POLITECNICO DI MILANO

Facoltà di Ingegneria dell'Informazione

Corso di Laurea Specialistica in Ingegneria Informatica

Result Diversification

Space Partitioning & Probing: Batched Access

Relatore: Prof. Davide MARTINENGHI

Tesi di Laurea di:

Davide GHIRALDELLI

Matricola n. 734540

Anno Accademico 2012–2013

A mia madre e mio padre che mi hanno sostenuto e spronato dal primo all'ultimo giorno.

A Simone che mi ha spinto ad essere un buon esempio ed è sempre stato dalla mia parte in ogni occasione.

Ad Ilaria che nei momenti più bui è stata la mia luce

DAVIDE.

Sommario

Negli ultimi anni le ricerche sul web hanno avuto un impatto sempre maggiore nella vita delle persone. In particolare motori di ricerca su argomenti specifici, come hotel, compagnie aeree, case in vendita e molti altri, sono sempre più utilizzati ed apprezzati. Per molte di queste ricerche, però, non sono sufficienti algoritmi semplici, infatti, molto spesso, le esigenze degli utenti sono molto varie e sfaccettate. Pensiamo un attimo a ricerche composte su più domini diversi, ad esempio una ricerca combinata di aerei e case vacanze. Per poter soddisfare ricerche di questo tipo servono algoritmi che estraggano risultati da più data service e li combinino secondo determinate funzioni, per ottenere dei dati rilevanti per l'utente finale. Uno di questi casi particolari, è il problema della diversificazione dei risultati, ossia l'esigenza da parte dell'utente di ottenere dei risultati che non siano ordinati secondo un solo parametro (ad esempio il prezzo), ma che siano fra loro il più vario possibile; quindi ad esempio una ricerca sugli hotel non dovrebbe fornire solo i risultati in ordine di prezzo, ma mostrare l'hotel più costoso, il più economico, quello con una valutazione dei clienti migliore e magari quello più vicino ad un luogo particolare. Alcuni algoritmi di diversificazione dei risultati sono stati proposti in letteratura, fra questi MMR è il più conosciuto ed usato. Il punto di partenza di questo lavoro di tesi è SPP un algoritmo di diversificazione spaziale basato su strutture geometriche. L'obiettivo di SPP era di ottenere gli stessi risultati di MMR accedendo ad un numero molto minore di oggetti, per migliorare le prestazioni del siste-

ma. Il mio scopo, con questo lavoro, è stato quello di studiare la politica di accesso agli oggetti di SPP per trovare dei possibili miglioramenti. Mi sono focalizzato in particolare su una versione dell'algoritmo che accede, ad ogni iterazione, ad un numero di oggetti determinato da un budget, studiando in che modo far variare questo budget per ottenere risultati migliori.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Nel 2012 sono state effettuate in media 5,134,000,000 ricerche al giorno su Google (fonte: [12]).
Questo dato ci fa capire quale importanza abbiano assunto, negli ultimi anni, i motori di ricerca nel web.
Un motore di ricerca è un sistema automatico che, tramite algoritmi matematici, recupera dati, raccolti da esso stesso o presenti in altri sistemi, e li ordina per rispondere alle richieste di un'utente.
Nel caso base, la ricerca viene effettuata a seconda di un determinato parametro detta chiave; la chiave può essere singola o composta da piu elementi. Il risultato avrà un punteggio più alto a seconda della rilevanza del suo contenuto rispetto alle chiavi ricercate.
In molti casi però, le esigenze degli utenti, sono complicate e difficilmente possono essere soddisfatte con ricerche semplici. Ad esempio, se un'utente volesse organizzare una vacanza, dovrebbe ricercare prima gli aerei per raggiungere la destinazione, poi gli hotel nel luogo desiderato. Per fare una ricerca di questo tipo devono essere eseguite diverse interrogazioni a diversi servizi sul web: prima ai siti delle diverse compagnie aeree, poi ai siti di ricerca di hotel.

Negli ultimi anni, però, si stanno sviluppando sempre più servizi di ricerca multi-dominio, che cercano di unire i risultati ricavati da diversi motori di ricerca singoli che agiscono su domini differenti, per fornire risultati composti che soddisfino maggiormente l'utente.
Un altro esempio dell'evoluzione delle ricerche nei sistemi informativi è l'utilizzo di tecniche di diversificazione dei risultati. La diversificazione nasce dall'esigenza degli utenti di ottenere risultati su una determinata chiave che coprano maggiormente lo spettro delle possibilità . Ad esempio effettuando una ricerca sugli hotel, normalmente vengono restituiti i risultati ordinati secondo un determinato parametro. Molte volte questo parametro è scelto dall'utente, ad esempio il numero di stelle, ma, facendo questo, si ottengono come primi risultati della ricerca soluzioni molto simili tra loro. Nel nostro caso otterremmo per primi tutti gli hotel di lusso e solo alla fine gli hotel peggiori, ma probabilmente più economici.
Una soluzione che potrebbe essere più utile all'utente finale, sarebbe quella di fornire, come prime risposte di una query, dei risultati che abbiano, oltre ad uno score alto sulla chiave di ricerca, anche un alto livello di diversificazione fra loro. Nel caso degli hotel ad esempio potrebbe essere utile restituire come prime risposte l'hotel con più stelle, quello valutato meglio, l'hotel più economico, quello più vicino ad un punto scelto e cosi via...
La diversificazione dei risultati acquista dunque una grande importanza nelle ricerche svolte dagli utenti, che diventano sempre più complesse di giorno in giorno.
In letteratura sono presenti diversi algoritmi di diversificazione, ma noi ne vedremo uno in particolare: SPP, algoritmo di diversificazione spaziale basato su particolari strutture geometriche, i diagrammi di Voronoi. Di questo algoritmo ne implementeremo una variante per cercare di ottenere prestazioni migliori.

1.1 Lavoro Svolto

Il lavoro da me svolto è stato un'analisi della versione pre-esistente di SPP e della sua estensione SPP-BA.
Partendo dagli studi effettuati su questi due algoritmi, ho trovato una direzione di miglioramento che consiste nell'estendere la versione Batched Access di SPP studiando un metodo efficace per regolare il numero di oggetti da recuperare ad ogni passo dell'algoritmo.
Varie prove sperimentali sono state effettuate per raggiungere il risultato cercato, e, una volta trovato, molte alte prove sono state effettuate per verificare la sua effettiva utilità .

1.2 Organizzazione del Documento

La scrittura di questo documento di Tesi è stata organizzata in questo modo:

- **CAP 2** : Nel secondo capitolo della tesi è presente lo stato dell'arte nell'ambito degli algoritmi di ricerca. In particolare sono presenti descrizioni dettagliate delle query Top-K, spiegando la loro utilità e le loro applicazioni. Inoltre, viene presentato il problema della diversificazione dei risultati, mostrando alcuni degli algoritmi più conosciuti ed utilizzati, fra i quali spicca MMR.

- **CAP 3** : Il capitolo 3 è focalizzato sugli algoritmi di diversificazione, in particolare sull'algoritmo SPP, del quale vengono descritti nel dettaglio lo scopo, il funzionamento e l'implementazione. Dopodichè è il turno di SPP-BA, algoritmo che nasce come estensione di SPP aggiungendo una nuova politica di accesso agli oggetti. Anche SPP-BA viene descritto dettagliatamente.

- **CAP 4** : Il quarto capitolo mostra gli studi effettuati sulla versione pre-esistente di SPP-BA, e le modifiche apportate all'algoritmo, con particolare focus al parametro che determina il budget di oggetti da recuperare ad ogni passo dell'algoritmo, e alla sua variazione durante l'esecuzione.

- **CAP 5** : In questo capitolo vengono mostrate nel dettaglio le implementazioni software realizzate dell'algoritmo SPP-BA, presentando sia la versione Java che la versione MatLab oltre ai programmi di supporto. Inoltre vengono spiegate le modifiche effettuate durante il lavoro di tesi alla versione Java del codice.

- **CAP 6** : Il capitolo 6 presenta i risultati ottenuti applicando ad SPP-BA la funzione di gestione del budget ipotizzata nel capitolo 4. I risultati vengono calcolati considerando tutte le possibili variazioni di parametri dell'algoritmo, per poter trarre una conclusione che sia valida per ogni istanza dell'algoritmo.

- **CAP 7** : Nell'ultimo capitolo verrà fatto un riassunto del lavoro svolto, verranno descritte le conclusioni che si possono trarre da quanto esposto e , infine, verranno mostrati i possibili sviluppi futuri sul tema trattato.

Capitolo 2

Stato Dell'Arte

2.1 Top-K

Nell' eseguire una ricerca, in molti sistemi informativi, soprattutto se basati sul web, l'utente finale è solitamente interessato ad una parte ristretta dei risultati possibili. Infatti fra tutte le informazioni che vengono recuperate usando determinati parametri di ricerca, solo alcune risultano effettivamente utili.
Normalmente quindi, ad ogni singolo risultato di una query, viene associata una valutazione che cerca di dare un'idea della qualità dell'informazione in esso contenuta e, all'utente finale, vengono presentati per primi i risultati con una valutazione più alta, perché sono quelli con una più alta possibilità di soddisfare le necessità iniziali della ricerca.
Le query di tipo TOP-K si pongono dunque l' obiettivo di trovare e fornire all'utente i K risultati migliori, classificandoli tra tutti i risultati possibili.
Per poter effettuare questa classificazione, vengono utilizzate delle funzioni di scoring, che si occupano di associare ad ogni risultato uno score che ne rappresenti la significatività.

I problemi nella realizzazione di un buon sistema di query top-k sono quindi molteplici. Bisogna innanzitutto riuscire a definire delle funzioni di scoring che riescano davvero ad associare ai risultati un valore che ben rappresenti, sia l'oggettiva coerenza dei risultati con il dominio della richiesta, sia il soggettivo grado di soddisfazione dei requisiti dell'utente finale.
Un altro problema è sicuramente quello di controllare le prestazioni del sistema di ricerca, infatti uno strumento che fornisca le migliori risposte possibili, ma in tempi lunghi, non è quasi mai più utile di un sistema che fornisce risposte in tempi brevi anche se magari non sono altrettanto significative.
Già da queste poche informazioni possiamo capire come nello sviluppare sistemi di ricerca top-k bisogna cercare di mediare diversi fattori come la bontà dei risultati e le prestazioni del sistema.
In realtà i diversi algoritmi di ricerca top-k si differenziano rispetto a molti altri parametri.
Una buona panoramica dei vari algoritmi presenti in letteratura viene fornita nell'articolo di Ilyas, Beskales e Soliman [1].

2.2 Diversification

Finora abbiamo preso in considerazione la possibilità di funzioni di scoring semplici, che diversificano i risultati in base ad un solo parametro. In realtà, in molti casi, vengono utilizzati più parametri in contemporanea, e quindi si calcola una funzione di scoring che assegni diversi *pesi* a ciascun parametro e calcoli un risultato unificato.
Oppure, oltre ad avere più parametri, si potrebbe avere un caso multi-dominio, nel quale si cerca di soddisfare contemporaneamente più tipi di ricerche diverse: ad esempio un utente potrebbe cercare un hotel ed un ristorante che siano vicini tra loro.

Per effettuare una ricerca di questo tipo si devono prendere in considerazione i risultati forniti dalla ricerca dell'hotel, quelli forniti dalla ricerca del ristorante ed aggiungere un parametro che è la distanza fra le varie coppie hotel-ristorante.
Normalmente in questi casi vengono effettuate operazioni di join fra i risultati delle differenti query, fornendo funzioni di scoring a volte complesse. Anche in questo caso sono stati proposti molti differenti approcci, che variano a seconda del metodo di query utilizzato[2], delle policy di accesso ai dati[3] e delle scoring function utilizzate[4].
Uno degli aspetti che può risultare fondamentale soprattutto in ambito multi-dominio è la diversificazione dei risultati.
Per spiegare cosa intendiamo come diversificazione, introduciamo un nuovo parametro che rappresenta quanto un risultato sia diverso da un altro: ad esempio nella ricerca di un hotel, uno a 5 stelle è molto *diverso* da uno a 2 stelle, mentre uno a 4 stelle è meno *diverso*.
La stessa cosa può essere fatta su ogni parametro di ricerca, ad esempio il costo (hotel con costi simili o meno), la distanza da un altro luogo e così via.
La diversificazione può risultare utile per coprire maggiormente lo spettro di possibilità rappresentate dai vari risultati. Così ad esempio un utente che ricerca hotel potrebbe preferire che i primi k risultati che gli vengono forniti non siano banalmente ordinati secondo il prezzo o il numero di stelle, ma che coprano maggiormente lo spazio delle possibilità, quindi ad esempio i primi risultati saranno l'hotel col maggior numero di stelle, il più economico, il più vicino ad una determinata posizione e così via.
In particolare la diversificazione diventa ancora più importante nell'ambito multi-dominio, infatti quando un oggetto ha un alto scoring fra quelli restituiti da una query, le coppie che conterranno quell'oggetto saranno con alta probabilità fra le coppie con scoring complessivo alto, in tal modo que-

st'oggetto apparirà più volte fra i primi k risultati rendendo la risposta poco significativa per l'utente.
Nell'esempio degli hotel e dei ristoranti, le coppie che contengono l' hotel con la valutazione migliore saranno facilmente ai primi posti della graduatoria composta, ma introducendo un parametro di diversificazione nella funzione di calcolo dello scoring composto otterremo risultati più vari, che magari perderanno un po' di precisione, ma risulteranno infine più significativi.

2.3 MMR

Possiamo ora definire formalmente un problema di diversificazione come una tupla <O,S_q, δ,K,F> dove:

- O è un set finito di oggetti, come ad esempio il risultato di una query q
- $S_q : O \to \mathbb{R}$ è una funzione di scoring sugli oggetti di O
- $\delta : O \times O \to \mathbb{R}^+$ è una funzione di distanza fra coppie di elementi di O
- K è un intero positivo tale che $K \leqslant |O|$
- $F : 2^O \times S_q \times \delta \to \mathbb{R}$ è una funzione obiettivo che associa ogni set $O_l \subseteq O$ ad un numero reale basandosi su S_q e δ tenendo in conto sia la rilevanza che la diversità.

Risolvere un problema di diversificazione vuol dire quindi trovare il set di oggetti che soddisfa la seguente funzione:

$$O_K^* = \underset{O_K \subseteq O, |O_K| = K}{\arg\max} F(O_K; S_q(\cdot, q), \delta(\cdot, \cdot)) \tag{2.1}$$

Quindi la rilevanza di un oggetto $o \in O$ rispetto alla query q viene espressa da $S_q(o) \in \mathbb{R}$ che associa all'oggetto un valore normalizzato da 0 a 1 dove 1 rappresenta la massima rilevanza.
La diversità viene invece definita come la distanza euclidea fra due oggetti.
La soluzione esatta del problema di diversificazione è dunque il set di K oggetti tali che O_K^* sia definito come in (2.1). Un algoritmo che mappa il problema <O,S_q, δ,K,F> alla soluzione ottimale O_K^* è dunque un algoritmo *corretto*.
Ma risolvere l'equazione (2.1) in modo esatto è stato provato in [2] essere un problema NP-difficile, per questo sono stati proposti e validati sperimentalmente diversi algoritmi approssimati di tipo *greedy*.
Fra questi algoritmi MMR (Maximum Marginal Relevance) proposto da Carbonell e Goldstein in [3] è uno dei più popolari.
MMR tenta di risolvere una forma particolare del problema di diversificazione, esso utilizza infatti una funzione obiettivo nella forma:

$$F(O_K) = (1-\lambda) \sum o \in O_K S_q(o) + \lambda \min_{o' \in O_K} \delta(o_u, o_v) \qquad (2.2)$$

dove λ è un parametro compreso fra 0 e 1 che rappresenta il peso da dare alla diversità e alla rilevanza nel calcolo del punteggio da assegnare all'oggetto o.
La funzione obiettivo di MMR è un ibrido fra 2 funzioni studiate in [2]: *MaxMin* e *MaxSum*, in particolare quando $\lambda = 0$ è equivalente a *MaxSum* mentre quando $\lambda = 1$ è equivalente a *MaxMin*.

L'algoritmo 1 illustra MMR nel dettaglio.

Input: Set of objects O; result size K

Output: $O_K \in O$

$o^* = \arg\max_{o \in O} S_q(o)$;

$O_K = o^*$;

while $|O_K| < K$ **do**

  $o^* = \arg\max_{o \in O \setminus O_K} \{(1-\lambda) S_q(o) + \lambda \min_{o' \in K} \delta(o, o')\}$;

  $O_K = O_K \cup (o^*)$;

end

Algorithm 1: Algoritmo MMR

MMR funziona trovando ad ogni passo un oggetto o^* che sia allo stesso tempo rilevante e il più possibile diverso dagli oggetti trovati in precedenza.

Per soddisfare entrambe queste condizioni l'oggetto da selezionare deve massimizzare un punteggio calcolato tramite la seguente funzione:

$$\gamma(o, O_K) = (1-\lambda) S_q(o) + \lambda \min_{o' \in O_K} \delta(o, o') \tag{2.3}$$

Il punteggio ottenuto viene detto *diversity weighted score*.

Il primo oggetto viene selezionato secondo una *initialization strategy IS*, e di solito è l'oggetto con lo score più alto.

Successivamente vengono aggiunti al set degli oggetti trovati gli oggetti come descritto in precedenza, fino a raggiungere il numero di risultati desiderati.

Capitolo 3

Algoritmi di diversificazione

3.1 SPP: dividere lo spazio ed esplorare

Il problema di diversificazione presentato nell'articolo [5], si riferisce ad ambiti nei quali gli oggetti sono rappresentabili come chiusi in una regione finita e confinata, e dove sia disponibile, come unica politica di accesso agli oggetti, un accesso ordinato.
In questo contesto è stato definito l'algoritmo SPP (Space Partitioning and Probing) il cui obbiettivo principale è di computare il set di risultati del problema di diversificazione accedendo ad un numero ristretto degli oggetti disponibili.
SPP estende e specifica una famiglia di algoritmi definiti da PBMMR (Pull/Bound MMR)

3.1.1 Pull/Bound MMR

PBMMR è un template che definisce una sottocategoria di algoritmi MMR, caratterizzati dalla presenza di regioni delimitate e di una pulling

strategy, ossia una strategia ben definita di accesso agli oggetti.
In questa categoria di algoritmi sono possibili 2 tipologie di accesso ordinato agli oggetti:

- accesso basato sulla distanza, che permette di accedere agli oggetti in ordine crescente rispetto alla funzione di distanza $\delta(\cdot, v)$
- accesso basato sulla funzione di scoring in maniera decrescente rispetto a $S_q(o)$

La v dell'accesso basato sulla distanza è un vettore in $\mathbb{R}^d$ chiamato *probing location* che contiene un insieme di punti dai quali partire per effettuare ricerche per distanza geografica, metodo ampiamente conosciuto ed utilizzato nei data source sul web.
L'algoritmo di PBMMR è il seguente:

Input: K,U

Output: O_K

Parametri: strategia d'inizializzazione IS, pulling strategy PS,

Bounding scheme BS

if *K==1* **then**

$P = P \cup IS.initialObject()$;

$D = \emptyset$; $S_q^{last} = 1$;$P = O_K$;

end

$\tau = \infty$;

$O_{K-1} = PBMMR(K-1, U)$;

$o_k{}^* = \arg\max_{o \in P \setminus O_{K-1}} \sigma(o, O_{K-1})$;

while $\sigma(o_k{}^*, OK-1) < \tau$ *and* $D \subset U$ **do**

m= PS.chooseAccessMethod() ;

o= m.getNextObject() ;

$P = P \cup o$;

if $\sigma(o, O_{K-1}) > \sigma(o_k{}^*, O_{K-1})$ **then**

$o_k{}^* = o$;

end

$(D, S_q^{last}) = updateDiscardedRegionAndScore(D, S_q^{last})$;

$\tau = BS.updateBound(D, S_q^{last}, O_l)$;

end

Algorithm 2: Algoritmo PBMMR

PBMMR comincia selezionando un'oggetto di partenza secondo una determinata strategia d'inizializzazione (IS), dopodichè vengono istanziate alcune variabili di controllo che rappresentano:

- la sottoregione già esplorata della regione U, ossia D, inizializzato a 0
- lo score massimo per gli oggetti non ancora visitati, S_q^{last} , inizializzato a 1

- il set P di ogetti selezionati correntemente, inizializzato all'oggetto selezionato con la IS definita

Successivamente l'algoritmo esegue altre k-1 ripetizioni per recuperare i restanti k-1 oggetti.
In ciascuna ripetizione vengono settati il limite massimo τ al punteggio pesato della diversità degli oggetti non ancora visitati e il punteggio più alto corrente fra gli oggetti già selezionati σ^*.
Se sono presenti degli oggetti non ancora valutati, trovati in una delle precedenti iterazioni, quello con il punteggio più alto viene selezionato come il miglior candidato o^* per questa ripetizione e σ^* viene aggiornato di conseguenza.
Il loop interno esegue dei passi di esplorazione della regione non ancora valutata per aggiornare o^* e τ fino a trovare un oggetto che raggiunga τ o fino a che l'intera regione non è stata esplorata.
Ad ogni esplorazione viene scelto un metodo di accesso agli oggetti (distanza o punteggio) seguendo la *pulling strategy* selezionata. Nel caso di accesso tramite distanza, la funzione decide anche da quale probing location effettuare la ricerca.
Non appena un oggetto viene selezionato esso viene aggiunto all'insieme di oggetti visitati P, e, se il suo punteggio è più alto di quello del miglior oggetto selezionato finora in questo passo, ossia τ^*, esso viene selezionato come nuovo oggetto migliore o^* e il valore di τ^* viene aggiornato.
Gli accessi basati sulla distanza aumentano la porzione di spazio visitato, mentre un accesso basato sul punteggio serve ad abbassare il limite massimo sul punteggio degli oggetti non ancora visitati; così D e S_q^{last} vengono aggiornati a seconda dell'accesso eseguito.
Infine l'algoritmo aggiorna la soglia adeguatamente.
Bisogna far notare che ogni algoritmo basato sul template definito da PBMMR risulta MMR-corretto se viene assicurato che la funzione di aggiornamento

del limite superiore restituisca effettivamente un limite allo score degli oggetti, e che la pulling strategy fornisca una politica di accesso che permetta di valutare tutti gli oggetti della regione in un tempo finito.
Per dare un metodo di valutazione della bontà degli algoritmi basati su PBMMR definiamo una metrica di costo: la sumdepth.
Definiamo depth come il numero di oggetti recuperati dall'algoritmo in un singolo passo effettuato tramite accesso basato sulla distanza oppure con un accesso basato sul punteggio.
In questo modo possiamo definire la sumdepth come la sommatoria delle varie depth calcolate ad ogni passo.
Quindi la sumdepth rappresenta gli oggetti visitati in totale durante la ricerca e diventa una buona metrica per valutare la bontà di un algoritmo che cerca di minimizzare il numero di oggetti visitati.

3.1.2 Algoritmo SPP

Fra i vari algoritmi ricavati dalle estensioni di PBMMR, quello sul quale mi sono concentrato è SPP.
Come abbiamo detto le caratteristiche che distinguono le varie istanze di PBMMR sono la PS (pulling strategy) e la strategia di aggiornamento della soglia.
Per prima cosa valutiamo la PS scelta: ad ogni passo dell'algoritmo, se si sceglie un accesso basato sulla distanza , bisogna selezionare anche un punto di accesso dal quale eseguire la ricerca. Il punto scelto dovrebbe essere il più promettente fra i punti possibili, in modo da riuscire a trovare l'oggetto migliore col minor numero di interrogazioni possibili.
Per definire questi punti promettenti (detti probing location) si parte dal presupposto che, per massimizzare il punteggio dato dalla distanza dei punti selezionati, bisogna valutare un insieme di punti che siano contemporaneamente distanti da tutti gli elementi selezionati fino a quel momento.

Per far questo in SPP vengono utilizzati i diagrammi di Voronoi.[8]

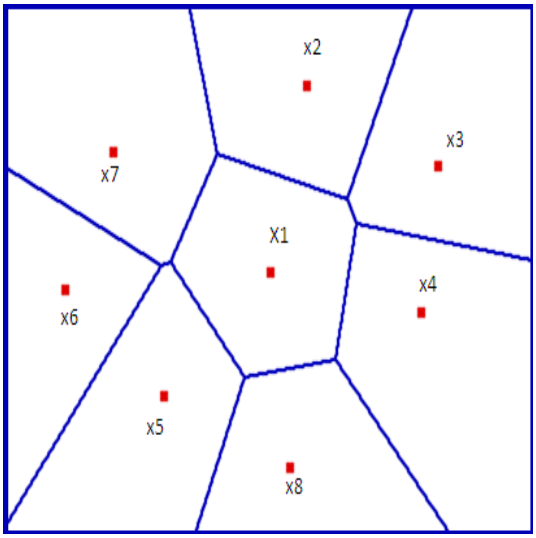

Figura 3.1: Diagramma di Voronoi

In [6] viene dimostrato che i vertici dei diagrammi di Voronoi calcolati partendo dagli oggetti già selezionati sono dei massimi locali per la funzione di distanza, perciò sono i punti migliori dai quali iniziare a cercare il nuovo oggetto da aggiungere.
Il diagramma di Voronoi è una suddivisione dello spazio effettuata calcolando le distanze da un insieme di punti chiamati centroidi. Lo spazio metrico viene suddiviso in modo che i punti all'interno di ogni regione abbiano la proprietà di essere più vicini al centroide di quella regione piuttosto che a qualsiasi altro centroide. I punti limite di ciascuna regione formano i lati delle celle e l'intersezione dei lati definiscono i vertici del diagramma.
Ad ogni passo dell'algoritmo, ogni volta che un nuovo oggetto viene aggiunto a quelli selezionati per la soluzione finale, l'algoritmo ricalcola il diagramma di Voronoi includendo questo nuovo punto nell'insieme dei centroidi.
Il diagramma calcolato da SPP è però limitato dalla regione iniziale U, quin-

di le celle sono intersecate con la regione di spazio delimitata, per far questo i vertici della regione U diventano vertici del diagramma di Voronoi calcolato.

Essendo stato dimostrato che i vertici del diagramma di Voronoi sono dei massimi locali alla funzione di distanza sugli oggetti della regione, il massimo assoluto sarà anch'esso uno dei vertici del diagramma.

Partendo da questo presupposto si sceglie come probing location di un accesso basato sulla distanza di volta in volta il vertice del diagramma con punteggio di diversificazione più alto.

Da questo punto si effettua una ricerca di tipo geografico , che accede agli oggetti ad un determinato raggio di distanza dal vertice.

Mano a mano che nuovi oggetti vengono aggiunti all'insieme degli oggetti visitati, il raggio che parte dal vertice si allarga, e una regione maggiore di spazio viene visitata.

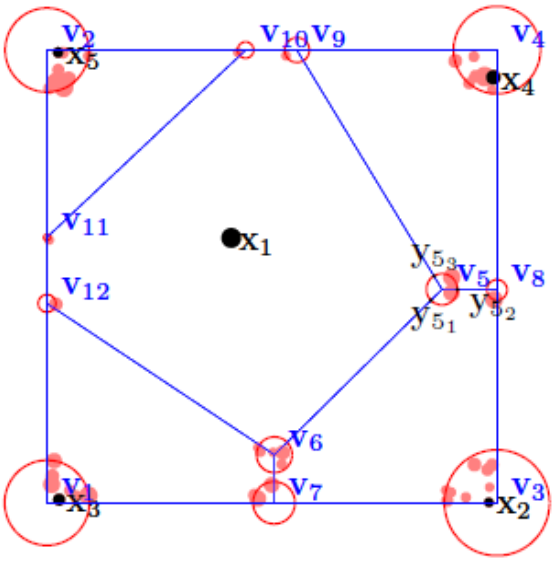

Figura 3.2: Sfere di ricerca per probing location

Ad ogni accesso dunque si copre una porzione di spazio pari alla circonferenza che ha come centro il vertice e come raggio la distanza fra il vertice e l'oggetto trovato. Quindi dopo ogni accesso viene ricalcolata la porzione di U non ancora visitata.
Possiamo ora vedere come aggiornare la soglia quando viene effettuato un accesso per distanza: in questo caso il limite superiore per la soglia viene determinato dallo score di un oggetto non ancora visitato che risolva l'equazione

$$x^* = \max_{X \in Z} \min_{y \in X} ||X - Y|| \tag{3.1}$$

Aggiungendo anche la possibilità di un accesso basato sul punteggio si ottiene che un limite superiore stringente sul punteggio pesato di diversificazione può essere calcolato in questo modo:

$$\tau = (1 - \lambda) S_q^{last} + \lambda \max_{X \in Z} \min_{y \in X} ||X - Y|| \tag{3.2}$$

In teoria la soluzione a questa equazione richiederebbe l'accesso a tutti i punti non ancora valutati, ma i vertici del diagramma di voronoi sono quelli che massimizzano la minima distanza, quindi il limite alla soglia può essere calcolata basandosi solo su questi punti.
Come strategia per scegliere di volta in volta il metodo di accesso migliore si può usare round robin, ossia utilizzare alternativamente prima una tipologia poi l'altra, oppure si può usare una strategia adattativa basata sul potenziale di ogni probing location in modo da abbassare la soglia più velocemente.

3.2 Batched Access

Di SPP ne è stata sviluppata in [7] anche una versione basata su una diversa metodologia di accesso agli oggetti: è la versione Batched Access.
Nella versione base di SPP, quando si effettua un accesso per distanza, gli oggetti vicini alla probing location scelta vengono valutati uno alla volta. Il raggio di ricerca aumenta ad ogni accesso, ma non è direttamente controllato. In alcuni algoritmi di ricerca per distanza pre-esistenti, è possibile invece definire un raggio di ricerca e trovare tutti gli oggetti compresi in quel raggio.
L'idea alla base degli accessi batched è di sfruttare questa tipologia di ricerca per poter accedere ad un maggior numero di oggetti ad ogni chiamata, infatti il costo di un singolo accesso ai dati potrebbe essere elevato, quindi effettuare un numero minore di accessi potrebbe portare un miglioramento delle prestazioni.
Il problema principale della versione Batched, che chiameremo SPP_{BA}, è quello di scegliere il raggio di ricerca ottimale per ogni probing location, con l'obiettivo di massimizzare la probabilità di trovare un uon oggetto da aggiungere alla selezione.
Per far questo bisogna cercare di minimizzare il limite superiore sullo score pesato degli oggetti non ancora visitati, esprimendo però il problema di ottimizzazione usando il raggio come variabile.
Il problema di minimizzazione viene dunque espresso come segue:

$$\begin{aligned} &minimize\ \tau_B(r1, ..., rv) \\ &subject\ to\ \sum_{u=1}^{V} n_u(r_u) \leqslant M_B \wedge r_u \leqslant R_u^{max} \end{aligned} \tag{3.3}$$

Però, in questo caso, stiamo ammettendo solo la possibilità di accessi per distanza, aggiungendo invece anche gli accessi per punteggio, dobbiamo

introdurre una nuova variabile $M = M_S + M_B$ che rappresenta il budget da spendere come numero totale di accessi per punteggio (M_S) e di accessi batched (M_B).
Aggiungendo questa variabile possiamo riformulare il problema di minimizzazione come segue:

$$\begin{aligned} &\text{minimize } \tau_B(r1, ..., rv, n^S) \\ &\text{subject to } \sum_{u=1}^{V} n_u(r_u) + (n^S - n^{=S}) \leqslant M \wedge r_u \leqslant R_u^{max}\ u = 1, ..., V \end{aligned} \tag{3.4}$$

Considernando una distribuzione uniforme degli oggetti nella regione possiamo stimare il raggio di ricerca di ogni probing location a seconda del budget M che ci proponiamo di utilizzare.
Nel caso invece di distribuzione non uniforme, come potrebbe essere in casi reali, possiamo usare una stima della densità in ogni zona della nostra regione. Tale stima dovrebbe essere un dato conosciuto, ed in tal caso, invece di avere nel nostro problema una sola densità, dovremo usare un vettore con le densità relative ad ogni probing location.
Se non avessimo nemmeno accesso a dati sulle densità locali, dovremo valutarle eseguendo preventivamente accessi random in probing location fasulle per stimarne i valori. In questo caso bisognerebbe però fare un trade off fra il numero di accessi preventivi (costosi in termini di sumdepth) e la qualità della stima.
Una possibile decisione per valutare M è quella di prenderlo costante in ogni ripetizione dell'algoritmo, giustificando questa scelta col fatto che non conoscendo la distribuzione iniziale, tutte le probing location sono equamente promettenti.
Quello che intendo fare col mio lavoro è estendere gli studi effettuati alla versione batched access in particolare valutando i possibili vantaggi derivanti da una scelta accurata dei valori di M ad ogni passo dell'algoritmo.

Capitolo 4

Modifiche a SPP-Batched Access

4.1 Studio Preliminare

Il primo passo del mio lavoro di studio della versione batched access, è stato ragionare sul valore del parametro M.
Come già detto M rappresenta il budget di oggetti da recuperare ad ogni accesso da una probing location. Da questo parametro e dalla stima della densità degli oggetti viene poi ricavato il raggio di ricerca.

4.1.1 M costante

In [7] M viene preso costante e uguale a 10. La scelta di un budget costante non è casuale, infatti supponendo una distribuzione uniforme degli oggetti nello spazio si può supporre che da qualsiasi punto dello spazio si effettui una ricerca, vi sia la medesima possibilità di trovare un oggetto da aggiungere alla soluzione.

Partendo da questo presupposto, ossia un valore di M costante in ogni passo dell'algoritmo, ho cercato di capire se ci fosse ed eventualmente quale fosse un valore ideale del budget.
Per far questo son partito da un'istanza base dell'algoritmo caratterizzata dai seguenti parametri:

- Numero di oggetti (N) : 10000
- Tipo di distribuzione: Uniforme
- Numero di risultati (K) : 10
- Pulling Strategy: Potential Adaptative
- Balance between relevance and diversity (λ) : 0.75

Definiti questi parametri ho racchiuso l'algoritmo in un ciclo variando ad ogni iterazione il valore di M.
L'output di una singola esecuzione di SPP-BA è la coppia di valori (sumDepths ; $F(O_K^{batched})$) ossia il numero di oggetti visitati e il valore della funzione obiettivo.
Per ottenere un risultato più parlante useremo però la differenza fra la funzioni obiettivo ottenuta da SPP-BA e quella calcolata con MMR usando gli stessi parametri iniziali:

$$\Delta F = |F(O_K^{batched}) - F(O_K^{MMR})| \tag{4.1}$$

Il risultato ottenuto da questa analisi iniziale è presentato in figura 4.1
L'asse delle x rappresenta la differenza di funzione obiettivo, mentre sull'asse y si trova la sumdephts raggiunta.
Ogni punto del grafico è il risultato di una singola esecuzione dell'algoritmo con il valore di M costante.

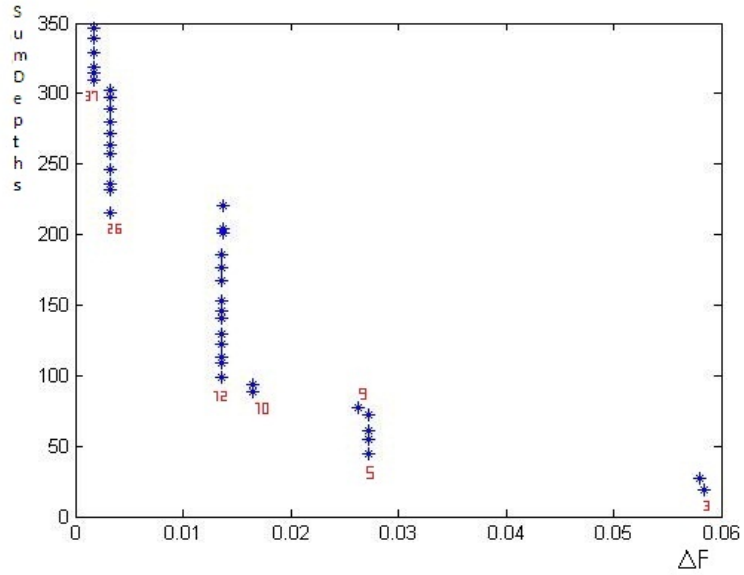

Figura 4.1: Grafico delle esecuzioni con M costante

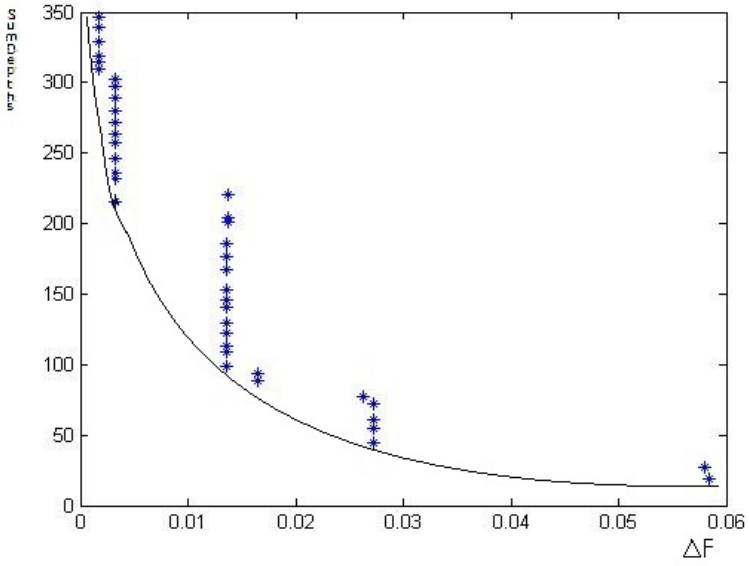

Figura 4.2: Disposizione delle M dominanti

La prima cosa che si nota ad occhio è che unendo i punti del grafico si otterrebbe una funzione a scalino. Questo vuol dire che per alcuni valori di M, cambia la sumdepths ma non il valore della funzione obiettivo.
Questo fatto implica anche che alcuni valori di M generano risultati che dominano gli altri, perciò i risultati dominati diventano inutili, infatti sono indice di peggioramento delle prestazioni.
In figura sono segnati in rosso i valori di M corrispondenti ai punti dominanti.
Un altro aspetto che salta all'occhio è che i valori dominanti sono disposti in una specie di ramo di iperbole, come si vede in figura 4.2, questo indica che non esiste un valore costante di M che sia migliore di ogni altro, ma che bisogna mediare fra una maggiore precisione della funzione obiettivo, e una minore sumdepths.

4.1.2 M variabile

Il passo seguente della mia ricerca è stato provare a far variare manualmente i valori di M nei vari passi dell'algoritmo.
Per prima cosa ho deciso di fissare il valore di M nel primo passo ad uno dei valori dominanti selezionati in precedenza, per poi eseguire di nuovo l'algoritmo facendo variare gli M in maniera costante nei passi seguenti.
Nell'esempio in figura vediamo come fissando il valore al primo passo, la disposizione dei risultati seguenti al variare della M scelta resta la stessa.
Facendo altri test (il risultato dei quali può essere visionato in appendice A) notiamo come scegliendo valori di M bassi la forma dei risultati resta la stessa ma traslata verso il basso (il che indica una minore sumdepth). Scegliendo invece valori alti di M si ottiene una traslazione verso l'asse delle ordinate, che indica un miglioramento della funzione obiettivo.
Ragionando sui risultati ottenuti da quest'indagine iniziale si apprendono

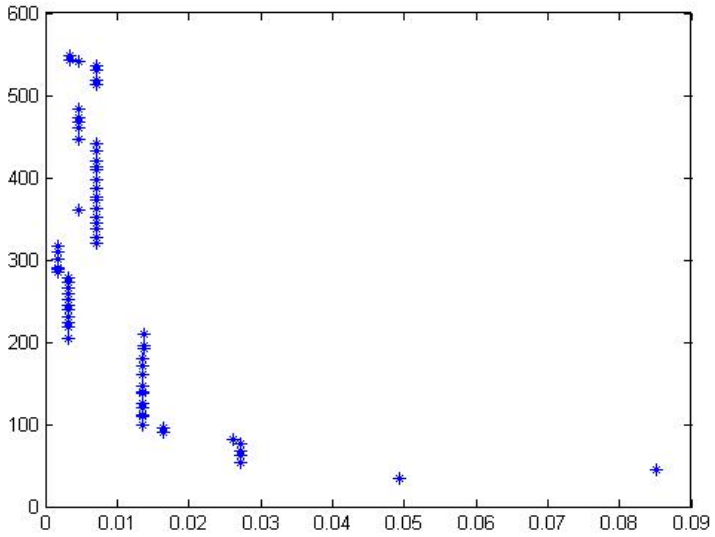

Figura 4.3: Andamento con M iniziale fisso a 10

alcune informazioni importanti sull'uso dell'accesso batched.
La prima questione è che il grafico dei risultati al variare di M si muove a 'scalino', la seconda è che fissando di volta in volta il valore di M nei vari passi dell'algoritmo la forma della distribuzione dei risultati non cambia.
La motivazione di questo comportamento è che in un determinato passo dell'algoritmo, con un determinato valore di M, si riesce a trovare un oggetto da aggiungere alla soluzione finale, che quindi abbassa drasticamente la soglia.
Scegliendo un valore di M più piccolo anche di una sola unità, in quel passo, quell'oggetto viene perso, quindi il valore della funzione obiettivo ne risente molto.
D'altro canto, scegliendo un valore di qualche unità piu alta, non viene selezionato in quel passo nessun oggetto migliore di quello già trovato, quindi si peggiora il valore della sumdepths senza migliorare la funzione obiettivo.
Quindi i valori di M dominanti trovati dopo la prima parte dell'esperimen-

to, sono valori che determinano un miglioramento della funzione obiettivo in almeno uno dei passi dell'algoritmo.
Questo è un risultato molto interessante, ma purtroppo è una valutazione che è possibile fare solo a posteriori, infatti solo analizzando i risultati delle varie esecuzioni di SPPBA si possono individuare questi valori dominanti, inoltre cambiando dataset, i valori ovviamente non sono più gli stessi.

4.2 Funzioni di M

Abbandonata l'idea di utilizzare i punti dominanti come valori di M dell'algoritmo, possiamo cercare di far variare M secondo una determinata funzione, per vedere se questo possa portare o meno ad un miglioramento delle prestazioni.
Il primo problema che ci si presenta, è valutare la natura di questa funzione.
Ad una prima analisi si potrebbe pensare che siano da preferirsi funzioni decrescenti. Infatti, assumendo un valore di M grande nei primi passi dell'algoritmo, ci si assicura una migliore copertura dello spazio, mentre negli ultimi passi può essere sufficiente usare un valore di M basso, per coprire i 'buchi' non ancora considerati.
In realtà, analizzando i dati estratti durante le prove con M costante, si evince un ulteriore dato: avere un valore basso di M nei primi passi dell'algoritmo in molti casi non peggiora la funzione obiettivo, ma riduce la sumdepths, migliorando quindi i risultati dell'algoritmo.
Quindi, dai dati in nostro possesso, scegliamo di concentrarci soprattutto su funzioni crescenti.
Quello che manca però è un sistema univoco per valutare la bontà di una determinata funzione di M, ossia un metodo che, dati i risultati ottenuti

usando nell'algoritmo due differenti funzioni di M, determini quale delle 2 funzioni ha prodotto un risultato migliore.
Per fare questo abbiamo la necessità di definire una Funzione di Qualità che indichi la 'bontà' di una determinata soluzione trovata.

4.2.1 Funzione di Qualità

Per definire in maniera univoca quando, una determinata soluzione, sia migliore di un'altra abbiamo bisogno di costruire una funzione che prenda in input i risultati dell'algoritmo, ossia la coppia (ΔF, Sumdepth), e fornisca in output un valore in $\mathbb{R}$.
Definiamo quindi la Qualità Q come:

$$Q = F(\Delta F, Sumdepth) \tag{4.2}$$

Il problema fondamentale di questa funzione di qualità è che dai dati che abbiamo trovato fin'ora, non è semplice definire quando un risultato è migliore di un altro, infatti le soluzioni migliori del caso costante erano posizionate su un ramo di iperbole. Questi valori erano sicuramente migliori di quelli da loro dominati, ma fra di loro risultavano equivalenti ai fini dell'algoritmo.
Dobbiamo quindi trovare una funzione di qualità che fornisca risultati equivalenti o quasi equivalente per i valori dominanti di M e valori peggiori per tutti i valori dominati.
Possiamo risolvere parzialmente il nostro problema usando proprio i dati ottenuti, per interpolare la nostra funzione di M.
La scelta effettuata è stata quindi di usare le coppie (ΔF, Sumdepth) trovate con i valori dominanti di M per definire una correlazione fra questi dati. Per far questo è stato utilizzato un algoritmo di interpolazione in matlab.
La funzione trovata è la seguente:

$$Q = 100 * \Delta F + \frac{4K * Sumdepth^2}{N} - 0.0112 * Sumdepth - \frac{17.152}{K}; \quad (4.3)$$

Questa funzione è stata trovata partendo dai dati di una specifica istanza dell'algoritmo. Cambiando i parametri, i coefficienti andrebbero ricalcolati, in particolare al cambiare di λ.
Tuttavia, questa stessa funzione di qualità, è stata testata ed utilizzata in molte altre istanze, fornendo dati che, sebbene non siano precisi come potrebbero esserlo con una funzione calcolata ad hoc per quella determinata istanza, indicano comunque in maniera adeguata il miglioramento o il peggioramento di un risultato rispetto ad un'altro.
La nostra funzione per come è stata definita fornirà un valore in $\mathbb{R}$ interpretabile come segue:

- $Q > 0$ cattiva soluzione, più alto è il valore di Q peggiore è il risultato
- $Q = 0$ soluzione accettabile, $Q = 0$ è un risultato che si avvicina ai punti dominanti della funzione costante
- $Q < 0$ buona soluzione, prestazioni migliori della versione costante, tanto più basso è il valore di Q tanto migliore è la soluzione

Per verificare l'adeguatezza della funzione ho provato ad applicarla ai risultati dominanti ottenuti con una M costante, ottenendo valori vicino allo 0, usando invece valori non ottimali di M si ottiene una $Q > 0$.

4.2.2 Funzioni crescenti

Una volta stabilita la funzione di qualità possiamo cominciare a testare diverse classi di funzioni crescenti (uniforme, esponenziale, logaritmica etc...).
Il primo test è stato fatto con funzioni crescenti in maniera uniforme del

tipo :

$$M = a * K \tag{4.4}$$

Dove K è il passo attuale dell'algoritmo. I risultati ottenuti al variare di a sono i seguenti:

a	Q
1	1.63
2	2.67
3	6.96
4	15.20
...	...
10	93.83

Come si vede i risultati peggiorano notevolmente al crescere di a, infatti si ottengono valori sempre più alti della sumdepth senza un grande miglioramento di funzione obiettivo.
Tentando invece con una funzione esponenziale del tipo:

$$M = a^K \tag{4.5}$$

I risultati ottenuti sono i seguenti:

a	Q
1.2	7.2
1.3	3.82
1.4	3.88
2	1703.06

Le prestazioni peggiorano notevolmente e rapidamente, sempre per lo stesso motivo di prima, all'aumentare del valore di M la sumdepth aumenta velocemente, ma il variare di ΔF risulta invece minimo.

Vediamo ora i risultati di una funzione logaritmica:

$$M = \log_a(K) \tag{4.6}$$

i risultati sono i seguenti:

a	Q
1.1	6.11
1.2	1.19
1.3	-0.053
1.4	1.36
1.5	1.66

Come si vede abbiamo ottenuto dei risultati molto incoraggianti.
Ragionando sul motivo di questo miglioramento sono giunto alla conclusione che probabilmente una crescita rapida nei primi passi dell'algoritmo permette di raggiungere subito livelli ottimali del valore di M, mentre rallentando la crescita nei passi successivi si evita che la sumdepth peggiori di molto.
Partendo dunque da questo tipo di funzione di M possiamo trovare dei parametri ideali e verificare i risultati ottenuti nei vari dataset e cambiando il valore dei parametri in input.

Capitolo 5

Software

5.1 Software Utilizzato

La prima versione dell'algoritmo SPP, nella sua forma originale,è stato scritto in MatLab, per avere uno strumento efficace di gestione grafica dei risultati, di risoluzione di problemi e di creazione e gestione dei diagrammi di Voronoi.
In Matlab sono disponibili infatti alcuni strumenti come CVX o QHull: applicativi di terze parti importabili nel progetto, che permettono di eseguire tutte le funzioni necessarie all'esecuzione di SPP.
Nel dettaglio CVX [9] è un sistema di modellazione per risolvere problemi di ottimizzazione convessa. CVX permette di specificare la funzione obiettivo e i vincoli utilizzando la sintassi MATLAB per la scrittura delle espressioni.
QHull [10] è invece un applicativo che fornisce diverse funzioni di calcolo geometrico, quali la costruzione di inviluppi convessi, triangolazioni di Delaunay e diagrammi di Voronoi. Le funzionalità di QHull sono utilizzate da molti software, tra i quali appunto MATLAB.

La versione matlab è stata utilizzata fino a quando Catallo e Ciceri, nel loro lavoro di tesi, hanno modificato la logica batched access e hanno cambiato il codice per integrarlo al progetto SeCo [11]. SeCo, abbreviativo di Search Computing, è un progetto di ricerca portato avanti all'interno del Politecnico di Milano che propone un nuovo paradigma per la risoluzione di query multi-dominio basato sulla combinazione di dati estratti da sorgenti distinte e sulla loro successiva integrazione per mezzo di engine specializzati.
Per effettuare l'integrazione di SPP in SeCo, è stato necessario replicare il codice MatLab in Java, adattandolo alle caratteristiche e potenzialità del nuovo linguaggio.

5.2 Versione pre-esistente

Il codice precedente di SPP è stato implementato sia in matlab che in Java. La versione Matlab, creata per la prima versione dell'algoritmo, presenta al suo interno sia il codice di SPP che il codice di MMR per poter confrontare i risultati dei due algoritmi.
All'inizio del codice sono presenti molti parametri settabili per definire varie modalità d'esecuzione del codice. Vengono definite la dimensione del dataset, la tipologia di distribuzione dei dati, il peso da dare agli accessi score-based e distance-based, l'utilizzo di dati sintetici o di dati reali presi da Nestoria, e altri parametri funzionali oltre a parametri di visualizzazione come la generazione di grafici dinamici che mostrano la generazione dei diagrammi di Voronoi passo per passo o la generazione di un video dell'evoluzione di questi dati.
La parte sucessiva del codice è l'inizializzazione di tutte le strutture dati necessarie all'esecuzione degli algoritmi; è qui che vengono inizializzati il dataset e i grafici del diagramma di Voronoi. Dopodichè vengono lanciate

in sequenza tutte le istanze degli algoritmi sviluppate e i risultati vengono salvati per poterli confrontare.
Durante l'esecuzione degli algoritmi vengono richiamati gli applicativi di supporto (CVX e QHull) per risolvere i problemi di programmazione lineare e per calcolare i diagrammi di Voronoi.
Per la realizzazione della versione Java dell'algoritmo è stata necessaria una re-ingegnerizzazione dell'intero codice. Il primo passo infatti è stato rendere la struttura dati dell'algoritmo compatibile con una programmazione ad oggetti. Per questo motivo sono stati individuati gli attori principali coinvolti in SPP, soprattutto per la gestione dei diagrammi di Voronoi. Sono state dunque create classi per descrivere celle, centroidi, edge, vertici, etc...
Una volta definiti gli oggetti col quale lavorare sono stati creati i metodi per la computazione delle varie operazioni da svolgere durante l'esecuzione: creare il dataset, calcolare il valore della funzione obiettivo, calcolare il raggio di esecuzione ad ogni passo, confrontare il punteggio di ogni elemento nuovo con quelli già trovati, calcolare la soglia e cosi via tutte le funzioni principali e tutte le funzioni di supporto per implementare lo pseudocodice mostrato nel capitolo 3.
Nel Main del codice viene replicata sostanzialmente la stessa logica del codice MatLab, infatti vengono settati i parametri da utilizzare nell'algoritmo, vengono inizializzate le strutture date e vengono lanciate in sequenza i due algoritmi da valutare, che in questo caso sono SPP e SPP_BA.
Anche nella versione Java i risultati vengono salvati per poterli confrontare.

5.3 Modifiche Apportate

Durante il mio lavoro la maggior parte delle modifiche sono state apportate al codice Java.

Innanzitutto è stato necessario racchiudere il core dell'algoritmo all'interno di un ciclo per permettere di impostare di volta in volta un valore di M differente. Inoltre anche l'output dell'algoritmo è stato modificato, perchè più che ottenere i valori intermedi ad ogni passo, ci servono i risultati finali di ogni istanza dell'algoritmo.
Bisogna considerare poi che il risultato della funzione obiettivo non è un dato molto parlante per fare un confronto fra diverse esecuzioni dell'algoritmo, per questo è stato dichiarata la nuova variabile

```
1 double delta = FOMmr-FOSpp;
```

Questa variabile rappresenta meglio le diverse prestazioni del sistema rispetto alla versione normale di SPP.
Per tenere traccia di tutte le esecuzioni dell'algoritmo è stato creata la variabile delta, definita come

```
1 String deltas = deltas + delta + " ";
```

deltas è una stringa che viene riempita iterativamente inserendovi di volta in volta i risultati delle esecuzioni dell'algoritmo.
Ovviamente a questo punto, anche per la sumdepth è stata creata una variabile stringa da popolare in modo da avere dati allineati.

```
1 String sumdephts = sumdephts + sd + " ";
```

Tramite questi 2 dati possiamo calcolare il valore della funzione di qualità come:

```
1 quality = 100*delta + (4*K/N)*sd*sd - 0.0112*sd - (17.152/K);
```

Con queste strutture possiamo valutare ogni tipo di funzione di M, ma ora dobbiamo capire dove definire le funzioni da utilizzare.
Le funzioni di M sono state definite all'interno di un metodo che dato il passo corrente dell'algoritmo restituisce il valore di M per quel passo. La logica della funzione è la seguente:

```
1  private double computeNewBudget(int k) {

3  double M;
   if (budgetType.equals(FUNCTION)) {
5
          M= logbase(1.3, k);
7  //      M=10*k;
   //      M=(k*k)/2;
9  //      M=Math.round(Math.exp(k));
   //      M=Math.round(Math.pow(1.3, k));
11
                  }
13 return M;
   }
```

Di volta in volta si può scegliere la funzione adatta decommentando il pezzo di codice.

Un'altra modifica al apportata è stata generare il codice per cambiare la struttura del dataset da normale ad esponenziale, e definire le costanti per permetterci di cambiare la struttura facilmente prima dell'esecuzione dell'algoritmo.

```
       if(type.equals(TIPOESPONENZIALE))
2              createSyntheticDataExponential(dimension);
       else createSyntheticData(dimension);
```

La logica delle funzioni createSyntheticDataExponential e createSyntheticData è presentata in Appendice B
Con queste semplici modifiche è stato possibile fare tutte le prove che servivano. I risultati però sono stati valutati usando di nuovo MatLab, per poter gestire in maniera agile la rappresentazione visiva dei dati che abbiamo trovato.

Capitolo 6

Risultati

Una volta modificato il codice per poter utilizzare in maniera estensiva la versione Batched Access di SPP, possiamo valutare la bontà della funzione di M trovata facendo variare i parametri iniziali piu importanti.
Cambiando di volta in volta le condizioni di esecuzione dell'algoritmo creiamo sempre nuove istanze dello stesso, ottenendo quindi nuovi risultati.
Per questo è importante testare la funzione trovata tenendo in considerazione tutte le possibili variazioni di ambiente, perchè ad esempio una funzione che si comporta benissimo in una determinata istanza, potrebbe avere risultati pessimi per altre istanze dell'algoritmo.
Vediamo quindi quali sono i risultati ottenuti facendo variare i vari parametri.

6.1 Variazione Dataset

Nella nostra implementazione dell'algoritmo, sono disponibili due possibili distribuzione degli oggetti nel dataset:

- Distribuzione Uniforme
- Distribuzione Esponenziale.

Vediamo ora i risultati ottenuti usando le diverse tipologie di funzioni nei due casi. Le seguenti prove sono state eseguite con i parametri iniziali:

- N = 10000
- $\lambda = 0.75$
- K = 10

6.1.1 Distribuzione Uniforme

Tipo Funzione	F(M)	Q
Costante	M = 10	2.1107
Lineare	M = 2K	2.6718
Polinomiale	$M = \frac{K^2}{2}$	11.3753
Esponenziale	$M = e^{\frac{K}{2}}$	34.805
Esponenziale $\neq e$	$M = 1.3^K$	3.8175
Logaritmica	$M = \log_{1.3} K$	-0.0539

Come si vede una funzione logaritmica è, fra quelle provate, quella con il risultato più promettente. Di ciascuna forma sono stati provati diversi parametri fino ad arrivare ai valori che fornivano i risultati migliori.

6.1.2 Distribuzione Esponenziale

Tipo Funzione	F(M)	Q
Costante	M = 10	1.7890
Lineare	M = 2K	3.2168
Polinomiale	$M = \frac{K^2}{2}$	6.5173
Esponenziale	$M = e^{\frac{K}{2}}$	24.3405
Esponenziale $\neq e$	$M = 1.3^K$	2.8867
Logaritmica	$M = \log_{1.3} K$	1.10

Anche per il caso esponenziale la funzione logaritmica è quella che si è

comportata meglio fra tutte. Un appunto sul caso esponenziale è che per costruzione della funzione di generazione del dataset, gli oggetti vengono distribuiti in maniera randomica ogni volta differente all'interno dello spazio, ottenendo cosi dei punteggi diversi ad ogni iterazione. Per questo i valori segnati sono stati calcolati ciascuno come media di 10 esecuzioni.

6.2 Variazione n^o di iterazioni

Il numero di oggetti da restituire come risultato dell'algoritmo, influisce molto sulle prestazioni, infatti per trovare più oggetti sono necessari più passi dell'algoritmo e quindi aumenterà inevitabilmente la sumdepth. Vediamo i risultati ottenuti dalle varie funzioni cambiando il numero di oggetti finali.
I parametri iniziali sono gli stessi di prima, nel caso di distribuzione uniforme.

6.2.1 K=15

Tipo Funzione	F(M)	Q
Costante	M = 10	3.1587
Lineare	M = 2K	13.2912
Polinomiale	$M = \frac{K^2}{2}$	93.3619
Esponenziale	$M = e^{\frac{K}{2}}$	3073.82
Esponenziale $\neq e$	$M = 1.3^K$	70.6315
Logaritmica	$M = \log_{1.3} K$	3.0571

6.2.2 K=20

Tipo Funzione	F(M)	Q
Costante	M = 10	6.0211
Lineare	M = 2K	26.8665
Polinomiale	$M = \frac{K^2}{2}$	398.700
Esponenziale	$M = e^{\frac{K}{2}}$	Too Big
Esponenziale $\neq e$	$M = 1.3^K$	1706.62
Logaritmica	$M = \log_{1.3} K$	4.3626

Possiamo notare come, anche variando il numero di iterazioni, la funzione logritmica si comporta in media meglio di ogni altra configurazione.

6.3 Variazioni λ

Un'altra variabile da far cambiare è λ. Essa rappresenta il peso da dare al risultato di diversificazione rispetto allo score. Un valore di $\lambda = 0$ equivale ad una funzione obiettivo determinata solo dalla componente di score, mentre un valore = 1 corrisponde ad una funzione obiettivo determinata solo dalla componente di diversificazione.
Diversi valori di λ sono stati testati.

6.3.1 $\lambda = 0.125$

Tipo Funzione	F(M)	Q
Costante	M = 10	2.5487
Lineare	M = 2K	3.2649
Polinomiale	$M = \frac{K^2}{2}$	15.5779
Esponenziale	$M = e^{\frac{K}{2}}$	41.4458
Esponenziale $\neq e$	$M = 1.3^K$	8.8257
Logaritmica	$M = \log_{1.3} K$	4.7634

6.3.2 $\lambda = 0.25$

Tipo Funzione	F(M)	Q
Costante	M = 10	0.0781
Lineare	M = 2K	2.758
Polinomiale	$M = \frac{K^2}{2}$	14.4113
Esponenziale	$M = e^{\frac{K}{2}}$	41.3563
Esponenziale $\neq e$	$M = 1.3^K$	6.6056
Logaritmica	$M = \log_{1.3} K$	2.6456

6.3.3 $\lambda = 0.5$

Tipo Funzione	F(M)	Q
Costante	M = 10	5.0950
Lineare	M = 2K	5.6785
Polinomiale	$M = \frac{K^2}{2}$	12.7838
Esponenziale	$M = e^{\frac{K}{2}}$	36.8468
Esponenziale $\neq e$	$M = 1.3^K$	7.2386
Logaritmica	$M = \log_{1.3} K$	2.6456

6.3.4 $\lambda = 0.875$

Tipo Funzione	F(M)	Q
Costante	M = 10	1.2163
Lineare	M = 2K	1.8033
Polinomiale	$M = \frac{K^2}{2}$	8.4385
Esponenziale	$M = e^{\frac{K}{2}}$	35.2458
Esponenziale $\neq e$	$M = 1.3^K$	1.1991
Logaritmica	$M = \log_{1.3} K$	-1.7007

Scopriamo da queste prove che la funzione logaritmica si comporta bene con tutte i valori di λ testati, ma in alcuni casi non risulta la scelta migliore.

In particolare per valori bassi di λ la funzione logaritmica ha comportamenti peggiori di un valore costante. Mentre quando il valore di λ cresce, e quindi il risultato di diversificazione comincia ad acquistare valore, la funzione trovata torna a mostrare risultati migliori delle altre funzioni.

6.4 Variazione Dimensione Dataset

L'ultimo parametro da modificare è la dimensione del DataSet, ossia il numero di oggetti totali presenti nello spazio considerato. Aumentando questo parametro aumenta la densità degli oggetti, quindi con uno stesso valore di M si vaglia una porzione minore dello spazio. Questi sono i risultati ottenuti dalle varie prove.

6.4.1 N=1000

Tipo Funzione	F(M)	Q
Costante	M = 10	0.8339
Lineare	M = 2K	1.1325
Polinomiale	$M = \frac{K^2}{2}$	11.7547
Esponenziale	$M = e^{\frac{K}{2}}$	33.7887
Esponenziale $\neq e$	$M = 1.3^K$	0.5693
Logaritmica	$M = \log_{1.3} K$	-1.2628

6.4.2 N=5000

Tipo Funzione	F(M)	Q
Costante	M = 10	1.4359
Lineare	M = 2K	3.2279
Polinomiale	$M = \frac{K^2}{2}$	14.5940
Esponenziale	$M = e^{\frac{K}{2}}$	36.903
Esponenziale $\neq e$	$M = 1.3^K$	3.0414
Logaritmica	$M = \log_{1.3} K$	1.2311

6.4.3 N=20000

Tipo Funzione	F(M)	Q
Costante	M = 10	3.7185
Lineare	M = 2K	4.384
Polinomiale	$M = \frac{K^2}{2}$	11.8289
Esponenziale	$M = e^{\frac{K}{2}}$	35.0697
Esponenziale $\neq e$	$M = 1.3^K$	5.7571
Logaritmica	$M = \log_{1.3} K$	2.929

6.4.4 N=30000

Tipo Funzione	F(M)	Q
Costante	M = 10	2.5487
Lineare	M = 2K	3.4574
Polinomiale	$M = \frac{K^2}{2}$	12.7745
Esponenziale	$M = e^{\frac{K}{2}}$	34.5774
Esponenziale $\neq e$	$M = 1.3^K$	6.7078
Logaritmica	$M = \log_{1.3} K$	2.2220

Anche variando la dimensione del dataset la funzione logaritmica resta la migliore fra quelle testate.

6.5 Variazioni Percentuali

Abbiamo visto quindi, che in generale, applicare una funzione di tipo logaritmico invece di usare un valore costante di M, porta ad un miglioramento delle prestazioni, ma vediamo di quantificare questo miglioramento in termini di percentuali:

Casistica	M costante	M logaritmica	Miglioramento
N = 10000	sd = 89	sd = 50	-43.8%
K = 10	$\Delta = 0.01654$	$\Delta = 0.0122$	-0.4%
N = 5000	sd = 80	sd = 76	-5.0%
K = 10	$\Delta = 0.0148$	$\Delta = 0.0148$	-0.00%
N = 20000	sd = 95	sd = 68	-28.4%
K = 10	$\Delta = 0.01654$	$\Delta = 0.0122$	+0.43%
N = 10000	sd = 146	sd = 88	-39.72%
K = 15	$\Delta = 0.0112$	$\Delta = 0.0025$	+9.71%
N = 10000	sd = 198	sd = 131	-33.83%
K = 20	$\Delta = 0.01255$	$\Delta = 0.0355$	+2.26%

Quindi si vede che in media, su un campione di dati ottenuto variando il maggior numero possibile di parametri, la versione si SPP-BA basata su aggiornamento della soglia di tipo logaritmico, contribuisce a diminuire il numero di oggetti visitati in media del 25.128% comportando una differenza di funzione obiettivo rispetto a MMR del $+2.24\%$ Questi dati, che sono molto positivi, vanno però ridimensionati considerando che SPP è nato per confrontarsi con MMR.

MMR per la sua esecuzione accede a tutti gli oggetti presenti nel dataset, mentre SPP solo ad una piccola porzione: in particolare nel caso con N=10000 SPP accede solo all' 1% degli oggetti, mantenendo però funzioni obiettivo comparabili. Quindi il miglioramento introdotto dal nostro lavoro va ridimensionato su questo dato dicendo che, con l'applicazione di una

funzione sulla quality il numero di oggetti visitati rispetto a MMR è dello 0.7487%, mantenendo una funzione obiettivo paragonabile a quella di MMR.
Sebbene il guadagno apportato dall'utilizzo di una funzione di M sia piccolo, dal punto di vista computazionale esso richiede uno sforzo aggiuntivo davvero irrisorio, rendendo questo miglioramento praticamente gratuito

Capitolo 7

Conclusioni e sviluppi futuri

7.1 Conclusioni

L'obiettivo del mio lavoro di tesi è stato quello di studiare la versione Batched Access di SPP per esplorare tutte le potenzialità di questa nuova metodologia di accesso agli oggetti.
Il lavoro è stato diviso in più fasi, la prima delle quali è stata una fase di studio della versione pre-esistente di SPP-BA per capire in che modo intervenire per migliorarne le prestazioni.
Da questo studio è emerso che il problema di diversificazione deve rispondere a due requisiti importanti, ossia minimizzare il numero di oggetti visitati e ottenere una valore della funzione obiettivo il piu grande possibile. Purtroppo queste 2 condizioni devono essere mediate, infatti si è visto che non esiste un risultato migliore in assoluto, ma una serie di risultati che dominano gli altri in quanto a prestazioni, ma che sono fra loro equiparabile. E' poi possibile scegliere fra queste soluzioni dominanti quella che più si adatta all'ambito nel quale si sta lavorando, ad esempio preferendo una soluzione più accurata o un numero di oggetti visitati minori.

La seconda fase del lavoro è consistita nello studio del valore di M, ossia il budget di accessi da spendere ad ogni passo dell'algoritmo.
M è stato fatto variare all'inizio in maniera costante, poi seguendo determinate funzioni, per vedere quale configurazione forniva i risultati migliori.
Da questo studio è stato mostrato che funzioni crescenti di M in generale si comportano meglio di funzioni decrescenti o costanti.
In particolare è stato notato che una funzione di tipo logaritmico è quella che fornisce i risultati migliori fra le varie funzioni crescenti testate. Questo perchè fare aumentare il valore di M velocemente all'inizio per poi rallentarne la crescita, permette di coprire maggiormente lo spazio durante i primi passi dell'algoritmo, diminuendo in seguito il numero di oggetti visitati.
Per verificare quanto scoperto, sono state effettuate varie prove cambiando i parametri d'esecuzione dell'algoritmo. Tutte queste prove hanno confermato che una funzione logaritmica di M contribuisce effettivamente ad ottenere buoni valori della funzione obiettivo diminuendo sensibilmente il numero di oggetti visitati.

7.2 Sviluppi futuri

Le possibili direzioni future sull'argomento trattato in questo lavoro di tesi sono molteplici.
Innanzitutto nel mio lavoro ho mostrato come una funzione logaritmica di M possa migliorare le prestazioni di SPP-BA, ma la mia funzione è stata calcolata da dati sperimentali ed è poco generica. Un primo miglioramento potrebbe essere quello di generalizzare questa funzione parametrizzandone i coefficienti a seconda delle variabili utilizzate per ogni istanza dell'algoritmo.

Una successiva direzione di ricerca potrebbe essere quella di applicare un'accesso di tipo batched anche ad altri algoritmi di differenziazione geografica, per vedere se anche in quei casi, questo tipo di accesso ai dati, possa portare un beneficio.
Inoltre, concentrandoci maggiormente sulla versione generica di SPP, si può cercare un miglioramento dell'euristica per la stima della densità degli oggetti appartenenti ad una popolazione reale, perchè la versione attuale non è efficace per aree di prelievo oggetti molto estese, in quanto la densità degli oggetti risulta diversa nelle varie sottoaree.

Bibliografia

[1] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. ACM Comput. Surv., University of Waterloo, 2008.

[2] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In WWW'09, pages 381-390, 2009.

[3] Carbonell, J. and Goldstein, J. 1998. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In SIGIR'98. 335-336.

[4] Van Kreveld Et Al., M. J. 2005. Multi-dimensional scattered ranking methods for geographic information retrieval. GeoInformatica 9, 1, 61-84.

[5] Piero Fraternali, Davide Martinenghi, Marco Tagliasacchi: Top-k bounded diversification. SIGMOD Conference 2012: 421-432

[6] I. Catallo, E. Ciceri, P. Fraternali, D. Martinenghi, M. Tagliasacchi, Top-k Diversity Queries over Bounded Regions, ACM Transactions on Database Systems, to appear.

[7] I. Catallo, E. Ciceri, Diversificazione dei risultati di ricerca in presenza di limiti di accesso, Master graduation thesis, Politecnico di Milano

[8] Wikipedia. Voronoi Diagram | wikipedia, The Free Encyclopedia, 2013. [Online; accessed 20-may-2013].

[9] http://cvxr.com/cvx/doc/

[10] Q-hull. http://www.qhull.org/.

[11] http://www.search-computing.it/

[12] http://www.statisticbrain.com/google-searches/

Appendice A

Test M costante

Come descritto nel capitolo 4, sono stati fatti vari test fissando i valori di M nei vari passi dell'algoritmo per studiare il comportamento della funzione obiettivo e della sumdepths al variare di M.
Qui di seguito mostriamo alcuni dei risultati ottenuti.

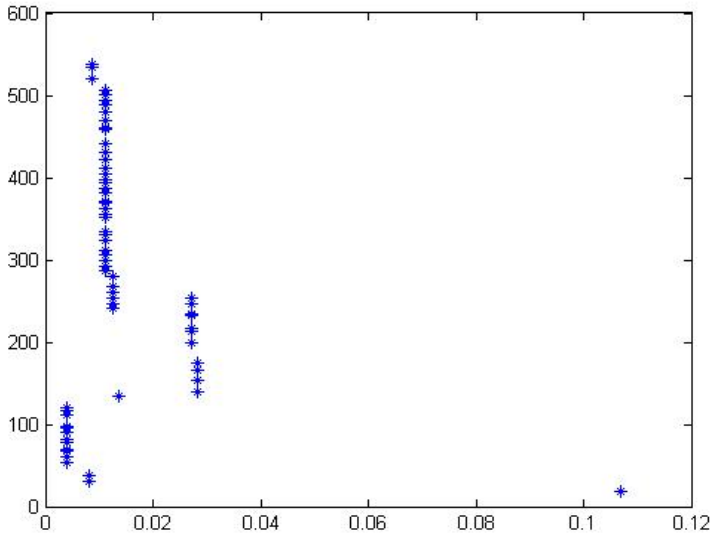

Figura A.1: Andamento con M iniziale fisso a 3

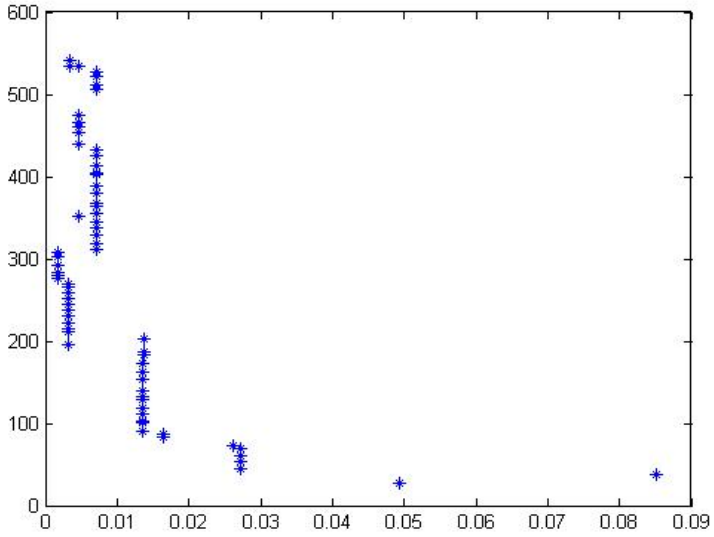

Figura A.2: Andamento con M iniziale fisso a 5

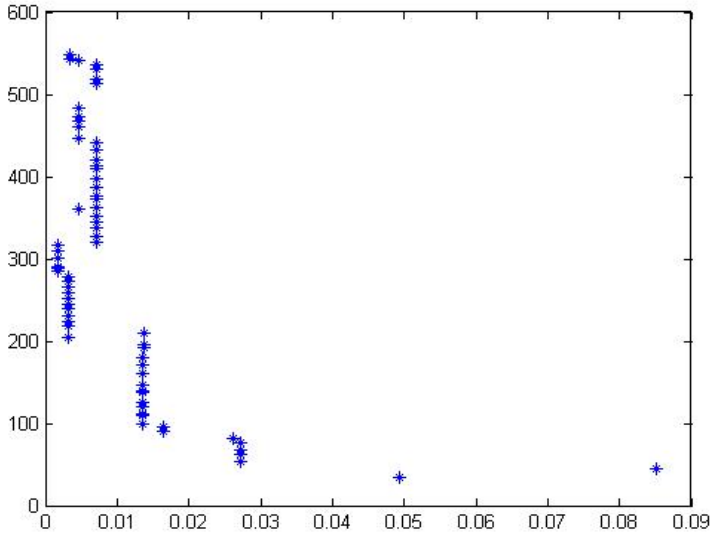

Figura A.3: Andamento con M iniziale fisso a 12

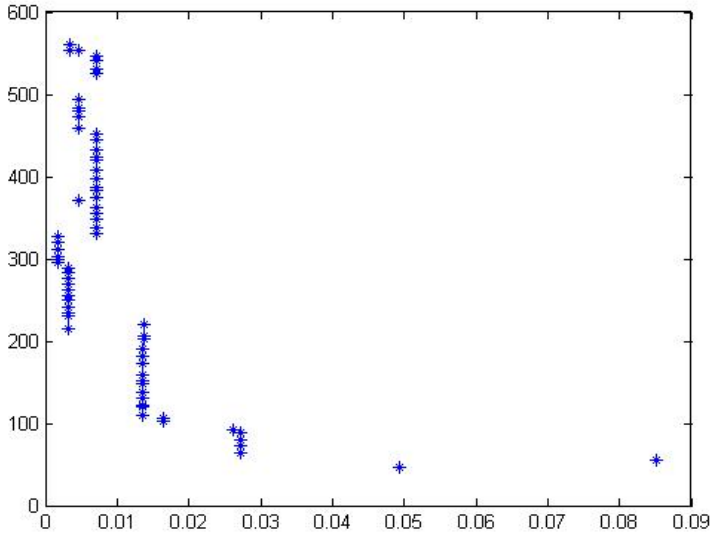

Figura A.4: Andamento con M iniziale fisso a 26

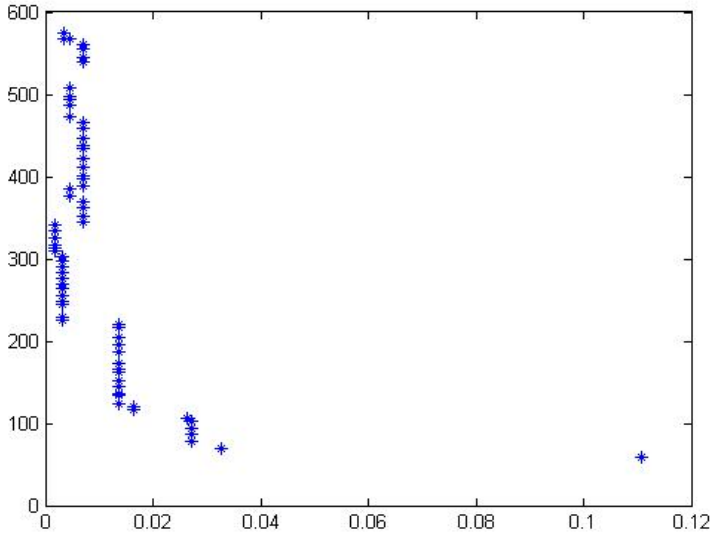

Figura A.5: Andamento con M iniziale fisso a 37

Appendice B

Funzioni Implementate

Elenchiamo ora il codice di alcune delle funzioni implementate durante il lavoro di tesi.

FUNZIONE createSyntheticData

```
1       /**
         * Method that enables the construction of synthetic data
3        */
       private void createSyntheticData(int dimension) {
5              // Variables
               double upperBound = 0.5;
7
               // Construction of the objects
9              objects = new ArrayList<Centroid>();
               for (int i = 0; i < N; i++) {
11                     double[] coordinates = new double[dimension];

13                     for (int j = 0; j < dimension; j++) {
                               coordinates[j] = (randGen.nextDouble() -
                                   upperBound);
15                     }
                       double score = randGen.nextDouble();
17
                       if (dimension == 2)
```

```
19                                  objects.add(new Centroid2D(coordinates,
                                        score, i + 1));
                            else
21                                  objects.add(new Centroid3D(coordinates,
                                        score, i + 1));
                    }
23                  System.out.println("Synthetic objects created. Number of
                        objects: "
                                    + objects.size());
25                  flushObjectsToFile();
            }
```

Questa funzione genera una distribuzione uniforme di dati all'interno del nostro spazio e gli assegna un valore di score casuale da 0 a 1

FUNZIONE createSyntheticData

```
/**
2    * Method that enables the construction of synthetic exponential data
     */
4           private void createSyntheticDataExponential(int dimension) {
                    // Variables
6                   double upperBound = 0.5;

8                   // Construction of the objects
                    objects = new ArrayList<Centroid>();
10                  for (int i = 0; i < N; i++) {
                            double[] coordinates = new double[dimension];
12
                            for (int j = 0; j < dimension; j++) {
14                                  double expA;
                                    do {
16                                          expA = StdRandom.exp(1);
                                    } while (expA > 1);
18
                                    coordinates[j] = (expA - upperBound);
20                          }
                            double score = randGen.nextDouble();
22
```

```
                    if (dimension == 2)
                        objects.add(new Centroid2D(coordinates,
                            score, i + 1));
                    else
                        objects.add(new Centroid3D(coordinates,
                            score, i + 1));
                }
                System.out.println("Synthetic objects created. Number of
                    objects: "
                        + objects.size());
                flushObjectsToFile();
            }
```

Questa funzione si comporta come quella di prima, ma si basa su una distribuzione esponenziale ricavata con la chiamata expA = StdRandom.exp(1); che funziona cosi:

```
    /**
     * Return a real number from an exponential distribution with rate
         lambda.
     */
    public static double exp(double lambda) {
        return -Math.log(1 - uniform()) / lambda;
    }
```