Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea in Informatica

BENCHMARK SU DATASET CON VALORI INCERTI CON ORION DBMS:

Tesi di Laurea in Basi di Dati e Sistemi Informativi

Relatore:
Chiar.mo Prof.
DANILO MONTESI

Presentata da:
PASQUALE PUZIO

Correlatore:
Ill.mo Dott.
MATTEO MAGNANI

Sessione I
Anno Accademico 2009/2010

Ai miei genitori,
che mi hanno sempre sostenuto ed incoraggiato
Alla mia fidanzata,
che mi ha sempre pazientemente ascoltato
Al mio caro nonno,
che sarebbe stato molto orgoglioso di me
Al Dott. Magnani,
che si è dimostrato sempre gentile e disponibile
Ad Edoardo,
senza di lui non sarei qui oggi

Indice

Capitolo 1

Introduzione

1.1 Problema

In molti campi scientifici, e non, spesso sorge il problema della gestione dei dati incerti Widom [2005].
Ad esempio, un sensore di rilevazione della posizione o della temperatura, difficilmente, praticamente mai, ha un margine di imprecisione nullo. Inoltre è impossibile essere costantemente aggiornati in tempo reale, in quanto questa esigenza andrebbe a scontrarsi contro i limiti esistenti quali la larghezza di banda della rete, quindi le rilevazioni verosimilmente avvengono ad intervalli di tempo regolari e discreti, e non in modo continuo.

I dati incerti compaiono spesso, anche se non ce ne accorgiamo, anche in applicazioni di uso quotidiano, ad esempio le diagnosi mediche, dove frequentemente non c'è una sicurezza totale sulla diagnosi da attribuire al paziente, in questo caso sarebbe utile poter attribuire percentuali di probabilità alle varie possibili diagnosi.

Un'altra situazione che può generare dati incerti, e la conseguente necessità di gestirli, è l'integrazione di più Dataset in un unico Database per estrarre informazioni o per combinarle tra di loro. Capita spesso che due o più Dataset abbiano dei dati tra di loro inconsistenti, in questo caso sarebbe molto utile poter gestire, e tenerne traccia, queste inconsistenze senza dover ricorrere ad approssimazioni o ad altre soluzioni che potrebbero causare perdita d'informazione. Widom [2010]

Proprio a causa di queste crescenti esigenze, negli ultimi anni la ricerca nel campo dei Database si è focalizzata sullo studio e la realizzazione di un

modello efficiente per la rappresentazione, l'indicizzazione ed infine l'interrogazione di dati incerti.

Sorge quindi il problema di gestire in maniera precisa ed efficiente questo tipo di dati. Per gestione efficiente si intende la possibilità di avere a disposizione meccanismi per rendere efficienti sia la memorizzazione su disco di queste informazioni che la valutazione delle interrogazioni sul Database.

1.2 Dati incerti

La gestione dei dati incerti è ovviamente molto più complessa rispetto a quella dei dati certi, tant'è che attualmente in varie applicazioni commerciali dove è necessario gestire dati incerti, vengono utilizzate varie tecniche di discutibile efficienza o che causano perdita d'informazione:

- si memorizza nel Database semplicemente il valore *più probabile*, perdendo quindi l'informazione sull'incertezza del dato e la possibilità di gestirla
- si aumenta il numero delle tabelle, memorizzando in un'apposita tabella i valori incerti e ricostruendo tutta l'informazione eseguendo dei join, un'operazione molto costosa

Inoltre l'incertezza dei dati di cui si è parlato precedentemente comporterebbe un'inconsistenza dei valori memorizzati nel Database e di conseguenza risultati errati delle interrogazioni.

1.2.1 Esempio 1

In questo primo esempio supporremo di dover gestire le rilevazioni di posizione di un oggetto effettuate da alcuni alcuni sensori che, come accennato in precedenza, eseguono queste rilevazioni ad intervalli di tempo regolare, e con un certo margine di errore.

Supponiamo quindi di avere una tabella strutturata nel seguente modo:

Sensor	**Date**	**Time**	**Position**
S1	01/01/2010	12:00	(50,50)
S2	01/01/2010	12:00	(51,51)
...	...	...	(...,...)

Come si può notare, è possibile che le rilevazioni di alcuni sensori siano tra di loro discordanti, oltre che con un margine di imprecisione non nullo.

Se volessimo sapere quindi in quali momenti l'oggetto rilevato si trovava *probabilmente* in una certa posizione avremmo bisogno di interrogare il Database tenendo conto del margine di errore dei vari sensori, ai quali dovrebbe essere attribuita una distribuzione di probabilità che rappresenti al meglio lo scenario. Nel caso dei sensori spesso si tratta della distribuzione Gaussiana.

Questa distribuzione, che è una delle più note, prevede due parametri, la previsione e la varianza, quest'ultimo direttamente proporzionale al margine di errore del sensore.

1.2.2 Esempio 2

Un ulteriore scenario nel quale la gestione dei dati incerti si rivelerebbe molto utile potrebbe essere il seguente: supponiamo di dover integrare i Database di due o più Social Network, e per alcune persone di avere dati, ad esempio l'età, tra di loro discordanti. Sarebbe utile quindi poter mantenere questa informazione, senza dover ricorrere per ciascun individuo alla memorizzazione di tutte le età possibili.

Una possibile soluzione, utilizzando i dati incerti, potrebbe essere quella di mappare l'età di ciascun individuo con una distribuzione discreta, eventualmente composta da una sola alternativa con probabilità massima nel caso non ci siano dati discordanti, che associ ad ogni possibile età una probabilità, ad esempio in base all'importanza attribuita alla fonte.

Ecco un esempio di come potrebbe essere la struttura della tabella contenente i dati incerti:

Name and Surname	**Data**	**Age**
Pasquale Puzio	...	discrete(21, 0.8, 22, 0.2)
...	...	discrete(...)

In questo esempio abbiamo utilizzato la distribuzione discreta, che si prestava meglio a rappresentare lo scenario descritto.

Immaginando che un utente esegua una ricerca e specifichi come parametro l'età, in questo modo potremmo restituire nel risultato tutti quegli individui con una probabilità maggiore di zero di rientrare nell'intervallo di età selezionato.

1.3 Obiettivi

Si evince quindi che occorrerebbe un supporto nativo da parte dei DBMS che permettesse di gestire direttamente questo tipo di dati utilizzando distribuzioni di probabilità per poi eseguire interrogazioni in maniera specifica.

Un ulteriore problema, dovuto sostanzialmente alla novità dello studio di questo tipo di dati, è la mancanza di riferimenti precisi e validi sia riguardo alle prestazioni di un DBMS con questo specifico supporto, che riguardo all'esistenza di Dataset (reali o sperimentali) e interrogazioni da poter utilizzare per Benchmarking, un processo di misurazione delle prestazioni di un sistema o delle caratteristiche di un prodotto, in questo caso l'oggetto delle misurazioni sono le prestazioni un DBMS.

L'obiettivo di questo progetto di tesi quindi, è innanzitutto studiare il più a fondo possibile lo stato dell'arte nel campo dei DBMS per dati incerti, nello specifico Orion DBMS, per verificarne le capacità ed eventuali limiti. In particolare, uno degli aspetti da approfondire riguarda l'utilizzo e l'eventuale efficacia degli indici per dati incerti.

Secondariamente i risultati ottenuti potrebbero essere un valido punto di partenza qualora ci fosse l'esigenza di paragonare le prestazioni di altri DBMS per valori incerti ad Orion DBMS.

1.4 Risultati ottenuti

Durante lo svolgimento di questo progetto, sono stati creati artificialmente, ma utilizzando dati realistici, dei dataset con dati incerti, ciascuno con delle peculiarità per renderlo interessante ai fini del benchmark, e in modo tale da coprire i maggiori scenari di utilizzo.

Dopo la creazione dei dataset, sono state studiate interrogazioni specifiche, che rispecchiano sostanzialmente le interrogazioni che potrebbero essere eseguite in uno scenario reale, con lo scopo di verificare l'efficienza del

DBMS Orion e l'eventuale overhead aggiunto dalla gestione dei dati incerti.

Infine, a scopo dimostrativo, è stato creato un indice fittizio, ma sostanzialmente identico ad un vero indice, al fine di verificare la prestazioni sia nel caso di presenza degli indici che nel caso contrario. Questa verifica ha evidenziato un notevole miglioramento delle prestazioni in caso di presenza degli indici.

1.5 Struttura della tesi

Nella sezione successiva, ovvero la sezione 2, vengono illustrate sia tramite esempio che formalmente, le principali query possibili su una base di dati con valori incerti. Nella sezione 3 viene introdotto il DBMS Orion, approfondendo gli aspetti legati sia al modello dei dati che al supporto programmatico per la gestione e la manipolazione dei dati incerti. Nelle sezioni 4 e 5 invece, si mostrano i dataset generati per l'analisi sperimentale, e i risultati ottenuti da quest'ultima. Nelle sezioni 6 e 7 vengono introdotti Orion 2.0 e gli indici per dati incerti che saranno implementati in esso, in particolare nella sezione 7 viene mostrata anche la creazione di un indice fittizio creato con lo scopo di verificare la differenza di performance con e senza la presenza di un indice. Nella sezione 8 infine, ci sono le conclusioni e i possibili sviluppi futuri.

Capitolo 2

Interrogazioni su basi di dati incerte

Uno dei motivi principali che dovrebbe spingere coloro che hanno l'esigenza di gestire una notevole quantità di dati incerti a gestire direttamente questa incertezza, interpretando i dati come distribuzioni discrete o continue, è la possibilità di eseguire interrogazioni specifiche, che permettano di estrarre dal Database informazioni difficilmente ottenibili utilizzando esclusivamente dati certi.

Ad esempio, supponiamo di dover eseguire misurazioni con sensori per il rilevamento della posizione. Come detto in precedenza, un sensore esegue rilevazioni ad intervalli di tempo regolari, e con un margine di imprecisione nella maggior parte dei casi non nullo.

La posizione rilevata dal sensore quindi può essere considerata a tutti gli effetti un dato incerto, e in quanto tale è possibile attribuirle una distribuzione, continua o discreta, che sia in grado di rappresentarla nella maniera più precisa e appropriata.

Si assuma di volere rappresentare la posizione di oggetti in movimento, ad esempio pesci, e di volerne rilevare la posizione tenendo conto anche delle caratteristiche delle varie specie di pesci esistenti.
Nel nostro caso, come illustrato in figura 2.1, è plausibile pensare che il movimento dei pesci sia costante, quindi una distribuzione uniforme che prevede lo spostamento di ogni singolo oggetto entro un raggio predefinito è idonea a rappresentare il nostro scenario.

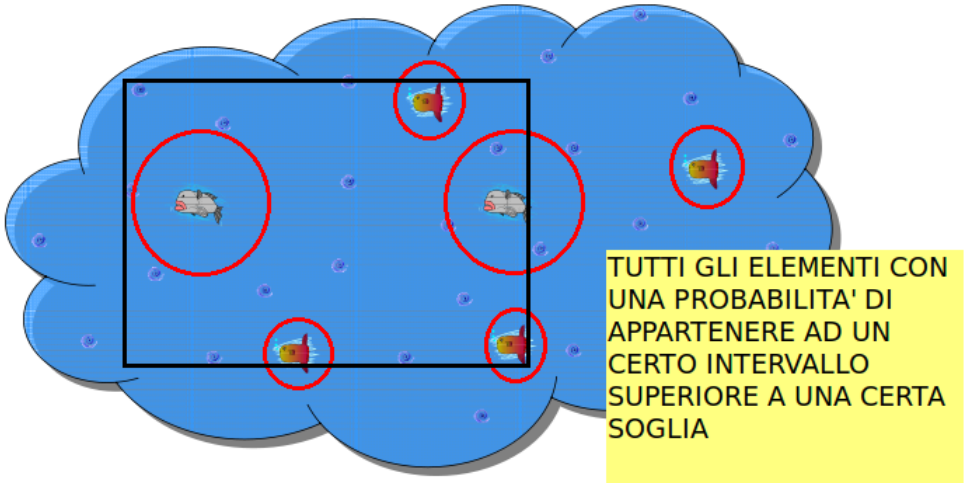

Figura 2.1: Scenario 1
Il rettangolo nero indica l'intervallo sul quale eseguiamo la query, mentre i cerchi rossi indicano l'area di incertezza della posizione del pesce

Immaginiamo poi di voler rappresentare la seguente realtà.
Esistono due specie differenti di pesci:

- la prima, di colore grigio, che ha una capacità di spostamento più rapida
- la seconda, di colore rosso, che ha una capacità di spostamento più ridotta.

Questa informazione può essere rappresentata con una scelta adeguata dei parametri della distribuzione che abbiamo deciso di utilizzare, nel nostro caso la distribuzione uniforme.

Adesso per quanto riguarda la nostra piccola base di dati contenente dati incerti, potremmo avere l'esigenza di ottenere tutti gli elementi appartenenti ad un certo intervallo (in figura il rettangolo nero). Considerando la natura incerta dei nostri dati, l'interrogazione dovrebbe essere riformulata nel seguente modo: *Ottenere tutti gli elementi che hanno una probabilità di appartenere ad un certo intervallo superiore ad una probabilità molto alta, ad esempio il 90%.*

Come si può facilmente notare dall'immagine 2.1, un elemento ha una probabilità del 100% di far parte del risultato finale, in quanto è contenuto totalmente nell'intervallo di selezione, un altro elemento invece non ha alcuna

probabilità di appartenere all'intervallo in quanto non ha alcuna intersezione con esso.

Negli altri casi invece, non è possibile determinare con sicurezza quali elementi appartengano all'intervallo e quali no, per questo motivo è necessario utilizzare gli eventuali strumenti messi a disposizione dal DBMS per calcolare la probabilità e determinare l'appartenenza o meno di un elemento al risultato finale.

Questa necessità rende quindi necessario un supporto diretto da parte del DBMS, che permetta di esplicitare l'incertezza di alcuni dati e poter eseguire operazioni specifiche (selezione, manipolazione, operatori aggregati, ecc.) su di essi.

Altri esempi verranno forniti nella Sezione 4.1 nella quale verranno presentati i Dataset generati per il Benchmark.

2.1 Probabilistic Threshold Queries

Nel caso visto in precedenza, abbiamo eseguito una semplice query di selezione, con due parametri in input:

- un intervallo di selezione
- una soglia di probabilità, **Probability Threshold**

Questo tipo di query è di certo una delle più utilizzate, se non la più utilizzata, nelle realtà di Database con valori incerti, prende il nome di **Probability Threshold Query** e fa parte delle modalità di interrogazione sulle quali si sono concentrati maggiormente gli sforzi della ricerca in questi ultimi anni.

Esaminiamo formalmente la semantica attribuita ad una **Probabilistic Threshold Query**.

Nelle seguenti definizioni assumeremo **T** come una tabella appartenente al Database, ciascuna tupla (o oggetto) appartenente alla tabella sarà identificata da T_i (i $\in$ [1,...., $\|T\|$]), *dove* T_i.a è un attributo di tipo incerto.

Definizione 1 *Un* ***Intervallo di incertezza*** *di $T_i.a$, indicato con $U_i.a$, è un intervallo [L_i,R_i] dove L_i, $R_i \in \Re$ e valgono sempre le condizioni $R_i \geq L_i$ e $T_i.a \in U_i.a$.*

Definizione 2 *Una **funzione di densità** (uncertainty pdf) di $T_i.a$, indicata con $f_i(x)$, è una funzione di densità tale che $f_i(x) = 0$ se $x \notin U_i$.*

Definizione 3 *Dato un intervallo chiuso [a, b] , dove a,b $\in \Re$ e $a \leq b$, una Probability Threshold Query restituisce un insieme di tuple T_i, tale che la probabilità di $T_i.a$ di appartenere all'intervallo [a,b], ovvero p_i, è maggiore o uguale di p, dove $0 < p \leq 1$.*

Una *Probabilistic Threshold Query* per semplicità può essere considerata una semplice range query con le peculiarità di operare con dati rappresentati da distribuzioni di probabilità e restituire nel risultato solo quelle distribuzioni, nel nostro caso tuple, che superano la soglia di probabilità.

2.1.1 Esempio

Nella sezione precedente è stato mostrato un esempio generico di Probabilistic Threshold Query.
In questo esempio invece verrà approfondito più nel dettaglio, il significato di *calcolo della probabilità* durante una PTQ.

Supponiamo di avere un Dataset, rappresentato in figura 2.2, con alcune distribuzioni Gaussiane e, come solito, un intervallo di interrogazione, e una soglia di probabilità, detta *probability threshold*.

Bisognerà quindi calcolare un integrale che abbia come estremi gli estremi dell'intervallo di interrogazione, e la tupla analizzata, in questo specifico caso, sarà inclusa nel risultato finale se e solo se il risultato dell'integrale sarà maggiore della soglia. In poche parole, si tratta di misurare l'area sotto la curva, ad esempio nell'intervallo [-1,1], e verificare che sia maggiore della soglia.

Questa particolare coincidenza ovviamente è dovuta alla particolarità della distribuzione Gaussiana.

2.2 Join Queries

Nel caso delle query in cui è previsto un join tra due o più tabelle, o anche nel caso di self-join, è neccessario distinguere due casi:

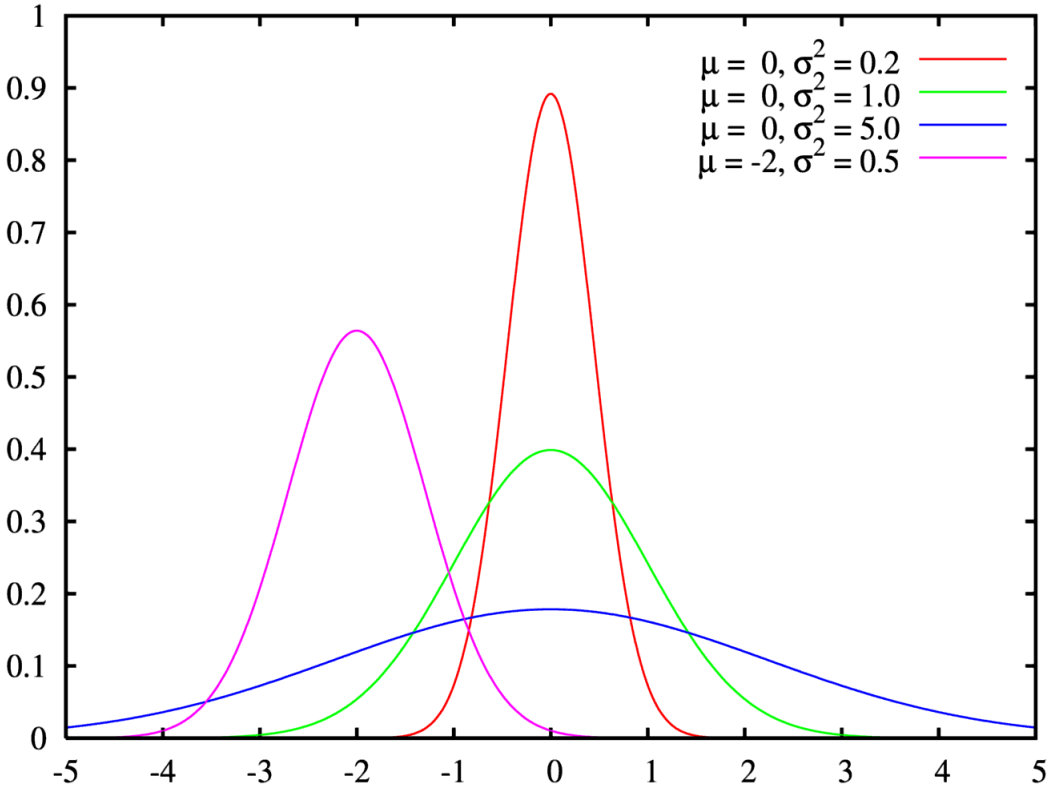

Figura 2.2: Distribuzioni Gaussiane memorizzate nel Database

- nel primo caso il join viene eseguito tra due attributi, entrambi incerti; questo comporta la necessità di dover definire la semantica di operazioni di confronto tra attributi incerti, come *uguaglianza*, *maggiore di*, ecc.
- nel secondo caso invece il join avviene tra un attributo incerto e un altro certo

Definizione 4 *Dato un attributo incerto a, e la sua funzione di densità f(x), a.F(x), detta Cumulative density function, indica il valore della funzione di densità nel punto x.*
Inoltre a.l e a.u indicano rispettivamente l'estremo inferiore e l'estremo superiore dell'intervallo di incertezza.

Da questa definizione possiamo ottenere la definizione di uguaglianza, e maggiore di:

Definizione 5 *La probabilità che un attributo a sia maggiore di un attributo b, entrambi incerti, è data da:*
$P(a > b) = \int_{max(a.l,b.l)}^{b.u} a.f(x)b.F(x)dx + 1 - a.F(b.u)$ *se vale la condizione* $a.l \leq b.u < a.u$
$= \int_{max(a.l,b.l)}^{a.u} a.f(x)b.F(x)dx - a.F(b.u)$ *se vale la condizione* $b.l \leq a.u \leq b.u$

Definizione 6 *Dato un parametro di risoluzione c, la probabilità che a sia uguale a b (con una distanza c) è data da:*
$P(a =_c b) = \int_{-\infty}^{\infty} a.f(x)(b.F(x+c) - b.F(x-c))dx$

Il parametro c in questo caso può essere interpretato come il margine di approssimazione utilizzato durante il confronto.

L'ultima definizione introdotta può essere utilizzata anche per il confronto tra un attributo incerto e un attributo certo, ovvero il confronto tra una distribuzione di probabilità e un singolo valore appartente all'insieme dei numeri reali.

Definizione 7 *Dato un parametro di risoluzione c, la probabilità che un attributo incerto a sia uguale ad un numero reale r con un margine di errore c, è data da:*
$P(a =_c r) = a.F(v+c) - a.F(v-c)$ *e* $P(a > r) = 1 - a.F(v)$

Definiamo un'estensione di *Probabilist Threshold Query* per le join query.

Prese due relazioni contenenti due attributi incerti, a e b, eventualmente anche con lo stesso nome:

Definizione 8 *Dato un operatore tra dati incerti θ_u (ad esempio, $=_c$, $<$, $\neq_c$) una* ***Probabilistic Threshold Join Query (PTJQ)*** *restituisce tuttle le tuple (R_i,S_j), tali che P($R_i.a$ θ_u $S_j.b$) > p, dove p ∈ [0,1] è la soglia di probabilità (probability threshold).*

Durante l'esecuzione delle *join queries* su valori incerti, cosi come quelle su valori certi, è molto importante, al fine di ridurre al minimo la lettura sul disco di tuple che non rientrano nel risultato, eliminare a priori dal risultato le tuple che con certezza non rispetta le condizioni di join. Questa operazione è detta *pruning*.

Nel caso in cui tra le condizioni ci sia almeno un operatore di uguaglianza, senza avere a disposizione particolari indici, una prima soluzione, molto semplice, potrebbe essere quella di eliminare a priori dal risultato quelle tuple i cui valori incerti non presentano sovrapposizione, ovvero tra i loro intervalli di incertezza non c'è alcuna intersezione Prabhakar [2008b].

Di questo problema comunque se ne parlerà approfonditamente nella sezione Sezione 6 sugli indici.

2.3 Nearest Neighbor Queries

Questo tipo di query prevede che nel risultato sia restituita la tupla che ha la distanza inferiore rispetto ad un particolare punto. Una variante di questo tipo di query è *K-nearest Neighbor Query* che prevede nel risultato le prime k tuple per distanza rispetto ad un determinato punto [Prabhakar [2008b]].

Meglio però fornire una definizione più precisa e formale:

Definizione 9 *Dato un punto di interrogazione q, detto query point, e un attributo incerto, cioè una distribuzione di probabilità, a_i, a_j è il risultato della* ***Nearest Neighbor Query*** *per q, se esiste x tale che dist(q, a_j) < x e per ogni tupla appartenente alla relazione con attributi a_j, dist(q, a_j) > x. Per gli attributi incerti, questo si tratta di un evento probabilistico.*
La probabilità P^i_{nn} che a_i sia il nearest neighbor di q è uguale a $\int_x Pr[(x \leq dist(q, a_i) \leq x + dx) \cap (\forall j \neq i, dist(q, a_j) \geq x + dx)]$

La definizione appena fornita è valida anche in presenza di eventuali dipendenze tra gli attributi incerti.

La seguente definizione invece descrive la struttura del risultato di una *Probabilistic Nearest Neighbor Query*

Definizione 10 *dato un query point q e un insieme di oggetti (tuple)* O_1, O_2, ... , O_n *con attributi incerti, una Probabilistic Nearest Threshold Query restituisce un insieme di coppie* (O_i, P^i_{nn}), *che associa a ciascun oggetto la probabilità di essere il nearest neighbor*

A giudicare da quanto detto finora, si potrebbe pensare che la gestione di dati incerti e la valutazione di una *Probabilistic Threshold Query* non siano molto complesse, poichè l'unica operazione aggiuntiva rispetto ai dati certi sarebbe il calcolo di un integrale per ottenere il valore della probabilità.
In realtà, oltre alla mera valutazione dell'interrogazione, nel caso delle interrogazioni sui dati incerti è necessario fornire un supporto maggiore da parte del DBMS, ad esempio creando degli indici appositi per i dati incerti.

Questo aspetto verrà discusso nella Sezione 6.

Capitolo 3

DBMS Orion

3.1 Descrizione generale

Orion DBMS è un sistema per la gestione dei dati incerti, sviluppato dal gruppo di lavoro di Sunil Prabhakar presso la *Purdue University* di West Lafayette, negli Stati Uniti. Questo sistema fornisce un valido supporto, per l'interrogazione (querying) e la memorizzazione (data storage) dei dati incerti, a cui manca, per il momento, un equivalente supporto per quanto riguarda l'indicizzazione.

Orion DBMS è un'estensione del noto DBMS PostgreSQL: si tratta di un insieme di funzioni, operatori e librerie condivise che permettono l'introduzione e la gestione di un nuovo tipo di dato: il dato incerto.

Le principale caratteristiche di Orion sono le seguenti:

- estensione degli operatori classici già presenti in PostgreSQL per supportare le query sui dati incerti
- possibilità di configurazione della *qualità* delle interrogazioni, ovvero la precisione con la quale avviene la discretizzazione delle distribuzioni
- nuovi operatori aggregati per eseguire query specifiche per i dati incerti

Un'altra caratteristica molto importante di Orion DBMS, è la non interferenza dei dati incerti con i dati certi, ovvero l'aggiunta di queste funzionalità per la manipolazione dei dati incerti non comporta alcun overhead nella normale gestione dei dati certi in quanto si tratta semplicemente di definizioni di

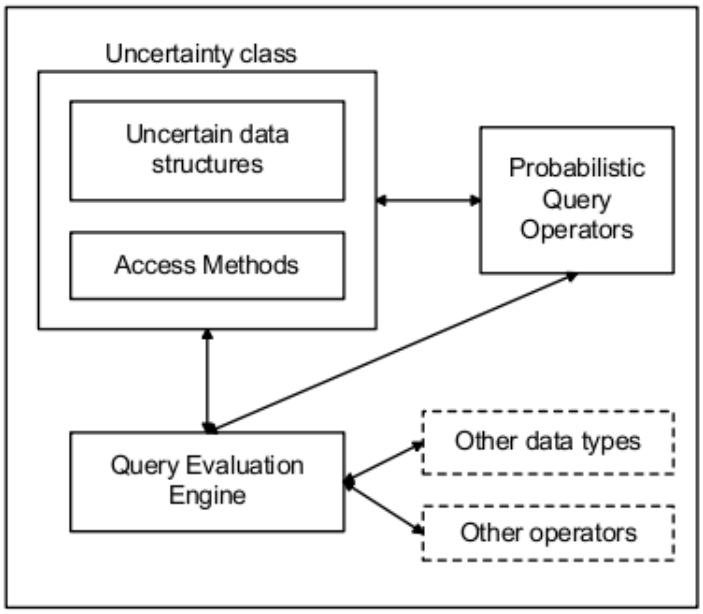

Figura 3.1: Architettura di Orion DBMS

nuove funzioni, operatori e librerie utilizzate esclusivamente per i dati incerti.

Di seguito verrà illustrato più nel dettaglio il modello dei dati di Orion ed il suo supporto per le distribuzioni sia discrete che continue.

3.2 Modello dei dati

Il modello dei dati utilizzato da Orion è basato innanzitutto sull'incertezza degli attributi (Prabhakar [2003]) e non delle tuple (*attribute-level uncertainty*): questo significa che l'incertezza non riguarda l'esistenza di una determinata tupla ma esclusivamente il valore assunto da un attributo. Per essere più precisi, un modello basato sull'incertezza *tuple-level*, attribuisce una probabilità di esistenza a ciascuna possibile tupla, invece un modello basato sull'incertezza *attribute-level* associa a ciascun attributo un range di possibili valori e una distribuzione di probabilità (*pdf*).

Il modello dei dati quindi è molto semplice e può essere sintetizzato formalmente con le definizioni già espresse nella Sezione 2.1.

Inoltre per ciascuna distribuzione di probabilità memorizzata nel Database vale la seguente proprietà:
$\int_{L_i}^{R_j} f_x dx = 1$

3.2.1 Classificazione delle query

Le query eseguibili in Orion DBMS. possono essere suddivise secondo due criteri. Per quanto riguarda il tipo di risultato, sono previste due categorie di interrogazioni, le **Entity-based Query** che restituiscono degli oggetti, e le **Value-based Query** che invece restituiscono semplicemente un singolo valore numerico. Un altro criterio di suddivisione è quello dell'utilizzo o meno di operatori aggregati, che determinano il modo in cui viene calcolato il risultato finale. Nel primo caso il risultato ottenuto dall'interrogazione dipende da tutti gli oggetti che hanno contribuito a formare il risultato, nel secondo caso invece ciascun oggetto presente nel risultato di un'interrogazione è indipendente dagli altri oggetti.

Basandoci su questi due criteri di classificazione, possiamo quindi suddividere le query in quattro grandi categorie:

- **Value-based Non-Aggregate**: un esempio di questa categoria è una query che restituisce nel risultato la previsione di ciascun attributo incerto
- **Entity-based Non-Aggregate**: un esempio di questa categoria potrebbe essere una query che dato un intervallo ed una soglia di probabilità, restituisce un insieme di coppie T_i,p_i, dove T_i è l'i-esima tupla della tabella T, e p_i e la probabilità che T_i appartenga all'intervallo.
- **Value-based Aggregate**: questo tipo di query restituiscono un singolo valore ottenuto dall'applicazione di operatori aggregati (e.g. *SUM, AVG, MIN, MAX*)
- **Entity-based Aggregate**: un esempio di questa categoria è ad esempio una *(K)Nearest Neighbor Query* che restituisce un insieme di coppie T_i,p_i, dove T_i è l'i-esima tupla della tabella T, e p_i e la probabilità che T_i sia il nearest neighbor del punto dato in input.

3.3 Architettura di Orion

L'architettura ad alto livello di Orion è rappresentata in Figura 3.1. Come si può notare non sono ancora inclusi in alcun modo gli indici studiati per i dati incerti, che probabilmente saranno presenti nella prossima versione di Orion.

Orion prevede tre tipi principali per dati incerti:

- **Gaussian** per rappresentare le distribuzione Gaussiane
- **Discrete** per rappresentare le distribuzioni discrete
- **Histogram** per rappresentare altri tipi di distribuzioni

Dei tre tipi presenti, assume un particolare significato il terzo, che è stato introdotto allo scopo di fornire un supporto generico per rappresentare più tipi di distribuzioni oltre a quelli previsti [Reynold Cheng and Prabhakar [2005]]. Questi tre tipi sono raggruppati in unica classe, *UNCERTAIN*, che è un tipo di dato a lunghezza variabile per memorizzare più tipi di distribuzioni (Discrete, gaussiane, Poisson, etc.), in pratica, facendo un piccolo paragone con i linguaggi a oggetti basati su classi, possiamo paragonare *UNCERTAIN* ad una superclasse, quindi ad un tipo più generico, e Gaussian, Discrete e Histogram a sottoclassi di Uncertain più specializzate.

Di seguito viene mostrata la definizione del tipo *UNCERTAIN*:

```
typedef struct {
  int32 size; /* TOTAL size of this data structure */
  int type;
  /* Type of uncertain data
  Histogram='h', Gaussian='g', Discrete='g' */
  char data[1]; /* Pointer to actual data */
} Uncertain
```

Di seguito vengono mostrate le type definitions delle strutture dati per distribuzioni incerte, ovvero *Discrete*, *Gaussian* e *Histogram*, quest'ultima, come detto in precedenza, è stata creata con lo scopo di poter fornire un supporto generico per più distribuzioni, sia discrete che continue, rispetto a quelle previste:

```
typedef struct {
   int32 num data;
```

```
   float4 data[1][2]; /* Data points and their probabilities */
} Discrete;

typedef struct {
   float mu; /* Mean */
   float sigma; /* Standard Deviation */
} Gaussian;

typedef struct {
   float min value; /* Value at which the sampling begins */
   float max value; /* Value at which the sampling ends */
   float interval; /* size of sampling interval */
   float data[1];
   /* The actual histogram data (cdf); exactly
   (max value-min value)/interval + 1 values in this array.
   Note that this is a variable length array */
} Histogram;
```

3.4 Operatori e funzioni

Oron DBMS fornisce un set abbastanza ricco di operatori e di funzioni specifici per i dati incerti. Alcuni di questi permettono operazioni molto semplici, come ad esempio operatori per il confronto sia tra dati incerti che tra dati incerti e certi, funzioni per l'estrazione della previsione e della varianza, ecc., altri invece forniscono strumenti più complessi per eseguire operatori aggregati per dati incerti, range query, ecc.

Vediamo ora una panoramica introduttiva degli operatori e delle funzioni presenti in Orion DBMS.

3.4.1 Operatori e funzioni per il confronto

In Orion esistono molti operatori binari per operazioni di confronto tra dati incerti, o tra dati incerti e certi, che risultano molto utili soprattutto in caso di query in cui sono previsti join tra attributi incerti (vedi Sezione 5).

- **Uguaglianza (=)** *u_eq(uncertain, uncertain)*, *u_eq(uncertain, real)*, *u_eq(real, uncertain)*, *u_eq_const_bool(uncertain, uncertain)*, *u_eq_const_bool(uncertain, real)*, *u_eq_const_bool(real, uncertain)*: queste

funzioni permettono il confronto di uguaglianza tra attributi incerti o tra attributi incerti e valori reali, la prima restituisce la probabilità di uguaglianza mentre la seconda restituisce un valore booleano utilizzando come threshold il valore di default memorizzato nel sistema

- **Disuguaglianza (! =)** *u_neq(uncertain, uncertain)*, *u_neq(uncertain, real)*, *u_neq(real, uncertain)*, *u_neq_const_bool(uncertain, uncertain)*, *u_neq_const_bool(uncertain, real)*, *u_eq_nconst_bool(real, uncertain)*

- **Maggiore di (>)** *u_greater(real, uncertain)*, *u_greater(uncertain, real)*: queste funzioni restituiscono la probabilità che il primo valore sia maggiore del secondo

- **Minore di (<)** *u_less(real, uncertain)*, *u_less(uncertain, real)*

E' possibile utilizzare alternativamente a queste funzioni anche gli operatori infissi: =%, >%, <%.

3.4.2 Operatori aritmetici

- **Addizione (+)** u_add(uncertain, uncertain): somma due valori incerti

Alternativamente alla funzione u_add è possibile utilizzare l'operatore infisso +.

3.4.3 Operatori di estrazione

In Orion, come detto in precedenza, sono presenti anche alcuni operatori che permettono di estrarre dalla distribuzione informazioni molto utili come la previsione, la varianza, ecc.

- **Previsione** *u_expected*: questa funzione permette di estrarre la previsione di una variabile incerta

- **Varianza** *u_variance*: questa funzione permette di estrarre la varianza di una variabile incerta

- **Estremo inferiore** *u_lower*: questa funzione permette di estrarre l'estremo inferiore di una variabile incerta

- **Estremo superiore** *u_upper*: questa funzione permette di estrarre l'estremo superiore di una variabile incerta

- **Intervallo** *u_interval*: questa funzione permette di estrarre l'intervallo di una variabile incerta
- **Appartenenza ad un intervallo** *u_prob(uncertain, lower, upper)*: questa funzione restituisce la probabilità che una variabile incerta appartenga all'intervallo definito dagli estremi lower e upper

Altri operatori sono illustrati nel manuale di Orion.

3.4.4 Operatori aggregati

Orion fornisce anche una piccola quantità di operatori aggregati per query *Entity-based Aggregate*:

- **Range query** *u_range(lower, upper, uncertain_set, threshold)*: questa funzione esegue una probabilistic threshold query utilizzando come intervallo [lower, upper] e la threshold indicata
- **Entity Nearest Neighbor Query** *u_ENNQ(uncertain set, x, divisions)*: questa funzione restituisce una serie di oggetti associati alla loro probabilità di essere il *Nearest Neighbor* di x, l'ultimo parametro è opzionale e indica il grado di approssimazione durante il calcolo

3.4.5 Codice

Di seguito viene mostrato il codice sorgente di alcune tra le funzioni principali di Orion DBMS. Come si può facilmente notare, l'implementazione, stando ad un primo sguardo, non sembra particolarmente complessa:

- **Previsione (u_expected(uncertain))**:

```
/* Function to calculate expected value */
PG_FUNCTION_INFO_V1(u_expected);

Datum
u_expected(PG_FUNCTION_ARGS) {
  Uncertain *input = (Uncertain *) PG_GETARG_POINTER(0);

  Histogram *h;
  Gaussian *g;
  Discrete *d;
```

```
  int i;
  float4 temp;

  switch(input->type) {
  case 'h':
    h = (Histogram *) input->data;

    temp=0;

    for(i=0; i< ((h->max_value - h->min_value)/h->interval); i++)
      temp += (calc_pdf(h,i)/2) *
      ( sqr(h->min_value + h->interval*(i+1)) -
       sqr(h->min_value + h->interval*i));


    PG_RETURN_FLOAT4(temp);
    break;

  case 'g':
    g = (Gaussian *) input->data;

    PG_RETURN_FLOAT4(g->mu);
    break;

  case 'd':
    d = (Discrete *) input->data;

    temp = 0;

    for(i=0; i<d->num_data; i++)
      temp += d_point(d->data[i]) * d_prob(d->data[i]);

    PG_RETURN_FLOAT4(temp);
    break;

  default:
    ereport(ERROR,
(errcode(ERRCODE_INTERNAL_ERROR),
errmsg("Fatal error:
Invalid data representation for uncertain type")));
```

```
  }

  /* control should never reach this point */
  PG_RETURN_FLOAT4(-1.0);

}
```

- **Confronto tra uncertain e real (u_eq(uncertain, real))**:

```
/*************** Internal function to check uncertain =_c v */

float4 equal_const(Uncertain *input, float4 v, float4 c, int N) {
  int i;

  Histogram *h;
  Gaussian *g;
  Discrete *d;

  switch(input->type) {
  case 'h':
    h = (Histogram *) input->data;

    return(calc_cdf(h,v+c) - calc_cdf(h,v-c));
    break;

  case 'g':
    g = (Gaussian *) input->data;

    return(calc_cdf_g(g,v+c,N) - calc_cdf_g(g,v-c,N));
    break;

  case 'd':
    d = (Discrete *) input->data;

    for(i=0; i<d->num_data; i++)
      if (v == d_point(d->data[i]))
return(d_prob(d->data[i]));

    return(0);
```

```
    break;

  default:
    ereport(ERROR,
    (errcode(ERRCODE_INTERNAL_ERROR),
     errmsg("Fatal error:
     Invalid data representation for uncertain type")));

  }

  /* control should never reach this point */
  return(-1.0);
}
```

- **Confronto tra uncertain e uncertain (u_eq(uncertain, uncertain))**:

```
/* Function to calculate probability of equality
of uncertain with another uncertain:
real u_eq_u(uncertain a, uncertain b, real c)*/
PG_FUNCTION_INFO_V1(u_eq_u);

Datum
u_eq_u(PG_FUNCTION_ARGS) {
  Uncertain *input_a = (Uncertain *) PG_GETARG_POINTER(0);
  Uncertain *input_b = (Uncertain *) PG_GETARG_POINTER(1);
  float4 c = PG_GETARG_FLOAT4(2);
  int N = PG_GETARG_INT32(3);

  Histogram *h_a, *h_b;
  float bound_l, bound_u, next_bound, prob=0, curr_bound;

  Discrete *d = NULL;

  int i;

  if (input_a->type == 'd')
    d = (Discrete *) input_a->data;
  else if (input_b->type == 'd') {
    d = (Discrete *) input_b->data;
    input_b = input_a; /* Now this is similar to case above */
```

```
}

if (d != NULL) { /* Atleast one of the input is discrete */
  for(i=0; i < d->num_data; i++)
    prob += d_prob(d->data[i]) *
    equal_const(input_b, d_point(d->data[i]), c, N);

  PG_RETURN_FLOAT4(prob);
}


/* None of the input is discrete, convert everything to histogram */

switch(input_a->type) {
case 'h':
  h_a = (Histogram *) input_a->data;
  break;
case 'g':
  h_a = gauss_to_histo((Gaussian *) input_a->data, N);
  break;
default:
  ereport(ERROR,
  (errcode(ERRCODE_INTERNAL_ERROR),
   errmsg("Fatal error:
   Invalid data representation for uncertain type")));
}

switch(input_b->type) {
case 'h':
  h_b = (Histogram *) input_b->data;
  break;
case 'g':
  h_b = gauss_to_histo((Gaussian *) input_b->data, N);
  break;
default:
  ereport(ERROR,
  (errcode(ERRCODE_INTERNAL_ERROR),
   errmsg("Fatal error:
   Invalid data representation for uncertain type")));
}
```

```
  // Max of these two
  bound_l = (h_a->min_value > (h_b->min_value - c)) ?
    h_a->min_value: (h_b->min_value - c);

  // Min of these two
  bound_u = (h_a->max_value < (h_b->max_value + c)) ?
    h_a->max_value: (h_b->max_value + c);

  if (bound_u <= bound_l) // No overlap
    PG_RETURN_FLOAT4(0.0);

  //We have an overlap, begin main algo.

  //printf("Overlap found\n");

  next_bound = bound_l;

  //printf("BOUNDS: %f,%f\n",bound_l,bound_u);

  do { // Atleast one loop required, because there IS an overlap
    curr_bound = next_bound;

     // The boundary for next interval beginning
    next_bound = next_interval_step(h_a, curr_bound);

    if (next_bound >= bound_u)
      next_bound = bound_u; // Work done, Exit before next loop

    //printf("%f, %f\n",curr_bound, next_bound);

    prob +=
    calc_pdf(h_a, (curr_bound - h_a->min_value)/h_a->interval) *
      (integrate_cdf(h_b, curr_bound + c, next_bound + c) -
      integrate_cdf(h_b, curr_bound - c, next_bound - c));

  } while (next_bound != bound_u);

  PG_RETURN_FLOAT4(prob);

}
```

È interessante anche mostrare il codice relativo alla funzione sperimentale **get_x_bound**, che pur essendo definita e implementata, non viene mai richiamata all'interno delle altre funzioni poichè, al contrario di quanto si possa pensare, è un'implementazione realizzata esclusivamente a fini sperimentali in quanto non sfrutta minimamente informazioni contenute in alcun indice:

```
float get_x_bound(Uncertain *input, bool left, float x_bound, int N) {
  Histogram *h;
  int i;
  float temp;

  if (x_bound > 0.5) {
    printf("WARNING: get_x_bound called with x_bound > 0.5");
    // Not returning, maybe we should ?
  }

  if (!left)
    x_bound = 1 - x_bound;

  switch(input->type) {
  case 'h':
    h = (Histogram *) input->data;
    break;

  case 'g':
    h = gauss_to_histo((Gaussian *) input->data, N);
    break;

  default:
    ereport(ERROR,
            (errcode(ERRCODE_INTERNAL_ERROR),
             errmsg("Fatal error:
             Invalid data representation for uncertain type")));
  }

  if (x_bound <= 0)
    return (h->min_value);
  else if (x_bound >= 1)
    return (h->max_value);
```

```
  for (i=1; i < number_intervals(h); i++)
    if (h->data[i] > x_bound) break;

  i--; /* We got the interval in which the x_bound falls,
  now find the point which corresponds to this x_bound */


  // This looks cryptic, but draw a simple figure
  to find out what is going on here

  if (h->data[i+1] == h->data[i]) // highly unlikely
    return (left?(h->min_value + h->interval*(i+1)) :
    (h->min_value + h->interval*i)); // Get a tight bound
  else
    return h->min_value + (h->interval * i) +
    (x_bound - h->data[i]) * h->interval / (h->data[i+1] - h->data[i]);
}


/* We don't really need this function, just for testing */
```

3.5 Esempio

Ora che sono state illustrare le principali funzioni presenti in Orion DBMS, è possibile mostrare in pratica come creare una tabella con uno o più attributi incerti ed eseguire interrogazioni.

Prendiamo come esempio il Dataset illustrato in Figura 7.2 nella sezione 7, quindi creiamo la tabella nel seguente modo:

```
CREATE TABLE esempio (
id int primary key,
pdf uncertain);
```

Dopo di che popoliamo la tabella con le distribuzioni rappresentate in Figura 7.2 nella sezione 7, il primo parametro indica la previsione e il secondo la varianza:

```
INSERT INTO esempio VALUES(1,'(g,0,0.2)');
INSERT INTO esempio VALUES(2,'(g,0,1.0)');
INSERT INTO esempio VALUES(3,'(g,0,5.0)');
INSERT INTO esempio VALUES(4,'(g,-2,0.5)');
```

Ora è possibile, ad esempio, eseguire una *Probabilistic Threshold Query* ed ottenere il risultato dell'interrogazione:

```
SELECT * FROM esempio where u_prob(pdf,-1,1) > 0.5
```

Questa interrogazione restituirà tutte le tuple che hanno una probabilità di appartenere all'intervallo [-1,1] superiore a 0.5:

```
 id |       pdf
----+-----------------
  1 | (g, 0.00, 0.20)
  2 | (g, 0.00, 1.00)
(2 rows)
```

Da come si può facilmente notare, senza particolari nozioni matematiche, e in modo analogo a come verrebbe definita, popolata e interrogata una tabella senza alcun dato incerto, è stato possibile memorizzare nel nostro Database distribuzioni di probabilità e manipolarle per poter eseguire un'interrogazione.

Capitolo 4

Generazione di Dataset incerti per Benchmarking

In questa sezione inizia la descrizione del lavoro preliminare che ho svolto per il Benchmarking su Orion DBMS.

Il lavoro è consistito nella creazione di tre Dataset realistici, con l'aggiunta di dati incerti generati casualmente. È stata scelta questa strada a causa dell'indisponibilità di dataset incerti già creati o di dataset reali fruibili per questo scopo.

Il lavoro di generazione quindi è stato svolto in parte utilizzando una versione leggermente modificata di uno script reperito online (GenerateData.com). Per quanto riguarda la generazione dei dati incerti invece ho creato dei generatori personalizzati, scritti utilizzando il linguaggio Java, che in base ad alcuni parametri generano degli statement SQL da eseguire per popolare i Dataset.

Nella sezione successiva verranno presentati i tre dataset e per ciascuno di essi verranno spiegate le sue peculiarità e le modalità e i parametri utilizzati per la generazione automatica.

4.1 Descrizione dei Dataset

Di seguito verranno illustrati in dettaglio i tre Dataset utilizzati per il Benchmarking:

- **Dataset Plane** (*Distribuzione uniforme*): in questo dataset è stato rappresentato uno scenario nel quale ci sono una certa quantità di aerei

dispersi ai quali è associata un'area entro la quale si prevede che siano precipitati, ed una serie di rottami ritrovati in determinati punti del globo che possono essere attribuiti ad uno o più areoplani

- **Dataset Meteo** (*Distribuzione gaussiana*): in questo dataset è stato rappresentato uno scenario nel quale esistono tre stazioni meteo dislocate in ogni città, e per ciascun giorno e ciascuna stazione vengono riportate le temperature rilevate

- **Dataset Clinico** (*Distribuzione discreta*): questo dataset rappresenta uno scenario molto più reale e quotidiano in quanto contiene varie diagnosi effettuate sui pazienti, alcune di queste senza un'assoluta certezza

Ognuno dei dataset appena descritti possiede una propria particolarità sia per il tipo di distribuzioni utilizzate per rappresentare l'incertezza, che per la percentuale di tuple incerte, che varia dal 5%, nel dataset clinico, al 100% nel dataset meteo.

4.1.1 Dataset Plane

Tabella Plane

Attributo	**Tipo**
ID	Bigint primary key
NAME	Text
DESCRIPTION	Text
LATITUDE	Uncertain
LONGITUDE	Uncertain
DATE	Date

900.000 tuple

Dump di esempio
1 Praesent eu Integer urna. Vivamus molestie dapibus ligula. Aliquam erat (d, 1, -1.00, 1.00) (d, 1, 20.00, 1.00) 2009-01-31

2 NIROZ19487 rhoncus id, mollis nec, cursus a, enim. Suspendisse aliquet, sem (h, -21.00, -14.00, 7.00, 0.14) (h, 108.00, 115.00, 7.00, 0.14) 2001-12-21

3 PMZVW74120 mauris. Suspendisse aliquet molestie tellus. Aenean egestas hendrerit neque. In ornare sagittis felis. Donec tempor, est ac mattis (d, 1, 22.00, 1.00) (d, 1, 0.00, 1.00) 2001-04-14

4 REAYK78905 tortor. Integer aliquam adipiscing lacus. Ut nec urna et arcu (h, 9.00, 10.00, 1.00, 1.00) (h, -147.00, -146.00, 1.00, 1.00) 2006-07-19

5 SKLOE97746 sapien imperdiet ornare. In faucibus. Morbi vehicula. Pellentesque tincidunt tempus risus. Donec (h, 51.00, 56.00, 5.00, 0.20) (h, 53.00, 58.00, 5.00, 0.20) 2006-11-01

Tabella Scrap

Attributo	**Tipo**
ID	Bigint primary key
DESCRIPTION	Text
LATITUDE	Integer
LONGITUDE	Integer
DATE	Date

950.000 tuple

Dump di esempio
1 Nunc ullamcorper, velit in aliquet lobortis, nisi nibh lacinia -77 18 2003-06-10

2 ipsum nunc id enim. Curabitur massa. Vestibulum accumsan neque et nunc. Quisque ornare tortor at risus. Nunc ac -12 -60 2009-05-02

3 non, cursus non, egestas a, dui. Cras pellentesque. Sed dictum. Proin eget odio. Aliquam -72 -176 2000-09-25

4 justo. Praesent luctus. Curabitur egestas nunc sed libero. Proin sed turpis nec mauris blandit mattis. Cras eget nisi dictum augue -28 69 2010-09-26

5 nec, mollis -18 -36 2005-05-06

Descrizione

Il Dataset Plane contiene il 50% di dati incerti, i quali interessano la tabella Plane relativamente ai campi *latitude* e *longitude*. La metà certa è rappresentata con distribuzioni discrete con un solo possibile valore, mentre l'altra metà è rappresentata con delle distribuzioni uniformi che individuano un rettangolo entro il quale può trovarsi l'aereo precipitato. Su questo dataset,

oltre che le normali operazioni di selezione o aggregazione, è stato molto interessante eseguire delle query di join per testare le funzione di confronto tra dati incerti e certi. Ad esempio se un aereo si è disperso alle coordinate (15,-42) con un'area di incertezza pari a 3, nel database sarà memorizzata la seguente distribuzione: (h,15,18,3,1.0) per la latitudine e (h,15,18,3,1.0) per la longitudine.

4.1.2 Dataset Clinico

Tabella Patient

Attributo	**Tipo**
ID	Bigint primary key
NAME	Text
SURNAME	Text
BIRTH	Date

600.000 tuple

Dump di esempio
1 Iris Ewing 1972-01-11
2 Lane Gamble 1973-03-17
3 Gillian Horton 1975-09-04
4 Amy Cash 2007-08-07
5 Janna Head 2002-05-18

Diagnosis

Attributo	**Tipo**
ID	Bigint primary key
PATIENT	Bigint
DIAGNOSIS	Uncertain
DATE	Date
DESCRIPTION	Text

700.000 tuple

Dump di esempio
1 17589 (d, 1, 4298.00, 1.00) 2003-12-15 dolor quam, elementum at, egestas a, scelerisque sed, sapien. Nunc pulvinar arcu et pede. Nunc sed orci lobortis augue scelerisque mollis. Phasellus libero mauris, aliquam eu, accumsan sed,

2 201570 (d, 1, 1937.00, 1.00) 2007-08-07 massa non ante bibendum ullamcorper. Duis cursus, diam at pretium aliquet, metus urna convallis erat, eget tincidunt dui augue eu tellus. Phasellus elit pede, malesuada vel, venenatis vel, faucibus id, libero. Donec consectetuer
3 277408 (d, 1, 6603.00, 1.00) 2003-05-02 ipsum sodales purus, in molestie tortor nibh sit amet orci. Ut sagittis lobortis mauris. Suspendisse aliquet molestie tellus. Aenean egestas hendrerit neque. In ornare sagittis felis. Donec tempor, est ac mattis
4 562474 (d, 1, 7895.00, 1.00) 2001-08-07 Duis mi enim, condimentum eget, volutpat ornare, facilisis eget, ipsum. Donec sollicitudin adipiscing ligula. Aenean gravida nunc sed pede. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Proin
5 529380 (d, 1, 5213.00, 1.00) 2005-11-01 ornare, elit elit fermentum risus, at fringilla purus mauris a nunc. In at pede. Cras vulputate velit eu sem. Pellentesque ut ipsum ac mi eleifend egestas. Sed pharetra, felis eget varius ultrices, mauris ipsum porta elit, a feugiat tellus lorem eu metus. In lorem. Donec
6 559020 (d, 1, 8590.00, 1.00) 2009-08-27 erat. Vivamus nisi. Mauris nulla. Integer urna. Vivamus molestie dapibus ligula. Aliquam erat volutpat. Nulla dignissim. Maecenas ornare egestas ligula. Nullam feugiat placerat velit. Quisque varius. Nam porttitor scelerisque neque. Nullam nisl. Maecenas malesuada
7 275402 (d, 1, 5402.00, 1.00) 2002-10-19 neque vitae semper egestas, urna justo faucibus lectus, a sollicitudin
8 91289 (d, 2, 5285.00, 0.43, 7839.00, 0.57) 2001-04-16 purus mauris a nunc. In at pede. Cras vulputate velit eu sem. Pellentesque ut ipsum ac mi eleifend egestas. Sed pharetra, felis eget varius ultrices, mauris ipsum porta elit, a feugiat tellus lorem eu metus. In lorem. Donec elementum, lorem ut aliquam iaculis, lacus
9 419506 (d, 1, 774.00, 1.00) 2008-12-11 at pede. Cras vulputate velit eu sem. Pellentesque ut ipsum ac mi eleifend egestas. Sed pharetra, felis eget varius ultrices, mauris ipsum porta elit, a feugiat tellus lorem eu metus. In lorem. Donec elementum, lorem ut aliquam iaculis, lacus pede sagittis augue, eu tempor erat neque non
10 393213 (d, 1, 2204.00, 1.00) 2005-08-13 ipsum leo elementum sem, vitae aliquam eros turpis non enim. Mauris quis turpis vitae purus gravida sagittis.

700.000 tuple

Tabella Disease

Attributo	Tipo
ID	Bigint primary key
NAME	Text
DESCRIPTION	Text

1.200 tuple

Dump di esempio
37 Achondroplasia Swiss type agammaglobulinemia lobortis quis, pede. Suspendisse dui. Fusce diam nunc, ullamcorper eu, euismod ac,
39 Achondroplastic dwarfism leo. Vivamus nibh dolor,
77 Acromicric dysplasia ante. Vivamus non lorem vitae odio sagittis semper. Nam tempor diam dictum sapien. Aenean massa. Integer vitae nibh. Donec est mauris, rhoncus id, mollis
151 Anguillulosis magna. Phasellus dolor elit, pellentesque a,
173 Anodontia Sed molestie. Sed id risus quis
176 Anonychia microcephaly Donec fringilla. Donec feugiat metus sit amet ante. Vivamus non
209 Antinolo Nieto Borrego syndrome libero at auctor ullamcorper,
540 Parasitophobia et magnis dis parturient montes,
606 Peripheral blood vessel disorder lectus,
775 Rabies eu tellus. Phasellus elit pede, malesuada vel,

Descrizione

Il Dataset Clinico contiene il 5% di dati incerti, i quali interessano la tabella Diagnosis relativamente al campo *diagnosis*. Le diagnosi quindi possono essere viste come una serie di coppie formate dai valori, che si riferiscono all'id di una patologia memorizzata nella tabella Disease, e dalle probabilità attribuite a ciascuna patologia. Ad esempio se un paziente ha una probabilità del 90% di avere l'Alzheimer e del 10% di essere afflitto da Andropausa nel database verrà memorizzata la seguente distribuzione: (d, 7, 0.9, 8, 0.1).

4.1.3 Dataset Meteo

Tabella Meteo

Attributo	Tipo
PLACE	Text
SOURCE	Text
DATE	Date
TEMPERATURE	Uncertain

300.000 tuple

Dump di esempio
San Luis Obispo Station 1 1800-01-01 (g, 36.00, 0.24)
San Luis Obispo Station 2 1800-01-01 (g, 37.00, 0.26)
San Luis Obispo Station 3 1800-01-01 (g, 35.00, 0.29)
Sandy Station 1 1800-01-02 (g, 37.00, 0.24)
Sandy Station 2 1800-01-02 (g, 35.00, 0.26)
Sandy Station 3 1800-01-02 (g, 37.00, 0.29)
Deadwood Station 1 1800-01-03 (g, 32.00, 0.24)
Deadwood Station 2 1800-01-03 (g, 30.00, 0.26)
Deadwood Station 3 1800-01-03 (g, 31.00, 0.29)
Huntsville Station 1 1800-01-04 (g, 22.00, 0.24)

Descrizione

Il Dataset Meteo contiene il 100% di dati incerti, i quali interessano l'unica tabella presente in questo dataset, ovvero Meteo, relativamente al campo *Temperature*, che rappresenta la temperatura rilevata da una stazione meteo in un certo giorno in una certa località. La temperatura è rappresentata con una distribuzione Gaussiana che, come detto più volte nelle sezioni precedenti, è solitamente la distribuzione più indicata per rappresentare i dati ottenuti da sensori di rilevamento della posizione o della temperatura. Ad esempio se il sensore di una stazione di rilevamento della temperatura ha un basso margine di errore, verrà memorizzata nel database una distribuzione gaussiana con una varianza molto bassa, al contrario se il sensore ha un margine di errore piuttosto alto, verrà memorizzata una distribuzione con una varianza alta.

Capitolo 5

Analisi Sperimentale

In questo capitolo verrà descritta nei dettagli la fase principale del mio lavoro di tesi, ovvero l'analisi sperimentale sui dataset creati artificialmente (vedi sezione precedente) con alcune interrogazioni particolari, realizzate con lo scopo di mettere alla prova l'efficienza e la correttezza delle funzioni e degli operatori per la gestione e la manipolazione dei dati incerti, messi a disposizione dal DBMS Orion.

Non avendo ancora a disposizione un DBMS con indici specifici per i dati incerti (questi indici sperimentali verranno descritti nelle sezioni 6 e 7) non è stato possibile eseguire test in maniera più approfondita, ad esempio eseguendo dei join con cardinalità più alte, ma in alcuni casi è stato necessario utilizzare condizioni aggiuntive per diminuire la cardinalità.

Per il Dataset Plane invece, che rappresenta l'area spaziale di incertezza con delle distribuzioni uniformi, per puro scopo dimostrativo, è stato simulato un indice PTI con un x-bound fittizio per verificare le prestazioni nel caso fossero stati presenti gli indici.

5.1 Query

Le query utilizzate per il Benchmark sono differenti per ciascun dataset, in quanto per ognuno di loro è stato approfondito un particolare aspetto.

Ciò che accomuna tutte le query è che in ciascuna di esse sono stati raccolti i risultati al variare della soglia di probabilità, tecnicamente detta *threshold*. Questa variazione permette di verificare un'eventuale incidenza della threshold nelle prestazioni.

5.1.1 Dataset Plane

Questo dataset, come detto in precedenza, rappresenta uno scenario di aerei dispersi o precipitati, e di rottami ritrovati da attribuire, con una certa soglia di probabilità, ad uno o più aerei dispersi.

La percentuale iniziale di dati incerti è del 50%, ma per verificare il cambiamento delle prestazioni al variare della percentuale di dati incerti sono state eseguite le stesse interrogazioni anche con il 10%, 20%, 30% e 40% di dati incerti.

Le query utilizzate per il benchmark sono le seguenti:

- Tutti gli aerei attribuibili, con una probabilità superiore ad una certa soglia, a rottami ritrovati a precise coordinate:

```
SELECT P.id, P.name, S.id
FROM Plane P join Scrap S on (S.date - P.date > 0)
WHERE u_eq(P.latitude, S.latitude::real) > prob and
u_eq(P.longitude, S.longitude::real) > prob and
S.latitude = latitude and S.longitude = longitude
order by P.id, P.name;
```

- Tutti gli aerei attribuibili, con una probabilità superiore ad una certa soglia, ad uno specifico rottame

```
SELECT P.id, P.name, P.date
FROM Plane P join Scrap S on (S.date > P.date)
WHERE u_eq(P.latitude, S.latitude::real) > prob and
u_eq(P.longitude, S.longitude::real) > prob and
S.id = scrap_id;
```

Come si può vedere, sono entrambe *join query*, con l'unica differenza che la prima filtra i rottami in base alla latitudine e alla longitudine, la seconda invece filtra un solo specifico rottame.

5.1.2 Dataset Clinico

Questo dataset, rappresenta uno scenario decisamente più quotidiano e pragmatico, infatti rappresenta uno scenario nel quale vengono raccolte alcune diagnosi per determinati pazienti. Le diagnosi quindi rappresentano il dato

incerto, in quanto una diagnosi può non essere certa, e quindi il medico attribuisce alcune percentuali di probabilità ad alcune patologie.

Questo dataset presenta una bassa percentuali di dati incerti, solo il 5%, e le query utilizzate per il benchmark sono state le seguenti:

- I nomi e cognomi dei pazienti che hanno una probabilità di essere affetti da una certa patologia superiore ad una certa soglia:

```
select patient.name, patient.surname
from patient join diagnosis on
(diagnosis.patient = patient.id), disease
where disease.id = disease_id and
u_eq(disease.id::real, diagnosis) > prob;
```

- Per una particolare patologia e per ciascun anno, il numero di pazienti che hanno una probabilità di essere affetti superiore ad una certa soglia:

```
select disease.name, date_part('year', date) as year,
count(patient) as NPatient
from disease, diagnosis
where u_eq(disease.id::real, diagnosis.diagnosis) > prob and
disease.id = disease_id
group by disease.name, year;
```

La prima query ha una semantica molto semplice, ed è certamente una delle interrogazioni più semplici e più frequenti che si possano eseguire su un dataset di tipo medico, consiste infatti nella raccolta dei nomi e cognomi dei pazienti che, in base alle diagnosi, hanno una probabilità di essere affetti da una certa patologia superiore ad una determinata soglia.

La seconda query raccoglie dati a fini statistici, ed è utile ad esempio per analizzare l'aumento, o eventualmente la diminuizione, delle diagnosi di una certa patologia.

5.1.3 Dataset Meteo

Il dataset meteo rappresenta uno scenario nel quale esistono, in varie località (in totale 10), tre stazioni di rilevamento della temperatura, e per ciascun giorno e località vengono riportare le temperature rilevate dalle tre stazioni,

che tra di loro sono leggermente differenti.

Questo dataset contiene un'altissima percentuale di dati incerti, il 100%, in quanto ogni rilevazione è rappresentata con una distribuzione gaussiana in cui la previsione è la temperatura presunta rilevata dal sensore, e la varianza rappresenta in qualche modo il margine di imprecisione dovuto sia ad un possibile errore del sensore che alla variabilità della temperatura in quel determinato giorno.

Le query utilizzate sono le seguenti:

- Media delle temperature previste dalle stazioni di rilevamento per ciascuna città in cia-
  scuna data:

  ```
  SELECT date, place, avg(u_expected(temperature)) as avg_temperature
  FROM meteo
  GROUP BY date, place
  ORDER BY avg_temperature desc;
  ```

- Media della temperatura giornaliera solo per quei giorni in cui le varie rilevazioni sono molto simili tra di loro:

  ```
  select M1.Date, M1.Place, AVG.avg_temperature
  from Meteo M1, Meteo M2, Meteo M3, AVG
  where (M1.Date = M2.Date and M2.Date = M3.Date and
  M1.Place = M2.Place and M2.Place = M3.Place and
  M1.Source < M2.Source and M2.Source < M3.Source and
  u_eq(M1.Temperature, M2.Temperature) >= 0.8 and
  u_eq(M2.Temperature, M3.Temperature) >= 0.8 and
  u_eq(M1.Temperature, M3.Temperature) >= 0.8 and
  M1.Date = AVG.Date and M1.Place = AVG.Place)
  ```

  dove AVG è la vista associata alla query precedente

È interessante notare che. mentre la prima query ha lo scopo di sperimentare l'utilizzo dei dati incerti nell'utilizzo degli operatori aggregati, la seconda query è decisamente più complessa, in quanto prevede un *4-way join* della tabella meteo su se stessa, quindi anche un *self-join*.

5.2 Risultati

Di seguito verranno presentati, e spiegati, i risultati ottenuti dal Benchmark sui dataset visti in precedenza. In particolare ci sarà una particolare enfasi, sulle variazioni di performance riscontrate al variare della percentuali di dati incerti e al variare della soglia di probabilità.

In particolare nei test eseguiti sul dataset plane, è stato eseguito uno studio incrociato sia sulla variazione della soglia di probabilità che sulla percentuale dei dati incerti.

In alcuni grafici è visibile in corrispondenza del passaggio da una treshold all'altra un brusco calo della funzione tridimensionale, in realtà tutto ciò non ha alcun significato reale ma si tratta semplicemente di un effetto grafico per unire i due punti, rispettivamente l'ultimo della treshold precedente e il primo della treshold corrente.

5.2.1 Dataset Plane: distribuzione uniforme

In questo dataset per la rappresentazione dei dati incerti, è stata utilizzata la distribuzione uniforme.

Come si può notare dalle Figure 5.1 e 5.3, il numero di IO ha un andamento abbastanza regolare e prevedibile, infatti aumentando la percentuale di dati incerti dal 10% al 50%, si ha un incremento del numero di blocchi letti dal disco notevole, in quanto inizialmente si ha un leggero incremento, ma tra il 30% e il 50% l'incremento è decisamente maggiore ma costante.

Per quanto riguarda i tempi di esecuzione invece, come si può notare guardando le Figure 5.2 e 5.4, si ha un forte aumento rispetto alla percentuale di dati incerti, con un comportamento molto simile, se non identico, rispetto al numero di IO.

Le due query utilizzate per il benchmark quindi, pur essendo sostanzialmente diverse tra di loro, evidenziano entrambe un comportamento comune rispetto alla variazione della quantità di dati incerti.

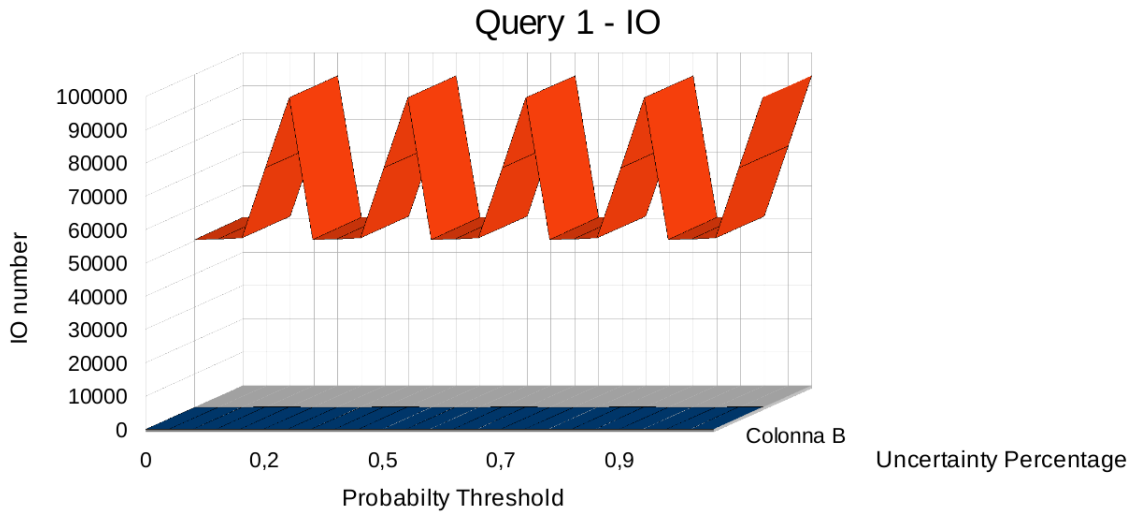

Figura 5.1: Dataset Plane: Risultati della query 1 relativamente al numero di IO

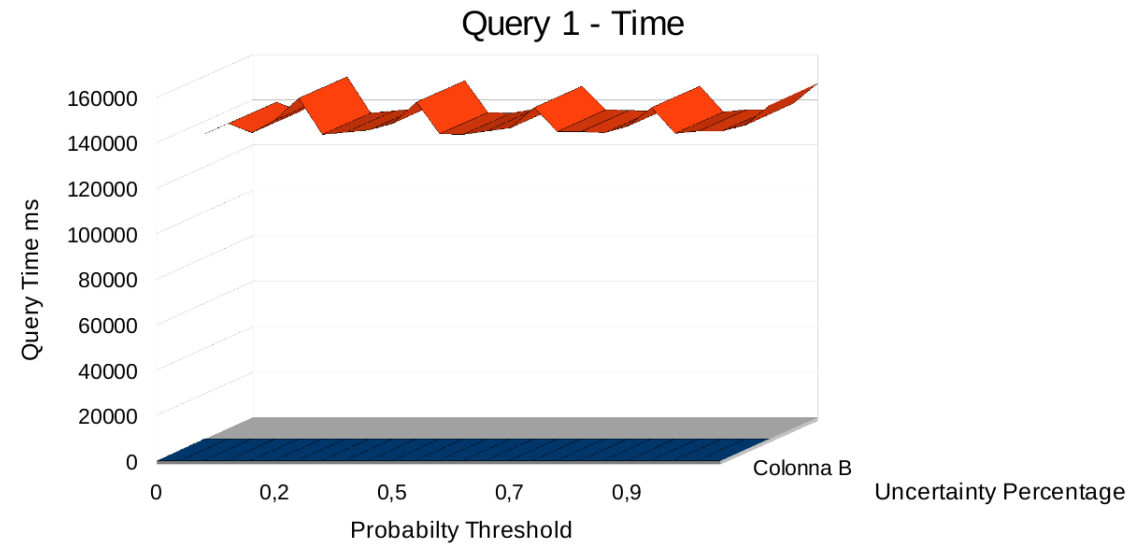

Figura 5.2: Dataset Plane: Risultati della query 1 relativamente al tempo impiegato

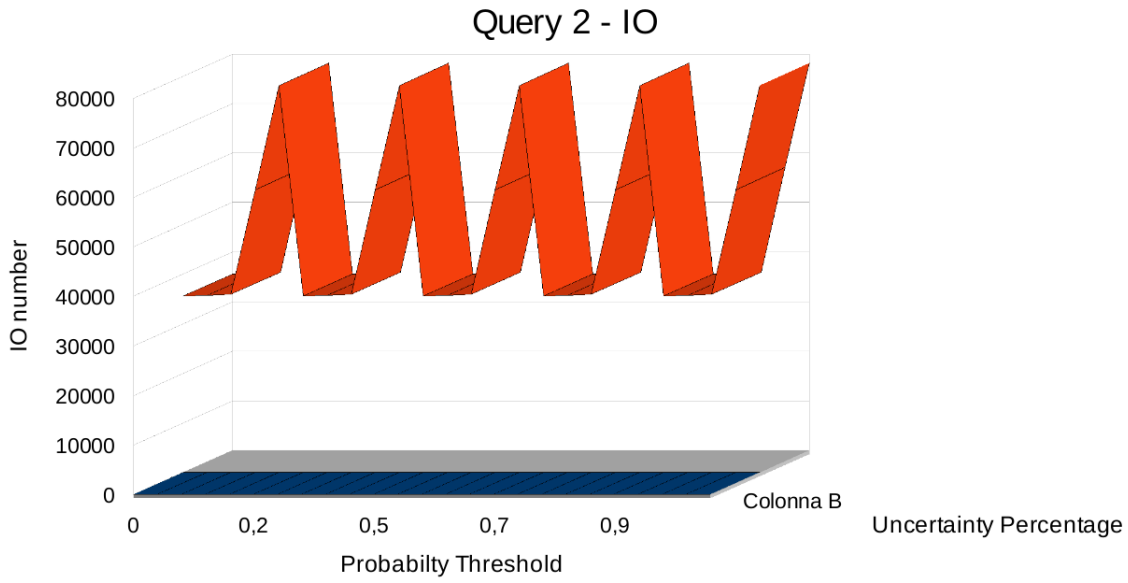

Figura 5.3: Dataset Plane: Risultati della query 2 relativamente al numero di IO

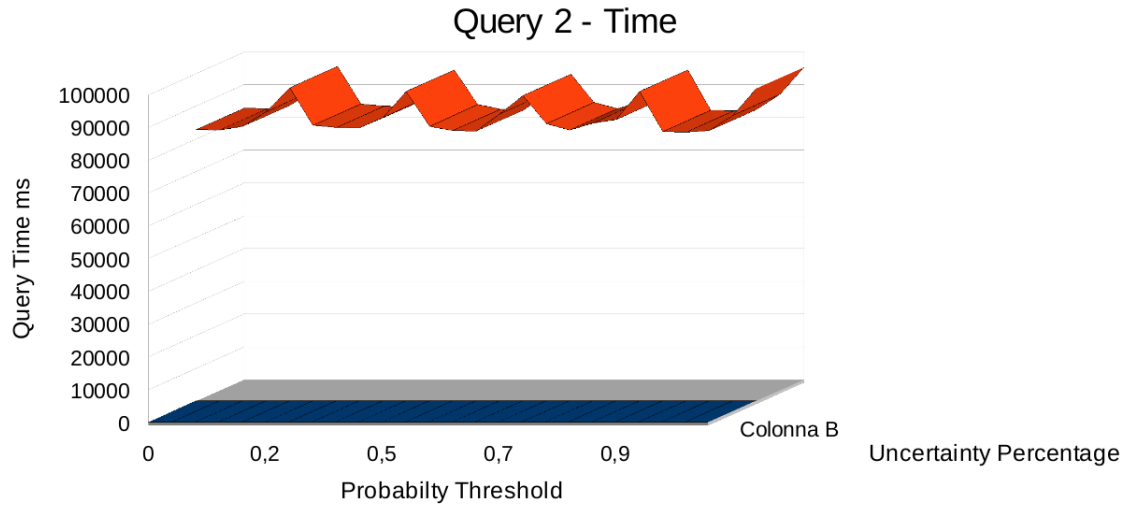

Figura 5.4: Dataset Plane: Risultati della query 2 relativamente al tempo impiegato

5.2.2 Dataset Clinico: distribuzione discreta

In questo dataset per la rappresentazione dei dati incerti è stata utilizzata la distribuzione discreta.

A differenza dei risultati riguardanti gli altri dataset, in questo dataset alcuni risultati potrebbero sembrare difficilmente interpretabili, in quanto, a giudicare dai risultati illustrati nelle Figure 5.5, 5.6, 5.7 3 5.8 il numero di IO rimane inviariato al variare della soglia di probabilità, ma questo è perfettamente spiegabile dal fatto che non sono presenti indici sui dati incerti. Ciò che è strano invece, e che difficilmente trova una spiegazione plausibile, è la variazione del tempo di esecuzione, che tende sempre ad avere un andamento molto particolare, in quanto dopo la soglia 0.7 tende a crescere.

Questo potrebbe essere in parte spiegato dall'algoritmo utilizzato per l'implementazione della funzione *u_eq(uncertain, real)*, in particolare nel caso di una distribuzione discreta:

```
/**************** Internal function to check uncertain =_c v */

float4 equal_const(Uncertain *input, float4 v, float4 c, int N) {

  ...

  switch(input->type) {

  ...

  case 'd':
    d = (Discrete *) input->data;

    for(i=0; i<d->num_data; i++)
      if (v == d_point(d->data[i]))
return(d_prob(d->data[i]));

    return(0);

    break;

    ...
```

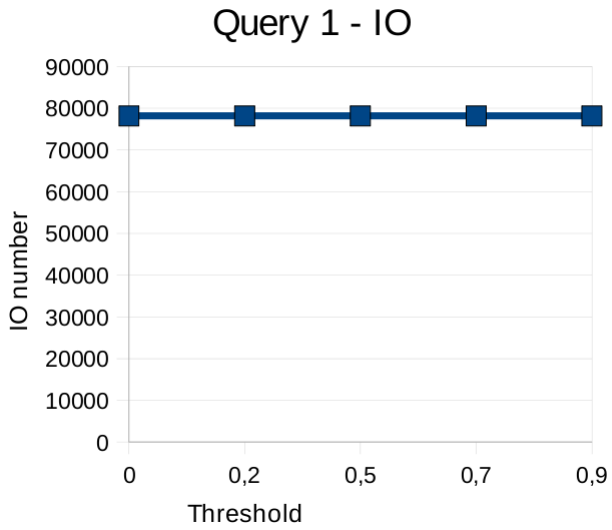

Figura 5.5: Dataset Clinico: Risultati della query 1 relativamente al numero di IO

```
  }

  ...
}
```

Come si può notare, nel caso in cui ci sia un confronto tra una variabile incerta rappresentata con una distribuzione discreta ed un valore certo, un numero reale, non viene sfruttata la caratteristica dell'ordinamento dei punti della distribuzione discreta. Si può ipotizzare quindi, che per verificare che esista o meno un valore nella distribuzione che soddisfi la soglia di probabilità, la funzione venga richiamata più di una volta.

5.2.3 Dataset Meteo: distribuzione gaussiana

In questo dataset per la rappresentazione dei dati incerti, è stata utilizzata la distribuzione gaussiana.

Su questo dataset sono state eseguite due interrogazioni: la prima è semplicemente una selezione con l'utilizzo di un operatore aggregato per il calcolo della media delle temperature, la seconda invece è un'interrogazione decisamente più interessante, in quanto non solo si tratta di un join piuttosto che

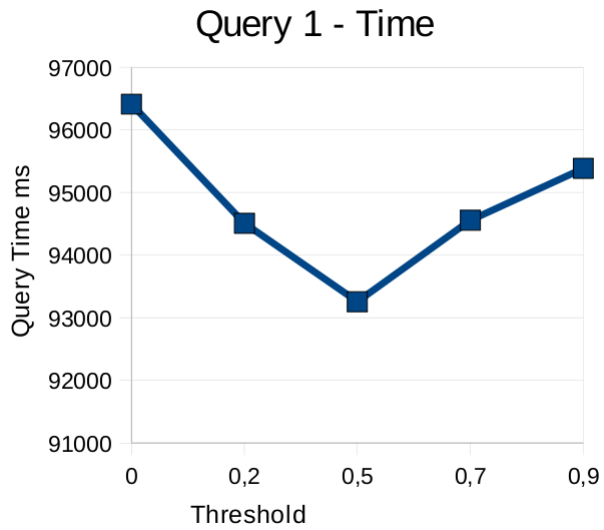

Figura 5.6: Dataset Clinico: Risultati della query 1 relativamente al tempo impiegato

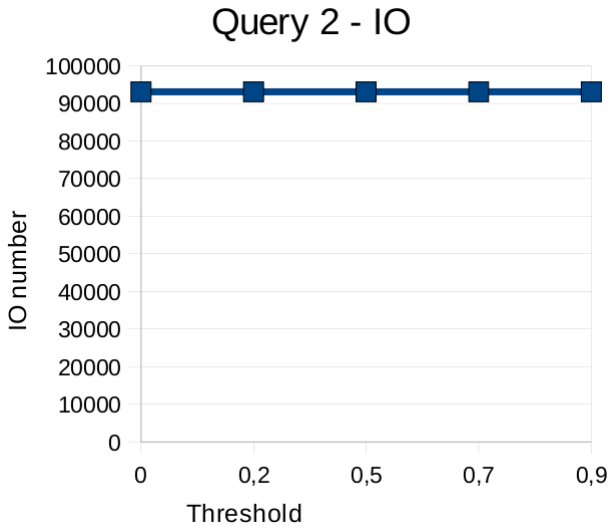

Figura 5.7: Dataset Clinico: Risultati della query 2 relativamente al numero di IO

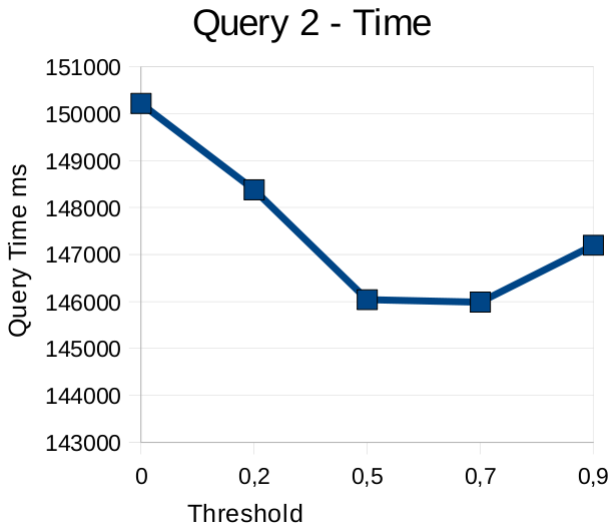

Figura 5.8: Dataset Clinico: Risultati della query 2 relativamente al tempo impiegato

una semplice selezione, ma è un *4-way join*, dal punto di vista delle performance sicuramente molto più interessante.

Inoltre, le varie interrogazioni sono state eseguite incrementando progressivamente la varianza, partendo da un valore molto basso come 0.5, fino ad arrivare ad un valore estremamente alto, 10.

I dati raccolti hanno riscontrato due fatti principali:

- **Selezione**: per quanto riguarda la selezione, l'incremento progressivo della varianza ha influito minimamente per quanto riguarda il tempo di esecuzione, e assolutamente per niente per quanto riguarda il numero di IO
- **4-way join**: per quanto riguarda il join, l'incremento progressivo della varianza, ha inciso sia sul tempo di esecuzione che sul numero di IO, infatti con varianze molto alte, quindi maggiori o uguali a 1.0, le condizioni presenti nell'interrogazione non erano valide per nessuna tupla, portando quindi la cardinalità del risultato a 0. Questo ha causando quindi non solo tempi di esecuzione più bassi ma anche un numero di IO inferiore.

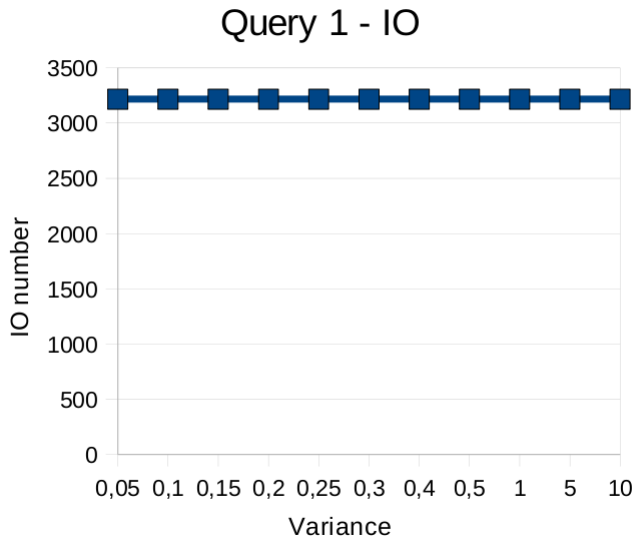

Figura 5.9: Dataset Meteo: Risultati della query 1 relativamente al numero di IO

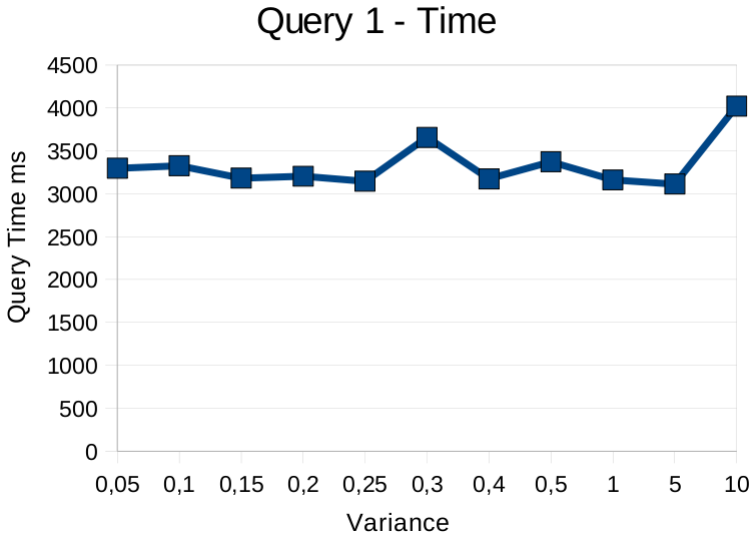

Figura 5.10: Dataset Meteo: Risultati della query 1 relativamente al tempo impiegato

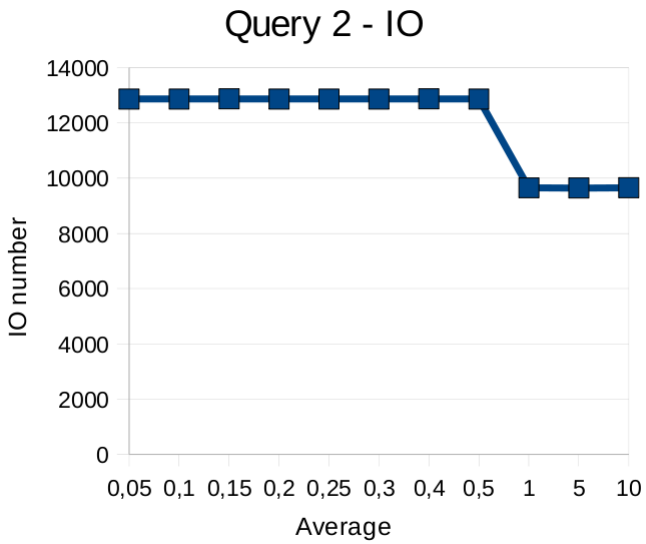

Figura 5.11: Dataset Meteo: Risultati della query 2 relativamente al numero di IO

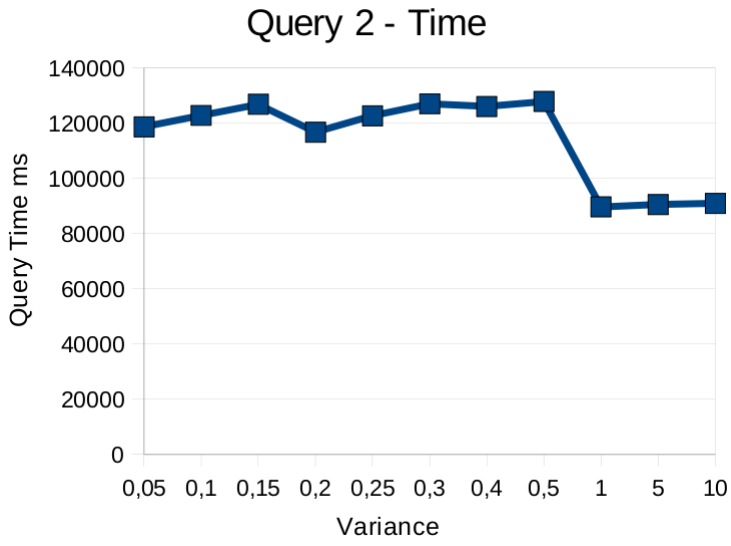

Figura 5.12: Dataset Meteo: Risultati della query 2 relativamente al tempo impiegato

5.3 Riflessioni

I risultati ottenuti dal benchmark hanno evidenziato alcuni fattori:

- La percentuale di dati incerti, ovvero il numero di tuple con attributi con valori incerti, è direttamente proporzionale sia al tempo di esecuzione delle interrogazioni che al numero di IO
- La soglia di probabilità (threshold) sembra non incidere particolarmente nel risultato, questo è dovuto probabilmente alla mancanza degli indici e, come mostrato in precedenza, alla poca efficienza degli algoritmi per il calcolo della probabilità
- I dati emersi dal benchmark evidenziano inoltre una maggiore inefficienza della gestione delle distribuzioni discrete rispetto alla gestione delle distribuzioni *Histogram* o *Gaussian*
- La distribuzione *Gaussian* sembra essere la distribuzione gestita più efficientemente

Capitolo 6

Indici per dati incerti

6.1 Orion 2.0

Come citato in precedenza, è in fase di sviluppo una nuova versione, molto più completa, del DBMS Orion, la versione 2.0.

Questa versione dovrebbe finalmente introdurre un modello dei dati molto più completo e flessibile, oltre che l'implementazione delle strutture dati per gli indici. Soprattutto questa ultima feature, pur essendo in attesa di verifica, porterà probabilmente ad una maggiore efficienza della gestione dei dati incerti, aumentando la possibilità di utilizzare questi sistemi per scopi reali e/o commerciali.

6.1.1 Modello dei dati

Il modello dei dati di Orion 2.0 prevede, come per la versione precedente, due tipi di attributi: certi e incerti. A differenza della versione precedente però, gli attributi incerti sono suddivisi in quattro *sottotipi*, ciascuno dei quali è trattato in maniera specifica, con indici e algoritmi ottimizzati per ciascun tipo di distribuzione [Prabhakar [2008a]].

Orion 2.0 prevede quattro tipi di dati incerti:

- **Distribuzioni numeriche continue** (*ucon*): ogni elemento è associato ad una funzione di densità per calcolare la probabilità di un determinato valore.
  Esempio: la temperatura ottenuta da un sensore
- **Distribuzioni numeriche discrete** (*udis*): ogni elemento ha una funzione di densità (discreta) che associa a ciascuna alternativa una

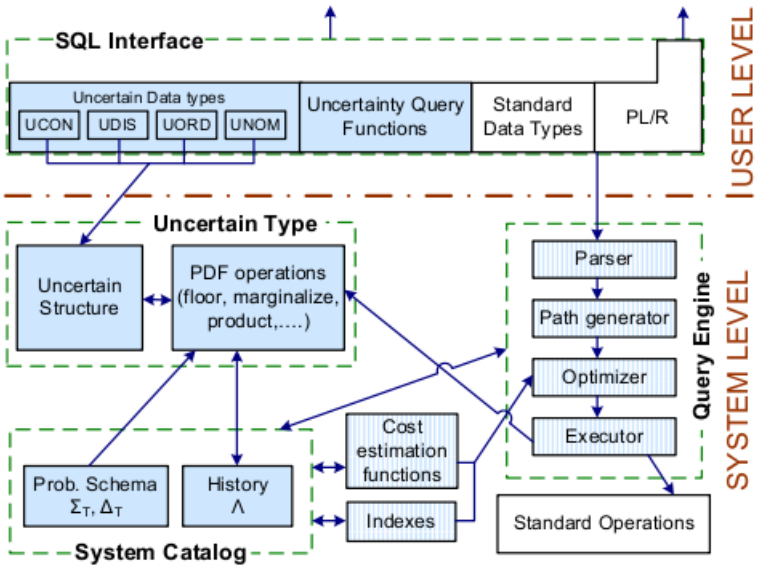

Figura 6.1: Architettura di Orion 2.0

probabilità.
Esempio: Numero di vicini in una rete di telefoni cellulari

- **Catagorici ordinati** (*uord*): simile alle distribuzioni numeriche, ogni elemento è visto come una funzione di densità che memorizza le probabilità per ciascuna categoria.
  Esempio: Fuzzy data

- **Categorici non ordinati** (*unom*): come i categorici ordinati, ma non esiste un ordine logico tra le categorie.
  Esempio: Classificazione dei documenti (keyword) (vedi Prabhakar [2007])

Gli attributi certi invece vengono gestiti esattamente come previsto in un classico DBMS, al contrario degli attributi incerti che richiedono particolari strutture dati per l'indicizzazione e per le operazioni di *selezione. proiezione e join*, oltre che algoritmi specifici per la selezione e il join.

Ogni tabella T è rappresentata da uno schema definito nel seguente modo:

- Σ_T l'insieme degli attributi (sia certi che incerti) appartenenti alla tabella, con informazioni sul loro tipo e nome

- Δ_T informazioni sugli attributi che possiedono distribuzioni di probabilità congiunte

Capitolo 7

Strutture dati per l'indicizzazione

Di seguito verrà descritta la struttura dati principale utilizzata dal DBMS Orion per l'indicizzazione dei dati incerti.

7.1 Inadeguatezza delle strutture dati tradizionali

Provare a gestire i dati incerti con le stesse metodologie previste per i dati certi, risulta essere poco efficiente in quanto si provoca un notevole degrado delle prestazioni e un aumento dello spazio occupato.

Ecco un esempio:

Id Auto	**Problema**	**Probabilità**
Car 1	Brake	0.1
Car 1	Tires	0.9
Car 2	Trans	0.2
Car 2	Suspension	0.8

Provando a gestire l'incertezza di questo piccolo database con un DBMS che prevede l'esistenza di soli dati certi, otterremmo innanzitutto 4 tuple invece che 2, e ancor peggio saremmo costretti a ripetere informazioni univoche come l'id dell'auto per ogni alternativa possibile dell'attributo incerto, ovvero il problema riscontrato nell'auto.

Inoltre bisognerebbe tener conto anche di situazioni più particolari ma non insolite, ad esempio distribuzioni continue anzichè discrete, come la gaus-

siana, molto adatte a rappresentare l'incertezza in sistemi di misurazione. In questo caso sarebbe molto difficile, riuscire a rappresentare questo tipo di distribuzioni col metodo visto in precedenza, la soluzione più efficiente probabilmente è quella di offrire un supporto nativo da parte del DBMS.

Le strutture dati standard utilizzate per l'indicizzazione sostanzialmente hanno due inconvenienti che le rendono inservibili per l'indicizzazione dei dati incerti:

- Non esiste un supporto nativo, quindi all'interno del DBMS, per la gestione delle varie distribuzioni di probabilità esistenti, che quindi con vari artefizi dovrebbero essere approssimate con valori discreti, che provocano di conseguenza un errore di approssimazione inversamente proporzionale al numero di punti utilizzati per l'approssimazione;
- Sarebbe preferibile, e comporterebbe un incremento dell'efficienza durante le interrogazioni, riuscire ad eliminare a priori alcune tuple dal risultato finale, senza essere costretti per ciascuna a calcolare la probabilità sull'intervallo desiderato, che comporterebbe il calcolo di un integrale, che è un'operazione con una complessità computazionale più che lineare.

Questo è un esempio di query in cui gli indici tradizionali non sarebbero adatti:

```
SELECT u_expected(a), u_expected(b)
FROM T
WHERE u_eq(a,b) >= 0.5
```

Supponiamo di avere una tabella T con due attributi incerti, a e b, e vogliamo ottenere le previsioni di a e b quando hanno una probabilità di essere *simili* superiore allo 0.5. Utilizzando i tradizionali indici sarebbero necessarie numerose computazioni per calcolare la probabilità di similarita tra le due distribuzioni e le loro previsioni, l'obiettivo invece è quello di ridurre al minimo queste computazioni inserendo alcune informazioni aggiuntive all'interno degli indici.

7.2 Strutture dati per l'indicizzazione

Di seguito verrà descritta la struttura dati che è allo studio del gruppo di ricera di Orion DBMS 2.0, che dovrebbe essere rilasciato entro breve. Prabhakar [2008d]

7.2.1 PTI (Probability Threshold Indexing)

Questo indice è una variante di una struttura dati già nota, gli R-tree.

Come abbiamo già detto in precedenza, una buona struttura dati per l'indicizzazione dei dati incerti, deve permettere di escludere a priori dal risultato dell'interrogazione, alcune tuple che certamente non apparterranno al risultato, in modo tale da evitare un inutile e non pratico calcolo della probabilità, che ricordiamo si tratta del calcolo di un integrale multidimensionale, quindi un calcolo con una complessità computazionale più che lineare. Questa operazione prende il nome di *pruning*.

Il metodo utilizzato da questa struttura dati per ottenere questo risultato, prevede innanzitutto la memorizzazione delle informazioni sulle tuple nelle sole foglie (caratteristica ereditata dai B^+-tree), memorizzando invece nei nodi interni gli *MBR* (Minimum Bounding Rectangle).

Durante un'interrogazione del tipo PTQ (Probabilistic Threshold Query), l'intervallo preso in input viene confrontato con ogni nodo figlio e se ha un'intersezione nulla viene immediatamente eliminato dal risultato finale.

Se invece l'intersezione è non nulla, vengono utilizzati gli x-bound per determinare se le tuple presenti nei nodi figli possono appartenere al risultato oppure essere immediatamente scartate.

Definizione 11 *L'x-bound inferiore(superiore) di un nodo N_j indicato con $N_j.lb(x)$ ($N_j.ub(x)$) è il valore massimo tale da garantire che per ogni intervallo [L_i, U_i] contenuto nel sottoalbero con radice N_j, la probabilità di essere inferiore (superiore) a $N_j.lb(x)$($N_j.ub(x)$) sia al massimo x.*
Questo equivale a dire il seguente:
$\int_{L_i}^{N_j.lb(x)} f_i dy \leq x$ *e* $\int_{U_i}^{N_j.ub(x)} f_i dy \leq x$

Ricapitolando un nodo può essere eliminato definitivamente dal risultato di una query Q sull'intervallo [a,b] con una soglia p se e solo se le seguenti due condizioni valgono per uno tra gli x-bounds inferiori o superiori del nodo:

- $[a, b]$ si trova al di sotto (al di sopra) dell'x-bound inferiore(superiore) di N_j, ad esempio b $< N_j$.lb(x) o a $> N_j$.ub(x)
- p $\geq$ x

7.3 Esempio di costruzione di un indice PTI

Per realizare l'esempio di costruzione di un indice basato su PTI utilizzeremo per semplicità la distribuzione uniforme, senza rischiare di mancare di generalità.
Abbiamo una tabella T, con due attributi a, b il primo certo e il secondo incerto.

Innanzitutto la query da eseguire nel sistema Orion è la seguente:

```
CREATE table T (
    a INTEGER primary key,
    b UNCERTAIN);

INSERT INTO T VALUES (1, '(h, 4, 12, 8, 1.0)');
INSERT INTO T VALUES (2, '(h, 1, 9, 8, 1.0)');
```

Otteniamo quindi la seguente tabella:

a	**b**
1	uniform(4,12)
2	uniform(1, 9)

Al termine di queste operazioni, possiamo eseguire una semplice *range query* per ottenere tutte le tuple che appartengono all'intervallo [1,5] con una probabilità maggiore o uguale di 0.20 :

```
SELECT b FROM T WHERE u_prob(b, 1, 5) >= 0.20
```

Nell'immagine 2.1 è raffigurato l'indice PTI per questa piccola tabella.
I due MBR contengono le seguenti informazioni:

- Il primo MBR, l'MBR 1, ha come lower bound 4 e come upper bound 12 in quanto il minimo estremo inferiore degli intervalli contenuti nei suoi figli è 4, e il massimo estremo superiore è 12.
- La stessa cosa vale per l'MBR 2

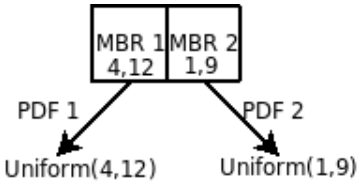

Figura 7.1: Indice PTI

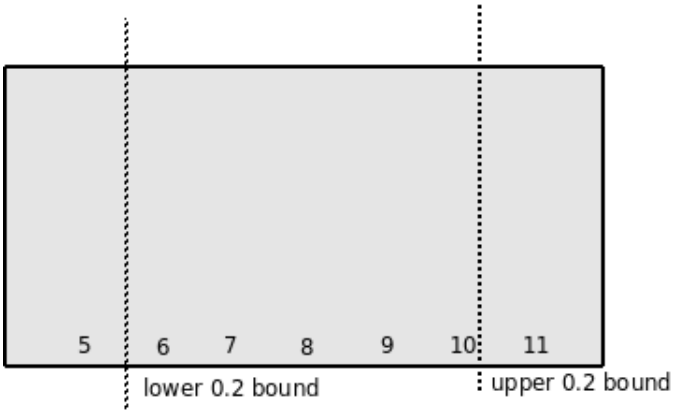

Figura 7.2: x-bound MBR 1

In figura 7.2 è raffigurato l'x-bound della prima foglia, ovvero quella che verrà esclusa dal risultato finale.

Come si può notare, l'intervallo [4,5] è a sinistra dello *0.2-lower-bound*, quindi per questa tupla, la probabilità di appartenere all'intervallo appena menzionato è inferiore a 0.2, e questo ci permette con estrema facilità di poter eliminare a priori questa tupla dal risultato finale.

La stessa cosa invece non avviene per la tupla 2, che dopo un appropriato calcolo, includiamo nel risultato finale in quanto la probabilità di appartenere all'intervallo [1,5] è superiore allo 0.2.

Riassumendo, l'introduzione degli x-bound nelle strutture dati classiche per l'indicizzazione, permette in molti casi, di poter eliminare molte tuple dal risultato finale, senza dover computare la loro probabilità di appartenere all'intervallo desiderato (in questo caso [1,5]).

Inoltre, vale la pena chiarire che i vari x-bound memorizzati nell'indice sono **precalcolati** per determinate soglie di probabilità, quindi non sono in alcun modo scelti dall'utente che crea la tabella.

7.4 Simulazione degli indici

Una delle principali motivazioni che spinge gli utenti finali ad utilizzare un DBMS piuttosto che un semplice file di archivio, ad esempio un file .dat, è sicuramente quella di avere a disposizione strumenti molto preziosi per aumentare l'efficienza sia nell'inserimento che nell'estrazione dei dati, per non parlare poi della possibilità di utilizzare varie funzioni per la manipolazione di essi.

Mentre questi ultimi sono presenti anche in Orion DBMS, gli indici per i dati incerti, come già detto in precedenza, non sono ancora stati implementati, quindi non è possibile disporre di indici specifici per questo tipo di dati, di conseguenza i dati ottenuti dal benchmark sono incompleti.

Tuttavia, almeno per quanto riguarda il dataset plane, cioè quello con distribuzioni uniformi, è stato possibile *simulare* gli indici PTI, di cui se ne parlerà approfonditamente nelle sezione successive, creando x-bound fittizi, in particolare gli x-bound corrispondenti alla soglia di probabilità 0.25.

La strategia utilizzata è molto semplice: sono stati aggiunti due attributi per ciascun attributo incerto, con lo scopo di assolvere le funzioni di *lower x-bound* e *upper x-bound*. Quindi la struttura della tabella *Plane* è cambiata nel seguente modo:

Attributo	**Tipo**
ID	Bigint primary key
NAME	Text
DESCRIPTION	Text
LATITUDE	Uncertain
UPPER_LAT	Real
LOWER_LAT	Real
LONGITUDE	Uncertain
UPPER_LON	Real
LOWER_LON	Real
DATE	Date

Di seguito vengono mostrati invece la struttura della tabella memorizzata nel database e gli indici creati appositamente:

```
   Column    |   Type    | Modifiers
-------------+-----------+-----------
 id          | bigint    | not null
 name        | text      |
 description | text      |
 latitude    | uncertain |
 longitude   | uncertain |
 date        | date      |
 upper_lat   | real      |
 upper_lon   | real      |
 lower_lon   | real      |
 lower_lat   | real      |

Indexes:
    "aereo_pkey" PRIMARY KEY, btree (id)
    "lower_lat_idx" btree (lower_lat)
    "lower_lon_idx" btree (lower_lon)
    "upper_lat_idx" btree (upper_lat)
    "upper_lon_idx" btree (upper_lon)
```

Con l'inserimento di questi indici fittizi, ma nella sostanza uguali a quelli reali, otteniamo tuple simili a queste:

```
 ...       | upper_lon | lower_lon | upper_lat | lower_lat
-----------+-----------+-----------+-----------+----------
 ...       |        20 |        20 |        -1 |        -1
 ...       |    113.25 |    109.75 |    -15.75 |    -19.25
 ...       |         0 |         0 |        22 |        22
 ...       |   -146.25 |   -146.75 |      9.75 |      9.25
 ...       |     56.75 |     54.25 |     54.75 |     52.25
 ...       |        54 |        54 |       -14 |       -14
 ...       |      -172 |      -172 |       -47 |       -47
 ...       |    -63.25 |    -65.75 |     82.75 |     80.25
 ...       |       150 |       150 |       -52 |       -52
 ...       |      -177 |      -177 |       -40 |       -40
 ...       |      -177 |      -177 |        68 |        68
 ...       |     47.25 |     45.75 |     39.25 |     37.75
 ...       |     56.25 |     52.75 |    -61.75 |    -65.25
 ...       |       143 |       143 |        63 |        63
 ...       |       -19 |       -21 |       -77 |       -79
 ...       |      -112 |      -112 |        -2 |        -2
 ...       |        91 |        91 |        71 |        71
 ...       |     -12.5 |     -13.5 |     -70.5 |     -71.5
 ...       |        38 |        38 |        38 |        38
 ...       |        35 |        35 |        41 |        41
```

Prendiamo ad esempio la seconda tupla mostrata nel dump precedente, *upper* e *lower_lon* rappresentano rispettivamente l'upper e il lower x-bound per la soglia di probabilità 0.25, come illustrato in Figura 7.4.

Durante le interrogazioni quindi, è possibile scartare a priori dal risultato le tuple che sicuramente non appartengono ad un certo intervallo con una probabilità superiore alla soglia (*threshold*) prestabilita, realizzando di fatto la funzione propria di un indice. Questa operazione infatti, è detta tecnicamente *pruning*, ed è tuttora un discusso argomento di ricerca.

Ad esempio, se l'intervallo di ricerca e la soglia fossero rispettivamente [-15,-14] 0.25 potremmo determinare in anticipo, senza calcolare l'integrale per ottenere la probabilità, che la tupla che abbiamo preso come esempio non appartiene al risultato. Questo risparmio in dataset molto grandi, come quello sperimentale di cui abbiamo parlato precedentemente, si traduce in un notevole guadagno di performance.

Per testare l'efficienza del DBMS con l'introduzione di questi indici fit-

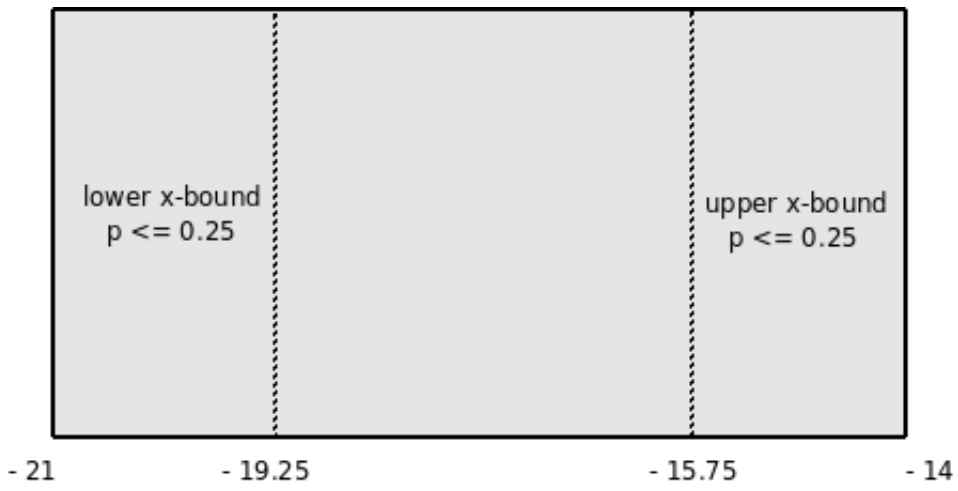

Figura 7.3: Simulazione dell'indice PTI

tizi, è stata utilizzata una semplice probabilistic threshold query di selezione, detta anche range query, la quale restituisce gli id e i nomi degli aereoplani che hanno una probabilità superiore a 0.25 di appartenere all'aerea con latitudine compresa nell'intervallo [15,17] e longitudine compresa nell'intervallo [-42,-40].

La query risultante nel caso di non utilizzo degli indici è la seguente:

```
SELECT id, name
FROM plane
WHERE u_prob(latitude, 15, 17) > 0.25 and
u_prob(longitude, -42,-40) > 0.25
```

mentre nel caso di utilizzo degli indici, la query equivalente è la seguente:

```
SELECT id, name
FROM plane
WHERE (not(15 > upper_lat)) and (not(17 < lower_lat)) and
(not(-42 > upper_lat)) and (not(-40 < lower_lat)) and
u_prob(latitude, 15, 17) > 0.25 and
u_prob(longitude, -42,-40) > 0.25
```

A conferma della correttezza di questi indici c'è la cardinalità del risultato, che in entrambe le query è uguale: col 50% di dati incerti è 98 invece col

10% è 65, con risultati identici.

Come si può vedere dalle Figure 7.4, 7.5, 7.6, 7.7 l'utilizzo degli indici comporta un notevole miglioramento delle prestazione sia in termini di tempo che in termini di numero di IO necessari, in quanto, come detto in precedenza, permette di evitare il calcolo della probabilità quando è *impossibile* che la tupla soddisfi le condizioni previste nella query, in questo caso quando l'intervallo di ricerca si trova tutto a sinistra(destra) del lower(upper) x-bound.

Nel caso preso come esempio, gli x-bound a disposizione erano esattamente uguali alla soglia di probabilità utilizzata nella query, questo però è solo un caso dovuto al carattere dimostrativo di questo test. Potrebbe capitare infatti, che non ci sia un x-bound uguale alla soglia di probabilità, in questo caso quindi basterebbe utilizzare come x-bound di riferimento il massimo tra quelli inferiori alla soglia.

7.4.1 Risultati e Riflessioni

Con l'introduzione di indici specifici per i dati incerti, il miglioramento delle prestazione è stato notevole, infatti il numero IO si è dimezzato e i tempi di esecuzione sono calati drasticamente fino ad arrivare al 4% o al 5%.

I dati ottenuti quindi, ci permettono di asserire con una certa sicurezza che con un appropriato supporto di indicizzazione (*x-bound*) e con una migliore implementazione delle funzioni per la gestione dei dati incerti, l'overhead per l'introduzione di questi ultimi sarebbe davvero molto baso, addirittura quasi nullo, poichè come abbiamo potuto osservare una selezione probabilistica su tutta la tabella, sfruttando gli indici, ha impiegato un tempo decisamente paragonabile ad una selezione senza acun dato incerto.

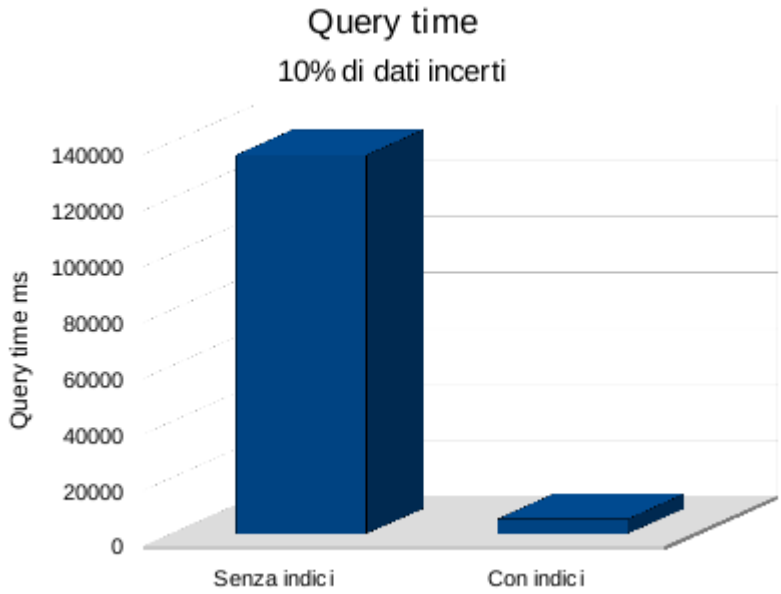

Figura 7.4: 10% di dati incerti: tempo

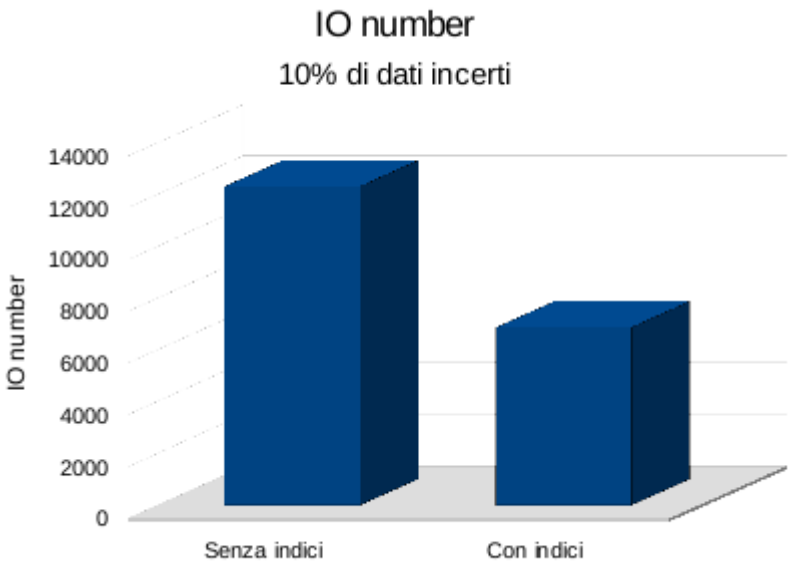

Figura 7.5: 10% di dati incerti: numero di IO

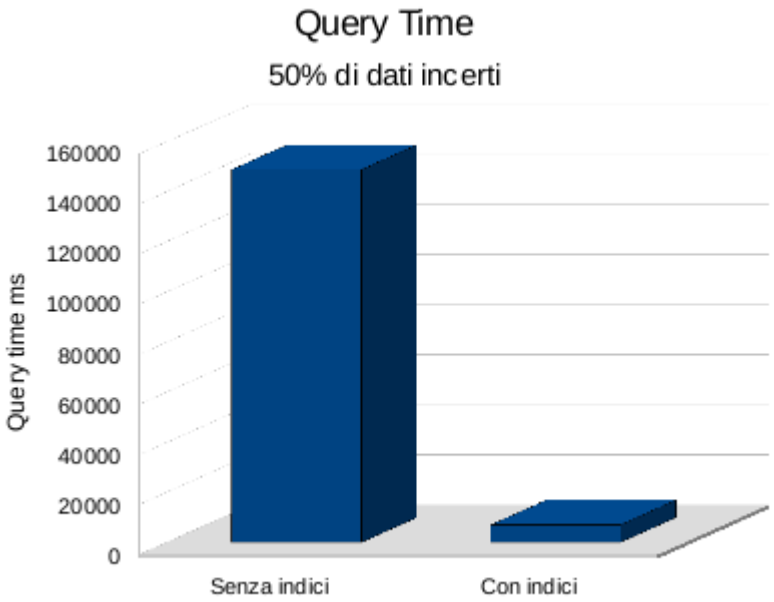

Figura 7.6: 50% di dati incerti: tempo

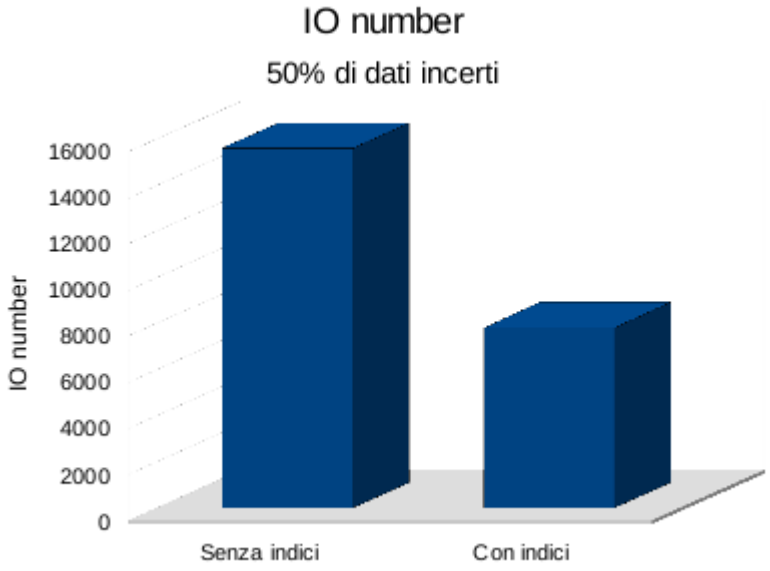

Figura 7.7: 50% di dati incerti: numero di IO

Capitolo 8

Conclusioni e Sviluppi Futuri

Lo svolgimento del suddetto progetto di tesi, come è emerso precedentemente, si è svolto in due fasi. La prima fase ha riguardato la generazione di dataset realistici con dati incerti, al fine di avere una base solida sulla quale eseguire il benchmark. La seconda fase invece ha riguardato il benchmark vero e proprio, con la formulazione delle interrogazioni da eseguire sui dataset e la raccolta dei risultati.

Proprio dalla raccolta di questi risultati è emersa con evidenza la temporanea inefficienza del DBMS Orion, dovuta sostanzialmente al suo carattere sperimentale e alla fondamentale mancanza di indici specifici per dati incerti. Utilizzando indici fittizi ma dal punto di vista pratico del tutto identici agli indici sviluppati come oggetto di ricerca l'incremento delle prestazioni è stato notevole, causando quindi un drastico calo dei tempi di esecuzione delle interrogazioni e un calo meno drastico, ma comunque importante, del numero di IO.

In particolare, com'era lecito aspettarsi, è emerso in maniera chiara il rapporto della percentuale dei dati incerti sia con il tempo di esecuzione che con il numero di IO. Infatti, queste grandezze sono tra loro direttamente proporzionali, quindi ad una percentuale di dati incerti molto alta corrispondono tempi di esecuzione ed un numero di IO maggiori, viceversa con una percentuale di dati incerti inferiore i tempi di esecuzione ed il numero di IO si abbassano con un andamento lineare.

La conclusione che si delinea quindi porta decisamente ad affermare che con un appropriato supporto a livello di indici e una qualità maggiore degli algoritmi per il calcolo della probabilità, il DBMS Orion potrebbe essere fruibile per la gestione dei dati incerti, poichè introduce un overhead mini-

mo. Inoltre la complessità della gestione dei dati incerti risulta poter essere facilmente nascosta, nel caso di utilizzo di interfacce user-friendly, rendendo quindi utilizzabile il DBMS anche da utenti non esperti, o comunque senza particolari conoscenze nel campo della statistica.

8.1 Sviluppi Futuri

Ovviamente uno tra gli sviluppi futuri più importanti riguarda l'integrazione di Orion DBMS con un sistema di indici specifico per i dati incerti. Questo comporterebbe naturalmente delle analisi, e di conseguenza delle valutazioni, più approfondite con la possibilità di fornire indicazioni ancora più precise.

Personalmente poi, credo che sarebbe molto interessante poter disporre di dataset incerti reali, o in alternativa di generatori di dataset incerti realistici, in questo modo chiunque potrebbe facilmente provare ad inserire nella propria base di dati valori incerti e valutare la convenienza della gestione diretta di questo tipo di dati, piuttosto che dover convertire faticosamente la propria base di dati.

Elenco delle figure

Bibliografia

Sunil Prabhakar. Orion 2.0: Native support for uncertain data. *In Proc. of the ACM Special Interest Group on Management of Data (SIGMOD 2008)*, 2008a.

Sunil Prabhakar. Indexing uncertain data. *Managing and mining uncertain data*, 2008b.

Sunil Prabhakar. *Database Support for Probabilistic Attributes and Tuples*. Springer, 2008c.

Sunil Prabhakar. Efficient indexing methods for probabilistic threshold queries over uncertain data. *In Very Large Databases Conference (VLDB 2004)*, 2004.

Sunil Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. *Accepted in Very Large Databases Conf. (VLDB 2005)*, 2005.

Sunil Prabhakar. Indexing uncertain categorical data. *In IEEE 23rd International Conference on Data Engineering (ICDE 2007)*, 2007.

Sunil Prabhakar. Database support for probabilistic attributes and tuples. *In proceedings of the IEEE International Conference on Data Engineering (ICDE 2008)*, 2008d.

Sunil Prabhakar. Evaluating probabilistic queries over imprecise data. *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, 2003.

Sarvjeet Singh Reynold Cheng and Sunil Prabhakar. U-dbms: A database system for managing constantly-evolving data. *Very Large Databases Conf. (VLDB 2005 Demo)*, 2005.

Jennifer Widom. Trio: A system for integrated management of data, accuracy, and lineage. *Proc. Conf. of Innovative Data Systems Research (CIDR)*, 2005.

Jennifer Widom. Foundations of uncertain-data integration. *Stanford University*, 2010.