Bollettino
Unione Matematica Italiana

Sezione A – La Matematica nella Società e nella Cultura

Diego Reforgiato Recupero

Strutture dati e algoritmi per problemi di ottimizzazione e di ricerca in spazi metrici e grafi

Bollettino dell'Unione Matematica Italiana, Serie 8, Vol. **8-A**—*La Matematica nella Società e nella Cultura* (2005), n.3-1, p. 621–624.

Unione Matematica Italiana

<http://www.bdim.eu/item?id=BUMI_2005_8_8A_3-1_621_0>

Bollettino dell'Unione Matematica Italiana, Unione Matematica Italiana, 2005.

Bollettino U. M. I.
La Matematica nella Società e nella Cultura
Serie VIII, Vol. VIII-A, Dicembre 2005/1, 621-624

Strutture dati e algoritmi per problemi di ottimizzazione e di ricerca in spazi metrici e grafi

DIEGO REFORGIATO RECUPERO

1. – Antipole Tree: Una nuova struttura dati per effettuare il Clustering su spazi metrici e per risolvere efficientemente due noti problemi di ricerca in una base di dati.

Il Clustering su spazi metrici è un problema ben noto in Informatica. Dati dei punti in uno spazio metrico, spesso ad alte dimensioni, il problema consiste nel raggruppare i punti in un certo numero di clusters, ognuno contenente gli elementi più simili tra loro in base in funzione alla metrica utilizzata. Si può definire nella seguente maniera:

DEFINIZIONE 1. – *Dato un insieme S e un numero intero k,* ***il problema del Clustering*** *consiste nel trovare una partizione di punti in k classi tale che una certa funzione è minimizzata. Il seguente elenco mostra delle possibili definizioni per tale funzione:*

- *la somma delle distanze degli oggetti del cluster al rappresentante (k-mediana),*
- *la massima distanza tra gli oggetti in un dato cluster (k-centri),*
- *la somma delle distanze tra i punti dello stesso cluster (k-clustering).*

Il clustering appartiene ad una classe di problemi, conosciuti come problemi di prossimità, che riguarda la distanza tra oggetti. Altri problemi che ricadono in tale classe sono il problema della range search e il problema della k-nearest neighbor search.

DEFINIZIONE 2. – *Dato un oggetto query q, un database di oggetti S, ed una soglia t, il problema della* **range search** *è quello di cercare tutti gli elementi $\{o \in S \mid dist(o, q) \leq t\}$.*

DEFINIZIONE 3. – *Dato un oggetto query q, un database di oggetti S, ed un intero k, il problema della* ***k-nearest neighbor search*** *è quello di trovare i k elementi più vicini a q.*

Entrambi i due problemi appena menzionati possono essere risolti calcolando un numero di distanze uguale al numero degli oggetti presenti nella base di dati.

Naturalmente, quello che ci si propone, è risolverli entrambi calcolando il minor numero possibile di distanze. Esistono numerose strutture dati di indicizzazione, come per esempio l'M-Tree, l'FQ-Tree e l'MVP-Tree che sono di valido aiuto per la risoluzione dei due problemi citati; grazie al loro utilizzo, si è in grado di calcolare un numero di distanze molto minore rispetto a quello calcolato da una semplice ricerca sequenziale.

Combinando ed estendendo le idee dall'M-Tree, l'FQ-Tree e l'MVP-Tree, insieme a tecniche randomizzate, è stato creato un semplice ed efficiente schema chiamato Antipole Tree [1]. Con l'uso di questa struttura dati si riescono a risolvere efficientemente i problemi di range search e k-nearest neighbor search in spazi metrici generici e confronti specifici hanno dimostrato le migliori prestazioni dell'Antipole Tree rispetto alle altre note strutture dati presenti in letteratura. Inoltre, quando si è in presenza di spazi con grandi dimensioni, gli algoritmi usati per risolvere i problemi citati diventano inefficienti. Questo è noto in letteratura come il problema della *curse of dimensionality*. È stato dimostrato sperimentalmente che l'Antipole riesce a risolvere efficientemente i problemi di ricerca citati anche in presenza di tali spazi ad alte dimensioni.

Dato un raggio σ, l'Antipole clustering [1] è effettuato da una procedura top-down ricorsiva che inizia da un insieme di punti finito S e controlla ad ogni passo se una data condizione di splitting è soddisfatta. Se non è soddisfatta, allora non si effettua nessuno splitting, e il dato sottoinsieme è un cluster: in tal caso, l'elemento avente distanza approssimativamente minore di σ da ogni altro elemento del cluster è considerato il centroide del cluster. Se invece la condizione è soddisfatta, allora una coppia di punti (A, B), chiamata coppia Antipole, è generata da un algoritmo approssimato randomizzato ed è usata per dividere S in due sottoinsiemi S_A e S_B; S_A contiene tutti gli elementi di S più vicini ad A mentre S_B contiene tutti gli elementi di S più vicini a B. Dalla precedente procedura si ottiene un albero binario chiamato Antipole Tree, dove nei nodi interni ci sono informazioni riguardo le coppie (A, B) via via trovate, mentre nelle foglie ci sono i clusters con i rispettivi centroidi.

L'Antipole Tree [1] è stato applicato con successo in molti campi dell'informatica: in elaborazione delle immagini per la sintesi delle tessiture [2] e per la colorazione di immagini; in reti per il problema del minimum weight matching; in bioinformatica per il calcolo dell'allineamento multiplo di DNA e di sequenze di proteine [3].

2. – GraphGrepVF: un nuovo metodo di indicizzazione per risolvere il problema del graph matching esatto ed inesatto.

Un grafo è composto da un insieme di vertici e un insieme di archi tra le coppie di vertici. Di solito, i vertici rappresentano dati e gli archi rappresentano le relazioni tra di essi. Molte applicazioni nelle industrie, nelle scienze e nell'ingegneria hanno bisogno di rappresentare dati come grafi. Per esempio, motori di ricerca su internet

come google, documenti XML, composti chimici, database genetici, etc., spesso modellano dati come grafi. Avendo quindi tale rappresentazione, un problema che spesso ricorre è quello di trovare tutti i grafi di un insieme che contengono un dato grafo query. Ci riferisce a questo problema con il nome di graph matching.

DEFINIZIONE 4. – *Dato un grafo query G_q e un database di grafi D, il problema del graph matching esatto consiste nel determinare tutte le occorrenze di G_q in ogni grafo in D.*

Questo è un problema NP-completo. Benchè si possano usare algoritmi uno-ad-uno che fanno il matching tra il grafo query ed ogni possibile grafo di input, è più ragionevole utilizzare speciali tecniche per filtrare i grafi del database, riducendo così lo spazio di ricerca e la complessità temporale. Un semplice algoritmo per trovare tutte le occorrenze di un grafo query in un dato grafo di un database, è quello di generare tutte le possibili mappe tra i vertici dei due grafi e di controllare se ogni mappa generata è un match. Naturalmente, la complessità computazionale di tale algoritmo è esponenziale.

Sono stati numerosi i tentativi di ridurre il costo computazionale del graph matching. La ricerca è andata avanti e si è spinta verso tre direzioni principali: la prima è stata quella di studiare algoritmi per grafi con particolari strutture; la seconda è stata quella di introdurre metodi che effettuino una riduzione dello spazio di ricerca (facendo un filtraggio dei grafi di input); la terza è stata quella di usare algoritmi approssimati (quindi si é passati allo studio del graph matching di tipo inesatto).

GraphGrepVF nasce dall'unione di due noti algoritmi presenti in letteratura: GraphGrep [4] e VF [5]. GraphGrep [4] è un metodo generale per trovare tutte le occorrenze di un grafo query in un database di grafi. Esso assume i grafi non direzionati e con una etichetta per ogni nodo; naturalmente si può generalizzare a grafi direzionati con archi etichettati. GraphGrep è basato sull'idea di ridurre lo spazio dei possibili matches. La sua principale componente algoritmica è quella di memorizzare tutti i percorsi di lunghezza minore o uguale ad una certa costante l_p. Questi percorsi sono usati per effettuare un filtraggio dei dati durante la quale si scartano i grafi che non contengono la query. Poi si effettua la fase finale di matching dove il matching viene eseguito solo su un insieme di percorsi candidati.

VF [5] è un algoritmo di graph matching che usa gli *Attributed Relational Graphs (ARG)*, i quali, usando un insieme di regole di ammissibilità, riescono a ridurre il costo computazionale del processo di matching. Il processo di matching è portato avanti usando la *State Space Representation*: uno stato rappresenta una soluzione parziale del matching tra due grafi, e una transizione tra stati corrisponde all'introduzione di una nuova coppia di nodi per cui esiste un match. Le regole di ammissibilità scartano quegli stati che corrispondono a soluzioni parziali che non soddisfano il matching globale.

GraphGrepVF, dunque, usa lo stesso sistema di filtraggio di GraphGrep per i grafi nel database e usa la *State Space Representation* utilizzata da VF per il processo di matching. In GraphGrepVF, viene inoltre introdotta una nuova tabella hash che contiene informazioni riguardante tutti gli archi dei grafi del database. Tale tabella hash, permette di ridurre ulteriormente lo spazio di ricerca, scartando, in base ad opportune considerazioni, sottografi che non conterranno di sicuro occorrenze del grafo query.

Inoltre, GraphGrepVF è in grado di effettuare anche il matching di tipo inesatto tra un database di grafi e un grafo query espresso in notazione lineare usando il linguaggio GLIDE. GLIDE permette l'uso di wildcards tramite i quali si dá la possibilitá all'utente di scegliere quale parte del grafo query deve contenere un'approssimazione. Un . indica un percorso di 1 nodo, un $*$ indica un percorso di 0 o più nodi, un $+$ indica un percorso di 1 o più nodi e, infine, un ? indica un percorso di 0 o 1 nodo. Ad esempio, con un grafo query del tipo $a/./b/+/c$ l'utente otterrebbe tutti i percorsi di tutti i grafi del database che iniziano con un nodo etichettato con a, finiscono con un nodo etichetattato con c e dove tra a e b sia presente un percorso avente esattamente due nodi e tra b e c ci sia un percorso avente almeno un nodo.

GraphGrepVF è risultato più veloce sia di GraphGrep che di VF per un'ampia classe di grafi.

BIBLIOGRAFIA

[1] Cantone D., Ferro A., Pulvirenti A., Reforgiato Recupero D. e Shasha D., *Antipole Tree Indexing to Support Range Search and K-Nearest-Neighbor Search in Metric Spaces*, IEEE Transactions on Knowledge and Data Engineering (TKDE), **17** (2005), 535-550.

[2] Battiato S., Pulvirenti A. e Reforgiato Recupero D., *Antipole Clustering For Fast Texture Synthesis*, Winter School of Computer Graphics (WSCG) (2003).

[3] Di Pietro C., Di Pietro V., Emmanuele G., Ferro A., Maugeri T., Modica E., Pigola G., Pulvirenti A., Purrello M., Ragusa M. e Scalia M., *ANTICLUSTAL: Multiple Sequence Alignment by Antipole Clustering and Linear Approximate 1-Median Computation*, IEEE Computer Society Bioinformatics Conference (CSB) (2003), 326-336.

[4] Giugno R. e Shasha D., *GraphGrep, a fast and universal method for querying graphs*, IEEE International Conference in Pattern Recognition (ICPR) (2002).

[5] Cordella L. P., Foggia P., Sansone C., Tortorella F. e Vento M., *Graph matching: a fast algorithm and its evaluation*, IEEE International Conference in Pattern Recognition (ICPR) (1998).

Via Simeto 5, 95100 San Gregorio di Catania (Catania)
e-mail: diegoref@dmi.unict.it
Dottorato in Scienze Computazionali ed Informatiche (sede amministrativa:
Università degli studi di Napoli Federico II) - Ciclo XVII
Relatore: Professore Alfredo Ferro