



UNIVERSITÀ DEGLI STUDI DI BARI "ALDO MORO"

FACOLTÀ DI SCIENZE MM. FF. NN. CON SEDE A BARI
CORSO DI LAUREA IN INFORMATICA E TECNOLOGIE PER LA
PRODUZIONE DEL SOFTWARE

*OLAPBIRCH: INTEGRAZIONE DI UN
ALGORITMO DI CLUSTERING
GERARCHICO A SUPPORTO DELLE
TECNOLOGIE OLAP*

Relatore:

Chiar.mo

Prof. MICHELANGELO CECI

Presentata da:

MARIA CRISTINA TARANTINO

Correlatori:

Prof. ALFREDO CUZZOCREA

Prof. DONATO MALERBA

Anno Accademico 2009-2010

Copyright ©2011

La presente tesi di laurea, in quanto “*opera dell’ingegno*”, è coperta da *copyright* e tutelata dalla legge. Gli autori autorizzano chiunque lo desideri a fare riferimento a parte del contenuto di questo documento, purché abbia cura di inserire la debita voce bibliografica.

Chiunque usi come strumento di *typesetting* L^AT_EX e utilizzi BIBT_EX per la gestione dei riferimenti bibliografici, può farlo agevolmente, inserendo nel proprio *database* bibliografico la voce che segue.

```
@MASTERSTHESIS{TT11,
  author = {Tarantino, M. Cristina and Turturo, Fabio},
  title = {OLAPBIRCH: integrazione di un algoritmo di clustering
           gerarchico a supporto delle tecnologie OLAP},
  school = {Università degli Studi di Bari "Aldo Moro"},
  year = {2011},
  month = {Feb},
  keywords = {clustering, birch, olap, data mining,
              data warehouse}
}
```

Indice

1	Introduzione	1
1.1	<i>Data Warehousing</i>	2
1.1.1	Architetture per il <i>Data Warehousing</i>	7
1.1.1.1	Architettura ad un livello	7
1.1.1.2	Architettura a due livelli	8
1.1.1.3	Architettura a tre livelli	10
1.1.2	Gli strumenti <i>ETL</i>	12
1.1.2.1	Estrazione	12
1.1.2.2	Pulitura	14
1.1.2.3	Trasformazione	14
1.1.2.4	Caricamento	15
1.1.3	Il modello multidimensionale	15
1.1.4	Accedere al <i>Data Warehouse</i>	17
1.1.4.1	Reportistica	18
1.1.4.2	<i>OLAP</i>	20
1.1.4.3	<i>Data Mining</i>	23
1.1.5	Modelli di Rappresentazione Fisica di un <i>DW</i> : <i>ROLAP</i> e <i>MOLAP</i>	23
1.2	<i>Knowledge Discovery in Databases</i>	25
1.2.1	Il Processo di <i>KDD</i>	26
1.3	<i>Data Mining</i> : obiettivi, metodologie e tecniche	28
1.3.1	Obiettivi del <i>Data Mining</i>	29
1.3.2	Metodi per diversi problemi	31
1.4	<i>DM</i> e <i>OLAP</i> : Tecniche alternative o complementari?	37
1.5	Il <i>Clustering</i>	39

INDICE

1.5.1	Tipi di dati	41
1.5.2	Misure di distanza	42
1.5.3	Categorizzazione dei metodi di <i>clustering</i>	43
1.5.3.1	<i>K-MEANS</i>	46
1.5.3.2	<i>K-MEDOIDS</i>	48
1.5.3.3	<i>CLARANS</i>	50
1.5.3.4	<i>DBSCAN (Density-Based Spatial Clustering of Applications with Noise)</i>	51
1.5.4	Metodi gerarchici	53
1.5.4.1	<i>CURE (Clustering Using REpresentative)</i>	57
1.5.4.2	<i>BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies)</i>	59
2	L'algoritmo <i>BIRCH</i>	61
2.1	Definizioni degli Attributi Metrici	61
2.1.1	Proprietà di un <i>cluster</i>	62
2.1.2	Distanza tra due <i>cluster</i>	63
2.1.3	Misure di qualità di un <i>cluster</i>	64
2.2	<i>Clustering Feature</i> e <i>CF-Tree</i>	65
2.2.1	<i>Clustering Feature</i>	65
2.2.2	<i>CF-Tree</i>	68
2.3	Funzionamento dell'algoritmo di <i>clustering BIRCH</i>	70
2.3.1	Fase 1	73
2.3.1.1	Inserzione nel <i>CF-Tree</i>	74
2.3.1.2	Ricostruzione del <i>CF-Tree</i>	77
2.3.1.3	Riducibilità	79
2.3.1.4	Anomalie	80
2.3.1.5	Opzione di trattamento degli <i>outlier</i>	80
2.3.1.6	Opzione di ritardo dello split	81
2.3.1.7	Scelta della soglia	82
2.3.2	Fase 2	85
2.3.3	Fase 3	87
2.3.4	Fase 4	89

2.4	Utilizzo della memoria	90
2.5	<i>PBIRCH - Parallel Birch Algorithm</i>	91
2.5.1	L'algoritmo <i>PBIRCH</i>	91
2.6	Studio delle prestazioni	94
2.6.1	Analisi della Complessità	94
2.6.2	Parametri e Impostazioni di <i>default</i>	95
2.6.3	Prestazioni sulla base del carico di lavoro	96
2.6.4	Confronto tra <i>BIRCH</i> , <i>CLARANS</i> e <i>K-MEANS</i>	98
2.6.5	Sensibilità ai Parametri	105
2.6.6	Scalabilità	108
3	Implementazione e Descrizione di <i>OLAPBIRCH</i>	117
3.1	Introduzione	117
3.1.1	Motivazioni alla base della scelta del nome <i>OLAPBIRCH</i>	118
3.1.2	Tipologia dell'applicazione	118
3.1.3	Applicazioni utilizzate per lo sviluppo	119
3.2	Architettura <i>software</i>	119
3.2.1	Classi algoritmo <i>OLAPBIRCH</i>	119
3.2.1.1	Classi package “ <i>birch.cf_tree.cf_node.cf_entry</i> ”	120
3.2.1.2	Classi package “ <i>birch.cf_tree.cf_node</i> ”	124
3.2.1.3	Classi package “ <i>birch.cf_tree</i> ”	126
3.2.1.4	Classe <i>DBSCAN</i>	127
3.2.2	Classi di utilità	128
3.2.2.1	Classe <i>VectorND</i>	128
3.2.2.2	Classe <i>LoadProps</i>	128
3.2.2.3	Classe <i>Params</i>	129
3.2.3	Classi per il calcolo della misura di similarità	129
3.2.4	Classi di interazione <i>database</i>	130
3.2.4.1	Classe <i>Alias</i>	130
3.2.4.2	Classe <i>DB_Access</i>	130
3.2.4.3	Classe <i>DAOFactory</i>	132
3.2.4.4	Interfaccia <i>iDAO</i>	132
3.2.4.5	Classe <i>InterazioneDB</i>	133

INDICE

4 Modifiche apportate all'algoritmo <i>BIRCH</i>	135
4.1 Motivazioni alla base delle modifiche	135
4.2 Sensibilità alla forma sferica dei <i>cluster</i>	137
4.3 Sensibilità alle dimensioni del <i>dataset</i>	140
4.4 Integrazione con tecnologie <i>OLAP</i>	141
5 Uso di <i>BIRCH</i> in un sistema <i>OLAP</i>	145
5.1 Integrazione di <i>BIRCH</i> con le tecnologie <i>OLAP</i>	145
5.2 Il file XML <i>Mondrian</i>	147
5.3 Integrazione del sistema con il motore <i>OLAP</i>	153
6 Risultati Sperimentali	159
6.1 Introduzione	159
6.2 Granularità dei <i>cluster</i> in ciascun livello della gerarchia	161
6.2.1 Granularità sul <i>dataset Spaeth_01</i>	161
6.2.2 Granularità sul <i>dataset TPC-H</i>	168
6.3 Effetto dei <i>Rebuild</i>	184
6.4 Fattore di Ramificazione vs Qualità dei <i>cluster</i>	188
6.5 Scalabilità	192
7 Sviluppi Futuri	195
8 Conclusioni	197
A Dimostrazioni del Teorema della Rappresentatività di un <i>CF</i>	199
B Algoritmo di Inserzione in un <i>CF-Tree</i>	203
C Algoritmo di <i>Rebuilding</i> di un <i>CF-Tree</i>	205

Elenco delle figure

1.1	Architettura ad un livello per un sistema di <i>Data Warehousing</i>	8
1.2	Architettura a due livelli per un sistema di <i>Data Warehousing</i>	9
1.3	Architettura a due livelli con <i>Data Mart</i> indipendenti	10
1.4	Architettura a tre livelli per un sistema di <i>Data Warehousing</i>	11
1.5	Estrazione, trasformazione e caricamento	13
1.6	Il cubo a tre dimensioni che modella le vendite in una catena di negozi . .	16
1.7	Gerarchie di aggregazione sulle dimensioni PRODOTTO e NEGOZIO . . .	17
1.8	Presentazione di un rapporto: tabella, diagramma, torta.	19
1.9	Operatore <i>roll-up</i> sulla gerarchia temporale	20
1.10	Operatore <i>drill-down</i> sulla gerarchia temporale	21
1.11	<i>Slicing</i> di un cubo sui prodotti	21
1.12	<i>Pivoting</i> su una tabella bidimensionale	22
1.13	<i>Drill-across</i> tra due cubi	22
1.14	Architettura <i>ROLAP</i>	24
1.15	Panoramica dei passi che compongono il processo di <i>KDD</i>	28
1.16	Tipologie di <i>Data Mining</i>	31
1.17	Ricerca di Serie Temporali	33
1.18	Acquisizione giornaliera clienti	37
1.19	Acquisizione giornaliera per tipologia clienti	38
1.20	Suddivisione indesiderata di un grande <i>cluster</i> , dovuta all'eccessiva vicinanza dei <i>cluster</i> componenti	44
1.21	Suddivisione indesiderata dei <i>cluster</i> , dovuta alla loro forma allungata .	44
1.22	Funzionamento di <i>K-MEANS</i>	47
1.23	I quattro possibili casi della funzione di costo per il metodo <i>K-MEDOIDS</i>	48

ELENCO DELLE FIGURE

1.24	<i>Density-reachable</i> e <i>Density-connected</i>	52
1.25	Fenomeno del <i>chaining</i>	53
1.26	Dendrogramma Agglomerativo	54
1.27	Dendrogramma Divisivo	55
1.28	<i>Single-link</i>	55
1.29	<i>Complete-link</i>	55
1.30	<i>Average-link</i>	56
1.31	Distanza tra i centroidi	56
1.32	Contrazione dei punti rappresentativi di un <i>cluster</i> ($\alpha = 0.3$)	58
1.33	Contrazione dei punti rappresentativi di due <i>cluster</i>	58
1.34	Errato assegnamento dei punti dei <i>cluster</i> in <i>BIRCH</i>	60
2.1	Esempio di <i>CF</i> in uno spazio dei punti <i>bi</i> -dimensionale	66
2.2	Esempio di <i>CF-Tree</i>	69
2.3	Algoritmo di <i>BIRCH</i>	70
2.4	Diagramma di flusso della fase di <i>pre clustering</i> (Fase 1)	73
2.5	Effetto dello <i>Split</i> , <i>Merge</i> e del <i>Resplit</i>	76
2.6	Costruzione di un nuovo <i>CF-Tree</i>	79
2.7	Diagramma di flusso della Fase 2	86
2.8	Diagramma di Flusso della Fase 3	88
2.9	Diagramma di Flusso della Fase 4	89
2.10	Schema dell'algoritmo <i>PBIRCH</i>	93
2.11	<i>Cluster</i> Previsti per il <i>DS1</i>	101
2.12	<i>Cluster</i> Previsti per il <i>DS2</i>	101
2.13	<i>Cluster</i> Previsti per il <i>DS3</i>	101
2.14	<i>Cluster</i> <i>BIRCH</i> per il <i>DS1</i>	102
2.15	<i>Cluster</i> <i>CLARANS</i> per il <i>DS1</i>	102
2.16	<i>Cluster</i> <i>K-MEANS</i> per il <i>DS1</i>	102
2.17	<i>Cluster</i> <i>BIRCH</i> per il <i>DS2</i>	103
2.18	<i>Cluster</i> <i>CLARANS</i> per il <i>DS2</i>	103
2.19	<i>Cluster</i> <i>K-MEANS</i> per il <i>DS2</i>	103
2.20	<i>Cluster</i> <i>BIRCH</i> per il <i>DS3</i>	104
2.21	<i>Cluster</i> <i>CLARANS</i> per il <i>DS3</i>	104

2.22	<i>Cluster K-MEANS</i> per il <i>DS3</i>	104
2.23	Scalabilità del tempo rispetto all'Incremento Del Numero di Punti per <i>Cluster</i>	109
2.24	Scalabilità del tempo rispetto all'Incremento Del Numero di <i>Cluster</i>	112
2.25	Scalabilità del tempo rispetto all'Incremento Del Numero delle Dimensioni	115
3.1	Diagramma <i>UML package</i> “ <i>birch.cf_tree.cf_node.cf_entry</i> ”	121
3.2	Diagramma <i>UML package</i> “ <i>birch.cf_tree.cf_node</i> ”	123
3.3	Diagramma <i>UML package</i> “ <i>birch.util.distance</i> ”	129
3.4	Diagramma <i>UML package</i> “ <i>birch.util.database</i> ”	131
4.1	<i>Sotto-Cluster</i> Prodotti dalla Fase 1	137
4.2	<i>Cluster</i> prodotti dalla Fase 3 con algoritmo <i>centroid-based</i>	138
4.3	Centroidi dei <i>cluster</i> prodotti dalla Fase 1	139
4.4	<i>Cluster</i> prodotti dal <i>DBSCAN</i> a partire dai centroidi dei <i>cluster</i> prodotti dalla Fase 1	139
4.5	Rappresentazione del <i>CF</i> in <i>OLAPBIRCH</i>	140
4.6	Panoramica delle nuove funzioni previste dal nostro sistema	143
5.1	Panoramica delle nuove funzioni previste dal nostro sistema	147
5.2	Schema TPC-H	149
6.1	<i>Dataset Spaeth_01</i>	161
6.2	<i>Cluster</i> risultanti nel Livello 2 su <i>Spaeth_01</i>	162
6.3	<i>Cluster</i> risultanti nel Livello 3 su <i>Spaeth_01</i>	163
6.4	<i>Cluster</i> risultanti nel Livello 4 su <i>Spaeth_01</i>	164
6.5	<i>Cluster</i> risultanti nel Livello 5 su <i>Spaeth_01</i>	165
6.6	<i>Cluster</i> risultanti nel Livello 6 su <i>Spaeth_01</i>	166
6.7	<i>Cluster</i> risultanti nel Livello 7 su <i>Spaeth_01</i>	167
6.8	Distribuzione della media aritmetica rispetto a <i>TOTALPRICE</i> sui 3 <i>cluster</i> individuati dal <i>DBSCAN</i> al Livello 7 della gerarchia	173
6.9	Distribuzione della media aritmetica rispetto a <i>QUANTITY</i> sui 3 <i>cluster</i> individuati dal <i>DBSCAN</i> al Livello 7 della gerarchia	174
6.10	Distribuzione della media aritmetica rispetto a <i>LINENUMBER</i> sui 3 <i>cluster</i> individuati dal <i>DBSCAN</i> al Livello 7 della gerarchia	174

ELENCO DELLE FIGURE

6.11 Distribuzione della media aritmetica rispetto a ACCTBAL sui 3 <i>cluster</i> individuati dal DBSCAN al Livello 7 della gerarchia	175
6.12 Distribuzione della proprietà REGION sul <i>cluster</i> 1 individuato dal DBSCAN al Livello 7 della gerarchia	175
6.13 Distribuzione della proprietà REGION sul <i>cluster</i> 2 individuato dal DBSCAN al Livello 7 della gerarchia	176
6.14 Distribuzione della proprietà REGION sul <i>cluster</i> 3 individuato dal DBSCAN al Livello 7 della gerarchia	176
6.15 Distribuzione della media aritmetica rispetto a TOTALPRICE sui 3 <i>cluster</i> individuati dal DBSCAN al Livello 6 della gerarchia	177
6.16 Distribuzione della media aritmetica rispetto a QUANTITY sui 3 <i>cluster</i> individuati dal DBSCAN al Livello 6 della gerarchia	177
6.17 Distribuzione della media aritmetica rispetto a LINENUMBER sui 3 <i>cluster</i> individuati dal DBSCAN al Livello 6 della gerarchia	178
6.18 Distribuzione della media aritmetica rispetto a ACCTBAL sui 3 <i>cluster</i> individuati dal DBSCAN al Livello 6 della gerarchia	178
6.19 Distribuzione della proprietà REGION sul <i>cluster</i> 1 individuato dal DBSCAN al Livello 6 della gerarchia	179
6.20 Distribuzione della proprietà REGION sul <i>cluster</i> 2 individuato dal DBSCAN al Livello 6 della gerarchia	179
6.21 Distribuzione della proprietà REGION sul <i>cluster Rumore</i> individuato dal DBSCAN al Livello 6 della gerarchia	180
6.22 Distribuzione della media aritmetica rispetto a TOTALPRICE sui 2 <i>cluster</i> individuati dal DBSCAN al Livello 5 della gerarchia	180
6.23 Distribuzione della media aritmetica rispetto a QUANTITY sui 2 <i>cluster</i> individuati dal DBSCAN al Livello 5 della gerarchia	181
6.24 Distribuzione della media aritmetica rispetto a LINENUMBER sui 2 <i>cluster</i> individuati dal DBSCAN al Livello 5 della gerarchia	181
6.25 Distribuzione della media aritmetica rispetto a ACCTBAL sui 2 <i>cluster</i> individuati dal DBSCAN al Livello 5 della gerarchia	182
6.26 Distribuzione della proprietà REGION sul <i>cluster</i> 1 individuato dal DBSCAN al Livello 5 della gerarchia	182

6.27	Distribuzione della proprietà REGION sul <i>cluster</i> 2 individuato dal <i>DBSCAN</i> al Livello 5 della gerarchia	183
6.28	<i>Dataset Spaeth_05</i>	184
6.29	Effetto <i>rebuild</i>	185
6.30	<i>Cluster</i> individuati da <i>DBSCAN</i> su <i>Spaeth_05</i>	186
6.31	<i>Cluster</i> individuati da <i>DBSCAN</i> su <i>Spaeth_05</i>	187
6.32	<i>Dataset Spaeth_06</i>	188
6.33	<i>Cluster</i> individuati su <i>Spaeth_06</i> con $B = L = 5$	189
6.34	<i>Cluster</i> individuati su <i>Spaeth_06</i> con $B = L = 3$	190
6.35	<i>Cluster</i> individuati su <i>Spaeth_06</i> con $B = L = 2$	191

Elenco delle tabelle

1.1	Differenze tra <i>database</i> operazionali e <i>Data Warehouse</i>	6
2.1	<i>BIRCH</i> : Parametri e relativi valori di <i>default</i>	96
2.2	<i>Dataset</i> usati per testare il carico di lavoro	97
2.3	<i>Performance</i> di <i>BIRCH</i> sulla base del carico di lavoro rispetto a Tempo, \bar{D} , Ordine di <i>Input</i> e Numero di Scansioni dei Dati	100
2.4	<i>Performance</i> di CLARANS sulla base del carico di lavoro rispetto a Tempo, \bar{D} , Ordine di <i>Input</i> e Numero di Scansioni dei Dati	100
2.5	<i>Performance</i> di <i>K-MEANS</i> sulla base del carico di lavoro rispetto a Tempo, \bar{D} , Ordine di <i>Input</i> e Numero di Scansioni dei Dati	100
2.6	Sensibilità al valore di Soglia Iniziale	105
2.7	Sensibilità all'opzione di gestione degli <i>outlier</i>	106
2.8	Sensibilità all'algoritmo di <i>clustering</i> globale per la Fase 3	107
2.9	Qualità della Scalabilità rispetto all'Incremento Del Numero di Punti per <i>Cluster</i>	110
2.10	Qualità della Scalabilità rispetto all'Incremento Del Numero di <i>Cluster</i> .	113
2.11	Qualità della Scalabilità rispetto all'Incremento Del Numero delle Dimensioni	116
6.1	Caratteristiche del calcolatore su cui sono state effettuate le prove sperimentali	160
6.2	Distribuzione della media aritmetica sui 3 <i>cluster</i> individuati dal <i>DBSCAN</i> al Livello 7 della gerarchia	173
6.3	Distribuzione della media aritmetica sui 3 <i>cluster</i> individuati dal <i>DBSCAN</i> al Livello 6 della gerarchia	177

ELENCO DELLE TABELLE

6.4	Distribuzione della media aritmetica sui 3 <i>cluster</i> individuati dal <i>DBSCAN</i> al Livello 5 della gerarchia	180
6.5	Caratteristiche del calcolatore su cui sono state effettuate le prove di scalabilità	192
6.6	Scalabilità del sistema	193

Elenco degli algoritmi

1	K-Means	47
2	PAM	49
3	Insert_Into_CF-Tree	203
4	Re-build_CF-Tree	205

Premessa

La funzione svolta dalle basi di dati in ambito aziendale è stata, fino a qualche anno fa, quella di memorizzare dati operazionali, cioè dati generati da operazioni svolte all'interno dei processi gestionali. D'altronde, per ogni azienda è fondamentale poter disporre in maniera rapida e completa delle informazioni necessarie al processo decisionale. Grazie alla connettività globale fornita dalla rete internet e allo sviluppo crescente del web, si è assistito ad un progressivo aumento della complessità del mondo economico e alla concretizzazione del processo di globalizzazione; di conseguenza, la quantità di dati che un'azienda media o grande deve mantenere è enorme, ed ancora più complesso risulta prendere decisioni in base a questo ampio insieme di dati. Il ruolo delle basi di dati ha così cominciato a cambiare, a partire dagli anni '80, con la nascita dei sistemi di supporto alle decisioni (Decision Support System), che permettono di estrapolare informazioni anche in presenza di grandi quantità di dati. Negli ultimi anni, in particolare, il mondo accademico e industriale ha focalizzato l'attenzione sui sistemi di Data Warehousing. Una classe di tecnologie integrate in tali sistemi è OLAP, che comprende un insieme di funzioni che consentono di rendere "dinamica" la visualizzazione delle informazioni, allo scopo di desumere conoscenza utile nei processi decisionali. Pur essendo strumenti molto potenti, i sistemi OLAP, specie se si poggiano su database relazionali (ROLAP), non sono esenti da problemi di efficienza e spreco di spazio: essi prevedono, infatti, che ogni operazione componente la richiesta dell'utente, provochi la generazione e l'esecuzione di query intermedie estremamente complesse, che richiedono molte risorse di elaborazione. Per superare questo problema è possibile sfruttare tecniche di data mining (principi e tecniche utili per "estrarre" conoscenza dai dati) per migliorare i sistemi OLAP. Tra tali metodologie, quelle rivelatesi più efficienti ed efficaci per il supporto alle decisioni sono quelle basate sul clustering gerarchico. Argomento della presente tesi è la progettazione di un algoritmo di clustering gerarchico, ideale per grandi moli di dati, in grado di migliorare notevolmente le prestazioni di un sistema OLAP.

Capitolo 1

Introduzione

La scienza è fatta di dati come una casa di pietre.

*Ma un ammasso di dati non è scienza
più di quanto un mucchio di pietre sia una casa.*

JULES HENRI POINCARÉ

VIAMI da tempo nella Società dell'Informazione¹, una società caratterizzata da elevato dinamismo e rapidi cambiamenti determinati anche dalla centralità assunta dall'informazione. Il suo ruolo di risorsa strategica ne fa uno strumento potente in grado di condizionare imprese e processi aziendali, ma anche sviluppo economico, crescita, ricchezza sociale e culturale. Informazione e conoscenza sono diventate materie prime strategiche, prodotti fondamentali di cui singole persone, organizzazioni, istituzioni e imprese non possono più fare a meno. Per stare al passo e competere globalmente, le imprese devono sempre più e sempre meglio fare affidamento su informazioni aggiornate, rinnovabili e disponibili in tempo reale, per poter coordinare risorse distribuite geograficamente, favorire l'interazione e la collaborazione tra membri di un *team* o agenti di un distretto o di una filiera produttiva e per la creazione di nuovo valore. Queste informazioni risiedono spesso all'interno di sistemi di *Data Warehouse*, *Data Mart* e *ODS (Operational Data Store)*, veri e propri magazzini di dati, che permettono alle aziende di prendere decisioni critiche in

¹Termine che connota la società odierna, caratterizzata da un'economia basata largamente sulla produzione di servizi, specialmente quelli in cui si manipolano informazioni, e sul valore economico della conoscenza come risorsa strategica.

tempo reale disponendo di tutti i dati necessari nelle varie fasi dei processi decisionali e rispettando i mandati relativi alla *compliance* e alle normative vigenti. I *Data Warehouse*, nelle varie configurazioni e forme che si sono affermate negli anni, sono strumenti potenti per l'integrazione e la trasformazione di dati distribuiti, complessi ed eterogenei, in informazioni e conoscenze utili a sostenere il *reporting*, i processi decisionali e le esigenze di analisi delle *performance* aziendali.

In questo capitolo verranno introdotte le definizioni di base del settore, le differenti architetture funzionali adottabili, alcuni aspetti critici per la comprensione dell'intero processo di *Data Warehousing*: l'alimentazione del *Data Warehouse* dalle sorgenti dati, il modello multidimensionale su cui il *Data Warehouse* si fonda e le principali modalità di fruizione del suo contenuto informativo da parte degli utenti. Quest'ultimo argomento sarà ulteriormente approfondito, al fine di condurre il lettore alla comprensione delle motivazioni che hanno cagionato la presente tesi.

1.1 *Data Warehousing*

“Un Data Warehouse è un contenitore di dati storici integrati e consistenti, dotato di strumenti che consentono alla direzione aziendale di estrarre facilmente informazioni attendibili di supporto al processo decisionale.”
(Golfanelli e Rizzi in [MG06]).

Quando nella letteratura si tratta di *Data Warehouse* (*DW*) o di altre strutture informative che hanno come funzione il supporto all'analisi dei dati di *business* o la realizzazione dei *DDS*², converrebbe ricordare una delle prime strutture informative create dall'uomo: le tavole della città di Ebla, risalente al XXIV secolo a.C. Queste famose tavole, scoperte negli anni '70, comprendevano rendiconti mensili “in entrata” e “in uscita”, ordinanze reali, trattati politici, verbali processuali, rendiconti amministrativi dell'attività agricola e pastorale. Praticamente, le tavole di Ebla costituivano un insieme di dati strutturati che rappresentavano e sostenevano le attività gestionali e decisionali di tipo economico, politico e logistico dell'antichissima città-stato. Erano un vero e proprio *DW*, forse il primo della storia, nucleo portante di un protosistema informativo decisionale.

²**D***ecision S***upport S***ystem* (Sistemi di Supporto Decisionale), termine con cui si intende l'insieme delle tecniche e degli strumenti informatici atti a estrapolare *informazioni* da un insieme di *dati* memorizzati su supporti elettronici.

Il concetto di *Data Warehouse* risale all'inizio degli anni '90 e si è evoluto di pari passo con l'affermarsi della Società dell'Informazione, di cui è divenuto uno dei capisaldi. Non si discosta poi molto, in prima approssimazione, dalla funzione svolta dalle tavole di Ebla: è una struttura informativa, stabilita per meglio organizzare il patrimonio dei dati disponibili all'interno di un'organizzazione, allo scopo di fornire una visione integrata delle attività di *business*. Di norma, la decisione di creare un *DW* scaturisce dalla volontà di utilizzare i dati aziendali per comprendere gli andamenti del passato e per fare previsioni sul futuro, migliorando di conseguenza la gestione del *business* corrente.

L'architettura di un *DW*, com'è attualmente concepita, è il risultato finale di un processo evolutivo che ha interessato l'organizzazione stessa dei sistemi informativi. Nel corso degli anni le organizzazioni si sono dotate di un insieme di dati e di applicazioni sempre più vasti ed articolati, risorse di enorme valore, la cui crescita disordinata ha reso però sempre più difficile l'integrazione dei vari componenti per comporre una visione globale dell'azienda.

I *DW* sono nati per contestualizzare e capire il significato dei dati che costituiscono la materia prima con la quale, attraverso opportune elaborazioni, si producono informazioni: una maggiore conoscenza dell'ambiente in cui si opera offre la possibilità di controllarlo, modificarlo e dirigerlo. Il concetto di "dato" stesso si è evoluto notevolmente nel corso dei cinquant'anni della storia dell'informatica, passando da un'accezione esclusivamente numerica a identificare entità rappresentabili con strutture alfanumeriche, organizzate in formati *standard*, costituiti generalmente da liste o da tabelle (dati anagrafici, ordini, ecc.). Ulteriori estensioni si sono rese necessarie quando, grazie ai progressi ed alle economie messe dall'evoluzione tecnologica di base, è stato possibile elaborare documenti a testo libero, forme figurative (disegni, immagini o grafici), filmati e suoni. La disponibilità di strumenti *hardware* e *software* sempre più accessibili, economici e potenti ha permesso alle imprese di raccogliere e memorizzare quantità sempre maggiori di dati dettagliati, anche se la capacità di analizzare e comprendere questo patrimonio, trasformando i dati in informazioni e queste in vantaggio competitivo, è ancora generalmente in ritardo rispetto alle tecniche sviluppate per costruirlo. L'utilizzo di *DSS*, permette di condurre analisi di mercato sofisticate, consente di ottenere una migliore conoscenza dei clienti, dei loro bisogni e delle loro attese, e quindi di migliorare l'efficienza dei processi aziendali e la profittabilità dell'impresa. Per cogliere questi obiettivi è necessario attivare un processo di sintesi e di valutazione dei dati che porti alla costruzione di un *DW* che, in quanto "struttura informativa", deve essere prodotto da una serie di processi di allestimento (*ETL*):

CAPITOLO 1. Introduzione

Extraction, Transformation, Loading), mentre, in quanto “*supporto decisionale*”, richiede un sostegno affidabile e continuo. Il processo in oggetto è detto *Data Warehousing*.

Il processo di *Data Warehousing* si occupa di estrarre dal sistema informativo aziendale i dati di interesse, di trasformarli, di inserirli e ripulirli da errori e inconsistenze, di inserirli nel *Data Warehouse* per poi renderli disponibili al fine di rispondere alle complesse interrogazioni di analisi e previsione formulate dagli utenti finali.

I sistemi che implementano questo processo, rappresentano la categoria di *DSS* su cui, negli ultimi anni, si è maggiormente focalizzata l’attenzione sia nel modo accademico che in quello industriale. Tali sistemi si possono definire come

“una collezione di metodi, tecnologie e strumenti di ausilio al knowledge worker per condurre analisi dei dati finalizzate all’attuazione di processi decisionali e al miglioramento del patrimonio informativo”. (Golfanelli, Rizzi in [MG06]).

Fattori distintivi della tipologia di sistemi in questione sono:

- *accessibilità* a utenti con conoscenze limitate di informatica e strutture dati;
- *integrazione* dei dati sulla base di un modello *standard* dell’impresa;
- *flessibilità di interrogazione* per trarre il massimo vantaggio dal patrimonio informativo esistente;
- *sintesi* per permettere analisi mirate ed efficaci;
- *rappresentazione multidimensionale* per offrire all’utente una visione intuitiva ed efficacemente manipolabile delle informazioni;
- *correttezza e completezza* dei dati integrati.

Un *Data Warehouse* è, dunque, la struttura informativa presente nei sistemi di *Data Warehousing*. Esso presenta le seguenti caratteristiche:

- **Orientata al soggetto:** nel *DW* i dati sono organizzati per soggetti rilevanti (prodotti, clienti, fornitori, periodi di tempo) al fine di offrire tutte le informazioni inerenti una specifica area.

- **Integrata e consistente:** il *DW* deve essere in grado di integrarsi perfettamente con la moltitudine di *standard* utilizzati nelle diverse applicazioni. I dati devono essere ricodificati, per risultare omogenei dal punto di vista semantico, e devono utilizzare le stesse unità di misura.
- **Variabile nel tempo:** a differenza dei dati operazionali, quelli di un *DW* hanno un orizzonte temporale molto ampio (anche 5-10 anni), risultando riutilizzabili in diversi istanti temporali.
- **Non volatile:** i dati operazionali sono aggiornati in modo continuo; nel *DW* i dati sono caricati inizialmente con processi integrali e successivamente aggiornati con caricamenti parziali; i dati, una volta caricati, non vengono modificati e mantengono la loro integrità nel tempo.

Ricordiamo che il *DW* deve essere orientato a produrre informazioni rilevanti per le decisioni del *management*. La vera differenza fra il *DW* e qualsiasi database aziendale è quella di configurarsi come un contenitore in cui sono collezionati dati utili per effettuare operazioni di *business intelligence*. Ulteriori e fondamentali differenze tra *database* operazionali e *DW* sono legate alle tipologie di interrogazioni. Per i primi, le interrogazioni eseguono transazioni che in genere leggono e scrivono un ridotto numero di *record* da diverse tabelle legate da semplici relazioni: per esempio, si ricercano i dati di un cliente per inserire un suo nuovo ordine. Questo tipo di elaborazione viene comunemente detto *On-Line Transactional Processing (OLTP)*. Al contrario, il tipo di elaborazione per cui nascono i *DW* viene detto *On-Line Analytical Processing (OLAP)*, ed è caratterizzato da un'analisi dinamica e multidimensionale che richiede la scansione di un'enorme quantità di *record* per calcolare un insieme di dati numerici di sintesi che quantificano le prestazioni dell'azienda. Le peculiari caratteristiche delle interrogazioni *OLAP* fanno sì che i dati nel *DW* siano normalmente rappresentati in forma multidimensionale. L'idea di base è quella di vedere i dati come punti in uno spazio le cui dimensioni corrispondono ad altrettante possibili dimensioni di analisi; ciascun punto, rappresentativo di un evento, viene descritto tramite un insieme di misure di interesse per il processo decisionale. Le principali differenze tra *database* operazionali e *DW* sono illustrate nella Tabella § 1.1 nella pagina seguente.

Tabella 1.1: Differenze tra *database* operazionali e *Data Warehouse*

	<i>Database</i> operazionali	<i>Data Warehouse</i>
utenti	migliaia	centinaia
carico di lavoro	transazione predefinita	interrogazioni di analisi <i>ad hoc</i>
accesso	a centinaia di <i>record</i> , in lettura e scrittura	a milioni di <i>record</i> per lo più in lettura
scopo	dipende dall'applicazione	supporto alle decisioni
dati	elementari, sia numerici sia alfanumerici	di sintesi, prevalentemente numerici
integrazione dei dati	per applicazioni	per soggetto
qualità	in termini di integrità	in termini di consistenza
copertura temporale	solo dati correnti	dati correnti e storici
aggiornamenti	continui	periodici
modello	normalizzato	de-normalizzato, multidimensionale
ottimizzazione	per accessi <i>OLTP</i> su una frazione del database	per accessi <i>OLAP</i> su gran parte del database
sviluppo	a cascata	iterativo

1.1.1 Architetture per il *Data Warehousing*

Le architetture proposte in letteratura per il *DW* sono ad uno, a due e a tre livelli. Qualsiasi architettura alla base di un *DW* deve necessariamente soddisfare le seguenti caratteristiche:

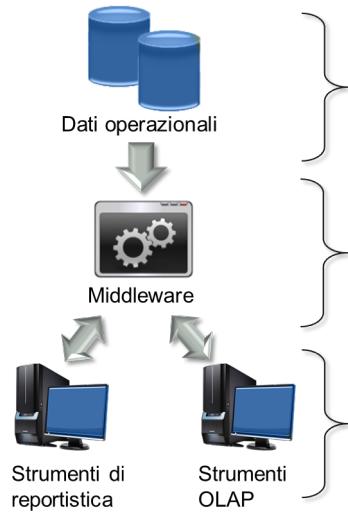
- *separazione*: l'elaborazione analitica e quella transazionale sono mantenute il più possibile separate.
- *scalabilità*: l'architettura *hardware* e *software* deve poter essere facilmente ridimensionata a fronte della crescita nel tempo dei volumi di dati e del numero di utenti.
- *estendibilità*: deve essere possibile accogliere nuove applicazioni e tecnologie senza riprogettare integralmente il sistema.
- *sicurezza*: a causa della natura strategica dei dati memorizzati, il controllo sugli accessi è essenziale.
- *amministrabilità*: la complessità dell'attività di amministrazione non deve risultare eccessiva.

Di seguito saranno analizzate le tre tipologie di architetture usate per la progettazione di un qualsiasi DW.

1.1.1.1 Architettura ad un livello

L'architettura ad un livello prevede che il *DW* sia un database virtuale, ovvero costituito da viste che saranno costruite tramite uno strato d'elaborazione intermedio, chiamato *middleware*. Tale tipo d'architettura evita il problema della ridondanza dei dati, ma comporta che le transazioni analitiche e quelle transazionali siano inoltrate sulla stessa base di dati. Per eseguire l'elaborazione dei dati, dal punto di vista analitico, il *middleware* effettua interrogazioni sui dati operazionali. Questo va contro l'idea di base di un *DW*, che prevede di separare le operazioni *OLAP* da quelle *OLTP*. Tale tipo d'architettura è utilizzato nel caso in cui non si hanno particolari esigenze d'analisi e rappresenta la sua formulazione più semplice.

Figura 1.1: Architettura ad un livello per un sistema di *Data Warehousing*



1.1.1.2 Architettura a due livelli

L'architettura a due livelli è così definita proprio per evidenziare che esistono due insiemi di dati: il **livello sorgente**, ovvero l'insieme dei dati operazionali dell'azienda o esterni all'azienda e il **livello del Warehouse**, ovvero il *DW*.

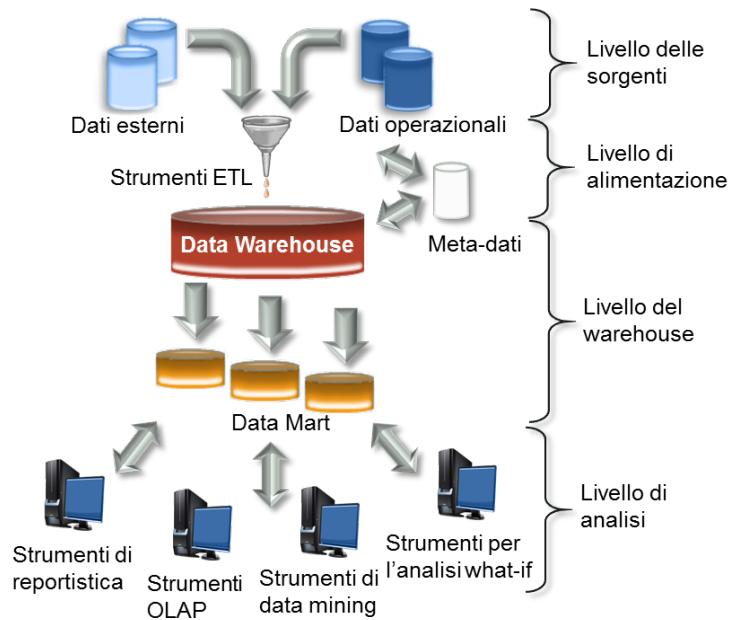
In questo caso, a differenza dell'architettura ad un livello, il *Data Warehouse* è fisicizzato. La ridondanza di dati permette di separare le operazioni d'analisi da quelle transazionali (requisito fondamentale del *DW*).

Come si può osservare dalla Figura § 1.2 nella pagina successiva, oltre ai due livelli appena descritti, esiste anche il livello d'alimentazione e quello d'analisi.

Il **livello d'alimentazione**, più propriamente detto *ETL (Extraction, Transformation, Loading)* prevede l'estrazione e la pulizia dei dati dal livello sorgente, la trasformazione ed il caricamento all'interno del *DW*.

Il **livello d'analisi** include l'insieme degli strumenti che permettono la consultazione efficiente e flessibile dei dati integrati a fini di stesura di *report*, di analisi e di simulazione. Le diverse tipologie sono discusse nella Sezione § 1.1.4.1 a pagina 18.

Al livello del Warehouse, oltre ad esserci il *DW* finora definito, sono presenti anche i cosiddetti *Data Mart*. Il cilindro etichettato con il nome “*Data Warehouse*” nella Figura

Figura 1.2: Architettura a due livelli per un sistema di *Data Warehousing*

§ 1.2 nella pagina successiva è di chiamato *DW primario* o *DW aziendale* mentre i *Data Mart* sono dei *DW locali*.

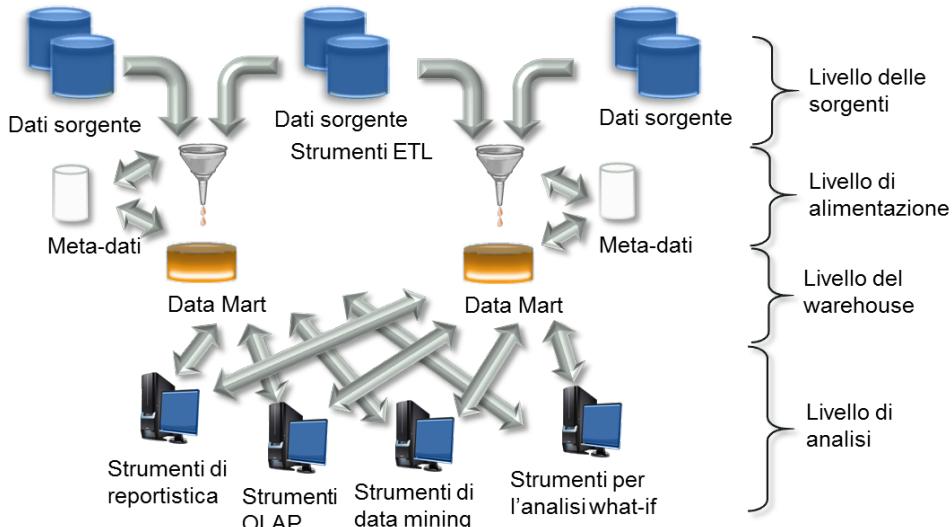
Un *Data Mart* è definito in letteratura, nel seguente modo:

“Per Data Mart si intende un sottoinsieme o un’aggregazione dei dati presenti nel Data Warehouse primario, che contiene l’insieme delle informazioni rilevanti per una particolare area di business, una particolare divisione dell’azienda, una particolare categoria di soggetti.” (Golfanelli, Rizzi in [MG06]).

I *Data Mart* alimentati dal *DW* primario sono detti *dipendenti*; per i sistemi collocati all’interno di realtà aziendali medio-grandi, essi assumono un ruolo importante:

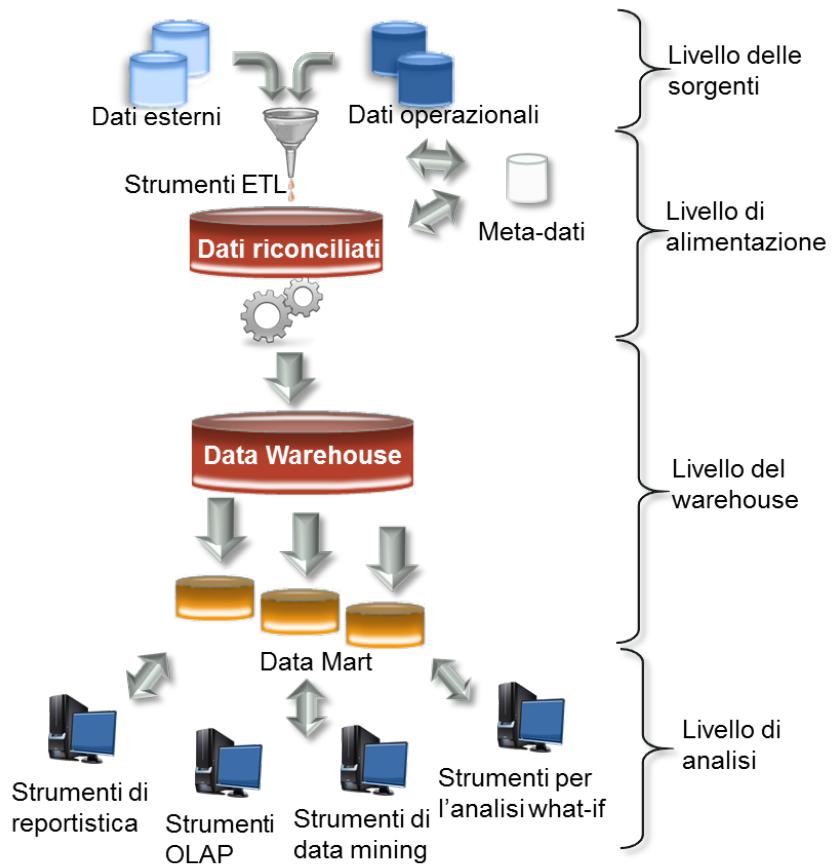
- sono blocchi costruttivi durante la realizzazione incrementale del *DW*;
- delineano i contorni delle informazioni necessarie ad un particolare tipologia di utente;
- permettono di raggiungere prestazioni migliori visto che risultano di dimensione inferiore al *DW*.

Figura 1.3: Architettura a due livelli con *Data Mart* indipendenti



In alcuni contesti, si preferisce adottare la struttura illustrata in Figura § 1.3 nella pagina seguente dove i *Data Mart* sono alimentati direttamente dalle sorgenti e vengono detti *indipendenti*. L'assenza di un *DW* primario snellisce le fasi progettuali, ma determina uno schema complesso di accessi ai dati e può causare inconsistenze tra i *Data Mart*. A volte si preferisce creare comunque un *DW* centrale alimentato dai *Data Mart*, comportando una semplificazione del *pattern* degli accessi. Le principali motivazioni a sostegno dell'architettura a due livelli, sono così riassumibili:

- nel livello del *Warehouse* è disponibile informazione di buona qualità anche quando è temporaneamente precluso l'accesso alle sorgenti;
- l'interrogazione analitica effettuata sul *DW* non interferisce con la gestione delle transazioni a livello operazionale;
- l'organizzazione logica del *DW* è basata sul modello multidimensionale;
- a livello del *Warehouse* è possibile impiegare tecniche specifiche per ottimizzare le prestazioni per applicazioni di analisi e reportistica.

Figura 1.4: Architettura a tre livelli per un sistema di *Data Warehousing*

1.1.1.3 Architettura a tre livelli

Il terzo livello introdotto in questa architettura è il *livello dei dati riconciliati*, che materializza i dati operazionali ottenuti a valle del processo di integrazione e ripulitura dei dati sorgente: quindi dati integrati, consistenti, corretti, volatili, correnti e dettagliati. Come mostrato in Figura § 1.4, il DW viene alimentato non più direttamente dalle sorgenti, ma piuttosto dai dati riconciliati.

Il livello dei dati riconciliati crea un modello di dati comune e di riferimento per l'intera azienda, introducendo una separazione netta tra le problematiche legate all'estrazione e integrazione dei dati dalle sorgenti e quelle inerenti l'alimentazione del DW. Di contro i dati riconciliati introducono un'ulteriore ridondanza rispetto ai dati operazionali sorgente e risultano inadeguati nel caso di applicazioni di piccola dimensione.

1.1.2 Gli strumenti *ETL*

L'acronimo *ETL* sta per *Extraction*, *Transformation*, *Loading*, letteralmente estrazione, trasformazione, caricamento, e rappresenta la componente di un sistema di *Data Warehousing* che si occupa di tradurre la transazione in informazione: è compito della componente *ETL* alimentare il *DW*.

Questi strumenti sono fondamentali e si occupano essenzialmente di

- estrarre i dati da sistemi informativi esterni o interni ad un'organizzazione (per questo eterogenei tra loro);
- effettuare la pulizia dei dati;
- uniformarli ad un determinato formato;
- identificare i dati mancanti;
- eliminare i duplicati;
- caricare e aggiornare i dati tramite un processo di *refresh* periodico.

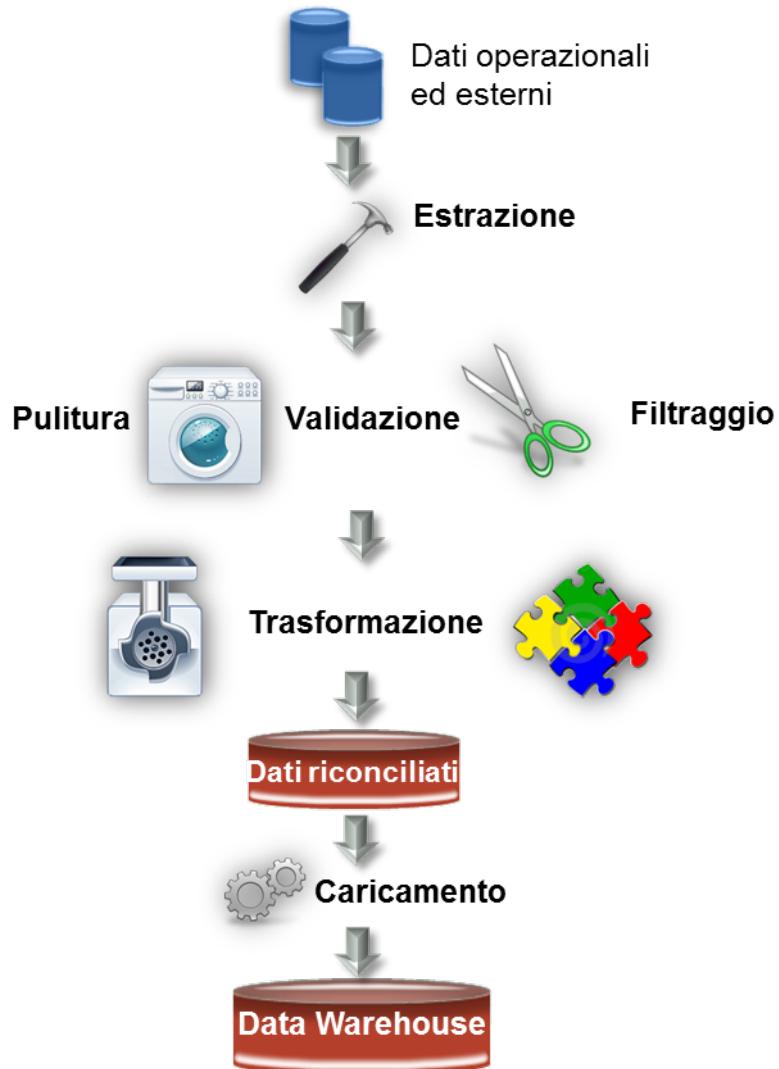
Come mostrato in Figura § 1.5 a fronte, la componente in oggetto consiste di quattro *step* distinti, detti rispettivamente *estrazione* (*extraction* o *capture*), *pulitura* (*cleaning* o *cleaning* o *scrubbing*), *trasformazione* (*trasformation*) e *caricamento* (*loading*). Di seguito verranno brevemente descritte le varie fasi appena citate.

1.1.2.1 Estrazione

La fase di estrazione si occupa di estrarre i dati rilevanti dalle sorgenti. Si possono avere due tipologie di estrazione:

- *estrazione statica*, effettuata quando il *DW* deve essere popolato per la prima volta.
- *estrazione incrementale*, usata per l'aggiornamento periodico del *DW* ed è in genere basata su *log*; se ai dati operazionali è associata una marcatura temporale (*time stamp*) che ne identifica l'istante di modifica o inserimento, essa è utilizzabile per semplificare l'estrazione.

Figura 1.5: Estrazione, trasformazione e caricamento



1.1.2.2 Pulitura

La fase di pulitura rappresenta un passaggio critico nel processo di *Data Warehousing*, poiché si incarica di migliorare la qualità dei dati (normalmente piuttosto scarsa nelle sorgenti). Tra gli errori e le inconsistenze tipiche che rendono “sporchi” i dati, si segnala

- dati duplicati;
- inconsistenza tra due valori logicamente associati (per esempio tra indirizzo e CAP);
- dati mancanti;
- uso non previsto di un campo (per esempio un campo codice fiscale utilizzato impropriamente per memorizzare il numero di telefono);
- valori impossibili o errati (ad esempio ’30/12/1999’);
- valori inconsistenti per la stessa entità dovuti a differenti convenzioni e abbreviazioni;
- valori inconsistenti per la stessa entità dovuti a errori di battitura.

Le principali funzionalità di pulitura dei dati riscontrabili negli strumenti *ETL* sono la *correzione* e l'*omogeneizzazione*, che utilizza dizionari appositi per correggere errori di scrittura e riconoscere sinonimi, e la *pulitura basata su regole*, che applica regole proprie del dominio applicativo per stabilire le corrette corrispondenze tra i valori.

1.1.2.3 Trasformazione

La fase di trasformazione si occupa di convertire i dati dal formato operazionale sorgente a quello del *DW*. Alcune situazioni che devono essere corrette durante questa fase sono:

- presenza di testi liberi che nascondono informazioni importanti;
- utilizzo di formati differenti per lo stesso dato: ad esempio, una data può essere memorizzata come stringa o tramite tre interi.

Tra le funzionalità di trasformazione è bene sottolineare quella di

- *conversione* e di *normalizzazione*, che operano sia a livello di formato di memorizzazione sia a livello di unità di misura, al fine di uniformare i dati;
- *matching* che stabilisce corrispondenze tra campi equivalenti in sorgenti diverse;
- *selezione* che riduce il numero di campi e di *record* rispetto alle sorgenti.

Per l'alimentazione del *DW* si hanno invece la *de-normalizzazione*, al posto della normalizzazione, e si introduce l'*aggregazione* che realizza le opportune sintesi dei dati.

1.1.2.4 Caricamento

L'ultimo *step* prevista dalla componente *ETL* è quella di caricamento dei dati nel *DW*, che può avvenire secondo due modalità:

Refresh: I dati vengono riscritti integralmente, sostituendo quelli precedenti.

Update: Vengono aggiunti nel *DW* i soli cambiamenti occorsi nei dati sorgenti, tipicamente senza alterare o distruggere i dati esistenti.

1.1.3 Il modello multidimensionale

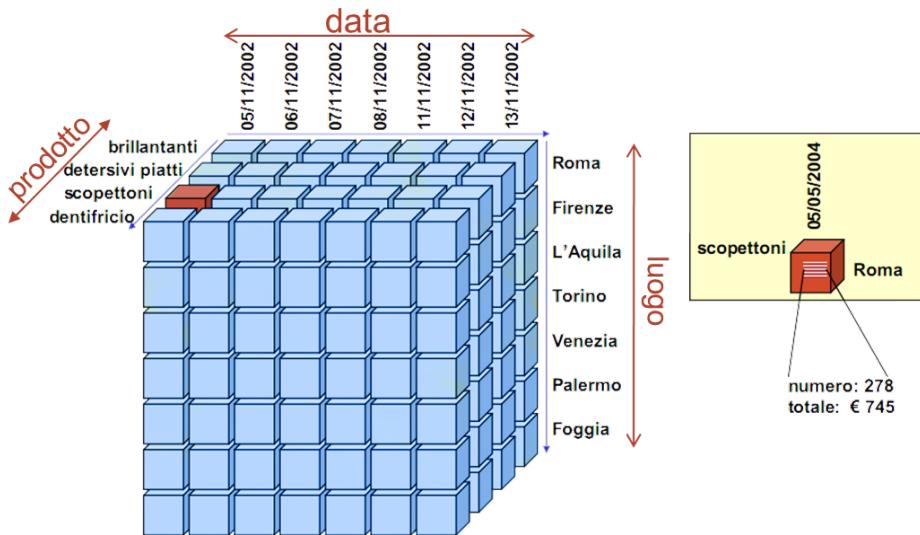
La rappresentazione multidimensionale dei dati è il concetto secondo il quale vengono organizzati logicamente i dati nel progetto di un *Data Warehouse*. Il motivo per cui il modello multidimensionale viene adottato come paradigma di rappresentazione dei dati nei *DW* è fondamentalmente legato alla sua semplicità e intuitività anche per utenti non esperti di informatica.

Nel modello multidimensionale gli oggetti che influenzano il processo decisionale sono *fatti* del mondo aziendale, quali per esempio le vendite, le spedizioni, i ricoveri. Le occorrenze di un fatto corrispondono ad *eventi* accaduti: ogni singola vendita o spedizione è un evento. Per ciascun fatto, interessano i valori di un insieme di *misure* che descrivono quantitativamente gli eventi: l'incasso di una vendita, la quantità spedita, il costo di un ricovero.

Gli eventi che occorrono nell'azienda sono troppi per essere analizzati singolarmente. Per agevolare la selezione ed il raggruppamento si immagina allora di collocarli in uno spazio *n*-dimensionale: ad esempio le vendite in una catena di negozi possono essere

CAPITOLO 1. Introduzione

Figura 1.6: Il cubo a tre dimensioni che modella le vendite in una catena di negozi



rappresentate in uno spazio tridimensionale le cui dimensioni sono i prodotti, i negozi e le date. Quindi gli eventi corrispondono a celle di un cubo i cui spigoli rappresentano le dimensioni di analisi. Ogni cella del cubo contiene un valore per ogni misura. La Figura § 1.6 mostra una rappresentazione grafica di un cubo in cui il fatto descritto è la vendita in una catena di negozi. Le dimensioni di analisi sono PRODOTTO, LUOGO e DATA; un evento corrisponde alla vendita di un certo prodotto in un certo luogo in un certo giorno, ed è descritto da due misure: la quantità venduta e l'incasso. Se si volesse rappresentare il cubo delle vendite attraverso il modello relazionale, si potrebbe adottare tale schema relazionale:

VENDITE (luogo, prodotto, data, quantità, incasso)

Non necessariamente ad un evento nel modello multi-dimensionale corrisponde un modello nel dominio applicativo; l'evento VENDITA nel modello multidimensionale corrisponde al venduto giornaliero di un prodotto in un determinato luogo, mentre lo stesso evento nel dominio applicativo corrisponde all'acquisto, da parte del cliente, di un insieme di prodotti in una certa data.

A ciascuna dimensione del modello è associata una *gerarchia* di livelli di aggregazione (chiamata *gerarchia di roll-up*). La Figura § 1.7 a fronte propone un esempio di gerarchie

Figura 1.7: Gerarchie di aggregazione sulle dimensioni PRODOTTO e NEGOZIO



sulle dimensioni PRODOTTO e NEGOZIO. In cima a ciascuna gerarchia si trova un livello fittizio in cui si raggruppano tutti i valori relativi ad una singola dimensione.

Le tecniche per ridurre la quantità di dati e ottenere informazioni utili sono essenzialmente due: la *restrizione* e l'*aggregazione*.

Restrizione. Restringere i dati significa ritagliare una porzione dal cubo circoscrivendo il campo di analisi. La forma più semplice di restrizione è lo *slicing* in cui si riduce la dimensionalità del cubo fissando un valore per una o più dimensioni. La *selezione*, invece, è una generalizzazione dello *slicing* in cui si riduce la grandezza del cubo esprimendo condizioni sugli attributi dimensionali. Infine, la *proiezione* che si riconduce alla scelta di mantenere, per ciascun evento, solo un sottoinsieme di misure, scartando le altre.

Aggregazione. L'aggregazione è un meccanismo fondamentale nelle basi di dati multidimensionali. Si supponga di voler analizzare le vendite a livello mensile: ciò significa raggruppare, per ciascun prodotto e negozio, tutte le celle relative ai giorni di uno stesso mese in un'unica macro cella. L'aggregazione può essere operata contemporaneamente su più dimensioni; inoltre combinandola con la selezione, si permette un processo di analisi mirato alle esigenze dell'utente.

1.1.4 Accedere al *Data Warehouse*

Una volta che i dati sono stati ripuliti, integrati e trasformati, occorre capire come trarne vantaggio informativo. Esistono in sostanza tre approcci differenti, supportati da altrettante

categorie di strumenti, all’interrogazione di un *DW* da parte degli utenti finali: *reportistica*, *OLAP* e *Data Mining*.

1.1.4.1 Reportistica

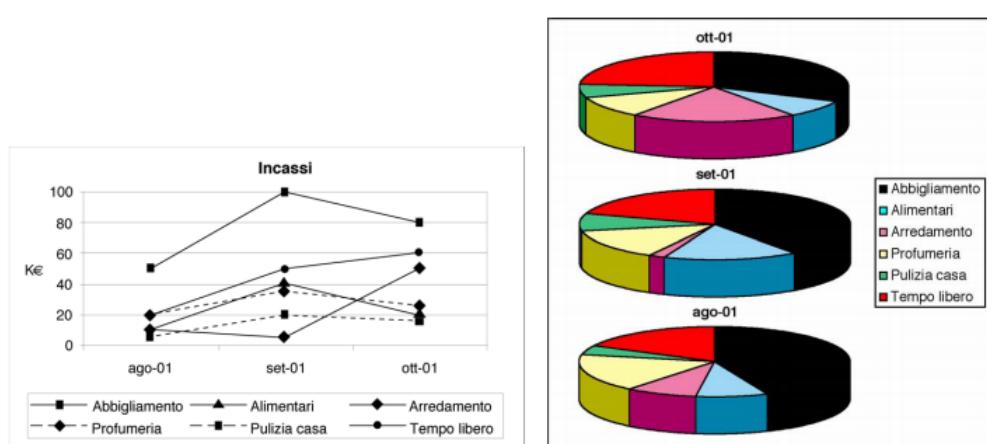
Questo approccio è orientato agli utenti che hanno necessità di accedere, a intervalli di tempo predefiniti, a informazioni strutturate in modo pressoché invariabile. Un *rappporto* è definito da un’interrogazione e da una presentazione. L’interrogazione comporta in genere la selezione e l’aggregazione dei dati multidimensionali: per esempio, può richiedere gli incassi mensili durante l’ultimo trimestre per ciascuna categoria di prodotto. La presentazione può essere in forma tabellare oppure grafica (diagramma, istogramma, torta, ecc.). Alcuni esempi sono mostrati in Figura § 1.8 nella pagina successiva.

Uno strumento di reportistica si valuta non solo in base alla ricchezza nella presentazione dei rapporti, ma anche considerando la flessibilità dei meccanismi di distribuzione. Il rapporto può essere generato su richiesta esplicita dell’utente oppure distribuito automaticamente e periodicamente.

Il *Data Warehousing* ha arrecato alla reportistica due grossi benefici: ha migliorato affidabilità e correttezza dei risultati, poiché i dati di cui i rapporti offrono le sintesi, sono ora consistenti e integrati, e incrementato la tempestività, viste le significative prestazioni di calcolo.

Figura 1.8: Presentazione di un rapporto: tabella, diagramma, torta.

	Ottobre 2001	Settembre 2001	Agosto 2001
Abbigliamento	80	100	50
Alimentari	20	40	10
Arredamento	50	5	10
Profumeria	25	35	20
Pulizia Casa	15	20	5
Tempo Libero	60	50	20



1.1.4.2 OLAP

OLAP è la principale e la più nota modalità di fruizione delle informazioni contenute in un DW. Consente a utenti le cui necessità di analisi *non siano facilmente identificabili a priori*, di analizzare ed esplorare i dati sulla base del modello multidimensionale. Gli utenti *OLAP* sono in grado di costruire attivamente una sessione di analisi complessa. Il carattere estemporaneo delle sessioni di lavoro, l’approfondita conoscenza dei dati richiesti, la complessità delle interrogazioni e l’orientamento verso utenti tipicamente non esperti di informatica, rendono cruciale il ruolo dello strumento utilizzato, la cui interfaccia deve necessariamente presentare ottime caratteristiche di flessibilità, facilità d’uso ed efficacia.

Una sessione *OLAP* consiste in pratica in un *percorso di navigazione* che riflette il procedimento di analisi di uno o più fatti di interesse sotto diversi aspetti e a diversi livelli di dettaglio. Questo percorso si concretizza in una sequenza di interrogazioni che spesso non vengono formulate direttamente, ma per differenza rispetto all’interrogazione precedente. Il risultato delle interrogazioni è di tipo multidimensionale: poiché le capacità umane di ragionare su più di tre dimensioni sono molto limitate, gli strumenti *OLAP* rappresentano tipicamente i dati in modo tabellare evidenziando le diverse dimensioni mediante intestazioni multiple, colori, ecc.

Ogni passo della sessione di analisi è scandito dall’applicazione di un *operatore OLAP* che trasforma l’ultima interrogazione formulata in una nuova interrogazione. Gli operatori più comuni sono *roll-up*, *drill-down*, *slice-and-dice*, *pivoting*, *drill-across* e *drill-through*.

Roll-up significa letteralmente arrotolare o alzare, e induce un *aumento nell’aggregazione dei dati eliminando un livello di dettaglio da una gerarchia*. Un esempio di utilizzo è mostrato in Figura § 1.9. L’operatore di *roll-up* può anche portare alla diminuzione della dimensionalità del risultato qualora tutti i dettagli di una gerarchia vengano eliminati.

Figura 1.9: Operatore *roll-up* sulla gerarchia temporale

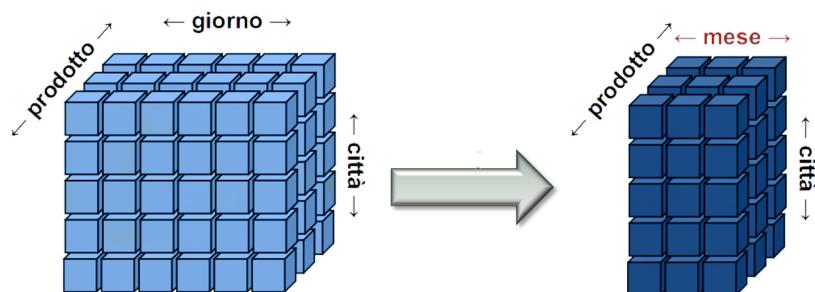
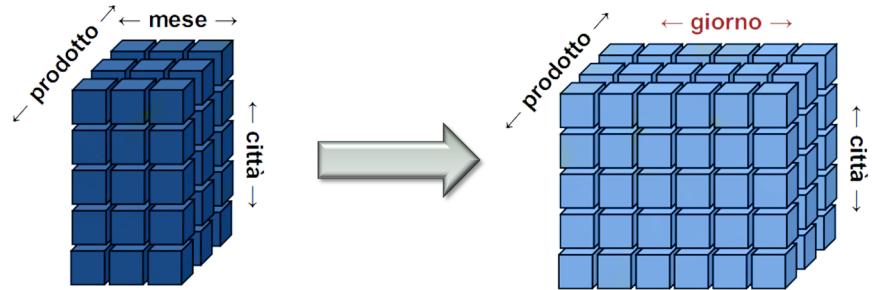


Figura 1.10: Operatore *drill-down* sulla gerarchia temporale

L'operatore *drill-down* è il duale del *roll-up*: infatti, come mostrato nella Figura § 1.10, esso diminuisce l'aggregazione dei dati introducendo un ulteriore livello di dettaglio in una gerarchia.

Con il termine *slice-and-dice* (letteralmente, tagliare a fette e cubetti) si indicano due operazioni: lo *slicing*, che riduce la dimensionalità del cubo fissando un valore per una delle dimensioni, e la *selezione* o *filtraggio*, che riduce l'insieme dei dati oggetto di analisi attraverso la formulazione di un criterio di selezione. In Figura § 1.11 è mostrato un esempio di *slicing*.

Figura 1.11: *Slicing* di un cubo sui prodotti

L'operatore *pivoting* comporta un cambiamento nella modalità di presentazione con l'obiettivo di analizzare le stesse informazioni sotto diversi punti di vista. Nella Figura § 1.12 nella pagina seguente si nota un esempio di *pivoting*: si passa dalle vendite di prodotti per mese e per città alle vendite di prodotti per città per mese.

Figura 1.12: *Pivoting* su una tabella bidimensionale



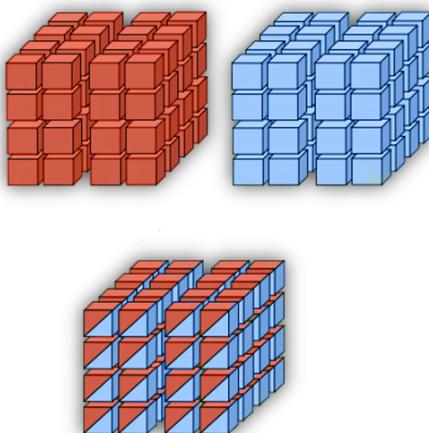
		RM	FI	AQ	TO	PA
brillantanti	ago-02	34	23	12	56	65
	set-02	56	45	23	44	67
	ott-02	76	34	34	55	45
detersivi piatti	ago-02	57	46	35	79	88
	set-02	79	68	46	67	90
	ott-02	99	57	57	78	68
scopettoni	ago-02	46	35	24	68	77
	set-02	68	57	35	56	79
	ott-02	88	46	46	67	57

		ago-02	set-02	ott-02
brillantanti	RM	34	56	76
	FI	23	45	34
	AQ	12	23	34
	TO	56	44	55
	PA	65	67	45
detersivi piatti	RM	57	79	99
	FI	46	68	57
	AQ	35	46	57
	TO	79	67	78
	PA	88	90	68
scopettoni	RM	46	68	88
	FI	35	57	46
	AQ	24	35	46
	TO	68	56	67
	PA	77	79	57

Con il termine *drill-across* si intende, invece, la possibilità di stabilire un collegamento tra due o più cubi correlati al fine di compararne i dati, per esempio calcolando espressioni che coinvolgono misure prese dai due cubi (Figura § 1.13).

Infine l'operatore di *drill-through*, resa possibile solo da alcuni strumenti *OLAP*, consiste nel passaggio dai dati aggregati multidimensionali del *DW* ai dati operazionali presenti nelle sorgenti o nel livello riconciliato.

Figura 1.13: *Drill-across* tra due cubi



1.1.4.3 *Data Mining*

Pur essendo strumenti molto potenti, i sistemi *OLAP* non sono esenti da problemi di efficienza e spreco di spazio: essi prevedono, infatti, che ogni operazione componente la richiesta dell’utente, provochi la generazione e l’esecuzione di *query* intermedie estremamente complesse, che richiedono molte risorse di elaborazione. Per superare questo problema è possibile sfruttare tecniche di *Data Mining* (principi e tecniche utili per “estrarre” conoscenza dai dati) per migliorare i sistemi *OLAP*.

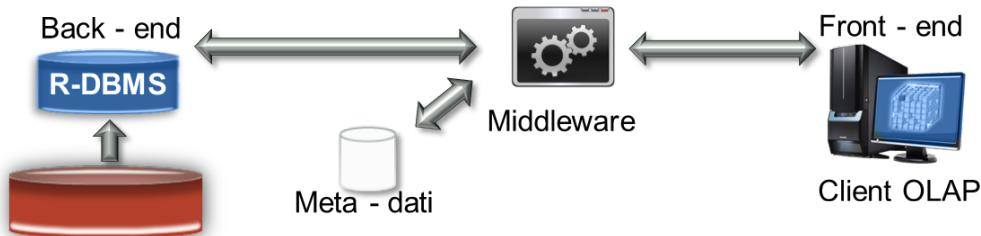
Tenendo conto dell’obiettivo finale che la presente tesi si propone di raggiungere (ovvero la progettazione di un algoritmo di *clustering* in grado di migliorare le prestazioni di un sistema *OLAP*), le tecniche di *Data Mining* saranno approfondite nella Sezione § [1.2 a pagina 25](#).

1.1.5 Modelli di Rappresentazione Fisica di un DW: *ROLAP* e *MOLAP*

Ci sono diversi approcci per implementare i sistemi di *DW* che dipendono dal modello logico utilizzato per la rappresentazione dei dati. La scelta di un sistema piuttosto che un altro, dipende dalla distribuzione dei dati (densi o sparsi o entrambi). Quindi in alcuni casi può essere più conveniente utilizzare una rappresentazione relazionale tramite server *ROLAP* (*Relational OLAP*) piuttosto che una rappresentazione matriciale implementata attraverso un server *MOLAP* (*Multidimensional OLAP*). Un altro motivo che spiega l’esistenza di diversi modelli per l’implementazione è la scarsa espressività del modello relazionale, il quale non include concetti di dimensione, misura e gerarchia.

L’idea di adottare tecnologie relazionali per la memorizzazione dei dati nel *DW* è giustificata dalla diffusa esperienza aziendale sull’utilizzo e l’amministrazione di basi di dati relazionali, e dall’elevato livello di prestazioni e flessibilità raggiunto dai *DBMS* relativi. Il problema principale delle implementazioni *ROLAP* è legato alle prestazioni, che soffrono della necessità di eseguire costose operazioni di *join* tra tabelle di elevate dimensioni. Per evitare tali *join*, si rende necessario *de-normalizzare*, al fine di massimizzare le prestazioni. Per ridurre i costi di esecuzione si ritiene, invece, necessario materializzare un certo numero di tabelle che offrono una lettura sintetica dei dati utile per le tipiche interrogazioni *OLAP* fortemente aggregate.

Figura 1.14: Architettura *ROLAP*



Dal punto di vista architettonico, questa soluzione richiede l'adozione di un *middleware* specializzato intermediario tra un *server back-end* relazionale e il lato *front-end*, come illustrato in Figura § 1.14.

Diversamente dal *ROLAP*, un sistema *MOLAP* si basa su un modello logico *ad hoc* sul quale i dati e le operazioni multidimensionali possono essere direttamente rappresentati: infatti, i dati vengono fisicamente memorizzati in vettori (*array*) e l'accesso è di tipo posizionale. Il grosso vantaggio dell'approccio *MOLAP* rispetto a quello *ROLAP* è che le operazioni multidimensionali sono realizzate in modo semplice e naturale senza *join*, portando a prestazioni ottime.

Esiste anche un terzo approccio, intermedio tra i precedenti: il cosiddetto *HOLAP (Hybrid OLAP)*. In questo approccio si possono utilizzare entrambi i sistemi (relazionale o multidimensionale), a seconda di cosa è conveniente. Se la distribuzione dei dati è meno densa normalmente si utilizza il sistema relazionale, altrimenti quello multidimensionale.

1.2 *Knowledge Discovery in Databases*

Abbiamo visto che i sistemi di *Data Warehousing* sono finalizzati all’orientamento delle attività di business di collezionare e pulire i dati, provenienti da sistemi transazionali, e renderli facilmente consultabili per analisi in linea e supporto alle decisioni. Una volta che l’organizzazione ed i propri membri hanno migliorato il patrimonio informativo, nasce spontanea la domanda: “cosa farne di tutti questi dati?”. Qui entrano in gioco le grandi potenzialità del processo di scoperta della conoscenza a partire da un capitale informativo, volte ad agevolare notevolmente il lavoro dei *knowledge worker*: il processo di *Knowledge Discovery in Databases (KDD)*.

Ad un alto livello di astrazione, il *KDD* può essere visto come il processo di sviluppo di metodi e tecniche che possano interpretare i dati. Il problema di fondo di tale processo è quello di rendere possibile un *mapping* fra i dati di basso livello, tipicamente troppo voluminosi da essere umanamente intelligibili, ed altre forme di rappresentazione più compatte, come un *report*, o più astratte, come un’approssimazione del modello che ha generato i dati, oppure più utili, come un modello predittivo che aiuti nella previsione di casi futuri.

Nell’approccio classico, per trasformare i dati in conoscenza, l’analista esperto, che ha grande confidenza con il problema da trattare, funge da interfaccia fra i dati e gli utilizzatori dei risultati. Ad esempio è comune in Scienza, Finanza, Commercio, Sanità, ecc, che gli specialisti analizzino andamenti e mutazioni dei dati con cadenza trimestrale e riportino poi le osservazioni fatte al direttivo gestionale dell’organizzazione, la quale, sulla base di questi *report*, opera scelte strategiche e pianificazioni future. Si intende subito che questa indagine manuale sui dati risulta essere lenta, faticosa, dispendiosa ed altamente soggettiva. Nasce quindi la necessità di automatizzare, almeno parzialmente, tale processo, specialmente laddove la crescita esponenziale della mole di dati renderebbe l’approccio classico completamente impraticabile. Infatti, i *database* crescono in due direzioni: il numero di *record* ed oggetti, ed il numero *d* di campi ed attributi di un oggetto. La possibilità di scalare le capacità umane di analisi su grandi moli di dati è una questione sia economica che scientifica. Le aziende osservano i dati per ottenere vantaggi competitivi sul mercato, incrementare l’efficienza delle proprie attività, ed offrire servizi di qualità ai clienti. Poiché i calcolatori hanno reso possibile la collezione di molti più dati di quanto un essere umano possa assimilarne, è naturale pensare allo sviluppo di metodologie che possano ricavare da essi strutture e modelli significativi per il supporto alle decisioni.

1.2.1 Il Processo di KDD

Abbiamo visto che gli strumenti *OLAP* hanno come obiettivo il semplificare e supportare l’analisi interattiva dei dati, coadiuvata dall’analisi multidimensionale, per sua natura più potente dei costrutti *SQL*. L’obiettivo del processo *KDD* è più ambizioso, ovvero automatizzare il più possibile tale processo analitico. Quindi il *KDD* va oltre ciò che è attualmente supportato dalla maggior parte dei *DBMS*.

Di seguito si riporta la definizione di Fayyad e Piatetsky-Shapiro, i primi ad usare il termine *KDD* in [FPSS96]:

“Il KDD è il processo non banale di identificazione di pattern nei dati che siano validi, nuovi, potenzialmente utili ed infine comprensibili”.

Il termine *processo* implica la natura multifase del *KDD*. Con “*non banale*” si sottolinea il fatto che tale processo non si limita ad una semplice computazione di parametri predefiniti, ma bensì consta di varie forme di inferenza³. I *pattern* scoperti devono essere *validi*, cioè adattabili, con un certo grado di certezza, anche ai dati “del futuro”. Si vuole inoltre che i *pattern* siano *nuovi*, se non per l’utente, almeno per il sistema; ed anche *potenzialmente utili*, cioè in grado di apportare benefici agli utenti; infine, che siano *comprendibili*, se non immediatamente, almeno dopo averli opportunamente processati. Questa discussione implica che si possono definire misure quantitative di valutazione dei *pattern* estratti. In molti casi è possibile definire una misura della certezza, ad esempio una stima dell’accuratezza di previsione su nuovi dati; o una misura dell’utilità, ad esempio il guadagno, misurato in denaro risparmiato grazie a predizioni migliori o a diminuzioni del tempo di risposta del sistema.

Il processo di *KDD* è interattivo ed iterativo (Figura § 1.15 a pagina 28): comprendente numerosi passi che necessitano di molte scelte da parte dell’utente, che di solito è l’analista, e possono ripetersi in iterazioni multiple.

La precondizione del processo è la definizione degli obiettivi di analisi.

Il primo *step* è la creazione di un insieme di *dati di riferimento*. I dati grezzi vengono segmentati e selezionati secondo l’obiettivo dell’analisi in corso, al fine di pervenire ad

³L’inferenza è il processo con il quale da una proposizione accolta come vera, si passa a una proposizione la cui verità è considerata contenuta nella prima.

Inferire X significa concludere che X è vero; un’inferenza è la conclusione tratta da un insieme di fatti o circostanze.

un sottoinsieme di dati, che rappresentano il nostro *target* data o dati obiettivo, sui quali effettuare il processo di scoperta. Per esempio, se l’obiettivo è lo studio delle associazioni tra i prodotti di una catena di supermercati, non ha senso conservare i dati relativi alla professione dei clienti.

Il secondo passo è quello della pulizia dei dati e del *pretrattamento*. Spesso, pur avendo a disposizione i dati obiettivo, non è conveniente né, d’altra parte, necessario analizzarne l’intero contenuto. È, invece, necessario:

- effettuare una prima valutazione della rilevanza dei dati raccolti;
- effettuare la rimozione del “rumore” dai dati e la raccolta di informazioni per modellare o stimare il tipo di rumore;
- decidere strategie e politiche per la gestione dei campi dei dati mancanti.

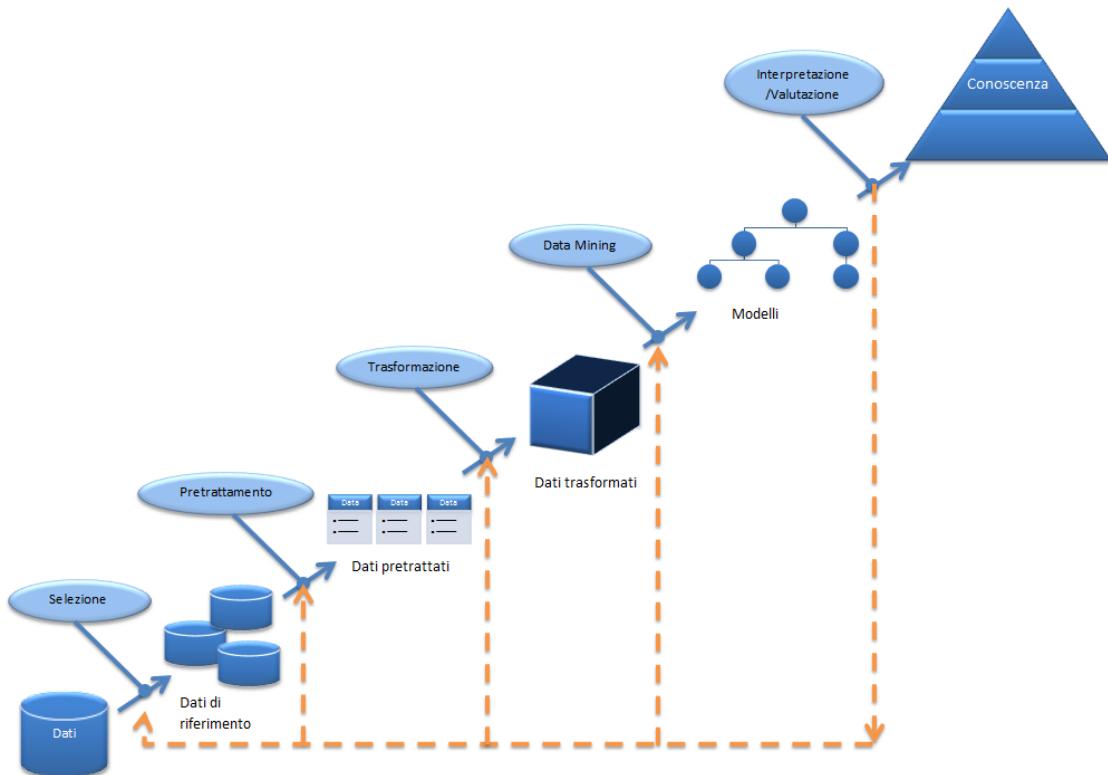
Terzo punto è la *trasformazione dei dati*, nel quale si effettua una riduzione dimensionale degli stessi, con vari metodi di trasformazione.

La quarta fase è quella in cui si effettua *Data Mining (DM)*, ovvero la scoperta e l’estrazione dei modelli nascosti nei dati, sfruttando sofisticate analisi statistiche e tecniche di modellazione dei dati, cercando di inquadrare gli obiettivi dell’intero processo (lo studio di tali tecniche sarà oggetto della Sezione § 1.3 nella pagina successiva). I tipi di dati che si hanno a disposizione e gli obiettivi che si vogliono raggiungere possono dare un’indicazione circa il tipo di metodo/algoritmo da scegliere per la ricerca di informazioni dai dati.

Il quinto stadio del processo è l’*interpretazione* e la *valutazione* dei *pattern* estratti eventualmente ripercorrendo i passi precedenti, attraverso opportune retroazioni, come descritto in Figura § 1.15. Non basta quindi interpretare i risultati dei grafici prodotti dal *DM*; occorre valutare questi modelli e capire in quale misura questi possono essere utili, e se necessario eliminare i *pattern* ridondanti.

Il passo ulteriore e finale del processo, che ne rappresenta anche lo scopo ultimo, è quello di integrare i modelli acquisiti con altri sistemi per operare ulteriori decisioni.

Figura 1.15: Panoramica dei passi che compongono il processo di *KDD*



1.3 *Data Mining: obiettivi, metodologie e tecniche*

Il *DM* è ritenuta la fase più significativa dell'intero processo di *KDD*, e questa sua enorme importanza, rende sempre più difficile distinguere il processo di *KDD* dal *DM*. Gli sviluppi di tale tecnica, per lo più abbastanza recenti, e gli ampi campi di applicazione, soprattutto nelle diverse tipologie di business, fanno sì che il *DM* risulti spesso un concetto piuttosto vago e caratterizzato da varie definizioni. Ecco allora le definizioni più comuni di *DM*:

“Il DM è la non banale estrazione di informazione implicita, precedentemente sconosciuta e potenzialmente utile attraverso l'utilizzo di differenti approcci tecnici.” (Frawley, Piatetsky-Shapiro e Matheus in [[FPSM91](#)]).

“Il DM è l'esplorazione e l'analisi, attraverso mezzi automatici e semiautomatici, di grosse quantità di dati allo scopo di scoprire modelli e regole significative (Berry e Linoff in [[BL97](#)]).

“Il DM è la ricerca di relazioni e modelli globali che sono presenti in grandi database, ma che sono nascosti nell’immenso ammontare di dati, come le relazioni tra i dati dei pazienti e le loro diagnosi mediche. Queste relazioni rappresentano una preziosa conoscenza del database e, se il database è uno specchio fedele, del mondo reale contenuto nel database.” (Holshemier e Siebes in [HS94]).

“Il Data Mining è un processo atto a scoprire correlazioni, relazioni e tendenze nuove e significative, setacciando grandi quantità di dati immagazzinati nei repository, usando tecniche di riconoscimento delle relazioni e tecniche statistiche e matematiche.” (Gartner Group)

Chiarita questa distinzione concettuale, è possibile procedere nella panoramica degli obiettivi primari del *DM* e dei metodi che si adottano per raggiungere tali obiettivi.

1.3.1 Obiettivi del *Data Mining*

IBM ha identificato due modi di operare⁴, che possono essere usati per estrarre informazioni di interesse per l’utente: i *Verification model*, con un approccio al *DM* di tipo *top-down*, e i *Discovery model*, con un approccio di tipo *bottom-up*.

Nel primo caso, si tratta di utilizzare la statistica come guida per l’esplorazione dei dati, cercando di trovare conferme a fatti che l’utente ipotizza o già conosce, o per migliorare la comprensione di fenomeni parzialmente conosciuti. Per la divisione *marketing* di un’azienda, ad esempio, che ha a disposizione un *budget* limitato per il lancio di un nuovo prodotto, è importante identificare la sezione di popolazione più propensa all’acquisto dell’articolo. L’utente formula un’ipotesi per identificare i potenziali clienti e le caratteristiche in comune. I dati storici sugli acquisti dei clienti e le informazioni demografiche verranno recuperate per rivelare acquisti simili e le caratteristiche condivise dai clienti, che saranno usati come obiettivo della campagna promozionale.

In quest’ambito vengono utilizzate le statistiche di base, che permettono di ottenere descrizioni brevi e concise del dataset, di evidenziare interessanti e generali proprietà dei dati. Il processo di ricerca è iterativo nel senso che l’*output* è revisionato e riesaminato;

⁴http://www.almaden.ibm.com/cs/projects/iis/hdb/Projects/data_mining/whitepaper.html#data-mining

CAPITOLO 1. Introduzione

si pongono nuove domande e vengono riformulate le ipotesi allo scopo di raffinare la ricerca. L’utente verifica i fatti a partire dai dati, utilizzando una varietà di tecniche come *query* articolate, analisi multidimensionale e visualizzazione dati per guidare l’esplorazione. Ciò tipicamente viene fatto su sistemi *OLAP* con gli strumenti standard incorporati. Si tratta di tecniche che hanno alla base un’analisi essenzialmente deduttiva. Tuttavia, quando il numero delle variabili cresce, l’utilizzo delle metodologie *OLAP* diventa sempre più difficoltoso, perché diventa difficile, e anche dispendioso, in termini di tempo, formulare delle buone ipotesi da saggiare. Risulta quindi più utile ricorrere alle tecniche di *DM* che liberano l’utente da compiti specifici.

Con il modello di indagine (*Discovery model*) il sistema trova autonomamente nuovi *pattern*. I dati sono setacciati alla ricerca di comportamenti frequenti, *trend* e generalizzazioni senza l’intervento dell’utente. Gli strumenti di *DM* permettono l’individuazione di molti “fatti”, in un tempo relativamente breve. La reale utilità del processo di *DM* è quello di scoprire andamenti nei dati che sarebbero altrimenti invisibili “ad occhio nudo”, quindi il modello di scoperta ne rappresenta la vera essenza.

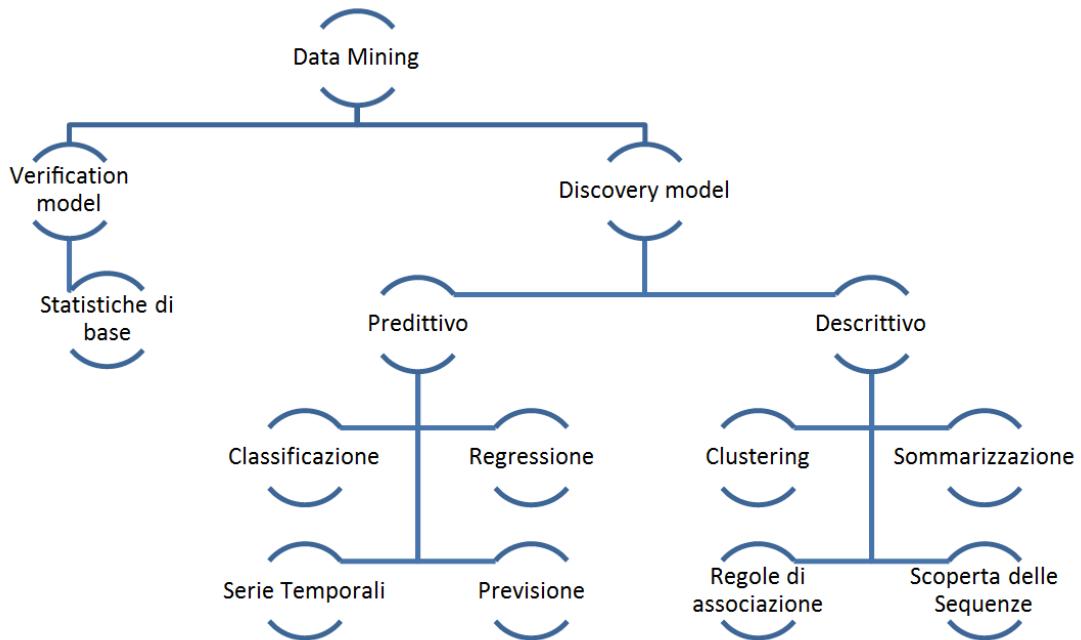
Si possono ulteriormente suddividere gli obiettivi di scoperta in *predittivi*, dove il sistema scopre i *pattern* per predire comportamenti futuri di alcune entità, e *descrittivi*, dove il sistema presenta i *pattern* scoperti, in forma umanamente intelligibile.

La maggior parte dei metodi di *DM* sono tecniche consolidate che provengono dall’intelligenza artificiale, dal riconoscimento di strutture e dalla statistica: classificazione, clusterizzazione, regressione e così via.

Alla base di questi metodi, c’è l’apprendimento induttivo (*inductive learning*), cioè l’inferenza di informazioni dai dati per il processo costruttivo del modello dove l’ambiente, ovvero il *database*, è analizzato in modo da ritrovare i *pattern*.

Il processo di inferenza della conoscenza ha due principali strategie:

- **Apprendimento supervisionato:** è l’apprendimento basato su esempi che vengono forniti al sistema per la costruzione del modello, che definisce le classi in cui raggruppare i dati. Vengono insomma fornite classi esemplari. Una volta formulata la descrizione che forma la regola, essa può essere utilizzata per predire la classe di nuovi oggetti.
- **Apprendimento non supervisionato:** è l’apprendimento dall’osservazione e dalla scoperta. Il sistema è munito degli oggetti ma non è definita nessuna classe, co-

Figura 1.16: Tipologie di *Data Mining*

sicché esso deve osservare gli esempi o riconoscere i *pattern* autonomamente. Il risultato consiste in un insieme di descrizioni di classe, una per ogni classe osservata nell’ambiente.

1.3.2 Metodi per diversi problemi

I due obiettivi principali di alto livello del *DM*, come si è visto, sono la predizione e la descrizione. Come prima accennato la predizione concerne l’uso di alcune variabili o campi dei dati per prevedere valori futuri (e sconosciuti) di altre variabili d’interesse, mentre la descrizione si concentra sul ritrovamento di forme descrittive dei dati di facile comprensione per l’utente. Tuttavia la distinzione fra *DM* predittivo e descrittivo non è nitida (alcuni modelli predittivi possono essere considerati descrittivi per il loro grado di comprensibilità e viceversa), ma è importante per la comprensione dell’obiettivo ultimo dell’applicazione da sviluppare. Gli obiettivi della predizione e della descrizione possono essere raggiunti tramite l’utilizzo di svariate tecniche e metodi che caratterizzano la tipologia del problema.

CAPITOLO 1. Introduzione

Di seguito saranno descritte sinteticamente le tecniche di *Data Mining* più conosciute, per poi approfondire solo quelle sulle quali si focalizza il lavoro della presente tesi.

Classificazione: mira a determinare l'appartenenza di un insieme di oggetti, caratterizzati da alcuni attributi, ad alcune classi, ed etichettarli opportunamente.

È forse la più comune tecnica applicata nelle analisi del *Data Mining*. L'algoritmo di classificazione viene applicato inizialmente a un insieme di esempi di transazioni per l'addestramento, così da determinare un modello di classificazione chiamato “classificatore”. Una volta che quest'ultimo è stato sviluppato, viene usato per classificare nelle stesse predefinite classi, gli altri dati non presenti nel *training set*. Ad esempio, l'assegnazione di un punteggio ad un nuovo cliente da parte di una banca, serve a stimare il rischio di credito su un finanziamento. Questo può essere visto come un problema di classificazione creando 2 classi: “clienti buoni” e “clienti cattivi”. Il modello di classificazione è costruito a partire dai clienti esistenti, in base agli attributi che ne descrivono il comportamento. Tale modello è usato poi per assegnare al nuovo cliente una delle due etichette.

Regressione: mira ad identificare una relazione funzionale tra variabili misurate sulla base di dati campionari estratti da un'ipotetica popolazione infinita.

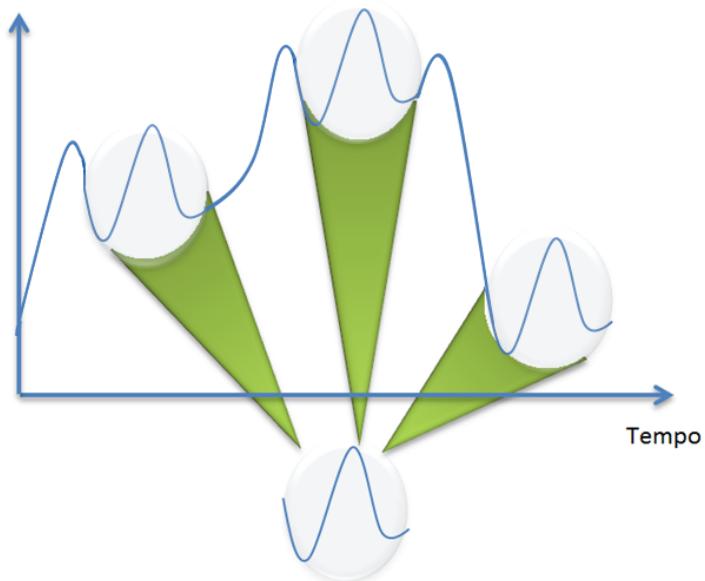
L'obiettivo della regressione è valutare se una variabile si manifesta in funzione di un'altra. Ad esempio le entrate annuali di una compagnia internazionale sono correlate ad altri attributi come il tasso di inflazione, il tasso di cambio, le campagne pubblicitarie ecc. Esistono due tipi di regressione:

- **Regressione lineare** è la tecnica matematica che può essere usata per discriminare delle variabili con una linea e che quindi generalizza un insieme di dati numerici tramite la creazione di un'equazione matematica.
- **Regressione logistica** stima la probabilità che si verifichi un evento in determinate circostanze, utilizzando i fattori osservati assieme all'occorrenza o non occorrenza dell'evento.

Serie temporali: mirano ad individuare *pattern* ricorrenti o atipici in sequenze di dati complesse.

Sono simili alle regressioni, ma usano le supplementari proprietà delle informazioni temporali per predire valori futuri. La caratteristica di questa classe di tecniche è proprio la dipendenza dei valori dal tempo. Possono essere rilevanti per il futuro, la gerarchia dei periodi (come le 13 mensilità e i giorni lavorativi), le stagionalità, le festività, le vacanze, ecc.. Una volta creato un modello, è opportuno continuare a testarlo e modificarlo non appena sono disponibili nuovi dati. Un esempio di applicazione delle serie storiche sono le analisi finanziarie che provano a predire l'andamento di titoli (o azioni), partendo da una serie di eventi precedenti. La Figura §1.17 illustra un'esemplificazione della ricerca di serie temporali ricorrenti.

Figura 1.17: Ricerca di Serie Temporali



Previsione: mira a trovare valori noti per la predizione di quantità non note.

È legato all'analisi numerica e alla statistica, in quanto si utilizzano funzioni di interpolazione (lineari o non) che descrivono i valori predetti. Un esempio potrebbe essere la stima del fatturato di un punto vendita sulla base delle sue caratteristiche.

Clusterizzazione: mira alla separazione dei dati in sottogruppi o classi significative, non assegnate a priori, e allo stesso tempo, all'individuazione di oggetti somiglianti.

Sebbene la classificazione sia un mezzo efficace per distinguere gruppi o classi di oggetti, essa richiede una costruzione ed un’etichettatura del *training set*, operazioni spesso costose. Può essere utile, invece, procedere in senso inverso, partizionando prima i dati in gruppi sulla base della loro similarità, e successivamente, assegnando le etichette ad un numero relativamente piccolo di gruppi ottenuti.

L’analisi dei *cluster* si utilizza largamente in numerose campi dello scibile, per esempio per il trattamento delle immagini, ricerche di mercato, scoprire gruppi distinti di clienti caratterizzandoli in base ai loro acquisti, derivare le tassonomie delle piante e degli animali, categorizzare i geni con funzionalità simili, esaminare varie caratteristiche delle popolazioni, identificare aree terrestri con uso simile, identificare gruppi di assicuratori di macchine che hanno la stessa politica, identificare gruppi di case in una città a seconda del tipo di casa, del suo valore e della sua locazione geografica, ecc..

Il *clustering* trova utilizzo anche nella ricerca degli *outlier*⁵; in alcune applicazioni gli *outlier* sono ancora più importanti dei valori comuni. Si pensi, ad esempio, alla ricerca delle frodi nelle carte di credito: casi eccezionali nelle transazioni delle carte di credito, ad esempio acquisti molto costosi e frequenti, possono essere indici di attività fraudolenta. Come funzionalità del *Data Mining*, il *clustering* si utilizza per esaminare le distribuzioni dei dati, per osservare le caratteristiche di ciascuna distribuzione e per focalizzarsi su quelle di maggiore interesse. Alternativamente, si potrebbe utilizzarlo come un passo di *pre-processing* per altri algoritmi, quali la classificazione e la caratterizzazione, che operano sui *cluster* individuati.

È un metodo di *Data Mining* con apprendimento non supervisionato, e quindi utile per raggruppare una grande mole di dati, sulla base di relazioni non note a priori: permette di scegliere dinamicamente quali sono le caratteristiche distintive dei vari gruppi. È il caso delle sessioni *OLAP*, caratterizzate da analisi dinamiche e multidimensionali, che richiedono la scansione di una grande quantità di dati per calcolare un insieme di dati numerici di sintesi. Per i motivi appena esposti, alla luce dell’obiettivo finale della presente tesi, la tecnica di *Data Mining* in esame, sarà approfondita nella Sezione § 1.5 a pagina 39.

⁵Definiti anche punti singolari, sono valori molto lontani da ciascun *cluster*

Sommarizzazione: mira alla descrizione concisa della natura dei dati, in forma sia elementare che aggregata.

Questo dà all'utente una veduta generale della struttura dei dati, per esempio, serve per indicare le varie caratteristiche per cui un fungo si può dire velenoso.

Regole di Associazione: mirano ad individuare gli argomenti che occorrono insieme con un dato evento o *record*.

La presenza di un insieme di argomenti implica la presenza di un altro insieme. Vengono così individuate delle regole del tipo: se si verifica l'evento A, allora per una determinata percentuale di volte si verifica anche l'evento B. Per esempio, se un cliente acquista dei pop-corn, allora acquista anche la cola senza che sia in promozione il 63% delle volte, mentre se è in promozione il cliente acquista la cola l'84% delle volte. Visto il largo uso delle casse con scanner, le quali raccolgono i dettagli delle transazioni delle vendite al minuto, le analisi delle associazioni vengono anche chiamate “*market-basket analysis*” (MBA). I risultati di questa analisi possono essere usati per pianificare strategie di *marketing* e pubblicitarie, così come lo sviluppo del catalogo dei prodotti, ma anche suggerire il posizionamento dei prodotti sugli scaffali all'interno del locale, ovvero la progettazione della disposizione dei prodotti in un negozio. Ad esempio se una regola dice che chi compra un computer, probabilmente comprerà anche un software di gestione finanziaria, un approccio è quello di disporre su scaffali vicini i due prodotti per incentivare il cliente all'acquisto di entrambi; un'altra strategia è quella di posizionare i prodotti alle estremità del locale, cosicché il cliente intenzionato all'acquisto degli articoli possa notare e comprare altri articoli durante il tragitto. Ancora, le regole di associazione possono suggerire anche le politiche promozionali, ovvero quali articoli mettere in vendita a prezzo ridotto e come comporre i pacchetti in promozione; se è noto da una regola che chi compra un computer compra anche una stampante, si possono vendere i due articoli in un'unica promozione, ciò verosimilmente incoraggerà l'acquisto di entrambi.

Scoperta delle Sequenze: mira ad individuare gli argomenti che occorrono insieme con un dato evento o *record*, in un determinato lasso di tempo.

È simile all'analisi delle associazioni, eccetto che le relazioni tra gli argomenti sono condizionate dal tempo. Di regola per trovare queste sequenze, si devono catturare

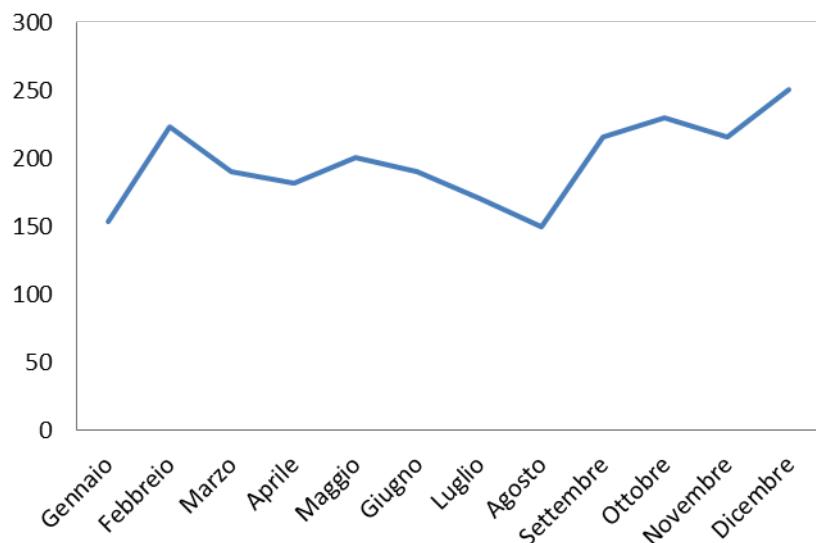
CAPITOLO 1. Introduzione

non solo i dettagli di ogni transazione, ma anche l'identità di chi fa la transazione. Per esempio, eseguito un trattamento chirurgico X, occorre il 57% delle volte, un'infezione Y, entro 3 giorni da esso. Questa analisi può anche riscontrare, ad esempio, che dopo 3 giorni, la probabilità dell'infezione Y, si riduce al 4%. Altri esempi possono essere la scoperta che i genitori comprano una determinata marca di *videogame* entro due settimane dalla visione di un film per ragazzi.

1.4 ***DM e OLAP: Tecniche alternative o complementari?***

È utile riportare un **esempio**⁶ tratto dal sito internet della società *SPSS*⁷ la quale, fra le altre attività, comprende anche quella della implementazione e diffusione di algoritmi di *DM*. Nell'esempio la procedura *OLAP* e il *DM* vengono viste come tecniche complementari: il *DM* consente agli utenti di strumenti *OLAP* di andare oltre i *report* riassuntivi. Il *DM* dice perché un certo fenomeno sta succedendo, mentre l'*OLAP* si limita a dire cosa sta succedendo. Per esempio, il *DM* può scoprire gruppi di clienti o di prodotti che condividono caratteristiche simili. Per capire cosa significa, diamo uno sguardo ai dati giornalieri di acquisizione di clienti di una banca (Figura § 1.18).

Figura 1.18: Acquisizione giornaliera clienti



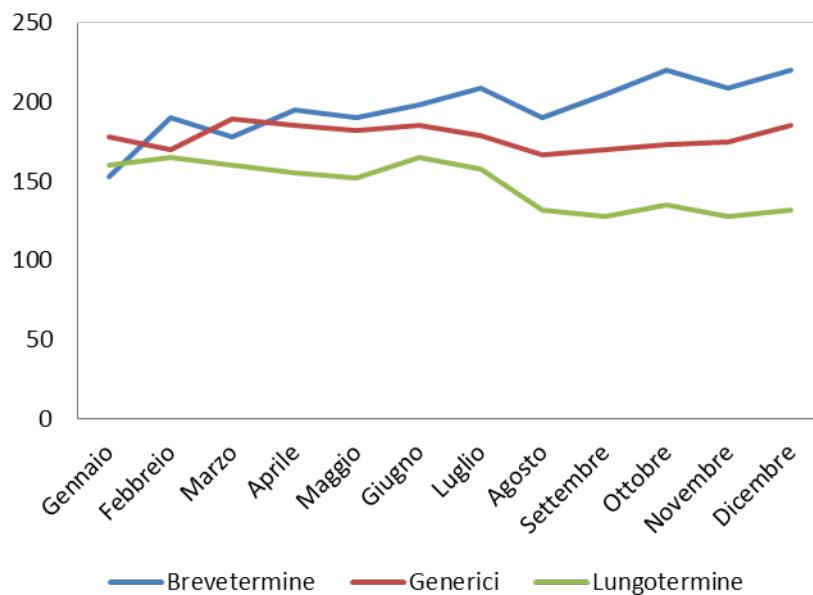
Il risultato dello strumento *OLAP* è un grafico che fornisce un'informazione molto chiara: l'acquisizione di clienti sta seguendo un *trend* positivo, nonostante abbia avuto una flessione nei mesi centrali dell'anno. Se però si procede con ulteriori analisi e si affiancano alle tecniche *OLAP* le tecniche di *DM*, emergono delle informazioni diverse. Procedendo ad una segmentazione dei clienti, fatta tramite una *cluster analysis*, si osserva la distribuzione della clientela rappresentata nella Figura § 1.19 nella pagina seguente.

⁶<http://www.spss.it/solutions/datamine/techniques.htm>

⁷*Statistical Package for Social Science*, società del gruppo IBM, leader mondiale nello sviluppo e nella distribuzione di *software* e soluzioni per l'analisi predittiva.

CAPITOLO 1. Introduzione

Figura 1.19: Acquisizione giornaliera per tipologia clienti



La figura mostra che l'acquisizione sta aumentando fra i clienti di Breve Termine (*BT*), è sostanzialmente stabile fra i clienti definiti Generici (*GEN*) e sta calando fra quelli di Lungo Termine (*LT*). Dal momento che i clienti di Lungo Termine sono i più interessanti per la banca, questa tendenza rappresenta un problema. Disponendo di questa ripartizione è innanzitutto possibile rilevare il problema, e di conseguenza è possibile studiare azioni specifiche di marketing dirette a invertire la tendenza. È quindi evidente che gli strumenti *OLAP* rappresentano una base di partenza, ma non sono in grado di fornire lo stesso contributo informativo delle tecniche di *DM*. Tuttavia l'esempio dimostra come le tecniche di *OLAP* e *DM* siano **tecniche complementari**⁸ piuttosto che alternative.

⁸http://www.01net.it/articoli/0,1254,1_ART_62598,00.html

1.5 Il *Clustering*

In questa sezione saranno illustrate le principali metodologie e tecniche di *clustering*, approfondendo solo quelle di maggiore interesse per il fine ultimo della presente tesi.

Il *Clustering* o *analisi dei cluster* (dal termine inglese *cluster analysis* introdotto da Robert Tryon nel 1939), è

“il processo di raggruppamento di un insieme di oggetti fisici o astratti in classi di oggetti simili”. (Han 2001)

Come già accennato, il *clustering* sfrutta l'apprendimento non supervisionato, quindi esso apprende dall'osservazione, piuttosto che dagli esempi: per intenderci è come avere un insieme di oggetti da analizzare per cui, a differenza della classificazione, non sono note né il “significato” di ogni cluster (l'etichetta della classe), né il numero di cluster da creare (non sempre).

L'obiettivo della tecnica in questione è creare gruppi di oggetti massimamente omogenei al loro interno, e massimamente eterogenei tra loro, autonomamente, al fine di ottenere una riduzione della dimensione dei dati da analizzare, garantendo però la minima perdita di informazioni.

Il risultato di una *cluster analysis*, quindi, è il numero di gruppi, detti “*cluster*”, e la loro relativa composizione. Un *cluster* è definito come una collezione di oggetti “simili” tra loro, e “dissimili” rispetto agli oggetti degli altri *cluster*.

Come abbiamo già accennato nella Sezione § 1.3.2 a pagina 33, il clustering rappresenta un ampio campo di ricerca, i cui sforzi sono mirati a soddisfare e migliorare alcuni dei requisiti di base per la sua applicabilità al *DM*. I maggiori requisiti sono:

1. **Scalabilità.** Molti algoritmi di *clustering* lavorano bene su piccoli insiemi di dati; tuttavia, un grande database può contenere milioni di oggetti. Si rendono, quindi, necessari algoritmi di *clustering* altamente scalabili.
2. **Capacità di trattare diversi tipi di attributi.** Molti algoritmi sono progettati per clusterizzare dati numerici. Tuttavia, le applicazioni possono richiedere il *clustering* di altri tipi di dati, quali dati binari, categorici e ordinali, oppure una loro combinazione.

3. **Capacità di individuare *cluster* di forma arbitraria.** Molti algoritmi individuano i *cluster* basandosi su misure di distanza Euclidea o di *Manhattan*. Questi tendono a costruire *cluster* simili-sferici con dimensioni e densità simili. Tuttavia, un *cluster* potrebbe avere una forma qualunque. È importante sviluppare algoritmi che possano individuare *cluster* di forma arbitraria.
4. **Sensibilità ai parametri di *input*.** Molti algoritmi richiedono parametri di *input* (quali il numero di *cluster* desiderati), costringendo i risultati ottenuti ad essere piuttosto sensibili a tali parametri. Inoltre, quest'ultimi sono spesso difficili da determinare specialmente quando si ha a che fare con oggetti ad alta dimensionalità e non si ha una sufficiente conoscenza del dominio applicativo.
5. **Capacità di trattare valori che costituiscono rumore.** Gran parte dei database del mondo reale contengono dati mancanti, sconosciuti o errati. Alcuni algoritmi di *clustering* sono sensibili a questi dati e possono portare a *cluster* di scarsa qualità.
6. **Clustering incrementale.** Alcuni algoritmi di *clustering* non sono capaci di incorporare nuovi dati nei *cluster* già esistenti; pertanto, in presenza di nuovi dati, è necessario riprocessare interamente il *dataset*. Sarebbe importante sviluppare algoritmi capaci di aggiornare i *cluster* in modo incrementale.
Questa caratteristica è essenziale nel caso di interazione delle tecnologie di *DM* con quelle di DW OLAP, in quanto in un *Data Warehouse* i dati non vengono mai eliminati, ma sono in continua crescita. In questi casi è quindi necessario che l'algoritmo di *clustering* sia capace di unire, ai dati già processati, i nuovi.
7. **Insensibilità all'ordinamento dei dati di *input*.** Alcuni algoritmi sono sensibili all'ordine dei dati di *input*; pertanto, lo stesso insieme dei dati, quando è presentato in un ordine diverso, può generare *cluster* differenti. È importante sviluppare algoritmi che siano indipendenti dall'ordinamento dell'*input*.
8. **Alta dimensionalità.** Un *database* o un *Data Warehouse* possono contenere diverse dimensioni o attributi. Molti algoritmi di *clustering* sono capaci di gestire dati, che coinvolgono poche dimensioni (due o tre). È importante che un algoritmo possa operare in uno spazio ad alta dimensionalità, specialmente considerando che i dati in esso presenti sono tipicamente sparsi e con molti valori deviati.

9. **Adattamento ai vincoli.** Le applicazioni del mondo reale possono dover operare tenendo conto di vari tipi di vincoli.
10. **Interpretabilità e usabilità.** Gli utenti si aspettano che i risultati del *clustering* siano interpretabili, comprensibili e usabili.

1.5.1 Tipi di dati

Si supponga che un insieme di dati da raggruppare contenga n elementi. Gli algoritmi di *clustering* tipicamente operano su una delle seguenti strutture dati:

- *Una matrice di dati o binomiale* (o struttura *object-by-variable*): la riga i -esima individua l'oggetto (o individuo) i , mentre la colonna j -esima individua il valore associato alla j -esima proprietà (o caratteristica) dell'oggetto. La struttura è nella forma di una tabella relazionale, o matrice $n \times p$ (n oggetti per p attributi):

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1p} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2p} \\ x_{31} & x_{32} & x_{33} & \dots & x_{3p} \\ \dots & \dots & \dots & \dots & \dots \\ x_{n1} & x_{n2} & x_{n3} & \dots & x_{np} \end{bmatrix}$$

- *Una matrice di dissimilarità* (o struttura *object-by-object*): questa memorizza il grado di dissimilarità di ciascuna coppia degli oggetti coinvolti. È spesso rappresentata da una tabella $n \times n$, come di seguito specificata:

$$\begin{bmatrix} 0 & 0, & 0 & \dots & 0 \\ d(2, 1) & 0 & 0 & \dots & 0 \\ d(3, 1) & d(3, 2) & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ d(n, 1) & d(n, 2) & d(n, 3) & \dots & 0 \end{bmatrix}$$

dove $d(i, j)$ è la differenza o *dissimilarità* misurata tra gli oggetti i e j . Si noti che $d(i, j) = d(j, i)$ e che $d(i, i) = 0$.

Molti algoritmi di *clustering* operano sulla matrice delle dissimilarità. Se i dati sono presentati sotto forma di una matrice di dati è necessario trasformare quest'ultima in una

matrice di dissimilarità prima di applicare tali algoritmi. Il calcolo della dissimilarità può avvenire in svariati modi differenti: la scelta dipende, tanto, dal tipo di variabili coinvolte, quanto, dal contesto di riferimento.

Spesso le variabili sono di tipo diverso e possiedono quindi diverse unità di misura. A seconda della scelta dell'unità di misura, i risultati dell'analisi possono cambiare e per questo è necessario effettuare una preventiva *standardizzazione* o *normalizzazione* dei dati. Quando le variabili sono continue, la standardizzazione può consistere nel dividere la variabile per la *deviazione standard*⁹ (riducendo le variabili a varianza unitaria), calcolata utilizzando l'insieme completo di oggetti da classificare (in questo caso si parla anche di *autoscaling*). Un metodo alternativo è quello di dividere la variabile per il proprio *range*.

1.5.2 Misure di distanza

Tutte le tecniche di *clustering* si basano sul concetto di distanza tra due elementi. Infatti la bontà delle analisi ottenute dagli algoritmi di *clustering* dipende molto dalla scelta della metrica, e quindi da come è calcolata la distanza. Gli algoritmi di *clustering* raggruppano gli elementi sulla base della loro distanza reciproca, e quindi l'appartenenza o meno ad un insieme dipende da quanto l'elemento preso in esame è distante dall'insieme stesso. Di conseguenza una fase importante, in molti algoritmi di *clustering*, è quella della selezione della misura di distanza, che determina come calcolare la similarità di due elementi. Due individui sono “vicini” quando la loro dissimilarità o distanza è piccola o, equivalentemente, quando la loro similarità è grande. La scelta della misura di distanza influenza la forma del *cluster*, in quanto alcuni elementi possono risultare vicini secondo una distanza e lontani secondo un'altra.

Una misura di distanza, per essere definita tale, deve soddisfare le seguenti proprietà:

1. non negatività: $d(x, y) \geq 0 \quad \forall x, y \in \mathbb{R}^p$
2. identità: $d(x, y) = d(y, x) \iff x = y$
3. simmetria: $d(x, y) = d(y, x) \quad \forall x, y \in \mathbb{R}^p$
4. disegualianza triangolare: $d(x, y) \leq d(x, z) + d(z, y) \quad \forall x, y, z \in \mathbb{R}^p$

Per approfondimenti e dettagli relativi alle diverse misure di distanza si rimanda a [HK00].

⁹La deviazione *standard*, o scarto quadratico medio, misura la dispersione dei dati intorno al valore atteso.

1.5.3 Categorizzazione dei metodi di *clustering*

Negli anni sono stati sviluppati un gran numero di algoritmi di *clustering*. La scelta dell’algoritmo da utilizzare, in un dato contesto, dipende dal tipo di dati disponibili, dal particolare scopo e dall’applicazione. In generale, i principali metodi di *clustering* possono essere classificati come di seguito specificato.

- **Metodi di partizionamento:** organizzano gli n oggetti in k partizioni ($k \leq n$), dove ciascuna partizione rappresenta un *cluster*.

In altre parole, l’algoritmo classifica i dati in k gruppi che, nel loro insieme, soddisfano i seguenti requisiti:

1. ciascun gruppo deve contenere almeno un oggetto;
2. ciascun oggetto deve appartenere esattamente ad un gruppo.

I *cluster* sono costruiti con il fine di *ottimizzare* un criterio di partizionamento, spesso denominato funzione di similarità, come la distanza, in modo tale che gli oggetti all’interno di un *cluster* siano “*simili*”, mentre gli oggetti di *cluster* differenti siano “*dissimili*”.

Dato k , il numero di partizioni da costruire, un metodo di partizionamento crea innanzitutto un partizionamento iniziale. Su tale partizionamento viene, successivamente, applicata una tecnica di *rilocazione iterativa* che tenta di migliorarlo spostando oggetti da un gruppo ad un altro.

Il problema risiede nel fatto che, tale metodo, funziona bene solo se i *cluster* sono di forma sferica e ben distanziate gli uni dagli altri (rispettivamente Figure § 1.20 nella pagina seguente e § 1.21 nella pagina successiva); inoltre sono computazionalmente dispendiosi, quindi applicabili solo su *database* di piccole dimensioni, e non sono incrementalni.

L’ulteriore svantaggio di questo metodo risiede nel fatto che si costringe l’utilizzatore a specificare il numero dei k cluster in cui raggruppare il *dataset*, svilendo dunque il fine ultimo del metodo di apprendimento non supervisionato che costituisce il motore delle tecniche di *clustering*.

Gli algoritmi più noti che sfruttano questo metodo sono *K-MEANS* [HW79], *K-MEDOIDS* [KR90], *CLARANS* [NH94].

Figura 1.20: Suddivisione indesiderata di un grande *cluster*, dovuta all'eccessiva vicinanza dei *cluster* componenti

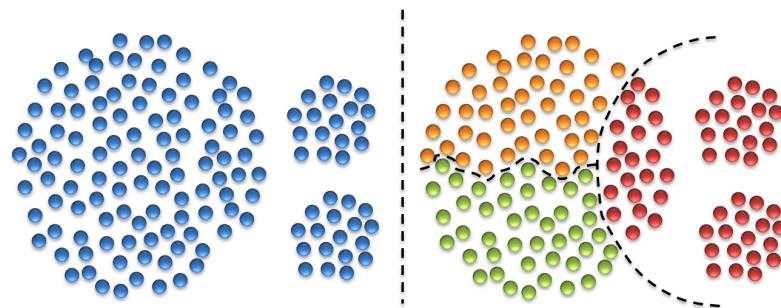
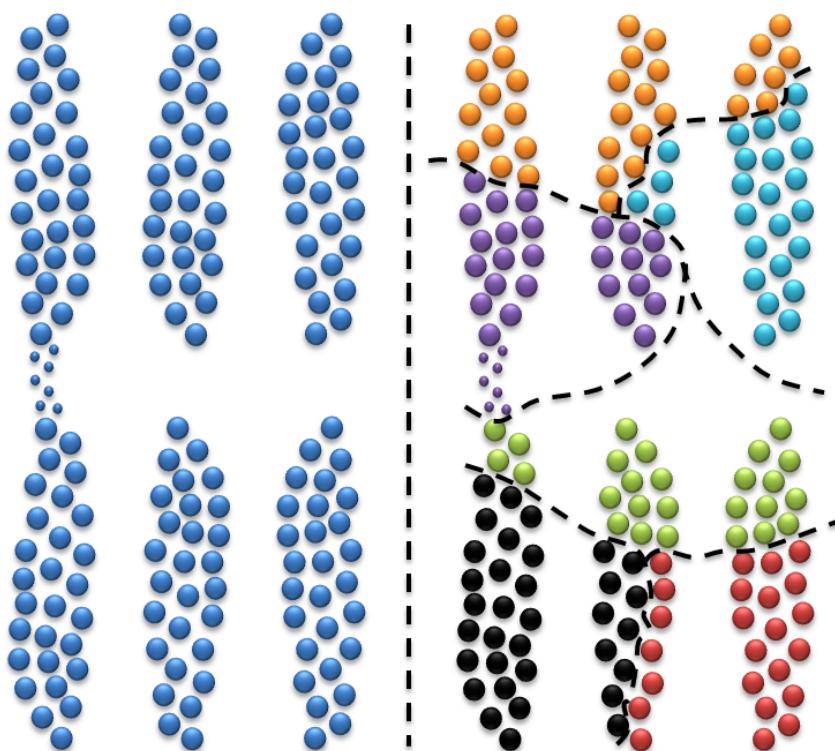


Figura 1.21: Suddivisione indesiderata dei *cluster*, dovuta alla loro forma allungata



- **Metodi basati su griglia:** quantizzano i dati in singole celle che vanno a formare una struttura a griglia e le operazioni di raggruppamento sono effettuate su questa struttura. Il vantaggio principale che si ottiene è la velocità di processazione, che è tipicamente indipendente dal numero n di oggetti. Lo svantaggio è che la qualità del *clustering* dipende dalla granularità delle celle e che il numero delle stesse aumenta esponenzialmente con la dimensionalità dei dati. Gli algoritmi più noti che sfruttano questo metodo sono *STING* [WYM97], *CLIQUE* [AGGR98] e *WaveCluster* [SCZ98].
- **Metodi basati su modelli:** suppone che le istanze del *dataset* siano state generate a partire da un mix di modelli probabilistici. Per questa ragione si parla anche di *Probabilistic Clustering*. L'algoritmo, quindi, deve trovare il migliore assegnamento dei dati rispetto al modello. Un algoritmo basato sul modello può localizzare i *cluster* costruendo una funzione di densità che riflette la distribuzione spaziale dei punti associati ai dati. Esso consente anche di determinare automaticamente il numero di *cluster* basandosi su statistiche *standard*, tenendo in considerazione il rumore e ottenendo, pertanto, metodi di *clustering* robusti.

Lo svantaggio di questo tipo di algoritmi è la richiesta, da parte dell'utilizzatore, la conoscenza dei modelli probabilistici su cui si fonda per poterne determinare i paramenti di input.

Gli algoritmi più noti che sfruttano questo metodo sono *EM* [Bil98], *COBWEB* [Fis87].

- **Metodi basati sulla densità:** si basano sul concetto di densità, considerando i *cluster* come regioni dense di oggetti nello spazio dei dati, separate da regioni a bassa densità (che rappresentano gli *outlier*). Il vantaggio è che producono cluster di ottima qualità, ma sono molto sensibili ai parametri di impostazione. Gli algoritmi più noti che sfruttano questo metodo sono *DBSCAN* [EKSX96] e tutte le sue innumerevoli versioni, *OPTICS* [ABKS99] e *DENCLUE* [HK98].
- **Metodi gerarchici:** creano una gerarchia di cluster la quale può essere rappresentata tramite una struttura ad albero chiamata *albero di classificazione gerarchica* o *dendrogramma*. La radice dell'albero rappresenta un singolo *cluster* contenente

tutte le osservazioni, e le foglie corrispondono alle singole osservazioni. Gli algoritmi più noti che sfruttano questo metodo sono *BIRCH* [ZRL96], *CURE* [GRS98], *CHAMELEON* [KHK99], *DIANA* e *AGNES* [KR90].

La natura gerarchica di questi metodi, consente una più facile integrazione con le tecnologie *OLAP*, in quanto mettono a frutto le gerarchie delle dimensioni che caratterizzano tale tecnologia. Proprio per questo motivo, saranno approfonditi tali algoritmi nella Sezione § 1.5.4 a pagina 53.

Segue un’approfondimento dei metodi più noti, che hanno portato alla maturazione della presente tesi.

1.5.3.1 K-MEANS

L’algoritmo *K-MEANS* [Mac67] si propone l’obiettivo di minimizzare la varianza totale *inter-cluster*. Ogni *cluster* viene identificato mediante un *centroide* (detto anche baricentro o punto medio). L’Algoritmo § 1 nella pagina successiva segue una procedura iterativa. Inizialmente crea k partizioni e assegna ad ogni partizione i punti d’ingresso o casualmente o usando alcune informazioni euristiche. Quindi calcola il centroide di ogni gruppo, costruisce una nuova partizione e associa i punti d’ingresso al cluster il cui centroide è più vicino. Vengono ricalcolati i centroidi per i nuovi cluster e così via, finché l’algoritmo non raggiunge la convergenza di una determinata funzione criterio. Tipicamente viene utilizzata come criterio l’errore quadratico, definito come:

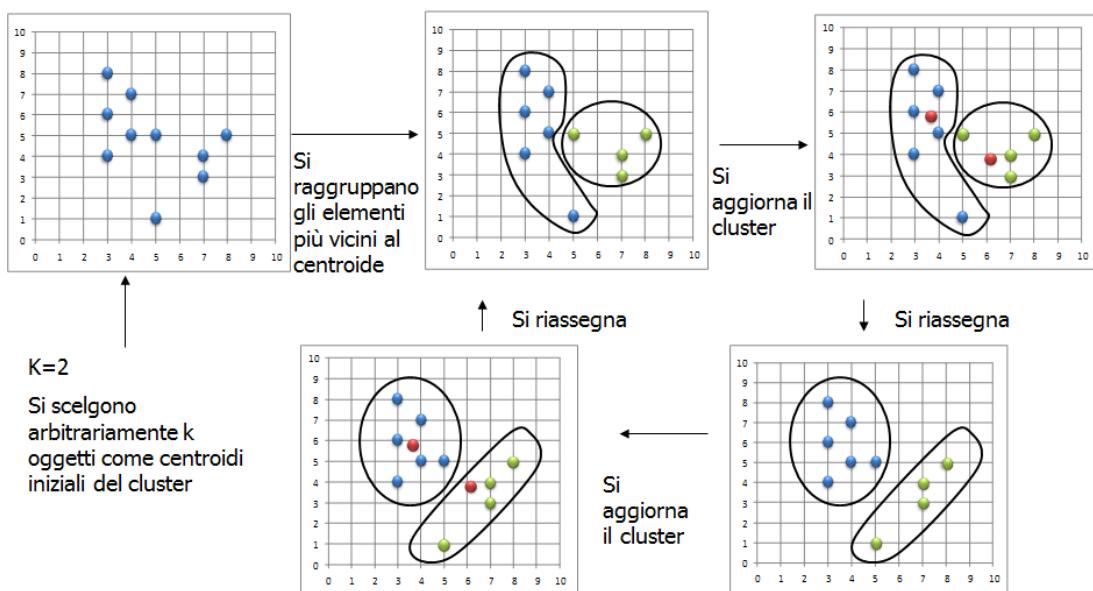
$$E = \sum_{i=1}^k \sum_{x \in c_i} d(x, vm)^2$$

dove E è la somma dell’errore quadratico per tutti gli oggetti del database, x è il punto nello spazio che rappresenta il dato oggetto ed vm è la media del cluster C_i (sia x che vm sono multidimensionali).

Questo criterio cerca di rendere i *cluster* risultanti quanto più compatti e separati possibili. Il metodo, tuttavia, può essere applicato soltanto quando è definita la media di un *cluster*. Questo può non accadere in molte applicazioni, ad esempio quando sono coinvolti dati con attributi categorici. La necessità per gli utenti di specificare all’inizio il numero dei *cluster*, può essere vista come uno svantaggio. *K-MEANS* non è adatto per scoprire

Algoritmo 1 K-Means

- 1: Scegli k punti come centri $\{c_j\}$ dei cluster
- 2: **repeat** {
- 3: Assegna ciascun pattern x_i al cluster P_j che abbia il centro c_j più vicino ad x_i (ovvero quello che minimizza $d(x_i, c_j)$)
- 4: Ricalcolare i centri $\{c_j\}$ dei cluster utilizzando i punti che appartengono a ciascun cluster
- 5: } **until** (criterio di convergenza soddisfatto)

Figura 1.22: Funzionamento di *K-MEANS*

cluster con forme non convesse o di dimensioni molto differenti. Inoltre, è sensibile al rumore e agli *outlier* dal momento che un piccolo numero di questi dati possono influenzare sostanzialmente il valore medio. La Figura § 1.22 mostra un esempio di funzionamento dell'algoritmo.

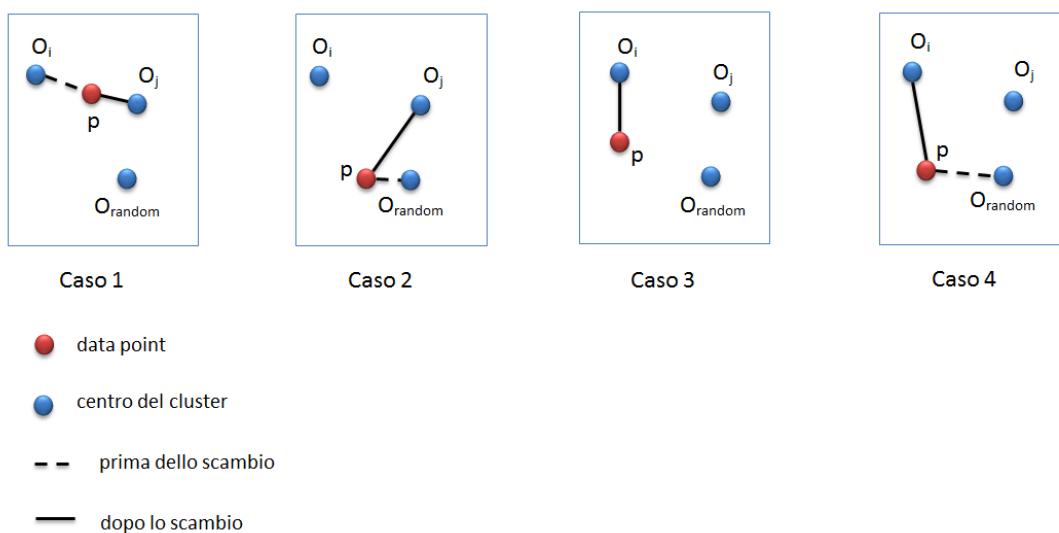
Vi sono alcune varianti del metodo *K-MEANS*. Esse possono differire nella selezione delle k medie iniziali, nel calcolo della similarità e nelle strategie per il calcolo delle medie del *cluster*. Una strategia interessante, che spesso restituisce buoni risultati, consiste nell'applicare prima un algoritmo di agglomerazione gerarchica per determinare il numero di *cluster* e per trovare una classificazione iniziale, e nell'utilizzare successivamente la rilocazione iterativa per migliorare la classificazione.

1.5.3.2 K-MEDOIDS

Come precedentemente descritto, l'algoritmo *K-MEANS* risulta sensibile agli *outlier*; per ovviare a tale inconveniente invece di prendere come punto di riferimento il valore medio degli oggetti in un *cluster*, può essere utilizzato il medoide, ovvero l'oggetto localizzato più centralmente in un *cluster*. In questo modo è possibile comunque eseguire il metodo di partizionamento cercando di minimizzare la somma delle dissimilarità tra ciascun oggetto e il suo punto di riferimento corrispondente.

La strategia di base dell'algoritmo *K-MEDOIDS* è quella di raggruppare n oggetti in k *cluster*. Innanzitutto esso trova arbitrariamente un oggetto rappresentativo per ciascun *cluster*. Ciascun oggetto rimanente viene inserito nel *cluster* associato al medoide più simile. La strategia sostituisce iterativamente medoidi con non medoidi e termina quando la qualità del *clustering* risultante non può più essere migliorata. Questa qualità viene stimata utilizzando una *funzione di costo* che misura la dissimilarità media tra un oggetto e il medoide del suo *cluster*. Durante la determinazione di un oggetto non medoide, o_{random} , come buon sostituto di un medoide corrente, o_j , è possibile imbattersi nei seguenti quattro casi per ciascun non medoide p :

Figura 1.23: I quattro possibili casi della funzione di costo per il metodo *K-MEDOIDS*



- caso 1: p appartiene al medoide o_j . Se o_j viene sostituito da o_{random} come medoide e p è più vicino ad uno degli o_i , $i \neq j$, allora p viene riassegnato ad o_i .

- caso 2: p appartiene a o_j . Se o_j è sostituito da o_{random} come medoide e p è più vicino ad o_{random} , allora p viene assegnato ad o_{random} .
- caso 3: p appartiene al medoide o_i , $i \neq j$. Se o_j è sostituito da o_{random} come medoide e p è ancora più vicino ad o_i , allora l'assegnamento non comporta cambiamenti per p .
- caso 4: p appartiene a o_i , $i \neq j$. Se o_j è sostituito da o_{random} come medoide e p è più vicino ad o_{random} , allora p viene riassegnato ad o_{random} .

Ogni volta che viene effettuato un riassegnamento, si presenta una differenza nella funzione di costo che può essere positiva o negativa. Definiamo *costo totale di swapping* la somma di tutte le differenze nelle funzioni di costo che si ottengono in seguito al riassegnamento degli oggetti non medoidi. Se il costo totale di *swapping* risulta negativo allora o_j viene sostituito con o_{random} dal momento che l'errore quadratico complessivo sarebbe ridotto. Se il costo totale di *swapping* risulta positivo, il medoide corrente o_j è considerato accettabile e non viene sostituito.

Il metodo *K-MEDOIDS* risulta essere più robusto di *K-MEANS* in presenza di rumore, anche se l'elaborazione è più costosa. Entrambi i metodi richiedono all'utente di specificare k , il numero di *cluster* finali.

La più comune realizzazione dell'algoritmo *K-MEDOIDS* è il *Partitioning Around Medoids* (*PAM* descritto in), che lavora in modo efficiente su piccoli insiemi di dati ma presenta lo svantaggio di non essere molto scalabile. L'Algoritmo § 2 descrive il funzionamento di *PAM*.

Algoritmo 2 PAM

```
1: seleziona  $k$  degli  $n$  punti di dati come medoidi
2: repeat {
3:   associa ogni punto al più vicino medoide
4:   foreach ( medoide  $m$  ) {
5:     foreach ( non medoide  $o$  ) {
6:       scambia  $m$  con  $o$  e calcola il costo totale della configurazione
7:     }
8:   }
9:   seleziona la configurazione con il minor costo
10: } until ( i medoidi non cambiano )
```

Per l'individuazione del punto più vicino può essere usata una qualsiasi misura di distanza, quale la misura di distanza Euclidea, la distanza di *Manhattan* o la distanza di *Minkowski*.

1.5.3.3 CLARANS

Al fine di trattare insiemi di dati più grandi così da risolvere il problema della scalabilità, è possibile usare un metodo basato sul campionamento, denominato *CLARA* (*Clustering LARge Applications* descritto in [KR90]). L'algoritmo *CLARA* invece di prendere in considerazione l'intero insieme dei dati, sceglie un sottoinsieme di dimensione fissa dei dati effettivi supponendo che essa sia rappresentativa di tutti i dati. I medoidi vengono, quindi, scelti da questo campione usando *PAM*. Se i campioni vengono selezionati in maniera piuttosto randomica, si dovrebbe riuscire a rappresentare abbastanza fedelmente l'intero insieme dei dati originario e i medoidi rappresentativi individuati dovrebbero essere simili a quelli che si sarebbero costruiti utilizzando l'intero insieme dei dati. Tali operazioni vengono ripetute per ogni sottoinsieme individuato.

Per migliorare la qualità e la scalabilità di *CLARA* è stato proposto un algoritmo di tipo *K-MEDOIDS* denominato *CLARANS* (*Clustering Large Applications based upon RANdomized Search*). Concettualmente, il processo di *clustering* può essere visto come una ricerca in un grafo dove ciascun nodo è una soluzione potenziale (un insieme di k medoidi). Due nodi sono vicini (ovvero, *connessi* da un arco nel grafo) se i loro insiemi differiscono soltanto per un oggetto. A ciascun nodo può essere assegnato un costo definito dalla dissimilarità totale tra ciascun oggetto e il medoide del suo *cluster*. A ciascun passo viene applicato l'algoritmo *PAM* per esaminare tutti i vicini del nodo corrente nella ricerca della soluzione con costo minimo. Il nodo corrente viene sostituito dal vicino che presenta costo minimo.

Mentre *CLARA* considera un campione di nodi all'inizio di una ricerca, *CLARANS* considera dinamicamente un campione random di vicini ad ogni passo della ricerca. Il numero di vicini da campionare casualmente è ristretto da un parametro selezionato dall'utente. In questo modo *CLARANS* non confina la ricerca ad un'area localizzata. Se viene trovato un vicino migliore (ovvero un vicino che ha un costo minore), *CLARANS* si sposta al nodo del vicino e il processo ricomincia; in caso contrario il *cluster* corrente produce un minimo locale. Se viene trovato un minimo locale, *CLARANS* ricomincia con nuovi nodi selezionati

in maniera *random*, sempre al fine di cercare un nuovo minimo locale. Una volta che un numero di minimi locali specificato dall’utente è stato individuato, *CLARANS* restituisce il migliore tra i minimi locali.

Tuttavia l’algoritmo *CLARANS* può richiedere diverse scansioni del database, il costo di esecuzione delle quali diventa proibitivo nel caso di grandi data set. Per di più c’è la possibilità che la soluzione fornita sia in realtà un ottimo locale, problema che affligge d’altra parte anche gli altri algoritmi di clustering partizionali.

1.5.3.4 DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*)

La maggior parte degli algoritmi di clustering partizionali e non solo, forniscono risultati soddisfacenti solo quando i cluster sono convessi, ipersferici e di dimensioni uniformi. In caso contrario, il clustering prodotto non è in generale di buona qualità. Per risolvere il problema è stato proposto l’algoritmo basato sulla densità *DBSCAN*. L’algoritmo costruisce ciascun cluster mettendo insieme regioni con densità sufficientemente alta; esso è in grado di individuare cluster di forma arbitraria e fornisce risultati interessanti anche in presenza di rumore. In *DBSCAN* un cluster viene definito come un insieme massimale di punti *density-connected*. Per capire il funzionamento del *clustering* basato sulla densità sono richieste alcune definizioni, che di seguito introduciamo.

Definizione 1. *L’intorno di raggio ε di un determinato punto è chiamato ε -neighborhood del punto.*

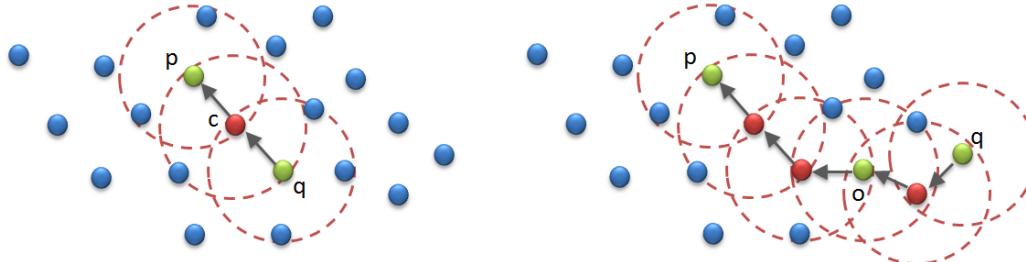
Definizione 2. *MinPts è il numerominimo di punti nell’intorno di un determinato punto.*

Definizione 3. *Un punto p è denominato core object se nel suo ε -neighborhood contiene almeno un numero minimo, MinPts, di punti.*

Definizione 4. *Dato un insieme di punti D diciamo che un punto p è direttamente density-reachable dal punto c , rispetto ad ε e MinPts, se p è all’interno dell’ ε -neighborhood di c e c è un core object.*

Definizione 5. *Un punto p è density-reachable dal punto q rispetto ad ε e MinPts in un insieme di punti D se vi è una catena di punti p_1, \dots, p_n , $p_1 = q$ e $p_n = p$ tale che p_{i+1} è direttamente density-reachable da p_i rispetto ad ε e MinPts, per $1 \leq i \leq n$, $p_i \in D$.*

Figura 1.24: *Density-reachable* e *Density-connected*

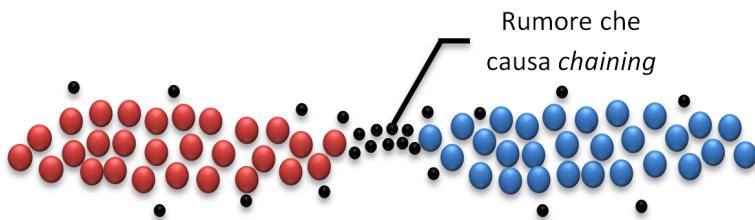


Definizione 6. Un punto p è *density-connected* ad un punto q rispetto ad ε e $MinPts$ in un insieme di punti D se esiste un punto $o \in D$ tale che sia p che q sono *density-reachable* da o rispetto ad ε e $MinPts$.

Un *cluster* basato sulla densità è un insieme di oggetti *density-connected* che risulta essere massimale rispetto alla *density-reachability*. Se un punto non è contenuto in nessun cluster viene considerato rumore. *DBSCAN* costruisce i cluster determinando l' ε -neighborhood di ciascun punto nel database. Se l' ε -neighborhood di un punto p contiene più di $MinPts$, viene creato un nuovo cluster con p come *core object*. Successivamente vengono individuati gli oggetti *density-reachable* dai vari *core object*; se due *core object* sono direttamente *density-reachable*, i cluster corrispondenti vengono fusi. Il processo termina quando nessun nuovo punto può essere aggiunto ad un cluster. Se viene utilizzato un indice spaziale la complessità computazionale di *DBSCAN* è $O(n \log n)$, dove n è il numero di oggetti del *database*; in caso contrario tale complessità diventa $O(n^2)$. Con una definizione appropriata dei parametri ε e $MinPts$ definiti dall'utente, l'algoritmo è efficace nel trovare *cluster* di forma arbitraria. Per approfondimenti si consulti [EKSX96].

Nonostante *DBSCAN* riesca a identificare *cluster* di forma arbitraria, esso presenta i seguenti svantaggi:

- *DBSCAN* è sensibile in modo considerevole nei confronti dei parametri ε e $MinPts$, che a loro volta sono difficili da determinare.
- *DBSCAN* può fondere erroneamente due *cluster* che siano congiunti da una stretta ma densa linea di punti (fenomeno del *chaining*).

Figura 1.25: Fenomeno del *chaining*

- DBSCAN introduce pesanti costi di elaborazione di *I/O*, in quanto esso non effettua nessuna forma di *pre-clustering*, lavorando pertanto sull'intero *database*. Questo non lo rende adatto ad elaborazioni su grandi volumi di dati.
- DBSCAN non è di tipo incrementale.

1.5.4 Metodi gerarchici

In questa sezione analizzeremo in maniera più approfondita i metodi gerarchici e gli algoritmi più noti che rientrano in tale tipologia.

Le tecniche di *clustering* gerarchico si possono ulteriormente suddividere in:

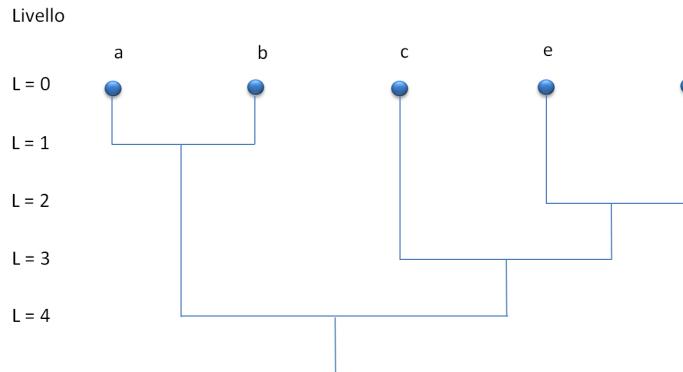
1. **metodi gerarchici agglomerativi** (approccio “*bottom up*”), in cui ciascun oggetto forma inizialmente un gruppo. Successivamente gli oggetti o i gruppi vicini vengono fusi, fino a quando non si ottiene un unico gruppo (il livello più in alto della gerarchia), oppure fino a quando non si verifica una condizione di terminazione.
2. **metodi gerarchici divisivi** (approccio “*top down*”): inizia con tutti gli oggetti posti nello stesso *cluster*. Durante ciascuna iterazione successiva, un *cluster* viene suddiviso in *sotto-cluster* più piccoli, fino a quando ciascun oggetto si trova in un *cluster* differente o fino a quando non si verifica una determinata condizione di terminazione.

La condizione di terminazione può essere, ad esempio, il numero di *cluster* da raggiungere. Per rappresentare il *clustering* gerarchico viene comunemente utilizzata una struttura ad albero, denominata *dendrogramma*: esso mostra come i *cluster* vengono raggruppati passo

dopo passo. Generalmente sull’asse delle ascisse viene posizionata la distanza logica dei *cluster* secondo la metrica definita, mentre nell’asse delle ordinate il livello gerarchico di aggregazione (valori interi positivi). La scelta del livello gerarchico (valore dell’asse *Y*) definisce la partizione rappresentativa del processo di aggregazione.

Mostriamo di seguito due esempi di dendrogramma. La Figura § 1.26 rappresenta un dendrogramma relativo ad un *clustering* gerarchico agglomerativo. Al livello $L = 0$ vengono mostrati cinque cluster costituiti da un singolo elemento. Al livello 1 gli elementi $\{a\}$ e $\{b\}$ vengono raggruppati insieme per formare il primo cluster; essi rimarranno insieme per tutti i livelli successivi. Al livello 2 vengono fusi gli elementi $\{e\}$ ed $\{f\}$; nel livello 3 vengono fusi gli elementi $\{e, f\}$ con $\{c\}$. Nell’ultimo livello vengono fusi i restanti insiemini $\{a, b\}$ e $\{c, e, f\}$. La Figura § 1.27 nella pagina successiva mostra, invece, un esempio di dendrogramma relativo ad un *clustering* gerarchico divisivo.

Figura 1.26: Dendrogramma Agglomerativo

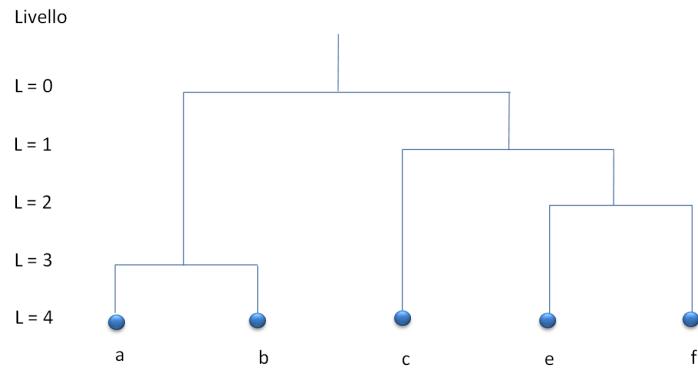
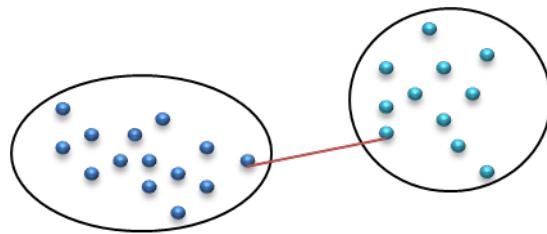


In entrambe le tipologie di *clustering* gerarchico sono necessarie funzioni per selezionare la coppia di *cluster* da fondere o dividere (“agglomerativo” o “divisivo”). Nel primo caso, sono necessarie funzioni che misurino la similarità tra due *cluster*, in modo da fondere quelli più simili. Le funzioni più utilizzate nel *clustering* agglomerativo sono:

- *single-link proximity*, calcola la distanza tra i due *cluster* come la *distanza minima* tra elementi appartenenti a *cluster* diversi:

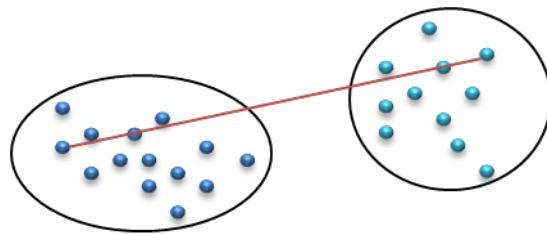
$$sim(c_i, c_j) = \min_{x \in c_i, y \in c_j} d(x, y)$$

Figura 1.27: Dendrogramma Divisivo

Figura 1.28: *Single-link*

- *complete-link proximity*, calcola la distanza tra i due *cluster* come la distanza massima tra elementi appartenenti ai due *cluster*:

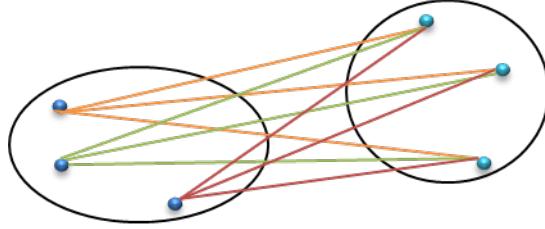
$$\text{sim}(c_i, c_j) = \max_{x \in c_i, y \in c_j} d(x, y)$$

Figura 1.29: *Complete-link*

- *average-link proximity*, calcola la distanza tra i due *cluster* come la media delle distanze tra i singoli elementi:

$$sim(c_i, c_j) = \frac{1}{|c_i||c_j|} \sum_{x \in c_i} \sum_{y \in c_j} d(x, y)$$

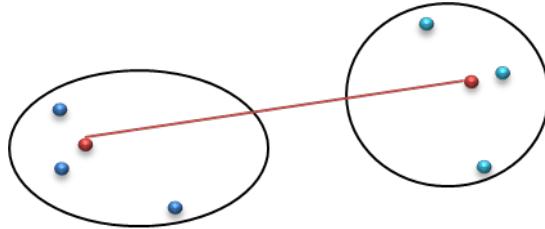
Figura 1.30: *Average-link*



- *distanza tra i centroidi*, calcola la distanza tra i due *cluster* come la distanza tra i centroidi:

$$sim(c_i, c_j) = d(\bar{x}, \bar{y})$$

Figura 1.31: Distanza tra i centroidi



Nel *clustering* divisivo, invece, è necessario individuare il *cluster* da suddividere in due sottogruppi. Per tale ragione sono necessarie funzioni che misurino la compattezza del *cluster*, la densità o la sparsità dei punti assegnati ad un *cluster*. Le funzioni normalmente utilizzate in questo metodo sono:

- *average-internal proximity*, valuta la similarità media tra i punti interni ad un *cluster*: più sono tra loro dissimili (bassi valori di similarità), più il *cluster* è da suddividere in sottogruppi:

$$sim(c_i) = \frac{1}{|c_i|(|c_i| - 1)} \sum_{x, y \in c_i, x \neq y} d(x, y)$$

- *maximum internal proximity*, valuta la distanza massima tra due punti interni ad un *cluster*. Tale valore è noto anche come “*diametro del cluster*”: più tale valore è basso, più il *cluster* è compatto:

$$\text{sim}(c_i) = \max_{x,y \in c_i} d(x, y)$$

Il metodo di *clustering* gerarchico, sebbene semplice, incontra spesso difficoltà relative alla selezione dei punti di *merge* o di *split*. Tale decisione è critica perché una volta che un gruppo di oggetti viene fuso o suddiviso, il processo al passo successivo opererà sui nuovi cluster. Esso non annulla mai quello che è stato fatto né effettua uno scambio di oggetti tra *cluster*. Pertanto, le decisioni di *merge* o di *split*, se vengono prese in modo sbagliato in qualche passo, possono portare a cluster di bassa qualità. Inoltre il *metodo non è molto scalabile* dal momento che ciascuna decisione sul *merge* o sullo *split* richiede l'esame e la valutazione di un buon numero di oggetti o di *cluster*. Una direzione promettente per migliorare la qualità del *clustering* dei metodi gerarchici è di integrare il *clustering* gerarchico con altre tecniche di *clustering*. Questo è quello che fanno alcuni metodi molto noti quale il *BIRCH*.

Di seguito verranno mostrati gli algoritmi di *clustering* gerarchico ritenuti potenzialmente integrabili con le tecnologie *OLAP*.

1.5.4.1 CURE (*Clustering Using REpresentative*)

CURE è un algoritmo di *clustering* che implementa un approccio intermedio tra l'approccio *centroid-based*, che prevede un solo punto rappresentativo per ogni *cluster* (il centroide), e l'approccio *all-points*, che considera tutti i punti del *cluster* come punti rappresentativi. L'algoritmo *CURE* prevede, infatti una preliminare scelta di un numero costante di c punti ben sparsi (*well scattered*) all'interno di ogni *cluster*. I punti sparsi hanno lo scopo di “catturare” la forma e la dimensione dei *cluster*. Tali punti vengono poi contratti verso il centroide di una frazione α (Figure § 1.32 nella pagina seguente e § 1.33 nella pagina successiva). I punti sparsi, dopo aver subito la contrazione, sono usati come punti rappresentativi del *cluster*. A ogni passo dell'algoritmo vengono fusi i gruppi che possiedono la coppia di punti rappresentativi più vicini.

Figura 1.32: Contrazione dei punti rappresentativi di un *cluster* ($\alpha = 0.3$)

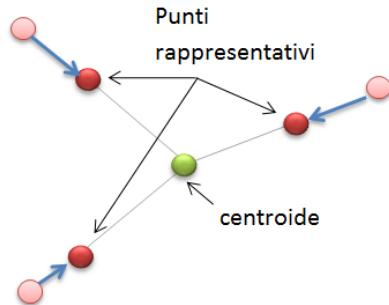
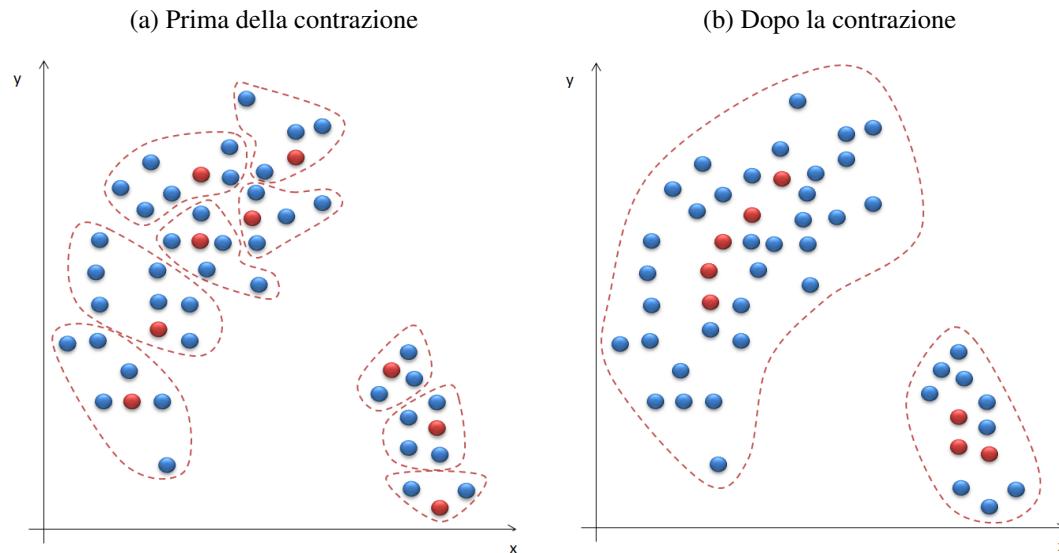


Figura 1.33: Contrazione dei punti rappresentativi di due *cluster*



Grazie all’approccio basato sui punti ben sparsi, *CURE* è meno sensibile alla presenza di punti singolari (*outlier*), in quanto la contrazione degli *outlier* verso il punto medio del *cluster* attenua il loro effetto negativo: questi sono, infatti, tipicamente più lontani dal centroide degli altri punti e risultano quindi spostati di una quantità maggiore a causa della contrazione. L’utilizzo di più punti rappresentativi consente, inoltre di identificare correttamente *cluster* di forma allungata, cosa che, invece, non è possibile se si utilizza il centroide come unico punto rappresentativo, dato che si tende a privilegiare il rilevamento

di *cluster* a geometria ipersferica. Permette, inoltre, di processare grandi volumi di dati, poiché in tal caso prevede una forma di pre-processazione dei dati.

Il *CURE*, come il *K-MEANS*, costringe l'utilizzatore a specificare il numero dei k *cluster* in cui raggruppare il *dataset*, venendo meno, quindi, all'obiettivo per il quale un utilizzatore è spinto all'uso dei metodi ad apprendimento non supervisionato; inoltre, non essendo un algoritmo incrementale non si ritiene da considerare per i fini della presente tesi, che ricordiamo essere l'integrazione di metodi di *clustering* con le tecnologie *OLAP*.

Per una trattazione più approfondita del *CURE*, si rimanda a [GRS98].

1.5.4.2 *BIRCH* (*Balanced Iterative Reducing and Clustering using Hierarchies*)

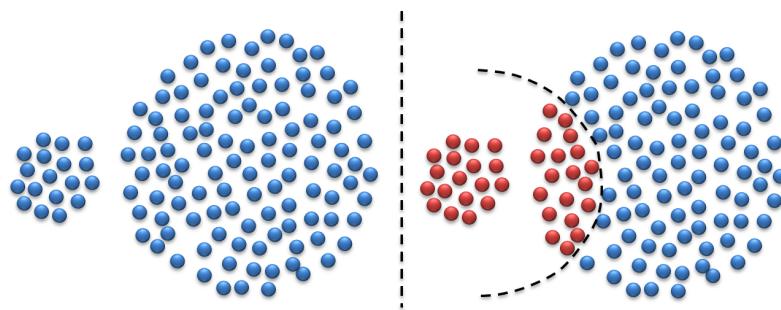
BIRCH è un algoritmo di *clustering* gerarchico agglomerativo, efficiente anche per *database* di grandi dimensioni. I vantaggi che presenta questo algoritmo sono diversi:

- essendo gerarchico consente di integrare lo stesso con le tecnologie di *DW*;
- ha la capacità di essere incrementale, il che, come abbiamo già avuto modo di dire, gli consente di interagire facilmente con tecnologie *OLAP*;
- è in grado di fornire un *clustering* di buona qualità con una singola scansione dei dati, consentendo così di minimizzare i tempi di *I/O*;
- è stato riconosciuto come il “*primo algoritmo di clustering proposto nell'ambito dei database che gestisce il 'rumore' (outlier) in maniera efficace*” ([ZRL96]).
- rispetta i vincoli sulla quantità di memoria disponibile (che è tipicamente molto inferiore alla dimensione dei dati);
- utilizza, con vantaggio, la programmazione parallela e concorrente;
- è altamente scalabile, ha infatti, un costo computazionale proporzionale alla dimensione del *database* stesso
- regola le *performance*, basandosi dinamicamente e interattivamente sulla conoscenza del *database*, ottenuta durante l'esecuzione dell'algoritmo.

Come tutti gli algoritmi, anche questo, non è esente da svantaggi. *BIRCH*, infatti, considera solo valori numerici ed è molto sensibile all'ordine di *input* dei dati. Inoltre è un algoritmo

di clustering *centroid-based*, ovvero, un punto viene assegnato ad un *cluster* in base alla vicinanza con il suo centroide. L'utilizzo di un solo punto rappresentativo (centroide) per i *cluster*, costituisce un problema nel caso in cui i *cluster* abbiano dimensioni e forme non omogenee, come mostrato in Figura § 1.21 a pagina 44. In questa situazione, infatti, si può manifestare l'assegnamento errato di un certo numero di punti. Non solo, può anche accadere che i punti di un grosso cluster vengano erroneamente assegnati ad una classe molto più piccolo, perché il suo centroide è più vicino (Figura § 1.34).

Figura 1.34: Errato assegnamento dei punti dei *cluster* in *BIRCH*



I vantaggi elencati sono particolarmente rilevanti per un algoritmo di *clustering* che debba interagire con le tecnologie *OLAP*. Per questi motivi, si è scelto di approfondire lo studio del *BIRCH* e, allo stesso tempo, di riprogettarlo e implementarlo, al fine di superare gli svantaggi in esso presenti e ottenere una performante integrazione con le tecnologie di *DW* (Capitolo § 2 a fronte).

Capitolo 2

L'algoritmo *BIRCH*

*e come abete in alto si digrada
di ramo in ramo, così quello in giuso,
cred'io, perché persona su non vada.*

PURGATORIO, CANTO XXII

NEl presente capitolo formiamo una descrizione dell'algoritmo della cui implementazione ci siamo occupati nel presente lavoro di tesi: l'algoritmo di *clustering* gerarchico agglomerativo *BIRCH* (*Balanced Iterative Reducing and Clustering using Hierarchies*). L'esposizione che proponiamo fa ampio riferimento all'articolo [ZRL95], redatto dagli inventori stessi dell'algoritmo.

2.1 Definizioni degli Attributi Metrici

L'obiettivo principale di questa sezione è quello di esporre tutte le misure usate nell'algoritmo di *clustering* *BIRCH*. Generalmente, da un *dataset*, è possibile estrarre due tipi possibili di attributi: *metrici* e *non metrici*. *Tout court*, un attributo è *metrico*, quando il suo valore può essere rappresentato o mappato attraverso esplicite coordinate in uno spazio vettoriale *Euclideo*; al contrario, un attributo è *non metrico*, quando il suo valore non può essere né rappresentato né mappato mediante esplicite coordinate in uno spazio vettoriale

Euclideo. *BIRCH* considera solo attributi metrici, come anche il *K-MEANS*. Ciò significa che ogni tupla di d attributi può essere rappresentata tramite un vettore d -dimensionale.

2.1.1 Proprietà di un *cluster*

Dati N punti d -dimensionali appartenenti ad un *cluster* $C = \{\vec{X}_i\}$, dove $i = 1, 2, \dots, N$, si forniscono le seguenti definizioni.

Definizione 7. *Centroide del cluster*

Si definisce centroide del *cluster* C il seguente vettore:

$$\vec{X}_0 = \frac{\sum_{i=1}^N \vec{X}_i}{N} \quad (2.1)$$

Esso costituisce, in pratica, il punto medio del *cluster*.

Definizione 8. *Raggio del cluster*

Il raggio del *cluster* C è definito come la distanza media dei punti appartenenti a C dal centroide:

$$R = \sqrt{\frac{\sum_{i=1}^N (\vec{X}_i - \vec{X}_0)^2}{N}} \quad (2.2)$$

Definizione 9. *Diametro del cluster*

Il diametro del *cluster* C è definito come la distanza media tra tutte le coppie di punti appartenenti a C :

$$D = \sqrt{\frac{\sum_{i=1}^N \sum_{j=1}^N (\vec{X}_i - \vec{X}_j)^2}{N(N-1)}} \quad (2.3)$$

Il diametro e il raggio del *cluster* costituiscono due misure alternative di dispersione intorno al centroide per un *cluster*.

2.1.2 Distanza tra due *cluster*

Dati N_1 punti d -dimensionali di un *cluster* $C_1 = \{\vec{X}_i\}$ ove $i = 1, 2, \dots, N_1$, con centroide \vec{X}_{0_1} e N_2 punti d -dimensionali di un *cluster* $C_2 = \{\vec{X}_j\}$ ove $j = N_1 + 1, N_1 + 2, \dots, N_1 + N_2$, con centroide \vec{X}_{0_2} , si definiscono cinque misure di distanza alternative per valutare la vicinanza tra C_1 e C_2 .

Definizione 10. *Distanza Euclidea tra centroidi (centroid Euclidian distance)* D_0 :

$$D_0 = \sqrt{(\vec{X}_{0_1} - \vec{X}_{0_2})^2} \quad (2.4)$$

Definizione 11. *Distanza Manhattan tra centroidi (centroid Manhattan distance)* D_1 :

$$D_1 = \left| \vec{X}_{0_1} - \vec{X}_{0_2} \right| = \sum_{i=1}^d \left| \vec{X}_{0_1}^{(i)} - \vec{X}_{0_2}^{(i)} \right| \quad (2.5)$$

Definizione 12. *Distanza media inter-cluster* D_2 (*average inter-cluster distance*) tra C_1 e C_2 :

$$D_2 = \sqrt{\frac{\sum_{i=1}^{N_1} \sum_{j=1}^{N_1+N_2} (\vec{X}_i - \vec{X}_j)^2}{(N_1 + N_2 - 1)(N_1 + N_2)}} \quad (2.6)$$

Definizione 13. *Distanza media intra-cluster* D_3 (*average intra-cluster distance*) tra C_1 e C_2 :

$$D_3 = \sqrt{\frac{\sum_{i=1}^{N_1} \sum_{j=N_1+1}^{N_1+N_2} (\vec{X}_i - \vec{X}_j)^2}{N_1 + N_2}} \quad (2.7)$$

Essa è in sostanza il diametro D del *cluster* ottenuto dalla fusione di C_1 e C_2 .

Definizione 14. *Distanza ad incremento di varianza* D_4 (*variance increase distance*) tra C_1 e C_2 :

$$D_4 = \sum_{k=1}^{N_1+N_2} \left(\vec{X}_k - \frac{\sum_{l=1}^{N_1+N_2} \vec{X}_l}{N_1 + N_2} \right)^2 - \sum_{i=1}^{N_1} \left(\vec{X}_i - \frac{\sum_{l=1}^{N_1} \vec{X}_l}{N_1} \right)^2 - \sum_{j=N_1+1}^{N_1+N_2} \left(\vec{X}_j - \frac{\sum_{l=N_1+1}^{N_1+N_2} \vec{X}_l}{N_2} \right)^2 \quad (2.8)$$

2.1.3 Misure di qualità di un *cluster*

Si assume che N punti siano stati raggruppati in K *cluster*. Un *cluster* i , $i = 1, \dots, K$, contiene N_i punti, ha raggio R_i e diametro D_i . $\sum_{i=1}^K N_i = N$. Di seguito sono riportate le quattro differenti misure alternative, proposte dagli autori, per verificare la qualità dei *cluster*:

Definizione 15. *Media ponderata del quadrato del raggio di un cluster* Q_1 (o \vec{R}) (*weighted average cluster radius square*):

$$Q_1 = \frac{\sum_{i=1}^K N_i R_i^2}{\sum_{i=1}^K N_i} \quad (2.9)$$

Definizione 16. *Media ponderata del quadrato del diametro di un cluster* Q_2 (o \vec{D}) (*weighted average cluster diameter square*):

$$Q_2 = \frac{\sum_{i=1}^K N_i(N_i - 1) D_i^2}{\sum_{i=1}^K N_i(N_i - 1)} \quad (2.10)$$

Definizione 17. *Totale ponderato del quadrato del raggio di un cluster* Q_3 (*weighted total cluster radius square*):

$$Q_3 = \sum_{i=1}^K N_i R_i^2 \quad (2.11)$$

Definizione 18. *Totale ponderato del quadrato del diametro di un cluster* Q_4 (*weighted average cluster diameter square*):

$$Q_4 = \sum_{i=1}^K N_i(N_i - 1) D_i^2 \quad (2.12)$$

2.2 *Clustering Feature* e *CF-Tree*

L'algoritmo *BIRCH* sceglie di rappresentare un *dataset* come un insieme di *sotto-cluster*, per ridurre l'entità del problema. Per fare ciò sfrutta due nuovi concetti, che sono alla base del suo *clustering* di tipo incrementale:

- *Clustering feature (CF)*;
- *Albero dei clustering feature (CF-Tree)*.

Queste strutture hanno lo scopo di comprimere le rappresentazioni dei *cluster*, consentendo al metodo di classificazione di ottenere una maggiore velocità e scalabilità.

2.2.1 *Clustering Feature*

Definizione 19. Clustering Feature: *Un clustering feature, indicato con CF, è una tripletta che sintetizza le informazioni riguardanti un cluster.*

Si consideri dunque un *cluster* C , contenente N punti d-dimensionalni:

$$C = \{\vec{X}_i\} \quad \text{ove } i = 1, 2, \dots, N$$

il *CF* di C è definita da:

$$CF = (N, \vec{LS}, \vec{SS})$$

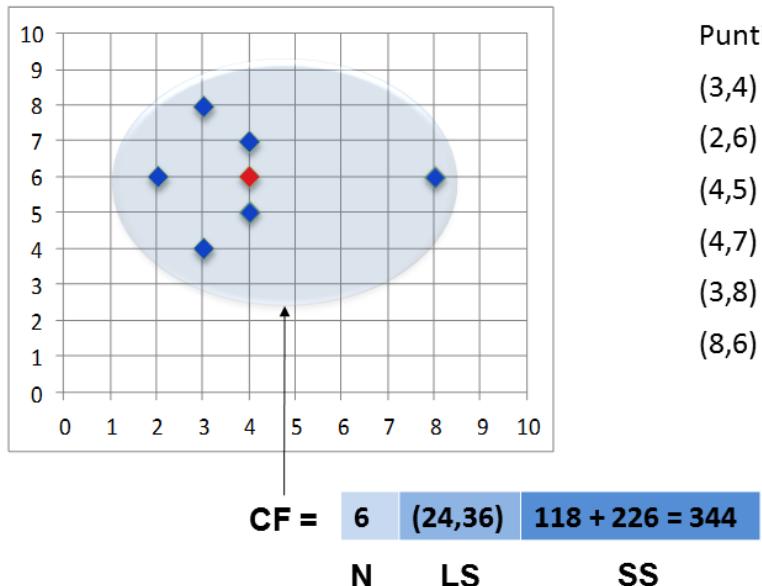
dove

- N è il numero di punti appartenenti al *cluster* C ;
- $\vec{LS} = \sum_{i=1}^N \vec{X}_i$ è la somma lineare degli N punti;
- $\vec{SS} = \sum_{i=1}^N \vec{X}_i^2$ è la somma quadratica degli N punti.

Questa tripletta non solo condensa la distribuzione dei punti all'interno di un *sotto-cluster*, ma permette anche di operare misure di vicinanza tra due *sotto-cluster* e rilevare la qualità generale del *clustering*.

Un semplice esempio di *CF* nel caso di punti bidimensionali è riportato in Figura § 2.1 nella pagina successiva.

Figura 2.1: Esempio di *CF* in uno spazio dei punti *bi*-dimensionale



Di seguito sono riportati i teoremi riguardanti le operazioni possibili attraverso le informazioni contenute in un *CF*.

Teorema 1. Teorema della Rappresentatività di un *CF*: Dati le *CF* entry dei cluster, tutte le misure definite nella Sezione § 2.1 a pagina 61 possono essere accuratamente calcolate.

E' facile provare che, avendo a disposizione il *CF* di un *cluster*, è possibile ricavare le grandezze \vec{X}_0 (2.1), R (2.2), D (2.3), D_0 (2.4), D_1 (2.5), D_2 (2.6), D_3 (2.7), D_4 (2.8) e in generale ogni altra grandezza statistica di interesse (le dimostrazioni sono presenti nell'Appendice § A a pagina 199 con alcune trasformazioni matematiche delle formule di definizione). Si può immaginare, quindi, un *CF*, come un *sotto-cluster*, cioè come un insieme di punti, anche se in realtà esso è una rappresentazione sintetica delle statistiche e delle informazioni associate a un *cluster*: contiene, infatti, i momenti statistici di ordine zero, uno e due. Questa proiezione dei dati è:

- efficiente, poiché non è necessario memorizzare tutti i punti contenuti in un *cluster*, occupando, così, molto meno spazio;

- sufficiente, perché contiene tutte e sole le informazioni necessarie per supportare l'elaborazione accurata di tutte le misure opportune per prendere decisioni nel sistema.

Di seguito è riportato il teorema di additività dei *CF*, che mostra come i *CF* dei *sotto-cluster* possano essere memorizzati e calcolati in modo incrementale, man mano che si fondono tra loro, o all'atto dell'inserimento di nuovi punti di dati.

Teorema 2. Additività delle CF: Si supponga che $CF_1 = (N_1, \vec{LS}_1, \vec{SS}_1)$ e $CF_2 = (N_2, \vec{LS}_2, \vec{SS}_2)$ siano le *CF* di due cluster disgiunti A e B. Allora il vettore CF_3 del cluster C, ottenuto dalla fusione dei cluster A e B, è dato da:

$$CF_3 = (N_1 + N_2, \vec{LS}_1 + \vec{LS}_2, \vec{SS}_1 + \vec{SS}_2) \quad (2.13)$$

La dimostrazione consiste di passaggi di algebra lineare e, quindi, viene omessa.

2.2.2 *CF-Tree*

Definizione 20. CF-Tree: Un *CF-Tree* è un albero bilanciato¹ che memorizza le *CF* durante il processo di clustering gerarchico. Ad un *CF-Tree* sono associati due parametri:

- Fattore di ramificazione o branching factor (*B* per i nodi non-foglia, *L* per i nodi foglia)
- Soglia o threshold (*T*)

Come tutti gli alberi, un *CF-Tree* ha nodi foglia e nodi non foglia.

Ogni nodo non foglia contiene al più *B* campi del tipo:

$$[CF_i, Figlio_i] \quad \text{con} \quad i = 1, 2, \dots, B$$

ove:

- *Figlio_i* è il puntatore al nodo figlio *i*-esimo;
- *CF_i* è il clustering feature del *sotto-cluster* rappresentato dal nodo figlio *i*-esimo.

Quindi un nodo non-foglia rappresenta un *cluster* costituito da tutti i *sotto-cluster* in esso presenti, rappresentati dai suoi campi (Figura § 2.2 a fronte).

Un nodo foglia contiene al più *L* campi del tipo:

$$[CF_i] \quad \text{con} \quad i = 1, 2, \dots, L$$

e due puntatori, “*prev*” e “*next*”, usati per concatenare tra loro tutti i nodi foglia, in modo da rendere più efficiente la scansione. Anche un nodo foglia rappresenta un *cluster* composto dai *sotto-cluster*, ma ciascuno di quest’ultimi deve soddisfare il requisito di soglia (*threshold requirement*): il diametro (o talvolta il raggio) dei *sotto-cluster* che essi rappresentano, deve essere minore del parametro *T*. In Figura § 2.2 nella pagina successiva, è possibile osservare la struttura dei nodi foglia.

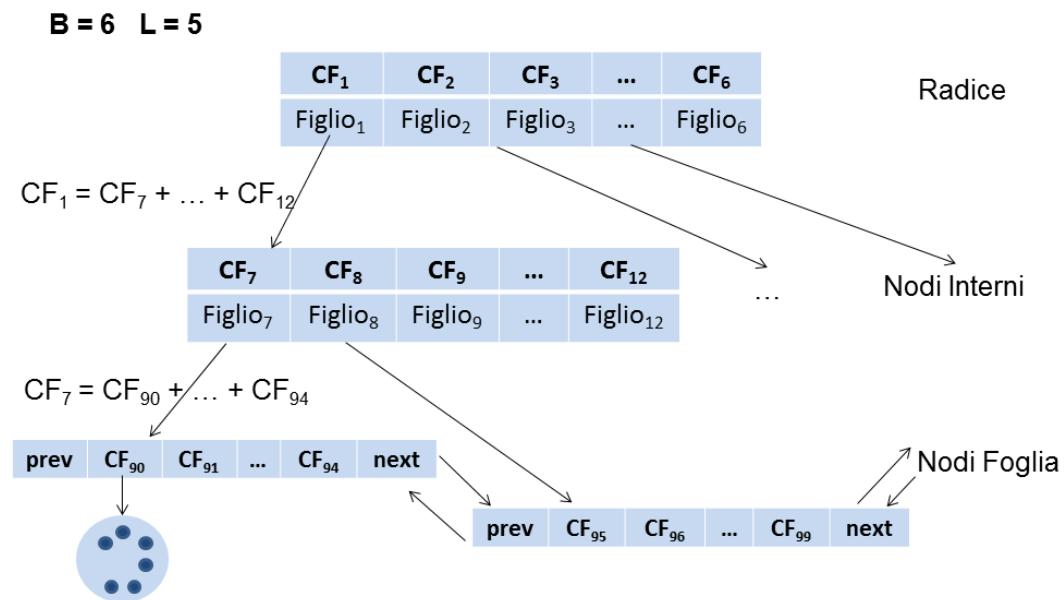
Con la definizione di *CF-Tree* di cui sopra, è facile intuire che la dimensione dell’albero, varia in funzione del valore di soglia *T*. Al crescere del valore in questione, la dimensione dell’albero dovrebbe diminuire. Per ragioni di efficienza, quindi, si richiede

¹La distanza dalle foglie alla radice (*profondità o height*) è costante

che un nodo (foglia o non-foglia) si adatti a una pagina di dimensione P . Una volta data la dimensione d dello spazio vettoriale, sono note sia le dimensioni dei nodi foglia, sia dei nodi non foglia, poiché B e L sono determinati dalla dimensione della pagina P . Dunque, per ottimizzare le prestazioni dell'algoritmo, è necessario modificare opportunamente il parametro P .

Il *CF-Tree* può essere costruito dinamicamente, inserendo via via nuovi punti di dati. È usato per guidare una nuova inserzione nel corretto *sotto-cluster*, nello stesso modo in cui un *B+ Tree* è usato per guidare una nuova inserzione nella corretta posizione, in questo caso allo scopo di ottenere un ordinamento efficiente. Il *CF-Tree* è una rappresentazione molto compatta del *dataset*, perché ogni campo in un nodo foglia non è un singolo punto, ma un *sotto-cluster* che contiene molti punti, con l'unico vincolo di avere diametro inferiore alla soglia T .

Figura 2.2: Esempio di *CF-Tree*

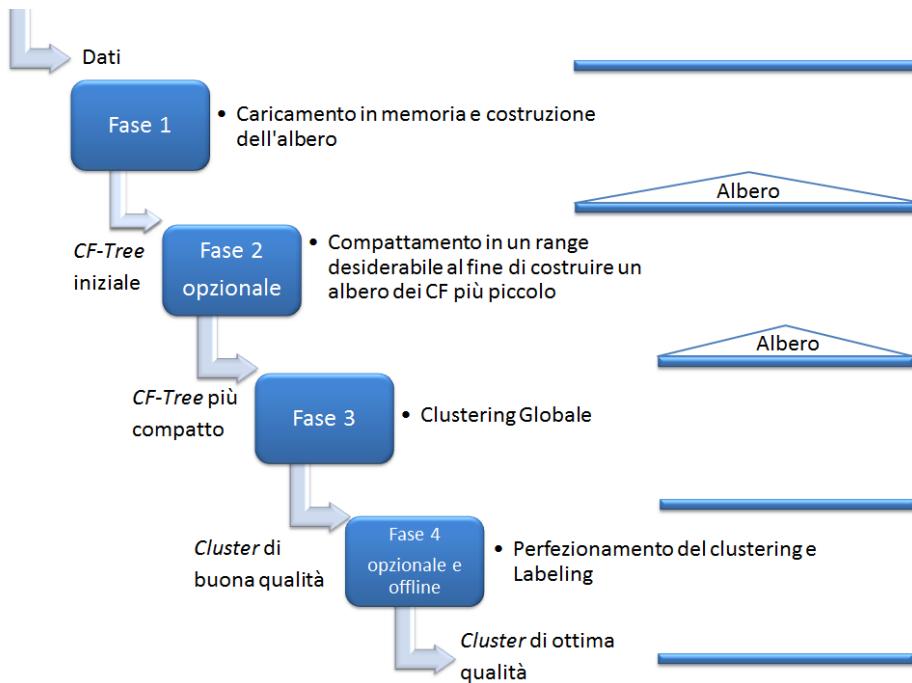


2.3 Funzionamento dell'algoritmo di *clustering BIRCH*

La Figura § 2.3, fornisce una rappresentazione schematica dell'algoritmo *BIRCH*. In questa sezione, si descriverà il ruolo di ogni fase che compone l'intero algoritmo in esame, e le relazioni tra le stesse.

L'algoritmo di *clustering BIRCH* consiste di quattro fasi: *Creazione* dell'albero, *Compattamento* (opzionale) dell'albero, *Clustering Globale* e *Raffinamento* (opzionale) dei *cluster*.

Figura 2.3: Algoritmo di *BIRCH*



Il compito principale della Fase 1 è di scansionare i dati e di costruire un *CF-Tree* iniziale, occupando una quantità di memoria prestabilita e riciclando lo spazio su disco. Questo albero dei *CF* cerca di proiettare le informazioni del *dataset* nel modo migliore possibile, pur rispettando il vincolo sulla massima quantità di memoria utilizzabile.

Raggruppando i punti delle zone dense in *sotto-cluster* e rimuovendo i punti dispersi, considerati *outlier*, questa fase crea una sintesi compatta dei dati in memoria (i dettagli relativi alla Fase 1 saranno affrontati nella Sezione § 2.3.1 a pagina 73). Di conseguenza le computazioni che seguono la Fase 1 assumono le seguenti caratteristiche:

1. **velocità**, infatti:

- (a) non sono necessarie operazioni di *I/O*;
- (b) il problema di effettuare il *clustering* dei dati originari è ridotto al sotto-problema (di complessità molto minore) di effettuare il *clustering* dei *sotto-cluster* nei campi dei nodi foglia;

2. **accuratezza**, infatti:

- (a) vengono eliminati molti *outlier*;
- (b) i rimanenti dati sono rappresentati con la massima granularità consentita dalla quantità di memoria disponibile;

3. **robustezza rispetto all'ordinamento dei dati** (*less order sensitive*), infatti, i campi delle foglie del *CF-Tree* iniziale presentano un ordine di *input*, a livello locale migliore dell'ordine di *input* arbitrario dei dati originari.

Una volta raccolte tutte le informazioni necessarie nel *CF-Tree*, è possibile applicare, nella Fase 3, metodi di *clustering* globale o semi-globale per raggruppare tutte le *entry-leaf* presenti nei nodi foglia e così eliminare tutti i problemi legati alle anomalie dovute alla dimensione artificiale dei nodi. Come algoritmi di *clustering* globale, gli autori di [ZRL96] propongono l'*HC*, il *K-MEANS* e il *CLARANS*, che lavorano con punti che possono essere facilmente adattati per lavorare con un insieme di *sotto-cluster*, ognuno descritto attraverso la sua *CF entry*. Diversi sono i punti degni di nota per l'adattamento:

1. Si può usare qualsiasi algoritmo disponibile in letteratura;
2. Qualsiasi algoritmo potrebbe essere modificabile per utilizzare le informazioni delle *CF entry* presenti nel *CF-Tree*.
3. L'algoritmo di *clustering* globale o semi-globale, congiuntamente con l'alto livello di densità di punti presenti nelle *entry-leaf*, favorisce la diminuzione della sensibilità della qualità finale del *clustering* rispetto all'ordine iniziale dei dati.

I dettagli riguardanti l'adattamento degli algoritmi esistenti per lavorare con le *CF entry* saranno discussi nella Fase 3 (Sezione § 2.3.3 a pagina 87).

La Fase 2 è opzionale. Gli autori di [ZRL96] hanno osservato che i metodi di *clustering* globale o semi-globale utilizzabili nella Fase 3, lavorano efficientemente con quantità di dati in ingresso comprese in diversi *range* di valori, sia in termini di velocità che di qualità. Per esempio, se si sceglie di adattare il *CLARANS* nella Fase 3, è noto che questo algoritmo lavora piuttosto bene su un insieme minore di 5000 oggetti, poiché all'interno di questo *range*, la frequenza di scansione dei dati è ancora accettabile e la probabilità che i dati vengano collocati in *cluster* con bassa densità è bassa. È evidente che esiste un divario tra la dimensione dei risultati della Fase 1, e il *range* ottimale di dati di ingresso degli algoritmi della Fase 3. La Fase 2 serve dunque a colmare questo *gap*: durante la Fase 2, infatti, i campi delle foglie del *CF-Tree* iniziale vengono scansionate con lo scopo di costruire un albero più contenuto, rimuovendo nel frattempo altri punti isolati (*outlier*) e raggruppando *sotto-cluster* affollati in *cluster* più grandi.

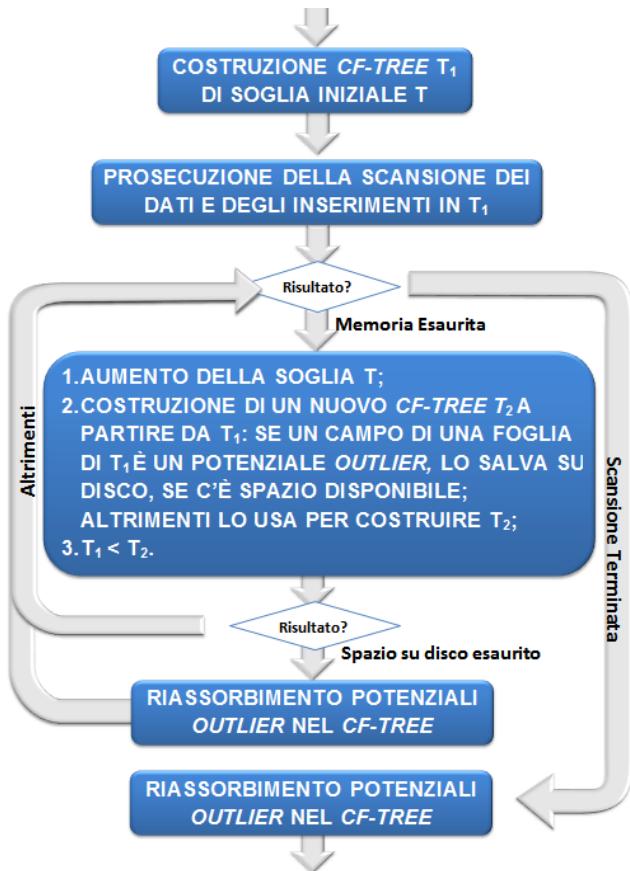
Gli effetti nefasti dovuti all'ordine sparso dei dati di ingresso, e allo *splitting* generato dalla dimensione della pagina di memoria, può renderci dubiosi riguardo alla veridicità degli effettivi *cluster* rilevati nei dati. A questo si pone rimedio nella Fase 3, usando un algoritmo di classificazione globale o semi-globale per classificare tutti i campi delle foglie. Si noti che gli algoritmi di *clustering* esistenti, possono essere adattati facilmente per lavorare con un *set* di *sotto-cluster*, ognuno dei quali descritto dai suoi vettori *CF*.

Dopo la Fase 3, si ottiene un insieme di *cluster* che rileva l'andamento delle distribuzioni principali nei dati. Tuttavia, potrebbero essere presenti, a livello locale, imprecisioni di lieve entità, dovute a (rari) errori di piazzamento, (Sezione § 2.3.1.1 a pagina 74) e al fatto che la Fase 3 non è applicata direttamente sui dati, bensì su informazioni di sintesi degli stessi. La Fase 4 è opzionale, e richiede scansioni aggiuntive dei dati per correggere le imprecisioni e migliorare ulteriormente la classificazione. È bene sottolineare che, fino alla Fase 4, i dati sono stati scanditi una sola volta (l'albero e l'informazione riguardante gli *outlier* può essere stata, invece, scandita molte volte).

2.3.1 Fase 1

La Figura § 2.4, mostra i dettagli relativi alla Fase 1. Si assume di avere a disposizione M byte di memoria per la costruzione del *CF-Tree*, e R byte di spazio sul disco per la processazione degli *outlier* ($R = 0$ significa che l'opzione del trattamento degli *outlier* è disattivata). La Fase 1 inizia con un piccolo valore di soglia, pari a zero, scandisce i dati e inserisce i punti nell'albero attraverso l'algoritmo di inserzione descritto nella Sezione § 2.3.1.1 nella pagina successiva. Se la memoria disponibile si esaurisce prima che l'algoritmo abbia terminato di scansionare i dati, il valore di soglia viene aumentato e viene costruito un nuovo (e più piccolo) *CF-Tree*, reinserendo al suo interno, i campi delle foglie del vecchio albero attraverso l'algoritmo di *rebuilding* descritto nella Sezione § 2.3.1.2 a pagina 77. Dopo l'innesto di tali campi, la scansione dei dati e la loro relativa aggiunta nel nuovo albero, riprende dal punto in cui era stata interrotta.

Figura 2.4: Diagramma di flusso della fase di *pre clustering* (Fase 1)



2.3.1.1 Inserzione nel *CF-Tree*

Di seguito sarà descritto l'algoritmo d'inserzione di un nuovo oggetto *CF* nel *CF-Tree*. Per comodità si denomina tale oggetto “*Entry*”. Ogni “*Entry*” è rappresentata dal suo *CF*, e può rappresentare un unico punto, oppure un *sotto-cluster* di punti di dati, rispettivamente se deve essere inserito un nuovo punto, oppure se deve essere reinserita una *entry-leaf*.

1. *Identificazione della foglia appropriata.* Partendo dalla radice si discende ricorsivamente tutto l'albero seguendo un cammino opportuno: ogni volta che si scende al livello inferiore, viene scelto il figlio più vicino, utilizzando come misura di distanza una delle misure proposte (D_0 (2.4), D_1 (2.5), D_2 (2.6), D_3 (2.7), D_4 (2.8)). Ci si ferma quando è stato raggiunto un nodo foglia.
2. *Modifica della foglia.* Una volta identificato il nodo foglia, si cerca al suo interno la *entry-leaf* più vicina ad “*Entry*”: si supponga essere L_i . A questo punto si testa se L_i può contenere “*Entry*” senza violare il limite di soglia: il nuovo *cluster* ottenuto dalla fusione di “*Entry*” con L_i deve avere diametro minore di T^2 . Se la condizione è rispettata, il nodo foglia *CF* di L_i è aggiornato per rispecchiare l'inserimento. Altrimenti si deve aggiungere alla foglia un nuovo campo per “*Entry*”. Se la foglia non ha spazio sufficiente per il nuovo campo, si deve dividere (*split*) il nodo foglia in due sotto-nodi. I due campi tra loro più lontani nella foglia costituiranno i generatori delle due nuove foglie e i campi rimanenti saranno distribuiti tra i due nodi in base al criterio di maggior vicinanza rispetto alle *entry* generatrici.
3. *Modifica del percorso dalla foglia alla radice.* Dopo l'inserimento di “*Entry*” in una foglia, si devono aggiornare i vettori *CF* lungo tutto il percorso che va dal nodo foglia modificato, alla radice. Se lo *split* non si è verificato, è sufficiente aggiungere al *CF* padre il *CF* che rappresenta “*Entry*”. Se invece si è verificato lo *split* della foglia, è necessario inserire un nuovo campo nel nodo padre, per descrivere la nuova foglia generata. Se il nodo padre ha spazio per il nuovo campo, è sufficiente aggiornare i vettori *CF* nei livelli superiori. In generale comunque può essere necessario effettuare uno *splitting* anche sul nodo padre e così via fino alla radice. Se anche la radice subisce uno *split*, la profondità del *CF-Tree* risulta aumentata di uno.

²Si noti che il *CF* del nuovo *cluster* può essere calcolato sommando le *CF* di “*Entry*” e L_i

4. *Perfezionamento del processo di fusione al termine dello split.* Gli *split* sono causati dalla dimensione della pagina di memoria. La presenza di un ordine sparso nei dati di ingresso può degradare la qualità della classificazione e ridurre l'utilizzo dello spazio. Un semplice passo di fusione addizionale aiuta ad attenuare questo problema: si supponga che si verifichi lo *split* di una foglia e che questo si propaghi ai livelli superiori fino ad arrestarsi ad un certo nodo N_j , il quale è quindi in grado di ospitare il campo aggiuntivo relativo allo *split*. A questo punto si scansiona il nodo N_j alla ricerca dei due campi più vicini, purché siano entrambi diversi da quelli che poco prima hanno subito uno *split*. Si cerca di fonderli e di conseguenza di fondere i due corrispondenti nodi figli. Se il numero di campi contenuti nei due nodi figli è maggiore di quello che può essere contenuto in una pagina di memoria, si effettua un nuovo *split* del risultato della fusione. Durante questo *resplit*, nel caso in cui uno dei due generatori attragga un numero di pagine sufficiente a riempire una pagina, si inseriscono i campi rimanenti nel nodo dell'altro generatore. Ricapitolando, se i campi fusi assieme stanno su una singola pagina, si è così:

- (a) liberato un nodo, che può essere utilizzato con vantaggio in futuro, e
- (b) creato lo spazio per un nuovo campo nel nodo N_j , migliorando l'utilizzo della memoria e post ponendo gli *split* futuri.

In caso contrario si è comunque migliorata la distribuzione dei campi sui due figli più vicini.

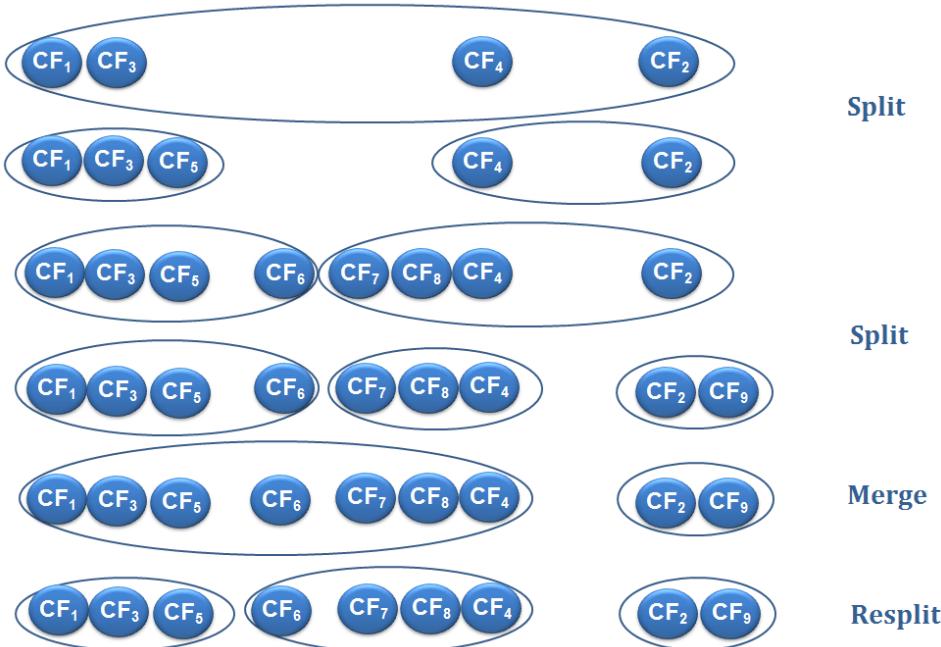
Lo pseudo-codice dell'algoritmo di inserzione è presente nell'Appendice § [B a pagina 203](#).

La Figura § [2.5 nella pagina seguente](#) mostra, intuitivamente, gli effetti dei meccanismi di *splitting*, *merging* e *resplitting*. I grandi ovali rappresentano i nodi foglia (o non-foglia), mentre i piccoli ovali rappresentano le *entry* dei nodi. Si immagini che ogni nodo foglia possa contenere al massimo quattro oggetti e che le *entry* dei nodi foglia siano stati creati nello stesso ordine con il quale sono etichettati: da CF_1 a CF_9 .

Segue l'elenco dei passaggi ottenuti per la creazione dei nodi foglia e dei relativi campi:

1. Man mano che i dati vengono inseriti, vengono create le *entry* da CF_1 a CF_4 nello stesso nodo foglia.

Figura 2.5: Effetto dello *Split*, *Merge* e del *Resplit*



2. Ad una successiva inserzione, viene creata CF_5 la quale causa lo *split* del nodo in questione, creando così un secondo nodo.
3. Con le successive inserzioni, vengono creati i campi da CF_6 a CF_8 e inseriti nei due nodi foglia.
4. Ad una successiva inserzione, viene creata CF_9 la quale causa lo *split* del nodo in questione, creando così un secondo nodo.
5. Viene fusa la coppia di *entry* più vicine presenti nei due nodi foglia (o non-foglia), che evidentemente sono entrambe diverse da quelle che hanno generato lo *split*.
6. La fusione appena effettuata comporta il superamento del valore di *branching* (L o B); è quindi necessario un immediato *resplit* del nodo.

Dai passi appena analizzati, è evidente che i meccanismi di *splitting*, *merging* e spesso anche quello di *resplitting*, operano insieme al fine di migliorare dinamicamente il *CF-Tree* e così ridurre la sua sensibilità all'ordine dei dati in ingresso.

2.3.1.2 Ricostruzione del *CF-Tree*

Di seguito si discute su come compattare (o ricostruire) il *CF-Tree* incrementando il valore di soglia T , nel caso in cui la dimensione dell'albero superi la quantità di memoria riservata per la sua computazione.

Si supponga che t_i sia una *CF-Tree* con soglia T_i , profondità h e dimensione³ S_i . Data una soglia $T_{i+1} \geq T_i$, si vuole disporre di un algoritmo che utilizzi tutti i campi delle foglie di t_i per costruire un nuovo *CF-Tree* t_{i+1} tale che:

$$S_{i+1} \leq S_i$$

ove S_{i+1} è la dimensione dell'albero t_{i+1} , la cui soglia è pari a T_{i+1} .

Si supponga che i campi all'interno di ogni nodo del *CF-Tree* t_i siano numerati in modo contiguo da 0 a $n_k - 1$, dove n_k è il numero di campi nel nodo stesso. Allora un cammino (*path*) da un campo nella radice (livello 1) ad un campo in una foglia (livello h) può essere rappresentato da una h -upla:

$$(i_1, i_2, \dots, i_{h-1})$$

dove

$$i_j \quad (j = 1, 2, \dots, h-1)$$

è l'etichetta del campo al j -esimo livello nel cammino. Si può definire un ordinamento tra i cammini in modo naturale: il cammino $(i_1^{(1)}, i_2^{(1)}, \dots, i_{h-1}^{(1)})$ è minore del cammino $(i_1^{(2)}, i_2^{(2)}, \dots, i_{h-1}^{(2)})$:

$$(i_1^{(1)}, i_2^{(1)}, \dots, i_{h-1}^{(1)}) < (i_1^{(2)}, i_2^{(2)}, \dots, i_{h-1}^{(2)})$$

se :

$$i_1^{(1)} = i_1^{(2)}, i_2^{(1)} = i_2^{(2)}, \dots, i_{j-1}^{(1)} = i_{j-1}^{(2)} \quad e \quad i_j^{(1)} < i_j^{(2)}$$

con $0 < j < h - 1$. Si ha corrispondenza biunivoca tra foglie e cammini: per questo motivo da ora in poi ci riferiremo a foglie o cammini in modo intercambiabile.

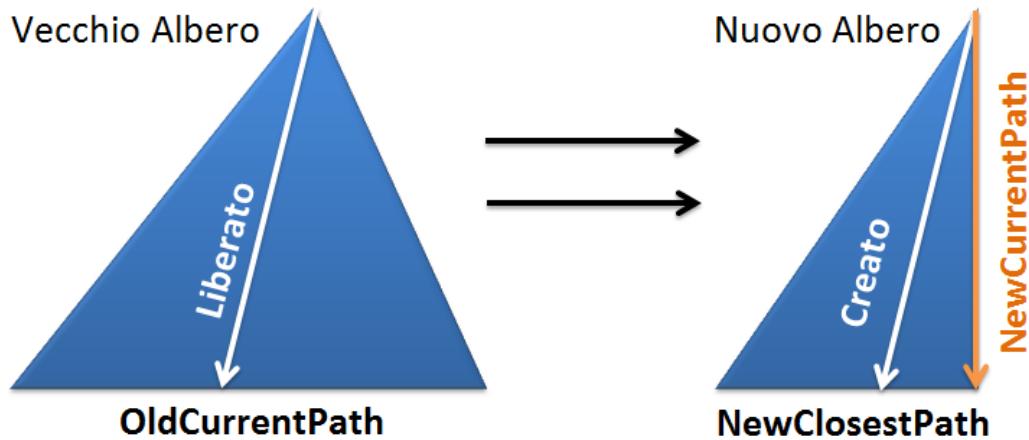
³Intesa come numero di nodi.

Utilizzando l'ordinamento naturale suddetto, l'algoritmo scansiona e libera il vecchio albero, e allo stesso tempo crea il nuovo albero, cammino per cammino. Il nuovo albero assume all'inizio il valore *NULL*, e “*OldCurrentPath*” assume il valore del cammino più a sinistra nel vecchio albero. Per quanto riguarda “*OldCurrentPath*” l'algoritmo procede come segue:

1. *Crea il corrispondente “NewCurrentPath” nel nuovo albero:* i nodi del vecchio albero sono aggiunti al nuovo albero esattamente nello stesso ordine, quindi non c'è possibilità che il nuovo albero possa diventare più grande del vecchio.
2. *Inserisce i campi delle foglie in “OldCurrentPath” nel nuovo albero:* con la nuova soglia si controlla che ogni campo del nodo foglia in “*OldCurrentPath*” sia contenuto nel “*NewClosestPath*” che viene trovato percorrendo il nuovo albero dall'alto al basso seguendo il criterio del cammino più vicino. Se così è, e se “*NewClosestPath*” è minore di “*NewCurrentPath*”, allora la *entry* viene inserita in “*NewClosestPath*” e lo spazio in “*NewCurrentPath*” è reso disponibile per usi futuri; in caso contrario è inserito in “*NewCurrentPath*” senza creare alcun nodo.
3. *Libera lo spazio in “OldCurrentPath” e “NewCurrentPath”:* una volta che tutti i campi in “*OldCurrentPath*” sono stati elaborati, si possono liberare i nodi non necessari lungo “*OldCurrentPath*”. E' anche probabile che qualche nodo lungo il percorso “*NewCurrentPath*” sia vuoto, in quanto i campi delle foglie che originariamente corrispondevano a questo cammino, sono ora “spinti in avanti”. In questo caso anche i nodi vuoti possono essere liberati.
4. *“OldCurrentPath” è impostato al prossimo cammino nel vecchio albero, se ne esiste uno, e i passi precedenti vengono ripetuti.*

La ricostruzione del *CF-Tree* è illustrata nella Figura § 2.6 nella pagina successiva.

Durante le fasi di costruzione del nuovo albero, i vecchi campi delle foglie vengono reinserite, ma il nuovo albero non può mai diventare più grande del vecchio: poiché solo i nodi corrispondenti a “*OldCurrentPath*” e “*NewCurrentPath*” necessitano di esistere simultaneamente, la quantità massima di memoria aggiuntiva necessaria per la trasformazione dell'albero è pari ad h pagine. Quindi, incrementando la soglia, è possibile costruire un nuovo e più piccolo *CF-Tree* con una modesta quantità di memoria aggiuntiva.

Figura 2.6: Costruzione di un nuovo *CF-Tree*

Lo pseudo-codice dell'algoritmo di inserzione è presente nell'Appendice § C a pagina 205.

2.3.1.3 Riducibilità

Di seguito è riportato il teorema di riducibilità che segue immediatamente l'algoritmo di *rebuilding* appena esposto.

Teorema 3. Teorema di Riducibilità: *Si supponga di dover costruire un *CF-Tree* t_{i+1} di soglia T_{i+1} a partire da un albero dei *CF* t_i di soglia T_i e dimensione S_i , utilizzando l'algoritmo precedentemente visto. Se $T_{i+1} \geq T_i$, allora $S_{i+1} \leq S_i$ e la trasformazione da t_i a t_{i+1} richiede al massimo h pagine aggiuntive di memoria, dove h è la profondità dell'albero t_i .*

La dimostrazione segue direttamente dall'algoritmo di *rebuilding*, e quindi è omessa. Con l'algoritmo di *rebuilding*, così come anche il teorema di riducibilità, è garantito che, una volta terminata la memoria, incrementando il valore di soglia, si possa ulteriormente compattare il *CF-Tree* per creare spazio e poter, dunque, consentire l'inserimento di nuovi punti.

2.3.1.4 Anomalie

Poiché ogni nodo può contenere solo un numero limitato di campi, non sempre corrisponde a un *cluster* naturale. Può accadere che due *sotto-cluster*, che dovrebbero appartenere allo stesso *cluster*, risultino divisi su più nodi. A seconda dell'ordine con cui i dati in ingresso vengono processati e in base al loro livello di dispersione, è anche possibile che due *sotto-cluster*, che non dovrebbero trovarsi nello stesso *cluster*, finiscano col trovarsi nello stesso nodo. Queste anomalie, dovute alle dimensioni della pagina di memoria, occorrono raramente, ma sono nondimeno indesiderate. Vi si può porre rimedio tramite un algoritmo globale (o semi-globale) che ridispone i campi delle foglie tra i nodi (Fase 3, Sezione § 2.3.3 a pagina 87).

Un altro fenomeno indesiderato è quello dei dati duplicati: lo stesso punto può presentarsi più volte, ma in tempi diversi. Può accadere che esso sia inserito in diversi campi dei nodi foglia. Oppure, nei casi in cui vi sia dispersione dei dati in ingresso, è possibile che un punto venga inserito in un campo di una foglia a cui non sarebbe dovuto appartenere. Questo problema può essere risolto con dei passaggi algoritmici aggiuntivi sui dati (Fase 4, Sezione § 2.3.4 a pagina 89).

2.3.1.5 Opzione di trattamento degli *outlier*

Gli *outlier*, sono campi delle foglie a bassa densità, considerati non importanti per quanto concerne la distribuzione complessiva dei *cluster*. Si supponga di utilizzare R byte di spazio su disco per gestire gli *outlier* (come caso particolare, potremmo non avere pagine del disco disponibili, cioè, $R = 0$). Questo caso potrebbe essere gestito escludendo il trattamento degli *outlier* nella Fase 1). Quando si ricostruisce l'albero dei *CF*, innestando in esso i campi delle foglie del vecchio albero, la dimensione del nuovo albero viene ridotta per due motivi:

1. aumentando il valore della soglia, si consente ad ogni campo delle foglie di poter “assorbire” più punti;
2. si considerano alcuni campi delle foglie come potenziali *outlier* e li si scrivono su disco.

Un campo delle foglie del vecchio albero è considerato un potenziale *outlier* se ha “*molti meno punti*” rispetto alla media. Il numero dei campi di una foglia è noto dal *CF* che la

rappresenta. La media di tutti i campi delle foglie dell’albero può essere calcolato, mantenendo il numero totale di tutti i dati e il numero totale dei campi nell’albero, man mano che vengono inseriti. Gli autori di [ZRL95] propongono un metodo euristico, secondo il quale un campo è considerato *outlier*, se inferiore ad un quartile della media del numero dei dati per ogni campo delle foglie. Naturalmente, “*molti meno punti*” è una valutazione squisitamente euristica, quindi, per esempio, potrebbe anche significare un valore minore di un quartile del numero medio di punti per ogni *entry-leaf*.

Idealmente, si potrebbero processare tutti gli *outlier* una sola volta, dopo la scansione di tutti i dati di ingresso. Quando avviene il *rebuilding* del *CF-Tree* t_2 a partire da t_1 , può accadere che lo spazio su disco possa esaurirsi, nonostante ci siano ancora dei dati da scansionare. In questo caso, per liberare spazio sul disco, è possibile rianalizzare i potenziali *outlier* per verificare se possono essere riassorbiti nell’albero corrente, senza causare la crescita della dimensione dell’albero stesso. Questo implica che, in seguito all’incremento del valore di soglia o ad un cambiamento nella distribuzione, dovuta ai dati letti dopo che l’*outlier* potenziale è stato scritto su disco, il candidato punto isolato può non qualificarsi più come tale. Quando tutti i dati sono stati analizzati, i potenziali *outlier* che si trovano su disco, devono essere nuovamente scansionati, per verificare se sono da considerarsi effettivamente tali: se non viene assorbito in questa fase è effettivamente un *outlier* e deve essere rimosso.

Si può notare che l’intero ciclo (quantità di memoria insufficiente e conseguente ricostruzione dell’albero, spazio su disco insufficiente, che innesca un riassorbimento degli *outlier*) può essere ripetuto varie volte, prima che il *dataset* sia completamente scansionato. È necessario, dunque, sommare il costo della scansione dei dati al fine di valutare accuratamente il costo della Fase 1.

2.3.1.6 Opzione di ritardo dello split

Oltre all’opzione di trattamento degli *outlier*, *BIRCH* fornisce un’opzione di ritardo dello *split* per creare un *CF-Tree* concentrato in dense regioni e per non far aumentare troppo il valore di soglia. Si può dedurre che, quando termina la memoria principale a disposizione, e ci sono ancora dei punti di dati da processare, non è detto che quest’ultimi debbano tutti necessariamente causare uno *split* e quindi causare il relativo accrescimento della struttura. È molto probabile che una buona quantità di dati, ancora da analizzare, possano essere

assorbiti o aggiunti nelle foglie senza causare *split* di nessun nodo foglia. Una semplice idea è quella di scrivere i punti che causerebbero l'accrescimento dell'albero sul disco (in maniera simile a quello che accade ai potenziali *outlier*), e procedere nella lettura dei dati fino a quando termina lo spazio del disco disponibile per conservare questo tipo di dati. A questo punto si cambia il valore di soglia e si ricostruisce il *CF-Tree*, leggendo nuovamente e assorbendo i punti scritti sul disco precedentemente (esattamente come accade per gli *outlier*). Il vantaggio di questo approccio è che, in generale, vengono assorbiti o aggiunti nell'albero molti più punti, prima dell'incremento del valore di soglia e della conseguente ricostruzione della struttura. Con una buona dose di fortuna, non si dovrebbero più avere tanti *rebuild*!

2.3.1.7 Scelta della soglia

Una scelta oculata del valore di soglia può notevolmente ridurre il numero di ricostruzioni dell'albero. Poiché il valore iniziale della soglia T_0 è incrementato dinamicamente, si può rimediare al fatto di averlo scelto troppo piccolo all'inizio. D'altra parte, se il valore iniziale T_0 fosse troppo alto, si otterrebbe un albero dei *CF* meno dettagliato di quello che potremmo ottenere con la memoria disponibile. Quindi il valore di T_0 dovrebbe essere impostato conservativamente. *BIRCH* imposta il valore di T_0 a zero per *default*, ma un utente accorto potrebbe cambiarne opportunamente il valore.

Si supponga di aver scelto T_i troppo piccolo e che, dopo aver scansionato N_i punti ed aver costituito C_i *entry* nei nodi foglia, si esaurisca la memoria disponibile. Si ha la necessità di calcolare una stima del prossimo valore di soglia T_{i+1} da impostare, e si basa sulla porzione dei dati processati e sull'albero fino a quel momento costruito. Essendo questa una valutazione complessa, ci si accontenta di una soluzione di natura euristica. Gli approcci seguiti dagli autori sono le seguenti:

1. Euristiche basate sulla Regressione

- (a) Si cerca di scegliere T_{i+1} in modo tale che $N_{i+1} = \min(2N_i, N)$. Ciò significa che, se il numero totale di punti è maggiore di $2N_i$, allora T_{i+1} dovrà essere scelto in modo tale da assorbire solo gli N_i punti addizionali prima che lo spazio a disposizione termini nuovamente. In generale, però, il numero dei

punti del *dataset* non è nota⁴. Se così è, dovremmo stimare la dimensione dell’albero in proporzione ai dati analizzati fino a quel momento. Nel caso di un grande *dataset*, ciò potrebbe causare *rebuild* addizionali, ma le successive stime possono essere calcolate attraverso l’aumento del livello di conoscenza dell’insieme di dati.

- (b) Si incrementa la soglia in base ad una qualche misura di *volume*, quindi ogni *entry-leaf* potrebbe essere immaginata come occupante un volume nello spazio multidimensionale, e quindi la soglia è un limite al diametro di questo volume. È possibile usare due distinte nozioni di volume.

La prima è quella di volume medio (*average volume*), definito come $V_a = r^d$, dove r è il raggio medio del *cluster* nella radice del *CF-Tree* e d è la dimensionalità dello spazio. È facile intuire che questa è una misura dello spazio occupato dalla porzione dei dati vista fino a quel momento, detta anche “impronta” (*footprint*) dei dati.

La seconda nozione di volume è quella del *packed volume*, definito come $V_p = C_i * T_i^d$, dove C_i è il numero di campi dei nodi foglia e T_i^d è il volume massimo di un campo foglia. Intuitivamente questa è una misura del volume effettivamente occupato dai *cluster* delle foglie. Poiché ogni volta che si esaurisce la memoria il valore di C_i è essenzialmente lo stesso (si utilizza, infatti, una quantità fissa di memoria), si può approssimare V_p con T_i^d .

Si ipotizzi che r cresca con il numero di punti N_i . Tenendo memorizzati i valori di r ed N_i , è possibile stimare r_{i+1} , utilizzando il metodo di regressione lineare dei minimi quadrati. Si definisce il fattore di espansione f :

$$f = \max \left(1.0, \frac{r_{i+1}}{r_i} \right)$$

e lo si usa come una misura euristica di quanto l’impronta dei dati stia crescendo. La presenza del massimo è giustificata dalla considerazione che, per la maggior parte degli insiemi di dati, l’impronta tende ad assumere un valore costante abbastanza velocemente (a meno che, l’ordine dei dati di *input* sia

⁴nell’equazione si ha $N = \infty$

sparso). Allo stesso modo, basandoci sull'ipotesi che V_p cresca linearmente con N_i , si stima T_{i+1} usando il metodo di regressione lineare dei minimi quadrati.

- (c) Si percorre un cammino dalla radice ad un nodo nel *CF-Tree*, scegliendo sempre il figlio con il maggior numero di punti, nell' "avid" tentativo di trovare il nodo foglia più affollato. Si calcola la distanza (D_{min}) tra i due campi più vicini in questa foglia. Se si desidera costruire un albero più compatto è ragionevole aspettarsi che dovremo alzare il valore della soglia almeno al valore di D_{min} , affinché questi due campi possano essere fusi insieme.
- (d) Si moltiplica il valore di T_{i+1} ottenuto con il metodo della regressione per il fattore di espansione f e lo si modifica utilizzando D_{min} come segue:

$$T_{i+1} = \max(D_{min}, f * T_{i+1}).$$

Per essere sicuri che il valore della soglia cresca in maniera monotona, nel raro caso che il valore di T_{i+1} calcolato come sopra sia minore di T_i , si sceglie

$$T_{i+1} = T_i * \left(\frac{N_{i+1}}{N_i}\right)^{\frac{1}{2}}.$$

Questo equivale ad assumere che sia vera la "cruda" approssimazione che tutti i punti siano uniformemente distribuiti in una sfera d-dimensionale; fortunatamente è necessario ricorrervi molto raramente.

2. Euristiche basate sull'uso della memoria

L'approccio euristico di cui sopra, cerca il corretto valore di soglia per accogliere i dati sconosciuti in arrivo sulla base della regressione lineare dei campioni storici. La correttezza del valore di soglia stimato, e quindi l'uso della memoria così come la qualità di *clustering* finale, dipenderà molto sia dal modello di distribuzione dei dati, sia dal modello di inserimento degli stessi. Nella realtà, però, non è possibile conoscere né il modello di distribuzione dei dati né quello di inserimento senza alcuna conoscenza a priori dei dati, ed inoltre, quantunque fossero noti, il modello di *input* sarebbe comunque estremamente difficile da formalizzare. Quindi l'approccio della regressione lineare potrebbe produrre stime non accurate: una soglia troppo piccola causerebbe molte ricostruzioni dell'albero, mentre un valore di soglia troppo grande causerebbe una pessima qualità dei *cluster*. Si cerca, a questo punto, di applicare euristiche diverse dalla predizione attraverso regressione lineare.

Un altro approccio euristico in fase sperimentale è quello basato sull'utilizzo della memoria. In questo approccio, invece di predire il futuro per i dati da inserire, ci si concentra sul *CF-Tree* in uso, che è una sintesi dei dati analizzati fino a quel momento, per cercare di mantenere l'utilizzo della memoria al di sopra di un livello costante. Il problema da risolvere si può così formalizzare: se l'albero corrente occupa tutta la memoria, allora è necessario comprendere come incrementare il valore di soglia in modo tale che il nuovo *CF-Tree*, ricostruito a partire dal corrente, occupi approssimativamente la metà della memoria. In questo modo,

- l'altra metà della memoria sarà liberata per accogliere i dati di ingresso futuri, e non importa come quest'ultimi saranno distribuiti o inseriti, poiché l'utilizzo della memoria sarà sempre mantenuta approssimativamente sopra il 50%; e
- sarà rilevante e necessaria, per la stima della soglia, solo l'informazione della distribuzione dei dati fino a quel momento analizzati, che sono memorizzati nel *CF-Tree* corrente.

Per realizzare gli obiettivi suddetti, quando il *CF-Tree* corrente usa tutta la memoria, si incrementa il valore di soglia perché sia pari alla media delle distanze tra tutte le coppie più vicine di *entry-leaf*. Quindi, in media, saranno fuse circa due *entry-leaf* in un'unica. Poi per calcolare la distanza di ogni “coppia vicina” di *entry-leaf* efficientemente, è possibile ricercare solo all'interno dello stesso nodo foglia a livello locale, invece di cercare le *entry-leaf* più vicine a livello globale, perché con l'algoritmo di inserzione del *CF-Tree*, è molto probabile che la coppia delle *entry* più vicine si trovi nello stesso nodo foglia. Questo approccio normalmente non sopravvaluta il valore della soglia, ed è quindi molto più stabile.

2.3.2 Fase 2

La Figura § 2.7 nella pagina successiva mostra i dettagli della Fase 2. Come la Fase 1 scansiona tutti i dati e costruisce un albero dei *CF* in memoria, la Fase 2 scansiona tutte le *entry-leaf* ottenute dalla Fase 1 e costruisce un *CF-Tree* più piccolo, il cui numero totale di *entry-leaf* è al di sotto dei valori desiderati. Ci sono diversi aspetti unici degni di nota:

- Il diagramma di flusso della Fase 2, mostra la possibilità che il nuovo *CF-Tree* possa esaurire i valori desiderati prima che tutte le *entry-leaf* del vecchio albero, ottenuto

dalla Fase 1, siano sottoposte a scansione. In tal caso, si deve rivalutare il limite prefissato e ricostruire il nuovo *CF-Tree* esattamente come accadeva nella Fase 1. In realtà, con tutti i dati analizzati e sintetizzati nel *CF-Tree* e le euristiche di cui si è discusso, l'obiettivo di soglia è di solito stimato con molta precisione. Così la parte di ricostruzione e di ricalcolo è raramente richiesto.

- Notare che questa ulteriore fase, elimina ulteriormente valori anomali alcune voci che erano originariamente presenti in un nodo foglia possono, a questo punto, essere rilevati come valori anomali certi, perché è stato analizzato l'intero *set* di dati.
- Questa fase produce anche un *CF-Tree* meno sensibile all'ordine originale dei dati di *input* rispetto a quello della Fase 1, dal momento che le *entry-leaf* inserite sono state ordinate dalla Fase 1 attraverso un processo di *clustering* locale di ottima qualità.

Figura 2.7: Diagramma di flusso della Fase 2



2.3.3 Fase 3

Il compito principale della Fase 3 è di adeguare gli attuali metodi globali o semi-globali di *clustering* per raggruppare i *sotto-cluster*, o le *entry-leaf*, invece che i singoli punti dei dati.

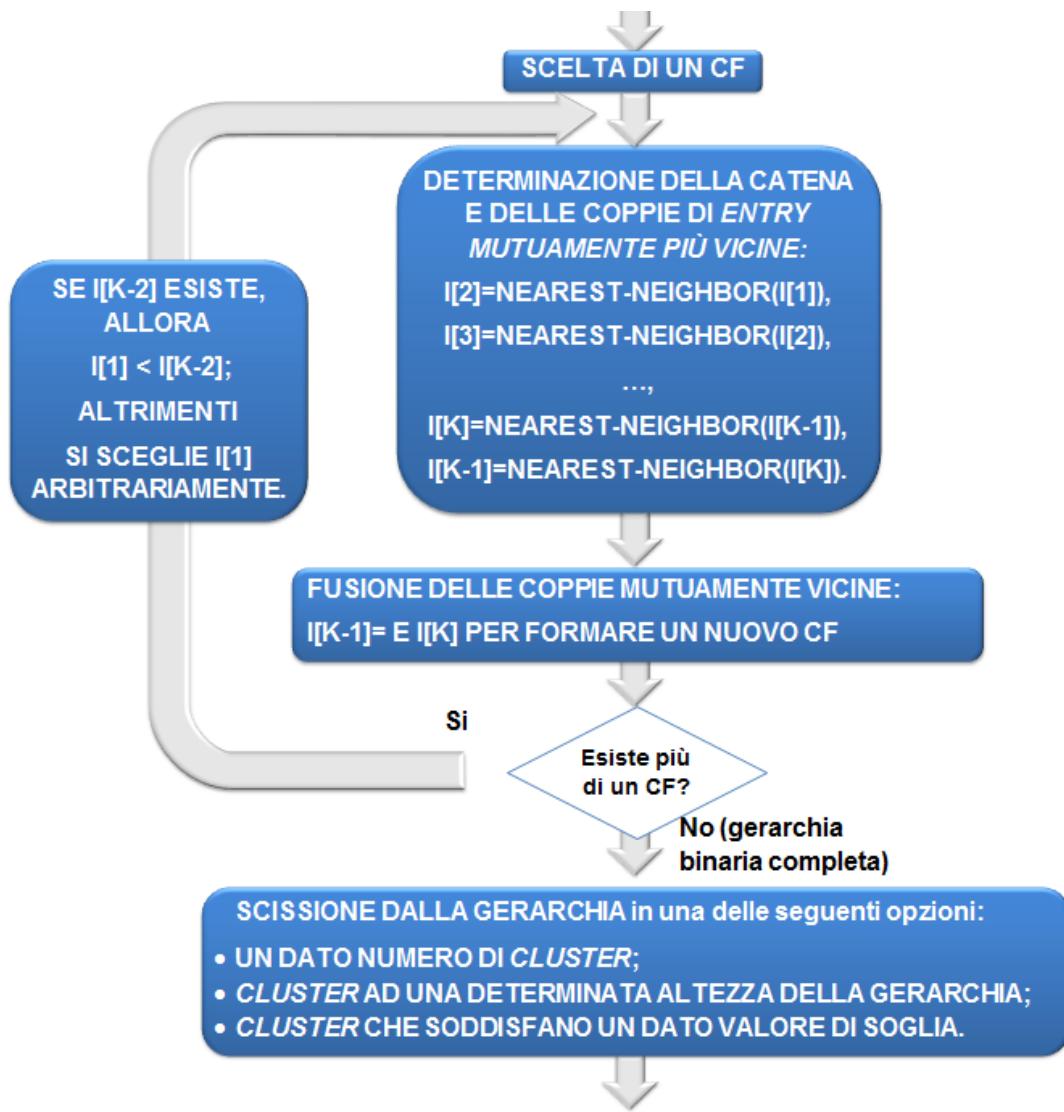
Gli autori di [ZRL95] hanno adattato quattro algoritmi di *clustering* per questa fase:

- *HC1*: Questo è un algoritmo agglomerativo $O(m^3)$ (dove m è il numero di *entry-leaf*) di tipo HC [DH73], che supporta tutte le definizioni di distanza viste (D_0 (2.4), D_1 (2.5), D_2 (2.6), D_3 (2.7) e D_4 (2.8) accuratamente definite nella Sezione §2.1.2), e permette all'utente di trovare i *cluster* specificando il numero di *cluster*, K , o il diametro del *cluster* (o raggio) come soglia T ; o i *cluster* che si trovano ad una data profondità della gerarchia formata.
- *HC2*: Questo è un miglioramento dell'algoritmo agglomerativo *HC* di complessità $O(m^2)$ [Mur83]. È simile a *HC1* ad eccezione del fatto che ha una complessità nel caso migliore di $O(m^2)$, e supporta solo le misure di distanza D_2 (2.6) e D_4 (2.8).
- *CLARANS1*: Il *CLARANS* originale [NH94] si applica ai centroidi dei *sotto-cluster* ottenuti direttamente dalla fase precedente. L'utente deve specificare il numero di K *cluster* perché CLARANS richiede di conoscere K in anticipo.
- *CLARANS2*: *CLARANS* modificato in modo che esso possa essere applicato alle *entry-leaf* invece che sui loro centroidi. Le misure di distanza D_0 (2.4), D_1 (2.5), D_2 (2.6), D_3 (2.7) o D_4 (2.8)) e di qualità (Q1 (2.9), Q2 (2.10), Q3 (2.11) o Q4 (2.12), sono dunque utilizzate nel processo di *clustering*.

La Figura § 2.8 nella pagina seguente mostra il diagramma di flusso del'HC2. Si compone di 2 fasi: la fusione per formare una gerarchia e la divisione dalla gerarchia. Prima si inizia da una *entry* arbitraria, si determina la catena del prossimo campo più vicino, fino a raggiungere la coppia mutualmente più vicina di *CF*. Poi si unisce quest'ultima coppia in un nuovo *CF*, e si ripete sia dalla coda della catena, o da un nuovo ingresso arbitrario *CF* se la coda è vuota. Tutte le fasi di fusione sono mantenuti come una gerarchia binaria. Questa fase di fusione si ferma quando c'è solo un campo rimanente, che è la radice della gerarchia binaria. In secondo luogo si divide dalla gerarchia binaria un determinato numero

di *cluster*, oppure si selezionano i *cluster* di una determinata altezza sulla gerarchia, oppure i *cluster* che soddisfano un dato raggio/diametro come valore di soglia.

Figura 2.8: Diagramma di Flusso della Fase 3

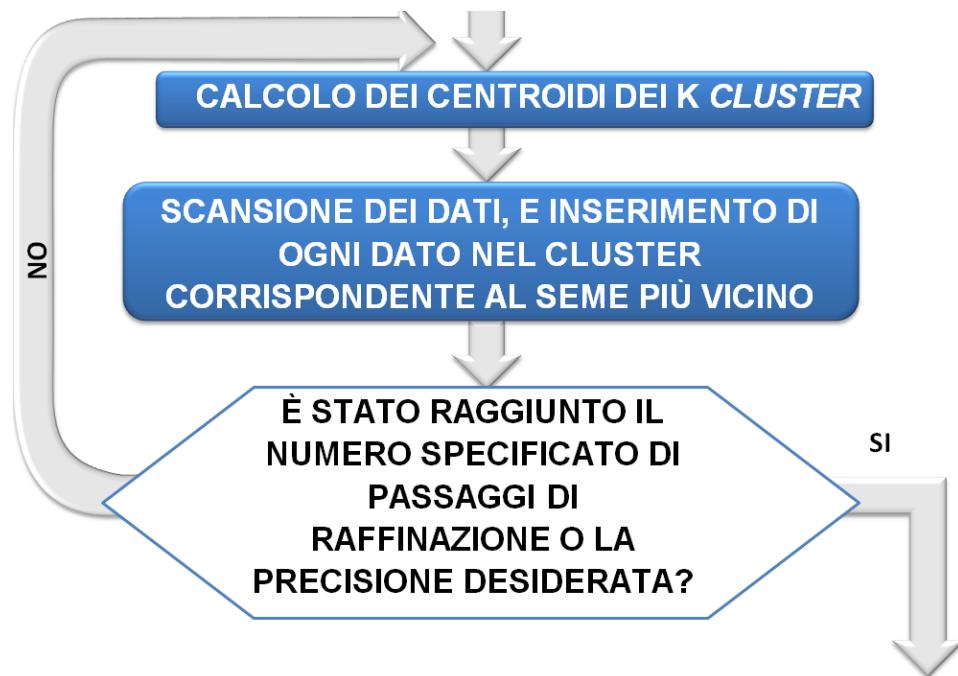


2.3.4 Fase 4

La Figura § 2.9 mostra l'algoritmo di raffinazione della Fase 4. Esso utilizza i centroidi dei *cluster* prodotti nella Fase 3 come semi, e ridistribuisce i punti al relativo seme più vicino per ottenere un insieme di nuovi *cluster*. Questo non solo permette ai punti appartenenti a un *cluster* di migrare in quello più appropriato, ma anche garantisce che tutte le copie di un punto vadano nello medesimo raggruppamento. La Fase 4 può essere estesa con ulteriori passaggi, se desiderato dall'utente, ed è stato dimostrato che converge ad un minimo [GG91]. Come opzionale aggiuntiva, nel corso di questa fase:

- ogni punto di dati può essere etichettato con il *cluster* a cui appartiene, se si vogliono individuare i punti appartenenti ad un medesimo cluster;
- i *set* di dati originali possono essere filtrati in sottoinsiemi in base ai *cluster* ottenuti;
- la Fase 4 fornisce anche la possibilità di disfarsi di valori anomali. Cioè, un punto che è troppo lontano dal suo seme più vicino, può essere trattato come un valore di disturbo, e non essere incluso nel risultato. "Troppo lontano" è di nuovo un'euristica.

Figura 2.9: Diagramma di Flusso della Fase 4



2.4 Utilizzo della memoria

Si sono osservati i seguenti fatti: in *BIRCH* la quantità di memoria necessaria per trovare *cluster* di buona qualità a partire da un *dataset*, non è determinata dalla dimensione dello stesso, ma dalla distribuzione dei dati in esso presenti. D'altra parte, la quantità di memoria disponibile per *BIRCH* è determinata dal sistema informatico. Quindi è molto probabile che la memoria necessaria e la memoria disponibile non corrispondano.

Se la memoria disponibile è inferiore alla memoria necessaria, *BIRCH* può prolungare il tempo di esecuzione, per cercare di ottimizzare l'uso della memoria. In particolare, nelle Fasi dalla 1 alla 3, si tenta di utilizzare tutta la memoria disponibile, per generare sotto-*cluster* la cui dimensione è espressa in funzione della memoria consentita; nella Fase 4, invece, raffinando il *clustering* con alcuni passaggi in più, si risolvono le imprecisioni causate dalla forma eccessivamente sinottica dei *cluster*, dovuta alla memoria insufficiente nella Fase 1.

Se la memoria disponibile è maggiore della memoria necessaria, *BIRCH* può raggruppare il *set* di dati su combinazioni multiple di attributi contemporaneamente, pur condividendo la stessa scansione del *dataset*. Così, di conseguenza, la memoria totale a disposizione sarà, quindi, suddivisa e assegnata al processo di *clustering* di ogni combinazione di attributi. Questo dà all'utente la *chance* di esplorare lo stesso *set* di dati da diverse prospettive contemporaneamente, se le risorse lo consentono.

2.5 *PBIRCH - Parallel Birch Algorithm*

L'obiettivo principale di questa sezione è quella di presentare una versione parallela di *BIRCH*, proposta in [GMGB06], con l'obiettivo di migliorare la scalabilità, senza compromettere la qualità del *clustering*. La parallelizzazione è una tecnica efficace per progettare algoritmi di *clustering* veloci per enormi *set* di dati. I dati in arrivo sono distribuiti in modo ciclico (o a blocchi ciclici se i dati sono sparsi) per bilanciare il carico tra i processori. L'algoritmo è implementato su un modello a trasmissione di messaggi senza condivisione. Gli esperimenti dimostrano che per insiemi di dati molto grandi, la complessità dell'algoritmo aumenta quasi linearmente al crescere del numero di processori. Gli esperimenti dimostrano inoltre che i *cluster* ottenuti con *PBIRCH* sono paragonabili a quelli ottenuti con *BIRCH*.

2.5.1 L'algoritmo *PBIRCH*

L'algoritmo *PBIRCH* viene eseguito sul singolo *Program Multiple Data (SPMD)*⁵ utilizzando il modello *message-passing*⁶. I processori comunicano tra loro attraverso un'interconnessione. Mentre la comunicazione latente comporta una sostanziale dispersione di tempo, quella mediante scambio di messaggi comporta una minore perdita della medesima risorsa.

Inizialmente, i dati da processare sono distribuiti equamente tra i processori. Si supponga di avere p processori e N dati. Quindi ogni processore riceve circa $\frac{N}{p}$ dati e con essi costruisce il proprio *CF-Tree*. Quando i nuovi elementi arrivano, vengono distribuiti, ciclicamente, a tutti i processori. Se i dati sono sparsi, come avviene in numerose applicazioni della vita reale, vengono suddivisi in p blocchi di dimensione approssimativamente uguale

⁵Un singolo programma è eseguito da più processi contemporaneamente. Ogni processo opera su dati diversi, ma, in un certo istante, i processi possono eseguire sia la stessa istruzione che istruzioni diverse. Il programma generalmente contiene opportune istruzioni che consentono al generico processo di eseguire solo parti del codice e/o di operare solo su un sottoinsieme dei dati. Può essere realizzato con diversi modelli di programmazione. Tutti gli eseguibili partono assieme (creazione statica dei processi).

⁶Il message passing è un paradigma di programmazione per la progettazione di algoritmi in un ambiente di calcolo parallelo per calcolatori a memoria distribuita. Nel progettare l'algoritmo, si deve, dunque, tener conto che esistono processori, ognuno con una “*propria*” memoria, capaci di eseguire ciascuno un “*proprio*” insieme di istruzioni. Chiaramente, i processori devono sincronizzarsi per risolvere un medesimo problema, e ciò avviene attraverso lo scambio di messaggi. Il concetto fondamentale alla base del paradigma in questione è che i processori comunicano tra loro inviando e ricevendo dati.

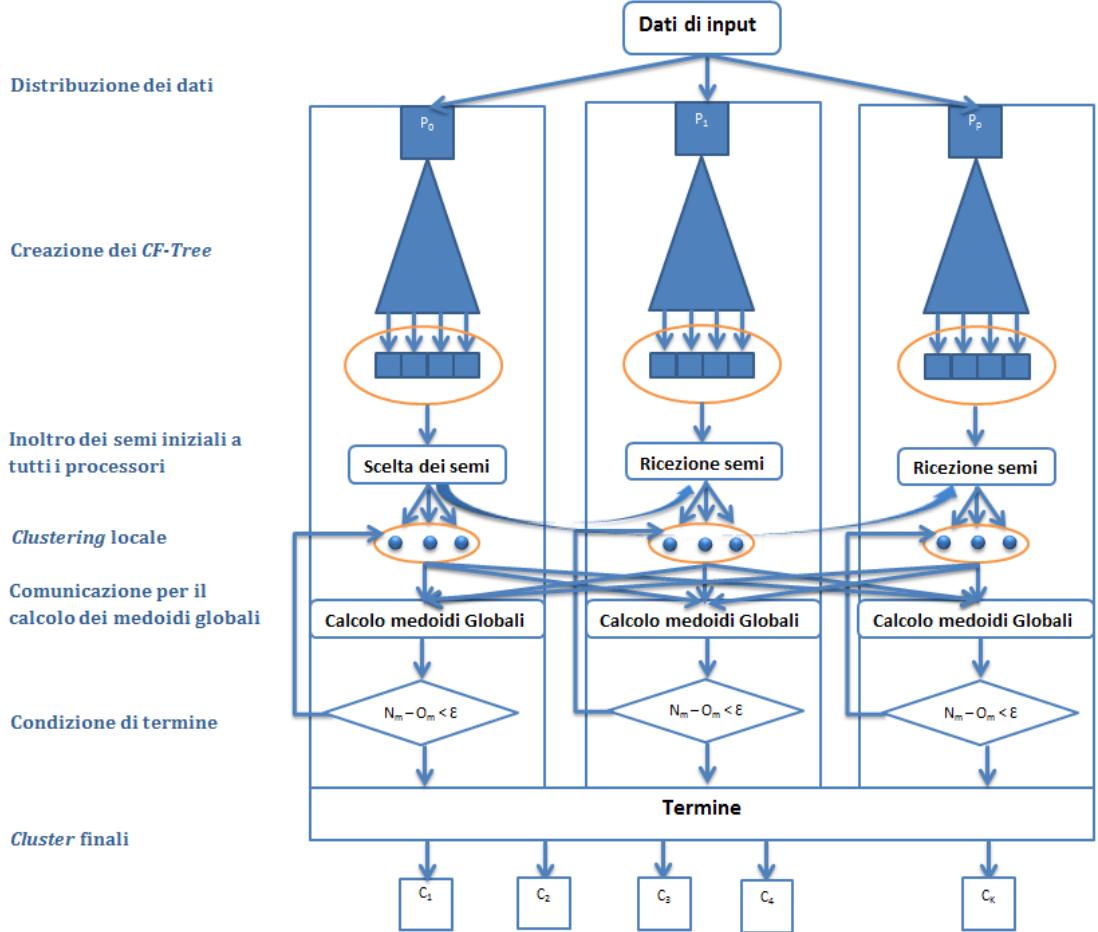
e distribuiti tra i processori. Ogni processore inserisce i nuovi dati nella propria struttura dei *CF* locale.

Dopo aver costruito l'albero dei *CF*, come nel caso di *BIRCH*, viene utilizzato un algoritmo di *clustering* per classificare i nodi foglia quando richiesto. Tuttavia, in un tale contesto, è necessario un algoritmo di *clustering* parallelo per eseguire queste operazioni [Ols93]. Questo è l'unico passo che richiede lo scambio di dati tra i processori.

Di seguito si elencano i passi principali dell'algoritmo *PBIRCH*:

1. Distribuzione dei dati: Inizialmente i dati vengono suddivisi equamente tra tutti i processori. Quando i nuovi elementi arrivano, sono distribuiti ciclicamente (oppure ciclicamente a blocchi) a tutti i processori.
2. Computazione degli alberi *CF* contemporaneamente: Il *CF-Tree* è generato da ogni processore utilizzando i dati locali. Se un albero esaurisce la memoria, in un qualsiasi momento, allora deve essere ricostruito aumentando il valore di soglia.
3. Applicare il *clustering* sui nodi foglia: applicare un algoritmo parallelo di *clustering* per i *CF* memorizzati nei nodi foglia.

La Figura § 2.10 a fronte mostra in generale l'algoritmo in questione utilizzando come algoritmo di *clustering* parallelo l'algoritmo *K-MEANS* [Jos03].

Figura 2.10: Schema dell'algoritmo *PBIRCH*

Gli autori di [GMGB06], poiché il numero di *CF* nei nodi foglia del *CF-Tree* è molto inferiore al numero dei dati, utilizzano, come algoritmo di *clustering* globale parallelo, il *K-MEANS* [Jos03]. Inizialmente, un processore sceglie i k semi iniziali, e li trasmette a tutti gli altri processori. Ogni processore utilizza il seme dei suoi *cluster* a livello locale. I centroidi di quest'ultimi, sono poi scambiati tra tutti i processori per poter calcolare i centroidi a livello globale. Questo è l'unico passo che richiede lo scambio dei dati tra i processori ad ogni iterazione. A questo punto ogni processore ha k centroidi globali. A partire da questi k medoidi, ogni processore ricalcola i *cluster* e ripete il procedimento. L'algoritmo termina quando i cambiamenti dei *cluster*, durante le iterazioni successive, diventano inferiori ad un determinato valore di soglia.

2.6 Studio delle prestazioni

In questa sezione, si analizzeranno, dapprima i costi di *BIRCH* in termini di utilizzo della *CPU* e di operazioni di *I/O*; successivamente saranno presentate le valutazioni sugli esperimenti circa le prestazioni di *BIRCH* (a confronto con *K-MEANS* e *CLARANS*) su *dataset* riprodotti per studiare e testare la fattibilità dell'algoritmo proposto.

2.6.1 Analisi della Complessità

In primo luogo, si analizza il costo in termini di utilizzo della *CPU* durante la Fase 1. La dimensione massima della struttura è $\frac{M}{P}$, dove M è la quantità di memoria espressa in *byte*, e P è la dimensione di ogni pagina di memoria. Per inserire un punto, abbiamo bisogno di seguire un percorso dalla radice alla foglia, toccando circa $1 + \log_B \frac{M}{P}$ nodi. Ad ogni nodo dobbiamo esaminare *B entry*, alla ricerca del più “vicino”; il costo di analisi di ogni campo è proporzionale alla dimensione d . Così il costo per l'inserimento è $O(d * N * B (1 + \log_B \frac{M}{P}))$. Nel caso in cui fosse necessario ricostruire l'albero, sia $C*d$ la dimensione di una *CF-entry*, dove C è una costante di mappatura della dimensione all'interno di un *CF-entry*. Ci sono più di $\frac{M}{C*d}$ nodi foglia da reinserire, quindi il costo di reinserimento dei nodi foglia è $O(d * \frac{M}{C*d} * B (1 + \log_B \frac{M}{P}))$. Il numero di volte che dobbiamo ricostruire l'albero, dipende dall'euristica di soglia. Attualmente, si tratta di $\log_2 \frac{N}{N_0}$, dove il valore 2 deriva dal fatto che, o non si stima mai più del doppio della dimensione dell'albero nell'approccio di regressione, oppure l'albero viene sempre ridotto a circa la metà della sua dimensione originale nell'approccio di utilizzo della memoria, e N_0 è il numero di punti caricati in memoria con soglia T_0 . Così il costo totale in termini di utilizzo della CPU durante la Fase 1 è $O\left(d * N * B (1 + \log_B \frac{M}{P}) + \log_2 \frac{N}{N_0} * d * \frac{M}{C*d} * B (1 + \log_B \frac{M}{P})\right)$. Poiché $B = \frac{P}{C*d}$ la formula appena ottenuta può essere riscritta nel modo seguente:

$$O\left(N * \frac{P}{C} \left(1 + \log_{\frac{P}{C*d}} \frac{M}{P}\right) + \log_2 \frac{N}{N_0} * \frac{M}{C*d} * \frac{P}{C} \left(1 + \log_{\frac{P}{C*d}} \frac{M}{P}\right)\right)$$

L'analisi della Fase 2, in termini di utilizzo della CPU, è simile e quindi omessa.

Per quanto riguarda il costo in termini di operazioni di *I/O*, si ha che, la scansione dei dati avviene una sola volta nella Fase 1. Considerando la gestione degli *outlier*, e le

opzioni di ritardo dello *split*, vi è un certo costo associato per la scrittura dei potenziali *outlier* su disco e per la lettura degli stessi nel corso di una ricostruzione. Considerando che la quantità di disco disponibile per la gestione degli *outlier* (e il ritardo degli *split*) non è molta, e che avvengono $\log_2 \frac{N}{N_0}$ ricostruzioni, il costo in termini di operazioni di *I/O* durante la Fase 1 non è significativamente diverso dal costo di lettura dal *set* di dati di origine.

Non ci sono operazioni di *I/O* nella Fase 3. Dal momento che i dati d'ingresso nella Fase 3 sono limitati, il costo della *CPU* della Fase 3 è quindi limitato da una costante che dipende dalla dimensione massima dei dati d'ingresso e dall'algoritmo globale scelto per questa fase. Sulla base delle analisi di cui, sopra il costo delle Fasi 1, 2 e 3 devono scalare linearmente con N .

La Fase 4 analizza nuovamente il *dataset* e inserisce ogni punto nel *cluster* corretto; il tempo impiegato è proporzionale a $N * K$. (Tuttavia con le nuove tecniche di ricerca del valore “più vicino” [GG91], può essere migliorato per essere quasi lineare ad N).

2.6.2 Parametri e Impostazioni di *default*

BIRCH è in grado di lavorare con varie impostazioni. La Tabella§ 2.1 nella pagina successiva elenca i parametri di *BIRCH*, la loro visibilità ed i loro valori di *default*, utilizzati negli esperimenti effettuati dagli autori dell'algoritmo proposto.

M , la dimensione della memoria usata in fase di *pre-clustering*, è di circa il 5% della dimensione del *set* di dati. Poiché lo spazio su disco (R) è usato solo per scrivere gli *outlier*, si sceglie $R < M$, e quindi $R = 20\%$ di M . Come misura di distanza è stata scelta D_2 , mentre per misurare la qualità del *clustering* è stata scelta la misura Q_2 (2.10 a pagina 64, indicata con \bar{D}). Minore è il valore di \bar{D} , migliore è la qualità dei *cluster*.

La soglia è definita per il diametro del *cluster*. Nella Fase 1, la soglia iniziale è impostata a 0. La dimensione della pagina di memoria è $P = 1024$. Per la costruzione di un *CF-Tree* più compatto viene gestito il ritardo dello *split*, mentre l'opzione di gestione degli *outlier* è disabilitata, per semplicità.

Per la Fase 3, poiché la maggior parte degli algoritmi globali sono in grado di gestire bene, almeno qualche migliaio di oggetti, gli autori hanno impostato la dimensione dell'*input* a 1000. Inoltre, negli esperimenti presenti in [ZRL95], gli autori utilizzano, come algoritmo di *clustering* globale, l'*HC*. Inoltre è stato scelto di operare il raffinamento

dei *cluster* nella Fase 4 solo una volta, e di non scartare gli *outlier*, in modo tale che tutti i punti siano conteggiati nella misura di qualità per il confronto reale con altri metodi di *clustering*.

Tabella 2.1: *BIRCH*: Parametri e relativi valori di *default*

Visibilità	Parametro	Valore di <i>default</i>
<i>Globale</i>	Memoria (M) Disco (R) Definizione di distanza Definizione di Qualità Definizione di Soglia	5% della dimensione del <i>dataset</i> 20% M D_2 \bar{D} Soglia per D
Fase 1	Soglia iniziale Split-Ritardato Dimensione della Pagina (P) Gestione <i>outlier</i>	0.0 si 1024 byte no
Fase 3	<i>Range</i> di <i>Input</i> Algoritmo	1000 <i>HC</i> adattato
Fase 4	Passi di raffinamento Scarto <i>outlier</i>	1 noi

2.6.3 Prestazioni sulla base del carico di lavoro

La prima serie di esperimenti è volta a valutare la capacità di *BIRCH*, di operare su *dataset* di grandi dimensioni e con varia distribuzione dei punti e ordine dei dati in ingresso. Il tempo è espresso in secondi. Sono stati utilizzati tre *dataset* bidimensionali:

- **Dataset 1 (DS1):** Contiene 100 *cluster*, aventi i centri disposti su una struttura a griglia, (Figura § 2.11 a pagina 101);
- **Dataset 2 (DS2):** Contiene 100 *cluster*, aventi i centri disposti lunga una curva che rappresenta il grafico di una funzione seno, e i punti contenuti in ogni *cluster* seguono una distribuzione normale, avenete media pari al centro del *cluster* stesso (Figura § 2.12 a pagina 101);
- **Dataset 3 (DS3):** Contiene 100 *cluster*, aventi i centri disposti in maniera del tutto casuale, e i punti contenuti in ogni *cluster* seguono una distribuzione normale, avenete media pari al centro del *cluster* stesso (Figura § 2.13 a pagina 101).

L'ordine dei dati con cui questi tre *dataset* sono processati è del tutto casuale ed i punti, contenuti in ogni *cluster*, seguono una distribuzione normale, avente media pari al centro del *cluster* stesso.

La tabella § 2.2 riassume le caratteristiche dei *dataset*. I diametri medi ponderati dei *cluster* previsti \bar{D}_{int} , sono indicazioni di massima qualità per ciascun *dataset*. Si noti che da ora in poi, si fa riferimento ai *cluster* dei vari *dataset* di prova come “*cluster previsti*”, mentre ai *cluster* individuati da *BIRCH*, come “*cluster BIRCH*”. Inoltre l'etichetta di ciascun *cluster* è data dal numero di punti ad esso appartenenti.

Tabella 2.2: *Dataset* usati per testare il carico di lavoro

<i>DS</i>	Impostazioni Generate	\bar{D}_{int}
1	$d = 2$, griglia, $K = 100$, $o=casuale$	2.0
2	$d = 2$, seno, $K = 100$, $o=casuale$	2.0
3	$d = 2$, casuale, $K = 100$, $o=casuale$	4.18

La Figura § 2.11 a pagina 101 visualizza i *cluster previsti* del *dataset DS1*. I *cluster* di *BIRCH* su *DS1* rappresentati nella Figura § 2.14 a pagina 102. Si osserva che i *cluster* *BIRCH* sono molto simili ai *cluster* previsti in termini di ubicazione, numero di punti, e raggi. La distanza massima e media tra i centroidi di un *cluster previsto* e il suo corrispondente *cluster BIRCH* è rispettivamente 0.17 e 0.07. Il numero di punti in un *cluster BIRCH* è non più del 5% rispetto al *cluster previsto* corrispondente. I raggi dei *cluster BIRCH* (che vanno da 1.25 a 1.40 con una media di 1.32) sono molto simili a quelli dei *cluster previsti* (1.41). Si noti che tutti i raggi dei *cluster BIRCH* sono più piccoli dei raggi dei *cluster previsti*. Analoghe conclusioni possono essere raggiunte analizzando le figure dei *cluster previsti* (Figura § 2.12) e dei *cluster BIRCH* per il *DS2* (Figura § 2.17).

Come la Tabella § 2.3 a pagina 100 evidenzia, ci sono voluti meno di 15 secondi con *BIRCH* (su un calcolatore *DEC Pentium-Pro* con *Solaris*) per raggruppare 100.000 punti di ogni *dataset*, considerando anche le 2 scansioni del *dataset* (circa 1.16 secondi per una scansione di file *ASCII* da disco). Il modello del *dataset* non ha avuto quasi alcun effetto sul tempo di *clustering*. La Tabella § 2.3, inoltre, presenta i risultati delle *performance* per tre *dataset* aggiuntivi - *DS1o*, *DS2o* e *DS3o* - che corrispondono rispettivamente a *DS1*, *DS2* e *DS3*, salvo per il fatto che i punti che li compongono, vengono processati in maniera ordinata e non casuale. Come dimostrato dalla Tabella 4, cambiando l'ordine dei punti non si ha alcun impatto sulle prestazioni di *BIRCH*.

2.6.4 Confronto tra *BIRCH*, *CLARANS* e *K-MEANS*

In questo esperimento, mettiamo a confronto le prestazioni di *BIRCH*, *CLARANS* e *K-MEANS* sulla base del carico di lavoro. È bene premettere che:

- in *CLARANS* e *K-MEANS*, la memoria si presume essere abbastanza grande da contenere tutto il *set* di dati così come alcune dimensioni lineari che assistono le strutture dati. Quindi hanno bisogno di molta più memoria rispetto a *BIRCH*. (Chiaramente, questa ipotesi favorisce notevolmente questi due algoritmi in termini di confronto sul tempo di esecuzione!)
- l'implementazione del *CLARANS* e del *K-MEANS* è quella presentata rispettivamente in [NH94] e [HW79].

Le Figure § 2.15 a pagina 102 e § 2.16 a pagina 102 visualizzano i *cluster* *CLARANS* e *K-MEANS* per *DS1*. Confrontandoli con i *cluster* previsti per *DS1*, si osserva che:

1. Il modello di localizzazione dei centri di *cluster* è distorto;
2. Il numero di punti in un *cluster* *CLARANS* o *K-MEANS* possono essere fino al 40% diversi dal numero dei *cluster* previsti;
3. I raggi dei *cluster* *CLARANS* variano ampiamente da 1.15 a 1.94 con una media di 1.44 (più grandi di quelle dei *cluster* previsti, 1.41). I raggi dei *cluster* *K-MEANS* variano ampiamente da 0.99 a 2.02 con una media di 1.38 (più grandi di quelle dei *cluster* *BIRCH*, 1.32).

Un comportamento simile può essere osservato nelle Figure § 2.18 a pagina 103 e § 2.19 a pagina 103, rispettivamente dei *cluster* *CLARANS* e *K-MEANS* su *DS2*. Le Tabelle § 2.4 a pagina 100 e § 2.5 a pagina 100 riassumono le prestazioni degli algoritmi di confronto.

Per tutti e tre i *set* di dati sulla base del carico di lavoro si può notare che:

1. i due algoritmi di confronto eseguono la scansione del *dataset* di frequente. Quando vengono eseguiti gli esperimenti usando *CLARANS* e *K-MEANS*, tutti i dati vengono caricati in memoria, solo la prima scansione è da file *ASCII* sul disco, e le scansioni restanti sono in memoria. Considerando il tempo necessario per ogni scansione sul disco (1.16 secondi per la scansione), *CLARANS* e *K-MEANS* sono molto più lenti di *BIRCH* e i tempi di esecuzione sono più sensibili ai modelli dei *dataset*;

2. i valori di \bar{D} per i *cluster* *CLARANS* e *K-MEANS* sono maggiori di quelli per i *cluster* di *BIRCH*, sia su *DS1* che su *DS2*. Ciò implica che la qualità dei *cluster* *CLARANS* e *K-MEANS* è peggiore rispetto a quella dei *cluster* *BIRCH*, poiché i due algoritmi di confronto possono trascorrere molto tempo alla ricerca di una partizione locale ottimale.
3. i risultati su *DS1o*, *DS2o* e *DS3o* mostrano che se i punti sono inseriti in ordine diverso, il tempo e la qualità dei *cluster* *CLARANS* e *K-MEANS* cambia. Quindi le prestazioni dei due algoritmi di confronto sono estremamente instabili rispetto all'ordine di *input* dei dati, mentre *BIRCH*, in questo, risulta essere molto più stabile.

In conclusione, sulla base del carico di lavoro, *BIRCH* utilizza molta meno memoria, la scansione dei dati viene eseguita solo due volte, è più veloce e accurato, ed è meno sensibile all'ordine dei dati rispetto a *CLARANS* e *K-MEANS*.

Tabella 2.3: *Performance* di *BIRCH* sulla base del carico di lavoro rispetto a Tempo, \bar{D} , Ordine di *Input* e Numero di Scansioni dei Dati

<i>DS</i>	Tempo	\bar{D}	#Scansioni	<i>DS</i>	Tempo	\bar{D}	#Scansioni
1	11.5	1.87	2	1o	13.6	1.87	2
2	10.7	1.99	2	2o	12.1	1.99	2
3	11.4	3.95	2	3o	12.2	3.99	2

Tabella 2.4: *Performance* di CLARANS sulla base del carico di lavoro rispetto a Tempo, \bar{D} , Ordine di *Input* e Numero di Scansioni dei Dati

<i>DS</i>	Tempo	\bar{D}	#Scansioni	<i>DS</i>	Tempo	\bar{D}	#Scansioni
1	932	2.10	3307	1o	794	2.11	2854
2	758	2.63	2661	2o	816	2.31	2933
3	835	3.39	2959	3o	934	3.28	3369

Tabella 2.5: *Performance* di *K-MEANS* sulla base del carico di lavoro rispetto a Tempo, \bar{D} , Ordine di *Input* e Numero di Scansioni dei Dati

<i>DS</i>	Tempo	\bar{D}	#Scansioni	<i>DS</i>	Tempo	\bar{D}	#Scansioni
1	43.9	2.09	289	1o	33.8	1.97	197
2	13.2	4.43	51	2o	12.7	4.20	29
3	32.9	3.66	187	3o	36.0	4.35	241

Figura 2.11: *Cluster Previsti* per il DS1

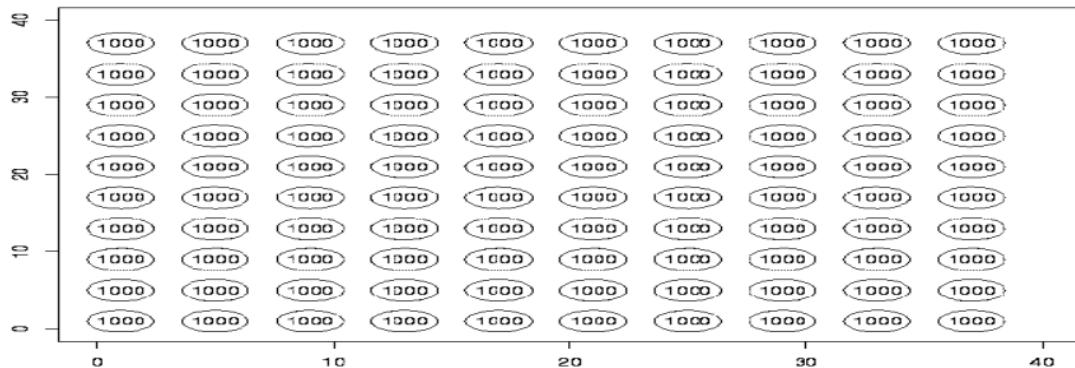


Figura 2.12: *Cluster Previsti* per il DS2

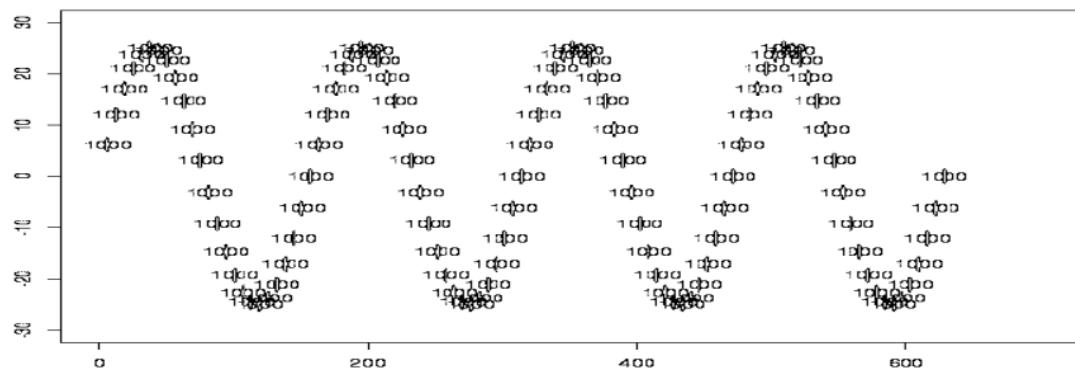


Figura 2.13: *Cluster Previsti* per il DS3

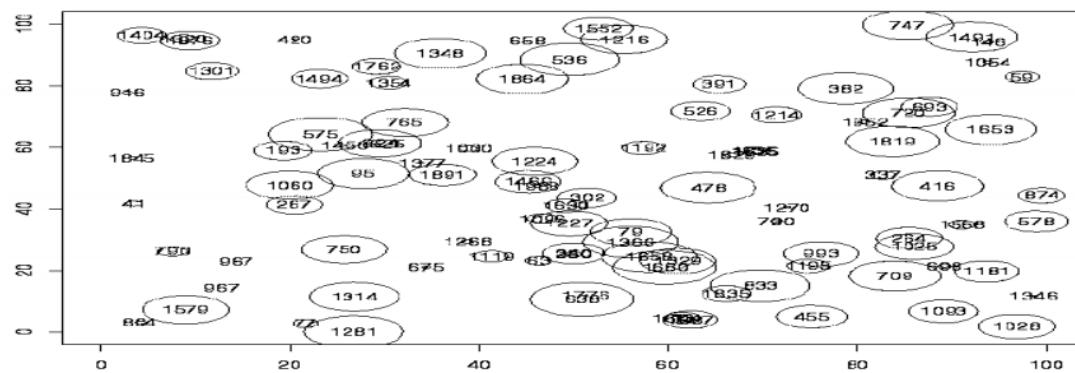


Figura 2.14: *Cluster BIRCH* per il *DS1*

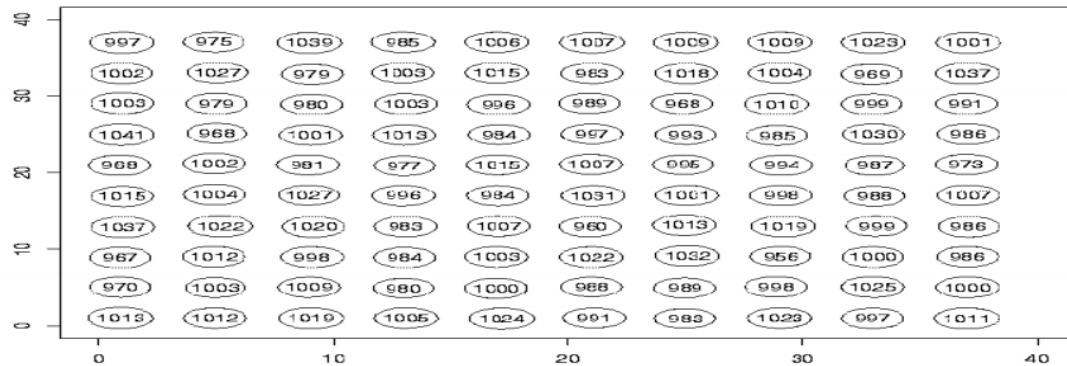


Figura 2.15: *Cluster CLARANS* per il *DS1*

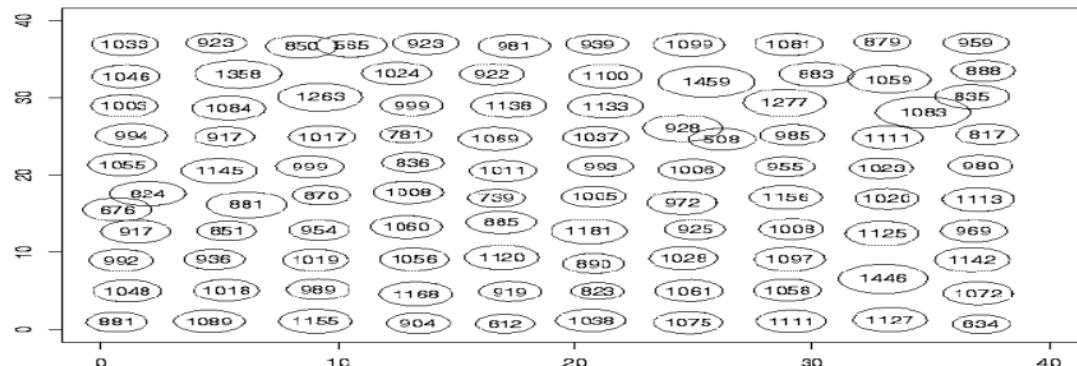


Figura 2.16: *Cluster K-MEANS* per il *DS1*

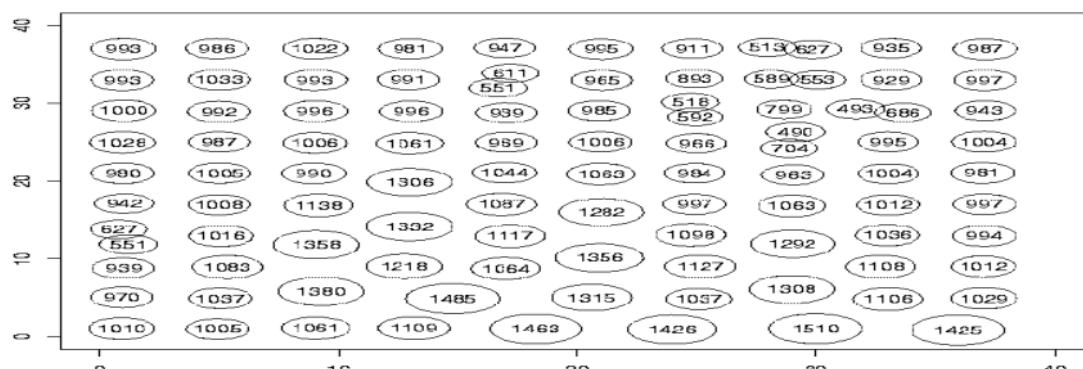


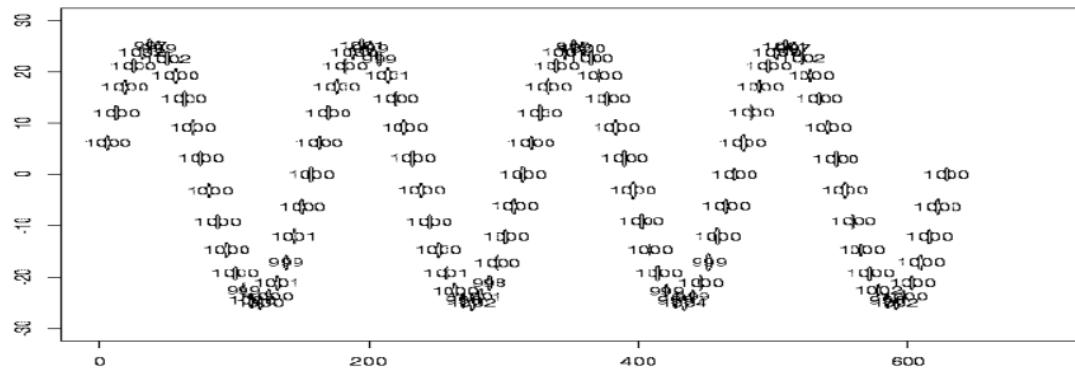
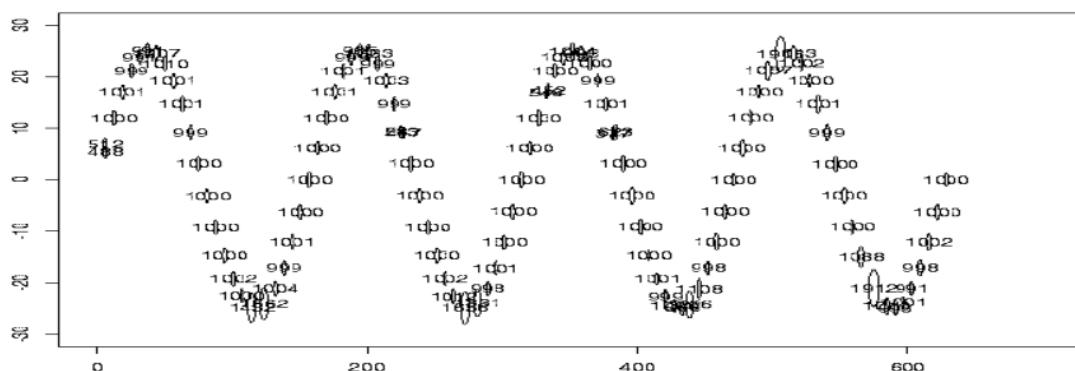
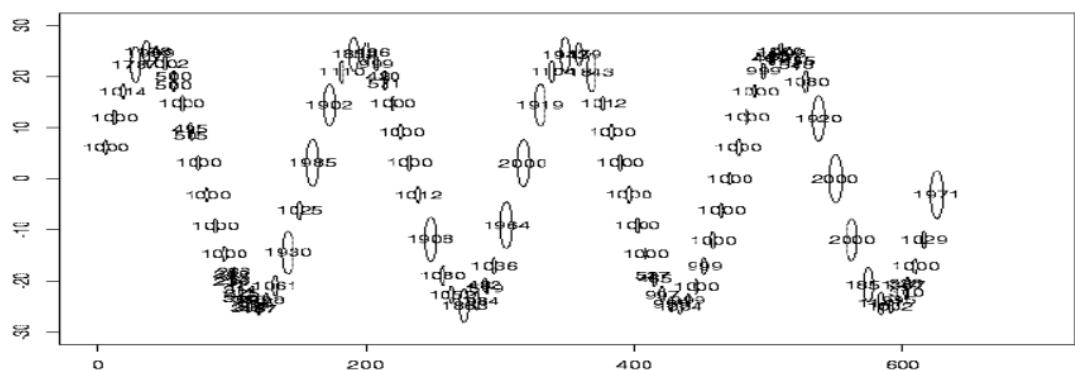
Figura 2.17: *Cluster BIRCH* per il DS2

 Figura 2.18: *Cluster CLARANS* per il DS2

 Figura 2.19: *Cluster K-MEANS* per il DS2


Figura 2.20: *Cluster BIRCH* per il DS3

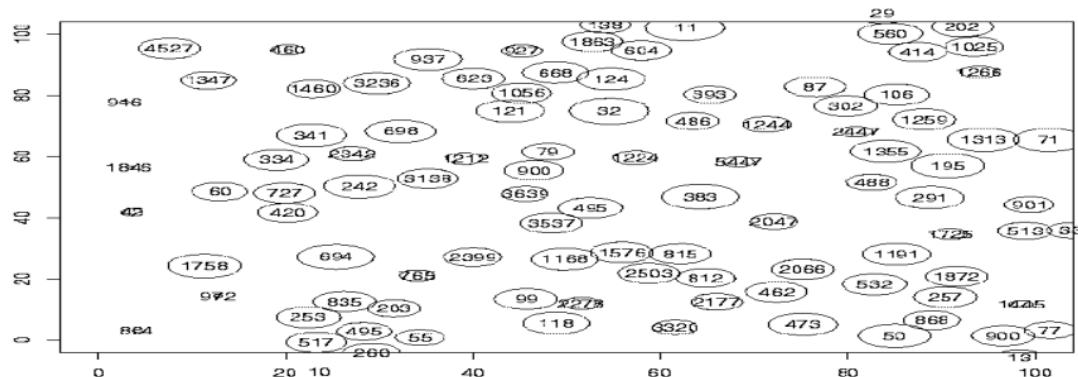


Figura 2.21: *Cluster CLARANS* per il DS3

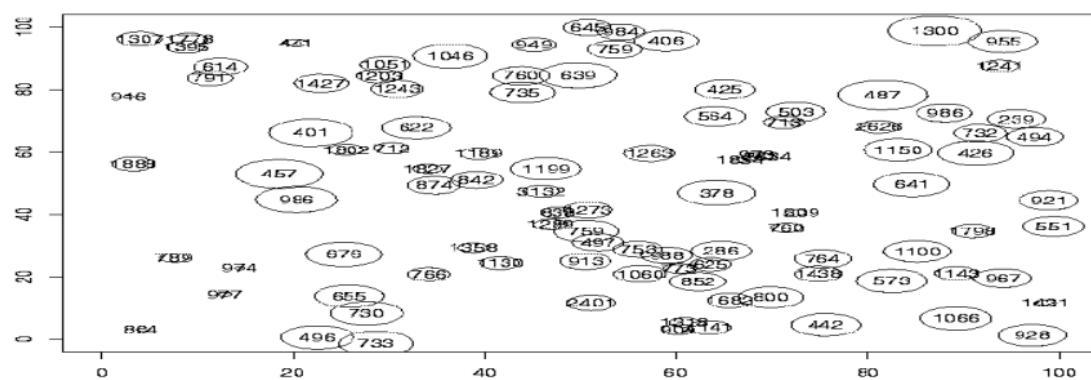
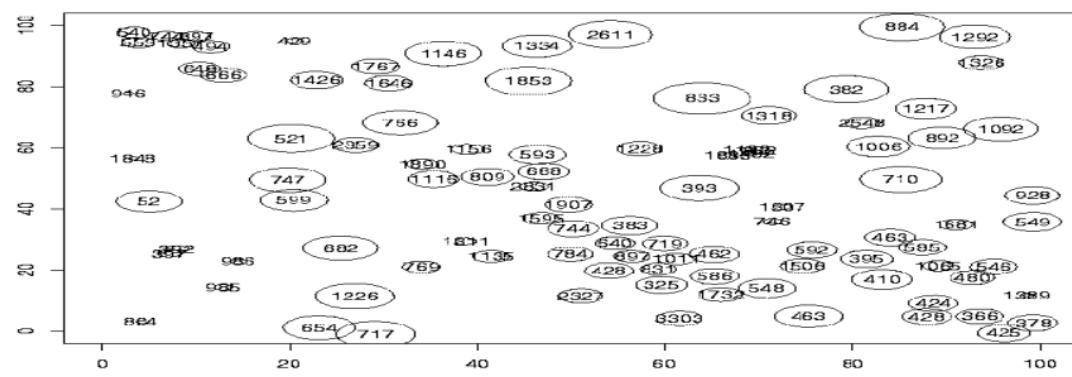


Figura 2.22: *Cluster K-MEANS* per il DS3



2.6.5 Sensibilità ai Parametri

Gli autori di [ZRL95] hanno analizzato la sensibilità delle *performance* di *BIRCH* a diversi parametri. Riportiamo di seguito alcune conclusioni importanti.

- **Soglia iniziale:** come si evince dalla Tabella § 2.6

Tabella 2.6: Sensibilità al valore di Soglia Iniziale

T_0	<i>DS1</i>		<i>DS2</i>		<i>DS3</i>	
	Tempo	D	Tempo	D	Tempo	D
0.0	47.14	1.87	47.5	1.99	49.52	3.39
0.001	46.39	1.87	47.75	1.99	49.83	3.39
0.01	46.75	1.87	45.77	1.99	49.75	3.39
0.1	47.34	1.87	46.14	1.99	48.47	3.39
1.0	44.75	1.87	43.98	1.99	45.77	3.39

- le *performance* di *BIRCH* (tempo e qualità) sono stabili fino a quando il valore di soglia iniziale non è troppo alto;
- il valore di soglia di *default* $T_0 = 0.0$ lavora bene, anche se comporta un piccolo aumento del tempo di esecuzione;
- ottimizzando il valore di soglia iniziale T_0 , si può risparmiare fino al 10% del tempo.
- **Fattore di Ramificazione B:** se si sceglie un *branching factor* molto piccolo, mantenendo uguali il valore di soglia e la quantità di dati, si ottiene un *CF-Tree* molto grande. Questo accade perché avere un valore molto piccolo di B significa avere meno informazioni per ogni livello, le quali sono necessarie per guidare la giusta collocazione dell'inserzione dell'albero. In tal caso, infatti, i punti che avrebbero potuto essere assorbiti in una *leaf-entry* esistente, se diretti nella foglia appropriata, potrebbero andare a finire nel nodo sbagliato e provocare la generazione di una nuova *leaf-entry*. Così un piccolo valore di B provoca un numero maggiore di ricostruzioni dell'albero, eventualmente può richiedere un valore di soglia più alto e generare meno *entry* alla fine della Fase 1, colpendo, dunque, l'efficienza e la qualità di *BIRCH*

- **Dimensione della pagina P:** nella Fase 1, un valore più piccolo (grande) di P tende a diminuire (aumentare) il tempo di esecuzione, e quindi a richiedere un valore di soglia finale più alto (basso), produrre un numero minore (maggiore) di *entry-leaf* più sintetiche (dettagliate), e quindi degrada (migliora) la qualità dei *cluster*. Tuttavia, con l'affinamento nella Fase 4, gli esperimenti suggeriscono che per valori di P compresi tra 256 a 4096 byte, anche se la qualità dei vari *cluster* al termine della fase 3 è differente, la qualità finale dopo il raffinamento è al massimo la stessa.
- **Opzione di gestione degli Outlier:** come si evince dalla Tabella § 2.7, *BIRCH* è stato testato su dataset “rumorosi”, sia attivando che disattivando l'opzione di gestione degli *outlier*. I risultati mostrano che gestendo gli *outlier*, *BIRCH* è addirittura più veloce che senza la gestione degli stessi, e al tempo stesso, la qualità dei *cluster* è di gran lunga migliore.

Tabella 2.7: Sensibilità all'opzione di gestione degli *outlier*

Dataset	Opzione di Gestione degli <i>outlier</i>			
	On		Off	
	Tempo	D	Tempo	D
<i>DS4</i>	48.02	1.92	48.63	1.94
<i>DS5</i>	47.14	2.05	51.51	8.44
<i>DS6</i>	47.41	4.41	49.68	5.06

- **Algoritmo di clustering globale della Fase 3:** come si evince dalla Tabella § 2.7,
 - *HC1* e *HC2* risultano esattamente di uguale qualità con la misura di distanza D_2 e D_4 , ma il secondo algoritmo è più veloce.
 - *CLARANS1* è meno veloce del *CLARANS2*, e la sua qualità non è decisamente la migliore.
 - *HC1* e *HC2* hanno qualità più stabili di *CLARANS1* e *CLARANS2*.

Gli autori del *BIRCH* fanno notare che questo tipo di esperimenti li hanno indotti ad utilizzare ufficialmente, come algoritmo di clustering globale per la Fase 3, l'*HC2*.

- **Dimensione della memoria:** Nella Fase 1, all'aumentare della dimensione della memoria (o dimensione massima dell'albero):

Tabella 2.8: Sensibilità all'algoritmo di *clustering* globale per la Fase 3

algoritmo di <i>clustering</i> globale	<i>DS1</i>		<i>DS2</i>		<i>DS3</i>	
	Tempo	D	Tempo	D	Tempo	D
<i>HC1</i>	55.29	1.87	39.79	1.99	47.21	3.39
<i>HC2</i>	2.27	1.87	1.84	1.99	1.98	3.39
<i>CLARANS1</i>	59.15	1.99	69.35	2.15	52.79	3.73
<i>CLARANS2</i>	27.43	2.04	22.22	2.48	47.34	3.14

1. aumenta il tempo di esecuzione a causa delle ricostruzioni dell'albero, ma di poco, poiché quest'ultime sono eseguite in memoria;
2. vengono generati un numero maggiore e più dettagliato di *sotto-cluster*, e quindi di questo si traduce in una migliore qualità dei *cluster*;
3. le imprecisioni causate dalla memoria insufficiente, possono essere compensate dalla Fase 4 di raffinamento.

In altre parole, *BIRCH* fornisce un buon compromesso tra utilizzo della memoria in funzione del tempo per raggiungere una buona qualità finale dei *cluster*.

- **Misure di distanza:** gli autori hanno osservato che tra tutte le misure di distanza, solo la D_3 modifica effettivamente le *performance* del *BIRCH*: essa, infatti, provoca una maggiore frequenza nella ricostruzione dell'albero dei *CF*, quindi un conseguente valore finale di soglia molto alto, che ne degrada efficienza e qualità dell'algoritmo proposto. Le rimanenti quattro misure (D_0 , D_1 , D_2 , D_4), non apportano, invece, significative differenze di prestazioni.

2.6.6 Scalabilità

Per testare la scalabilità di *BIRCH*, sono stati utilizzati, tre diversi metodi per aumentare le dimensioni del *dataset*.

Aumentare il numero di punti per *Cluster* (n): i *dataset* $DS1$, $DS2$ e $DS3$ sono stati opportunamente modificati al fine aumentare il numero dei dati senza alterare i modelli originali. Nella Figura § 2.23 nella pagina successiva sono riportati i tre modelli dei *dataset*, il loro tempo di esecuzione sia delle prime 3 fasi, sia di tutte e 4 le fasi in relazione alla dimensione N del *dataset*. Si può osservare che:

1. Il tempo di esecuzione sia delle prime 3 fasi, che di tutte le 4 fasi, scala linearmente rispetto a N ;
2. I tempi di esecuzione per le prime 3 fasi crescono in modo simile per tutti e tre i modelli dei *dataset*.
3. L'algoritmo migliorato per la ricerca del “vicino più prossimo” usato in Fase 4 [BKSS90], è leggermente più sensibile ai modelli dei dati. Funziona meglio per il modello sinusoidale, perché di solito ci sono meno centri di *cluster* intorno a un punto.

La Tabella § 2.9 a pagina 110 fornisce i valori di qualità corrispondenti ai *cluster previsti* (\bar{D}_{int}) e ai *cluster BIRCH* (\bar{D}) all'aumentare di N per tutti e tre i modelli. È facile osservare dalla tabella che, con la stessa quantità di memoria, per aumentando il numero massimo di dati di *input* N , la qualità dei *cluster BIRCH* (indicato da \bar{D}) è sempre vicina (o migliore, a causa della correzione degli “*outsider*”) a quella ai *cluster previsti* (indicato con \bar{D}_{int}).

Figura 2.23: Scalabilità del tempo rispetto all'Incremento Del Numero di Punti per *Cluster*

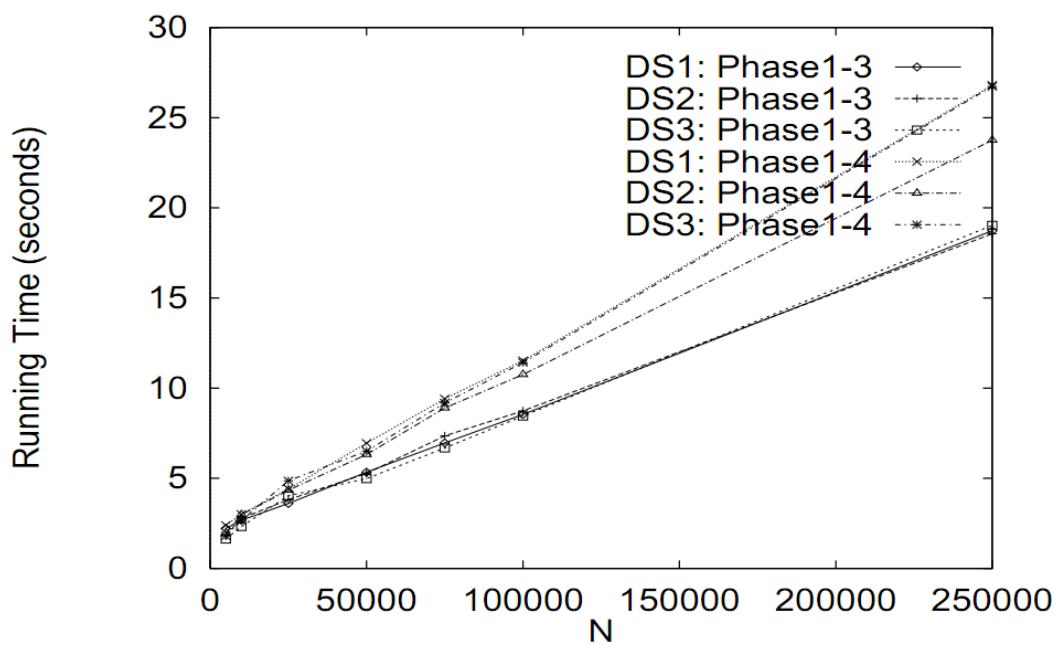


Tabella 2.9: Qualità della Scalabilità rispetto all'Incremento Del Numero di Punti per *Cluster*

<i>DS</i>	<i>n</i> in $n \in [n_l..n_h]$	<i>N</i>	\bar{D}/\bar{D}_{int}
1	50..50	5000	1.87/1.99
	100..100	10000	1.92/2.01
	250..250	25000	1.87/1.99
	500..500	50000	1.86/1.98
	750..750	75000	1.88/2.00
	1000..1000	100000	1.87/2.00
	2500..2500	250000	1.88/2.00
2	50..50	5000	1.98/1.98
	100..100	10000	2.00/2.00
	250..250	25000	2.00/2.00
	500..500	50000	2.00/2.00
	750..750	75000	1.99/1.99
	1000..1000	100000	1.99/2.00
	2500..2500	250000	1.99/1.99
3	0..100	5000	4.08/4.42
	0..200	10000	4.30/4.78
	0..500	25000	4.21/4.65
	0..1000	50000	4.00/4.27
	0..1500	75000	3.74/4.22
	0..2000	100000	3.95/4.18
	0..5000	250000	4.23/4.52

Aumentare il numero di *cluster* (K): i dataset $DS1$, $DS2$ e $DS3$ sono stati opportunamente modificati al fine aumentare il numero *cluster* K senza alterare i modelli originali. Nella Figura § 2.24 nella pagina successiva vengono riportati i tempi di esecuzione sia delle prime 3 fasi, che per tutte e 4 le fasi, rispetto alla dimensione del dataset N .

1. Ancora una volta, le prime 3 fasi si confermano scalabili linearmente rispetto a N per tutti e tre i modelli dei *dataset*.
2. I tempi di esecuzione per tutte e 4 le fasi sono lineari rispetto ad N , ma hanno pendenze leggermente diverse per ciascuno dei tre diversi modelli di *dataset*. Più in particolare, per il modello a griglia la pendenza è maggiore, mentre per il pattern sinusoidale, la pendenza è il minore. Ciò è dovuto al fatto che K e N crescono nello stesso tempo, e la complessità della Fase 4 è, nel caso peggiore, $O(K * N)$ (non lineare a N). Pur migliorando l'algoritmo di raffinazione della Fase 4, usando le tecniche di ricerca del “più vicino” proposta in [BKSS90], e pur ottenendo sostanziali miglioramenti che riducono il tempo di complessità $O(K * N)$ fino ad essere quasi valori lineari rispetto a N , la pendenza è comunque sensibile ai modelli di distribuzione dei dati. Nel caso in esame, per il modello a griglia, dal momento che di solito ci sono più centri di *cluster* intorno ad un punto, c’è bisogno di più tempo per trovare quello più vicino, mentre per il modello sinusoidale, dal momento che di solito ci sono meno centri di *cluster* intorno ad un punto di dati, c’è bisogno di meno tempo per trovare quello più vicino, il modello casuale è quindi al una via di mezzo.

Per quanto riguarda la stabilità della qualità, la Tabella §Table 2.10 on page 113 mostra che con la stessa quantità di memoria, per una valore limite sempre maggiore di K e N , la qualità dei *cluster* *BIRCH* (indicato da \bar{D}) è di nuovo costantemente vicina ai (o migliore dei) *cluster previsti* (indicato da \bar{D}_{int}).

Figura 2.24: Scalabilità del tempo rispetto all'Incremento Del Numero di *Cluster*

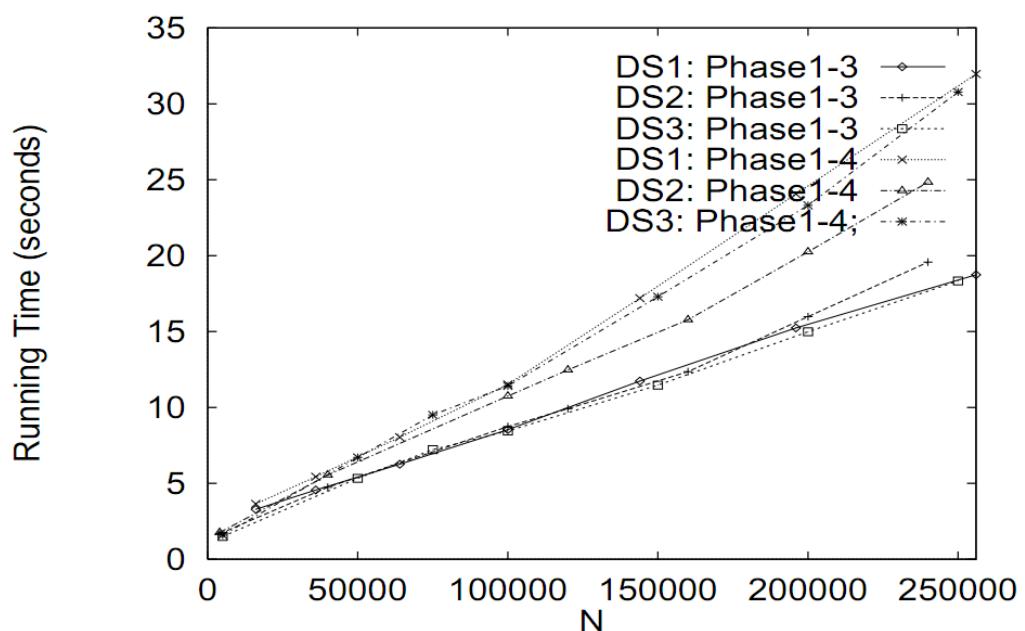


Tabella 2.10: Qualità della Scalabilità rispetto all'Incremento Del Numero di *Cluster*

<i>DS</i>	<i>K</i>	N	\bar{D}/\bar{D}_{int}
1	16	16000	2.32/2.00
	36	36000	1.98/2.00
	64	64000	1.93/1.99
	100	100000	1.87/2.00
	144	144000	1.87/1.99
	196	196000	1.87/2.00
	256	256000	1.87/2.00
2	4	4000	1.98/1.99
	40	50000	1.99/1.99
	100	100000	1.99/2.00
	120	120000	2.00/2.00
	160	160000	2.00/2.00
	200	200000	1.99/1.99
	240	240000	1.99/1.99
3	5	5000	5.57/6.43
	50	50000	4.10/4.52
	75	75000	4.04/4.76
	100	100000	3.95/4.18
	150	150000	4.21/4.26
	200	200000	5.22/4.49
	250	250000	5.52/4.48

Aumentare la dimensione (d): i *dataset* $DS1$, $DS2$ e $DS3$ sono stati opportunamente modificati al fine aumentare il numero delle dimensioni dei *dataset* senza alterar i modelli originali. Nella Figura § 2.25 a fronte vengono riportati i tempi di esecuzione sia per le prime 3 fasi, che per tutte e 4 le fasi rispetto alla dimensione. Osserviamo all'aumentare delle dimensioni ed al corrispondente aumento di memoria, aumenta il tempo di esecuzione. Ciò è causato dal fatto seguente: con M_0 (una quantità costante di memoria), la quantità di memoria corrispondente a un determinato *dataset* d -dimensionale, viene scalata mediante l'equazione $M_0 * d$. Ora, se N e P sono costanti, all'aumentare delle dimensioni, aumenta la complessità temporale secondo l'equazione $1 + \log_{\frac{P}{C*d}} \frac{M_0*d}{P}$ secondo l'analisi nella Sezione § 2.6.1 a pagina 94. Cioè, all'aumentare della dimensione e della memoria, aumenta in primo luogo la dimensione del *CF-Tree*; in secondo luogo diminuisce il fattore di ramificazione (*branching factor*), provocando l'aumento della profondità dell'albero. Così, avendo un valore grande di d , all'atto dell'inserzione di un nuovo punto, è necessario passare attraverso più livelli di un più grande albero di *CF*, e ciò evidentemente richiede più tempo. La cosa interessante è che regolando opportunamente P , si può fare il ridimensionamento delle curve sublineari o superlineari, e in questo caso, con $P = 1024$ byte, le curve sono leggermente superlineari.

Per quanto riguarda la stabilità di qualità, la Tabella § 2.11 a pagina 116 mostra che per un valore di d molto alto, la qualità dei *cluster* *BIRCH* (indicato da \bar{D}) è ancora una volta vicina a (o migliore di) quella dei *cluster* previsti (indicato da \bar{D}_{int}).

Figura 2.25: Scalabilità del tempo rispetto all'Incremento Del Numero delle Dimensioni

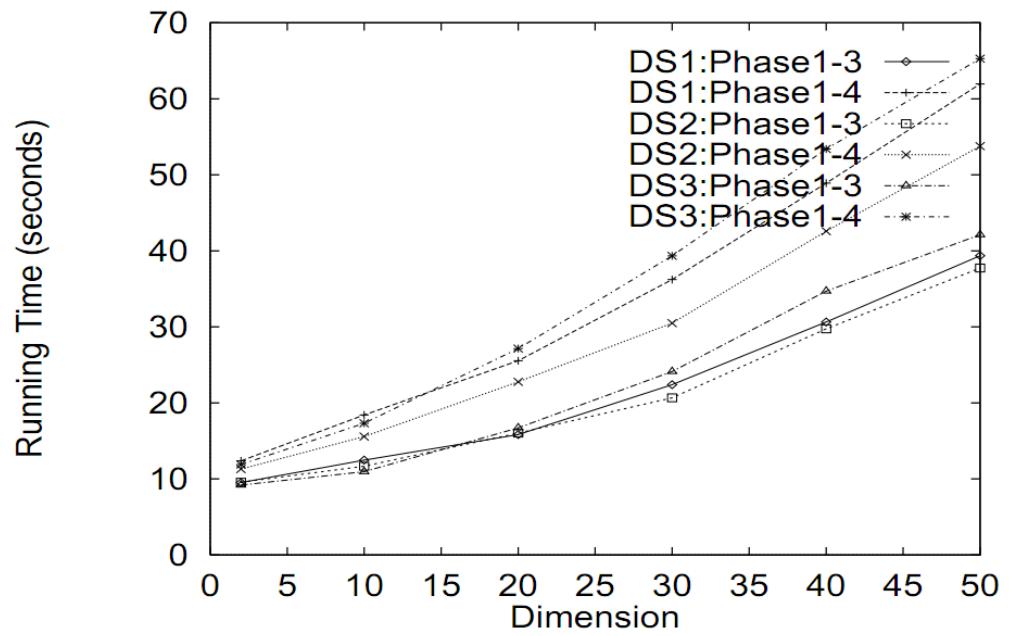


Tabella 2.11: Qualità della Scalabilità rispetto all'Incremento Del Numero delle Dimensioni

DS	d (dimensioni)	\bar{D}/\bar{D}_{int}
1	2	1.87/1.99
	10	4.68/4.46
	20	6.52/6.31
	30	7.87/7.74
	40	8.97/8.93
	50	10.08/9.99
2	2	1.99/1.99
	10	4.46/4.46
	20	6.31/6.31
	30	7.74/7.74
	40	8.93/8.93
	50	10.02/9.99
3	2	3.95/4.28
	10	11.96/9.62
	20	17.11/13.62
	30	20.76/16.70
	40	25.34/19.28
	50	26.64/21.56

Capitolo 3

Implementazione e Descrizione di *OLAPBIRCH*

*Ragionamento matematico può essere considerato
piuttosto schematicamente come
l'esercizio di una combinazione di due impianti,
che possiamo chiamare intuizione e inventiva.*

ALAN TURING

3.1 Introduzione

IN questo capitolo si fornisce una descrizione dettagliata di **OLAPBIRCH**, l'implementazione *software* dell'algoritmo *BIRCH* da noi sviluppata. Verranno illustrate le classi che compongono il sistema **OLAPBIRCH**, quelle che consentono la connessione ad una base di dati e le classi di utilità. L'esposizione sarà di tipo non formale, coadiuvata da una descrizione in linguaggio UML per spiegare al meglio l'interazione tra le classi.

La trattazione proposta pone enfasi sull'architettura *software* del progetto.

3.1.1 Motivazioni alla base della scelta del nome ***OLAPBIRCH***

L'implementazione *software* dell'algoritmo sviluppato è stato denominato ***OLAPBIRCH***.

I motivi alla base di tale scelta sono i seguenti:

- ***OLAPBIRCH*** estende l'algoritmo ***BIRCH***, presentato nel Capitolo § [2 a pagina 61](#);
- ***OLAPBIRCH*** è progettato col fine di favorire l'integrazione con le tecnologie ***OLAP***.

3.1.2 Tipologia dell'applicazione

OLAPBIRCH è un'applicazione per i sistemi operativi *Microsoft Windows*, *Linux*, *Solaris* ed *Apple OS X*, sviluppata utilizzando il linguaggio di programmazione *Java* nell'ambito dell'ambiente di sviluppo integrato¹ *Eclipse*². Si è optato per l'utilizzo del linguaggio *Java*, come strumento di codifica, sostanzialmente per due ragioni:

- è ormai saldamente parte del nostro *know-how*;
- consente di ottenere codice eseguibile su diversi sistemi operativi; inoltre, la codifica che si ottiene può essere elegante, ma soprattutto orientata agli oggetti.

OLAPBIRCH è un *software* fortemente orientato agli oggetti (*object oriented*). La scelta di un linguaggio di programmazione che supporti il paradigma *object oriented* deriva dall'esigenza di garantire:

- un possibile riuso delle componenti *software*;
- la semplificazione del processo di *porting* del *software* su piattaforme differenti. Quest'ultimo punto è stato reso possibile isolando specifiche funzionalità o servizi all'interno di appositi *package* o classi progettate appositamente, in modo tale da poter modificare singole componenti del progetto, senza doversi preoccupare dell'impatto che tali modifiche possono avere sulle restanti componenti *software*.

¹*IDE, Integrated Development Environment*

²<http://www.eclipse.org/>

3.1.3 Applicazioni utilizzate per lo sviluppo

Per la creazione di ***OLAPBIRCH*** sono state utilizzate le seguenti componenti *software*:

- *IDE Eclipse SDK* versione 3.6.1;
- *Java SDK* versione 6 *update 22*;
- *MySQL³* *Server* versione 5.1.53;
- *MySQL Connector⁴* *Java* versione 5.1.14;
- *Weka⁵* versione 3.6.4.

3.2 Architettura *software*

Nella presente sezione saranno descritte le principali classi componenti il progetto, sviluppate per creare i necessari tipi di dato. Le classi non vengono considerate a sé stanti, ma anche nel contesto della loro interazione reciproca.

Le classi componenti il progetto, verranno suddivise in tre categorie, secondo la loro funzione svolta all'interno del *software*. Tale suddivisione non è equivalente alla suddivisione per *package* adottata nel progetto.

3.2.1 Classi algoritmo OLAPBIRCH

In questa categoria rientrano le classi implementate per creare la struttura dati *CF-Tree* utilizzata nell'algoritmo *BIRCH*. Per realizzarla sono state individuate le tre componenti fondamentali di tale struttura:

1. la componente *clustering feature*, che ha il ruolo di rappresentare il *cluster*;
2. la componente *nodo*, che svolge il compito di “*nodo dell'albero Clustering Feature Tree*” e viene usata per contenere *clustering feature* nei diversi livelli della struttura;

³<http://mysql.com/>

⁴<http://mysql.com/products/connector/>

⁵<http://www.cs.waikato.ac.nz/ml/weka/>

3. la componente *Clustering Feature Tree*, che assolve al ruolo di “*albero delle clustering feature*” e che utilizza la componente nodo e i servizi da essi resi disponibili.

Tale divisione si è resa necessaria per rendere modulare la realizzazione, utilizzando un modello di sviluppo del tipo *bottom-up*.

Le classi implementate rientrano in una delle tre tipologie sopra illustrate e sono state suddivise in *package*⁶ a seconda della funzione svolta. Pertanto i *package* creati sono:

- “*birch.cf_tree*”, racchiude la classe che implementa la struttura ad albero *CF-Tree* e la classe di gestione degli *outlier*.
- “*birch.cf_tree.cf_node*”, raggruppa le tipologie di nodi presenti nella struttura ad albero adoperata.
- “*birch.cf_tree.cf_node.cf_entry*”, racchiude le diverse tipologie di *entry* adoperate nella struttura e contenute nei relativi nodi.

3.2.1.1 Classi *package* “*birch.cf_tree.cf_node.cf_entry*”

Nel *package* in esame, le classi implementano il *clustering feature*, ossia la struttura alla base dell’albero usato nell’algoritmo *BIRCH*. Il *clustering feature*, definito § [19 a pagina 65](#), costituisce una rappresentazione sintetica del *cluster* che consente di calcolare tutte le grandezze statistiche di interesse ad esso relative, richiedendo uno spazio di memorizzazione esiguo.

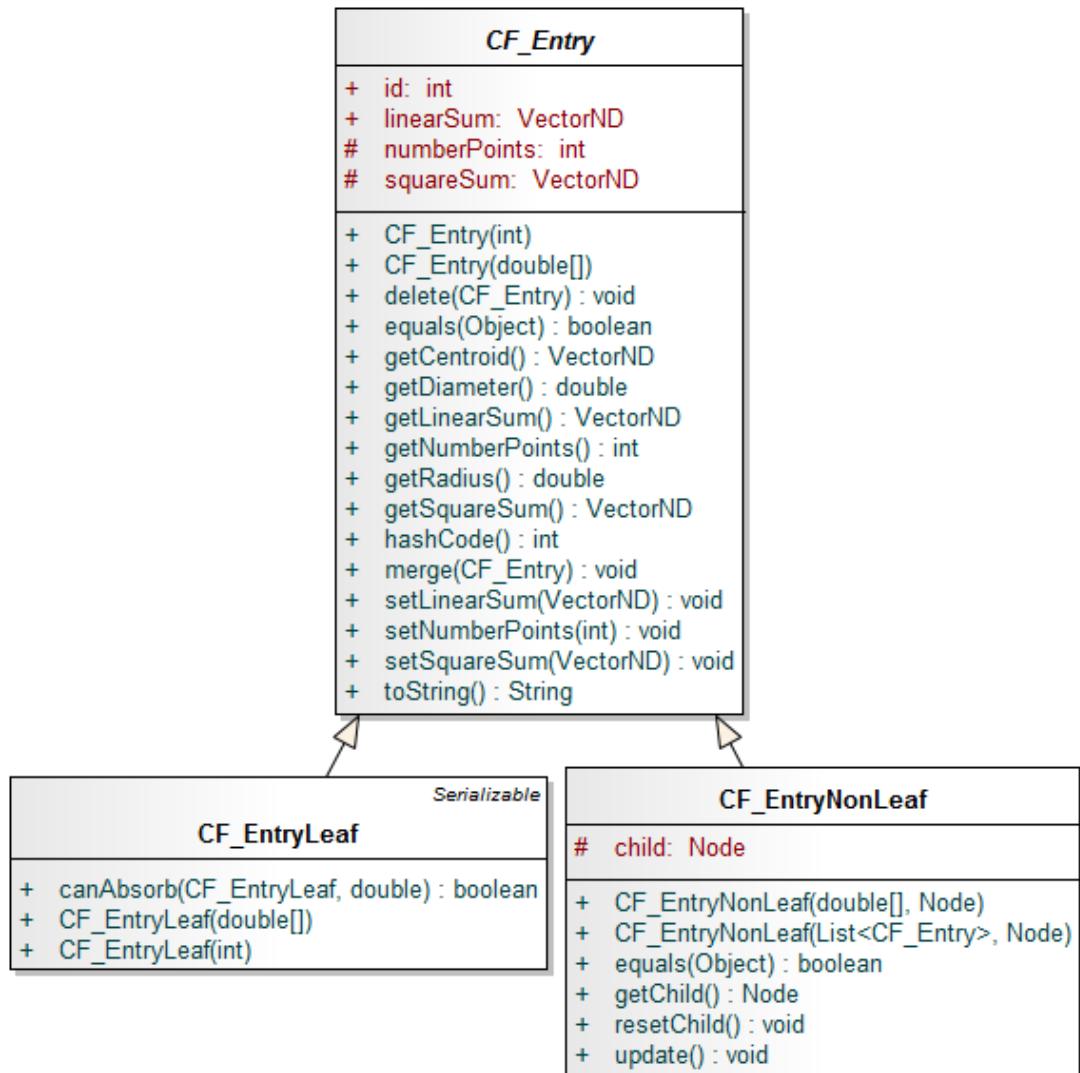
L’oggetto *clustering feature* è concettualizzato tramite la classe *CF_Entry*, la quale presenta la tripla che sintetizza le informazioni relative ai dati raggruppati nel *cluster*.

Si descrivono, di seguito, i membri della classe *CF_Entry* riportati nel Listato § [3.1 a pagina 122](#):

- l’intero *numberPoints* rappresenta il numero di punti appartenenti al *cluster*;
- l’oggetto *linearSum*, di tipo *VectorND*, è il vettore ottenuto dalla somma lineare di tutti i punti contenuti nel *cluster*;
- l’oggetto *squareSum*, di tipo *VectorND*, è il vettore ottenuto dalla somma quadratica di tutti i punti contenuti nel *cluster*;

⁶Un *package Java* è un meccanismo per organizzare classi *Java* all’interno di sottogruppi ordinati

Figura 3.1: Diagramma UML package “birch.cf_tree.cf_node.cf_entry”



Listato 3.1 Campi classe *CF_Entry*

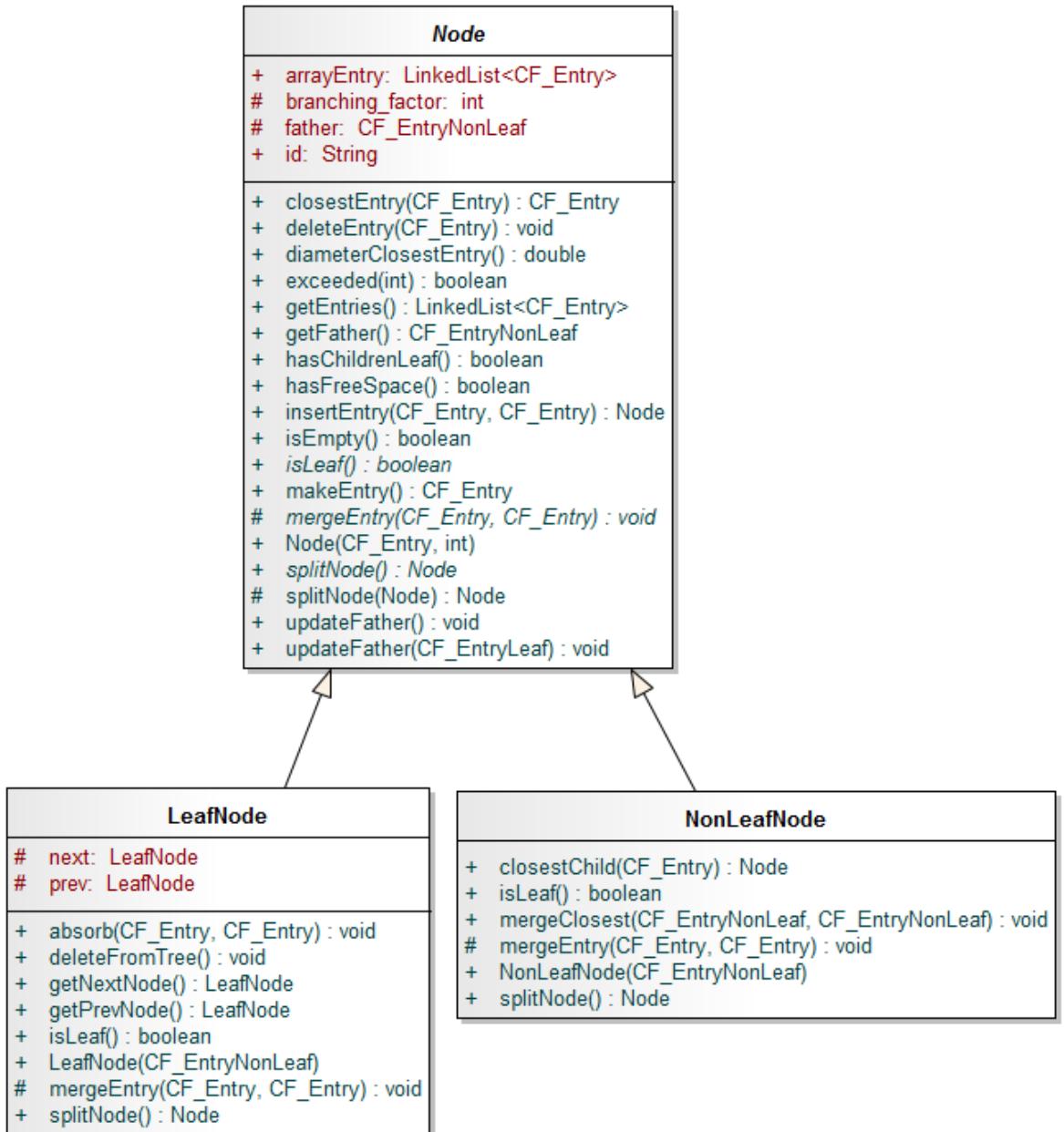
```
1 public abstract class CF_Entry {  
2  
3     // Numero di punti del cluster  
4     protected int numberPoints;  
5  
6     // Somma lineare dei punti del cluster  
7     protected VectorND linearSum;  
8  
9     // Somma quadratica dei punti del cluster  
10    protected VectorND squareSum;  
11  
12    ...  
13 }
```

La classe *CF_Entry* costituisce la classe base da cui vengono derivate, tramite relazione di ereditarietà, le sottoclassi *CF_EntryLeaf* e *CF_EntryNonLeaf*, rispettivamente *clustering feature* di nodi foglia e non foglia (nodi intermedi). Come illustrato nella Sezione § 2.1 a pagina 61, attraverso le informazioni contenute nella tripletta che rappresenta ciascun *CF*, è possibile ottenere indicazioni circa il centroide, il raggio o il diametro di ciascun *cluster*. La classe *CF_Entry* è naturalmente dotata di tutte le funzioni necessarie per calcolare le informazioni suddette, oltre alle funzioni che consentono di aggiornare le caratteristiche del *cluster*.

Sempre nel Listato § 3.1, è possibile notare come la classe *CF_Entry* sia stata definita astratta. L'utilizzo di tale tipologia di classe è motivato dal fatto che le classi figlie, anche se presenti in differenti nodi dell'albero, presentano funzionalità comuni; quindi si è deciso di estrarre tali funzionalità in una classe base astratta in modo da evitare la duplicazione del codice e facilitare la manutenzione dello stesso.

La classe *CF_EntryLeaf*, rappresenta un *CF* appartenente ad un nodo foglia. Essa estende la classe base con la sola definizione del metodo *canAbsorb()*, che testa la possibilità, da parte della *entry* in esame, di poter assorbire la *entry* passata come parametro. La classe *CF_EntryNonLeaf*, invece, estende la definizione base di *clustering feature* introducendo come ulteriore campo il riferimento al relativo nodo figlio.

Figura 3.2: Diagramma UML package “*birch.cf_tree.cf_node*”



Listato 3.2 Campi classe *Node*

```
1 public abstract class Node {  
2  
3     // Riferimento alla entry padre  
4     protected CF_EntryNonLeaf father;  
5  
6     // Entry presenti nel nodo  
7     protected LinkedList<CF_Entry> arrayEntry;  
8  
9     // Branching factor per il nodo  
10    protected int branching_factor;  
11  
12    // Identificativo della LeafNode  
13    public String id;  
14  
15    ...  
16 }
```

3.2.1.2 Classi package “birch.cf_tree.cf_node”

La classe *Node* realizza la componente “*nodo del CF-Tree*”. Nel Listato § 3.2 è riportato uno stralcio della sua interfaccia.

Si descrivono, di seguito, gli attributi della classe *Node* riportati nel Listato § 3.2:

- l’oggetto *father*, di tipo *CF_EntryNonLeaf*, è il riferimento alla *CF_Entry* padre del nodo in esame. Tale campo è stato adottato per accedere più velocemente al padre di un nodo senza dover scandire ogni volta l’intero albero.
- la struttura dati *arrayEntry*, di tipo *LinkedList<CF_Entry>*, si occupa di contenere tutti gli oggetti di tipo *CF_Entry* assegnati al nodo.
- l’attributo di tipo intero *branching factor*, indica il numero totale di *entry* che il nodo può contenere.
- la stringa *id* rappresenta l’identificatore di ciascun oggetto *Node* presente nell’albero, ovvero la posizione che ciascun nodo mantiene nel livello di appartenenza. La numerazione di ciascun nodo parte da zero e da sinistra verso destra.

Listato 3.3 Campi classe *LeafNode*

```
1 public class LeafNode extends Node {  
2  
3     // Riferimento al nodo foglia precedente  
4     protected LeafNode prev;  
5  
6     // Riferimento al nodo foglia successivo  
7     protected LeafNode next;  
8  
9     ...  
10 }
```

Come nel package “*birch.cf_tree.cf_node.cf_entry*”, anche in questo è stata adottata la tecnica della classe astratta, che implementa le funzionalità comuni delle sottoclassi *LeafNode* e *NonLeafNode*.

La classe *LeafNode*, adoperata per rappresentare un nodo a livello foglia, introduce due ulteriori attributi, come mostrato nel Listato § 3.3:

- il campo *prev*, riferimento al nodo di tipo *LeafNode* precedente;
- il campo *next*, riferimento al nodo di tipo *LeafNode* successivo.

Questi campi consentono una facile navigazione del livello foglia dell’albero.

La classe *NonLeafNode*, invece, è stata creata per rappresentare un nodo a livello non foglia; oltre ad implementare i metodi astratti della classe base, fornisce due ulteriori metodi:

- il metodo *mergeClosest()*, che consente di fonde la coppia di *entry* più vicine nel nodo, ad eccezione della coppia passata come parametro;
- il metodo *closestChild()*, che permette di recuperare il riferimento al nodo figlio più simile alla *entry* passata come parametro.

Questi metodi introdotti dalla classe *NonLeafNode* sono utilizzati nell’algoritmo di inserimento presente nella struttura dati *CF_Tree*.

Listato 3.4 Campi classe *CF_Tree*

```
1 public class CF_Tree {  
2  
3     // Soglia dell'albero  
4     private double threshold;  
5  
6     // Profondità dell'albero  
7     private int depth;  
8  
9     // Riferimento al nodo radice dell'albero  
10    private Node root;  
11  
12    // Flag di attivazione della gestione degli outlier  
13    private boolean outlier_handling_option;  
14  
15    ...  
16 }
```

3.2.1.3 Classi package “birch.cf_tree”

In questo *package* rientrano la classe *Outlier*, adibita alla gestione degli *outlier* individuati durante la fase di *rebuild* dell’albero, e la classe *CF_Tree*, che implementa la struttura dati ad albero usata nell’algoritmo *BIRCH*. Nel Listato § 3.4 si riportano gli attributi più importanti di tale struttura dati.

Il campo *root* memorizza il riferimento al nodo radice dell’albero; il campo *threshold* e *depth* memorizzano rispettivamente il valore della soglia e la profondità dell’albero in un determinato istante della costruzione.

Il campo booleano *outlier_handling_option* conserva il valore della chiave denominata “*outlier_handling_option*” presente nel *file* di configurazione. Il trattamento dei punti singolari viene eseguito solo se il campo in oggetto è impostato a “*true*”.

La classe *CF_Tree* possiede una serie di metodi che consentono di:

- inserire un punto *n*-dimensionale nell’albero;
- condensare l’albero (Fase 2 descritta nella Sezione § 2.3.2 a pagina 85);
- visitare l’albero in ampiezza;
- etichettare (*mapping*) i nodi che costituiscono i vari livelli dell’albero;

- individuare il *CF_EntryLeaf* a cui un determinato punto appartiene (*labeling*).

La classe *Outlier* ha il compito di gestire i punti singolari. Si presenta costituita da tre metodi statici:

- *isOutlier()*, verifica che la *CF_Entry* in esame sia un punto singolare;
- *saveOutlier()*, ha il compito di salvare il punto singolare in una tabella del *database* di nome *outliers*, costruita per contenere, appunto, i potenziali *outlier* del *dataset*;
- *tryToAbsorb()*, tenta il reinserimento dei potenziali *outlier* nella struttura gerarchica.

3.2.1.4 Classe DBSCAN

La presente classe si prefigge il compito di eseguire l'algoritmo di *DBSCAN* sui centroidi dei cluster prodotti dalla fase di *preclustering*. Alla fine di quest'ultima, i centroidi di tutti i nodi presenti in ciascun livello del *CF-Tree*, vengono salvati in un *file ARFF*. In questo modo si consente di processare i *file ARFF* in maniera asincrona, dando cioè, la possibilità all'utilizzatore di scegliere quando applicare l'algoritmo *DBSCAN*. Non solo, è anche possibile variare i parametri del *DBSCAN*, e quindi rieseguirlo, senza riprocessare l'intero *dataset*.

L'algoritmo di *DBSCAN* utilizzato è quello presente nel progetto *Weka*: esso processa i dati presenti in un *file ARFF*. Per questo motivo è stato scelto questo formato per salvare i centroidi dei *cluster* ottenuti in ciascun livello della struttura gerarchica.

Un *file* scelto per contenere i dati che verranno processati dal *DBSCAN* è il *file ARFF*; un *file ARFF* (*Attribute-Relation File Format*) è un *file* di testo *ASCII* che descrive una lista di istanze che condividono un insieme di attributi. Esso presenta due distinte sezioni. La prima è detta *Header information* e contiene:

1. il significato che ciascun punto rappresenta;
2. una lista di attributi (le colonne del *dataset*);
3. i tipi di ciascun attributo presente nella lista suddetta.

La seconda sezione è detta *Data information* e contiene le coordinate dei vari punti che il *DBSCAN* deve processare.

Per approfondimenti in merito al file ARFF si rimanda al manuale di Weka, alla sezione riguardante i dati⁷.

3.2.2 Classi di utilità

In questa categoria rientrano tutte le classi che non compongono concettualmente il progetto, ma che forniscono le strutture dati utili per la sua implementazione.

3.2.2.1 Classe *VectorND*

La classe *VectorND* rappresenta un vettore n -dimensionale. Si è scelto di utilizzare questa struttura per poter rappresentare un punto in uno spazio n -dimensionale (*punto* $\in \mathbb{R}^n$). Un’istanza della classe *VectorND* viene creata impostando la dimensione n , la quale si prevede subisca successive modifiche.

3.2.2.2 Classe *LoadProps*

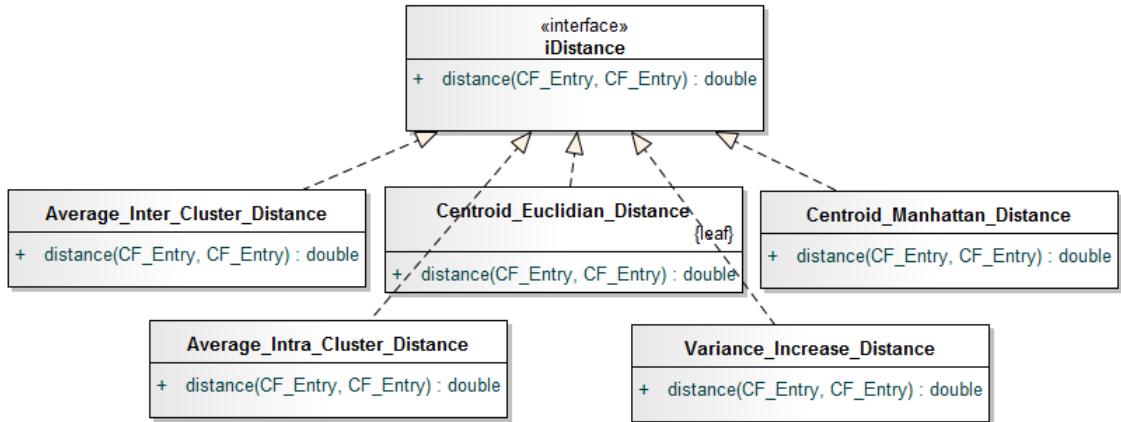
La classe *LoadProps* è stata sviluppata allo scopo di effettuare il caricamento in memoria del *file* di configurazione. Essa presenta due metodi statici di caricamento, a seconda della tipologia di *file* adottato:

- *getProperties()*, utilizzato per il caricamento di un *file* di testo contenente la serializzazione di un’istanza della classe “*java.util.Properties*”, ossia una lista di proprietà (coppia chiave-valore);
- *getPropertiesFromXML()*, ideato per il caricamento di un *file XML* contenente la serializzazione di un’istanza della classe “*java.util.Properties*” sotto forma di struttura *XML*.

Nel progetto in esame, è stato utilizzato il caricamento da *file* di testo nel contesto del caricamento delle impostazioni necessarie dal *file* di configurazione “*config.ini*”. Tale *file* può essere editato al fine di modificare i parametri di connessione con il *database* e i parametri di ***OLAPBIRCH***.

⁷<http://freefr.dl.sourceforge.net/project/weka/documentation/3.6.x/WekaManual-3-6-4.pdf> pagina 161

Figura 3.3: Diagramma UML package “*birch.util.distance*”



3.2.2.3 Classe *Params*

La classe *Params* memorizza al proprio interno tutti i parametri di configurazione dell’algoritmo ***OLAPBIRCH***, assieme ai dati di connessione con il *database*. Tali parametri sono ottenuti tramite caricamento del file di configurazione, presente nel *software*, attraverso la classe *LoadProps*. La classe *Params* risulta accessibile a tutte le classi del progetto: ogni classe può quindi agevolmente accedere in ogni istante ai valori dei parametri di configurazione. La scelta che ha portato alla creazione di tale classe è stata quella di creare una comoda raccolta di parametri visibile e accessibile da tutte le altre classi.

3.2.3 Classi per il calcolo della misura di similarità

In questa categoria rientrano le classi racchiuse nel package “*birch.util.distance*”. In questo package sono presenti le classi relative al calcolo della misura di distanza. Affinché l’algoritmo possa determinare la vicinanza tra due *cluster*, è necessario definire la misura di distanza da adoperare. Nel progetto sono state implementate tutte le misure definite in [ZRL95] e mostrate nella Sezione § 2.1.2 a pagina 63.

È possibile, comunque, utilizzare tutte le misure di distanza che si ritengono più appropriate per una determinata tipologia di dati da analizzare: questo è consentito dalla presenza dell’interfaccia denominata *iDistance*.

La struttura del package “*birch.util.distance*” è mostrata nella Figura § 3.3.

3.2.4 Classi di interazione *database*

In questa categoria rientrano le classi racchiuse nel package “*birch.util.database*”, ossia le classi necessarie all’accesso e all’interrogazione del *database*. Le classi implementate rientrano nel progetto di realizzazione del *pattern Data Access Objects (DAO)*.

Nella Figura § 3.4 nella pagina successiva è mostrato il diagramma *UML* relativo al package “*birch.util.database*”.

3.2.4.1 Classe *Alias*

La classe *Alias* presenta al suo interno il nome della tabella, e i relativi attributi, riservata per il salvataggio dei *cluster* considerati potenziali *outlier*. È utilizzata come classe di auxilio nella composizione delle *query* e, grazie ad essa, i cambiamenti apportati al *database* (quali il cambio dei nomi degli attributi o della tabella) non si ripercuotono sulle *query*, che rimangono inalterate.

Nel progetto, la classe presenta il nome della tabella OUTLIERS, adibita al salvataggio dei punti singolari, e relativi attributi:

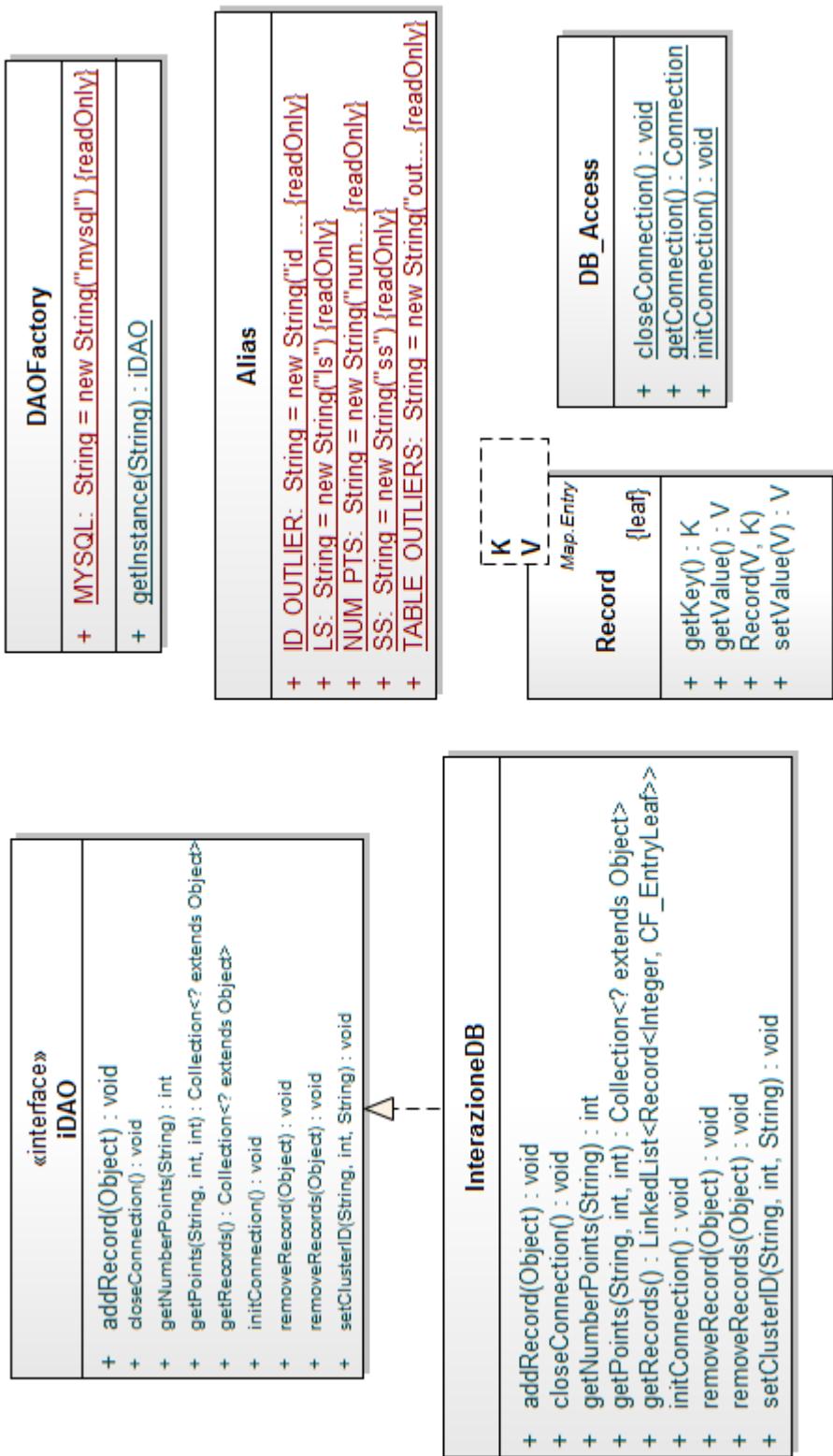
- ID_OUTLIER, identificativo univoco del punto singolare, nonché chiave primaria della tabella.
- NUM PTS, numero di punti che costituiscono il *cluster* identificato come rumore.
- LS, la somma lineare del cluster.
- SS, la somma quadratica del cluster.

3.2.4.2 Classe *DB_Access*

La classe *DB_Access* ha il compito di gestire la connessione alla base di dati, consentendo l’inizializzazione e la chiusura della connessione. I parametri di connessione al *database* sono caricati direttamente dal file di configurazione dalla classe *Params*, già descritta in precedenza.

Questa soluzione consente il massimo riuso di tale classe, permettendole di interfacciarsi facilmente con qualunque *database*: il cambiamento dei dati di accesso, salvati nel file “*config.ini*”, non comporta alcuna modifica nel codice sorgente della classe che, perciò, non necessita di essere ricompilata.

Figura 3.4: Diagramma UML package “birch.util.database”



I metodi implementati nella classe sono:

- *initConnection()*, provvede al caricamento del *driver* ed all'inizializzazione della connessione.
- *getConnection()*, restituisce la connessione effettuata alla base di dati.
- *closeConnection()*, permette la chiusura della connessione.

3.2.4.3 Classe *DAOFactory*

La classe *DAOFactory* consente la creazione di istanze di *iDAO* a seconda del *DBMS* utilizzato; per far questo implementa il *design pattern*⁸ *Factory method*. *DAOFactory* consente l'istanziazione di un'istanza di accesso ai dati senza tener conto dell'implementazione concreta che verrà utilizzata. La classe risulta, nel nostro caso, molto semplice poiché tiene conto di una sola fonte di dati da collegare, ma con delle semplici modifiche è possibile ampliare il numero di *database* con cui interagire. Grazie al *pattern*, si permette di ottenere il riferimento ad una implementazione *DAO*, consentendo di utilizzare esclusivamente i metodi definiti per accedere ai dati.

La classe si presenta costituita da un solo metodo, il metodo *getInstance()*, che consente di ottenere una classe di comunicazione col *database*, passando come parametro il nome del *DBMS*. I nomi dei *DBMS* con cui è possibile collegarsi sono definiti nella classe stessa sotto forma di costanti.

Nel nostro progetto, vista la sola interazione con un *DBMS*, è stata prevista la definizione di una sola costante che consenta il recupero di una implementazione *DAO* che interagisca con il *DBMS MySQL*.

3.2.4.4 Interfaccia *iDAO*

L'interfaccia *iDAO* definisce i metodi utilizzabili interrogare il *database*. Nel contempo, permette alle classi che la implementano l'utilizzo di una logica completamente propria per la selezione, la manipolazione e l'amministrazione dei dati. Tale scelta offre due grandi vantaggi:

⁸Il *Design Pattern* (*schemma di progettazione*) può essere definito come "una soluzione progettuale generale a un problema ricorrente". Esso non è una libreria o un componente di software riusabile, quanto piuttosto una descrizione o un modello da applicare per risolvere un problema che può presentarsi in diverse situazioni durante la progettazione e lo sviluppo del software.

- nell'applicazione si lavora solo con l'interfaccia *DAO*: l'implementazione concreta rimane nascosta. Ciò implica una manutenzione dell'applicazione più facile, poiché la classe di accesso ai dati può essere modificata senza che il codice applicativo debba essere aggiornato.
- gli accessi al *database* sono mediati da una classe centrale. Se il modello di dati nel *database* cambia, l'adattamento riguarderà soltanto questa classe centrale: le chiamate ai metodi nei vari elementi dell'applicazione non richiederanno modifiche.

Esistono quindi delle valide ragioni che giustificano l'adozione di una interfaccia di accesso ai dati, dal momento che i vantaggi che ne derivano sono importanti:

- netta separazione tra dati e applicazioni;
- manutenzione delle applicazioni semplificata;
- implementazione dei metodi facilitata;
- migliore protezione dei dati, grazie al loro trattamento centralizzato;
- maggiore flessibilità dell'applicazione, in particolare per quanto riguarda il supporto ai dati e i tipi di *database* utilizzabili.

3.2.4.5 Classe *InterazioneDB*

InterazioneDB si occupa dell'implementazione dell'interfaccia *iDAO* e quindi dell'effettiva interazione con il database.

Per poter inserire e richiamare i *record* del *database*, ci si è serviti dell'interfaccia *PreparedStatement*, per la creazione di dichiarazioni *SQL* con parametri. La scelta di tale interfaccia è motivata dalle seguenti ragioni:

- *sicurezza*: i *PreparedStatement* offrono maggior sicurezza rispetto alle implementazioni *Statement*, in quanto di norma sono in grado di proteggere efficacemente il sistema dagli attacchi di tipo *SQL Injection*⁹.

⁹Tecnica dell'*hacking* mirata a colpire le applicazioni *web* che si appoggiano su un *database* di tipo *SQL*. Questo *exploit* sfrutta l'inefficienza dei controlli sui dati ricevuti in *input* ed inserisce codice maligno all'interno di una *query SQL*. Le conseguenze prodotte sono imprevedibili per il programmatore: l'*Sql Injection* permette al malintenzionato di autenticarsi con ampi privilegi in aree protette del sito (ovviamente, anche senza essere in possesso delle credenziali d'accesso) e di visualizzare e/o alterare dati sensibili.

- *prestazioni*: i *PreparedStatement* vengono eseguiti in modo più rapido perché le *query* sono precompilate in funzione del tipo di database e i parametri inviati separatamente.
- *leggibilità*: le classi che usano tale interfaccia sono più leggibili, poiché l’assegnazione dei valori avviene sotto forma di diverse chiamate di metodo.
- *approccio typesafe*: dal momento che i dati sono assegnati mediante metodi *Java*, è evidente che possono essere utilizzati soltanto i tipi di dato accettati dai metodi.

Capitolo 4

Modifiche apportate all'algoritmo *BIRCH*

*Il vero viaggio di scoperta non consiste nel cercare nuove terre,
ma nell'avere nuovi occhi.*
(Voltaire)

NEL presente capitolo saranno evidenziati gli aspetti per i quali la nostra realizzazione si differenzia maggiormente dall'algoritmo originale, illustrato in [ZRL96] (Capitolo § 2 a pagina 61).

4.1 Motivazioni alla base delle modifiche

Come già detto nella Sezione § 1.5.4.2 a pagina 59, *BIRCH* non è esente da problemi.

Identifica correttamente solo cluster sferici: esso, infatti, utilizza il concetto di *raggio* e *diametro* per controllare il confine di un *cluster*, sia nella fase di *pre-clustering*, che nella fase di *clustering* globale.

Lavora bene solo su dataset a bassa dimensionalità: in [AGGR98] sono stati testati gli algoritmi gerarchici su un *dataset* con dimensionalità variabile tra 5 e 50 dimensioni. Con più di cinque dimensioni *BIRCH* non è capace di identificare

cluster reali. Esso, infatti, con l'aumento della dimensionalità del *dataset*, nel calcolare la distanza tra due punti considera di uguale importanza tutte le dimensioni.

Le ulteriori motivazioni che ci hanno spinto a modificare l'idea originale del *BIRCH* sono state:

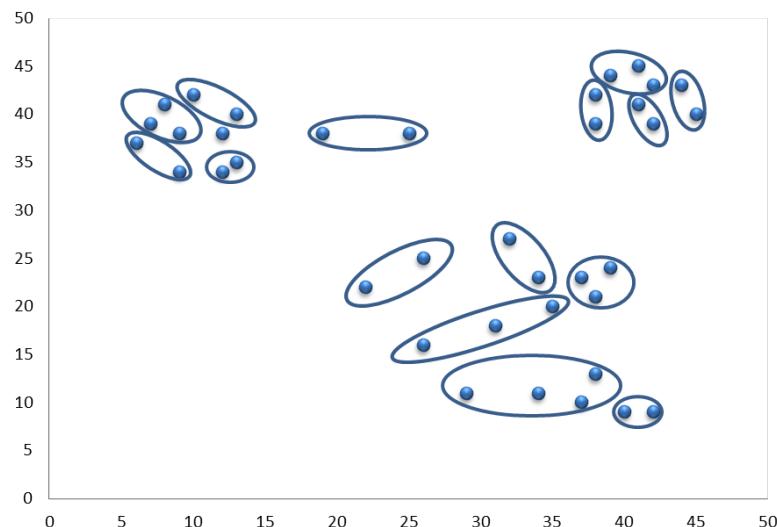
- la necessità di integrare l'algoritmo in oggetto con un *OLAP Data Warehouse*;
- il linguaggio di programmazione utilizzato per l'implementazione dell'algoritmo.

Poiché si è utilizzato JAVA, e in questo linguaggio di programmazione la gestione della memoria *Heap* è totalmente trasparente per il programmatore, non è stato possibile calcolare i fattori di ramificazione *B* e *L*, dipendenti dalla pagina di memoria.

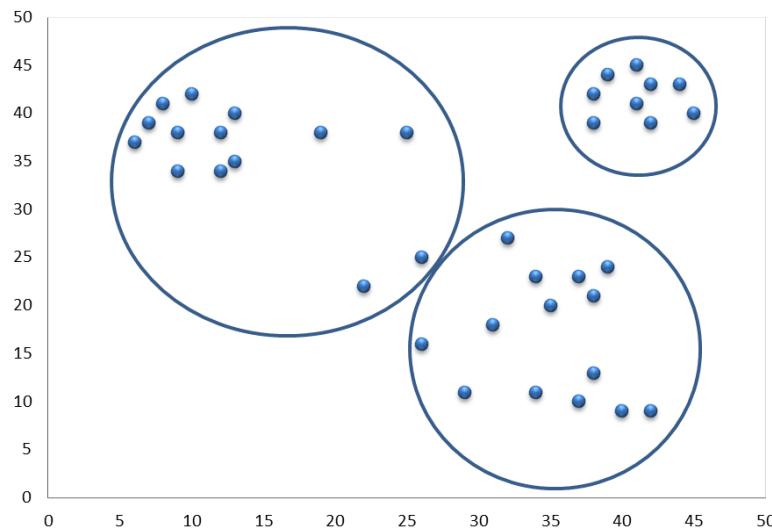
4.2 Sensibilità alla forma sferica dei *cluster*

Osservando i *cluster* ottenuti dalla fase di *pre-clustering* del *BIRCH*, è possibile notare che i raggruppamenti ottenuti sono molto piccoli e densi. I *cluster* prodotti dalla fase di *clustering* globale, se l'algoritmo utilizzato è di tipo *centroid-based*¹ (come accade in *BIRCH*), hanno una granularità maggiore, poiché raggruppano i *sotto-cluster* ottenuti dalla fase preliminare, e una forma sferica. Tuttavia, nei dataset reali, non sempre i raggruppamenti sono di forma ipersferica. La maggior parte delle volte i *cluster* sono di forma arbitraria ed è plausibile che i centroidi dei suoi *sotto-cluster* siano anche molto lontani tra di loro: questo può causare la divisione del *cluster*.

Figura 4.1: *Sotto-Cluster* Prodotti dalla Fase 1



¹ prende in considerazione i centroidi dei *cluster* come punti rappresentativi

Figura 4.2: *Cluster* prodotti dalla Fase 3 con algoritmo *centroid-based*

Come già detto, gli algoritmi che producono *cluster* di migliore qualità, senza specificare il numero dei raggruppamenti desiderati (che normalmente non si conosce), sono quelli basati sulla densità. Il *DBSCAN*, per esempio, produce *cluster* di ottima qualità, ma utilizza un approccio *all-points*²: quest’altro “estremo” presenta l’inconveniente di rendere l’algoritmo computazionalmente complesso. Ciò è dovuto al fatto che in *DBSCAN* non è prevista una fase di *pre-clustering*, che quindi non lo rende adatto per computazioni su grandi *databases*.

L’idea di base dalla quale siamo partiti è quella di utilizzare l’algoritmo *BIRCH* come fase di *pre-clustering* per il *DBSCAN*: il *BIRCH* raggruppa i dati in arrivo, incrementalmente, nel *CF-Tree*; quando è necessario consultare le informazioni contenute nell’albero, per migliorare la qualità dei *cluster*, si utilizza il *DBSCAN* o qualsiasi altro algoritmo basato sulla densità, utilizzando i centroidi dei *cluster*, presenti nell’ultimo livello della gerarchia, come punti rappresentativi del *DBSCAN*.

²prende in considerazione tutti i punti del *cluster* come punti rappresentativi

Figura 4.3: Centroidi dei *cluster* prodotti dalla Fase 1

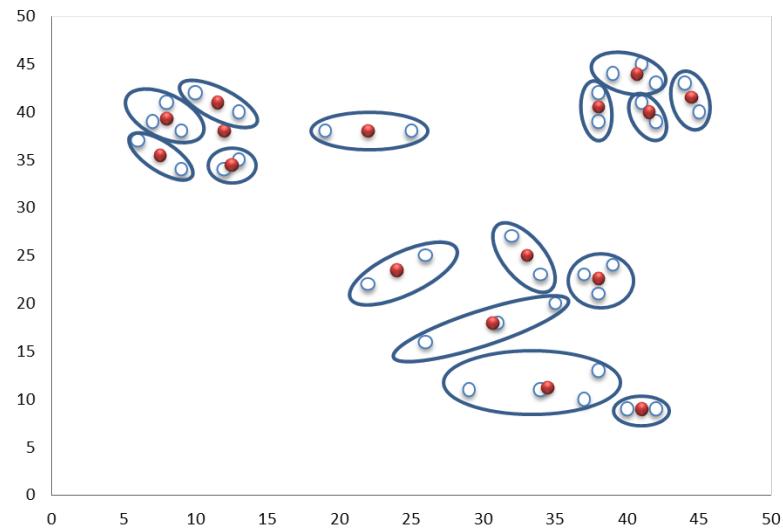
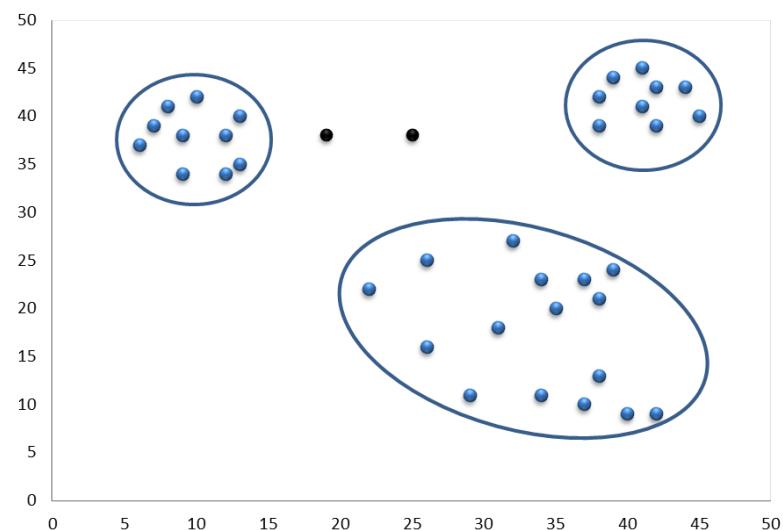


Figura 4.4: *Cluster* prodotti dal *DBSCAN* a partire dai centroidi dei *cluster* prodotti dalla Fase 1

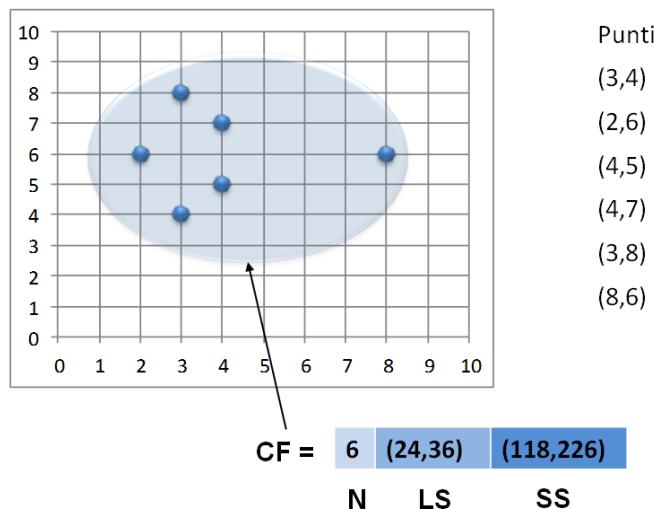


Come è possibile notare nella Figura § 4.4 nella pagina precedente, utilizzando il *DB-SCAN* sui centroidi dei raggruppamenti prodotti dalla Fase 1, vengono identificati anche *cluster* di forma non perfettamente sferica.

4.3 Sensibilità alle dimensioni del dataset

Nella definizione di *CF*, prevista dall'algoritmo originale, la somma quadratica (*SS*) è un valore scalare che collassa la somma quadratica su tutte le dimensioni. Con l'aumento della dimensionalità dello spazio dei dati, la qualità dei *cluster* prodotti da *BIRCH* degenera. Per risolvere tale problema abbiamo deciso di rendere la somma quadratica come un valore vettoriale che mantenga la somma quadratica di tutte le dimensioni che compongono lo spazio vettoriale, per meglio descrivere la distribuzione dei dati.

Figura 4.5: Rappresentazione del *CF* in *OLAPBIRCH*



4.4 Integrazione con tecnologie OLAP

L'obiettivo della presente tesi è quello di integrare l'algoritmo di *clustering BIRCH* con le tecnologie *OLAP*: è stato necessario, per cui, potenziare le caratteristiche principali dell'albero dei *CF*.

Costruito un *CF-Tree*, le uniche informazioni accessibili all'utilizzatore sono quelle relative all'ultimo livello della gerarchia³, poiché i nodi di quest'ultimo sono collegati tra loro da riferimenti incrociati. La caratteristica principale di un cubo *OLAP* è quella di prevedere operazioni di *roll-up* e *drill-down* sulle dimensioni del cubo, notoriamente organizzate all'interno di una gerarchia. È necessario, perciò, che la gerarchia contenente le informazioni sui *cluster*, consenta di visitare ogni suo livello: in tal modo si permette al motore *OLAP* di scegliere automaticamente il livello di granularità dei *cluster*.

Per dar seguito alle esigenze suddette, abbiamo previsto nel nostro sistema l'implementazione delle funzioni che andiamo di seguito a descrivere.

1. La codifica di un file *XML*, che descrive lo schema multidimensionale di una base di dati al motore *OLAP*, al fine di reperire informazioni riguardanti gli attributi e le dimensioni di interesse, sul quale effettuare operazioni di *clustering*.
2. Il *rebuild* dell'albero viene effettuato, non più in base alla percentuale di memoria libera per il sistema, ma rispetto alla profondità massima della gerarchia, specificata nel file *XML*. Il motore *OLAP*, infatti, deve conoscere a priori il massimo livello raggiungibile dall'albero, per poter operare adeguatamente interrogazioni sullo stesso.
3. Le interrogazioni che coinvolgono il *CF-Tree*, possono riguardare ciascun livello componente l'albero, e non solo, quindi, l'ultimo livello: è stato dotato il *CF-Tree* di un meccanismo di visita in ampiezza che consente di ottenere informazioni su tutti i nodi che compongono un livello specificato. Di conseguenza, anche l'algoritmo di *DBSCAN* è applicabile a qualsiasi livello della gerarchia.
4. Il *DBSCAN* raggruppa in base alla situazione dell'albero in un particolare istante: per poter dunque conoscere i punti che compongono ciascun *cluster* risultante dall'esito

³i nodi che lo compongono e i *CF-Entry* che compongono ciascun nodo

del DBSCAN è necessario effettuare prima il *mapping* dell'albero, operazione che consente di identificare univocamente i nodi dell'albero per livello.

Tutti i nodi che compongono ciascun livello dell'albero sono numerati da 0 ad n . Abbiamo scelto questo tipo di numerazione dei nodi, perché l'ordine con il quale il DBSCAN presenta l'appartenenza di ciascun nodo ad un determinato *cluster*, rispetta l'ordine di *input* dei dati; per cui dire ad esempio che i nodi 0, 3, 8 e 19 appartengono ad un determinato *cluster*, equivale a dire che i nodi che si trovano nelle rispettive posizioni del livello specificato compongono uno stesso raggruppamento. Questo facilita il ritrovamento dei dati che compongono ciascun nodo.

5. Dopo la funzione di mappatura dell'albero, si effettua l'operazione di etichettatura dei dati (Fase 4). Per evitare di mantenere in memoria tutti i punti interessati dal *clustering*, e per consentire interrogazioni sui dati raggruppati in diversi momenti, senza dover ripetere l'operazione di *labeling* anche se non vi è stata aggiunta di nuovi dati nel database, abbiamo deciso di etichettare i dati direttamente nel database. Tutte le dimensioni sul quale si intende effettuare operazioni di *clustering* devono quindi possedere un campo “*clusterID*”. In questo campo viene salvata la stringa che identifica l'appartenenza di ciascun dato a ciascun nodo della gerarchia. In altre parole, questa stringa contiene il percorso che il dato ha effettuato dell'albero per essere etichettato. La stringa in questione presenta il formato che ci accingiamo a descrivere mediante un'espressione regolare.

0; ([0 – 9]+;){0 – numero livelli intermedi}[0 – 9]+

La posizione di un numero nella stringa dopo un punto e virgola, identifica il livello dell'albero. I punti e vergola, invece, specificano la profondità dell'albero.

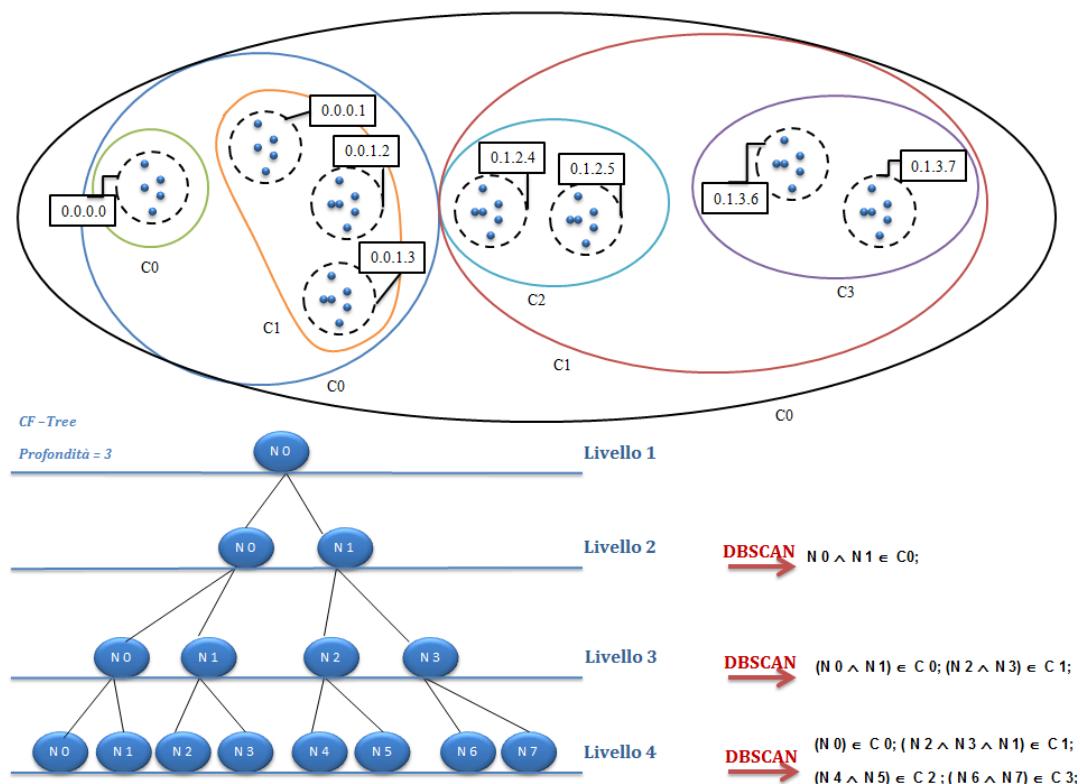
Il primo numero rappresenta il nodo *root*, che essendo unico nel suo livello, ha numero 0. Segue un punto e virgola e per ogni livello intermedio, il numero del nodo a cui il dato appartiene in quel determinato livello della gerarchia e un punto e virgola che separa dunque i vari livelli. Se non ci sono livelli intermedi (numero livelli = 2) è specificato direttamente l'identificativo del nodo nodo foglia del *CF-Tree*, che ad ogni modo conclude sempre questa tipologia di stringhe.

Questo tipo di stringhe, se pur dalla composizione macchinosa, consente di identificare facilmente nel *database* i punti che compongono ciascun nodo a qualsiasi livello della gerarchia.

6. L'algoritmo di clustering globale *DBSCAN*, il *mapping* dell'albero e il *labeling* sono funzioni opzionali, nel senso che non vengono eseguite per ogni inserzione di un punto nell'albero, ma solo quando richiesta dal motore *OLAP*. Le operazioni di *mapping* del *CF-Tree* e di *labeling* sono inscindibili e necessarie per effettuare interrogazioni sulla struttura. Nulla vieta, però, che, una volta eseguite le operazioni di *mapping* e di *labeling*, non si possa interrogare direttamente la struttura gerarchica senza effettuare il *DBSCAN*: la Fase 3, che nel *BIRCH* era obbligatoria, nel nostro sistema è dunque opzionale, lasciando così libertà all'utilizzatore di scegliere il grado di dettaglio delle query.

Di seguito mostriamo una rappresentazione delle nuove funzioni previste dal nostro sistema.

Figura 4.6: Panoramica delle nuove funzioni previste dal nostro sistema



Capitolo 5

Uso di *BIRCH* in un sistema *OLAP*

*Il vero viaggio di scoperta non consiste nel cercare nuove terre,
ma nell'avere nuovi occhi.*
VOLTAIRE

NEL presente capitolo saranno evidenziati gli aspetti che riguardano l'integrazione del nostro sistema con le tecnologie *OLAP*.

5.1 Integrazione di *BIRCH* con le tecnologie *OLAP*

L'obiettivo della presente tesi è quello di integrare l'algoritmo di *clustering BIRCH* con le tecnologie *OLAP*: è stato necessario, per cui, potenziare le caratteristiche principali dell'albero dei *CF*.

Costruito un *CF-Tree*, le uniche informazioni accessibili all'utilizzatore sono quelle relative all'ultimo livello della gerarchia¹, poiché i nodi di quest'ultimo sono collegati tra loro da riferimenti incrociati. La caratteristica principale di un cubo *OLAP* è quella di prevedere operazioni di *roll-up* e *drill-down* sulle dimensioni del cubo, notoriamente organizzate all'interno di una gerarchia. È necessario, perciò, che la gerarchia contenente le informazioni sui *cluster*, consenta di visitare ogni suo livello: in tal modo si permette al motore *OLAP* di scegliere automaticamente il livello di granularità dei *cluster*.

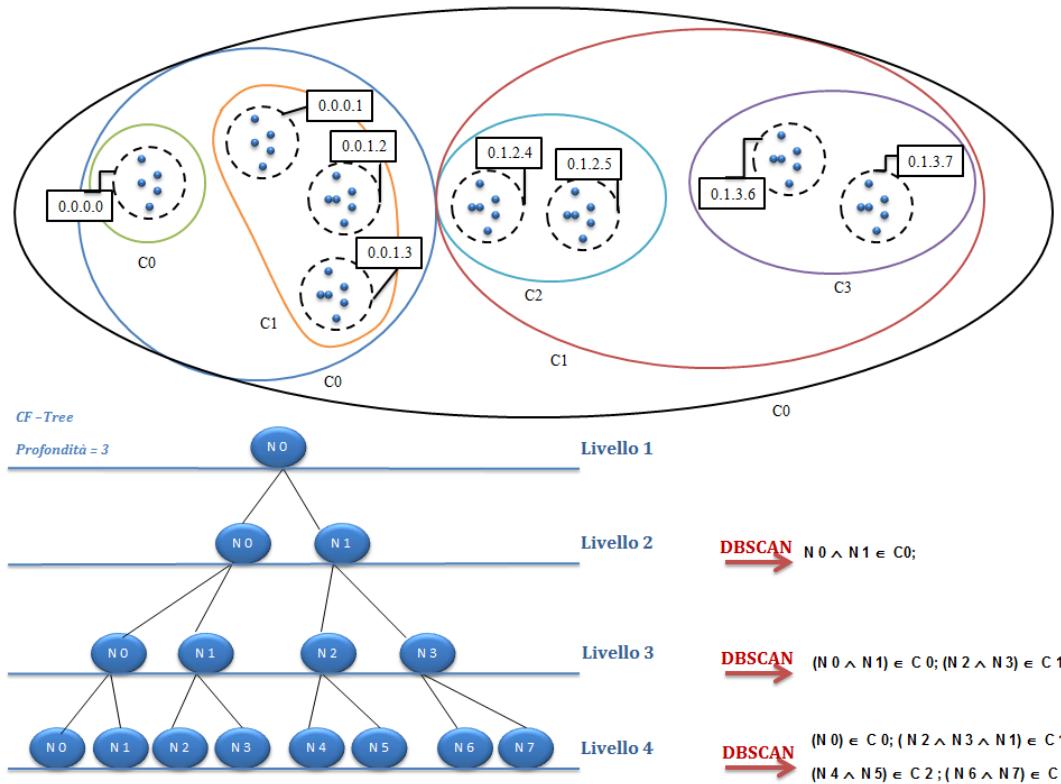
¹i nodi che lo compongono e i *CF-Entry Leaf* che compongono ciascun nodo

Per dar seguito alle esigenze suddette, abbiamo previsto nel nostro sistema l'implementazione delle funzioni che andiamo di seguito a descrivere.

1. La codifica di un file *XML*, che descrive lo schema multidimensionale di una base di dati al motore *OLAP*, al fine di reperire informazioni riguardanti gli attributi e le dimensioni di interesse, sul quale effettuare operazioni di *clustering*.
2. Il *rebuild* dell'albero viene effettuato, non più in base alla percentuale di memoria libera per il sistema, ma rispetto alla profondità massima della gerarchia, specificata nel file *XML*. Il motore *OLAP*, infatti, deve conoscere a priori il massimo livello raggiungibile dall'albero, per poter operare adeguatamente interrogazioni sullo stesso.
3. Le interrogazioni che coinvolgono il *CF-Tree*, possono riguardare ciascun livello componente l'albero, e non solo, quindi, l'ultimo livello: è stato dotato il *CF-Tree* di un meccanismo di visita in ampiezza che consente di ottenere informazioni su tutti i nodi che compongono un livello specificato. Di conseguenza, anche l'algoritmo di *DBSCAN* è applicabile a qualsiasi livello della gerarchia.
4. Il *DBSCAN* raggruppa in base alla situazione dell'albero in un particolare istante: per poter, dunque, conoscere i punti che compongono ciascun *cluster* risultante dall'esito del *DBSCAN* è necessario effettuare prima il *mapping* dell'albero, operazione che consente di identificare univocamente i nodi dell'albero in ciascun livello. Segue, infine, l'operazione di *labeling* dei dati.
5. L'algoritmo di clustering globale *DBSCAN*, il *mapping* dell'albero e il *labeling* sono funzioni opzionali, nel senso che non vengono eseguite per ogni inserzione di un punto nell'albero, ma solo quando richiesta dal motore *OLAP*. Le operazioni di *mapping* del *CF-Tree* e di *labeling* sono inscindibili e necessarie per effettuare interrogazioni sulla struttura. Nulla vieta, però, che, una volta eseguite le operazioni di *mapping* e di *labeling*, non si possa interrogare direttamente la struttura gerarchica senza effettuare il *DBSCAN*.

Di seguito mostriamo una rappresentazione delle nuove funzioni previste dal nostro sistema.

Figura 5.1: Panoramica delle nuove funzioni previste dal nostro sistema



5.2 Il file XML *Mondrian*

La tecnologia *OLAP* prevede un insieme di tecniche software per l’analisi interattiva e veloce di grandi quantità di dati, che è possibile esaminare in modalità piuttosto complesse. La creazione di un database *OLAP* consiste nell’effettuare una “istantanea” di informazioni in un determinato momento e trasformare queste singole informazioni in dati multidimensionali. Una struttura *OLAP* creata per questo scopo è chiamata cubo multidimensionale.

Pentaho Mondrian Analysis è un motore *OLAP* sviluppato in *Java*. Implementa le funzionalità indispensabili all’analisi dati (aggregazione, drill-down drill-through, slicing, dicing) ed è in grado di eseguire *query* leggendo i dati da un *RDBMS* e presentando i risultati in forma multidimensionale per mezzo di *API Java*. La connessione alla base di dati di *Data Warehouse* avviene via *JDBC*, il che rende indipendente *Mondrian* dal

particolare *RDBMS* utilizzato. Lo schema multidimensionale della base dati può essere sia a stella che a fiocco di neve, e la sua descrizione viene fornita al motore sotto forma di un file *XML*.

Il file XML contiene uno schema che concettualizza il modello logico di un database multidimensionale, costituito da cubi, gerarchie e membri, e le relazioni tra questo modello e quello fisico dei dati. Gli elementi principali di uno schema sono la sorgente dati, i cubi, le misure e le dimensioni.

- il cubo che identifica il modello logico di un database multidimensionale;
- una sorgente di dati che identifica e connette un cubo ad un database dove è presente l'informazione;
- le dimensioni, attributi strutturali formati da una o più gerarchie organizzate in livelli che l'utente utilizza come base per l'analisi dei dati; (una dimensione geografica, ad esempio, può includere una gerarchia con livelli per il paese o l'area, lo stato o la provincia e la città; una dimensione temporale può includere una gerarchia con livelli per l'anno, il trimestre, il mese e il giorno);
- le misure, un insieme di valori basati su una colonna della tabella dei fatti e costituiti in genere da valori numerici. Le misure sono i valori centrali del cubo che vengono elaborati, aggregati e analizzati (ad esempio i valori delle vendite, dei profitti, dei ricavi, dei costi, ecc...);

Per definire un Data Warehouse attraverso il linguaggio *XML*, *Mondrian* sfrutta la sua natura “gerarchica” utilizzando *tag* appropriati per descrivere la composizione del cubo *OLAP* (*Mondrian DTD*²). L'elemento con il quale si definisce il *Data Warehouse* è lo schema, composto da uno o più cubi.

Di seguito viene mostrato un esempio di file *XML Mondrian*, creato attraverso il *tool* grafico *Schema WorkBench*³, applicato ad un benchmark⁴ di supporto alle decisioni pro-

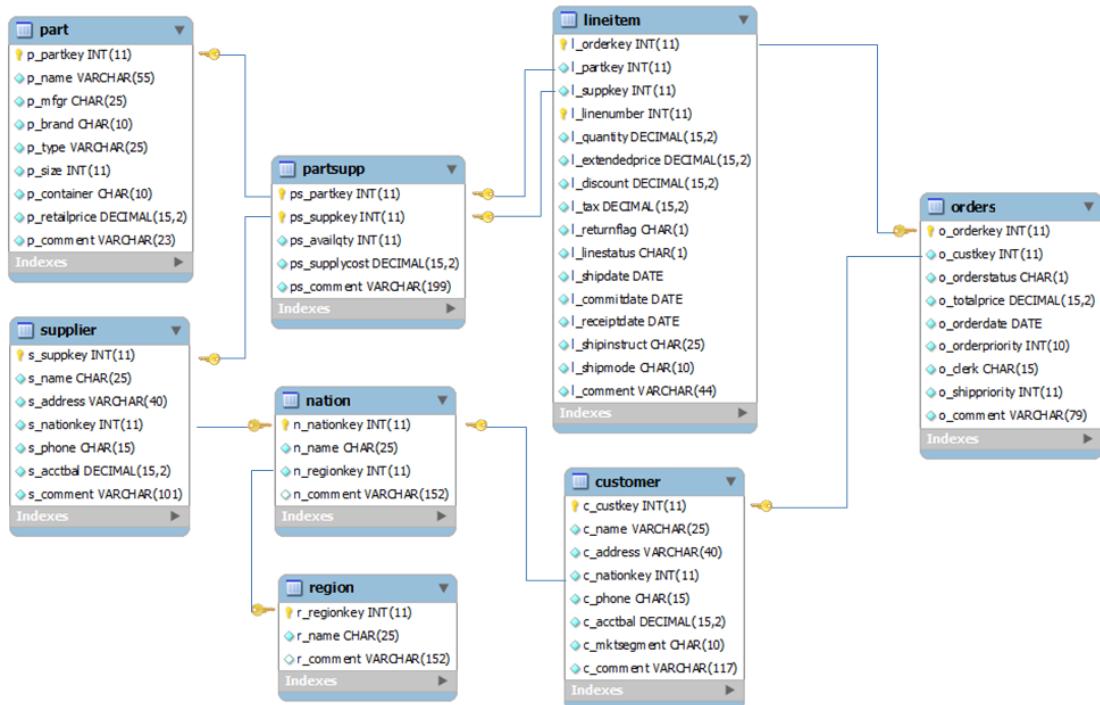
²<http://sourceforge.net/projects/mondrian/files/mondrian/>

³*Schema WorkBench* è uno strumento evoluto per la creazione e la modifica di schemi multidimensionali attavverso un'interfaccia grafica, interamente realizzato in *Java*, che permette di impostare la quasi totalità degli attributi che prevede la sintassi di definizione in *XML*.

⁴Punto di riferimento per la misurazione delle performance dei sistemi.

posto da *TPC* (*Transaction Processing Performance Council*)⁵: *TPC-H*⁶. La base dati in questione, è costituita da 8 tabelle che rappresentano i dati gestionali di un'azienda (ordini, linee d'ordine, prodotti, fornitori, ...). La base dati di partenza utilizza quasi 10.000.000 di record raggruppati per 10.000 fornitori per una dimensione di circa *1GB*.

Figura 5.2: Schema TPC-H



Il file XML per semplicità è applicato solo sulla dimensione “ORDERS”.

⁵TPC è un’organizzazione *non-profit* fondata con lo scopo di fornire degli strumenti validi per misurare le performance dei DBMS.

⁶TPC-H è il benchmark progettato per misurare le performance di un DBMS in ambiente *Data Warehouse*, cioè con grandi volumi di dati e query molto complesse.

Listato 5.1 XML Mondrian DTD di TPC-H

```
1 <Schema name="tpch">
2   <Cube name="tpchCube" cache="true" enabled="true">
3
4   <Table name="lineitem">
5     </Table>
6
7   <Dimension type="StandardDimension" name="Orders" foreignKey="
8     l_orderkey" highCardinality="false">
9     <Hierarchy name="Orders" hasAll="true" AllMemberName="All
10    Customers" primaryKey="o_orderkey" primaryKeyTable="orders
11    ">
12    <Join leftKey="o_custkey" rightAlias="customer" rightKey="
13      c_custkey">
14      <Table name="orders">
15      </Table>
16      <Join leftAlias="customer" leftKey="c_nationkey"
17        rightAlias="c_nation" rightKey="n_nationkey">
18        <Table name="customer">
19        </Table>
20        <Join leftAlias="c_nation" leftKey="n_regionkey"
21          rightAlias="c_region" rightKey="r_regionkey">
22          <Table name="nation" alias="c_nation">
23          </Table>
24          <Table name="region" alias="c_region">
25          </Table>
26        </Join>
27      </Join>
28    </Join>
29    <Level name="Customer" table="customer" column="c_name" type=
30      "String" uniqueMembers="false" levelType="Regular"
31      hideMemberIf="Never">
32    </Level>
33    <Level name="Nation" table="c_nation" column="n_name" type="

34      String" uniqueMembers="true" levelType="Regular"
35      hideMemberIf="Never">
36    </Level>
37    <Level name="Region" table="c_region" column="r_name" type="

38      String" uniqueMembers="true" levelType="Regular"
```

```
                hideMemberIf="Never">
28          </Level>
29      </Hierarchy>
30  </Dimension>
31
32  <Measure name="quantity" column="l_quantity" datatype="Numeric"
33      aggregator="distinct-count" visible="true">
34      </Measure>
35  <Measure name="extendedprice" column="l_extendedprice"
36      datatype="Numeric" aggregator="distinct-count" visible="
37      true">
38      </Measure>
39  <Measure name="discount" column="l_discount" datatype="Numeric
40      " aggregator="distinct-count" visible="true">
41      </Measure>
42 </Cube>
43 </Schema>
```

Le tabelle dimensionali sono ORDERS, SUPPLIER, PART e rappresentano le varie dimensioni di analisi. Ognuna di queste contiene al suo interno, oltre ad un identificatore univoco, campi che rappresentano i livelli della gerarchia della dimensione. Prendiamo in considerazione la dimensione ORDERS.

La tabella ORDERS rappresenta la dimensione di analisi degli ordini.

La sua dichiarativa *SQL* è:

```
CREATE TABLE 'tpch'.`orders` (
    `o_orderkey` int(11) NOT NULL,
    `o_custkey` int(11) NOT NULL,
    `o_orderstatus` char(1) NOT NULL,
    `o_totalprice` decimal(15,2) NOT NULL,
    `o_orderdate` date NOT NULL,
    `o_orderpriority` char(15) NOT NULL,
    `o_clerk` char(15) NOT NULL,
    `o_shipppriority` int(11) NOT NULL,
    `o_comment` varchar(79) NOT NULL,
    PRIMARY KEY (`o_orderkey`),
    KEY `orders_fk1` (`o_custkey`))
```

TPC-H ha un tipico schema a fiocco di neve (*snowflake*) per cui, il cubo formato dalla sola dimensione ORDERS, è in realtà ottenuto attraverso il *join* tra la *fact-table* LINEITEMS, ORDERS e tutte le tabelle secondarie nel livello di aggregazione, CUSTOMER, NATION e REGION. La dimensione ORDERS, contiene come livelli REGION, NATION e CUSTOMER comune, nome del negozio, e indirizzo (si potrebbe supporre che ogni ordine sia destinato a vari clienti all'interno di diverse regioni e nazioni, e che si vogliano analizzare i dati relativi ad ognuno di questi).

La tabella dei fatti (LINEITEMS), oltre ad avere il riferimento agli *ID* delle tre tabelle dimensionali, contiene delle misure come *quantità*, *prezzo esteso*, *discount* e *tasse* del prodotto venduto.

5.3 Integrazione del sistema con il motore *OLAP*

BIRCH opera soltanto su attributi metrici, e la crescita della gerarchia è controllata in base ad una profondità massima specificata. Per poter integrare il nostro sistema con un motore *OLAP Mondrian*, abbiamo dovuto eseguire le modifiche che di seguito illustreremo.

- Rendere i campi enumerativi, dei campi di tipo numerico.

Per esempio, “O_ORDERPRIORITY” in *TPC-H*, è un campo enumerativo che ammette i seguenti valori:

- “1-URGENT”;
- “2-HIGH”;
- “3-MEDIUM”;
- “4-NOT SPECIFIED”;
- “5-LOW”.

Affinché i dati del campo in oggetto siano processabili dal nostro sistema, l’attributo “O_ORDERPRIORITY”, deve essere trasformato in attributo numerico con valori ammissibili compresi tra 1 e 5.

- Prevedere un estensione del *Mondrian DTD*.

L’estensione è quella che vi andiamo di seguito a presentare. Gli elementi aggiunti sono evidenziati in rosso.

```

<!ELEMENT Hierarchy ((%Relation;)?,(Level)*,
                     (MemberReaderParameter)*, (Attribute)+, (Depth))>

<!ATTLIST Hierarchy
      hasAll (true|false) #REQUIRED
      allMemberName CDATA #IMPLIED
      allMemberCaption CDATA #IMPLIED
      primaryKey CDATA #IMPLIED
      primaryKeyTable CDATA #IMPLIED
      defaultMember CDATA #IMPLIED
      memberReaderClass CDATA #IMPLIED>

<!ELEMENT Attribute EMPTY>

<!ATTLIST Attribute
      name CDATA #IMPLIED
      table CDATA #REQUIRED
      column CDATA #REQUIRED
      nameColumn CDATA #REQUIRED
      type (Numeric) Numeric #REQUIRED>

<!ELEMENT Depth EMPTY>

<!ATTLIST Depth
      value (Numeric) Numeric #REQUIRED>

```

Il *DTD* così esteso, permette di definire due nuovi elementi annidati all'interno del *tag* padre *< Hierarchy >*: *< Attribute >* e *< Depth >*.

L'elemento “ATTRIBUTE” rappresenta attributo metrico sul quale **OLAPBIRCH** dovrà operare e, affinché ciò accada, è necessaria la definizione di almeno un elemento di tipo “ATTRIBUTE” nel file *XML*.

Il *tag* “Attribute” contempla al suo interno gli attributi:

- “name”, specifica il nome dell'attributo metrico da considerare;
- “table”, necessario per specificare la tabella da cui reperire l'attributo metrico;
- “column” e “nameColumn”, necessari per individuare la colonna della tabella della quale considerare l'attributo metrico;
- “type”, specifica il tipo dell'attributo che si deve considerare (ovviamente solo di tipo numerico).

Il *tag* “DEPTH” permette di specificare al suo interno il valore della profondità massima che il *CF-Tree* può raggiungere.

Secondo la *DTD* descritta il file *XML* che descrive la dimensione ORDERS del database *TPC-H*, diventerebbe come segue.

Listato 5.2 XML Mondrian DTD Esteso di TPC-H

```
1 <Schema name="tpch">
2   <Cube name="tpchCube" cache="true" enabled="true">
3
4   <Table name="lineitem">
5     </Table>
6
7   <Dimension type="StandardDimension" name="Orders" foreignKey="
8     l_orderkey" highCardinality="false">
9     <Hierarchy name="Orders" hasAll="true" AllMemberName="All
10    Customers" primaryKey="o_orderkey" primaryKeyTable="orders
11    ">
12      <Join leftAlias="orders" leftKey="o_custkey" rightAlias="
13        customer" rightKey="c_custkey">
14        <Table name="orders">
15        </Table>
16        <Join leftAlias="customer" leftKey="c_nationkey"
17          rightAlias="nation" rightKey="n_nationkey">
18          <Table name="customer">
19          </Table>
20          <Join leftAlias="nation" leftKey="n_regionkey"
21            rightAlias="region" rightKey="r_regionkey">
22            <Table name="nation" alias="c_nation">
23            </Table>
24            <Table name="region" alias="c_region">
25            </Table>
26            </Join>
27            </Join>
28        </Join>
29      <Level name="Customer" table="customer" column="c_name" type
30        ="String" uniqueMembers="false" levelType="Regular"
31        hideMemberIf="Never">
32      </Level>
33      <Level name="Nation" table="c_nation" column="n_name" type="

34        String" uniqueMembers="true" levelType="Regular"
35        hideMemberIf="Never">
36      </Level>
37      <Level name="Region" table="c_region" column="r_name" type="

38        String" uniqueMembers="true" levelType="Regular"
```

```
        hideMemberIf="Never">
28    </Level>
29
30    <Attribute name="totalprice" table="orders" column=
31        o_totalprice" nameColumn="o_totalprice" type="Integer"/>
31    <Attribute name="orderpriority" table="orders" column=
32        o_orderpriority" nameColumn="o_orderpriority" type="

32        Integer" />
32    <Depth value="20"/>
33
34    </Hierarchy>
35  </Dimension>
36
37  <Measure name="quantity" column="l_quantity" datatype="Numeric"
38      aggregator="distinct-count" visible="true">
38  </Measure>
39  <Measure name="extendedprice" column="l_extendedprice"
40      datatype="Numeric" aggregator="distinct-count" visible="

40        true">
40  </Measure>
41  <Measure name="discount" column="l_discount" datatype="Numeric
42      " aggregator="distinct-count" visible="true">
42  </Measure>
43  <Measure name="tax" column="l_tax" datatype="Numeric"
44      aggregator="distinct-count" visible="true">
44  </Measure>
45
46  </Cube>
47 </Schema>
```

Il nostro sistema, in questo modo, ha tutte le informazioni per procedere:

- nel recupero dei dati metrici dal database, che sono quindi tutti i dati contenuti nelle tabelle specificate nei *tag* “ATTRIBUTE” e “MEASURE”, quest’ultimi anch’essi attributi metrici della *fact table*. La *query SQL* risultante dall’interpretazione del file *XML*, da parte del nostro sistema è:

```
SELECT l_quantity, l_extendedprice, l_discount, l_tax, o_totalprice,  
      o_orderpriority  
   FROM lineitem, orders  
 WHERE l_orderkey = o_orderkey
```

- nella costruzione dell’abero dei *CF*, in quanto nel file *XML* è automaticamente definito lo spazio dimensionale dei dati. Nel nostro esempio le dimensioni dello spazio dei dati è uguale a sei.
- nel controllo della crescita della struttura gerarchica. Nel nostro esempio la dimensione massima è pari a dieci.
- nel salvataggio delle etichette dei dati, in quanto nel file *XML* sono specificate le tabelle delle dimensioni di interesse per il *clustering*. Nel caso dell’esempio, l’unica *dimension table* primaria esistente è ORDERS, quindi tutte le etichette saranno salvate nel campo “CLUSTERID” della tabella in questione.

Il campo “CLUSTERID” è un campo aggiunto da noi in ciascuna *dimension table* primaria di *TPC-H*, per contenere la stringa che identifica l’appartenenza di ciascun dato a ciascun nodo della gerarchia.

Capitolo 6

Risultati Sperimentali

Il pensiero è azione in fase d'esperimento.

SIGMUND FREUD

In questa sezione illustriamo i risultati forniti da una serie di analisi effettuate da **OLAPBIRCH** su un serie di *dataset* di varia distribuzione e quantità di dati. Scopo di tali analisi è quello di accertare che la nostra implementazione dell'algoritmo *BIRCH* rilevi correttamente i *cluster* presenti nei *dataset* e di testare le sue funzionalità.

6.1 Introduzione

Le prove sperimentali volte a dimostrare la correttezza del *clustering* del nostro sistema, sono state effettuate su *dataset* con distribuzione e numero di punti differenti prelevati da **SPAETH Cluster Analysis Datasets**¹. Sono stati adoperati questi piccoli *dataset 2D*, poiché essendo tali, risulta di più facile realizzazione la dimostrazione grafica di come effettivamente il nostro sistema lavori.

Per testare, invece, la scalabilità del nostro sistema, abbiamo utilizzato il *dataset TPC-H*, già citato a proposito dell'integrazione del nostro sistema con il motore *OLAP Mondrian* (Sezione § 5.2 a pagina 149).

¹<http://people.sc.fsu.edu/~jb Burkardt/datasets/spaeth/spaeth.html>

CAPITOLO 6. Risultati Sperimentali

Tabella 6.1: Caratteristiche del calcolatore su cui sono state effettuate le prove sperimentali

Processore	<i>Intel® Core© 2 Duo CPU P8600 @ 2.40 GHz</i>
Memoria	<i>4.00 GB</i>
Sistema Operativo	<i>Windows 7 Professional 32 bit</i>

Le prove sperimentali sono state eseguite su un calcolatore di fascia media, avente le caratteristiche riportate in Tabella § 6.1.

La distanza adottata per il calcolo della similarità nel nostro sistema è la *Variance Increase Distance* (Sezione § 2.8 a pagina 63), la quale risulta essere la più precisa, come documentato dagli stessi autori del *BIRCH* in [ZRL95].

Il *DBSCAN* utilizzato è quello presente nel *software* di *Weka*². La versione adottata nel nostro progetto è la 3.6.4, che di *default* permette di utilizzare come misura di similarità la distanza *Euclidean*³. Il criterio in base al quale abbiamo scelto i parametri per le varie sessioni di *DBSCAN* è lo stesso studiato in [EKSX96].

Per misurare la qualità del *clustering* è stata scelta la misura Q_2 (Sezione § 2.10 a pagina 64). Minore è il valore di Q_2 , migliore è la qualità dei *cluster*.

Segnaliamo, inoltre che, nelle immagini che seguono, i punti non inscritti in alcun *cluster*, sono da considerarsi *outlier*.

²Liberamente scaricabile da http://www.cs.waikato.ac.nz/~ml/weka/index_downloading.html

³Dati due punti in uno spazio n -dimensionale, $P = (p_1, p_2, \dots, p_n)$ e $Q = (q_1, q_2, \dots, q_n)$, la distanza è calcolata come $\sqrt{\sum_{k=1}^n (p_k - q_k)^2}$

6.2 Granularità dei *cluster* in ciascun livello della gerarchia

Lo scopo di questa sezione è quello di mostrare i *cluster* ottenuti in ciascun livello del *CF-Tree*, in correlazione con i *cluster* ottenuti dal *DBSCAN* sul medesimo livello.

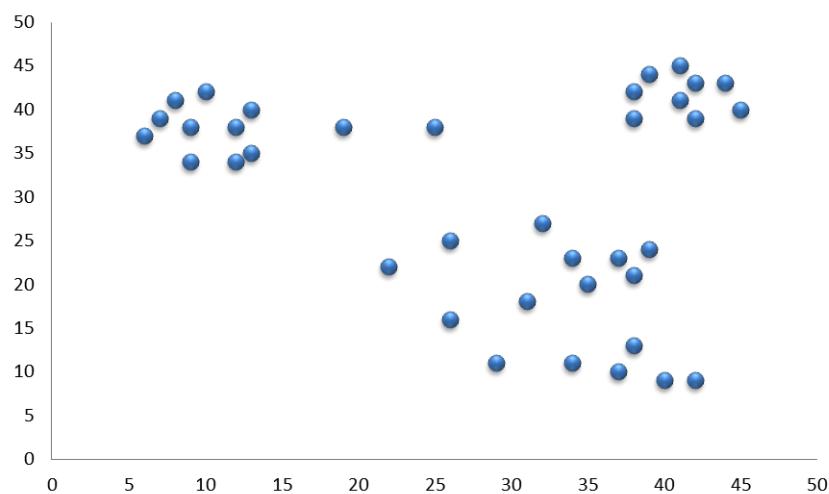
Per la seguente sessione di *test*, utilizzeremo due *dataset*:

1. il primo di ridotte dimensioni e bidimensionale, al fine di mostrare un riscontro grafico della qualità di *clustering* prodotta dal nostro sistema;
2. il secondo, di dimensione consistente, per mostrare sia i risultati sperimentali circa la granularità dei *cluster*, sia per mostrare le effettive potenzialità del nostro sistema circa la possibilità di effettuare operazioni di *roll-up* e *drill-down* sul *CF-Tree*, e le conseguenti analisi e deduzioni possibili sui risultati di tali operazioni.

6.2.1 Granularità sul *dataset Spaeth_01*

Il *dataset* utilizzato in questa sezione è *Spaeth_01*, composto da 37 punti. I parametri utilizzati saranno esplicitati nelle immagini di seguito riportare.

Figura 6.1: *Dataset Spaeth_01*

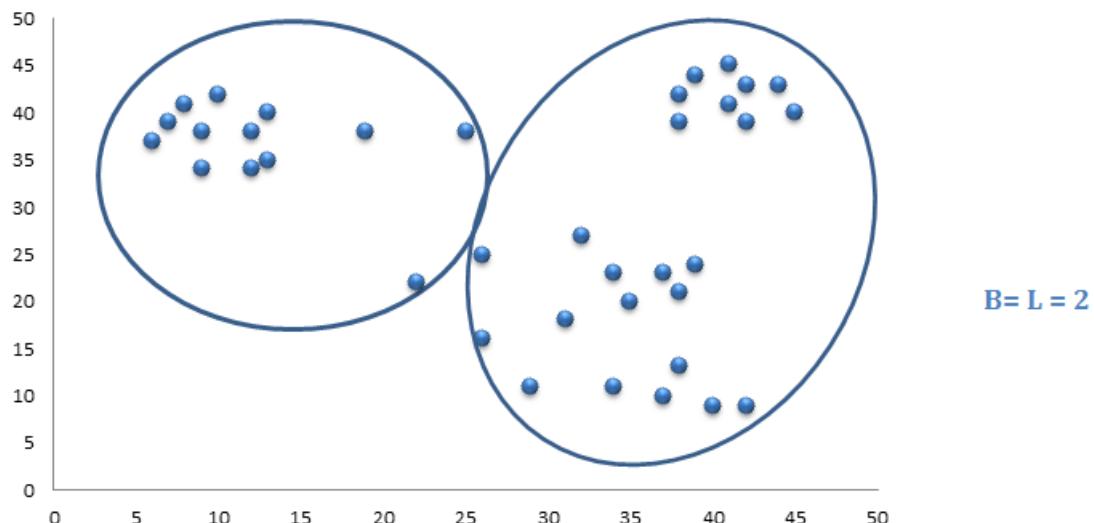


CAPITOLO 6. Risultati Sperimentali

Nelle successive immagini sono mostrati nelle Figure “a” i *cluster* presenti nel livello n -esimo del *CF-Tree*; nelle Figure “b” è rappresentato, invece, l’esito del *DBSCAN* ottenuto mediante processazione dei rispettivi centroidi.

Figura 6.2: *Cluster* risultanti nel Livello 2 su *Spaeth_01*

(a) *Cluster* individuati dal *CF-Tree* al Livello 2



(b) *Cluster* individuati dal *DBSCAN*

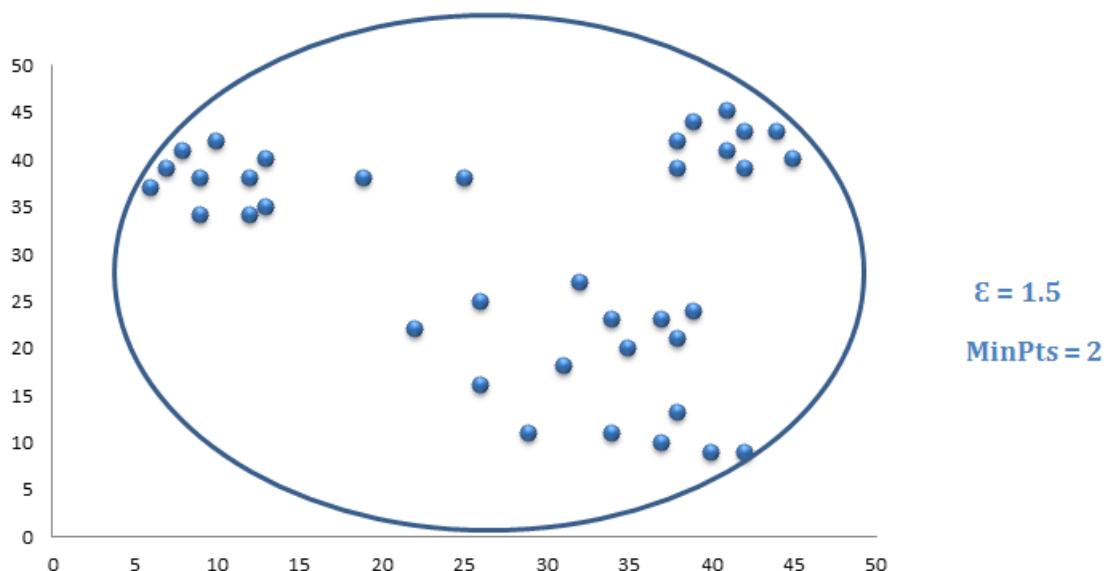
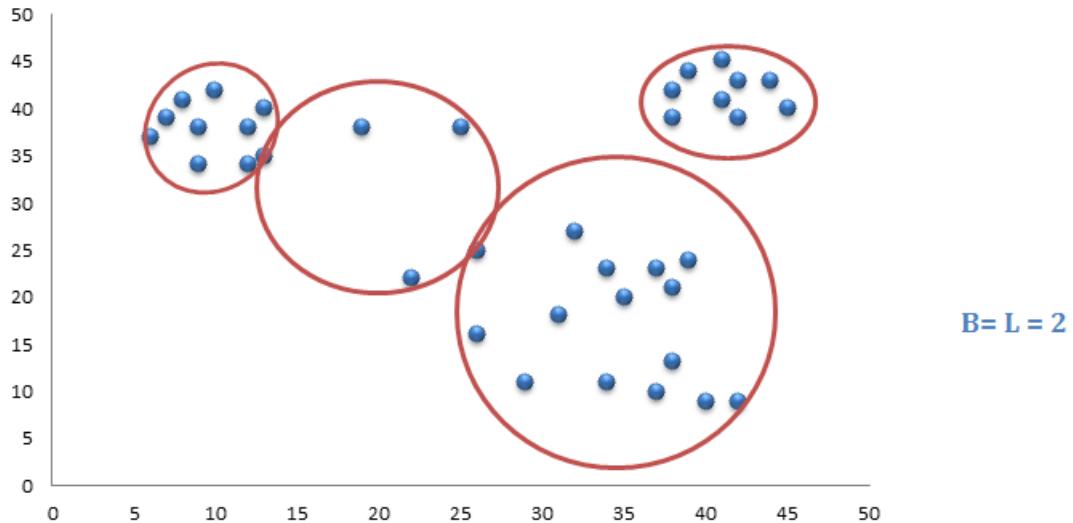


Figura 6.3: *Cluster* risultanti nel Livello 3 su *Spaeth_01*

(a) *Cluster* individuati dal *CF-Tree* al Livello 3



(b) *Cluster* individuati dal *DBSCAN*

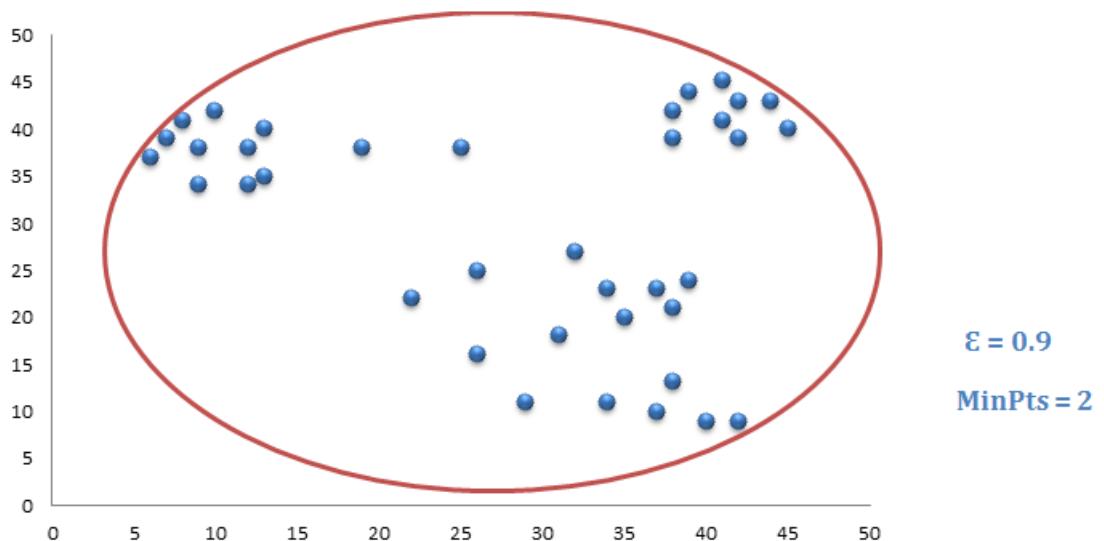
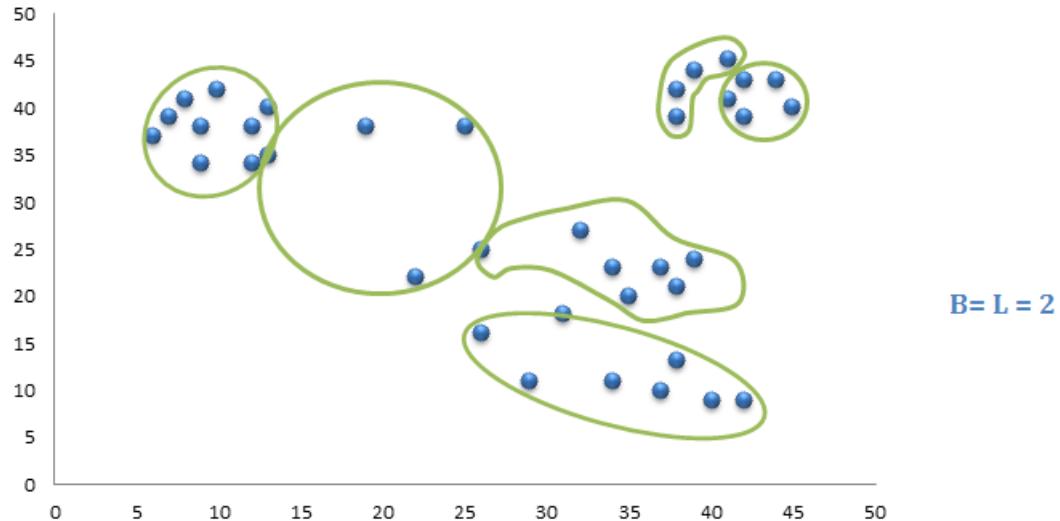


Figura 6.4: *Cluster* risultanti nel Livello 4 su *Spaeth_01*

(a) *Cluster* individuati dal *CF-Tree* al Livello 4



(b) *Cluster* individuati dal *DBSCAN*

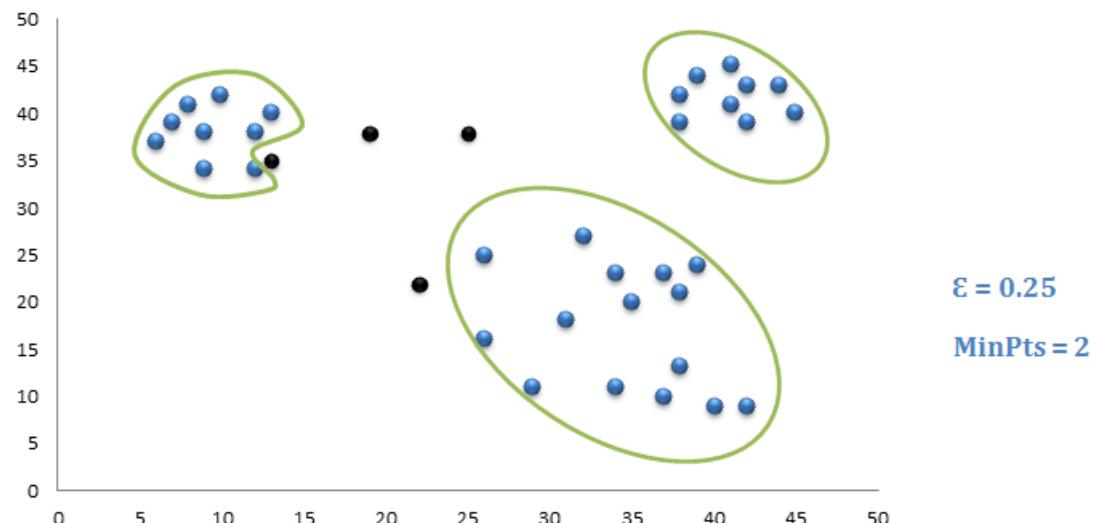
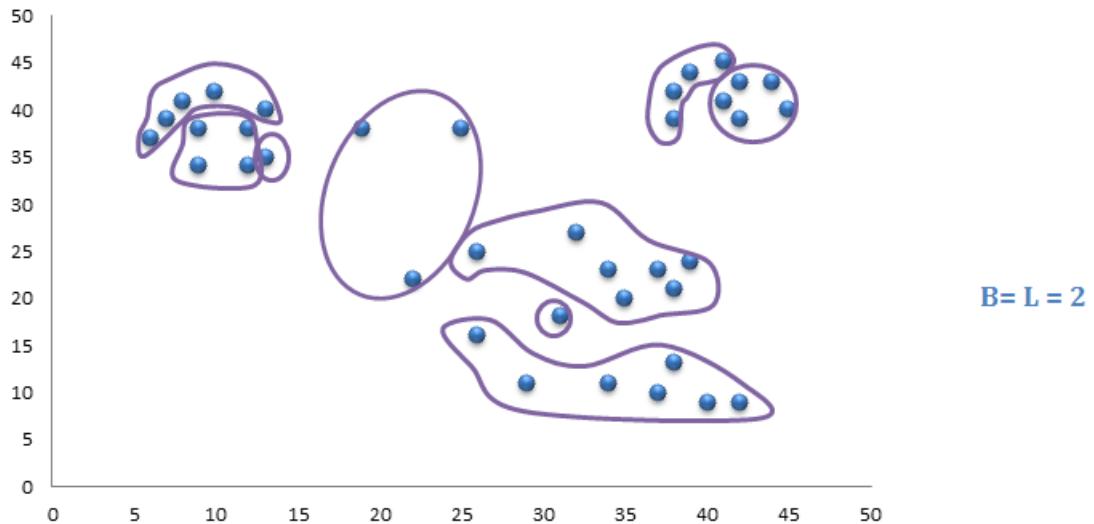


Figura 6.5: Cluster risultanti nel Livello 5 su *Spaeth_01*

(a) *Cluster* individuati dal *CF-Tree* al Livello 5



(b) *Cluster* individuati dal *DBSCAN*

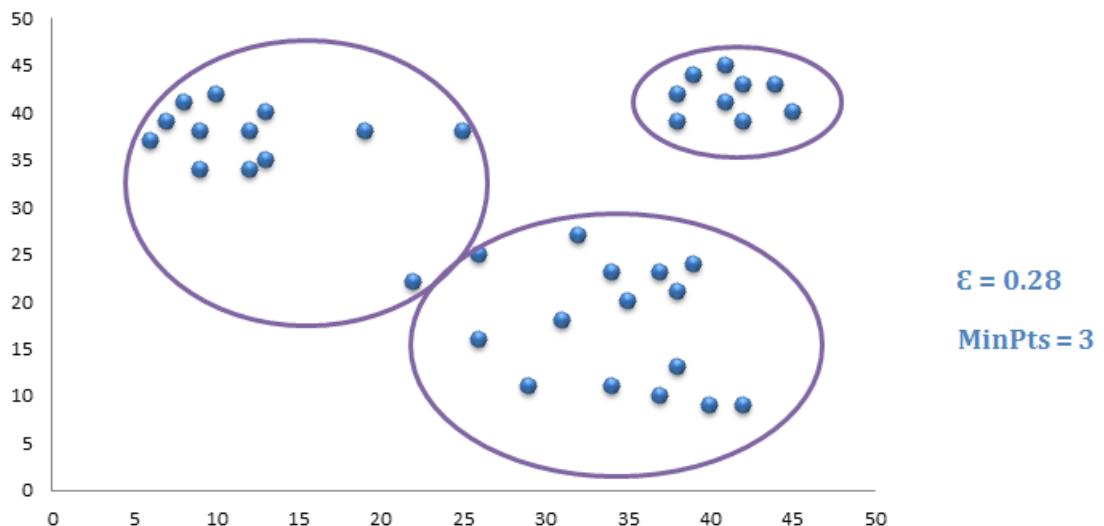
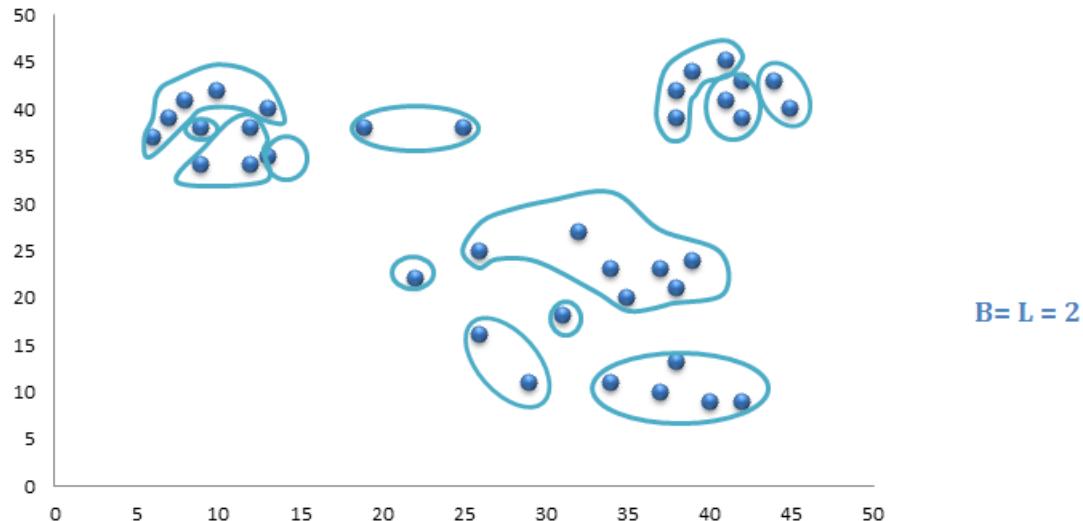


Figura 6.6: *Cluster* risultanti nel Livello 6 su *Spaeth_01*

(a) *Cluster* individuati dal *CF-Tree* al Livello 6



(b) *Cluster* individuati dal *DBSCAN*

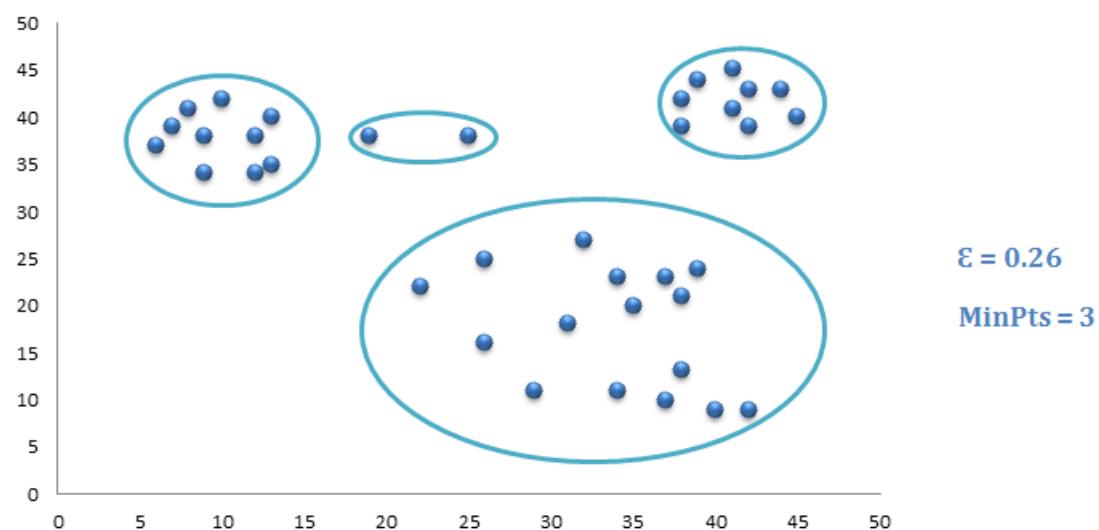
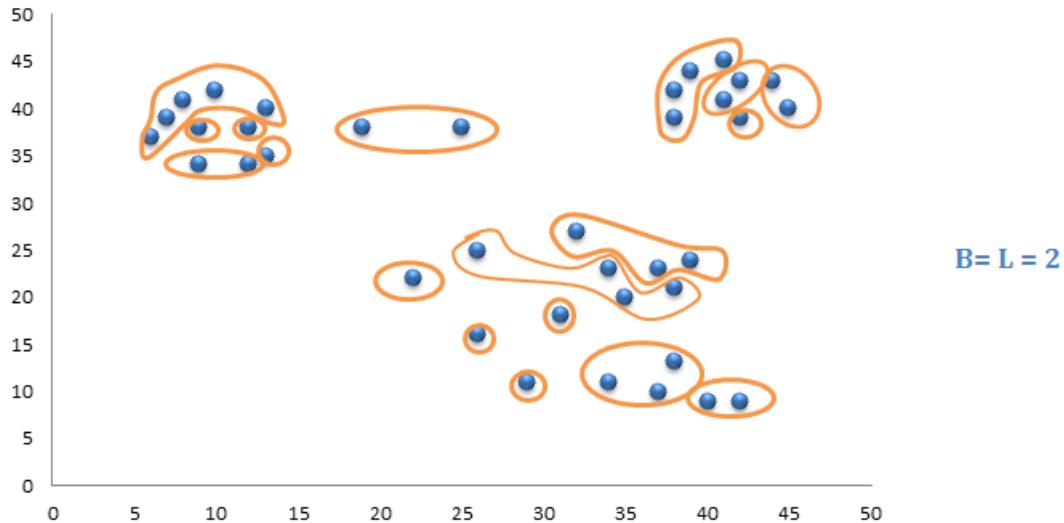
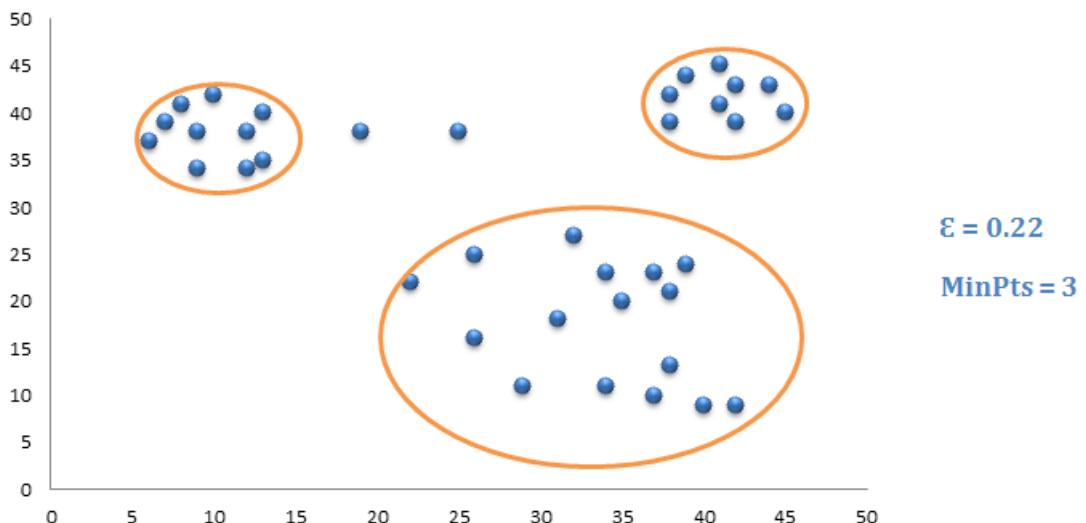


Figura 6.7: Cluster risultanti nel Livello 7 su Spaeth_01

(a) Cluster individuati dal CF-Tree al Livello 7



(b) Cluster individuati dal DBSCAN



Come è possibile notare, il DBSCAN produce cluster più accurati lavorando sugli ultimi livelli della gerarchia, ossia su quelli che presentano una granularità maggiore.

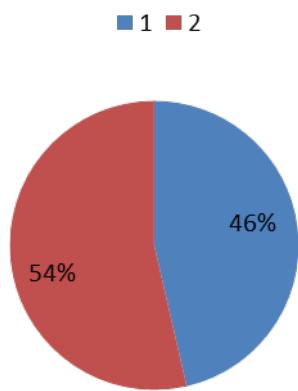
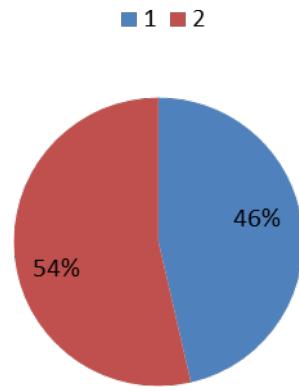
6.2.2 Granularità sul dataset *TPC-H*

Il *dataset* utilizzato in questa sezione è il *benchmark TPC-H* sulla dimensione ORDERS, composto da 6.001.215 punti. Per la costruzione del *CF-Tree* sono stati utilizzati come fattore di ramificazione il valore 2, e come profondità massima dell’albero il valore 20. Per semplicità mostriamo l’analisi sui primi 6 livelli dell’albero escluso il primo che agglomba tutti i punti nel *CF root*.

N° Cluster	Cardinalità	Percentuale %
1	2.784.240	46
2	3.216.975	54

(a) Livello 2 *CF-Tree*

N° Cluster	Cardinalità	Percentuale %
1	2.760.559	46
2	3.240.656	54

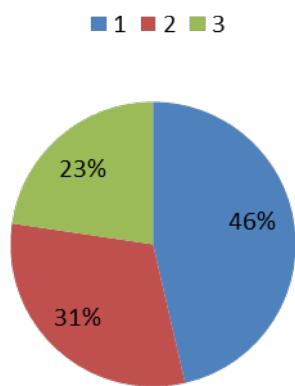
(b) Livello 2 *DBSCAN*(c) Diagramma Livello 2 *CF-Tree*(d) Diagramma Livello 2 *DBSCAN*

N° Cluster	Cardinalità	Percentuale %
1	2.784.240	46
2	1.857.184	31
3	1.359.791	23

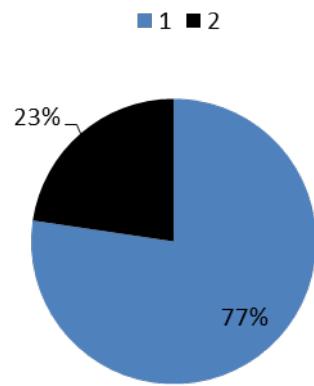
(c) Livello 3 *CF-Tree*

N° Cluster	Cardinalità	Percentuale %
1	4.620.936	77
R	1.380.279	23

(d) Livello 3 *DBSCAN*



(e) Diagramma Livello 3 *CF-Tree*



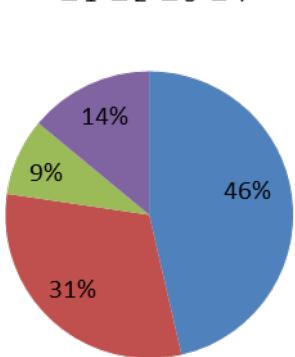
(f) Diagramma Livello 3 *DBSCAN*

N° Cluster	Cardinalità	Percentuale %
1	2.784.240	46
2	1.857.184	31
3	521.241	9
4	838.550	14

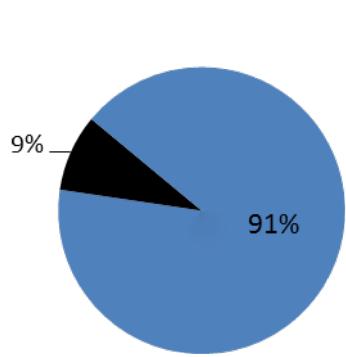
(e) Livello 4 *CF-Tree*

N° Cluster	Cardinalità	Percentuale %
1	5.461.106	91
R	540.109	9

(f) Livello 4 *DBSCAN*



(g) Diagramma Livello 4 *CF-Tree*



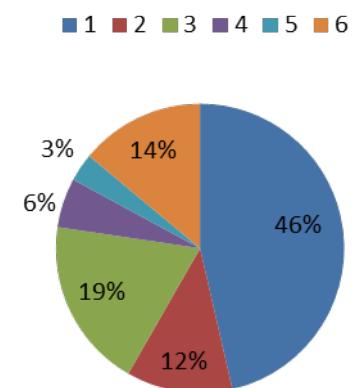
(h) Diagramma Livello 4 *DBSCAN*

N° Cluster	Cardinalità	Percentuale %
1	2.784.240	46
2	714.049	12
3	1.143.135	19
4	332.719	6
5	188.522	3
6	838.550	14

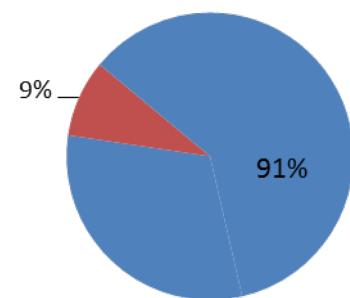
(g) Livello 5 *CF-Tree*

N° Cluster	Cardinalità	Percentuale %
1	5.461.106	91
2	540.109	9

(h) Livello 5 *DBSCAN*



(i) Diagramma Livello 5 *CF-Tree*



(j) Diagramma Livello 5 *DBSCAN*

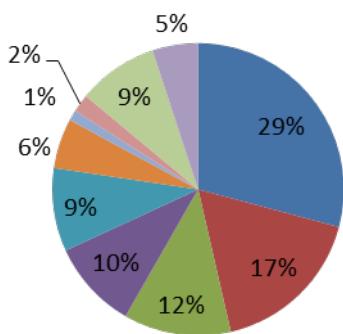
N° Cluster	Cardinalità	Percentuale %
1	1.749.594	29
2	1.034.645	17
3	714.049	12
4	590.172	10
5	552.964	9
6	332.719	6
7	69.134	1
8	119.388	2
9	534.972	9
10	303.578	5

(i) Livello 6 *CF-Tree*

N° Cluster	Cardinalità	Percentuale %
1	4.560.923	76
2	900.183	15
R	540.109	9

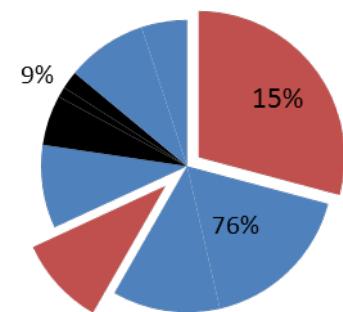
(j) Livello 6 *DBSCAN*

■ 1 ■ 2 ■ 3 ■ 4 ■ 5 ■ 6 ■ 7 ■ 8 ■ 9 ■ 10



(k) Diagramma Livello 6 *CF-Tree*

■ 1 ■ 2 ■ R



(l) Diagramma Livello 5 *DBSCAN*

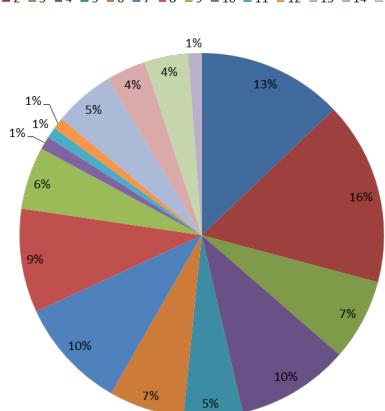
N° Cluster	Cardinalità	Percentuale %
1	773.437	13
2	976.158	16
3	434.236	7
4	600.410	10
5	313.203	5
6	400.845	7
7	590.171	10
8	552.964	9
9	332.719	6
10	69.134	1
11	58.068	1
12	61.320	1
13	321.053	5
14	213.919	4
15	231.959	4
16	71.619	1

(k) Livello 7 *CF-Tree*

N° Cluster	Cardinalità	Percentuale %
1	2.040.413	34
2	720.146	12
3	3.240.656	54

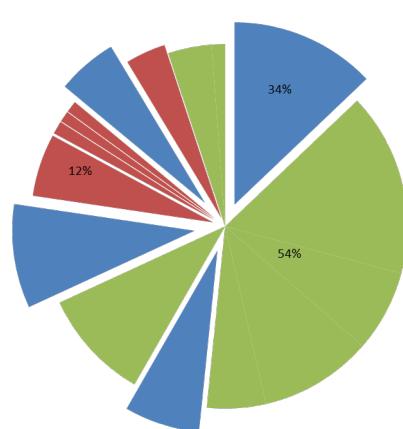
(l) Livello 7 *DBSCAN*

■ 1 ■ 2 ■ 3 ■ 4 ■ 5 ■ 6 ■ 7 ■ 8 ■ 9 ■ 10 ■ 11 ■ 12 ■ 13 ■ 14 ■ 15 ■ 16



(m) Diagramma Livello 7 *CF-Tree*

■ 1 ■ 2 ■ 3



(n) Diagramma Livello 7 *DBSCAN*

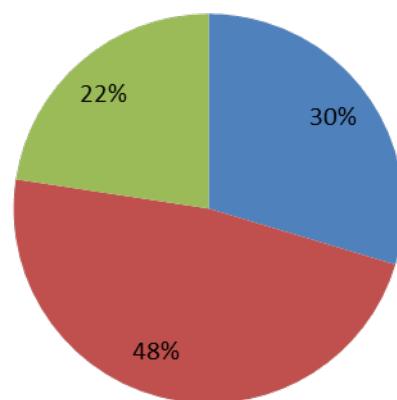
Di seguito verranno riportate le distribuzioni, rispetto ai cluster determinati dal *DBSCAN* sui livelli 5, 6 e 7 della gerarchia relativa a *TPC-H*, sia delle proprietà interessate e non dal *clustering*, che delle proprietà su cui non è possibile applicare l'algoritmo ***OLAPBIRCH***, in quanto non attributi di metrici.

Tabella 6.2: Distribuzione della media aritmetica sui 3 *cluster* individuati dal *DBSCAN* al Livello 7 della gerarchia

	Totalprice	Quantity	Linenumber	Acctbal
Cluster 1	199627	27,65	3,02	4477
Cluster 2	321591	31,85	3,81	4498
Cluster 3	152139	22,73	2,8	4484

Figura 6.8: Distribuzione della media aritmetica rispetto a TOTALPRICE sui 3 *cluster* individuati dal *DBSCAN* al Livello 7 della gerarchia

■ Cluster 1 ■ Cluster 2 ■ Cluster 3



CAPITOLO 6. Risultati Sperimentali

Figura 6.9: Distribuzione della media aritmetica rispetto a QUANTITY sui 3 *cluster* individuati dal *DBSCAN* al Livello 7 della gerarchia

■ Cluster 1 ■ Cluster 2 ■ Cluster 3

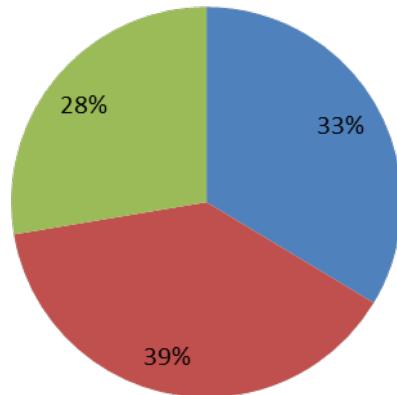


Figura 6.10: Distribuzione della media aritmetica rispetto a LINENUMBER sui 3 *cluster* individuati dal *DBSCAN* al Livello 7 della gerarchia

■ Cluster 1 ■ Cluster 2 ■ Cluster 3

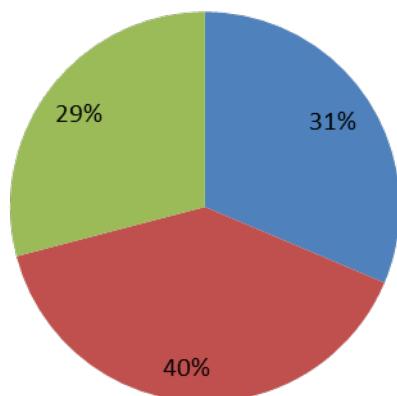


Figura 6.11: Distribuzione della media aritmetica rispetto a ACCTBAL sui 3 *cluster* individuati dal DBSCAN al Livello 7 della gerarchia

■ Cluster 1 ■ Cluster 2 ■ Cluster 3

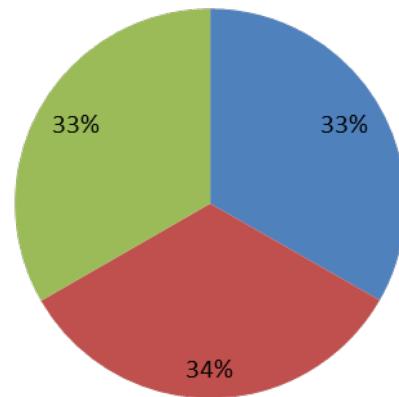
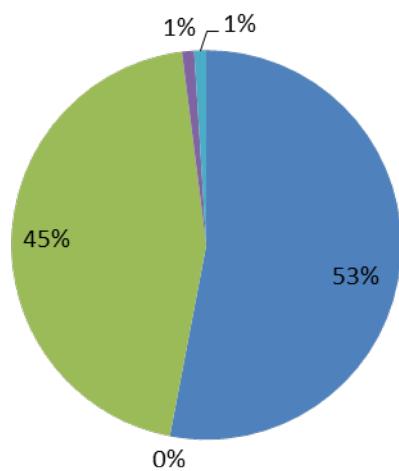


Figura 6.12: Distribuzione della proprietà REGION sul *cluster* 1 individuato dal DBSCAN al Livello 7 della gerarchia

■ Europa ■ Africa ■ Medio Oriente ■ Asia ■ America



CAPITOLO 6. Risultati Sperimentali

Figura 6.13: Distribuzione della proprietà REGION sul *cluster* 2 individuato dal DBSCAN al Livello 7 della gerarchia

■ Europa ■ Africa ■ Medio Oriente ■ Asia ■ America

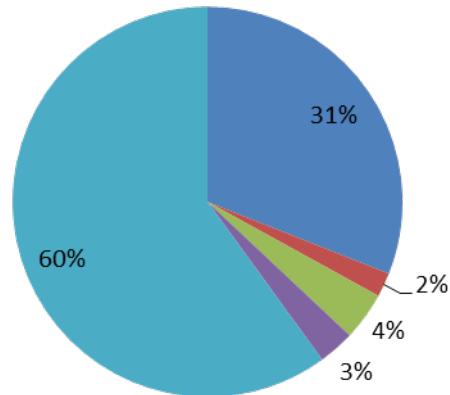


Figura 6.14: Distribuzione della proprietà REGION sul *cluster* 3 individuato dal DBSCAN al Livello 7 della gerarchia

■ Europa ■ Africa ■ Medio Oriente ■ Asia ■ America

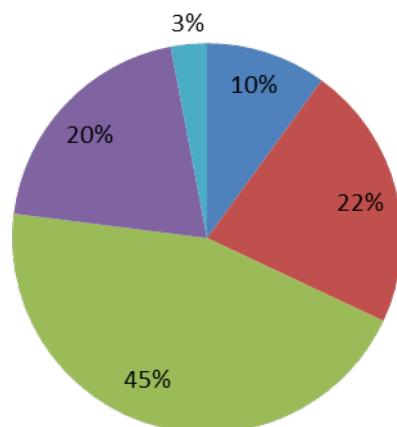


Tabella 6.3: Distribuzione della media aritmetica sui 3 *cluster* individuati dal *DBSCAN* al Livello 6 della gerarchia

	Totalprice	Quantity	Linenumber	Acctbal
Cluster 1	161282	24,55	2,78	4489
Cluster 2	244690	26,35	3,65	4475
Rumore	338602	32,75	3,85	4454

Figura 6.15: Distribuzione della media aritmetica rispetto a TOTALPRICE sui 3 *cluster* individuati dal *DBSCAN* al Livello 6 della gerarchia

■ Cluster 1 ■ Cluster 2 ■ rumore

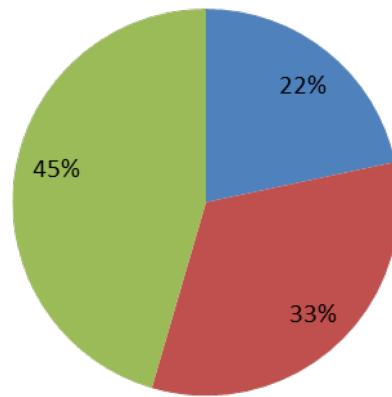
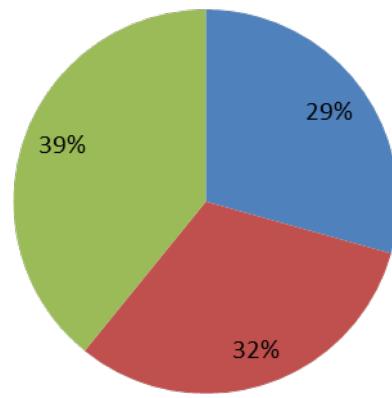


Figura 6.16: Distribuzione della media aritmetica rispetto a QUANTITY sui 3 *cluster* individuati dal *DBSCAN* al Livello 6 della gerarchia

■ Cluster 1 ■ Cluster 2 ■ rumore



CAPITOLO 6. Risultati Sperimentali

Figura 6.17: Distribuzione della media aritmetica rispetto a LINENUMBER sui 3 *cluster* individuati dal DBSCAN al Livello 6 della gerarchia

■ Cluster 1 ■ Cluster 2 ■ rumore

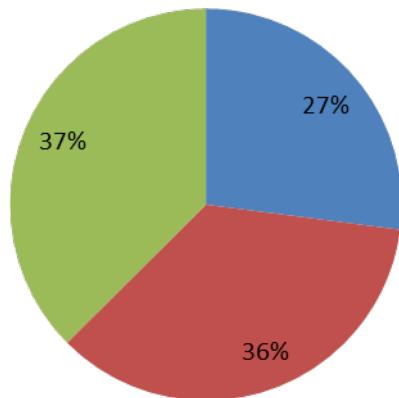


Figura 6.18: Distribuzione della media aritmetica rispetto a ACCTBAL sui 3 *cluster* individuati dal DBSCAN al Livello 6 della gerarchia

■ Cluster 1 ■ Cluster 2 ■ rumore

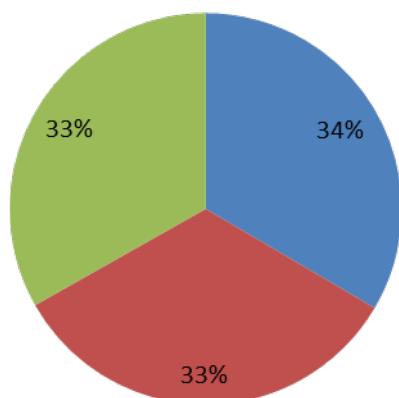


Figura 6.19: Distribuzione della proprietà REGION sul *cluster* 1 individuato dal DBSCAN al Livello 6 della gerarchia

■ Europa ■ Africa ■ Medio Oriente ■ Asia ■ America

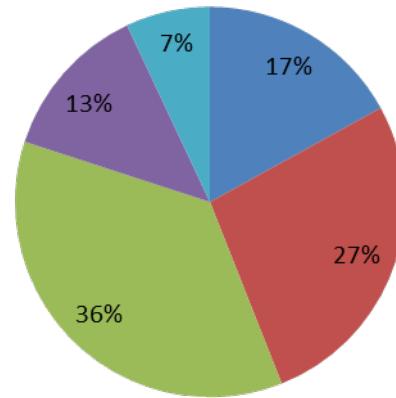
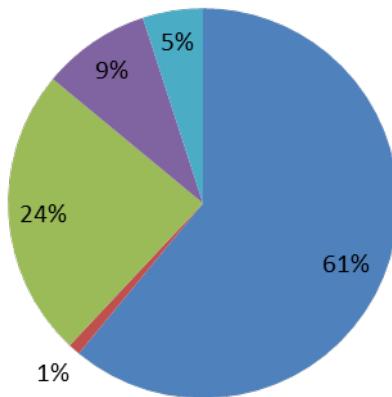


Figura 6.20: Distribuzione della proprietà REGION sul *cluster* 2 individuato dal DBSCAN al Livello 6 della gerarchia

■ Europa ■ Africa ■ Medio Oriente ■ Asia ■ America



CAPITOLO 6. Risultati Sperimentali

Figura 6.21: Distribuzione della proprietà REGION sul *cluster Rumore* individuato dal DBSCAN al Livello 6 della gerarchia

■ Europa ■ Africa ■ Medio Oriente ■ Asia ■ America

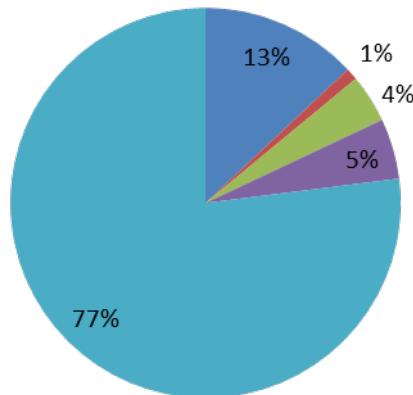


Tabella 6.4: Distribuzione della media aritmetica sui 3 *cluster* individuati dal DBSCAN al Livello 5 della gerarchia

	Totalprice	Quantity	Linenumber	Acctbal
Cluster 1	338602	32,75	3,85	4453
Cluster 2	174886	24,84	2,92	4486

Figura 6.22: Distribuzione della media aritmetica rispetto a TOTALPRICE sui 2 *cluster* individuati dal DBSCAN al Livello 5 della gerarchia

■ Cluster 1 ■ Cluster 2

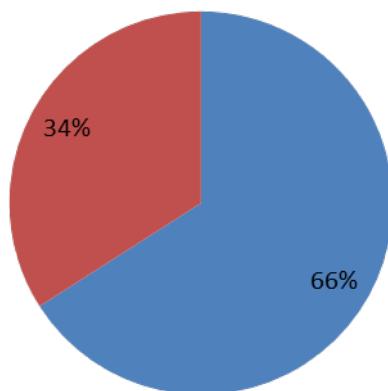


Figura 6.23: Distribuzione della media aritmetica rispetto a QUANTITY sui 2 *cluster* individuati dal DBSCAN al Livello 5 della gerarchia

■ Cluster 1 ■ Cluster 2

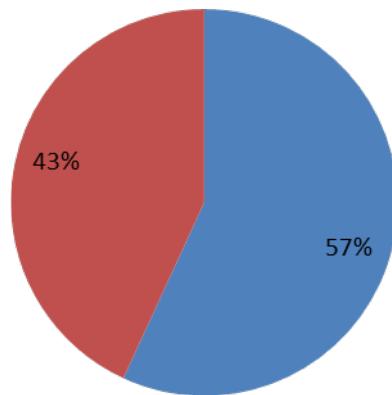


Figura 6.24: Distribuzione della media aritmetica rispetto a LINENUMBER sui 2 *cluster* individuati dal DBSCAN al Livello 5 della gerarchia

■ Cluster 1 ■ Cluster 2

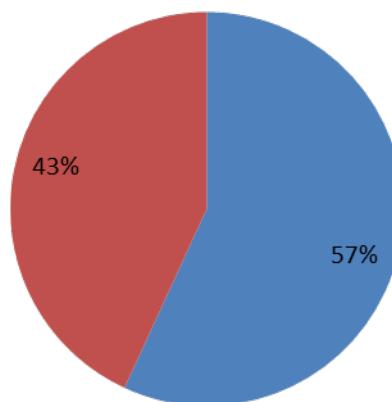


Figura 6.25: Distribuzione della media aritmetica rispetto a ACCTBAL sui 2 *cluster* individuati dal DBSCAN al Livello 5 della gerarchia

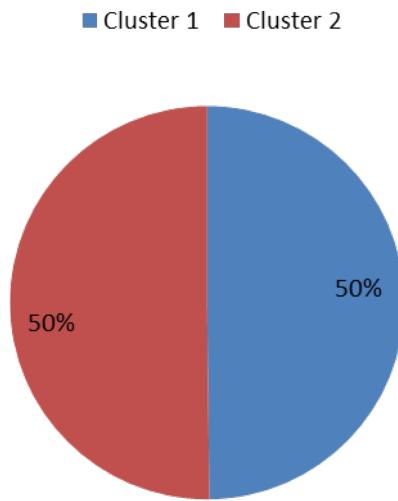


Figura 6.26: Distribuzione della proprietà REGION sul *cluster* 1 individuato dal DBSCAN al Livello 5 della gerarchia

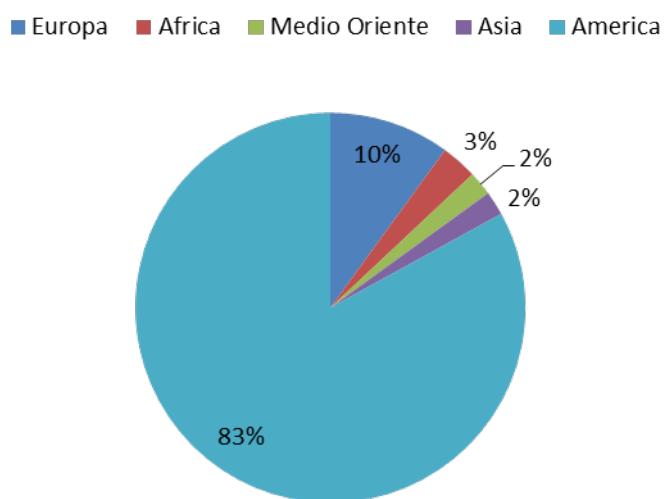
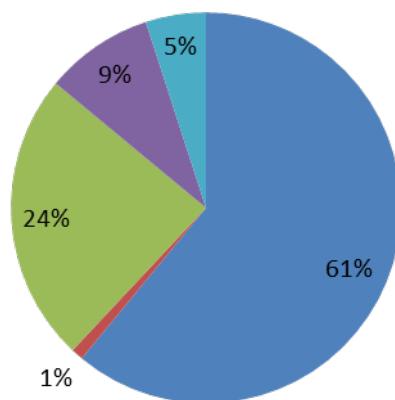


Figura 6.27: Distribuzione della proprietà REGION sul *cluster* 2 individuato dal DBSCAN al Livello 5 della gerarchia

■ Europa ■ Africa ■ Medio Oriente ■ Asia ■ America



6.3 Effetto dei *Rebuild*

Lo scopo di questa sezione è quello di dimostrare che i *rebuild* del *CF-Tree*, se non eccessivi, non precludono la qualità dei *cluster*. I grafici mostrati nelle figure seguenti si riferiscono all'ultimo livello della struttura. Il *dataset* utilizzato per tale scopo è *Spaeth_05*, composto da 55 punti. I parametri utilizzati saranno esplicitati nelle immagini di seguito riportare.

Figura 6.28: *Dataset Spaeth_05*

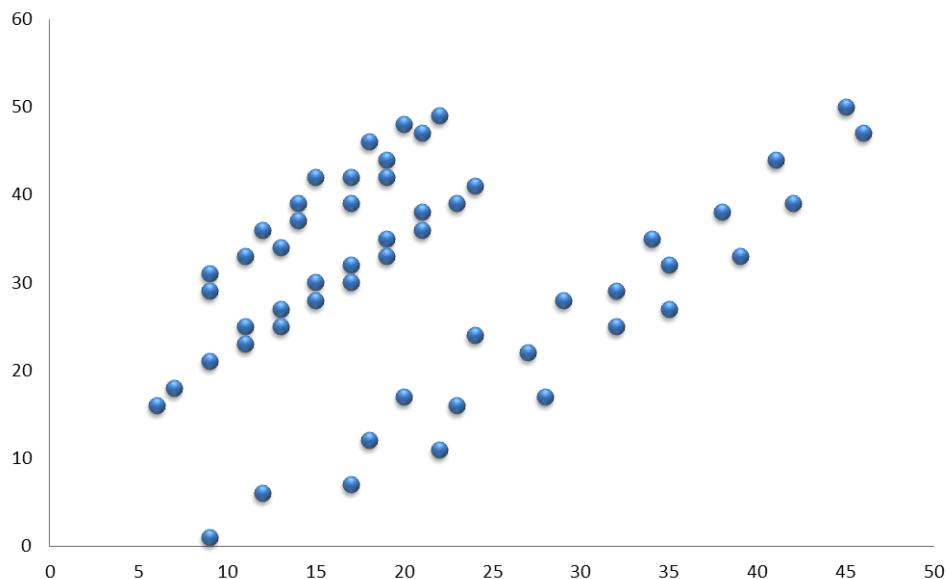
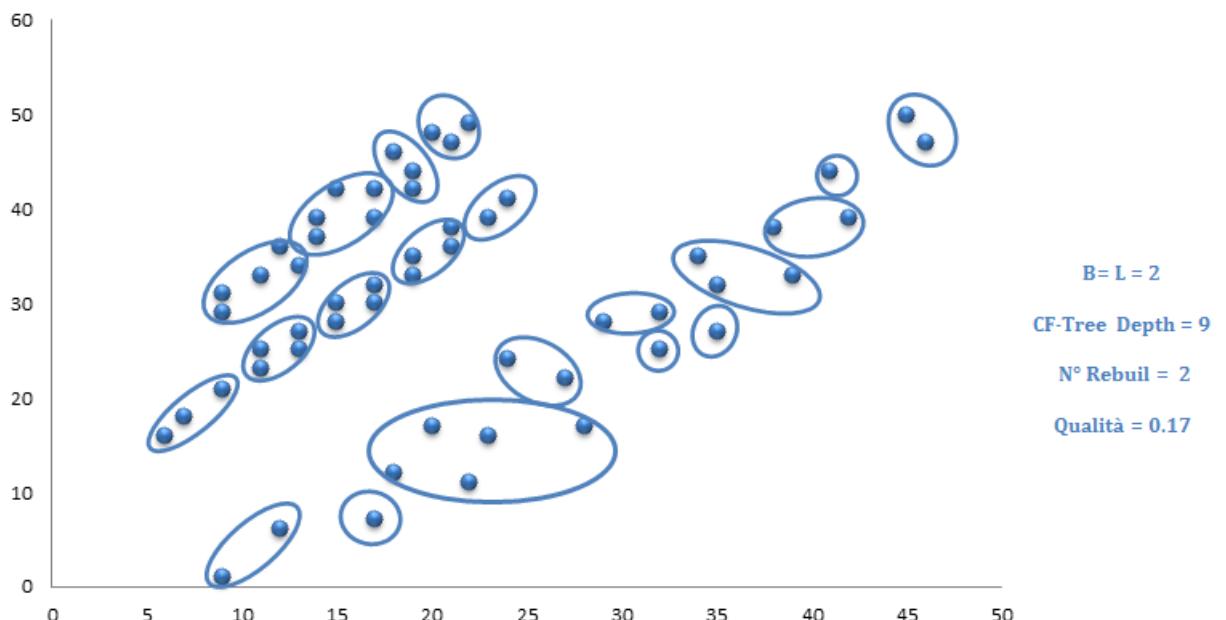
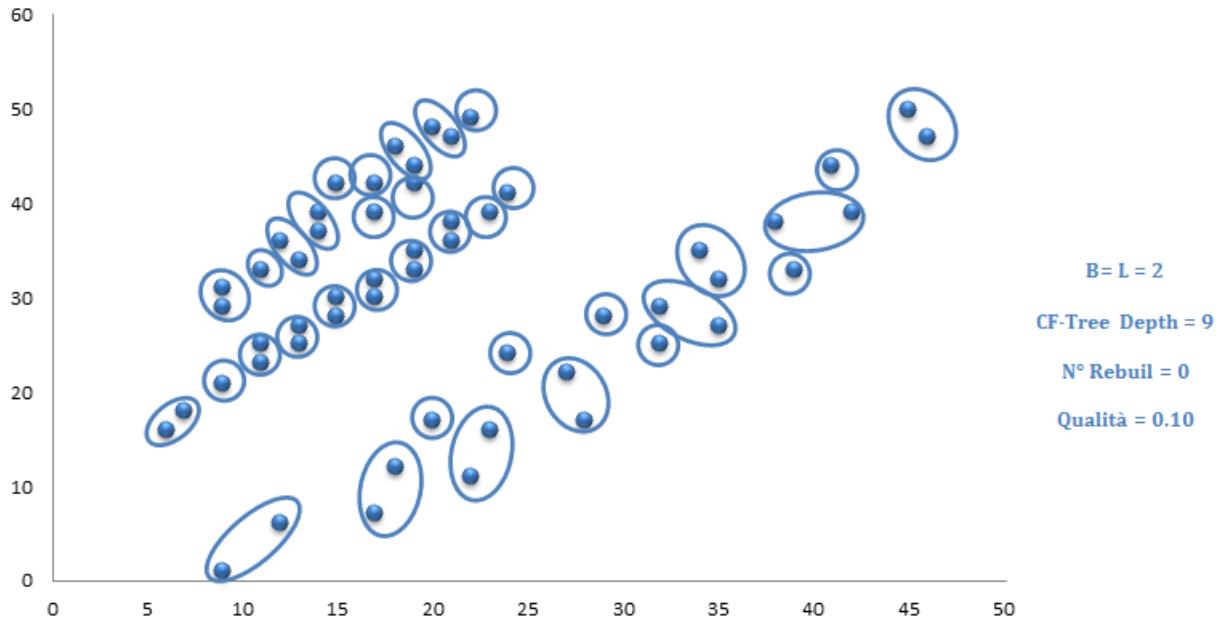


Figura 6.29: Effetto *rebuild*

(a) *Cluster* presenti nell'ultimo livello del *CF-Tree* costruito senza *rebuild*



(b) *Cluster* presenti nell'ultimo livello del *CF-Tree* costruito mediante *rebuild*

Nella Figura § 6.29 nella pagina precedente sono mostrati a sinistra i *cluster* presenti nell'ultimo livello del *CF-Tree*, costruito senza alcun *rebuild*; a destra sono rappresentati, invece, i *cluster* presenti nell'ultimo livello del *CF-Tree*, costruito attraverso 2 *rebuild*. Come è possibile notare, i *cluster* ottenuti in quest'ultimo caso, mantengono la qualità dei *cluster* sostanzialmente accettabile.

Infatti, la Figura § 6.30, rappresenta i *cluster* ottenuti dal *DBSCAN* sia mediante processazione dei centroidi dei *cluster* presenti in Figura § 6.29a nella pagina precedente che in Figura § 6.30b nella pagina precedente.

Figura 6.30: *Cluster* individuati da *DBSCAN* su *Spaeth_05*

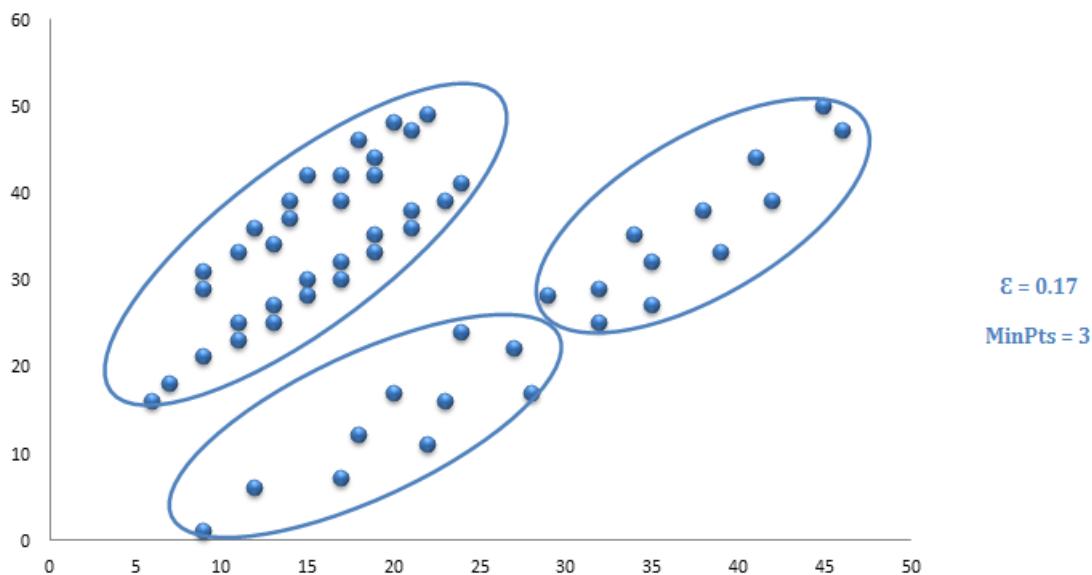
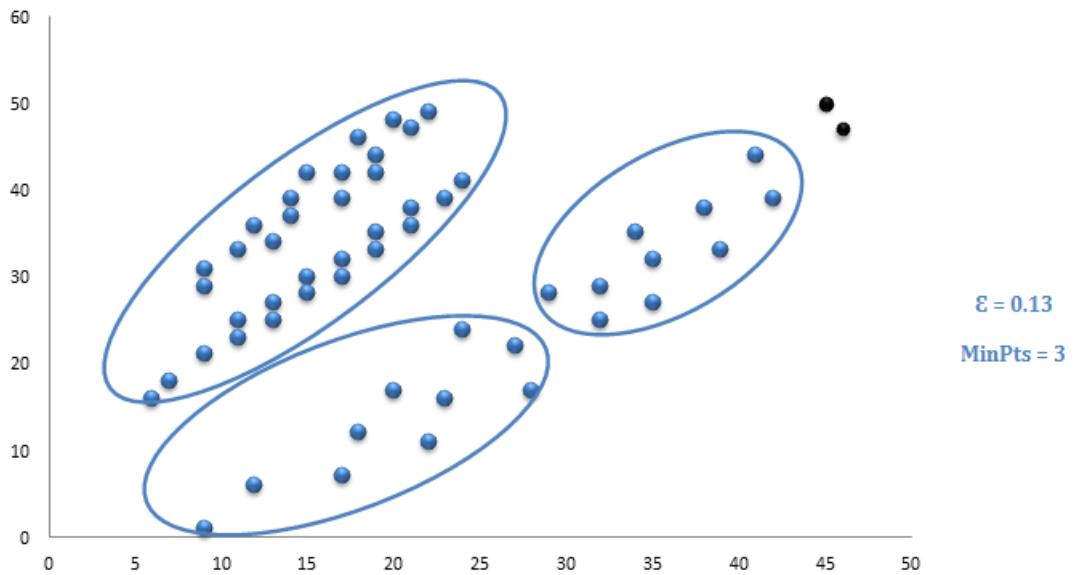


Figura 6.31: Cluster individuati da DBSCAN su Spaeth_05



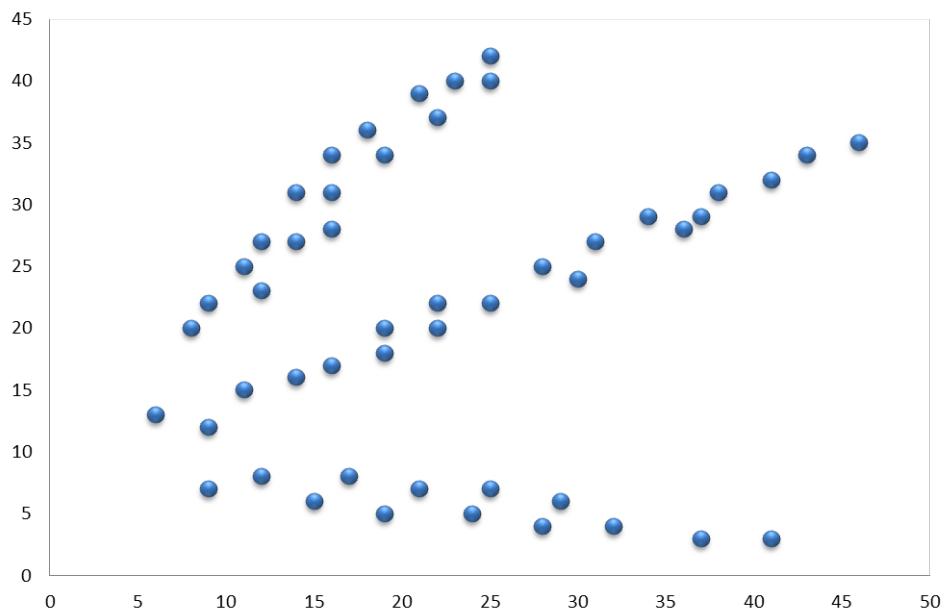
La Figura § 6.31, rappresenta i *cluster* ottenuti dal DBSCAN direttamente sui punti del *dataset*. Come è evidente, utilizzando il *CF-Tree* si garantisce un buon *trade-off* tra qualità dei *cluster* e livello di complessità spazio-temporale, rispetto ai classici algoritmi di *clustering* basati sulla densità. Il DBSCAN, infatti, processando i centroidi dei *cluster* ottenuti dal *BIRCH*, processa una quantità di punti di gran lunga inferiore alla dimensione dell'intero *dataset*. Tuttavia, un eccessivo numero di *rebuild*, causa una crescita repentina del valore di soglia e ciò a sua volta causa *cluster* troppo grossolani che determinano una cattiva qualità dei *cluster*.

6.4 Fattore di Ramificazione vs Qualità dei *cluster*

Lo scopo di questa sezione è quello di dimostrare che il fattore di ramificazione è inversamente proporzionale alla qualità del *clustering*, ovvero minore è il valore del fattore di ramificazione, maggiore sarà il livello di qualità dei *cluster*.

Il *dataset* utilizzato per tale scopo è *Spaeth_06*, composto da 50 punti. I parametri utilizzati saranno esplicitati nelle immagini di seguito riportare.

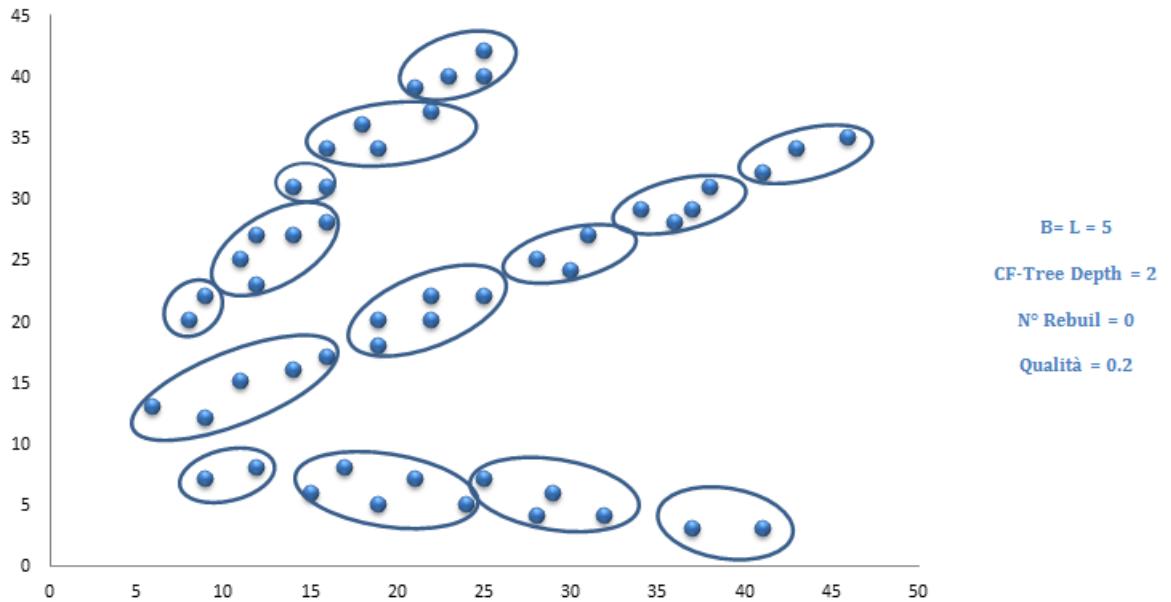
Figura 6.32: Dataset *Spaeth_06*



Nelle successive figure sono mostrati a sinistra i *cluster* presenti nell'ultimo livello del *CF-Tree* con i diversi valori di fattore di ramificazione; a destra è rappresentato, invece, l'esito del *DBSCAN* ottenuto mediante processazione dei rispettivi centroidi.

Figura 6.33: *Cluster* individuati su *Spaeth_06* con $B = L = 5$

(a) *Cluster* individuati dal *CF-Tree*



(b) *Cluster* individuati dal *DBSCAN*

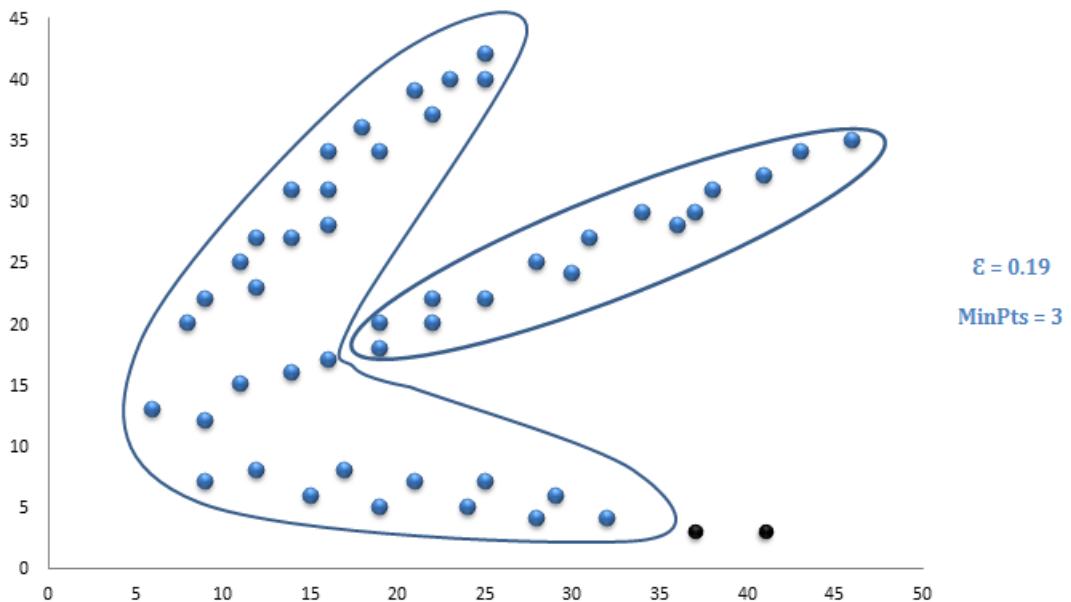
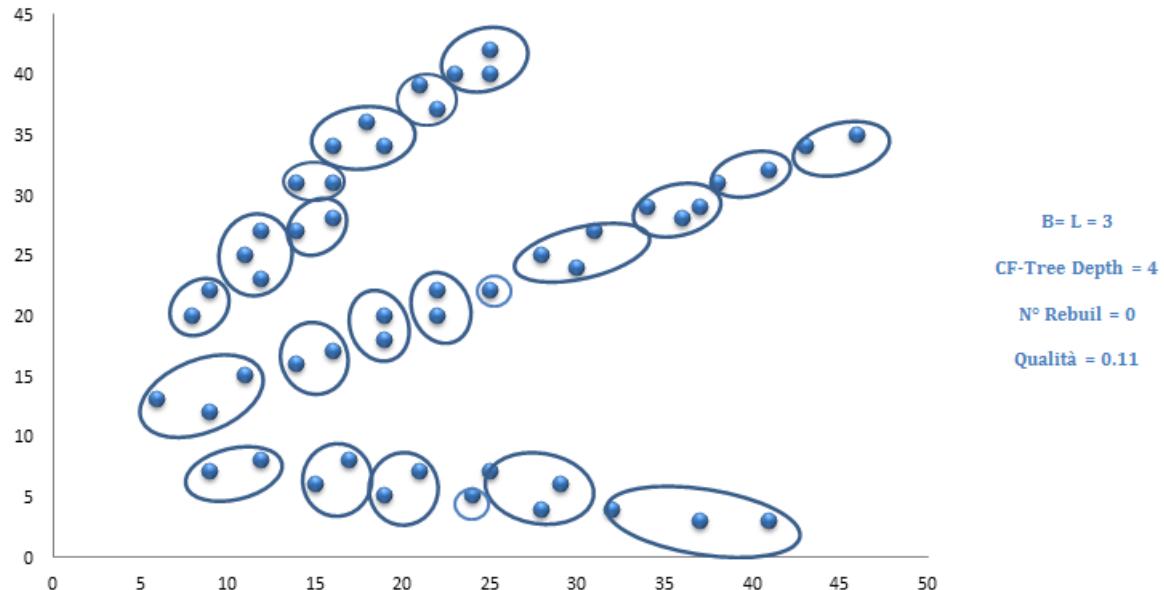


Figura 6.34: *Cluster* individuati su *Spaeth_06* con $B = L = 3$

(a) *Cluster* individuati dal *CF-Tree*



(b) *Cluster* individuati dal *DBSCAN*

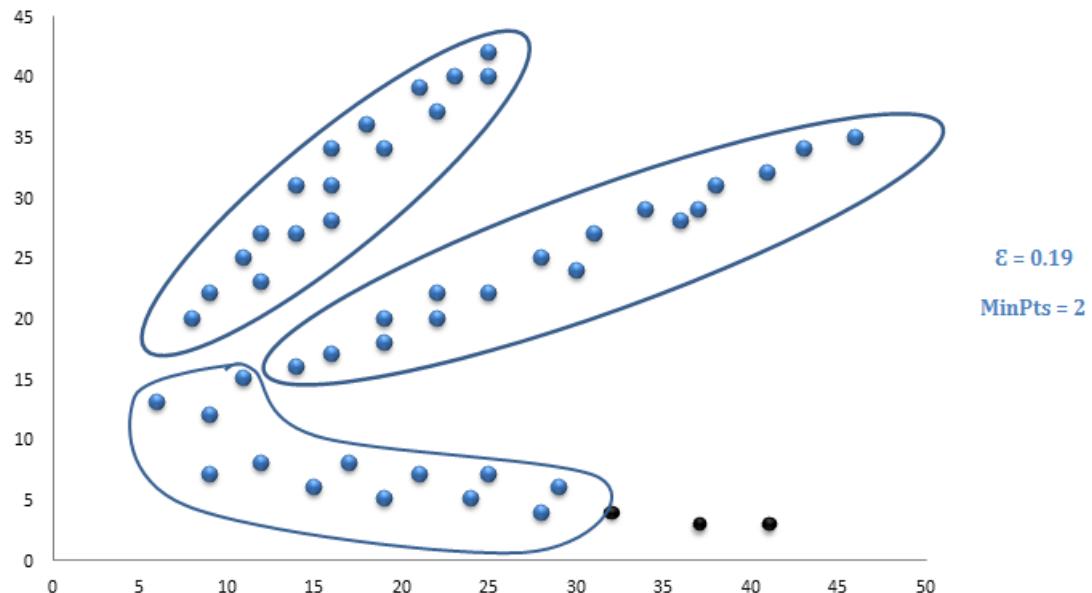
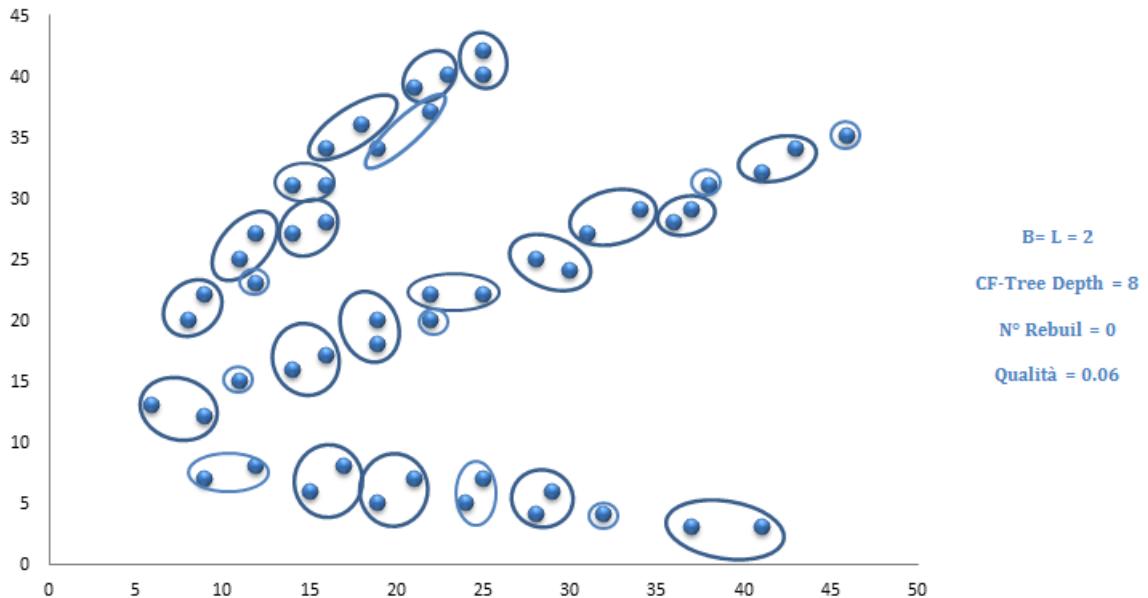
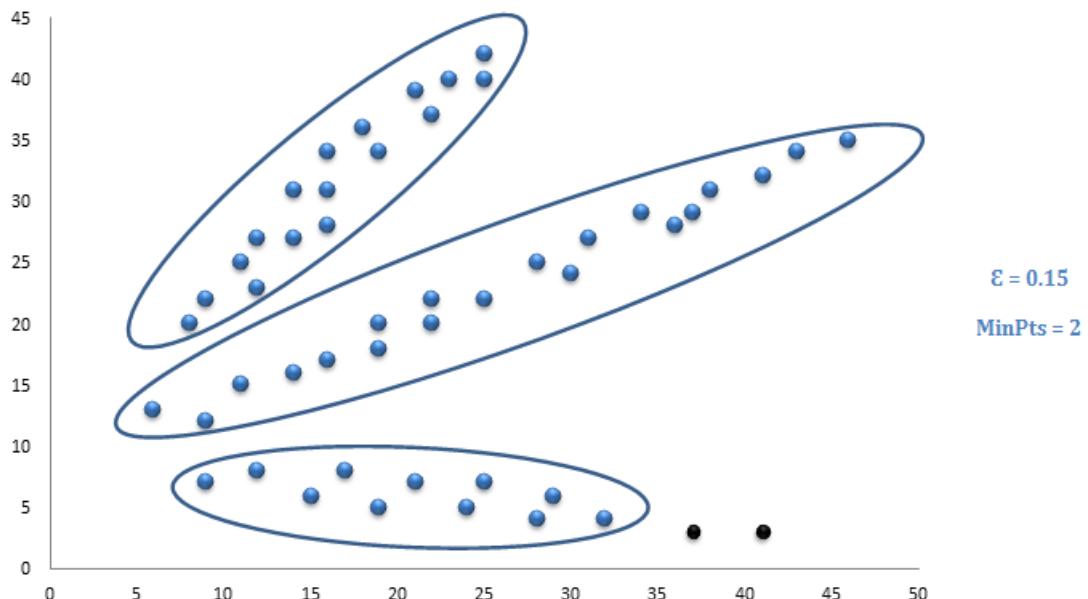


Figura 6.35: Cluster individuati su *Spaeth_06* con $B = L = 2$

(a) Cluster individuati dal *CF-Tree*



(b) Cluster individuati dal *DBSCAN*



Le immagini enfatizzano il fatto che ad un valore di fattore di ramificazione basso,

corrisponde un alto livello di qualità dei *cluster*. Inoltre, è evidente che il nostro sistema individua correttamente anche *cluster* allungati.

6.5 Scalabilità

Lo scopo di questa sezione è quello di dimostrare che il nostro sistema è sostanzialmente scalabile rispetto all'aumentare del numero di punti, e inoltre produce *cluster* di buona qualità pur processando dati ad alta dimensionalità.

Le prove sperimentali relative alla scalabilità sono state eseguite su un calcolatore avente le caratteristiche riportate in Tabella § 6.5.

Tabella 6.5: Caratteristiche del calcolatore su cui sono state effettuate le prove di scalabilità

Processore	Intel® Xeon© CPU 5130 @ 2.00 GHz
Memoria	5.80 GB
Sistema Operativo	Ubuntu Release 9.10 (karmic) Kernel Linux 2.6.31-22-server GNOME 2.28.1

La Tabella § 6.6 nella pagina successiva, mostra varie prove eseguite modificando i parametri relativi al numero di punti, profondità massima dell'albero e fattore di ramificazione.

Tabella 6.6: Scalabilità del sistema

Tempo	N° Punti Processati	Dimensionalità	Qualità <i>Cluster</i>	Massima Profondità	B=L	Numero di <i>Re-build</i>
16 sec	1.000	6	0.012	10	2	2
64 sec	10.000	6	0.049	10	2	7
776 sec (≈ 13m)	60.000	6	0.080	15	2	5
1819 sec (≈ 30m)	100.000	6	0.070	15	2	5
41.646 sec (≈ 11h)	500.000	6	0.018	20	2	4
345.580 sec (≈ 96h)	566.000	6	0.041	10	5	17
777.800 sec (≈ 216h)	810.000	6	0.059	10	5	24
172.800 sec (≈ 48h)	1.126.310	6	0.039	20	2	9

Capitolo 7

Sviluppi Futuri

Nessuno ha concluso niente: rimandiamo sempre tutto al futuro.

SENECA

SUCCESSIVI lavori di tesi possono prendere in considerazione lo sviluppo degli aspetti che andiamo di seguito ad elencare.

1. Attualmente il sistema non dispone di un’interfaccia grafica; dotare **OLAPBIRCH** di un’interfaccia grafica, renderebbe l’utilizzo del sistema più facile, e fruibile ad un bacino di utenza meno specializzato.
2. Condizione necessaria per l’effettivo utilizzo del sistema è quella di dotare il sistema di *query* per il recupero dei punti che compongono ciascun *cluster*, a qualsiasi livello della gerarchia.
3. Il *CF-Tree* attualmente lavora solo su attributi metrici; una possibile evoluzione potrebbe essere quella di rendere il *CF-Tree* operabile su attributi non metrici;
4. Il valore di soglia è limitato al concetto di raggio o diametro; una possibile evoluzione dell’algoritmo proposto, potrebbe essere quello di pensare al valore di soglia come ad un valore basato sulla densità.
5. Per ottenere una migliore integrazione di **OLAPBIRCH** con le tecnologie *OLAP*, si potrebbe pensare di costruire, a partire dall’esito del *DBSCAN*, un dendrogramma

fruibile al motore *OLAP*. In tal modo non si limiterebbe la crescita in profondità dell’albero dei *CF*, operazione che produce un numero consistente di rebuild, riducendo la qualità complessiva dei *cluster*.

6. Come algoritmo di *clustering* globale, abbiamo scelto il *DBSCAN*, uno tra gli algoritmi di *clustering* basato sulla densità largamente trattato in letteratura. Un possibile miglioramento del sistema potrebbe essere quello di utilizzare un algoritmo di *clustering* differente, che produca *cluster* di migliore qualità.

Capitolo 8

Conclusioni

La vita è l'arte di trarre conclusioni sufficienti da premesse insufficienti

SAMUEL BUTLER

NEL progetto di ricerca condotto, è stato presentato il recente campo di studi interdisciplinare denominato *OLAP Mining*.

Basandoci sulla letteratura riguardante l'argomento abbiamo analizzato il *Data Mining*, ed in particolare il *Clustering*, che sta ricevendo una crescente attenzione da parte degli esperti di *Data Warehousing*.

Analizzando le varie tecniche di *clustering*, abbiamo compreso quali metodologie tra queste, risultassero più adatte per l'integrazione con i sistemi di *Data Warehousing*. Tra queste, abbiamo operato un'ulteriore selezione, che ci ha condotto ad approfondire un algoritmo di *clustering* gerarchico agglomerativo incrementale quale è *BIRCH*.

Al fine di strutturare una corretta integrazione tra l'algoritmo in oggetto e le tecnologie *OLAP*, abbiamo apportato modifiche e, dove necessario, riprogettato le sequenze necessarie alla loro interazione.

I risultati di questo studio hanno condotto ad un'evoluzione dell'algoritmo di *clustering BIRCH* e ad una sua parziale integrazione con il motore *OLAP Mondrian*, con risultati sostanzialmente soddisfacenti.

Analizzando questa tesi in retrospettiva e considerando il fatto che questo campo di studi è in continuo sviluppo ad opera di autorevoli ricercatori scientifici, si spera che questo lavoro sia riuscito a dare una visione introduttiva dell'argomento.

Appendice A

Dimostrazioni del Teorema della Rappresentatività di un *CF*

Definiti infatti

$$\vec{LS} = \sum_{i=1}^N \vec{X}_i \quad e \quad \vec{SS} = \sum_{i=1}^N \vec{X}_i^2 \quad ,$$

\vec{X}_0 (2.1), R (2.2), D (2.3), D_0 (2.4), D_1 (2.5), D_2 (2.6), D_3 (2.7) e D_4 (2.8) possono essere calcolate dai vettori CF dei *cluster* corrispondenti come dimostrano le loro definizioni e le formule seguenti.

- *Centroide*:

$$\vec{X}_0 = \frac{\sum_{i=1}^N \vec{X}_i}{N} = \frac{\vec{LS}}{N}$$

- *Raggio*:

$$\begin{aligned} R &= \sqrt{\frac{\sum_{i=1}^N (\vec{X}_i - \vec{X}_0)^2}{N}} = \sqrt{\frac{\sum_{i=1}^N \vec{X}_i^2}{N} + \frac{\sum_{i=1}^N \vec{X}_0^2}{N} - \frac{2 \sum_{i=1}^N \vec{X}_i \vec{X}_0}{N}} = \\ &= \sqrt{\frac{\vec{SS}}{N} + \left(\frac{\vec{LS}}{N}\right)^2 - 2 \left(\frac{\vec{LS}}{N}\right)^2} = \sqrt{\frac{\vec{SS}}{N} - \left(\frac{\vec{LS}}{N}\right)^2} \end{aligned}$$

- *Diametro*:

$$\begin{aligned} D &= \sqrt{\frac{\sum_{i=1}^N \sum_{j=1}^N (\vec{X}_i - \vec{X}_j)^2}{N(N-1)}} = \sqrt{\frac{\sum_{i=1}^N \sum_{j=1}^N \vec{X}_i^2}{N(N-1)} + \frac{\sum_{i=1}^N \sum_{j=1}^N \vec{X}_j^2}{N(N-1)} - \frac{2 \sum_{i=1}^N \sum_{j=1}^N \vec{X}_i \vec{X}_j}{N(N-1)}} = \\ &= \sqrt{\frac{N\vec{SS}}{N(N-1)} + \frac{N\vec{SS}}{N(N-1)} - \frac{2(\vec{LS})^2}{N(N-1)}} = \sqrt{\frac{2N\vec{SS} - 2\vec{LS}^2}{N(N-1)}} \end{aligned}$$

- *Distanza Euclidea tra centroidi (centroid Euclidian distance)*

$$D_0 = \sqrt{(\vec{X}_{01} - \vec{X}_{02})^2} = \sqrt{\left(\frac{L\vec{S}_1}{N_1} - \frac{L\vec{S}_2}{N_2}\right)^2}$$

- *Distanza Manhattan tra centroidi (centroid Manhattan distance)*

$$D_1 = \left| \vec{X}_{01} - \vec{X}_{02} \right| = \left| \frac{L\vec{S}_1}{N_1} - \frac{L\vec{S}_2}{N_2} \right|$$

- Distanza media *inter-cluster* (*average inter-cluster distance*)

$$\begin{aligned} D_2 &= \sqrt{\frac{\sum_{i=1}^{N_1} \sum_{j=N_1+1}^{N_1+N_2} (\vec{X}_i - \vec{X}_j)^2}{N_1 N_2}} = \\ &= \sqrt{\frac{\sum_{i=1}^N \sum_{j=N_1+1}^{N_1+N_2} \vec{X}_i^2 + \sum_{i=1}^N \sum_{j=N_1+1}^{N_1+N_2} \vec{X}_j^2 - 2 \sum_{i=1}^N \sum_{j=N_1+1}^{N_1+N_2} \vec{X}_i \vec{X}_j}{N_1 N_2}} = \\ &= \sqrt{\frac{N_2 \vec{S}_1 + N_1 \vec{S}_2 - 2(L\vec{S}_1 * L\vec{S}_2)}{N_1 N_2}} = \sqrt{\frac{N_2 * S\vec{S}_1 + N_1 * S\vec{S}_2 - 2L\vec{S}_1 L\vec{S}_2}{N_1 N_2}} \end{aligned}$$

- *Distanza media intra-cluster (average intra-cluster distance)*

$$\begin{aligned} D_3 &= \sqrt{\frac{\sum_{i=1}^{N_1} \sum_{j=1}^{N_1+N_2} (\vec{X}_i - \vec{X}_j)^2}{(N_1 + N_2)(N_1 + N_2 - 1)}} = \\ &= \sqrt{\frac{\sum_{i=1}^{N_1+N_2} \sum_{j=1}^{N_1+N_2} \vec{X}_i^2 + \sum_{i=1}^{N_1+N_2} \sum_{j=1}^{N_1+N_2} \vec{X}_j^2 - 2 \sum_{i=1}^{N_1+N_2} \sum_{j=1}^{N_1+N_2} \vec{X}_i \vec{X}_j}{(N_1 + N_2)(N_1 + N_2 - 1)}} = \\ &= \left(\frac{\sum_{i=1}^{N_1} \sum_{i=N_1+1}^{N_1+N_2} \sum_{j=1}^{N_1+N_2} \vec{X}_i^2 + \sum_{i=1}^{N_1} \sum_{i=N_1+1}^{N_1+N_2} \sum_{j=1}^{N_1+N_2} \vec{X}_j^2}{(N_1 + N_2)(N_1 + N_2 - 1)} + \right. \\ &\quad \left. - \frac{2 \sum_{i=1}^{N_1} \sum_{i=N_1+1}^{N_1+N_2} \sum_{j=N_1+1}^{N_1+N_2} \vec{X}_i \vec{X}_j}{(N_1 + N_2)(N_1 + N_2 - 1)} \right)^{\frac{1}{2}} \\ &= \sqrt{\frac{(N_1 + N_2)(S\vec{S}_1 + S\vec{S}_2) + (N_1 + N_2)(\vec{S}\vec{S}_1 + \vec{S}\vec{S}_2) - 2(L\vec{S}_1 * L\vec{S}_2)(L\vec{S}_1 * L\vec{S}_2)}{(N_1 + N_2)(N_1 + N_2 - 1)}} = \\ &= \sqrt{\frac{2(N_1 + N_2)(S\vec{S}_1 + S\vec{S}_2) - 2(L\vec{S}_1 * L\vec{S}_2)^2}{(N_1 + N_2)(N_1 + N_2 - 1)}} \end{aligned}$$

- *Distanza ad incremento di varianza (variance increase distance)*

$$D_4 = \sqrt{\sum_{k=1}^{N_1+N_2} \left(\vec{X}_k - \frac{\sum_{l=1}^{N_1+N_2} \vec{X}_l}{N_1 + N_2} \right)^2 - \sum_{i=1}^{N_1} \left(\vec{X}_i - \frac{\sum_{l=1}^{N_1} \vec{X}_l}{N_1} \right)^2 - \sum_{j=N_1+1}^{N_1+N_2} \left(\vec{X}_j - \frac{\sum_{l=N_1+1}^{N_1+N_2} \vec{X}_l}{N_2} \right)^2}$$

Suddividendo D_4 come segue:

$$\begin{aligned}
 1. \quad & \sum_{k=1}^{N_1+N_2} \left(\vec{X}_k - \frac{\sum_{l=1}^{N_1+N_2} \vec{X}_l}{N_1+N_2} \right)^2 = \\
 & = \sum_{k=1}^{N_1+N_2} \left(\vec{X}_k - \frac{\sum_{l=1}^{N_1} \sum_{l=N_1+1}^{N_1+N_2} \vec{X}_l}{N_1+N_2} \right)^2 = \sum_{k=1}^{N_1+N_2} \left(\vec{X}_k - \frac{L\vec{S}_1 + L\vec{S}_2}{N_1+N_2} \right)^2 = \\
 & = \sum_{k=1}^{N_1+N_2} \vec{X}_k^2 + \sum_{k=1}^{N_1+N_2} \left(\frac{L\vec{S}_1 + L\vec{S}_2}{N_1+N_2} \right)^2 - 2 \sum_{k=1}^{N_1+N_2} \vec{X}_k \left(\frac{L\vec{S}_1 + L\vec{S}_2}{N_1+N_2} \right) = \\
 & = \sum \vec{X}_k^2 + \sum_{k=N_1+1}^{N_1+N_2} \vec{X}_k^2 + N_1 + N_2 \left(\frac{L\vec{S}_1 + L\vec{S}_2}{N_1+N_2} \right)^2 + \\
 & \quad - 2 \left(\sum_{k=1}^{N_1} \vec{X}_k + \sum_{k=N_1+1}^{N_1+N_2} \vec{X}_k \right) \left(\frac{L\vec{S}_1 + L\vec{S}_2}{N_1+N_2} \right) = \\
 & = \vec{S}\vec{S}_1 + \vec{S}\vec{S}_2 + N_1 + N_2 \left(\frac{L\vec{S}_1 + L\vec{S}_2}{N_1+N_2} \right)^2 - 2 \left(L\vec{S}_1 + L\vec{S}_2 \right) \left(\frac{L\vec{S}_1 + L\vec{S}_2}{N_1+N_2} \right) = \\
 & = \vec{S}\vec{S}_1 + \vec{S}\vec{S}_2 + \frac{(L\vec{S}_1 + L\vec{S}_2)^2}{N_1+N_2} - 2 \frac{(L\vec{S}_1 + L\vec{S}_2)^2}{N_1+N_2} = \vec{S}\vec{S}_1 + \vec{S}\vec{S}_2 - \frac{(L\vec{S}_1 + L\vec{S}_2)^2}{N_1+N_2} \\
 2. \quad & \sum_{i=1}^{N_1} \left(\vec{X}_i - \frac{\sum_{l=1}^{N_1} \vec{X}_l}{N_1} \right)^2 = \sum_{i=1}^{N_1} \left(\vec{X}_i - \frac{L\vec{S}_1}{N_1} \right)^2 = \\
 & = \sum_{i=1}^{N_1+N_2} \vec{X}_i^2 + \sum_{i=1}^{N_1} \left(\frac{L\vec{S}_1}{N_1} \right)^2 - 2 \sum_{i=1}^{N_1} \vec{X}_i \left(\frac{L\vec{S}_1}{N_1} \right) = \\
 & = \vec{S}\vec{S}_1 + N_1 \left(\frac{L\vec{S}_1}{N_1} \right)^2 - 2L\vec{S}_1 \frac{L\vec{S}_1}{N_1} = \vec{S}\vec{S}_1 + \frac{(L\vec{S}_1)^2}{N_1} - 2 \frac{(L\vec{S}_1)^2}{N_1} = \\
 & = \vec{S}\vec{S}_1 - \frac{(L\vec{S}_1)^2}{N_1} \\
 3. \quad & \sum_{j=N_1+1}^{N_1+N_2} \left(\vec{X}_j - \frac{\sum_{l=N_1+1}^{N_1+N_2} \vec{X}_l}{N_2} \right)^2 = \sum_{j=N_1+1}^{N_1+N_2} \left(\vec{X}_j - \frac{L\vec{S}_2}{N_2} \right)^2 = \\
 & = \sum_{j=N_1+1}^{N_1+N_2} \vec{X}_j^2 + \sum_{j=N_1+1}^{N_1+N_2} \left(\frac{L\vec{S}_2}{N_2} \right)^2 - 2 \sum_{j=N_1+1}^{N_1+N_2} \vec{X}_j \left(\frac{L\vec{S}_2}{N_2} \right) = \\
 & = \vec{S}\vec{S}_2 + N_2 \left(\frac{L\vec{S}_2}{N_2} \right)^2 - 2L\vec{S}_2 \frac{L\vec{S}_2}{N_2} = \vec{S}\vec{S}_2 + \frac{(L\vec{S}_2)^2}{N_2} - 2 \frac{(L\vec{S}_2)^2}{N_2} = \\
 & = \vec{S}\vec{S}_2 - \frac{(L\vec{S}_2)^2}{N_2}
 \end{aligned}$$

è possibile riscrivere D_4 come segue:

$$\begin{aligned}
 D_4 &= \sqrt{1 - 2 - 3} = \\
 &= \sqrt{\vec{S}\vec{S}_1 + \vec{S}\vec{S}_2 - \frac{(\vec{L}\vec{S}_1 + \vec{L}\vec{S}_2)^2}{N_1 + N_2} - \vec{S}\vec{S}_1 + \frac{(\vec{L}\vec{S}_1)^2}{N_1} - \vec{S}\vec{S}_2 + \frac{(\vec{L}\vec{S}_2)^2}{N_2}} = \\
 &= \sqrt{\frac{(\vec{L}\vec{S}_1)^2}{N_1} + \frac{(\vec{L}\vec{S}_2)^2}{N_2} - \frac{(\vec{L}\vec{S}_1 + \vec{L}\vec{S}_2)^2}{N_1 + N_2}}
 \end{aligned}$$

Appendice B

Algoritmo di Inserzione in un *CF-Tree*

Algoritmo 3: Insert_Into_CF-Tree

Input: Node** *Root*, Node* *CurrentNode*, Entry *Ent*, float *T*
Output: Node*

```
1: Node* NewNode; Entry NewEntry
2: if ( CurNode is Nonleaf Node ) {
3:   Ci = Closest_Child(CurNode, Ent)
4:   NewNode = Insert_Into_CF_tree(Root, Ci, Ent, T)
5:   if ( NewNode == NULL ) {
6:     Update_CF(CurNode, Ci, Ent)
7:     return null
8:   } else {
9:     Update_CF(CurNode, Ci, Ent)
10:    NewEnt = Make_Entry_From_Node(NewNode)
11:    NewNode = Insert_To_NonLeafNode_Might_Split(CurNode, NewEnt)
12:    if ( NewNode == NULL ) {
13:      Merge_Closest_But_Not_Split_Pair_Might_Resplit(CurNode)
14:      return null
15:    } else {
16:      if ( CurNode == *Root ) {
17:        *Root = Create_New_Root(CurNode, NewNode)
18:        return NULL
```

```
19:         } else {
20:             return NewNode
21:         }
22:     }
23: }
24: } else {
25:     /* Questo è un commento */
26:      $L_i = \text{Closest\_Entry}(CurNode, Ent)$ 
27:     if ( Absorb( $L_i$ , Ent) Satisfies  $T$  ) {
28:         Absorb( $L_i$ , Ent)
29:         return null
30:     } else {
31:         NewNode = Insert_To_LeafNode_Might_Split(CurNode, Ent)
32:         if ( NewNode == NULL ) {
33:             return NULL
34:         } else {
35:             return NewNode
36:         }
37:     }
38: }
```

Appendice C

Algoritmo di *Rebuilding* di un *CF-Tree*

Algoritmo 4: Re-build_CF-Tree

Input: $t_i, t_{i+1}, T_i, T_{i+1}$

```
1:  $t_{i+1} = \text{NULL}$ 
2:  $\text{CurrentPath} = \text{Path\_of\_Tree}(t_i, (0, \dots, 0))$ 
3: while (  $\text{CurrentPath}$  exists ) {
4:    $\text{Attach\_Nodes\_To\_NewTree\_By\_CurrentPath}(t_{i+1}, \text{CurrentPath})$ 
5:   foreach ( leaf entry on  $\text{CurrentPath}$  of OldTree, say  $\text{CurrentEntry}$  ) {
6:      $\text{Status} = \text{Can\_Fit\_In\_ClosestPath}(t_{i+1}, T_{i+1}, \text{ClosestPath}, \text{CurrentEntry})$ 
7:     if (  $\text{Status} == \text{YES} \&\& \text{ClosestPath} < \text{CurrentPath}$  ) {
8:        $\text{Fit\_In\_Path}(t_{i+1}, T_{i+1}, \text{ClosestPath}, \text{CurrentEntry})$ 
9:     } else {
10:       $\text{Fit\_In\_Path}(t_{i+1}, T_{i+1}, \text{CurrentPath}, \text{CurrentEntry})$ 
11:    }
12:  }
13:   $\text{Free\_CurrentPath}(t_i, \text{CurrentPath})$ 
14:   $\text{CurrentPath} = \text{Next\_Path\_Of\_Tree}(t_i, \text{CurrentPath})$ 
15: }
16:  $t_{i+1} = \text{Free\_Empty\_Nodes}(t_{i+1})$ 
```

Ringraziamenti

*Nella nostra vita quotidiana, dobbiamo capire che
non è la felicità che ci rende grati,
ma la gratitudine che ci rende felici*
(Albert P. Clark)

Al termine di questo lavoro e percorso, sento il bisogno di rivolgere un ringraziamento ad una serie di persone, grazie alle quali questo lavoro ha potuto essere svolto e portato a termine.

Il primo ringraziamento va alla mia Mamma, per aver sempre creduto in me, nelle mie capacità, nonostante tutto e tutti; per aver dato alla mia vita un significato che non avevo il diritto di pretendere; per i duri sacrifici che ha dovuto affrontare per farmi dono di un'istruzione e diventare ciò che sono oggi.

La mia riconoscenza va al Prof. Ceci, per la professionalità e la cordialità con le quali ha diretto il presente lavoro e, soprattutto, per la passione con la quale svolge il suo lavoro e per come trasmette la stessa, ai suoi studenti.

Rivolgo un pensiero di gratitudine al mio collega di tesi Fabio, per aver creduto nella mia professionalità, piuttosto che a falsi pregiudizi.

Colgo l'occasione per ringraziare tutti quegli educatori della mia vita che, inconsapevolmente, hanno colmato quell'Assenza, dispensando consigli preziosi che mi hanno aiutato a crescere e maturare: il Prof. Scaraggi, la maestra Tonia, Angela e Grazia.

Un ringraziamento speciale va a mio fratello, per avermi fortemente voluta e perché, grazie a lui ho potuto assaporare, nella sua interezza, una meravigliosa opportunità chiamata vita.

Una carezza a Michele, che mi ha supportato e sopportato...

Grazie anche ad Annamaria, per aver reso le ore del *part-time* un'esperienza costruttiva e di grande valore, un'esperienza nella quale ho avuto modo di scoprire un tesoro: una sincera amicizia.

Grazie ai miei due angeli custodi: la nonna da lassù, e il nonno da quaggiù che con immenso amore mi guidano costantemente.

Grazie a tutti, soprattutto a quelli che non hanno mai creduto in me e che mi hanno ostacolato, perché mi hanno permesso di attingere a piene mani dalle vasche della buona volontà, portandomi a concludere questo percorso di studi.

Infine, in occasione del 150° anniversario dell'Unità d'Italia, colgo l'occasione per ringraziare Tutti coloro i quali mi hanno permesso di viverLa *una, libera e preziosa*.

Ed ora inizia la parte difficile...

Bibliografia

- [ABKS99] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: ordering points to identify the clustering structure. In *Proceedings of the 1999 ACM SIGMOD international conference on Management of data, SIGMOD '99*, pages 49–60, New York, NY, USA, 1999. ACM.
- [AGGR98] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopoulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data, SIGMOD '98*, pages 94–105, New York, NY, USA, 1998. ACM.
- [Bil98] Jeff Bilmes. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models, 1998.
- [BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: an efficient and robust access method for points and rectangles. *SIGMOD Rec.*, 19:322–331, May 1990.
- [BL97] Michael J. Berry and Gordon Linoff. *Data Mining Techniques: For Marketing, Sales, and Customer Support*. John Wiley & Sons, Inc., New York, NY, USA, 1997.
- [CKS88] Peter Cheeseman, James Kelly, and Matthew et al. Self. Autoclass: A bayesian classification system. *Proc of the Fifth Intl Workshop on Machine Learning*, pages 54–64, 1988.
- [CLR95] Michael Cheng, Miron Livny, and Raghu Ramakrishnan. Visual analysis of stream data. 1995.

- [DH73] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, New York, 1973.
- [DJ80] Richard C. Dubes and Anil K. Jain. Clustering methodologies in exploratory data analysis. *Advances in Computers*, 19:113–228, 1980.
- [EKSX96] Martin Ester, Hans-Peter Kriegel, Joerg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Evangelos Simoudis, Jiawei Han, and Usama M. Fayyad, editors, *Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231. AAAI Press, 1996.
- [Fis87] Douglas H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Mach. Learn.*, 2:139–172, September 1987.
- [Fis96] Doug Fisher. Iterative optimization and simplification of hierarchical clusterings, technical report cs- 95-01. Technical report, Nashville, TN 37235, USA, 1996.
- [FPSM91] William J. Frawley, Gregory Piatetsky-Shapiro, and Christopher J. Matheus. Knowledge discovery in databases: An overview. In *Knowledge Discovery in Databases*, pages 1–30. AAAI/MIT Press, 1991.
- [FPSS96] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, pages 37–54, 1996.
- [FS84] E. A. Feigenbaum and H. Simon. Epam-like models of recognition and learning. *Cognitive Science*, 8:305–336, 1984.
- [GG91] Allen Gersho and Robert M. Gray. *Vector quantization and signal compression*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.
- [GLF89] J. H. Gennari, P. Langley, and D. Fisher. Models of incremental concept formation. *Artif. Intell.*, 40:11–61, September 1989.
- [GMGB06] Ashwani Garg, Ashish Mangla, Neelima Gupta, and Vasudha Bhatnagar. Pbirch: A scalable parallel clustering algorithm for incremental data. In *Proceedings of the 10th International Database Engineering and Applications*

- Symposium*, pages 315–316, Washington, DC, USA, 2006. IEEE Computer Society.
- [GRS98] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: an efficient clustering algorithm for large databases. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 73–84, New York, NY, USA, 1998. ACM.
- [Gut84] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. *SIGMOD Rec.*, 14:47–57, June 1984.
- [HK98] Alexander Hinneburg and Daniel A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *KDD*, pages 58–65, 1998.
- [HK00] Jiawei Han and Micheline Kamber. *Data mining: concepts and techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.
- [HS94] Marcel Holsheimer and Arno P.J.M. Siebes. Data mining: the search for knowledge in databases. Technical report, Amsterdam, The Netherlands, The Netherlands, 1994.
- [HW79] J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *Applied Statistics*, 28:100–108, 1979.
- [Jos03] Manasi N. Joshi. Parallel k- means algorithm on distributed memory multiprocessors, project report. 2003.
- [KHK99] G. Karypis, Eui-Hong Han, and V. Kumar. Chameleon: hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75, Aug 1999.
- [Koh88] Teuvo Kohonen. *Self-organized formation of topologically correct feature maps*, pages 509–521. MIT Press, Cambridge, MA, USA, 1988.
- [KR90] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data An Introduction to Cluster Analysis*. Wiley Interscience, New York, 1990.
- [Leb87] Michael Lebowitz. Experiments with incremental concept formation: Unimem. *Mach. Learn.*, 2:103–138, September 1987.

- [Mac67] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.
- [MG06] S. Rizzi M. Golzanelli. *Data Warehouse Teoria e pratica della progettazione*. 2 edition, January 2006 2006.
- [Mur83] Fionn Murtagh. A survey of recent advances in hierarchical clustering algorithms. *Comput. J.*, 26(4):354–359, 1983.
- [NH94] Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB ’94, pages 144–155, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.
- [Ols93] Clark F. Olson. Parallel algorithms for hierarchical clustering, technical report. Technical report, Amsterdam, The Netherlands, The Netherlands, Dec 1993.
- [SCZ98] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang. Wa-vecluster: A multi-resolution clustering approach for very large spatial databases. In *VLDB*, pages 428–439, 1998.
- [SD91] Jude W. Shavlik and Thomas E. Dietterich. *Readings in Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 1991.
- [WYM97] Wei Wang, Jiong Yang, and Richard R. Muntz. Sting: A statistical information grid approach to spatial data mining. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB ’97, pages 186–195, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.
- [Zha97] Tian Zhang. *Data clustering for very large datasets plus applications*. PhD thesis, Madison, WI, USA, 1997.

- [ZRL95] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: An efficient data clustering method for very large databases, technical report. Technical report, Madison, WI, USA, 1995.
- [ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. *SIGMOD Rec.*, 25:103–114, June 1996.

