 e 6.

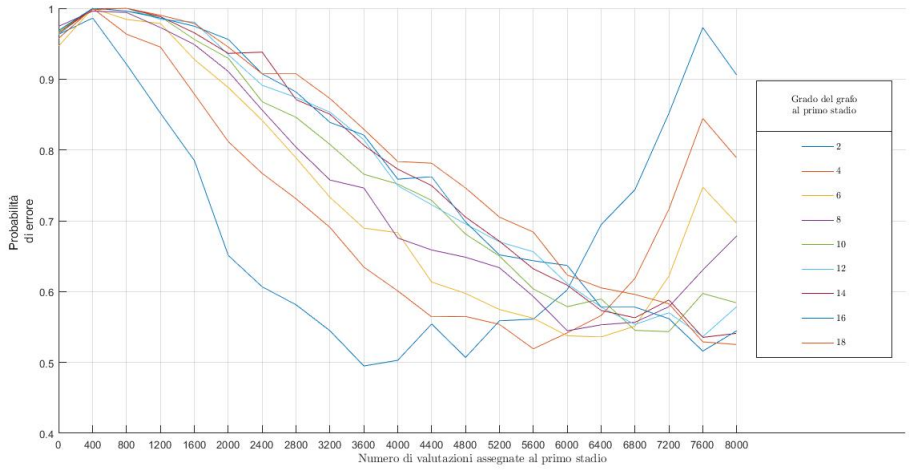


Figura 5.1. P_e al variare del numero di valutazioni assegnate al primo stadio e fissato il grado del grafo al primo stadio nel caso in cui il grafo al secondo stadio ha grado 2, $N = 20$, $W = 8000$ ed $\epsilon = 0.04$

Abbiamo studiato, in particolare, il comportamento dell'algoritmo per tutti i gradi pari ammissibili per il grafo al primo stadio e per i gradi 2, 4 e 6 per il grafo utilizzato nella seconda fase. Per ciascuna combinazione di questi parametri abbiamo studiato il comportamento della P_e al variare del numero di valutazioni assegnate al primo stadio dell'algoritmo. In particolare, si è scelto di assegnare inizialmente 0 valutazioni alla prima fase e, dunque, di effettuare, di fatto, solo il secondo stadio, nel quale vengono concentrate tutte le risorse. In questo caso iniziale si sta dunque applicando l'algoritmo descritto nel terzo capitolo, ovvero quello a singolo stadio, utilizzando un grafo a catena aperta con grado molto basso (2, 4 e poi 6). In seguito abbiamo deciso di aumentare di 400 in 400 il numero di risorse che assegniamo al primo stadio fino a quando ad esso non vengono assegnate tutte le 8000 valutazioni. In questo ultimo caso, poiché non vengono assegnate risorse al secondo stadio, applichiamo nuovamente l'algoritmo a singolo stadio, utilizzando però, questa volta, un grafo a catena chiusa.

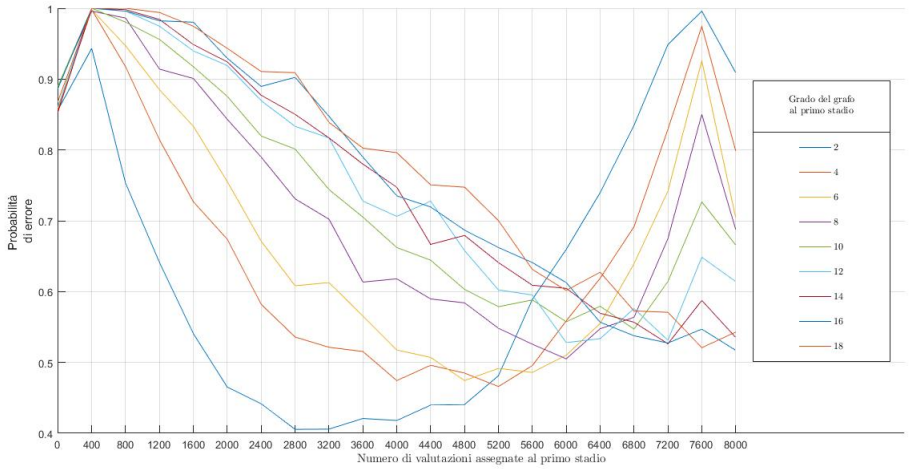


Figura 5.2. P_e al variare del numero di valutazioni assegnate al primo stadio e fissato il grado del grafo al primo stadio nel caso in cui il grafo al secondo stadio ha grado 4, $N = 20$, $W = 8000$ ed $\epsilon = 0.04$

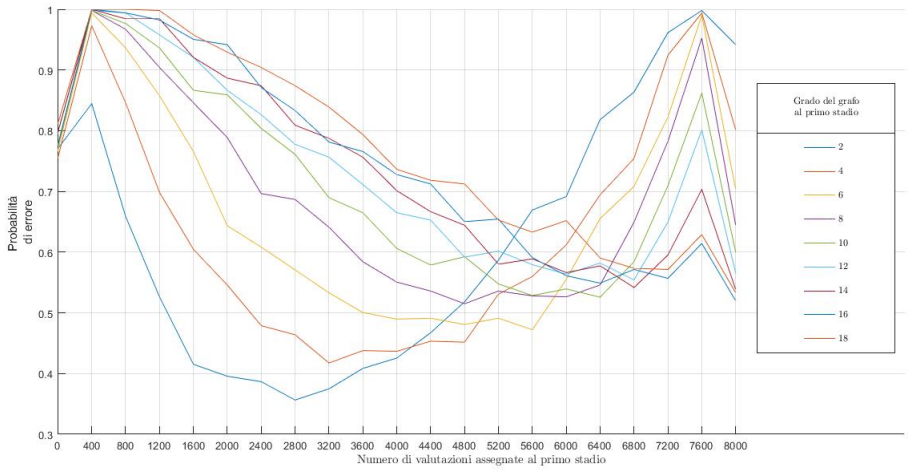


Figura 5.3. P_e al variare del numero di valutazioni assegnate al primo stadio e fissato il grado del grafo al primo stadio nel caso in cui il grafo al secondo stadio ha grado 6, $N = 20$, $W = 8000$ ed $\epsilon = 0.04$

Dalle figure considerate si osserva come la P_e sia, nel caso in cui non venga assegnata alcuna risorsa al primo stadio, molto simile ai valori ottenuti nel capitolo precedente mediante l'utilizzo di grafi a catena chiusa regolari di grado corrispondente. Piccole differenze nei risultati ottenuti possono essere dovute all'utilizzo di un grafo a catena aperta

e ad effetti imputabili alla simulazione statistica. Poi, a differenza di quanto ci saremmo aspettati di osservare, riscontriamo un iniziale e generale aumento della P_e che si nota quando vengono assegnate poche osservazioni al primo stadio e molte al secondo, ovvero nella parte sinistra dei grafici. Quindi la probabilità di errore, al crescere del numero di valutazioni assegnate al primo stadio, incomincia a scendere fino a raggiungere un minimo locale, per poi tornare a crescere. In modo speculare a quanto avviene quando decidiamo di assegnare 400 valutazioni al primo stadio e 7600 al secondo, osserviamo come le curve che riportano il comportamento della P_e continuino a salire fino a raggiungere un massimo locale quando assegniamo 7600 valutazioni al primo stadio e 400 al secondo. Dopo aver raggiunto questo massimo le curve tornano infatti a scendere per il caso in cui assegniamo l'intero insieme di valutazioni al primo stadio. Il comportamento della P_e ci suggerisce dunque che è preferibile assegnare tutte le valutazioni al primo stadio ed eseguire l'algoritmo a singolo stadio rispetto ad assegnare la grande maggioranza di esse al primo stadio e poi cercare di migliorare la classifica prodotta sulla base delle poche valutazioni fornite al secondo stadio.

I due fenomeni descritti, che sono evidenti nelle figure 5.1., 5.2. e 5.3. nella parte iniziale e finale delle curve, possono essere motivati dal fatto che, utilizzando la versione non pesata dell'algoritmo, le probabilità assegnate agli archi al primo ed al secondo stadio non tengono conto del numero di valutazioni che sono state utilizzate per ricavarle. Ricordiamo come la probabilità in questione sia legata ad un arco e rappresenti la probabilità che il primo oggetto dell'arco stesso abbia una qualità che è maggiore rispetto al secondo oggetto e che, dunque, venga preferito nelle valutazioni espresse. In questo modo assegniamo la stessa credibilità a valori di probabilità che sono stati ricavati sulla base di poche valutazioni e a quelli che sono ricavati sulla base di molte di esse. Il problema in questione risulta infatti più evidente nella parte iniziale e finale delle curve, dove è riscontrabile una grande differenza nel numero di valutazioni che vengono assegnate ai due stadi. Osserviamo infatti che, se assumiamo di porci in un caso intermedio dove consideriamo un grafo al primo stadio di grado 8 ed un grafo al secondo stadio di grado 4, quando assegniamo 400 valutazioni al primo stadio e le rimanenti 7600 al secondo, vale:

$$k_1 = \left\lfloor \frac{\#\text{ valutazioni assegnate al primo stadio}}{\#\text{ archi al primo stadio}} \right\rfloor = 5$$

$$k_2 = \left\lfloor \frac{\#\text{ valutazioni assegnate al secondo stadio}}{\#\text{ archi al secondo stadio}} \right\rfloor = 205$$

dove k_1 e k_2 rappresentano rispettivamente il numero di valutazioni per arco assegnate al primo ed al secondo stadio.

Tale fenomeno è riscontrabile in modo ancora più netto nelle figure 5.4., 5.5. e 5.6. dove viene presentato il comportamento della D al variare del numero di valutazioni totali assegnate al primo stadio, fissato il grado del grafo al primo stadio rispettivamente nel caso in cui il grado del grafo al secondo stadio è fissato a 2, 4 e 6.

Vedremo come sarà possibile risolvere il problema descritto utilizzando una versione pesata dell'algoritmo. Attraverso l'utilizzo dei pesi, come vedremo, sarà infatti possibile tenere traccia del numero di valutazioni che vengono assegnate ad un determinato arco al fine di ricavare il valore di probabilità ad esso associato. In questo modo attribuiremo ad

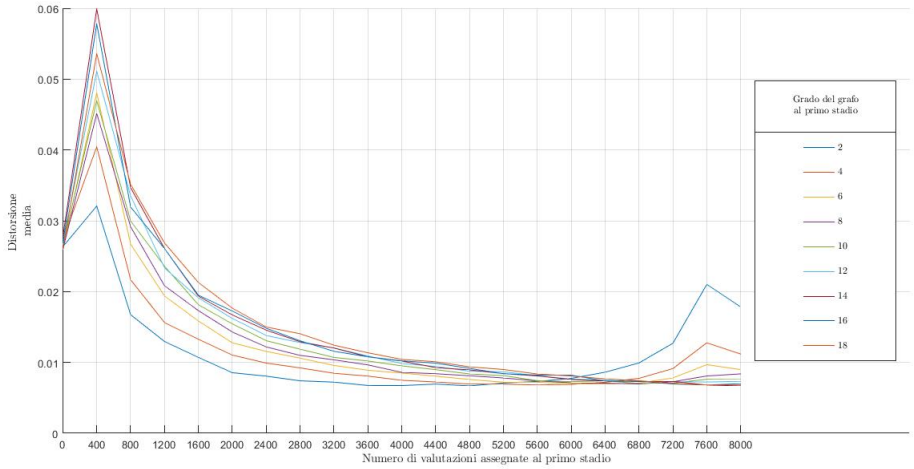


Figura 5.4. D al variare del numero di valutazioni assegnate al primo stadio e fissato il grado del grafo al primo stadio nel caso in cui il grafo al secondo stadio ha grado 2, $N = 20$, $W = 8000$ ed $\epsilon = 0.04$

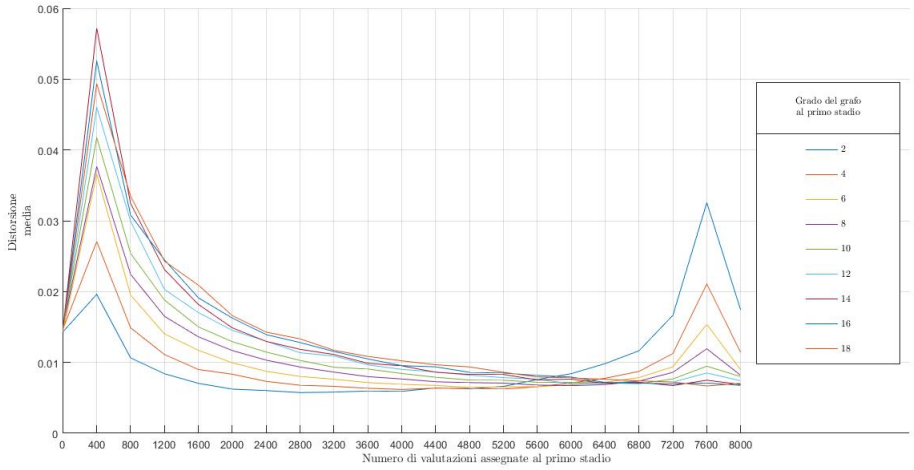


Figura 5.5. D al variare del numero di valutazioni assegnate al primo stadio e fissato il grado del grafo al primo stadio nel caso in cui il grafo al secondo stadio ha grado 4, $N = 20$, $W = 8000$ ed $\epsilon = 0.04$

ogni valore di probabilità la credibilità che gli spetta e l'algoritmo trarrà grossi benefici da questo fatto.

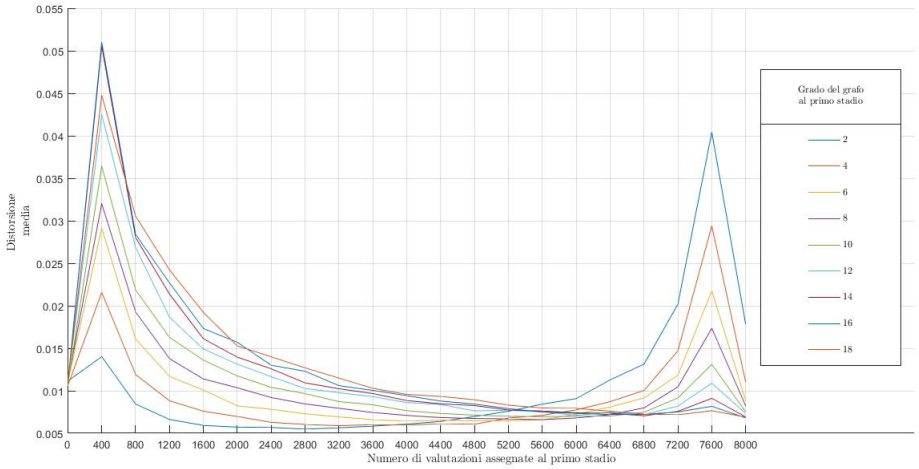


Figura 5.6. D al variare del numero di valutazioni assegnate al primo stadio e fissato il grado del grafo al primo stadio nel caso in cui il grafo al secondo stadio ha grado 6, $N = 20$, $W = 8000$ ed $\epsilon = 0.04$

Caso ϵ=0.08

In questa sezione riproponiamo esattamente l'esperimento descritto nella sezione precedente nel caso in cui ϵ=0.08. Studiamo cioè, fissato tale parametro che regola la tolleranza che abbiamo sugli errori, il comportamento della P_e al variare del numero di valutazioni assegnate al primo stadio e fissato il grado dei due grafi ai due stadi. In modo del tutto analogo a quanto mostrato nella sezione precedente, si ottengono allora le figure 5.7., 5.8. e 5.9.. Le curve ottenute, le quali presentano il comportamento della probabilità di errore, risultano, per qualche motivo, essere più stabili e questo permette di cogliere, in modo ancora più chiaro ed evidente, quanto è stato possibile osservare per il caso di $\epsilon = 0.04$.

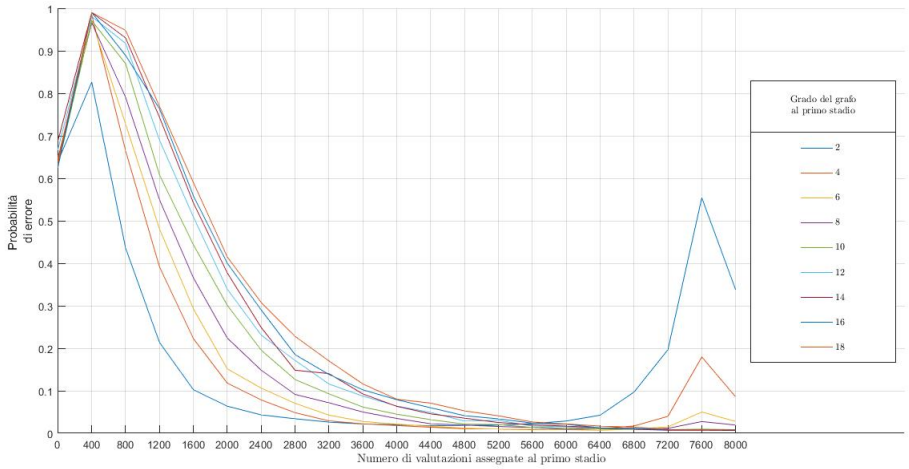


Figura 5.7. P_e al variare del numero di valutazioni assegnate al primo stadio e fissato il grado del grafo al primo stadio nel caso in cui il grafo al secondo stadio ha grado 2, $N = 20$, $W = 8000$ ed $\epsilon = 0.08$

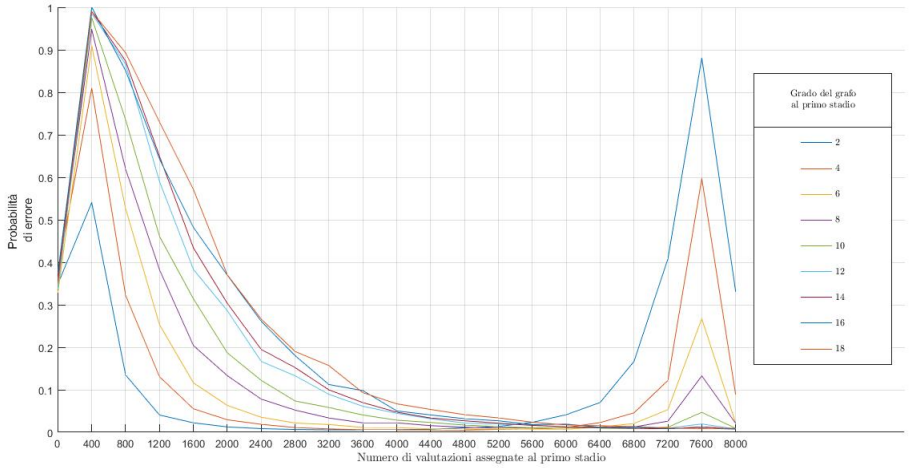


Figura 5.8. P_e al variare del numero di valutazioni assegnate al primo stadio e fissato il grado del grafo al primo stadio nel caso in cui il grafo al secondo stadio ha grado 4, $N = 20$, $W = 8000$ ed $\epsilon = 0.08$

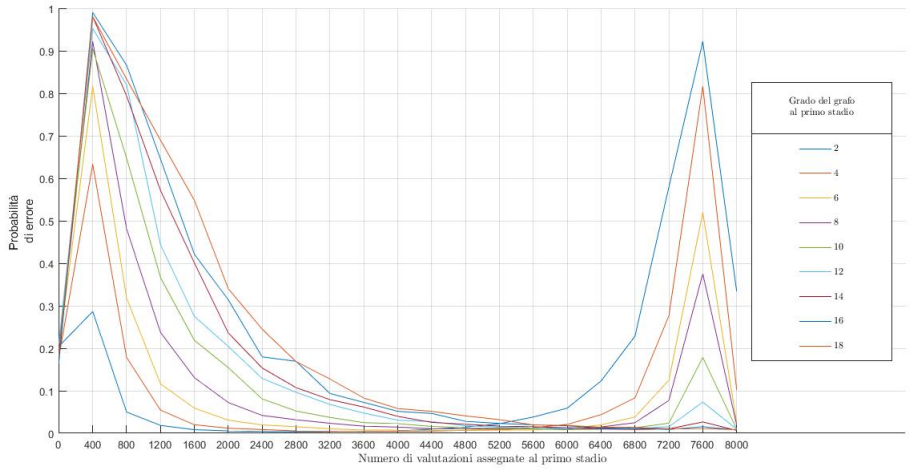


Figura 5.9. P_e al variare del numero di valutazioni assegnate al primo stadio e fissato il grado del grafo al primo stadio nel caso in cui il grafo al secondo stadio ha grado 6, $N = 20$, $W = 8000$ ed $\epsilon = 0.08$

5.0.2 Studio dell'algoritmo WLS

Non soddisfatti delle prestazioni ottenute mediante l'utilizzo dell'algoritmo a due stadi nella sua versione non pesata, al fine di tentare di risolvere il problema intrinseco all'algoritmo evidenziato nella sezione precedente, si è deciso di implementare la versione pesata dell'algoritmo a due stadi. Il peso che viene associato a ciascun arco riesce, infatti, come vedremo nel dettaglio in seguito, a risolvere questo problema.

Per comprendere meglio il problema dell'algoritmo nella versione non pesata e la soluzione che è stata adottata per risolverlo è conveniente vedere il procedimento presentato nella sezione precedente nel modo seguente. Se assumiamo di avere già a disposizione all'istante iniziale l'ordinamento prodotto al primo stadio (cosa che nella realtà non abbiamo), allora, di fatto, si sta applicando l'algoritmo a singolo stadio descritto nel capitolo 3, dove il grafo utilizzato è quello generato dalla sovrapposizione dei grafi ai due stadi. Dato che i nodi dei grafi ai due stadi sono esattamente gli stessi è infatti possibile costruire un grafo che ha come nodi i nodi dei due grafi precedenti e come archi l'unione degli insiemi di archi presenti nei due grafi ai due stadi. Osservando il problema da questa angolazione, la causa del mal funzionamento dell'algoritmo a due stadi non pesato appare evidente. È chiaro infatti come non venga assegnata la stessa quantità di valutazioni a tutti gli archi e come, in particolare, si abbiamo 3 classi di archi: quelli che ricevono solo le valutazioni dal primo stadio, quelli che le ricevono solo dal secondo e quelle che le ricevono da entrambi. Osserviamo come in alcuni casi la differenza nel numero di valutazioni che vengono fornite agli archi appartenenti a classi diverse può anche essere decisamente significativa.

Mentre, allora, nella versione non pesata dell'algoritmo, la matrice $\tilde{\mathbf{H}}$, descritta nel

capitolo 3, è definita come proporzionale alle matrice seguente:

$$\hat{H}(i,j) = \begin{cases} 1, & \text{se l'arco } (i,j) \in \text{al grafo al I e/o II stadio} \\ 0, & \text{altrove} \end{cases}$$

la matrice $\tilde{\mathbf{H}}$ associata all'algoritmo nella versione pesata risulta essere proporzionale alla matrice:

$$\hat{H}(i,j) = \begin{cases} \frac{k_1}{p_{i,j}^{(1)}\left(1-p_{i,j}^{(1)}\right)\left(\frac{dF^{-1}(p)}{dp}\Big|_{p=p_{i,j}^{(1)}}\right)^2}, & \text{se l'arco } (i,j) \in \text{solo al grafo al I stadio} \\ \frac{k_2}{p_{i,j}^{(2)}\left(1-p_{i,j}^{(2)}\right)\left(\frac{dF^{-1}(p)}{dp}\Big|_{p=p_{i,j}^{(2)}}\right)^2}, & \text{se l'arco } (i,j) \in \text{solo al grafo al II stadio} \\ \frac{k_1+k_2}{p_{i,j}^{(\text{fin})}\left(1-p_{i,j}^{(\text{fin})}\right)\left(\frac{dF^{-1}(p)}{dp}\Big|_{p=p_{i,j}^{(\text{fin})}}\right)^2}, & \text{se l'arco } (i,j) \in \text{al grafo al I e II stadio} \end{cases}$$

In questa formula k_1 e k_2 indicano il numero di valutazioni assegnate all'arco (i,j) rispettivamente al primo ed al secondo stadio ed F è, in generale, la funzione tale che: $F(q_i - q_j) = F(d_{i,j}) = p_{i,j}$. I valori $p_{i,j}^{(1)}$ e $p_{i,j}^{(2)}$ rappresentano, invece, la componente associata all'arco ad (i,j) del vettore delle probabilità quando esso appartiene rispettivamente solo al primo o al secondo stadio. Se, invece, l'arco (i,j) appartiene contemporaneamente al primo ed al secondo stadio viene allora utilizzato il valore di probabilità:

$$p_{i,j}^{(\text{fin})} = \frac{k_1 p_{i,j}^{(1)} + k_2 p_{i,j}^{(2)}}{k_1 + k_2}.$$

Come anticipato, consideriamo quindi 3 classi di archi: quelli che appartengono solo al primo stadio e usufruiscono di k_1 valutazioni, quelli che appartengono solo al secondo stadio ed usufruiscono di k_2 valutazioni e quelli che sono presenti sia al primo che al secondo stadio ed usufruiscono dunque di $k_1 + k_2$ valutazioni. Mediante questa particolare scelta della matrice $\tilde{\mathbf{H}}$ (che è stata motivata nel capitolo 3), teniamo dunque conto del fatto che non tutte le probabilità associate agli archi sono ricavate sulla base dello stesso numero di valutazioni e che, dunque, non sono tutte affidabili allo stesso modo. Anche in questo caso, nel seguito presenteremo le prestazioni dell'algoritmo per i valori di $\epsilon = 0.04$ ed $\epsilon = 0.08$.

Caso ϵ=0.04

I miglioramenti apportati da questo tipo di approccio sono immediatamente riscontrabili nei grafici presentati nelle figure 5.10., 5.11. e 5.12., dove viene mostrato il comportamento della probabilità di errore al variare del numero di valutazioni assegnate alla prima fase e tenendo fisso il grado dei grafi al primo ed al secondo stadio. In particolare ciascuna curva presente nei tre grafici rappresenta il comportamento della P_e per un dato valore del grado del grafo al primo stadio ed in ciascuna figura è fissato il grado del grafo al secondo stadio rispettivamente a 2, 4 e 6.

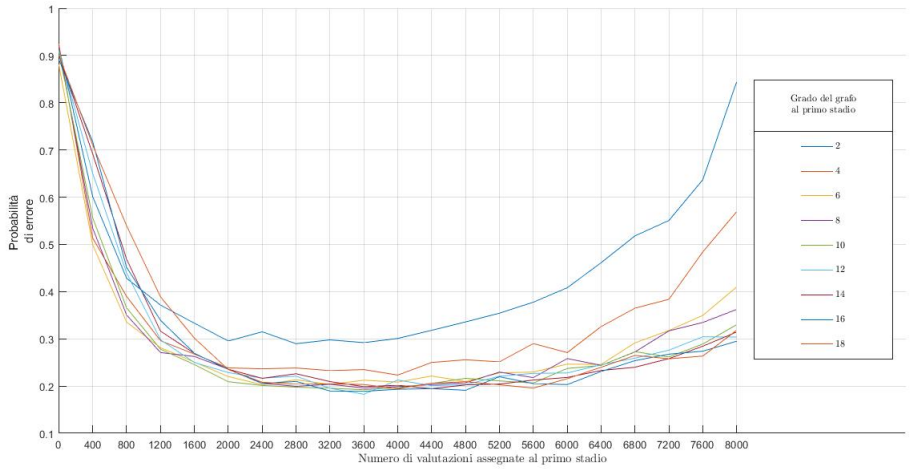


Figura 5.10. P_e al variare del numero di valutazioni assegnate al primo stadio e fissato il grado del grafo al primo stadio nel caso in cui il grafo al secondo stadio ha grado 2, $N = 20$, $W = 8000$ ed $\epsilon = 0.04$

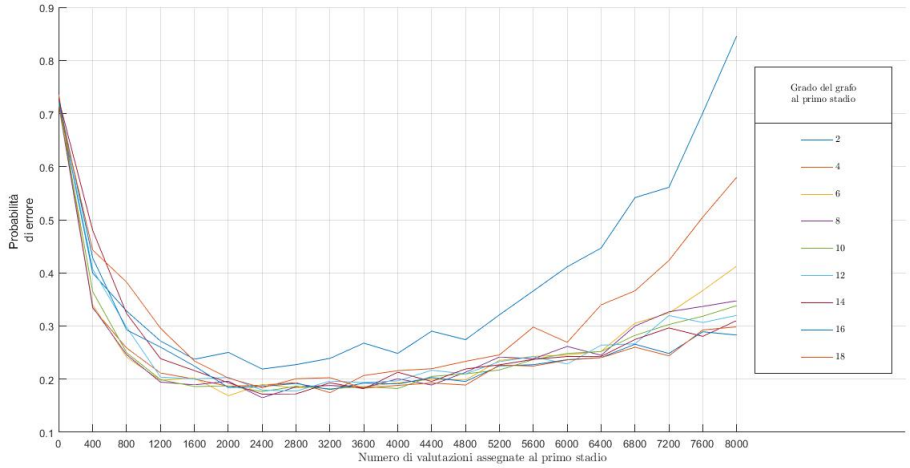


Figura 5.11. P_e al variare del numero di valutazioni assegnate al primo stadio e fissato il grado del grafo al primo stadio nel caso in cui il grafo al secondo stadio ha grado 4, $N = 20$, $W = 8000$ ed $\epsilon = 0.04$

Osservando tali figure è immediatamente possibile riscontrare come i due fenomeni evidenziati alle estremità dei grafici ricavati mediante simulazione dell'algoritmo a due stadi nella sua versione non pesata, descritti nella sezione precedente, non siano più presenti. Era infatti stato osservato come assegnare un piccolo numero di valutazioni al primo stadio

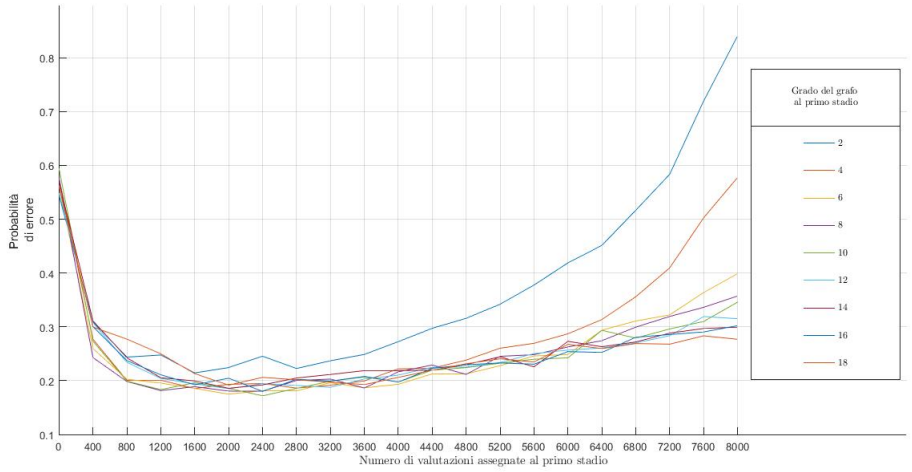


Figura 5.12. P_e al variare del numero di valutazioni assegnate al primo stadio e fissato il grado del grafo al primo stadio nel caso in cui il grafo al secondo stadio ha grado 6, $N = 20$, $W = 8000$ ed $\epsilon = 0.04$

e la grande maggioranza di esse al secondo, così come assegnare la grande maggioranza delle osservazioni al primo stadio ed una piccola parte al secondo, risulti sconveniente rispetto a concentrare l'intero insieme di valutazioni in un singolo stadio. È infatti possibile notare come, applicando la versione a due stadi dell'algoritmo pesato, risulti in generale conveniente assegnare, in modo più o meno parsimonioso, qualche risorsa al primo stadio, in modo da poter sfruttare le informazioni da esso provenienti per scegliere in modo più mirato le coppie di oggetti da confrontare allo stadio successivo. Essendo infatti, in tutte e 3 le figure, il grado del grafo al secondo stadio comunque basso, la probabilità di errore iniziale, cioè quella che si ottiene quando tutte le valutazioni sono assegnate al secondo stadio, è piuttosto alta. Poi si osserva che essa incomincia a calare con l'aumentare del numero di valutazioni assegnate al primo stadio fino a quando l'algoritmo non trae più giovamento nell'aggiungere ulteriori risorse a tale stadio. Raggiunto questo punto, osserviamo infatti come la probabilità di errore cominci a crescere, in modo più o meno repentino a seconda del grado dei grafi ai due stadi, fino a quando non si raggiunge un massimo, eventualmente locale, nella situazione estrema in cui tutte le valutazioni vengono assegnate al primo stadio. Ricordiamo come in tal caso viene applicato l'algoritmo a singolo stadio pesato descritto nel capitolo 3.

Osserviamo inoltre come, una volta fissati i parametri che regolano il grado dei grafi ai due stadi ed il numero di valutazioni totali che ci vengono fornite, sia presente un chiaro trade-off nella scelta del numero di risorse, in termini di valutazioni, da assegnare ai due stadi. Al crescere del numero di valutazioni che vengono assegnate al primo stadio, infatti, le informazioni che vengono fornite al secondo stadio saranno più accurate, ma tale stadio avrà a disposizione sempre meno risorse per tentare di migliorare la classifica che gli viene fornita in input. Al decrescere del numero di valutazioni assegnate al primo

stadio, viceversa, il secondo stadio, nonostante abbia a disposizione un numero crescente di valutazioni, non riceverà una classifica sufficientemente accurata tale da permettere all'algoritmo di selezionare in modo opportuno le coppie di oggetti da confrontare per aggiornare in modo conveniente la classifica finale. Osservando le figure 5.10., 5.11. e 5.12. si riscontra che, fissato il grado dei due grafi ai due stadi, è sempre conveniente assegnare un certo numero di valutazioni ad entrambi gli stadi in quantità che varia a seconda del caso. Osserviamo inoltre come tutte e tre le figure, che mostrano il comportamento della P_e fissato il grado del grafo al secondo stadio rispettivamente a 2, 4 e 6, presentino un comportamento simile della probabilità di errore e che il valore più basso di tale misura di prestazione è comparabile nei 3 casi. È possibile però notare come tale valore ottimale venga raggiunto per valori diversi del numero di valutazioni assegnate al primo stadio.

Considerazioni analoghe possono essere fatte osservando le figure 5.13., 5.14. e 5.15. che mostrano le prestazioni dell'esperimento appena presentato, ma in termini di distorsione media.

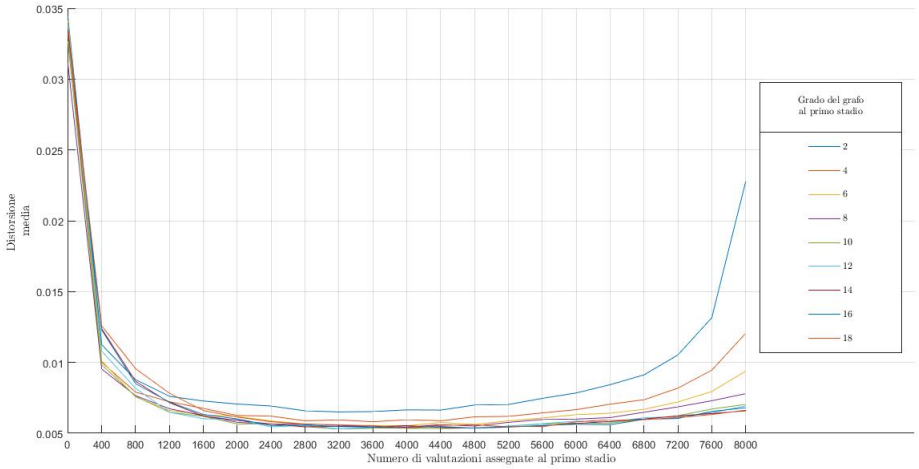


Figura 5.13. D al variare del numero di valutazioni assegnate al primo stadio e fissato il grado del grafo al primo stadio nel caso in cui il grafo al secondo stadio ha grado 2, $N = 20$, $W = 8000$ ed $\epsilon = 0.04$

In particolare, nella figura 5.13., dove il grado del grafo al secondo stadio è 2, è possibile osservare come la distorsione media nella parte iniziale del grafico sia molto maggiore rispetto all'errore che si osserva nelle due figure successive (5.14. e 5.15.) dove il grado del grafo al secondo stadio è, seppure di poco, più alto. Osserviamo invece come, al crescere del numero di valutazioni assegnate alla prima fase, la distorsione media rimanga sempre bassa. Come era lecito aspettarsi, in base al comportamento già evidenziato della P_e, la D torna poi a crescere nella parte finale del grafico fino a raggiungere un massimo quando tutte le 8000 valutazioni sono assegnate al primo stadio.

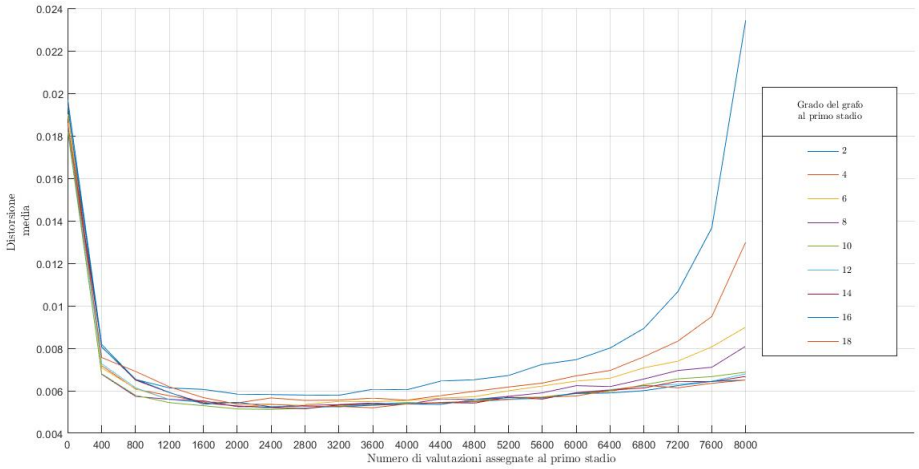


Figura 5.14. D al variare del numero di valutazioni assegnate al primo stadio e fissato il grado del grafo al primo stadio nel caso in cui il grafo al secondo stadio ha grado 4, $N = 20$, $W = 8000$ ed $\epsilon = 0.04$

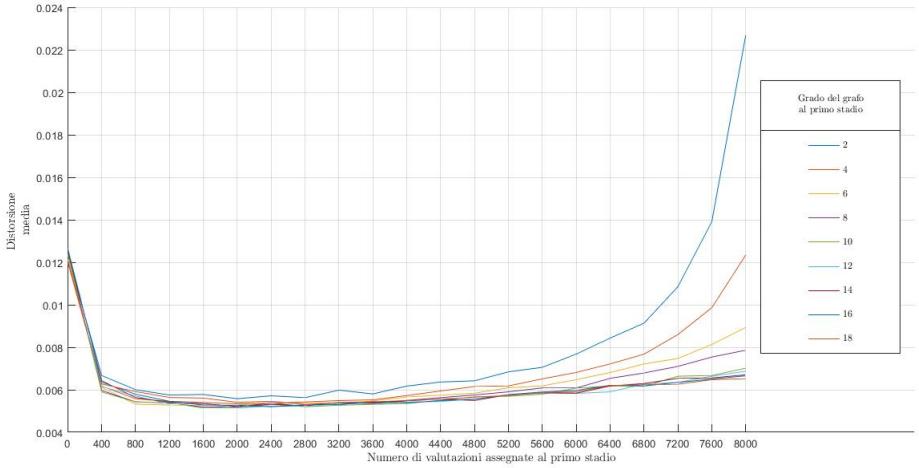


Figura 5.15. D al variare del numero di valutazioni assegnate al primo stadio e fissato il grado del grafo al primo stadio nel caso in cui il grafo al secondo stadio ha grado 6, $N = 20$, $W = 8000$ ed $\epsilon = 0.04$

Caso ϵ=0.08

Riportiamo infine, nelle figure 5.16., 5.17. e 5.18., il comportamento della P_e al variare del numero di valutazioni assegnate al primo stadio e tenendo fissi gli altri parametri (cioè il grado dei grafi ai due stadi). In tali figure è possibile osservare in modo ancora più

netto i fenomeni decritti nella sezione precedente. Questo a causa del fatto che le curve prodotte mediante simulazione, fissando il parametro ϵ di tolleranza dell'errore a 0.08, sono maggiormente stabili ed evidenziano in modo chiaro il comportamento della P_e.

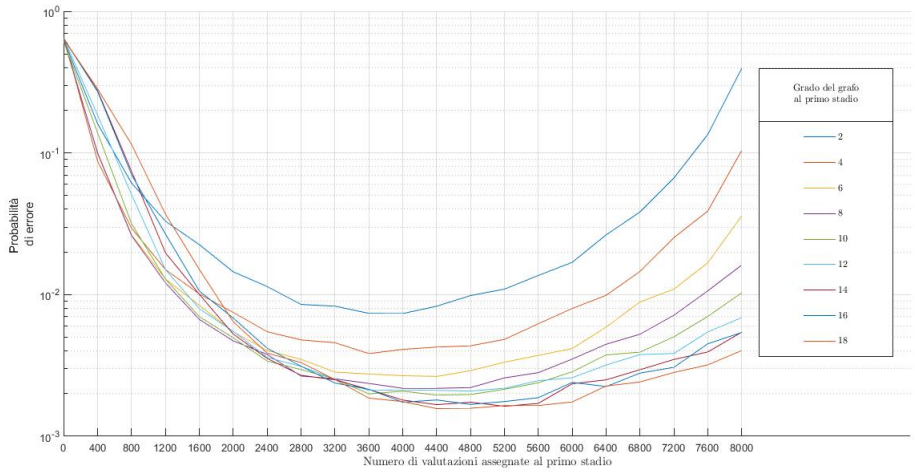


Figura 5.16. P_e al variare del numero di valutazioni assegnate al primo stadio e fissato il grado del grafo al primo stadio nel caso in cui il grafo al secondo stadio ha grado 2, $N = 20$, $W = 8000$ ed $\epsilon = 0.08$

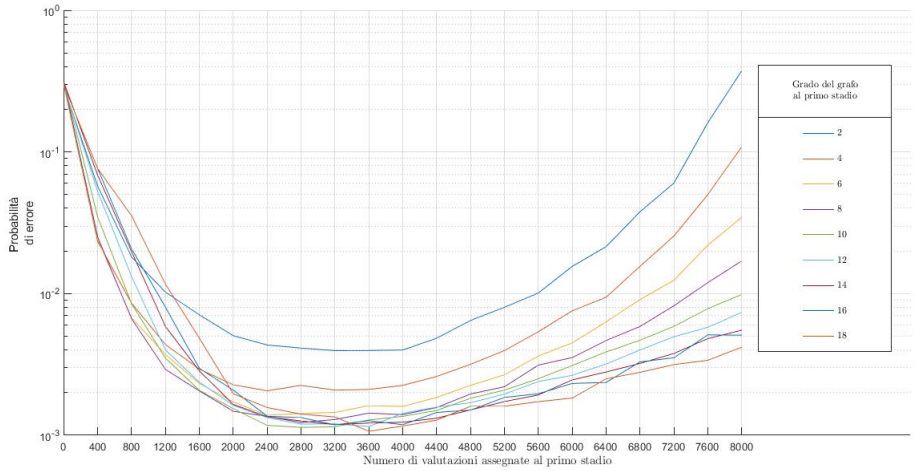


Figura 5.17. P_e al variare del numero di valutazioni assegnate al primo stadio e fissato il grado del grafo al primo stadio nel caso in cui il grafo al secondo stadio ha grado 4, $N = 20$, $W = 8000$ ed $\epsilon = 0.08$

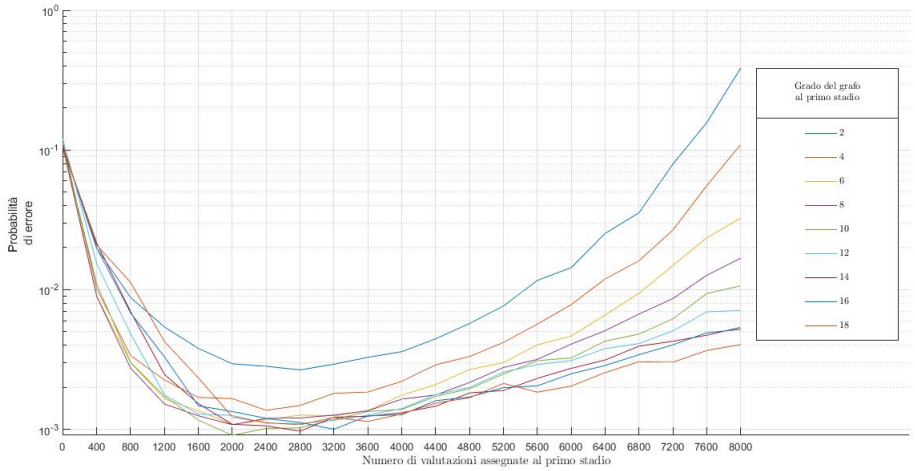


Figura 5.18. P_e al variare del numero di valutazioni assegnate al primo stadio e fissato il grado del grafo al primo stadio nel caso in cui il grafo al secondo stadio ha grado 6, $N = 20$, $W = 8000$ ed $\epsilon = 0.08$

Osservando tali grafici è inoltre possibile rilevare come i benefici, in termini di P_e, che si traggono nell'utilizzare l'algoritmo a due stadi rispetto a quello a singolo stadio siano più evidenti quando il grado dei grafi al primo e secondo stadio è più basso. In situazioni di questo tipo, infatti, l'algoritmo a due stadi, a parità di risorse utilizzate, ottiene prestazioni migliori di almeno un ordine di grandezza rispetto al caso a singolo stadio. Il vantaggio, in termini di probabilità di errore, è meno netto quando associamo all'algoritmo al primo stadio un grafo con grado sufficientemente alto, ma il miglioramento che si osserva è comunque significativo.

Appendice A

Codici MATLAB utilizzati nelle simulazioni

Riportiamo, in appendice, i codici MATLAB prodotti ed utilizzati per la simulazione degli algoritmi proposti al fine di misurarne le prestazioni. Presenteremo dunque, nel seguito, i codici relativi all'algoritmo a singolo stadio ed all'algoritmo a due stadi, abili a simulare sia le versioni LS che quelle WLS.

A.1 Codici per la simulazione dell'algoritmo a singolo stadio

Riportiamo, di seguito, la funzione main utilizzata per ricavare, mediante la chiamata di opportune funzioni, le matrici perr e Dmean che rappresentano, rispettivamente, il valore della P_e e della D al variare del grado del grafo scelto e del numero di valutazioni utilizzate. Settando in modo opportuno i parametri Graph_type, Algoritmo e max_th, in particolare, è possibile testare le prestazioni dell'algoritmo al variare del tipo di struttura scelta per il grafo, di algoritmo e di tolleranza che assumiamo sugli errori sulle classifiche che generiamo.

```
1  clear
2  close all
3
4  % Object set
5  N=20;  % number of objects
6  W=10000; % maximum number of evaluations
7
8  % initialisation of data structures to measure prestazioni:
9  dim_1=floor(W/500);
10 dim_2=floor((N-1)/2);
11 perr=zeros(dim_1,dim_2);
12 Dmean=zeros(dim_1,dim_2);
13
```

```
14 sigma = 0.4;  % noise stddev
15
16 %Objects:
17 Qualities_distribution = 1; % 0=Equally spaced [0,1); 1=Uniform [0,1]
18
19 % Input qualities:
20 Qualities_ordering =2; % 1 = ordered (Genie Aided); 2 = random
21
22 % Graph type:
23 Graph_type = 3;   % 1 = chain;  2 = closed chain;  3 = connected ...
       regular random graph;  4 = multiple chain (for 2, 4 or 6 degree)
24
25 % Algorithm:
26 Algoritmo=1; % 0 = LS; 1 = WLS
27
28 max_th = 0.08;
29
30 switch Qualities_distribution
31     case 0
32         q = (0:1/N:1-1/N)';   % equally spaced objects
33     case 1
34         q = sort(rand(N,1));  % uniform
35 end
36
37 [q_sort, ranking] = sort(q);
38
39 switch Qualities_ordering
40     case 1
41         q1 = q;
42     case 2
43         q1 = q(randperm(N));
44 end
45
46
47 cont_1=0;
48 for t = 500:500:W          %t = # evaluations used
49     cont_1=cont_1+1;
50     cont_2=0;
51     for p = 2:2:N-1        %p = graph degree
52         cont_2=cont_2+1;
53
54         K = floor(t/((N*p)/2));     %K = # evaluation for edge
55
56         Graph_degree = p;
57
58         max_errors = 200;
59         max_runs = 1000000;
60
61         errors = zeros(1,1);
62         Distortion=zeros(max_runs,1);
63
64         for s=1:1:max_runs
65
```

```
66              [Edges] = CreateGraph(N,Graph_degree, Graph_type);
67              switch  Algoritmo
68                  case 0
69                      [ordering(:,1)] = ...
                            random_graph_linear(q1,N,K,sigma,Edges);
70                  case 1
71                      [ordering(:,1)] = ...
                            random_graph_linear_weighted(q1,N,K,sigma,Edges);
72              end
73
74              % evaluation of the algorithm prestazioni:
75              absdiff = abs(q-q1(ordering(:,1)));
76              max_diff= max(absdiff);
77              Distortion(s) = sum(absdiff)/N;
78              if (max_diff>max_th)
79                  errors = errors+1;
80              end
81              if(errors≥ max_errors)  % stop condition
82                  break;
83              end
84          end
85          perr(cont_1,cont_2) = errors/s;
86          Dmean(cont_1,cont_2) = sum(Distortion)/s;
87          C = ((K*N*p)/2);
88          [t p C perr(cont_1,cont_2) Dmean(cont_1,cont_2)]
89      end
90  end
```

La funzione CreateGraph viene invocata dal main quando è necessario generare il grafo. Poiché l'insieme dei nodi è fissato, quando occorre costruire il grafo ciò che resta da definire è l'insieme degli archi indiretti che mette in relazione i nodi. Questo è il compito svolto da questa funzione, la quale prende in input il numero di nodi, il grado e la tipologia del grafo stesso e restituisce, in output, una matrice di dimensione # archi $\times$ 2, che contiene, in ogni sua riga, un arco indiretto definito mediante una coppia di nodi. La funzione CreateGraph permette dunque, al variare del parametro di ingresso, di generare un grafo a catena aperta, uno a catena chiusa, uno regolare, connesso e random o uno multi-corona. Permette inoltre di generare grafi con grado desiderato.

```
1   function [Edges] = CreateGraph(N_nodes, degree, graphtype)
2
3   switch graphtype
4       case 1 % open chain
5           deg = floor(degree/2);
6           indices = (1:1:N_nodes)';
7
8           Edges = [];
9           for i=1:1:N_nodes-deg
10              neighbors = indices(i+1:1:i+deg);
11              edges = [indices(i)*ones(length(neighbors),1),neighbors];
12              Edges = [Edges; edges];
13          end
```

```
14
15          for i=N_nodes-deg+1:1:N_nodes-1
16              neighbors = indices(i+1:1:N_nodes);
17              edges = [indices(i)*ones(length(neighbors),1),neighbors];
18              Edges = [Edges; edges];
19          end
20
21      case 2 % closed chain
22          deg = degree/2;
23          indices = (1:1:N_nodes)';
24
25          Edges = [];
26          for i=1:1:N_nodes-deg
27              neighbors = indices(i+1:1:i+deg);
28              edges = [indices(i)*ones(length(neighbors),1),neighbors];
29              Edges = [Edges; edges];
30          end
31
32          cont=0;
33          for i=N_nodes-deg+1:1:N_nodes
34              cont=cont+1;
35              neighbors = indices(i+1:1:N_nodes);
36              edges = [indices(i)*ones(length(neighbors),1),neighbors];
37              Edges = [Edges; edges];
38              neighbors = indices(1:1:cont);
39              edges = [indices(i)*ones(cont,1),neighbors];
40              Edges = [Edges; edges];
41          end
42
43      case 3 % random
44          flag_regolare = 0;
45          while flag_regolare==0
46              Edges = [];
47
48              % crown graph construction:
49              for i = 1:1:N_nodes-1
50                  edge = [i,i+1];
51                  Edges = [Edges; edge];
52              end
53              edge = [N_nodes, 1];
54              Edges = [Edges; edge];
55
56              % generation of the remaining edges:
57              cont = zeros(N_nodes,1);
58              if(degree>2)
59                  % first node case:
60                  neighbors = [];
61                  while cont(1)+2<degree
62                      ind = randi([3,19]);
63                      flag=0;
64                      if ¬isempty(neighbors)
65                          for m = 1:1:length(neighbors)
66                              if ind==neighbors(m)
```

```
                                flag=1;
                            end
                        end
                    end
                    if flag==0
                        neighbors = [neighbors; ind];
                        cont(1) = cont(1)+1;
                        cont(ind) = cont(ind)+1;
                    end
                end
                edges_new = [ones(length(neighbors),1),neighbors];
                Edges = [Edges; edges_new];

                % other nodes case:
                for i = 2:1:N_nodes-2
                    vett = i+2:1:N_nodes;
                    vett_disp = [];
                    for j = 1:1:length(vett)
                        if cont(vett(j))<degree-2
                            vett_disp = [vett_disp; vett(j)];
                        end
                    end
                    ind_neighbors = [];
                    while cont(i)+2<degree
                        if  length(vett_disp)≥degree-cont(i)-2
                            ind = randi(length(vett_disp));
                            flag=0;
                            if ¬isempty(ind_neighbors)
                                for m = 1:1:length(ind_neighbors)
                                    if ind==ind_neighbors(m)
                                        flag=1;
                                    end
                                end
                            end
                            if flag==0
                                ind_neighbors = [ind_neighbors; ind];
                                cont(i) = cont(i)+1;
                            end
                        else
                            break;
                        end
                    end
                    neighbors = vett_disp(ind_neighbors);
                    edges_new = [i*ones(length(neighbors),1),neighbors];
                    Edges = [Edges; edges_new];

                    % counters updating:
                    for k = 1:1:length(neighbors)
                        cont(neighbors(k)) = cont(neighbors(k))+1;
                    end
                end
            end
            if length(Edges)==N_nodes*degree/2
```

```
                flag_regolare=1;
            end
        end

        case 4 % multiple crown
        Edges = [];

        % crown graph construction:
        for i = 1:1:N_nodes-1
            edge = [i,i+1];
            Edges = [Edges; edge];
        end
        edge = [N_nodes, 1];
        Edges = [Edges; edge];

        if degree≥4
            g = 7;
            Edges = [Edges; 20, 7];
            for i=1:1:N_nodes-2
                edge = [mod(i*g,20), mod((i+1)*g,20)];
                Edges = [Edges; edge];
            end
            Edges = [Edges; 13, 20];
        end

        if degree≥6
            g = 3;
            Edges = [Edges; 20, 3];
            for i=1:1:N_nodes-2
                edge = [mod(i*g,20), mod((i+1)*g,20)];
                Edges = [Edges; edge];
            end
            Edges = [Edges; 17, 20];
        end
end
end
```

La funzione random_graph_linear, riportata di seguito, viene invocata dal main al fine di produrre l'ordinamento finale sulla base delle valutazioni che vengono simulate, all'interno della funzione stessa, sugli archi definiti nella funzione CreateGraph. Tale funzione riceve infatti, in input, le qualità reali degli oggetti q, la struttura del grafo, ovvero il numero di nodi N e l'insieme degli archi Edges, il numero di valutazioni k di cui usufruisce ciascun arco e sigma, ovvero un parametro che regola la variabilità nell'insieme di valutazioni simulato per ogni arco. La funzione in questione si occupa, in prima battuta, di simulare le valutazioni prodotte dai lavoratori sulle coppie di oggetti ricevute in input a partire dalle qualità reali presenti in q e seguendo il modello di Thurstone, con $\sigma = 0.4$. Quindi la funzione popola le metrici H, dalla quale è possibile ricavare la matrice M, e, sulla base delle valutazioni simulate, D_est. A partire da tali matrici è possibile ricavare una stima delle qualità degli oggetti, dalla quale, per la funzione, è banale ricavare e restituire l'ordinamento stimato.

```
function [ordering] = random_graph_linear(q,N,K,sigma, Edges)

N_edges = size(Edges,1);

% matrix initialisation:
H = zeros(N,N);
D_est= zeros(N,N);

for edge=1:1:N_edges
    h = Edges(edge,1);
    l = Edges(edge,2);
    Δ = q(h)-q(l);
    y = Δ+randn(K,1)*sigma;
    % probability:
    p = mean(y>0);

    if(p==0)
        p=0.01;
    end
    if(p==1)
        p=0.99;
    end

    H(h,l) = 1;
    H(l,h) = 1;

    d_est = sigma*norminv(p);
    D_est(h,l) = d_est;
    D_est(l,h) = -d_est;
end

sH = sum(H,2);
H = diag(1./sH)*H;
M = eye(N)-H;

M(end,:) = [zeros(1,N-1),1];
H(end,:) = zeros(1,N);

% calculation of estimated q and subsequent ranking:
b = sum(H.*D_est,2);
q_est = M^(-1)*b;
[¬, ordering] = sort(q_est);
```

Riportiamo, infine, la funzione random_graph_linear_weighted, la quale è del tutto analoga alla precedente, ma simula l'algoritmo nella sua versione pesata.

```
function [ordering] = random_graph_linear_weighted(q,N,K,sigma, Edges)

N_edges = size(Edges,1);

% matrix initialisation:
H = zeros(N,N);
```

```
D_est= zeros(N,N);

for edge=1:1:N_edges
    h = Edges(edge,1);
    l = Edges(edge,2);
    Δ = q(h)-q(l);
    y = Δ+randn(K,1)*sigma;
    % probability:
    p = mean(y>0);

    if(p==0)
        p=0.01;
    end
    if(p==1)
        p=0.99;
    end

    H(h,l) = exp(-2*erfinv(2*p-1).^2)/p/(1-p);
    H(l,h) = H(h,l);

    d_est = sigma*norminv(p);
    D_est(h,l) = d_est;
    D_est(l,h) = -d_est;
end

sH = sum(H,2);
H = diag(1./sH)*H;
M = eye(N)-H;

M(end,:) = [zeros(1,N-1),1];
H(end,:) = zeros(1,N);

% calculation of estimated q and subsequent ranking:
b = sum(H.*D_est,2);
q_est = M^(-1)*b;
[q_est_sorted, ordering] = sort(q_est);
```

A.2 Codice per la simulazione dell'algoritmo a due stadi

L'obiettivo di questa sezione, come anticipato, è di presentare i codici utilizzati per la simulazione dell'algoritmo a due stadi descritto nel quinto capitolo.

Di seguito riportiamo, in particolare, la funzione main che, mediante la chiamata di alcune funzioni ausiliarie, permette di calcolare, in modo analogo all'algoritmo a singolo stadio, le matrici perr e Dmean. Tali matrici riportano la P_e e la D al variare del numero di valutazioni utilizzate e del grado scelto per il grafo al primo e secondo stadio. È possibile, in particolare, settando in modo opportuno i parametri Algoritmo_1, Algoritmo_2, Graph_type_1, Graph_type_2 e max_th, scegliere l'algoritmo LS o WLS e la tipologia

di grafo da utilizzare al primo ed al secondo stadio e la tolleranza assunta sulle classifiche generate.

```
clear
close all

% Object set
N=20;   % number of objects
W=8000; % maximum number of evaluations

% initialisation of data structures to measure prestazioni:
dim_1=floor(W/400)+1;
dim_2=floor((N-1)/2);
perr=zeros(dim_1,dim_2,3);
Dmean=zeros(dim_1,dim_2,3);


sigma = 0.4;  % noise stddev

%Objects:
Qualities_distribution = 1; % 0=Equally spaced [0,1); 1=Uniform [0,1]

% Input qualities:
Qualities_ordering =2; % 1 = ordered (Genie Aided]; 2 = random

% Graph type: 1 = chain;  2 = closed chain;  3 = connected regular ...
    random graph;  4 = multiple chain (for 2, 4 or 6 degree)
Graph_type_1 = 2;
Graph_type_2= 1;

% Algorithm: 0 = LS; 1 = WLS
Algoritmo_1 = 0;
Algoritmo_2 = 0;

max_th = 0.08;

max_errors = 200;
max_runs = 1000000;

switch Qualities_distribution
    case 0
        q = (0:1/N:1-1/N)';   % equally spaced objects
    case 1
        q = sort(rand(N,1));  % uniform
end

[q_sort, ranking] = sort(q);

switch Qualities_ordering
    case 1
        q1 = q;
    case 2
```

```
49          q1 = q(randperm(N));
50  end
51
52  cont_1=0;
53  for t = 0:400:W          % t = # evaluations used
54      cont_1=cont_1+1;
55      cont_2=0;
56      for p = 2:2:N-1      % p = graph degree at the first stage
57          cont_2=cont_2+1;
58          cont_3=0;
59          for r =2:2:6     % r = graph degree at the second stage
60              cont_3=cont_3+1;
61
62              K_1 = floor((2*t)/(N*p));                          % # ...
                    evaluation for edge at the first stage
63              K_2 = floor((W-t)/(((N*r)/2)-((r^2)/8)-(r/4)));  % # ...
                    evaluation for edge at the second stage
64
65              Graph_degree_1 = p;
66              Graph_degree_2 = r;
67
68              % evaluation of the algorithm prestazioni:
69              errors = zeros(1,1);
70              Distortion=zeros(max_runs,1);
71
72              for s=1:1:max_runs
73
74                  ordering=zeros(N,2);
75                  if(K_1≠0 && K_2≠0)
76
77                      % FIRST STAGE:
78                      [Edges_1] = ...
                            CreateGraph(N,Graph_degree_1,Graph_type_1);
79                      switch  Algoritmo_1
80                          case 0
81                              [p_1] = ...
                                    allocate_ratings(q1,K_1,sigma,Edges_1);
82                              [H_1,D_est_1,ordering(:,1)] = ...
                                    get_ranking(N,sigma,p_1,Edges_1);
83                          case 1
84                              [p_1] = ...
                                    allocate_ratings(q1,K_1,sigma,Edges_1);
85                              [H_1,D_est_1,ordering(:,1)] = ...
                                    get_ranking_weighted(N,sigma,p_1,Edges_1);
86                      end
87
88                      % SECOND STAGE:
89                      [Edges_new] = CreateGraph(N,Graph_degree_2, ...
                            Graph_type_2);
90                      ord = ordering(:,1);
91                      Edges_2=ord(Edges_new);
92                      switch  Algoritmo_2
93                          case 0
```

```
94                              [ordering(:,2)] = ...
                                    get_final_ranking(N,q1,H_1,D_est_1, ...
                                    K_1,K_2,sigma,Edges_1, Edges_2,p_1);
95                          case 1
96                              [ordering(:,2)] = ...
                                    get_final_ranking_weighted(N,q1,H_1, ...
                                    D_est_1,K_1,K_2,sigma,Edges_1,Edges_2,p_1);
97                      end
98
99              end
100
101             if K_1==0 % apply only the second stage
102
103                 % SECOND STAGE:
104                 [Edges_2] = ...
                        CreateGraph(N,Graph_degree_2,Graph_type_2);  ...
                        % Chain graph
105                 ordering(:,1) = zeros(N,1);
106                 switch  Algoritmo_2
107                     case 0
108                         [p_1] = ...
                                allocate_ratings(q1,K_2,sigma,Edges_2);
109                         [¬,¬,ordering(:,2)] = ...
                                get_ranking(N,sigma,p_1,Edges_2);
110                     case 1
111                         [p_1] = ...
                                allocate_ratings(q1,K_2,sigma,Edges_2);
112                         [¬,¬,ordering(:,2)] = ...
                                get_ranking_weighted(N,sigma,p_1,Edges_2);
113                 end
114             end
115
116             if K_2==0 % apply only the first stage
117
118                 % FIRST STAGE:
119                 [Edges_1] = ...
                        CreateGraph(N,Graph_degree_1,Graph_type_1);
120                 ordering(:,1)=zeros(N,1);
121                 switch  Algoritmo_1
122                     case 0
123                         [p_1] = ...
                                allocate_ratings(q1,K_1,sigma,Edges_1);
124                         [¬,¬,ordering(:,2)] = ...
                                get_ranking(N,sigma,p_1,Edges_1);
125                     case 1
126                         [p_1] = ...
                                allocate_ratings(q1,K_1,sigma,Edges_1);
127                         [¬,¬,ordering(:,2)] = ...
                                get_ranking_weighted(N,sigma,p_1,Edges_1);
128                 end
129             end
130
131             % evaluation of the algorithm prestazioni:
```

```
132                 absdiff = abs(q-q1(ordering(:,2)));
133                 max_diff= max(absdiff);
134                 Distortion(s) = sum(absdiff)/N;
135                 if (max_diff>max_th)
136                     errors = errors+1;
137                 end
138                 if(errors≥ max_errors) % stop condition
139                     break;
140                 end
141             end
142             perr(cont_1,cont_2,cont_3) = errors/s;
143             Dmean(cont_1,cont_2,cont_3) = sum(Distortion)/s;
144             [t p r perr(cont_1,cont_2,cont_3) ...
                    Dmean(cont_1,cont_2,cont_3)]
145         end
146     end
147 end
```

La funzione CreateGraph, invocata dal main al fine di costruire i grafi necessari al primo ed al secondo stadio, è la medesima proposta nella simulazione dell'algoritmo a singolo stadio.

La funzione allocate_ratings nasce invece con lo scopo di restituire, quando invocata dal main, il vettore p, descritto nel capitolo 5, il quale rappresenta, in ogni sua componente, il valore, associato ad un determinato arco, che stima la probabilità che il primo oggetto dell'arco venga preferito, sulla base delle valutazioni simulate dei lavoratori, al secondo oggetto dell'arco stesso. Al fine di ricavare tale valore, la funzione riceve in input il vettore q delle qualità reali, il numero K di valutazioni di cui può usufruire ciascun arco, il parametro sigma che regola la variabilità nell'insieme di valutazioni che vengono simulate per ciascun arco del grafo ed Edges che rappresenta l'insieme di archi del grafo (e dunque la struttura del grafo in sè). Al fine di ricavare il vettore p, per ogni arco presente in Edges, vengono simulate, esattamente come descritto nella versione a singolo stadio dell'algoritmo, le K valutazioni espresse dai lavoratori.

```
1  function [p] = allocate_ratings(q,K,sigma,Edges)
2
3  N_edges = size(Edges,1);
4
5  p = zeros(N_edges,1);
6  for edge=1:1:N_edges
7    h = Edges(edge,1);
8    l = Edges(edge,2);
9    Δ = q(h)-q(l);
10   y = Δ+randn(K,1)*sigma;
11   % probability:
12   p(edge) = mean(y>0);
13
14   if(p(edge)==0)
15       p(edge)=0.01;
16   end
17   if(p(edge)==1)
```

```
18        p(edge)=0.99;
19    end
20 end
```

La funzione get_ranking viene invocata nel main al fine di ricavare l'ordinamento al primo stadio utilizzando l'algoritmo LS. La funzione, in particolare, riceve in ingresso il numero N di nodi e l'insieme Edges degli archi del grafo, il solito parametro sigma ed il vettore p descritto in precedenza (che il main riceve dalla funzione allocate_ratings). Tale funzione si occupa, in prima battuta, di popolare le matrici H e, mediante l'utilizzo del vettore p, D_est. Queste matrici racchiudono in sé tutta l'informazione relativa alle valutazioni espresse a questo stadio e vengono dunque restituite in output dalla funzione. Queste verranno passate, come vedremo, in ingresso alla funzione get_final_ranking, che applica l'algoritmo LS al secondo stadio. A questo punto, la funzione calcola la matrice H_fin, dalla quale è possibile ricavare la matrice M. Sulla base di tali matrici la funzione è in grado di stimare le qualità per gli N oggetti e, dunque, ricavare e restituire la classifica prodotta al primo stadio.

```
1  function [H,D_est,ordering] = get_ranking(N,sigma,p,Edges)
2
3  N_edges = size(Edges,1);
4
5  % matrix initialisation:
6  H = zeros(N,N);
7  H_fin = zeros(N,N);
8  D_est= zeros(N,N);
9
10 for edge=1:1:N_edges
11     h = Edges(edge,1);
12     l = Edges(edge,2);
13
14     H(h,l) = 1;
15     H(l,h) = 1;
16
17     d_est = sigma*norminv(p(edge));
18     D_est(h,l) = d_est;
19     D_est(l,h) = -d_est;
20 end
21
22 sH = sum(H,2);
23 H_fin = diag(1./sH)*H;
24 M = eye(N)-H_fin;
25
26 M(end,:) = [zeros(1,N-1),1];
27 H_fin(end,:) = zeros(1,N);
28
29 % calculation of estimated q and subsequent ranking:
30 b = sum(H_fin.*D_est,2);
31 q_est = M^(-1)*b;
32 [¬, ordering] = sort(q_est);
```

Di seguito riportiamo invece la funzione get_ranking_weighted, la quale è del tutto analoga alla precedente, ma simula l'algoritmo al primo stadio nella sua versione pesata.

```
function [H,D_est,ordering] = get_ranking_weighted(N,sigma,p,Edges)

N_edges = size(Edges,1);

% matrix initialisation:
H = zeros(N,N);
H_fin = zeros(N,N);
D_est= zeros(N,N);

for edge=1:1:N_edges
    h = Edges(edge,1);
    l = Edges(edge,2);

    H(h,l) = exp(-2*erfinv(2*p(edge)-1).^2)/p(edge)/(1-p(edge));
    H(l,h) = H(h,l);

    d_est = sigma*norminv(p(edge));
    D_est(h,l) = d_est;
    D_est(l,h) = -d_est;
end

sH = sum(H,2);
H_fin = diag(1./sH)*H;
M = eye(N)-H_fin;

M(end,:) = [zeros(1,N-1),1];
H_fin(end,:) = zeros(1,N);

% calculation of estimated q and subsequent ranking:
b = sum(H_fin.*D_est,2);
q_est = M^(-1)*b;
[¬, ordering] = sort(q_est);
```

La funzione get_final_ranking viene invocata nel main al fine di eseguire la versione LS dell'algoritmo al secondo stadio. A tal fine essa riceve in input il numero N di nodi del grafo, il vettore q delle qualità reali degli oggetti, le matrici H e D_est che rappresentano l'informazione fornita dalle valutazioni simulate al primo stadio (ricevute dal main come output della funzione get_ranking), il numero K_1 e K_2 di valutazioni per arco riservato al primo ed al secondo stadio, il solito parametro sigma, l'insieme degli archi del grafo al primo ed al secondo stadio Edges_1 ed Edges_2 ed il vettore delle probabilità p_1 (ricavato dal main come output dalla funzione allocate_ratings). Tale funzione, dopo aver simulato, assumendo sempre il medesimo modello di comportamento per i lavoratori, le valutazioni prodotte al secondo stadio e ricavato il valore di p_2 per ogni arco considerato, effettua, per l'arco considerato, un controllo per verificare se esso fosse già presente al primo stadio. Nel caso in cui non lo sia la funzione aggiorna le relative componenti delle matrici H e D_est utilizzando il valore della componente relativa all'arco di p_2 appena calcolato. Se era già presente al primo stadio, allora calcola la p_fin, così come descritta

nel capitolo 5, e sovrascrive in modo opportuno le relative componenti delle matrici H e D_est. A questo punto la funzione ricava la matrice M ed il vettore b con il quale può stimare le qualità degli N oggetti e, sulla base di tali stime, ricavare e restituire l'ordinamento finale.

```
function [ordering] = ...
    get_final_ranking(N,q,H,D_est,K_1,K_2,sigma,Edges_1,Edges_2,p_1)

N_edges_1 = size(Edges_1,1);
N_edges_2 = size(Edges_2,1);

for edge2=1:1:N_edges_2
    h = Edges_2(edge2,1);
    l = Edges_2(edge2,2);
    Δ = q(h)-q(l);
    y = Δ+randn(K_2,1)*sigma;
    % probability:
    p_2_componemnte = mean(y>0);

    % check if the edge in question was already present at the first ...
       stage:
    flag_trovato=0;
    for edge1=1:1:N_edges_1
        m = Edges_1(edge1,1);
        n = Edges_1(edge1,2);
        if (m==h && n==l)
            flag_trovato=1;
            p_1_componemnte = p_1(edge1);
            break;
        end
        if (m==l && n==h)
            flag_trovato=1;
            p_1_componemnte = 1-p_1(edge1);
            break;
        end
    end

    % combine, when necessary, the assessments at the two stages:
    if(flag_trovato==1)
        p=((K_1*p_1_componemnte)+(K_2*p_2_componemnte))/(K_1+K_2);
    else
        p=p_2_componemnte;
    end

    if(p==0)
        p=0.01;
    end
    if(p==1)
        p=0.99;
    end

    H(h,l) = 1;
```

```
46      H(l,h) = 1;
47
48      d_est = sigma*norminv(p);
49      D_est(h,l) = d_est;
50      D_est(l,h) = -d_est;
51  end
52
53  sH = sum(H,2);
54  H = diag(1./sH)*H;
55  M = eye(N)-H;
56
57  M(end,:) = [zeros(1,N-1),1];
58  H(end,:) = zeros(1,N);
59
60  % calculation of estimated q and subsequent ranking:
61  b = sum(H.*D_est,2);
62  q_est = M^(-1)*b;
63  [¬, ordering] = sort(q_est);
64  end
```

La funzione get_final_ranking_weighted è del tutto analoga alla funzione precedente ed ha la finalità di restituire la classifica prodotta al secondo stadio eseguendo la versione pesata dell'algoritmo, a partire dalle informazioni di input che provengono dalla funzione get_ranking_weighted.

```
1   function [ordering] = ...
        get_final_ranking_weighted(N,q,H,D_est,K_1,K_2,sigma,Edges_1, ...
        Edges_2,p_1)
2
3   H=K_1*H;
4
5   N_edges_1 = size(Edges_1,1); %#archi primo stadio
6   N_edges_2 = size(Edges_2,1); %#archi secondo stadio
7
8   for edge2=1:1:N_edges_2
9       h = Edges_2(edge2,1);
10      l = Edges_2(edge2,2);
11      ∆ = q(h)-q(l);
12      y = ∆+randn(K_2,1)*sigma;
13      % probability:
14      p_2_componemnte = mean(y>0);
15
16      % check if the edge in question was already present at the first ...
            stage:
17      flag_trovato=0;
18      for edge1=1:1:N_edges_1
19          m = Edges_1(edge1,1);
20          n = Edges_1(edge1,2);
21          if (m==h && n==l)
22              flag_trovato=1;
23              p_1_componemnte = p_1(edge1);
24              break;
```

```
25          end
26          if (m==l && n==h)
27              flag_trovato=1;
28              p_1_componemnte = 1-p_1(edge1);
29              break;
30          end
31      end
32
33      % combine, when necessary, the assessments at the two stages:
34      if(flag_trovato==1)
35          p=((K_1*p_1_componemnte)+(K_2*p_2_componemnte))/(K_1+K_2);
36      else
37          p=p_2_componemnte;
38      end
39
40      if(p==0)
41          p=0.01;
42      end
43      if(p==1)
44          p=0.99;
45      end
46
47      H(h,l) = H(h,l)+K_2*exp(-2*erfinv(2*p-1).^2)/p/(1-p);
48      H(l,h) = H(h,l);
49
50      d_est = sigma*norminv(p);
51      D_est(h,l) = d_est;
52      D_est(l,h) = -d_est;
53
54  end
55
56  sH = sum(H,2);
57  H = diag(1./sH)*H;
58  M = eye(N)-H;
59
60  M(end,:) = [zeros(1,N-1),1];
61  H(end,:) = zeros(1,N);
62
63  % calculation of estimated q and subsequent ranking:
64  b = sum(H.*D_est,2);
65  q_est = M^(-1)*b;
66  [q_est_sorted, ordering] = sort(q_est);
67  end
```

Bibliografia

Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

Yuxin Chen and Changho Suh. Spectral mle: Top-k rank aggregation from pairwise comparisons. In *International Conference on Machine Learning*, pages 371–380. PMLR, 2015.

E. Christoforou, A. Nordio, A. Tarable, and E. Leonardi. Ranking a set of objects: A graph based least-square approach. *IEEE Transactions on Network Science and Engineering*, 8(1):803–813, 2021. doi: 10.1109/TNSE.2021.3053423.

Mihai Cucuringu. Sync-rank: Robust ranking, constrained ranking and rank aggregation via eigenvector and sdp synchronization. *IEEE Transactions on Network Science and Engineering*, 3(1):58–79, 2016.

Alexandre d'Aspremont, Mihai Cucuringu, and Hemant Tyagi. Ranking and synchronization from pairwise measurements via svd. *arXiv preprint arXiv:1906.02746*, 2019.

Moein Falahatgar, Alon Orlitsky, Venkatadheeraj Pichapati, and Ananda Theertha Suresh. Maximum selection and ranking under noisy comparisons. In *International Conference on Machine Learning*, pages 1088–1096. PMLR, 2017.

Moein Falahatgar, Ayush Jain, Alon Orlitsky, Venkatadheeraj Pichapati, and Vaishakh Ravindrakumar. The limits of maxing, ranking, and preference learning. In *International Conference on Machine Learning*, pages 1427–1436. PMLR, 2018.

Gregory Gutin and Anders Yeo. Ranking the vertices of a complete multipartite paired comparison digraph. *Discrete applied mathematics*, 69(1-2):75–82, 1996.

Reinhard Heckel, Nihar B Shah, Kannan Ramchandran, Martin J Wainwright, et al. Active ranking from pairwise comparisons and when parametric assumptions do not help. *Annals of Statistics*, 47(6):3099–3126, 2019.

Julien Hendrickx, Alexander Olshevsky, and Venkatesh Saligrama. Graph resistance and learning from pairwise comparisons. In *International Conference on Machine Learning*, pages 2702–2711. PMLR, 2019.

Ralf Herbrich, Tom Minka, and Thore Graepel. Trueskill™: a bayesian skill rating system. In *Proceedings of the 19th international conference on neural information processing systems*, pages 569–576, 2006.

Anil N Hirani, Kaushik Kalyanaraman, and Seth Watts. Least squares ranking on graphs. *arXiv preprint arXiv:1011.1716*, 2010.

Minje Jang, Sunghyun Kim, Changho Suh, and Sewoong Oh. Top-k ranking from pairwise comparisons: When spectral ranking is optimal. *arXiv preprint arXiv:1603.04153*, 2016.

R Duncan Luce. *Individual choice behavior: A theoretical analysis*. Courier Corporation, 2012.

Sahand Negahban, Sewoong Oh, and Devavrat Shah. Iterative ranking from pair-wise comparisons. *Advances in neural information processing systems*, 25:2474–2482, 2012.

Sahand Negahban, Sewoong Oh, and Devavrat Shah. Rank centrality: Ranking from pairwise comparisons. *Operations Research*, 65(1):266–287, 2017.

R. L. Plackett. The analysis of permutations. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 24(2):193–202, 1975. ISSN 00359254, 14679876. URL http://www.jstor.org/stable/2346567.

Matthew Richardson, Ewa Dominowska, and Robert Ragno. Predicting clicks: estimating the click-through rate for new ads. In *Proceedings of the 16th international conference on World Wide Web*, pages 521–530, 2007.

Nihar Shah, Sivaraman Balakrishnan, Joseph Bradley, Abhay Parekh, Kannan Ramchandran, and Martin Wainwright. Estimation from pairwise comparisons: Sharp minimax bounds with topology dependence. In *Artificial Intelligence and Statistics*, pages 856–865. PMLR, 2015.

Nihar B Shah and Martin J Wainwright. Simple, robust and optimal ranking from pairwise comparisons. *The Journal of Machine Learning Research*, 18(1):7246–7283, 2017.

Balázs Szörényi, Róbert Busa-Fekete, Adil Paul, and Eyke Hüllermeier. Online rank elicitation for plackett-luce: A dueling bandits approach. 2015.

Louis L Thurstone. The method of paired comparisons for social values. *The Journal of Abnormal and Social Psychology*, 21(4):384, 1927.

Yun Zhai. *Non-numerical ranking based on pairwise comparisons*. PhD thesis, 2010.