UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA
Facoltà di Ingegneria - Sede di Modena
Corso di Laurea in Ingegneria Informatica

Elaborazione delle interrogazioni in un sistema a mediatore: Fusione dei dati con Risoluzione di conflitti

Relatore
Chiar.mo Prof. Sonia Bergamaschi

Tesi di Laurea di
Marco Mattioli

Correlatore
Prof. Ing. Domenico Beneventano

Controrelatore
Prof. Ing. Paolo Tiberio

Anno Accademico 2002 - 2003

Parole chiave:

Risoluzione dei conflitti
Fusione dei dati
Full Disjunction
Ciclicitá degli ipergrafi
Riscrittura della query

RINGRAZIAMENTI

Ringrazio tutti coloro che mi hanno supportato durante la stesura di questa tesi, aiutandomi a concluderla, e tutti coloro che lo hanno fatto anche prima, dandomi la possibilitá di arrivare a cominciarla.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Il mondo di Internet ha conosciuto qualche anno fa un incremento esponenziale sia nel numero di utenti che nella quantità di risorse; questo fenomeno continua ancora oggi, favorito dalla maggiore accessibilità di programmi e sistemi operativi da parte di utenti anche inesperti.
Questo boom ha ovviamente portato ad un'esplosione anche delle informazioni reperibili; per questo oggi tramite Internet è possibile trovare davvero di tutto. Questo processo ha però anche degli svantaggi: la natura "libera" di questo mezzo di diffusione ha fatto sì che ognuno mettesse a disposizione le proprie informazioni non secondo uno standard, ma seguendo criteri soggettivi. Per questo motivo le sorgenti di informazione risultano essere eterogenee e quindi risulta difficile integrare informazioni provenienti da fonti differenti. In particolare, è molto attuale il problema di integrare dati strutturati, come quelli immagazzinati nelle basi di dati; questo perché il compito di rispondere ad una query non è per niente banale. L'utente infatti, soprattutto se inesperto, desidera una risposta omogenea, quindi è necessario fare in modo di rendere trasparente il processo di integrazione sia dei dati che degli schemi dei database. La trattazione di questo problema è particolarmente complessa, ed è resa ulteriormente difficoltosa per via delle sue innumerevoli sfaccettature. Questa tesi si propone di fornire una soluzione per una parte di questo problema: quella riguardante la fusione dei dati. Si parla di questo aspetto durante la fase di integrazione dei dati; nonostante a prima vista sembri solo una piccola parte del problema, in realtà gli aspetti da tenere in considerazione sono numerosi e solo per alcuni di questi esistono in letteratura diversi studi. La soluzione verrà proposta sia in termini più generali che relativi ad un contesto ben preciso, nel quale poi la soluzione dovrà trovare posto in futuro. Per la precisione si parla dell'ambito MOMIS-SEWASIE, un progetto europeo che mira ad integrare basi di dati indipendenti. Il resto della tesi è strutturato come segue: il capitolo 2 analizza a fondo quelle che sono le principali difficoltà nella fusione dei dati e propone alcune soluzioni. Il capitolo 3 entra nello specifico, analizzando e

mostrando i limiti dei principali metodi per ottenere da database indipendenti una risposta che preservi tutte le connessioni tra i fatti e proponendone uno nuovo. Il capitolo 4 specializza i concetti espressi nei capitoli precedenti nell'ambito di un sistema a mediatore come MOMIS. Infine il capitolo 5 conclude il lavoro.

Capitolo 2

La fusione dei dati

Quello dell'integrazione di database eterogenei è un problema tuttora molto sentito in ambito di ricerca. La soluzione non è semplice, in quanto vi sono svariati aspetti critici da tenere in considerazione: dall'integrazione degli schemi, per fornire all'utente una vista omogenea, si passa alla pulizia dei dati per arrivare all'aspetto della fusione dei dati provenienti da diverse fonti.
La maggior parte dei lavori di ricerca si è concentrata sull'integrazione degli schemi, mentre poco è stato ancora fatto per quanto riguarda la fusione dei dati, mettendolo quasi in secondo piano. Ancora meno sono le ricerche che riguardano fonti non collaborative. Invece, quello della fusione è un aspetto molto importante e pone diversi problemi interessanti da trattare e assolutamente non banali.
Principalmente, la ricerca si è sviluppata in due ambiti: la qualità dei dati e la risoluzione dei conflitti.

2.1 La qualità nei dati

Quello della qualità è un argomento in piena fase di sviluppo, ma la soggettività della definizione stessa di "qualità" complica ulteriormente le cose. Non è infatti facile definire oggettivamente quello che è da considerarsi importante per la qualità di un dato e quello che può essere trascurato.
Ossia, non è ancora stato definito uno standard secondo il quale un dato può essere visto assolutamente come qualitativamente migliore di un altro. Probabilmente questo standard non verrà introdotto nemmeno in futuro, proprio perché l'importanza dei vari aspetti della qualità varia sì in base all'utente, ma anche in base all'ambito dell'applicazione.
Ad esempio, alcune persone potrebbero preferire una risposta più accurata anche se più lenta, mentre altri potrebbero pretendere la velocità innanzitutto, a discapito della completezza della risposta. Inoltre, un'applicazione atta a mostrare l'anda-

mento della borsa in tempo reale dovrà mirare soprattutto alla velocità, mentre in altri ambiti (ricerca di un libro, ad esempio) potrebbero essere altri i parametri più significativi.

In linea di principio si possono dividere gli aspetti concernanti la qualità in tre gruppi principali:

- **costo:** in questa categoria ricadono tutti i parametri che hanno a che fare con i costi, sia temporali (es: tempo di risposta) che monetari (es: costo di una query).

- **tempo:** comprende sia la parte finanziaria legata al tempo (es: costo di sviluppo e di mantenimento) che parametri come la disponibilità o l'efficienza.

- **utilità:** in questa ultima categoria ricadono moltissimi parametri, riguardanti il tempo (es: tempo di update) la consistenza dei dati (es: integrità, coerenza), la convenienza (es: sicurezza, accuratezza) e i servizi di valore aggiunto (es: scelta di opzioni).

Fra tutte le categorie si possono contare più di sessanta parametri; un numero senza dubbio proibitivo, che rende inutile qualsiasi approccio che li tenga in considerazione tutti.

Comunque, molti sforzi sono stati fatti per limitare il numero di parametri. In particolare, in [10] si tenta di stabilire quali siano i criteri più importanti per definire la qualità di una sorgente:

- **accuratezza:** è definita come "il grado in cui i dati sono corretti, affidabili e certificati". Costituisce una misura di prossimità fra il valore assunto da un dato ed un altro valore ritenuto corretto.
  L'accuratezza può essere danneggiata da errori di sintassi, da ambiguità nella rappresentazione o da valori troppo vecchi. Altre cause di inaccuratezza possono essere causate da rilevazioni di campioni effettuate manualmente piuttosto che tramite sensori.
  Una misura dell'accuratezza può essere associata ad una sorgente come il rapporto fra il numero corretto dei valori ed il numero totale dei valori.

- **completezza:** è il grado in cui valori specifici di un dato sono inclusi nella collezione di dati.
  Alcune fonti la vedono come dipendente dal compito che si sta eseguendo.

- **consistenza:** è definita come la proprietà dei dati di non entrare in conflitto con altri.
  Può anche essere definita su tre livelli:

  - *consistenza delle viste:* è definita come consistenza semantica se sono considerati i componenti della vista e come consistenza strutturale se si riferisce agli attributi fra le varie entità.
  - *consistenza dei valori:* esamina i conflitti fra i valori dei dati. Si ha inconsistenza quando due o più valori non possono essere considerati corretti contemporaneamente.
  - *consistenza della rappresentazione:* si occupa del fatto che i dati siano rappresentati con lo stesso formato corretto.

- **aggiornamento:** è definito come l'intervallo di tempo fra l'ultimo update di un dato ed il momento in cui lo si usa.
  Oppure può essere considerato come il grado in cui i dati sono aggiornati. Per calcolarlo si può utilizzare la seguente formula:

$$Aggiornamento = Età + (IstanteDiUtilizzo - IstanteDiInput)$$

  dove $Età$ esprime quanto era vecchio il dato al momento dell'immissione, $IstantediUtilizzo$ è il momento in cui l'utente accede al dato e $IstantediInput$ è il momento in cui il dato è stato inserito.
  Si può anche definire il *livello di aggiornamento* di una sorgente, dato dal rapporto fra i dati ritenuti aggiornati e l'insieme di tutti i dati.

- **volatilità:** può essere vista come il periodo di tempo durante il quale un dato rimane valido o come la frequenza di variazione del valore di un dato.

- **tempestività:** è definita tramite una formula matematica:

$$Tempestività = \left\{ max\left[1 - \frac{Aggiornamento}{Volatilità}; 0\right] \right\}$$

  Il risultato è compreso fra 0 e 1. Più il valore è alto, più il dato è aggiornato. Se il risultato è 0, allora il dato è considerato vecchio e quindi inaffidabile.

- **interpretabilità:** si riferisce al formato in cui i dati sono presentati all'utente. Differenti formati possono portare a vari gradi di qualità, in quanto uno può essere più comprensibile di un altro. La qualità percepita può variare anche in base al compito a cui i dati sono preposti, poiché in generale non esiste un formato migliore a priori.

- **affidabilità:** può essere suddivisa in affidabilità dei dati ed affidabilità delle sorgenti.
  La prima si riferisce alla proprietà dei dati di recare informazioni corrette, mentre la seconda si basa sulla reputazione della sorgente.

- **rilevanza:** definisce quanto i dati estratti sono appropriati per il compito che si sta svolgendo.

- **accessibilità:** è "il grado in cui i dati sono disponibili o facili e veloci da trovare".

- **sicurezza nell'accesso:** è definita come "il grado in cui l'accesso ai dati può essere limitato e quindi mantenuto sicuro". La sicurezza delle informazioni comprende quattro aspetti principali:

  – *segretezza:* si riferisce alla prevenzione ed al rilevamento di accessi non autorizzati ai dati.

  – *integrità:* si riferisce alla prevenzione ed al rilevamento di modifiche improprie ai dati.

  – *disponibilità:* si riferisce alla prevenzione ed al rilevamento di erronei rifiuti nell'accesso a servizi elargiti dal sistema.

  – *non ripudio:* garantisce che un dato sia fornito da una certa sorgente e che questa non possa negarlo.

  Vista come parametro di qualità la sicurezza nell'accesso fornisce un indicazione del livello di sicurezza fornito. Può essere rappresentata tramite una percentuale. Per calcolarla si possono tenere in conto gli aspetti di protezione da accessi impropri e da inferenza, integrità del database e dei dati, autenticazione dell'utente, protezione multilivello.

- **storia:** rappresenta i miglioramenti qualitativi apportati ai dati. Può essere calcolata come una percentuale che tiene in considerazione i miglioramenti nei dati dal momento della loro immissione fino a quello di utilizzo da parte di un utente.

- **costo:** per costo si intende quello dovuto alla bassa qualità. In altri termini, serve per valutare l'impatto del costo degli errori dovuti a dati di scarsa qualità. Per misurarlo è necessario stilare una lista di tutti i possibili errori. Ogni tipo d'errore nella lista avrà associato una *frequenza di errore* ed un *costo di errore* (quest'ultimo specifica il costo medio di errore per unità di dato). Il costo si calcola tramite la seguente formula matematica:

$$Costo = \sum_{i=1}^{N} CostoDiErrore_i * FrequenzaDiErrore$$

dove $Costo$ è il costo globale degli errori che hanno colpito i dati in un certo periodo di tempo, $CostoDiErrore_i$ è il costo di un particolare tipo di errore in una singola unità di dato e $FrequenzaDiErrore$ è il numero medio di errori appartenenti alla categoria i che hanno colpito i dati nel periodo di tempo analizzato.
Per valutare i costi di errore correttamente è bene prendere come riferimento i rispettivi costi dei dati di qualità alta.

si tenga comunque presente che queste sono solo alcune delle definizioni presenti in letteratura. Ve ne sono molte altre possibili, proprio per il grande grado di soggettività del termine "qualità" e delle sue innumerevoli sfaccettature.
Nonostante il numero di parametri analizzato sia limitato, si possono già intravedere le difficoltà che questi introducono.

In primo luogo, solo pochi parametri sono calcolabili tramite una formula matematica (seppur determinata empiricamente), mentre per altri la letteratura propone definizioni anche completamente differenti da quelle qui riportate.
Quindi, non solo la selezione dei parametri da utilizzare è soggettiva, ma anche il calcolo di alcuni di essi lo può diventare.
In secondo luogo, non tutti i parametri sono dipendenti unicamente dai dati mantenuti da una sorgente. L'accessibilità o la sicurezza nell'accesso sono dipendenti in larga parte da meccanismi messi in atto dai gestori delle sorgenti rispettivamente per rendere disponibili e più sicuri i dati, senza considerare i dati effettivamente contenuti.
Altri parametri come il costo o la rilevanza hanno senso solamente se calcolati a run-time, ossia quando si conoscono il compito da svolgere e le eventuali preferenze dell'utente.
Inoltre, sorge il problema di dove mantenere le informazioni sulla qualità delle possibili sorgenti da interrogare.
Per ultimo, va ricordato che l'attribuzione di valori ai parametri qualitativi richiede una conoscenza approfondita sia della sorgente (ogni quanto vengono aggiornati i dati? C'è sicurezza nella transazione?), sia dei compiti che questa sorgente sarà chiamata a svolgere (quali parametri scegliere? A quali di questi attribuire un'importanza maggiore?), sia dei dati stessi in essa contenuti (le rilevazioni dei dati vengono effettuate con strumenti affidabili? Quanti attributi hanno un valore non nullo?).

Nonostante tutte le difficoltà che si possono incontrare, la qualità riveste un ruolo importantissimo in molti campi, soprattutto qundo si parla di fusione.

Molte volte infatti una fusione non accorta di dati provenienti da diverse fonti eterogenee può portare ad avere risultati della stessa qualità di quelli forniti dalle singole sorgenti, se non addirittura inferiore. Per ovviare a questa perdita di informazioni (in quanto è lecito supporre che più qualità porti a più informazione) è necessario dare la precedenza nelle interrogazioni alle sorgenti più promettenti.

2.2 La fusione dei dati

Come già detto, la fusione dei dati è un processo molto delicato, tramite il quale tuple provenienti da sorgenti differenti, ma rappresentanti lo stesso oggetto nel mondo reale, vengono unite in una unica tupla.

Lo scopo della fusione è quello di fornire all'utente una visione omogenea, non ridondante e integra dei dati, anche in presenza di eterogeneità fra le fonti.
In presenza di poche sorgenti è anche possibile presentare all'utente tutti i risultati della richiesta, senza ricorrere alla fusione. Ciò diventa però impossibile quando la quantità di informazione è enorme, come nel caso di Internet.

Esistono due tipi principali di eterogeneità:

- **eterogeneità di schema:** nella stragrande maggioranza dei casi le sorgenti differiscono nello schema delle relazioni. Ad esempio una sorgente potrebbe avere un campo "Nome e Cognome", mentre un'altra potrebbe avere la coppia di attributi "Nome" e "Cognome". Ovviamente all'utente finale questo conflitto dovrà risultare trasparente, quindi sorge il problema di come fondere le tuple provenienti da più sorgenti.

- **eterogeneità semantica:** si parla di eterogeneità semantica quando vi sono conflitti causati dai dati stessi.

Poiché ciò che ai fini della tesi interessa maggiormente è l'eterogeneità semantica, nel proseguimento del capitolo sarà solo questa ad essere presa in considerazione, assumendo che il problema dell'eteroeneità degli schemi sia già stato risolto a monte.
Per questo motivo quando si parlerà di eterogeneità, sarà sottinteso che si tratti di quella semantica; inoltre si supporrà di avere già uno schema globale omogeneo per presentare i risultati all'utente.

La sovrapposizione fra due sorgenti può essere di due tipi: estensionale ed intensionale.
Si verifica una sovrapposizione estensionale quando due o più sorgenti contengono informazioni sullo stesso oggetto del mondo reale. Ad esempio, diverse

sorgenti possono contenere informazioni riguardanti lo stesso elettrodomestico.
Si verifica una sovrapposizione intensionale quando due o più sorgenti hanno attributi in comune. Ad esempio, diverse sorgenti possono fornire l'informazione "Costruttore" per ogni dato elettrodomestico.
Il primo problema da trattare per realizzare la fusione e per risolvere i problemi causati dalle sovrapposizioni è il riconoscimento degli oggetti. Ossia: come si può essere sicuri che tuple presenti in diverse sorgenti (o anche nella stessa sorgente) facciano riferimento allo stesso oggetto reale?
Il problema è di facile risoluzione se si dispone di un identificatore univoco per tutte le sorgenti (ad esempio, a livello statale, il CAP), ma ciò, nel mondo di Internet, avviene molto raramente.
Si pensi ad esempio a più sorgenti dove esista un campo "Nome e Cognome". I valori "Rossi Mario", "Rossi M.", "Ing. Rossi Mario" possono rappresentare lo stesso oggetto nel mondo reale. Molte volte il problema si presenta anche su di un'unica relazione. È sufficiente infatti che siano persone diverse (e quindi con diversi standard) ad inserire i valori, ma anche le tuple introdotte dallo stesso individuo possono avere più di uno standard, soprattutto se l'immissione avviene in diversi momenti, distanti temporalmente fra loro. Lo stesso discorso vale per le date forse anche in misura maggiore, per via delle numerose differenze di formato possibili.
In assenza di un identificatore univoco, bisogna ricorrere ad altri metodi per stabilire l'identità di un oggetto.
Esistono svariate tecniche, molte delle quali prese in prestito dall'ambito dei data warehouse. Infatti il problema dell'identità di un oggetto può essere visto come parte del processo di data cleaning.

Una volta stabilita univocamente l'identità di un oggetto, si può procedere alla fusione vera e propria.
Ammettiamo di avere due tuple t_1 e t_2 appartenenti a relazioni diverse con schemi S_1 e S_2 rispettivamente. Durante il processo di fusione si possono verificare due casi:

- 1. $S_1 \cap S_2 = \emptyset$: ossia, le due tuple non hanno attributi in comune. In questo caso la tupla risultante sarà data semplicemente dall'unione di t_1 e t_2.

- 2.$S_1 \cap S_2 = \{a_1, a_2, \ldots, a_n\}$: dove $a_1, a_2, \ldots, a_n$ sono gli attributi in comune fra le due tuple. Se è questo caso a verificarsi, allora vi sono due ulteriori possibilità:

  - i valori degli attributi in comune sono uguali per entrambe le tuple oppure solo uno dei due è non nullo. In questo caso i valori degli attributi $a_1, a_2, \ldots, a_n$ sono definiti univocamente.

– i valori di uno o più degli attributi in comune variano da una tupla all'altra. In questo caso si genera un conflitto, che andrà risolto utilizzando tecniche appropriate.
Si tenga presente che questo caso non si verifica se è presente l'ipotesi di omogeneità semantica.

Ora, supponendo le tuple t_1 e t_2 con un certo numero di valori nulli come è lecito aspettarsi, vediamo con un esempio cosa accade a seguito del verificarsi di ognuno dei due casi.

- **caso 1.:** supponiamo t_1 e t_2 con il seguente schema:

t_1			t_2		
a	**b**	**c**	**a**	**d**	**e**
a1	1	'a'	a1	NULL	NULL
a2	NULL	'b'	a2	-4	NULL
a3	3	NULL	a3	-3	'x'
a4	4	NULL	a4	0	NULL

Come si può notare, non ci sono attributi in comune fra le due tuple. Il risultato della fusione è riportato nella seguente tabella:

t				
a	**b**	**c**	**d**	**e**
a1	1	'a'	NULL	NULL
a2	NULL	'b'	-4	NULL
a3	3	NULL	-3	'x'
a4	4	NULL	0	NULL

- **caso 2.:** questa volta consideriamo le seguenti tuple:

t_1			t_2		
a	**b**	**c**	**a**	**b**	**c**
a1	1	'a'	a1	NULL	NULL
a2	NULL	'b'	a2	2	NULL
a3	3	NULL	a3	3	'c'
a4	4	NULL	a4	4	NULL

In questo caso le due tuple si sovrappongono completamente. Per semplicità è stata fatta l'ipotesi di omogeneità semantica.
Il risultato è:

t		
a	**b**	**c**
a1	1	'a'
a2	2	'b'
a3	3	'c'
a4	4	NULL

si può notare come il verificarsi del primo caso porti sì a tuple con più attributi, ma in generale queste conterranno più valori nulli. Se invece si ricade nel secondo caso, le tuple avranno meno campi, ma questi conterranno un maggior numero di valori significativi, in virtù della sovrapposizione e quindi della maggiore probabilità di trovare un valore significativo.

A questo punto è chiaro come le eventuali sovrapposizioni fra schemi di relazioni provochino non pochi problemi in caso di fusione. Mentre, fino a poco tempo fa, la sovrapposizione era considerata come l'unico aspetto critico in fase di fusione, recenti studi ([11], [12], [13]) hanno presentato alcune osservazioni che mettono in discussione questa assunzione. In particolare, il discorso è quello della qualità. Per mettere in luce il problema, si consideri il seguente esempio.

Ammettiamo di avere tre fonti L_1, L_2, L_3 dalle quali è possibile reperire diverse informazioni complementari:

- L_1 ha schema $S_1 = \{a_1, a_2, a_3\}$ ed è considerata molto affidabile.
- L_2 ha schema $S_2 = \{a_4, a_5\}$ ed è considerata molto affidabile.
- L_3 ha schema $S_3 = \{a_4\}$ ed è considerata poco affidabile.

Ovviamente, in condizioni normali la soluzione avrà schema $S = \{a_1, a_2, a_3, a_4, a_5\}$ e le fonti interrogate saranno solamente L_1 e L_2.
Ammettiamo invece che la fonte L_2 non sia disponibile per problemi di rete. In problemi in cui la qualità non ha un ruolo fondamentale il buon senso detterebbe di interrogare due fonti rimanenti, per poter così fornire all'utente informazioni su un numero maggiore di attributi.
Considerando invece la qualità, quella appena esposta potrebbe non essere l'unica soluzione.
Le tuple in output avrebbero infatti informazioni sugli attributi a_1, a_2, a_3, a_4 ma, mentre i primi tre attributi sarebbero da considerarsi di ottima qualità, il quarto proverrebbe da una fonte poco attendibile. Ciò porterebbe ad avere tuple con qualità non uniforme, cosa che per alcune applicazioni può risultare poco appropriato.
Per porre rimedio a questo problema sono possibili due alternative:

1. vengono fornite all'utente informazioni solamente sugli attributi di alta qualità, lasciando gli altri a NULL. In questo modo, la qualità della tupla in uscita è preservata, anche se a discapito della completezza.

2. vengono forniti tutti gli attributi insieme ad informazioni sulla qualità degli stessi. In particolare, si possono riportare informazioni sulla fonte. In questo modo l'utente potrà decidere quali siano da ritenersi affidabili secondo i propri personali criteri. Questa soluzione però implica una conoscenza di tutte le fonti (o di almeno la maggior parte di queste) da parte dell'utente finale, ma nel mondo di Internet le fonti sono troppo numerose per conoscerle tutte. Con questo approccio poi si tendono a favorire fonti storicamente affidabili, dato che l'utente è solitamente portato a preferire soluzioni che conosce già, mentre eventuali fonti nuove verrebbero viste sempre con diffidenza, a prescindere dal livello qualitativo offerto.

A questo punto, la domanda che un progettista dovrebbe porsi è: "è meglio un NULL di una fonte affidabile o un valore significativo proveniente da una qualunque altra sorgente?". La risposta non è banale e varia in base al campo applicativo (quanto è importante la qualità? E la completezza?) e al livello di conoscenza degli utenti (informazioni sulle fonti sarebbero interpretabili correttamente dall'utente medio?). Inoltre il senso comune porta a supporre che un valore significativo sia sempre da preferirsi ad un NULL, anche se ciò non sempre è vero: in determinate situazioni il valore NULL può essere considerato come fonte d'informazioni. Ad esempio, in un database di una biblioteca potrebbero essere memorizzati utenti abbonati, e quindi in possesso di un numero di tessera, e non abbonati, per i quali il valore di tessera potrebbe essere rappresentato da un NULL. Qui "NULL" indica la mancanza di tessera da parte di un utente e quindi un'informazione aggiuntiva.

Il problema appena esposto, comunque, interessa l'utente finale solamente in misura limitata. Soprattutto se confrontato con quello della risoluzione dei conflitti.

Quello della risoluzione dei conflitti è un problema di primo piano quando si parla di fusione dei dati. Infatti sono molte le cause che possono portare ad un conflitto, ad esempio:

- errori grammaticali nell'immissione dei dati
- diverso aggiornamento dei dati
- differenze nel rilevamento dei dati

si ricorda si ha un conflitto quando, durante una fusione, un oggetto presenta due o più valori differenti per lo stesso attributo.

In ambito di ricerca, come precedentemente affermato, non esistono molti riferimenti riguardo questo problema; nonostante ciò gli approcci presentati sono molto vari.
Ad esempio c'è chi propone di affrontare il problema dei conflitti semplicemente non risolvendolo ([3]), fornendo all'utente finale tutti i valori conflittuali. Questo metodo non sembra molto promettente per prima cosa perché obbliga l'utente a scegliere un valore, supponendo che abbia una conoscenza dell'ambiente e dei dati abbastanza approfondita, in secondo luogo perché con la presenza di un numero molto alto di fonti (e quindi di possibili conflitti) i valori mostrati all'utente sarebbero troppi e, nella maggior parte dei casi, troppo vari, generando confusione e diminuendo la qualità percepita. Eventualmente si potrebbe pensare di fornire, insieme ai dati conflittuali, la fonte da cui questi provengono. Questo potrebbe aiutare l'utente a scegliere un dato, ma del resto aumenterebbe ulteriormente la confusione del layout.
Un secondo approccio si basa sempre sulla non risoluzione dei conflitti, introducendo l'ipotesi di omogeneità semantica. Secondo questa ipotesi un oggetto ha uno ed un solo valore per ogni attributo, quindi si esclude la possibilità di eventuali conflitti.
Le limitazioni di questo approccio (adottato in via temporanea nel progetto SEWASIE) sono ben evidenti e quindi non verranno ripetute in questa sede.

Un approccio molto interessante è proposto da Bertossi e Chomicki in [14]. Gli autori fanno esplicito riferimento a basi di dati inconsistenti, ossia basi di dati che contengono alcuni dati che non rispettano vincoli di integrità.
Ciò che gli autori cercano di ottenere è una risposta consistente anche su database con questa caratteristica; in particolare si parla di *riparazione* quando si tenta di porre rimedio alle inconsistenze. Poichè l'obiettivo è ottenere una risposta consistente e non un database consistente, non è necessario riparare tutte le inconsistenze del database ma solo quelle che, di volta in volta, vengono interessate da una query. Per capire esattamente cosa si intende per "inconsistenza" si faccia riferimento alla seguente tabella:

Impiegato

nome	salario
J. Page	5000
J. Page	8000
V. Smith	3000
M. Stowe	7000

Si vede come la dipendenza funzionale f : nome→salario sia violata a causa delle prime due tuple. Quindi questo è un database inconsistente, ma ciò non esclude che siano anche presenti informazioni consistenti. Per eliminare l'inconsistenza

in maniera minimale, supponendo di poter solo aggiungere o togliere delle tuple, sono possibili due riparazioni:

Impiegato1	
nome	salario
J. Page	8000
V. Smith	3000
M. Stowe	7000

Impiegato2	
nome	salario
J. Page	5000
V. Smith	3000
M. Stowe	7000

Si nota come le informazioni presenti in entrambe le riparazioni siano quelle consistenti anche precedentemente. Comunque, confrontando le due riparazioni si può vedere che anche altre informazioni possono essere ottenute in maniera consistente; ad esempio si può capire che esiste un impiegato chiamato "J. Page". Fatto che non si sarebbe potuto ottenere con la semplice eliminazione delle tuple partecipanti all'inconsistenza.

Nello studio originale è presente una formalizzazione completa dei possibili vincoli di integrità basata su un linguaggio di logica di primo ordine; qui però verranno riportate solo i vincoli più importanti e comuni nella realtà. Nel seguito i simboli di relazione saranno indicati con $P_1, \ldots, P_n$, tuple di variabili e costanti con $\bar{x}_1, \ldots, \bar{x}_n$.

- dipendenze funzionali (FD):
  nella forma generale si ha: $\forall \bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{x}_4, \bar{x}_5.\ [\neg P(\bar{x}_1, \bar{x}_2, \bar{x}_4) \vee \neg P(\bar{x}_1, \bar{x}_3 \bar{x}_5) \vee \bar{x}_2 = \bar{x}_3]$
  Una formulazione più familiare della FD di cui sopra è $X \rightarrow Y$, dove X è l'insieme degli attributi di P corrispondenti a $\bar{x}_1$ e X è l'insieme degli attributi di P corrispondenti a $\bar{x}_2$ (e $\bar{x}_3$).

- vincoli di integrità referenziale o dipendenze d'inclusione (IND):
  $\forall \bar{x}_1 \exists \bar{x}_3.[\neg Q(\bar{x}_1) \vee P(\bar{x}_2, \bar{x}_3)]$
  dove $\bar{x}_i$ sono sequenze di variabili distinte, con $\bar{x}_2$ contenuta in $\bar{x}_1$ e P e Q sono relazioni del database. I vincoli di questo tipo sono spesso scritti come $Q[Y] \subseteq P[X]$ dove X (Y) è l'insieme degli attributi di P (Q) corrispondenti a $\bar{x}_2$.

La maggior parte dei DBMS commerciali restringe le FD a dipendenze di chiave e le IND a vincoli di foreign key.

Definizione 2.2.1 *Data un'istanza r di un database R ed un insieme di vincoli di integrità IC, diciamo che r è* consistente *se $r \models IC$ nel senso classico del modello teorico (ossia se IC è vero in r), altrimenti è* inconsistente.

Data un'istanza di database r, l'insieme $\Sigma(r)$ dei fatti di r è l'insieme delle formule atomiche $\{P(\bar{a}) \mid r \models P(\bar{a})\}$, dove P è una relazione e a una tupla. La distanza $\Delta(r, r')$ tra le istanze di database r e r' è definita come la differenza simmetrica di r e r':

$$\Delta(r, r') = (\Sigma(r) - \Sigma(r')) \cup (\Sigma(r') - \Sigma(r))$$

Definizione 2.2.2 *Un'istanza di database r' è una* riparazione *di un'istanza di database r rispetto ad un insieme di vincoli d'integrità IC se:*

1) r' è definito sullo stesso dominio e sullo stesso schema di r.

2) r' soddisfa IC.

3) la distanza $\Delta(r, r')$ è minimale tra le istanze che soddisfano le prima due condizioni.

Mentre per i vincoli di tipo FD ogni riparazione di r è un sottoinsieme di r, ciò non vale per vincoli di altro tipo: in generale le riparazioni possono contenere tuple che non appartengono a r. Ad esempio una violazione di un IND può essere risolta non solo eliminando ma anche aggiungendo delle tuple.
Supponiamo di avere un database con due relazioni: personale(CF,nome) e manager(CF). Supponiamo che esistano due FD: CF→nome e nome→CF ed una IND: manager[CF]⊆personale[CF]. Le relazioni hanno le seguenti istanze:

personale	
CF	nome
123456789	Smith
555555555	Jones
555555555	Smith

manager
CF
123456789
555555555

Le istanze non violano la IND, ma violano entrambe le FD. Considerando solo le FD sarebbero possibili due riparazioni: una ottenuta rimuovendo la terza tupla di "personale" e l'altra rimuovendo le prime due tuple delle stessa relazione. Comunque quest'ultima violerebbe la IND. Questo potrebbe essere riparato eliminando la prima tupla di "manager". Quindi, considerando tutti i vincoli sarebbero possibili due riparazioni:

personale	
CF	nome
123456789	Smith
555555555	Jones

manager
CF
123456789
555555555

e

personale		manager
CF	nome	CF
555555555	Smith	555555555

Inoltre ci sarebbero infinite altre riparazioni, ottenute da una combinazione di cancellazioni ed inserimenti, come:

personale		manager
CF	nome	CF
123456789	c	123456789
555555555	Smith	555555555

Dove "c" è un elemento arbitrario del dominio D del database diverso da "Smith". Ora introduciamo il concetto di query congiuntiva secondo la notazione della logica di primo ordine:

$$\exists \bar{x}_1, \ldots, \bar{x}_m.[P_1(\bar{x}_1) \wedge \cdots \wedge P_m(\bar{x}_m) \wedge \phi(\bar{x}_1, dots, \bar{x}_m)]$$

dove $\phi(\bar{x}_1, dots, \bar{x}_m)$ è una congiunzione di predicati atomici.

Definizione 2.2.3 *Una tupla $\bar{t}$ è una* risposta *ad una query $Q(\bar{x})$ in r se e solo se $r \models Q(\bar{t})$, ossia se la formula Q con $\bar{x}$ sostituito da $\bar{t}$ è vera in r.*

Ciò che l'autore vuole portare in uscita sono solo le tuple consistenti, ossia quelle che non violano vincoli di intergrità. Più formalmente:

Definizione 2.2.4 *Una tupla $\bar{t}$ è una* risposta consistente *(CQA) ad una query $Q(\bar{x})$ in un'istanza di database r rispetto ad un insieme di vincoli di integrità IC se e solo se $\bar{t}$ è una risposta alla query $Q(\bar{x})$ in ogni riparazione r' di r rispetto a IC. Una query Q in logica di primo ordine è* consistentemente vera *in r rispetto a IC se è vera in ogni riparazione di r rispetto a IC. In simboli:*

$$r \models_{IC} Q(\bar{t}) \textit{per ogni riparazione } r' \textit{ di } r \textit{ rispetto a } IC$$

Da questa definizione discende che per risolvere inconsistenze l'eliminazione di tutte le tuple che violano i vincoli non è l'unico approccio; infatti, riprendendo l'esempio iniziale, la formula:
$r \models f_1$ (impiegato(J. Page.5000)$\vee$(impiegato(J. Page.8000)
è consistentemente vera.
Per poter rispondere ad una query in modo consistente l'autore mostra due approcci:

- **Trasformazione della query**
  Data una query Q ed un insieme di vincoli IC, si costruisce una query

Q' tale che per ogni istanza r l'insieme delle risposte di Q' in r è uguale all'insieme delle risposte di Q in r rispetto a IC. La query viene riscritta con informazioni derivate dall'interazione tra la query ed i vincoli di integrità, ottenendo il soddisfacimento dei vincoli.

- **Rappresentazione compatta delle riparazioni**
  Dato un insieme di vincoli IC ed un'istanza r di database si costruisce una rappresentazione di tutte le riparazioni di r rispetto a IC e si utilizza questa per rispondere alla query.

Per brevità, qui si riporta solo il processo di trasformazione della query.

Data una query $\varphi(\bar{x})$ viene calcolata una nuova query $T^{\omega}(\varphi(\bar{x}))$ reiterando l'operatore T che trasforma la query congiungendo i corrispondenti *residui* ad ogni letterale di database che appare nella query finchè non viene raggiunto un punto di stabilità. I residui dei letterali dei database forzano il soddisfacimento dei vincoli di integrità per le tuple che soddisfano il letterale e sono ottenuti risolvendo i letterali con i vincoli di integrità.
Per chiarire la definizione supponiamo di avere i seguenti vincoli

$$IC = \{\forall x.[R(x) \vee \neg P(x) \vee \neg Q(x)], \forall x.[P(x) \vee \neg Q(x)]\}$$

e la query $Q(x)$. Il residuo di $Q(x)$ rispetto al primo vincolo è $R(x) \vee \neg P(x)$ in quanto, poichè sia la query $Q(x)$ sia i vincoli devono essere soddisfatti, allora il residuo deve essere vero. Allo stesso modo, il residuo di $Q(x)$ rispetto al secondo vincolo è $P(x)$. Di conseguenza, invece che la query $Q(x)$ si potrebbe utilizzare la query $Q(x) \wedge (R(x) \vee \neg P(x)) \wedge P(x)$. Il letterale $\neg Q(x)$ non ha nessun residuo rispetto ai dati vincoli di integrità, dal momento che questi non la interessano.
Ora prendiamo il primo esempio introdotto per vedere un caso con attributi significativi:
la FD f_1 può essere scritta come:

$$f_1 : \quad \forall x, y, z.[\neg\text{Impiegato}(x, y) \vee \neg\text{Impiegato}(x, z) \vee y = z] \qquad (2.1)$$

Data la query $Q(x, y) : \text{Impiegato}(x, y)$ ci aspettiamo di ottenere le tuple (V.Smith,3000) e (M.Stowe,7000), ossia solo quelle consistenti.
Il residuo ottenuto risolvendo la query con la FD f_1 è:

$$\forall z.[\neg\text{Impiegato}(x, z) \vee y = z]$$

e quindi, riscrivendo la query si ha:

$$T(Q(x, y)) = \text{Impiegato}(x, y) \wedge \forall z.[\neg\text{Impiegato}(x, z) \vee y = z]$$

che restituisce esattamente il risultato atteso. Le tuple (J.Page,5000) e (J.Page,8000) sono escluse in quanto, posto x =J.Page, y =5000 e z =8000, avremmo:

$$T(Q(x,y)) = \text{Impiegato}(\text{J.Page}, 5000) \wedge [\neg \text{Impiegato}(\text{J.Page}, 8000) \vee 5000 = 8000]$$

e quindi il residuo è falso.
Generalmente T ha bisogno di essere reiterato fino al raggiungimento di uno stato di stabilità, ossia $T^{\omega} = T^{\omega-1}$.
Riprendiamo i vincoli 2.1: i residui per ogni letterale sono:

Letterali	Residui
$P(x)$	$: \{R(x) \vee \neg Q(x)\}$
$Q(x)$	$: \{R(x) \vee \neg P(x), P(x)\}$
$\neg P(x)$	$: \{\neg Q(x)\}$
$\neg R(x)$	$: \{\neg P(x) \neg Q(x)\}$

gli altri letterali non generano residui.
Come si è visto prima, la query viene trasformata in $Q(x) \wedge (R(x) \vee \neg P(x)) \wedge P(x)$.
Applicando di nuovo T ai residui aggiunti si ha:

$$\begin{aligned} T^2(Q(x)) &= Q(x) \wedge (T(R(x)) \vee T(\neg P(x))) \wedge T(P(x)) \\ &= Q(x) \wedge (R(x) \vee (\neg P(x) \wedge \neg Q(x))) \wedge P(x) \wedge (R(x) \vee \neg Q(x)) \end{aligned}$$

e di nuovo:

$$T^3(Q(x)) = Q(x) \wedge (R(x) \vee (\neg P(x) \wedge T(\neg Q(x)))) \wedge P(x) \wedge (T(R(x)) \vee \neg T(Q(x)))$$

Dal momento che $T(\neg Q(x)) = \neg Q(x)$ e $T(R(x)) = R(x)$ si ottiene $T^3(Q(x)) = T^2(Q(x))$ e quindi è stata raggiunta la stabilità.
Mentre la correttezza di tale approccio è, nella pratica, sempre verificata, così non è per la completezza e per la terminazione.
Nel testo originale viene indicato come raggiungere risposte consistenti anche nel caso di funzioni di aggregazione.
Quindi questo approccio tramite le riparazioni fornisce solo le istanze consistenti, scartando le altre; il problema della risoluzione dei conflitti in questo modo non viene preso in considerazione. L'approccio di base di questo metodo si adatta perfettamente ad un ambito come il nostro, dove non è possibile modificare i dati locali e l'unico grado di libertà è lasciato alla formulazione della query.

Altre tecniche promettenti sono basate sull'utilizzo di una *funzione di risoluzione*.

2.3 La funzione di risoluzione

Questo tipo di funzione, come si può intuire dal nome che porta, ha il compito di risolvere gli eventuali conflitti di tipo semantico che si dovessero verificare durante un join fra due relazioni. Tra i molti studi sull'argomento, i più completi sono quelli svolti da Naumann ([3], [4]).
Si potrebbe pensare di definire la funzione in modo tale da prendere in ingresso le relazioni per intero e risolvere tutti i conflitti in un colpo solo. Purtroppo questa soluzione non è realizzabile, come risulterà chiaro in seguito.
Nella seguente definizione si assuma $\perp$ come simbolo di valore nullo.

Definizione 2.3.1 (Funzione di risoluzione) *Siano $R_1, R_2, \ldots, R_n$ un insieme di relazioni.*
Siano $X_1, X_2, \ldots, X_n$ i valori assunti dalle relazioni $R_1, R_2, \ldots, R_n$ per l'attributo A di una data tupla.
Sia D il dominio di A e sia $D^+ = D \cup \perp$
Allora si definisce funzione di risoluzione $f_{rA} = f_{rA}(X_1, X_2, \ldots, X_n) : {D^+}^n \rightarrow D^+$

$$f_{rA} = \begin{cases} \perp & \textit{se } \forall i X_i = \perp \\ X_i & \textit{se } \exists i \mid X_i \neq \perp \wedge \forall j \neq i (X_j = \perp \vee X_j = X_i) \\ g_A(X_1, X_2, \ldots, X_n) & \textit{altrimenti} \end{cases}$$

Dove $g_A(X_1, X_2, \ldots, X_n) : {D^+}^n \rightarrow D^+$

Quindi, in altre parole:

- se tutti i valori passati in ingresso sono nulli, allora in uscita si avrà un valore nullo.
- la seconda condizione viene eseguita in due casi:
  - fra i valori passati in input, uno solo è non nullo, quindi questo viene passato in output.
  - più di un parametro in ingresso è non nullo, ma tutti presentano lo stesso valore (quindi non si presenta una situazione conflittuale). Il valore comune viene restituito.
- la terza condizione viene eseguita quando le variabili in ingresso presentano due o più valori differenti. In questo caso bisogna ricorrere ad una funzione per risolvere il conflitto.

La funzione g_A è una generica funzione atta ad eliminare i conflitti tra i valori di un attributo (A, in questo caso).

Guardando alla funzione introdotta con la definizione 2.3.1 si può notare subito che il nodo del problema consiste nel definire opportunamente la funzione $g_A(X_1, X_2, \ldots, X_n)$ in quanto è proprio questa che risolve i conflitti.
La prima cosa da fare è rendere quindi la definizione della nostra funzione di risoluzione il più generale possibile. La funzione precedentemente introdotta infatti potrebbe non essere in grado di risolvere alcuni problemi in maniera adeguata.
Ammettiamo ad esempio di essere in una applicazione di borsa e di avere un conflitto sull'attributo che dà informazioni sul valore della quota delle azioni. Se tutte le fonti i cui dati generano conflitto mantengono anche un attributo che fornisce il volume degli scambi, allora una soluzione consiste nel prendere la quota per cui il valore del corrispondente volume sia maggiore degli altri (e quindi più recente). Una soluzione di questo tipo non è contemplata dalla nostra funzione, in quanto in ingresso presenta solamente il valore dei domini degli attributi conflittuali. Per questo è bene definire un'ulteriore funzione di risoluzione *generalizzata*.

Definizione 2.3.2 (Funzione di risoluzione generalizzata) *Siano $R_1, R_2, \ldots, R_n$ un insieme di relazioni. Siano $X_1, X_2, \ldots, X_n$ i valori assunti dalle relazioni $R_1, R_2, \ldots, R_n$ per l'attributo A. Sia D il dominio di A e sia $D^+ = D \cup \perp$. Sia $E^+ = \times_i E_i^+$ il prodotto cartesiano dei domini di altri attributi di dominio E_i e siano $Y_1, Y_2, \ldots, Y_m$ i valori di questi attributi. Allora si definisce* funzione di risoluzione $f_{rA} = f_{rA}(X_1, X_2, \ldots, X_n, Y_1, Y_2, \ldots, Y_m) : (D^+ \times E^+)^n \rightarrow D^+$

$$f_{rA} = \begin{cases} \perp & \textit{se } \forall i X_i = \perp \\ X_i & \textit{se } \exists i \mid X_i \neq \perp \wedge \forall j \neq i (X_j = \perp \vee X_j = X_i) \\ g_A(X_1, X_2, \ldots, X_n, Y_1, Y_2, \ldots, Y_m) & \textit{altrimenti} \end{cases}$$

Where $g_A(X_1, X_2, \ldots, X_n, Y_1, Y_2, \ldots, Y_m) : (D^+ \times E^+)^n \rightarrow D^+$

Utilizzando questa funzione è possibile ricorrere anche al valore di altri attributi per risolvere eventuali conflitti.
Vediamo ora quali sono le funzioni di risoluzione più utilizzate e diffuse ([4]), divise in tre tabelle. Le funzioni mostrate nella tabella 2.1 sono utilizzabili per attributi di ogni tipo, quelle nella 2.2 solo da attributi di tipo numerico mentre quelle nella 2.3 solo da attributi di tipo non numerico. Per ogni funzione esposta viene mostrato anche un esempio di applicazione.

nome	spiegazione	esempio di utilizzo
COUNT	Numero di valori non-nulli; ossia il numero di valori conflittuali. Il vero valore dei dati va perso.	Il valore può venire immagazzinato in un attributo special-purpose.
MIN	Minimo valore in ingresso. Per i dati numerici è il valore più piccolo, per i dati non numerici è il valore inferiore dal punto di vista lessicografico.	attributi come "record del mondo" o "prezzo".
MAX	Massimo valore in ingresso. Per i dati numerici è il valore più grande, per i dati non numerici è il valore superiore dal punto di vista lessicografico.	attributi come "età" o "volume degli scambi".
RANDOM	Valore in ingresso casuale non nullo.	Può essere utilizzato quando non ha importanza da quale fonte viene scelto il valore.
CHOOSE(*fonte*)	Sceglie il valore fornito dalla fonte *fonte*.	Viene usato quando si è a conoscenza che solo una fonte è in grado di fornire informazioni corrette ed affidabili su un certo attributo.
FAVOR($E(O)$)	Sceglie il primo valore non nullo contenuto nell'ordinamento totale di tutte le sorgenti $E(O)$.	Simile alla funzione precedente, può essere utilizzata se è possibile ordinare le fonti secondo un qualche criterio in base alle informazioni fornite per un attributo.
MAXIQ	Valore di qualità più alta. Necessita di un modello per valutare la qualità delle informazioni.	A differenza delle altre funzioni, oltre che ad un singolo attributo può essere applicato anche ad una intera fonte.
GROUP	Insieme di tutti i valori conflittuali. Questa funzione delega all'utente il compito di decidere quale valore sia il più appropriato.	È utilizzabile solamente nel caso in cui l'utente conosca bene il dominio dell'attributo.

Tabella 2.1: Funzioni di risoluzione per ogni tipo di attributo

nome	spiegazione	esempio di utilizzo
SUM	Somma di tutti i valori in ingresso.	Può essere utili per attributi come “entrate”.
MEDIAN	Valore mediano, ossia che ha un ugual numero di valori in ingresso minori e maggiori	Utilizzabile per attributi come “portata” o “altezza”
AVG	Media dei valori in ingresso.	Utilizzabile per rilevazioni statistiche.
VAR	Varianza dei valori in ingresso. È una funzione anomala in quanto ha un significato differente rispetto ai valori in ingresso.	Di solito viene immagazzinata in un attributo addizionale.
STDDEV	Deviazione standard dei valori in ingresso. Valgono le stesse osservazioni fatte per la funzione precedente.	Come la varianza, viene tenuta in un attributo addizionale.

Tabella 2.2: Funzioni di risoluzione per attributi numerici

nome	spiegazione	esempio di utilizzo
SHORTEST	Il valore in ingresso con il minor numero di caratteri, senza considerare gli spazi.	Attributi come “sommario”
LONGEST	Il valore in ingresso con il maggior numero di caratteri, senza considerare gli spazi.	Attributi come “titolo”
CONCAT	Concatenazione di tutti i valori in ingresso. Dal momento che la dimensione del risultato aumenta con l’aumentare delle fonti, è necessario prendere precauzioni per assicurarsi che l’output rimanga nei limiti di memoria del tipo dell’attributo.	Attributi come “descrizione”
ANNCONCAT	concatenazione con annotazione dei valori in ingresso. Si comporta come la funzione precedente, ma in più, insieme ad ogni valore è specificata la sorgente da cui è estratto.	Vale quanto detto nel caso precedente.

Tabella 2.3: Funzioni di risoluzione per attributi non numerici

Queste sono solo alcune delle funzioni proposte, ma comunque sono sufficienti per la maggior parte degli attributi usati più di frequente.
Se si volessero comunque prendere in considerazione altre funzioni non elencate nelle tabelle, si tenga presente che, per essere poi applicabili, queste funzioni devono soddisfare la proprietà *commutativa*. Questo perché le relazioni in SQL non sono ordinate.

A questo punto risulta chiaro il perché della scelta di definire la funzione di risoluzione tale da prevedere in ingresso solamente i valori di un attributo per volta, piuttosto che l'intera relazione.
Supponiamo di avere una relazione con i seguenti attributi: "prezzo" e "giacenza". Presentando all'ingresso di una ipotetica funzione di risoluzione la relazione per intero si dovrebbe, già con due soli attributi, scendere a compromessi: quale funzione di risoluzione utilizzare? Se si utilizzasse la funzione MIN, verrebbe restituito il minimo prezzo, ma anche la minima disponibilità, mentre effetto contrario ma ugualmente inadeguato avrebbe l'applicazione della funzione MAX.
Si vede subito come un'unica funzione di risoluzione sia inadeguata anche per un numero minimo di attributi.
La funzione di risoluzione introdotta non presenta invece problemi di questo tipo, poiché se ne può definire una per ogni attributo.

La sezione 2.5 parlerà in maniera più approfondita del complicato problema dell'identificazione degli oggetti.

2.4 La qualità nella selezione delle fonti

Se il numero di fonti a disposizione è troppo elevato, come nel caso di Internet, non è conveniente interrogarle tutte e nella quasi totalità dei casi ciò non risulta nemmeno necessario per ottenere la completezza della risposta.
Data l'importanza della qualità, in ambito di ricerca sono disponibili alcuni algoritmi che realizzano la selezione fra più fonti basandosi esclusivamente su questa caratteristica. Se ne ricordano in particolare quattro: il metodo Simple Additive Weighting (SAW), il metodo Tecnique for Order Preference by Similarity to Ideal Solution (TOPSIS), il metodo Analytical Hierarchy Process (AHP) ed il metodo Data Envelopment Analysis (DEA).
Prima di tutto è necessario introdurre un vettore W dei pesi. Questo vettore ha tanti valori quante sono le caratteristiche della qualità che si vogliono tenere in considerazione, fra quelle elencate in sezione 2.1. Solitamente, dato $W = (w_1, \ldots, w_n)$ si ha che $\sum_{i=1}^{n} w_i = 1$, ma ciò non è strettamente necessario. Si assume l'esi-

stenza di una *funzione di costo lineare*, ossia: più un criterio di qualità ottiene un punteggio elevato (o più un criterio di costo ottiene un punteggio basso), più questo è gradito all'utente. Inoltre è importante che i criteri siano *indipendenti dalla preferenza*. Con questo termine si indicano quei criteri che, a seconda del punteggio ottenuto, possono essere univocamente considerati positivi o negativi, indipendentemente dall'utente e dal contesto. Ad esempio, il criterio "dimensione della risposta" non rientra in questa categoria.
Per comodità si assume di dividere le caratteristiche della qualità in due grandi insiemi: *qualità* e *costi*.
I valori calcolati o determinati per ogni sorgente e per ogni criterio possono essere mantenuti in una matrice, dove ogni riga corrisponde ad una sorgente ed ogni colonna ad un criterio. Nel seguito si suppone di avere $j = 1, \ldots, n$ criteri e $i = 1, \ldots, m$ fonti.

Simple Additive Weighting:
Questo metodo è molto semplice e veloce, ma nonostante questo ha dei risultati molto vicini a quelli dei metodi più sofisticati. Il SAW è formato da tre passi: per prima cosa si scalano i valori per renderli comparabili, poi si applicano i pesi ed infine si sommano i valori per ogni fonte.
Per realizzare il primo passo si utilizzano due fattori di scala:

$$v_{ij} = \frac{d_{ij} - d_j^{min}}{d_j^{max} - d_j^{min}} \qquad per\ i\ criteri\ di\ qualità$$
$$v_{ij} = \frac{d_j^{max} - d_{ij}}{d_j^{max} - d_j^{min}} \qquad per\ i\ criteri\ di\ costo$$

In questo modo tutti i punteggi dei valori sono nell'intervallo $[0, 1]$; il punteggio migliore è 1 ed il peggiore è 0. Così tutti i criteri possono essere confrontati, indipendentemente da come sono memorizzati alle fonti. Viene calcolato infine il punteggio di ogni fonte con:

$$S_i = \sum_{i=1}^{n} w_j v_{ij}$$

Tecnique for Order Preference by Similarity to Ideal Solution:
Come nel metodo precedente, i punteggi vengono scalati e pesati. Il fattore di scala da utilizzare è

$$v_{ij} = \frac{d_{ij} * w_j}{\left[\sum_{i=1}^{n} d_{ij}^2\right]^{1/2}}$$

di modo da normalizzare ogni vettore di criteri. A questo punto, adifferenza del SAW, il TOPSIS non somma i valori, ma calcola la distanza relativa di ogni fonte da una fonte ideale fittizia. La fonte più vicina all'idealità positiva e più lontana dall'idealità negativa viene poi scelta come la migliore. Le due fonti ideali vengono così calcolate:

$$A^* = (v_1^*, \ldots, v_n^*) := \{(\max_{i=1,\ldots,m} v_{ij} \mid j \in J_Q), (\min_{i=1,\ldots,m} v_{ij} \mid j \in J_C)\}$$
$$A^- = (v_1^-, \ldots, v_n^-) := \{(\min_{i=1,\ldots,m} v_{ij} \mid j \in J_Q), (\max_{i=1,\ldots,m} v_{ij} \mid j \in J_C)\}$$

Dove J_Q identifica l'insieme dei criteri di qualità, J_C l'insieme dei criteri di costo e A^*, A^- rispettivamente la fonte ideale positiva e negativa. Dal calcolo emerge come il calcolo delle fonti fittizie sia relativo alle fonti in gioco e non un arbitrario valore assoluto. In particolare, la fonte ideale positiva viene calcolata come quella per cui i valori dei criteri di costo e di qualità sono rispettivamente i minori e maggiori possibili, in base alle fonti coinvolte nel calcolo. Discorso speculare vale per la fonte ideale positiva.
La distanza euclidea fra ogni fonte e le due fonti fittizie viene così calcolata:

$$S^{(*|-)}(S_i) := \sqrt{\sum_{j=1}^{n} (v_{ij} - v_j^{(*|-)})^2}$$

La vicinanza relativa delle fonti dall'idealità si calcola invece con:

$$C^*(S_i) := \frac{S^-(S_i)}{S^*(S_i) - S^-(S_i)}$$

Analytical Hierarchy Process
Questo metodo è composto da quattro passi principali: sviluppo di una gerarchia degli obiettivi, confronto paritario degli obiettivi, controllo di consistenza dei confronti, aggregazione dei confronti.
La gerarchia degli obiettivi per selezionare le fonti è rappresentata in figura senza perdita di generalità si suppone che $Q_j = \{j \mid j = 1, \ldots, j = k\}$ e che $C_j = \{j \mid j = k+1, \ldots, j = n\}$.

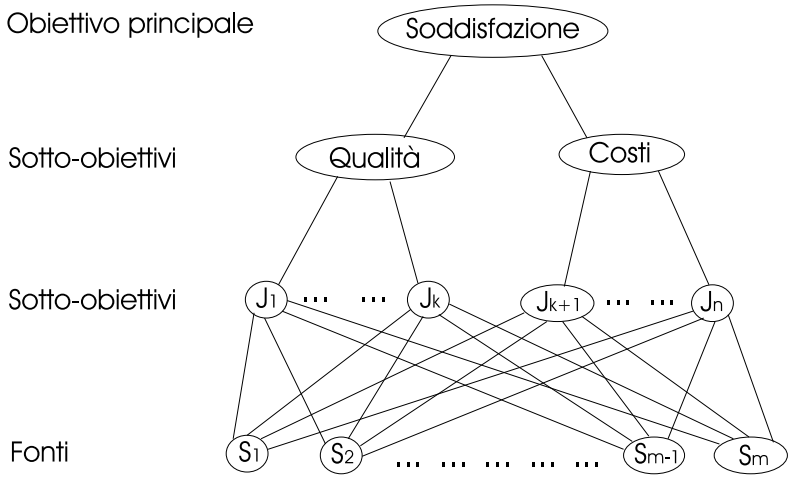


Figura 2.1: Gerarchia degli obiettivi

L'obiettivo principale consiste nella *soddisfazione* dell'utente, che si divide in obiettivi di *costo* e di *qualità*. entrambi si dividono poi in vari criteri, ognuno dei quali riflesso su tutte le fonti.
Per ottenere un confronto paritario fra gli obiettivi, si definiscono delle matrici: una per ogni obiettivo e sotto-obiettivo della gerarchia.
I valori mantenuti nelle matrici relative a *soddisfazione*, *costo* e *qualità* riflettono i pesi, mentre quelli nelle matrici dei criteri rispecchiano i punteggi. I valori vanno da $\frac{1}{9}$ (che significa "molto meno importante") a 9 (molto più importante); il valore 1 sta per "della stessa importanza".
Per esempio, se avessimo tre criteri di qualità j_1, j_2, j_3, una possibile matrice di *qualità* potrebbe essere:

	j_1	j_2	j_3
j_1	1	$\frac{1}{2}$	3
j_2	2	1	6
j_3	$\frac{1}{3}$	$\frac{1}{6}$	1

in cui si afferma che j_1 è poco meno importante rispetto a j_2 ma un po' più importante rispetto a j_3; j_2 è inoltre molto più importante rispetto a j_3. Ovviamente i criteri hanno stessa importanza rispetto a se stessi.
Discorso equivalente vale per le matrici dei singoli criteri; ammettendo di avere tre sorgenti S_1, S_2, S_3, la matrice relativa a *affidabilità* potrebbe essere:

	S_1	S_2	S_3
S_1	1	$\frac{1}{9}$	2
S_2	9	1	$\frac{1}{4}$
S_3	$\frac{1}{2}$	4	1

nella quale si afferma, ad esempio, che la fonte S_1 è ritenuta molto meno affidabile rispetto a S_2 e poco più affidabile rispetto a S_3.
A questo punto bisogna eseguire il controllo di consistenza dei confronti.
Questo passo fornisce aiuti nel trovare confronti transitivamente errati. Ad esempio, guardando la prima matrice si può vedere come si possa ricavare un ordinamento in base all'importanza attribuita: j_2 è più importante di j_2 che è più importante di j_3.
La seconda tabella invece presenta delle inconsistenze, in quanto si presentano dei valori incongruenti; infatti, si evince che S_1 è più affidabile di S_3, che è più affidabile di S_2, che è più affidabile di S_1. Ciò è chiaramente un errore.
Per compiere l'ultimo passo si calcolano dei vettori di pesi: uno per ogni matrice. Ognuna si calcola come l'autovettore normalizzato del massimo autovalore della matrice. Ogni arco della gerarchia è poi etichettato con il valore determinato dal vettore dei pesi. Il punteggio di preferenza finale di ogni sorgente è calcolato come la somma pesata lungo il percorso fra la sorgente e l'obiettivo principale.

Data Envelopment Analysis
Questo metodo è diverso da quelli finora esposti, in quanto non si preoccupa di stilare una classifica delle fonti, ma piuttosto suggerisce quali fonti siano migliori delle altre. Non è nato espressamente per questo scopo, ma come un metodo generale per classificare una popolazione di osservazioni.
L'efficienza di ogni sorgente S_{i0} separatamente, risolvendo un programma lineare:

PL-DEA:

$$\begin{array}{lrl} \text{massimizza} & \sum_{j\in Q_J} w_j d_{i0j} - \sum_{j\in C_J} w_j d_{i0j} & \\ \text{soggetta a} & \sum_{j\in Q_J} w_j d_{ij} - \sum_{j\in C_J} w_j d_{ij} & \leq 1;\ \forall S_j \\ & \sum_{j\in C_J} w_j d_{i0j} & = 1 \\ & w_j & \geq \varepsilon > 0;\ \forall 1 \leq j \leq n \end{array}$$

un'ulteriore differenza rispetto ai metodi precedenti è rappresentata dai pesi. Ora non sono più specificati dell'utente, bensì variabili determinate dal metodo; per questo motivo i valori non necessitano di operazioni di scalatura. Il risultato del programma lineare può essere 1 (e in questo caso la fonte viene dichiarata efficiente) o compreso fra 0 e 1 (in questo caso il valore identifica il grado di inefficienza della fonte).

Per ulteriori informazioni, in [15] si può trovare un confronto fra le tecniche precedentemente descritte.

Si tenga in considerazione che questa selezione, benché basata sulla qualità, non esclude la presenza di conflitti fra i dati. Per cui è necessario prevedere la

presenza di funzioni di risoluzione (scelte fra quelle di tabelle 2.1, 2.2 e 2.3). Volendo continuare sulla strada della qualità è possibile ricorrere esclusivamente alla funzione MAXIQ; in questo modo tutto il processo, dalla selezione della sorgente alla risoluzione dei conflitti, si basa unicamente sulla qualità.
In linea di principio, è possibile definire i criteri qualitativi anche per quanto riguarda i risultati. Ad esempio, il costo del risultato può essere visto come la somma dei costi delle singole fonti, oppure l'accessibilità come la probabilità che ogni fonte interrogata sia accessibile. Nello stesso modo si può prevedere di mostrare i risultati ordinati in base alla qualità.

2.5 L'identificazione degli oggetti

Nella sezione precedente è stato brevemente introdotto questo argomento, mostrandone le caratteristiche principali e le problematiche più evidenti. Nonostante ciò, non sono ancora chiare le enormi difficoltà che sorgono durante questa elaborata e, nonostante tutti gli sforzi, imprecisa operazione.
Le ricerche riguardanti questo campo e, soprattutto, le proposte di tecniche risolutive non mancano; infatti, nonostante il problema sia relativamente recente, la sua estrema similarità con l'operazione di data cleaning rende possibile guardare in molte direzioni per trovare possibili soluzioni.
Ovviamente, non tutte le tecniche specifiche per il data warehousing sono utilizzabili pedissequamente in questo ambito, ma comunque molte delle idee proposte possono essere applicate con lievi variazioni.

Questo perché, a causa di eventuali errori di battitura, anche un solo campo di una tupla può portare ad errori di vario tipo.
Nel caso in cui l'attributo inserito in maniera scorretta non sia di join, con un po' di fortuna, reiterate applicazioni della funzione di risoluzione possono portare a correggere l'errore o quantomeno ad arginarlo. Ciò ovviamente non è possibile nel caso in cui la fonte da cui proviene l'errore sia l'unica ad essere interrogata o a fornire un valore significativo.
Il problema si complica nel caso in cui l'errore si presenti su un attributo di join. Al momento del join con un'altra relazione infatti, tuple rappresentanti lo stesso oggetto non verrebbero fuse, portando ad un risultato errato.

La principale difficoltà che si incontra, tentando di risolvere questo problema, è quella di stabilire con assoluta certezza ed in modo automatico quando due tuple descrivono lo stesso oggetto reale.
Questo compito, come verrà messo in luce in sezione 2.5.1, è davvero difficile con le tecniche fino ad oggi a disposizione.
Per questo si ammette la possibilità di avere quegli errori che vengono definiti

falsi positivi e *falsi negativi*. In figura 2.2 vengono mostrati i possibili risultati dell'identificazione di un oggetto.

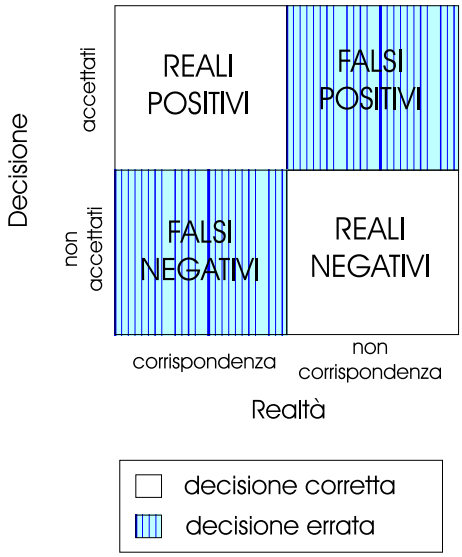


Figura 2.2: Schema dell'identificazione degli oggetti

Per cui:

- *reale positivo* indica che un oggetto è stato ritenuto uguale ad un altro dal sistema ed è così anche nella realtà.
- *falso positivo* indica che un oggetto è stato ritenuto uguale ad un altro dal sistema, ma nella realtà questi oggetti sono differenti.
- *reale negativo* indica che un oggetto è stato ritenuto diverso da un altro dal sistema ed è così anche nella realtà.
- *falso positivo* indica che un oggetto è stato ritenuto diverso da un altro dal sistema, ma nella realtà questi oggetti sono uguali.

A questo punto, per comprendere quali siano le innumerevoli difficoltà che non permettono di ottenere una soluzione perfetta al problema in esame, è utile introdurre alcune tecniche e mostrarne i punti deboli.

2.5.1 Le tecniche per l'identificazione degli oggetti

Prima di introdurre una tecnica vera e propria è bene spendere qualche parola sulla fase che riguarda il confronto fra due tuple, comune a molti metodi che seguono. Per rendere il sistema il grado di riconoscere i duplicati è necessario introdurre una funzione f_d che, date in ingresso due tuple t_1 e t_2, anche con schemi diversi, restituisca in uscita:

0 se t_1 e t_2 non rappresentano lo stesso oggetto.

1 se t_1 e t_2 rappresentano lo stesso oggetto.

-1 se la risposta è indeterminata.

Le funzioni più utilizzate prevedono un confronto sulla base di una qualche distanza. Se la distanza fra le due tuple è superiore ad una certa soglia, allora le due tuple non sono duplicati, se invece la distanza è inferiore ad una certa soglia, sono stati trovati due duplicati. È possibile prevedere un'isteresi fra le due soglie; se la distanza cade nell'isteresi, allora la risposta è indeterminata. Si ricordano soprattutto tre tipi di distanze:

- *edit distance*:la edit distance è definita come il più piccolo numero di inserimenti, cancellazioni o sostituzioni richieste per trasformare una stringa in un'altra. È conosciuta anche come distanza di Levenshtein. Ad esempio, la edit distance fra le stringhe "casa" e "costa" è 2, dato che per la trasformazione sono necessarie una sostituzione (da "a" ad "o") ed un inserimento (la "t").

- *phonetic distance*: due termini sono considerati uguali se hanno un suono simile. Questo metodo è detto anche "soundex", ed è utilizzato anche a fini di censimento. È un sistema di indicizzazione basato sul cognome, dove ogni ingresso viene trasformato in un'uscita formata da quattro simboli: un carattere alfabetico (l'iniziale) seguito da tre caratteri numerici.
  Data una stringa, ogni carattere viene mappato secondo la tabella seguente; vengono considerati solo i caratteri necessari a formare la stringa, quelli in eccesso si scartano. Se una stringa è troppo corta per formare la chiave, allora questa viene completata con degli zeri.

Numero	Lettera
1	B, F, P, V
2	C, G, J, K, Q, S, X, Z
3	D, T
4	M, N
6	R

i caratteri A, E, I, O, U, H, W, Y non vengono considerati. Perciò, ad esempio, Lee darebbe L-000, mentre Bianchini B-525. Esistono inoltre regole supplementari per particolari sequenze di caratteri:

  - quando una stringa presenta delle lettere doppie, solo la prima viene considerata; ad esempio: Mattioli dà M-340.
  - se in una stringa si presentano più lettere vicine che generano lo stesso codice, queste vengono considerate come un unico carattere. Ad esempio la chiave per Marsciano sarebbe M-625.
  - i cognomi con prefisso vengono memorizzati sia con il prefisso che senza. Le stringhe Mc e Mac non sono considerate prefissi.
  - se una vocale separa due lettere che producono lo stesso codice, entrambe vengono considerate. Ad esempio, Venditti darebbe V-533.

  Comunque si tenga presente che tale metodo è utilizzato per il censimento negli Stati Uniti; per questo alcune delle regole potrebbero risultare inopportune o inesatte quando applicate a lingue diverse dall'inglese.

- *typewriter distance*: nata soprattutto per venire incontro agli errori di battitura, il confronto avviene considerando vicini due caratteri se questi lo sono anche sulla tastiera di un computer. Ad esempio "Marco" è considerato più simile a "Marzo" che non a "Marmo", in quanto nelle normali tastiere qwerty la "c" è più vicina alla"z" rispetto alla "m".

Negli ultimi anni alcuni studi hanno messo in luce i difetti di un riconoscimento basato sulla distanza; ad esempio, il valore della soglia da utilizzare non è un valore semplice da determinare, in quanto varia a seconda del contesto e dei dati immessi. Per questo sono stati proposti metodi alternativi, che però mostrano la propria efficacia solamente nel contesto dello studio all'interno del quale vengono presentati; per questo si è ritenuto più opportuno rimandarne l'introduzione alla sezione 2.5.2.

2.5.2 Tecniche risolutive

Una tecnica storica, ritenuta abbastanza valida in passato, prende il nome di *sorted neighborhood method*. Questa tecnica, molto semplice nell'idea di fondo, si divide in quattro passi. Per comodità ammettiamo di avere una sola relazione:

1) si ordina la relazione secondo l'attributo (o gli attributi) ritenuto più discriminante.

2) si determina una dimensione w per una finestra e si confrontano le prime w tuple fra di loro, cercando eventuali duplicati.

3) si fa scorrere la finestra facendo entrare la tupla $w + 1$. Si confronta questa tupla con le altre nella finestra, cercando dei duplicati.

4) si reitera il procedimento fino al termine della relazione.

In questo modo, gli eventuali errori di battitura dovrebbero venire corretti.
Comunque resta il problema di trovare un valore per w tale da evitare falsi negativi (quindi non troppo basso) e tempi di esecuzione troppo lunghi (quindi non troppo alto). Questo aspetto si può risolvere con l'esperienza, ma i veri problemi sono altri. Ammettiamo di avere la seguente relazione:

carta identità	*nome e cognome*
BS2365674	Luigi Bianchi
AZ2693822	Mario Rossi
ZA2693822	Mario Rossi
⋮	⋮

si nota la presenza di un campo che esprime il numero di carta d'identità; questo tipo di campo, come molti altri, nonostante abbia una capacità discriminatoria molto elevata, è puramente simbolico e quindi è facile commettere un errore senza accorgersene. Ad esempio, in questo caso il valore AZ2693822 è diventato ZA2693822 solamente invertendo i primi due valori. A questo punto procediamo con l'ordinamento, che verrà effettuato con ogni probabilità secondo l'attributo *carrta d'identità*.

carta identità	*nome e cognome*
AZ2693822	Mario Rossi
BS2365674	Luigi Bianchi
⋮	⋮
ZA2693822	Mario Rossi
⋮	⋮

In questo modo, per ampia che sia la finestra, i due record duplicati non verranno mai confrontati.

Un metodo relativamente recente, presentato in [16] ed in [17] propone di modificare il metodo sorted neighborhood svolgendo diverse esecuzioni in parallelo su diversi attributi per poi realizzare la chiusura transitiva dei risultati ottenuti. Per *chiusura transitiva* si intende genericamente un'estensione di una relazione

binaria tale che ogni volta che (a, b) e (b, c) sono nell'estensione, allora nell'estensione è presente anche (a, c). L'efficacia di questo metodo si basa sulla bassa probabilità di commettere più errori nella stessa tupla.

Ammettiamo di avere la seguente relazione:

nome	*cognome*	*città*	*indirizzo*	*telefono*
Mario	Rossi	Bologna	via Indipendenza 27	123456
Andrea	Bianchi	Modena	via Campi 27	463523
Marco	Rossi	Bologna	via Indipendenza 27	123456

Ipotizziamo di voler controllare i valori di tutti gli attributi per cercare eventuali tuple duplicate. Si noti come la prima e la terza tupla siano identiche a meno di un carattere nell'attributo *nome*. Perciò, a prima vista, si direbbe che queste due tuple in realtà fanno riferimento allo stesso oggetto reale. Ma se invece le due tuple fossero inserite correttamente e Marco Rossi e Mario Rossi fossero padre e figlio? È quindi palese la necessità di informazioni aggiuntive o la presenza di un operatore umano che possa sciogliere questi dubbi.

Esiste anche un altro aspetto che può portare molti problemi, soprattutto per quanto riguarda i metodi che fanno affidamento ad una funzione basata sulla distanza: le sigle e le abbreviazioni.
Una funzione basata sulla distanza potrebbe individuare che "ONU" ed "Organizzazione delle Nazioni Unite" rappresentano lo stesso oggetto solo a patto di abbassare la soglia a livelli tali da introdurre un numero altissimo di falsi positivi.

Altri metodi propongono di abbandonare l'equijoin in favore di altri operatori. Ad esempio il *band join*, utilizzato in [18], [16] e [17].
Ammettiamo di avere due relazioni: L_1 ed L_2, e di voler fare il join fra gli attributi $L_1.A$ ed $L_2.B$. La condizione di join del band join può essere espressa come

$$L_1.A - c_1 \leq L_2.B \leq L_1.A + c_2 \tag{2.2}$$

Dove le costanti c_1 e c_2 possono assumere qualsiasi valore maggiore o uguale a zero e possono anche essere diverse fra loro.
In altre parole, il join avviene fra le tuple t_1 di L_1 e t_2 di L_2 tali che il valore di $t_1.A$ sia compreso in una banda di dimensione $c_1 + c_2$ attorno al valore di $t_2.B$.
Utilizzando il band join ci sono più possibilità di trovare corrispondenze che altrimenti non si potrebbero trovare, anche se una banda troppo larga può portare ad un incremento nel numero dei falsi positivi.
In [16] la definizione di band join è più generale, poiché ammette condizioni di join θ generiche, più complesse di quella in equazione (2.2).

Poiché il problema maggiore in questo campo è la mancanza di un identificatore comune, alcuni studi tentano di costruirne uno in base al valore degli attributi delle tuple. In [19], in particolare, viene creato un "token" per ogni tupla, da considerarsi poi come identificatore.
Inizialmente una persona con conoscenza del sistema sceglierà due o tre attributi ritenuti maggiormente discriminanti e li ordinerà per importanza.
Viene creata una nuova relazione, ottenuta proiettando quella originaria sugli attributi scelti.
Gli attributi scelti vengono precedentemente sottoposti all'eliminazione dei caratteri ininfluenti (ad esempio "Dr." nel campo del nome o "(", ")", "-" nel numero di telefono) e di parole presenti in una stop list (nel caso di campi contenenti un testo per esteso). I campi così ripuliti, a seconda che siano numerici, alfanumerici o alfabetici, danno origine a token diversi.

- **campo numerico:** è ottenuto da quegli attributi prevalentemente numerici, come il numero di telefono o la data di nascita

- **campo alfabetico:** è ottenuto da attributi contenenti, ad esempio, nomi. Il valore del campo viene sottoposto ad una funzione che restituisce solo le iniziali in ordine alfabetico. Ad esempio, per stringhe come "Mario Antonio Rossi", "Rossi Mario Antonio" e "M. A. Rossi" il risultato sarebbe sempre "AMR".
  Si noti che con questo metodo si risolve anche il problema, precedentemente esposto, causato dalle sigle.

- **campo alfanumerico:** è ottenuto da campi che hanno sia caratteri che numeri, come un indirizzo. Il valore del campo viene sottoposto ad una funzione che:

  1. decompone l'elemento nei suoi membri costituenti
  2. seleziona solo gli elementi che sono numerici o alfanumerici
  3. decompone ogni parte alfanumerica nella rispettiva parte numerica ed alfabetica
  4. ordina secondo un metodo arbitrario le parti così ottenute

  Ad esempio, dato il valore "Via ABC6D3 n 245" viene restituito "24563ABCD", assumendo di ordinare le parti in modo crescente. Infatti il secondo passo seleziona "ABC6D3 245" e dopo il terzo si ha "ABCD 63 245".

A questo punto si ordina la nuova relazione secondo ognuno dei campi scelti, separatamente; in questo modo si aumentano le probabilità che tuple di token rappresentanti lo stesso oggetto siano vicine.

Si passa poi alla ricerca dei duplicati. Date due tuple t_1 e t_2 di token, per prima cosa si divide il numero di token di ugual valore nelle due tuple per il numero totale di token delle due tuple. Il risultato così ottenuto è detto *similarity match count* (SMC). Questo valore è compreso fra 0 e 1; in particolare, se è maggiore di 0.67 le tuple vengono considerate corrispondenti; risultato opposto si ha se SMC è minore di 0.33. Se due tuple hanno invece un SMC compreso fra (0.33 e 0.66) è necessaria un'altra funzione, detta *similarity match ratio* (SMR), per stabilire se corrispondono. L'SMR, per ogni coppia di token che non corrispondono esattamente, ne confronta il valore carattere per carattere. Supponendo che i due token abbiano m ed n caratteri, dei quali c in comune. Allora: SMR $= \frac{2c}{m+n}$. I due token vengono considerati corrispondenti solo se SMR ≥ 0.67. Viene poi effettuato nuovamente il calcolo di SMC, il cui risultato varia in base alla maggiore conoscenza acquisita sulla corrispondenza dei token tramite SMR. Dopo aver applicato questo procedimento per tutti gli ordinamenti, viene effettuata l'unione dei risultati.

Un'ulteriore tecnica, introdotta in [20] si basa su ci che viene definita *chiave estesa*; in parole povere, per identificare un oggetto viene utilizzata una chiave composta da attributi, anche non di join, che provengono da diverse fonti. Questi attributi non devono necessariamente apparire in diversi schemi. Più formalmente:

Definizione 2.5.1 (Chiave estesa) *Siano $R_1, \ldots, R_n$ relazioni appartenenti a database distinti modellanti un insieme E di entità del mondo reale. e siano $K_1, \ldots, K_n$ le rispettive chiavi delle relazioni. Per chiave estesa,indicata con K_{ext} si intende l'insieme minimo di attributi, nella forma $K_1 \cup \cdots \cup K_n \cup \tilde{A}$, necessario per identificare univocamente ogni entità di E. Per $\widetilde{A}$ si intende un insieme di attributi $\tilde{a}$ tale che: $\widetilde{A} = \{\tilde{a} \mid \tilde{a} \in \bigcup_{i=1}^{n} Sch(R_i), \tilde{a} \notin \bigcup_{i=1}^{n} K_i\}$.*

Intuitivamente si può capire come questa nuova chiave risulti utile soprattutto nei casi in cui non vi sia un identificatore comune; in caso contrario si avrebbe infatti che $K_{ext} = K_1 = \cdots = K_n$. L'introduzione di $\widetilde{A}$ è necessaria per dare un carattere il più possibile generale alla definizione; comunque, nella pratica spesso si ha che $K_{ext} = K_1 \cup \cdots \cup K_n$.
La chiave estesa viene utilizzata in [20] per realizzare un approccio che, tramite l'utilizzo di regole, cerca di risolvere il problema dell'identificazione degli oggetti. Per la precisione, si parla di *regole di identità*.

Definizione 2.5.2 *Una regola di identità per l'insieme E di entità del mondo reale si presenta nella forma: $\forall e_1, e_2 \in E, P(e_1.A_1, \ldots, e_1.A_m, e_2.B_1, \ldots, e_2.B_n) \rightarrow (e_1 \equiv e_2)$ dove P è una congiunzione di predicati sugli attributi $A_1, \ldots, A_m$ e $B_1, \ldots, B_n$ rispettivamente di e_1 e e_2. Ogni predicato è nella forma $e_1.attributo \;\; op \;\; e_2.attributo$ o $e_i.attributo \;\; op \;\; valore$, dove $op \in \{=, <, >, \leq, \geq, \neq\}$ e $i = 1, 2$. Inoltre, per ogni $e_1.A_i$ o $e_2.A_i$ che appare nei predicati, P deve implicare $e_1.A_i = e_2.A_i$.*

Per esempio, se stessimo modellando dei fumetti, la regola:
$r_1: \ \forall e_1, e_2 \in E, (e_1.titolo = Kappa \ \ Magazine) \wedge (e_2.titolo = Kappa \ \ Magazine) \rightarrow (e_1 \equiv e_2)$ sarebbe corretta, invece:
$r_2: \ \forall e_1, e_2 \in E, (e_1.titolo = Kappa \ \ Magazine) \rightarrow (e_1 \equiv e_2)$ non lo sarebbe, in quanto la parte a sinistra non implica $e_1.titolo = e_2.titolo$.
In particolare si può vedere che con: $\forall e_1, e_2 \in E, (e_1.A_k = e_2.A_k) \rightarrow (e_1 \equiv e_2)$ supponendo che e_1 ed e_2 siano modellate in R_1 ed R_2 rispettivamente e che A_k sia la chiave utilizzata in entrambe le relazioni; allora con tale regola si ottiene la tecnica classica che identifica un oggetto tramite l'equivalenza delle chiavi.
A questo punto risulta chiaro come integrare la chiave estesa con regole come quelle appena introdotte: $\forall e_1, e_2 \in E, (e_1.A_1 = e_2.A_2) \wedge \cdots \wedge (e_1.A_k = e_2.A_k) \rightarrow (e_1 \equiv e_2)$, dove $K_{ext} = \{A_1, \ldots, A_k\}$. Comunque, data la differenza negli attributi che compongono gli schemi delle relazioni, alcune regole non possono essere applicate propriamente; in alcuni casi è però possibile "estendere" questi schemi, utilizzando i risultati apportati da un ulteriore tipo di regole, dette *dipendenze funzionali a livello di istanza* (ILFD).

Definizione 2.5.3 (ILFD) *Una ILFD è un vincolo semantico definito sulle entità del mondo reale. Si presenta nella forma:* $\forall e \in E, (e.A_1 = a_1) \wedge \cdots \wedge (e.A_n = a_n) \rightarrow (e.B = b)$*, dove* $A_1, \ldots, A_n$ *e* B *sono attributi e* $a_1, \ldots, a_n$ *e* b *sono i rispettivi possibili valori.*

Ad esempio, avendo le due seguenti relazioni

R_1

titolo pubblicazione	*numero pubblicazione*	*titolo fumetto*
Kappa Magazine	34	Kamikaze
Point Break	25	Sky High

R_2

titolo fumetto	*numero fumetto*
Kamikaze	12
Sky High	2

ed ammettendo che la chiave estesa sia $K_{ext} = \{$*titolo pubblicazione, titolo fumetto*$\}$, da cui la regola: $\forall e_1, e_2 \in E, (e_1.$*titolo pubblicazione*$= e_2.$*titolo pubblicazione*$) \wedge (e_1.$*titolo fumetto*$= e_2.$*titolo fumetto*$) \rightarrow (e_1 \equiv e_2)$, si vede come questa non sia direttamente applicabile. Se però esistono ILFD come:
$\forall e, (e.$*titolo fumetto*=Kamikaze$) \rightarrow (e.$*titolo pubblicazione*=Kappa Magazine$)$
e
$\forall e, (e.$*titolo fumetto*=Sky High$) \rightarrow (e.$*titolo pubblicazione*=Point Break$)$,
allora la relazione R_2 può essere così estesa:

R_2

titolo pubblicazione	*titolo fumetto*	*numero fumetto*
Kappa Magazine	Kamikaze	12
Point Break	Sky High	2

In questo modo, la regola può essere applicata correttamente e le coppie di oggetti possono essere identificate.

A fianco a quelle di identità sono definite anche le *regole di distinzione*, di cui si omette la definizione poiché analoga alla precedente; l'unica differenza risiede nella parte destra della regola, che in questo caso diventa $(e_1 \not\equiv e_2)$. Quindi sono definite anche regole per stabilire quando due oggetti sono diversi; in questo modo si creano, ma solo idealmente, due grandi gruppi: quello delle coppie di oggetti identici e quello delle coppie di oggetti diversi. In realtà però questo risultato è ben difficile da ottenere, in quanto le regole da produrre sarebbero un numero troppo elevato. Per questo si ammette l'esistenza di un terzo gruppo, quello delle coppie di oggetti indeterminate. Ovviamente, più questa categoria è limitata, meno decisioni sono relegate all'utente, meglio è. In linea di principio l'ultima categoria introdotta potrebbe non esistere, ponendo arbitrariamente una coppia indeterminata in uno degli altri due gruppi; così si otterrebbe anche un risultato "completo", nel senso che ogni coppia risulta appartenente ad uno dei due primi gruppi, ma così facendo si comprometterebbe la "correttezza" della risposta. Fra queste due proprietà, l'autore sceglie di soddisfare la seconda.
Un'osservazione importante è che, per arrivare ad un risultato che soddisfi la proprietà di correttezza, è necessario assicurare che la tecnica risulti *monotona*. Con questo termine si intende che ogni coppia precedentemente identificata (ossia, non appertenente all'insieme delle coppie incerte) rimane tale quando si aggiungono ulteriori informazioni. Con "informazioni" l'autore sembra riferirsi esclusivamente alle regole, non ad altre tuple. Questo perché, dalla definizione data, si vede come la monotonia possa essere violata solamente nel caso in cui sia appunto una nuova regola ad essere aggiunta. Ciò non toglie che l'inserimento di nuove tuple provochi altri tipi di problemi: una nuova tupla potrebbe infrangere una delle regole esistenti; continuando l'esempio dei fumetti, è probabile che un fumetto precedentemente apparso su una certa pubblicazione, nel caso di una riedizione venga ospitato in un'altra testata. Ciò potrebbe infrangere una delle regole (in particolare, nel nostro esempio ad essere a rischio è una ILFD), con il risultato di compromettere la correttezza della risposta. Ovviamente, tale difetto può essere corretto, rimuovendo o correggendo (ma in quest'ultimo caso bisogna fare attenzione anche a rispettare la monotonia) le regole, ma ciò non è un compito semplice, soprattutto nel caso in cui queste siano state introdotte da più persone. Il problema diventa più serio via via che ci si avvicina alla completezza (ossia, quando si aggiungono nuove regole), generalmente provocando anche effetti va-

langa all'atto della modifica/immissione di una regola.
L'autore non si pone tale problema e quindi non propone nemmeno una eventuale soluzione; una possibile scelta potrebbe consistere nell'implementare un meccanismo che segnali le regole che vengono violate all'immissione di una nuova tupla; in questo modo sarà possibile decidere sul da farsi.
Oltre a queste possibilità vi è quella di riconoscere istanze dello stesso oggetto utilizzando dei metodi stocastici. Come rappresentante di questa categoria verrà mostrato brevemente un metodo non moderno, ma molto significativo: il *record linkage*. Questo metodo è stato ideato negli anni sessanta da Felligi e Sunter ([21]).
Date due relazioni A e B e due tuple $a \in A$ e $b \in B$, si definisce una funzione di confronto $\gamma(a, b)$, definita su $A \times B$. Si scelgono poi casualmente due campioni da $A \times B$. Il primo campione M contiene solo coppie che corrispondono, dove gli elementi si riferiscono alla stessa entità, mentre il secondo campione N contiene solo coppie che non corrispondono. Assumendo una distribuzione polinomiale si possono stabilire le probabilità condizionate $P(\gamma \mid M)$ e $P(\gamma \mid N)$. Ora è possibile definire il rapporto di somiglianza:

$$\lambda = \lambda(\gamma) = \frac{P(\gamma \mid M)}{P(\gamma \mid N)}$$

questa funzione indica se le due istanze a, b con $\gamma(a, b)$ si riferiscono allo stesso oggetto:

- se $\lambda < 1$: è probabile che le due istanze rappresentino un diverso oggetto.
- se $\lambda > 1$: è probabile che le due istanze rappresentino lo stesso oggetto.
- se $\lambda \approx 1$: il metodo non giunge a un risultato.

In questo modo si vengono a creare tre classi: una contenente le coppie che corrispondono, una contenente le coppie che non corrispondono e l'ultima formata dalle coppie incerte.
Questo metodo per funzionare necessita di una fase di apprendimento.

2.5.3 Il cambio di contesto

Un ulteriore prolema che sorge in fase di identificazione è quello che in [22] viene definito come *cambio di contesto*. Con questo termine si vogliono indicare tutte le differenze che intercorrono nella memorizzazione dello stesso attributo in due fonti separate. Nel nostro ambito, ricorrendo alla terminologia adottata in [23] e precedenti, ciò può essere visto come le differenze nella memorizzazione di più

attributi locali mappati sullo stesso attributo globale. Per chiarire meglio il senso di tale affermazione, è bene ricorrere ad un esempio: ammettiamo di avere due sorgenti A e B sulle quali è richiesto di effettuare il join sul campo "prezzo", presente in entrambe le fonti. A prima vista questo sembrerebbe un caso banale, in quanto l'attributo compare con lo stesso nome da entrambe le parti. Supponiamo però, come nel nostro ambito può accadere, che una fonte sia, ad esempio, in Italia, e l'altra negli Stati Uniti. Con ogni probabilità i prezzi sarebbero memorizzati rispettivamente in Euro e in Dollari, portando così ad un risultato scorretto dell'operazione. Infatti, supponendo che il join da utilizzare sia un equijoin, i risultati corretti non sarebbero dati da $A.prezzo = B.prezzo$ ma da $A.prezzo = < cambio\,attuale > * B.prezzo$. Il caso delle valute è molto complesso in quanto le continue fluttuazioni del cambio non sono utilizzabili in tempo reale. Per questo una soluzione potrebbe prevedere l'uso dei band join.
Nell'esempio il problema si verificava su un attributo di join, ma anche sugli altri campi si possono creare difficoltà, soprattutto durante la risoluzione dei conflitti. Senza un adeguato sistema per risolvere il cambio di contesto, alcuni valori potrebbero risultare in contrasto, anche se così in realtà non è. Questo problema viene risolto durante la fase di fusione delle tuple, principalmente tramite l'utilizzo di condizioni di join (quasi delle vere e proprie espressioni) create ad hoc grazie ad informazioni aggiuntive derivate da una conoscenza delle fonti.

In [24], [25] e [26] Neiling propone un metodo che insieme tenta di risolvere i problemi del cambio di contesto, della risoluzione dei conflitti e dell'identificazione degli oggetti. Tale metodo utilizza tre funzioni:

1) **funzione di conversione** h
   Converte il valore degli attributi servendosi anche di informazioni contestuali.

2) **funzione di confronto** f
   Confronta coppie di attributi e restituisce il grado di somiglianza.

3) **funzione di classificazione** δ
   Restituisce la classe di appartenenza di ogni istanza confrontata.

Queste funzioni vengono utilizzate in sequenza per processare i dati preparandoli per l'operazione di full outerjoin.
Sia U l'insieme degli oggetti del mondo reale e siano A e B, formate rispettivamente dagli insiemi di attributi X e Y, sorgenti contenenti istanze di oggetti in U. Siano $a \in A$ e $b \in B$ istanze di oggetti e sia D_{X_i} il dominio dell'attributo X_i; per un insieme di attributi $X = \{X_1, \ldots, X_n\}$ il dominio D_X è definito come il prodotto cartesiano dei domini D_i, con $i = 1, \ldots, n$.

Definizione 2.5.4 (Conversione) *Dato un insieme di attributi X con dominio D_X ed un attributo Z_0 con dominio D_{Z_0} una funzione di conversione è una funzione suriettiva $h_X : D_X \rightarrow D_{Z_0}$. Allora l'attributo Z_0 è detto convertibile (derivabile) da X, indicato con $X \vdash Z_0$.*

La conversione ha come scopo ultimo quello di rendere confrontabili due istanze provenienti da classi aventi attributi differenti, di modo da ottenere più informazioni possibili per poi semplificare la fase di confronto. Proprio con la conversione ci si può imbattere in cambi di contesto. Ad esempio:
$h_X : D_Y \rightarrow D_{\{Età\}}$, $h_Y : D_Y \rightarrow D_{\{Età\}}$
$h_X(Data \mid Oggi) =$Anno$(Oggi - Data)$ $h_Y(Età) =$Identità$(Età) = (Età)$
Dove $X = \{A.DataDiNascita\}$, $Y = \{B.Età\}$ $D_X = \{DATA\}$ $D_Y = D_{\{Età\}} = \{0, 1, 2, \dots\}$ e l'informazione contestuale utilizzata è la data odierna $Oggi$. Si ha quindi: $X \vdash \{Età\}$ e $Y \vdash \{Età\}$.
D'ora in poi si indicherà con $\mathbf{h}_X$ il vettore contenente tutte le funzioni di conversione per X.

Definizione 2.5.5 (Confronto) *Sia Z un insieme di d attributi derivabili sia da X xhe da Y con $\boldsymbol{h}_X$ e $\boldsymbol{h}_Y$. Allora una funzione di confronto f è un mapping*

$$f = D_Z \times D_Z \rightarrow R \subset \mathbb{R}^k, f(a', b') = f(\boldsymbol{h}_X(a), \boldsymbol{h}_Y(b))$$

degli elementi a e b ad uno spazio di confronto $R \subset \mathbb{R}^k$, dove $1 \leq k \leq d$.
Un semplice esempio di funzione di confronto può essere la seguente:

$$f(CAP_1, CAP_2) = \begin{cases} 0 & CAP_1 = CAP_2 \\ 1 & PrimaCifra(CAP_1) = PrimaCifra(CAP_2) \\ 2 & altrimenti \end{cases}$$

Come si vede sono possibili più gradi di somiglianza.

Definizione 2.5.6 (Classificazione) *Una regola di classificazione su uno spazio V di dimensione finita è una funzione di decisione $\delta : V \rightarrow \{classe_1, \dots, classe_n\}$ che restituisce la classe di appartenenza di $v \in V$.*

La classificazione può avvenire anche solo in due classi: **stesso** e **diverso**. Ogni coppia verrà classificata in una di queste classi, a seconda che le due tuple rappresentino o meno lo stesso oggetto. Assieme alle coppie vengono memorizzati anche gli identificatori locali delle tuple, utilizzati in seguito come condizioni per il full outerjoin. Tra i metodi di classificazione si ricordano il record linkage e il sorted neighborhood.
Per svolgere queste funzioni, l'autore propone un componente aggiuntivo: un particolare mediatore detto *identificatore* che riceve i dati dal mediatore classico e li restituisce trasformati e pronti per essere fusi.

Il principale difetto di questo metodo sta nella fase di conversione: nel caso ideale tale fase dovrebbe ricondurre gli schemi delle relazioni da fondere ad uno schema omogeneo, di modo che il compito delle fasi successive diventi più semplice. Però una omogeneizzazione del genere non è sempre possibile; molto dipende da quanto gli schemi sono sovrapposti: se sono pochi gli attributi che possono essere ricondotti ad uno schema comune, alllora il risultato delle fasi successive può diventare complicato ed impreciso.

Capitolo 3

Vista d'insieme dei principali algoritmi per query multisorgente

L'integrazione di dati provenienti da sorgenti eterogenee è un argomento trattato da molti ricercatori, ognuno dei quali ha cercato di risolvere il problema utilizzando tecniche diverse, ottenendo a volte buoni risultati. Di particolare interesse risulta essere il metodo proposto da Galindo-Legaria, che si basa sull'uso esclusivo della famiglia degli outerjoin per realizzare ciò che lui ha chiamato *full disjunction*. La particolarità di questo nuovo operatore è di preservare ogni possibile connessione tra i fatti; in altri termini, nella full disjunction ogni oggetto viene rappresentato con una sola istnza contenente ogni possibile informazione proveniente dalle relazioni di base. Il calcolo per giungere alla full disjunction utilizza un numero minimo di join; purtroppo però l'autore pone un limite all'utilizzo di tale operatore: il calcolo è possibile solo se lo schema del database non è ciclico. Questo comporta un grosso problema in quanto nella stragranze maggioranza dei casi lo schema di un database è proprio ciclico. Ancor di più tale restrizione diventa forte nell'ambito SEWASIE, dove i grafi si suppongono totalmente connessi (e quindi, con almeno tre relazioni, ciclici).

Basandosi sullo studio di Galindo-Legaria, Anand Rajaraman e Jeffrey D. Ullman propongono un algoritmo che utilizza la full disjunction in grado di funzionare anche un condizioni di ciclicità dello schema. Unica eccezione viene fatta per quanto riguarda gli schemi γ-*ciclici*. Nonostante esistano database per i quali questo algoritmo possa funzionare, nel nostro caso questa limitazione è inaccettabile, in quanto in generale le fonti da cui si attingono informazioni possono essere con schema γ-*ciclico*.

Un ulteriore passo avanti viene fatto tramite un terzo algoritmo, in grado di fornire risultati corretti anche in caso di γ-*ciclicità*. Questo algoritmo funzione in tutti i casi, ma per portare a termine con successo la query utilizza l'operatore OR, che ne rallenta drammaticamente l'esecuzione.

L'ultimo algoritmo che verrà preso in esame è stato studiato in ambito di tesi da Konstantin Eugeniev Kostenarov.
Questo algoritmo si propone di essere il primo in grado di rispondere correttamente in caso di γ-*ciclicità* senza utilizzare l'operatore di OR. Proprio questo algoritmo verrà analizzato più a fondo per mostrarne il funzionamento e per darne una definizione più rigorosa.

3.1 Lo studio di Galindo-Legaria

L'operatore di outerjoin nasce dall'esigenza di preservare tutti gli elementi di una relazione anche se questi non hanno corrispondenza nell'altra relazione. In particolare si parla di *full outerjoin* se vengono mantenuti tutti i campi di entrambe le relazioni e di *left outerjoin* se sono solo gli elementi della prima delle due relazioni a essere preservati (esiste anche la controparte del left outerjoin, detto *right outerjoin*, con cui si mantegono gli elementi della seconda relazione).
Questo operatore soffre però di un problema: in generale non è nè associativo nè commutativo.
La risoluzione e l'ottimizzazione delle query di tipo select/project/join sono basate sul fatto che queste query di join possono essere rappresentate senza ambiguità tramite un *grafo della query*. Il grafo non impone un ordinamento sulle operazioni, ed è costruito usando le relazioni come nodi ed i predicati di join come lati. Poiché l'ordine non è importante per la semantica di una query le uniche informazioni che il sistema ha bisogno per costruire l'albero in maniera corretta sono contenute nelle clausole select, from e where della query. Ogni implementazione che realizza il grafo della query produce quindi sempre lo stesso risultato.
Una query di tipo join/outerjoin invece, per via del non soddisfacimento delle proprietà associativa e commutativa, porta ad un grafo della query ambiguo. Può quindi accadere che due query aventi risultato differente producano lo stesso grafo della query.

La ricerca di Galindo-Legaria presentato in [33] parte da questi presupposti e tenta di ottenere un metodo per aggirare l'ostacolo posto dall'utilizzo degli outerjoin.

3.1.1 Concetti preliminari

Prima di addentrarci nel problema è bene dare alcune definizioni che saranno utili nel seguito.

- lo schema di una tupla t verrà indicato con $sch(t)$.

- quando a seguito di un'operazione di outerjoin, una tupla di una relazione non ha corrispondenze nell'altra, nel risultato ai campi non specificati viene attribuito il valore *NULL*. Questo si dice *padding* della tupla.

- siano R_1 e R_2 relazioni con schema S_1 e S_2 rispettivamente. L'operatore di *outerunion*, rappresentato con $\uplus$, prima esegue il padding sulle tuple di ogni relazione in base allo schema $(S_1 \cup S_2)$ e poi calcola l'unione degli insiemi risultanti. Questo operatore ha precedenza inferiore rispetto al join.

- siano R_1 e R_2 relazioni con schemi disgiunti S_1 e S_2 rispettivamente e sia p un predicato tale che $sch(p) \subseteq (sch(R_1) \cup sch(R_2))$. Allora 1)il join, 2)l'antijoin, 3)il left outerjoin, 4)il full outerjoin si definiscono rispettivamente:

  1. $R_1 \overset{p}{\bowtie} R_2 = \{(t_1, t_2) \mid t_1 \in R_1, t_2 \in R_2, p(t_1, t_2)\}$
  2. $R_1 \overset{p}{\triangleright} R_2 = \{t_1 \mid t_1 \in R_1, (\nexists t_2 \in R_2)(p(t_1, t_2))\}$
  3. $R_1 \overset{p}{\rightarrow} R_2 = R_1 \overset{p}{\bowtie} R_2 \uplus R_1 \overset{p}{\triangleright} R_2$
  4. $R_1 \leftrightarrow R_2 = R_1 \overset{p}{\bowtie} R_2 \uplus R_1 \overset{p}{\triangleright} R_2 \uplus R_2 \overset{p}{\triangleright} R_1$

- si dice che una tupla t_1 *sussume* una tupla t_2 se sono definite sugli stessi attributi, t_2 ha più valori nulli di t_1 ed i valori non nulli di t_2 concidono con quelli di t_1.
  La rimozione delle tuple sussunte di una relazione R viene indicata con $R\downarrow$

- l'unione minima delle relazioni R_1 e R_2 è: $R_1 \oplus R_2 = (R_1 \uplus R_2)\downarrow$.
  L'unione minima ha precedenza inferiore al join ed è commutativa ed associativa.

3.1.2 Outerjoin e Full Disjunction

Galindo-Legaria definisce una query *join-disjunctive* se consiste nell'unione minima delle varie queries di join in un dato grafo della query. Per poterle introdurre correttamente bisogna prima definire alcuni concetti.

Definizione 3.1.1 (Valutazione del join) *Sia $G = (V, E)$ un grafo di una query. La* valutazione del join *di un grafo di una query è $\bowtie(G) = \sigma_{p_1 \wedge \cdots \wedge p_n}\{R_1 * \cdots * R_m\}$, dove $\{p_1 \ldots p_n\}$ sono le labels dei lati E e $\{R_1 \ldots R_m\}$ sono le labels dei vertici V.*

Definizione 3.1.2 (Sottografo indotto) *Sia $G = (V, E)$ un grafo e $V' \subseteq V$. Il* sottografo indotto, *denotato con $G\mid'_V$, è (V', E'), dove $E' = \{(u, v) \mid (u, v) \in E, u \in V', v \in V'\}$.*

Si considerano equivalenti i simboli $\bowtie (G\mid'_V)$ e $\bowtie (V')$.

Definizione 3.1.3 (join-disjunction) *Sia $G = (V, E)$ un grafo di una query. Siano $V_1, \ldots, V_n \subseteq V$ insiemi di nodi tali che ogni grafo indotto $G\mid_{V_1}, \ldots, G\mid_{V_n}$ sia connesso. Allora la query $\bowtie (V_1) \oplus \cdots \oplus \bowtie (V_n)$ è detta* join-disjunction *su G. Ogni $\bowtie (V_i)$ è detto termine della query.*

Poiché sia l'outerunion sia il join sono commutativi ed associativi si ha che il grafo individuato da una join-disjunction non è ambiguo.
Poiché gli outerjoin possono essere definiti nel seguente modo:

$$
\begin{array}{ll}
R_1 \xrightarrow{p} R_2 = R_1 \overset{p}{\bowtie} R_2 \oplus R_1 & se R_1 = R_1\downarrow \\
R_1 \overset{p}{\leftrightarrow} R_2 = R_1 \overset{p}{\bowtie} R_2 \oplus R_1 \oplus R_2 & se R_1 = R_1\downarrow, R_2 = R_2\downarrow \\
(R_1 \oplus R_2) \overset{p}{\bowtie} R_3 = R_1 \overset{p}{\bowtie} R_3 \oplus R_2 \overset{p}{\bowtie} R_3 & se R_3 = R_3\downarrow
\end{array}
$$

Perciò è evidente (e dimostrabile) che le query join-disjunctive sono in grado di fornire alle query join/outerjoin una forma normale. Ora è quindi facile stabilire se due query basate sull'outerjoin sono equivalenti; è sufficiente confrontare le forme normali.
A questo punto può essere introdotto il concetto di *full disjunction*.

Definizione 3.1.4 (Full disjunction) *La* full disjunction *di un grafo G di una query è data dalla query join-disjunctive $Q = \{V' \mid G\mid_{V'} è\ \ connesso\}$*

La full disjunction porta in output tutte le tuple delle relazioni base e le combina in una singola tupla ogni volta che si presenta una corrispondenza fra esse.

Teorema 3.1.1 *Si assuma che tutti gli operatori in una query Q siano full outerjoin, e sia $G = graph(Q)$. Q calcola la full disjunction di G se e solo se G è aciclico.*

Dal teorema 3.1.1 si evince che il problema di associatività legato all'operatore di outerjoin è dovuto alla ciclicità del grafo della query.
Con il seguente esempio si metterà in luce quale sia l'esatta conseguenza di questo problema sul risultato di una query.

Si voglia calcolare il full outerjoin delle seguenti relazioni S, R, J utilizzando come attributo di join "*città*". Siano i predicati p^{sr}, p^{rj}, p^{sj} rispettivamente $(S.città = R.città)$, $(R.città = J.città)$ e $(S.città = J.città)$.

S

SID	$città$
$S1$	$Londra$
$S2$	$Parigi$

R

RID	$città$
$R1$	$Parigi$
$R2$	$Oslo$

J

JID	$città$
$J1$	$Oslo$
$J2$	$Londra$

$$Q_1 = (S \overset{p^{sr}}{\leftrightarrow} R) \overset{p^{sj}}{\leftrightarrow} J$$

SID	$S.città$	RID	$R.città$	JID	$J.città$
$S1$	$Londra$	NULL	NULL	$J2$	$Londra$
$S2$	$Parigi$	$R1$	$Parigi$	NULL	NULL
NULL	NULL	$R2$	$Oslo$	NULL	NULL
NULL	NULL	NULL	NULL	$J1$	$Oslo$

$$Q_2 = (S \overset{p^{sr}}{\leftrightarrow} R) \overset{p^{rj}}{\leftrightarrow} J$$

SID	$S.città$	RID	$R.città$	JID	$J.città$
$S1$	$Londra$	NULL	NULL	NULL	NULL
$S2$	$Parigi$	$R1$	$Parigi$	NULL	NULL
NULL	NULL	$R2$	$Oslo$	$J1$	$Oslo$
NULL	NULL	NULL	NULL	$J2$	$Londra$

Nonostante la valutazione della query dei grafi $G1 = graph(Q1)$ e $G2 = graph(Q2)$ sia la stessa, la loro full disjunction, riportata nelle due tabelle precedenti, è differente. In particolare si può osservare che, mentre le informazioni presenti nelle sorgenti vengono tutte riportate in output, si ha che in entrambi i risultati manca una connessione fra le tuple. In particolare, nel risultato di Q_1 la terza e la quarta tupla dovrebbero essere state fuse, mentre nella seconda tabella sono la prima e la quarta riga che sarebbero dovute essere congiunte.
Il risultato corretto dovrebbe essere:

$$Q = (S \overset{p^{sr}}{\leftrightarrow} R) \overset{p^{rj}}{\leftrightarrow} J$$

SID	$S.città$	RID	$R.città$	JID	$J.città$
$S1$	$Londra$	NULL	NULL	$J2$	$Londra$
$S2$	$Parigi$	$R1$	$Parigi$	NULL	NULL
NULL	NULL	$R2$	$Oslo$	$J1$	$Oslo$

Ciò che emerge chiaramente da questo studio è che il problema dell'outerjoin risiede nella ciclicità del grafo rappresentativo della query a cui questo operatore appartiene. Quindi l'outerjoin non soddisfa la proprietà associativa solo se si ha a che fare con un ciclo.

3.2 L'algoritmo di Rajaraman e Ullman

Lo studio di Rajaraman e Ullman proposto in [34] parte da dove Galindo-Legaria era arrivato, ossia dalla tesi secondo la quale non sarebbe possibile calcolare la

Full Disjunction di un grafo se questo è ciclico.
Lo scopo dei due studiosi è quello di dimostrare che è possibile allargare la fascia di problemi trattabili con la Full Disjunction , provando che il limite non è posto da una ciclicità generica, bensì dalla γ-*ciclicità*.

3.2.1 Concetti preliminari

Prima di affrontare la trattazione, è opportuno introdurre alcuni concetti fondamentali per la comprensione della stessa.

- l'*ipergrafo* dello schema di un database è costruito assumendo come nodi gli attributi e come iperarchi gli schemi delle relazioni o degli insiemi di attributi. Prendendo ad esempio tre relazioni con schemi UDF, UDS, UA. L'ipergrafo corrispondente è riportato in figura 3.1

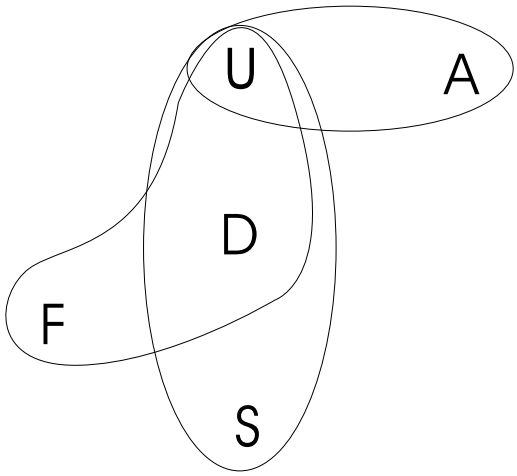


Figura 3.1: Esempio di ipergrafo

- si assume che le tuple siano *join-consistenti*, ossia che sugli attributi in comune i valori sino coincidenti. In altre parole non si prendono in considerazione i possibili conflitti.

- l'unico operatore trattato nel seguito sarà il *natural outerjoin*, indicato con $\overset{\circ}{\bowtie}$ e definito come segue.
  Siano R e S due relazioni. Lo schema di $R \overset{\circ}{\bowtie} S$ è $R \cup S$ e consiste nelle seguenti tuple:

1. tutte le tuple di $R \bowtie S$, il natural join di R e S. Questo tipo di join non consente che due valori NULL siano considerati uguali. Quindi le tuple t prodotte sono tutte quelle presenti nello schema $R \cup S$ tali che ci siano tuple r in R e s in S che concordino e che siano non nulle in tutti gli attributi di $R \cap S$. La tupla risultante t concorda con r e/o s in ogni attributo di $R \cup S$.
2. per tutte le tuple r di R che non fanno join con nessuna tupla di S, una tupla t che ha r negli attributi di R e NULL negli attributi di $R - S$.
3. per tutte le tuple s di S che non fanno join con nessuna tupla di R, una tupla t che ha s negli attributi di S e NULL negli attributi di $S - R$.

- un ipergrafo si dice *disconnesso* se è possibile partizionare i suoi iperarchi in due insiemi non vuoti tali che nessun nodo compaia in entrambi. Altrimenti l'ipergrafo si dice *connesso*.

- Sia $\mathcal{R} = R_1, R_2, \ldots, R_n$ un insieme di relazioni le cui tuple non hanno valori nulli. Diciamo che R è la Full Disjunction di $\mathcal{R}$ se valgono le seguenti condizioni:

  – *non c'è ridondanza:* Nessuna tupla di R sussume un'altra tupla di R.

  – *ogni tupla di R proviene da parti connesse di $\mathcal{R}$:* Sia t una tupla di R. Allora esiste un sottoinsieme connesso delle relazioni di $\mathcal{R}$ tali che t, ristretta ai suoi componenti non nulli, è il join delle tuple di tali relazioni.

  – *ogni connessione è rappresentata:* Siano $t_1, t_2, \ldots, t_k$ tuple scelte da relazioni distinte $R_{i1}, R_{i2}, \ldots, R_{in}$ rispettivamente, tali che $\{R_{i1}, R_{i2}, \ldots, R_{in}\}$ sia un ipergrafo connesso.
  Sia ogni t_i join-consistente.
  Sia t la tupla che concorda con ognuna delle t_i per ogni attributo che compare in almeno una delle relazioni $R_{i1}, R_{i2}, \ldots, R_{in}$ e che ha NULL in tutti gli altri attributi presenti negli schemi di $\mathcal{R}$.
  Allora t è sussunta da un'altra tupla di R.

- viene definito *ordinamento corretto* di outerjoin per uno schema $\mathcal{R} = R_1, R_2, \ldots, R_n$ un'espressione nella quale:

  1. ogni relazione appare una ed una sola volta.
  2. l'unico operatore utilizzato è $\overset{\circ}{\bowtie}$.
  3. il risultato è la Full Disjunction di $\mathcal{R}$.

A questo punto è bene introdurre il seguente teorema:

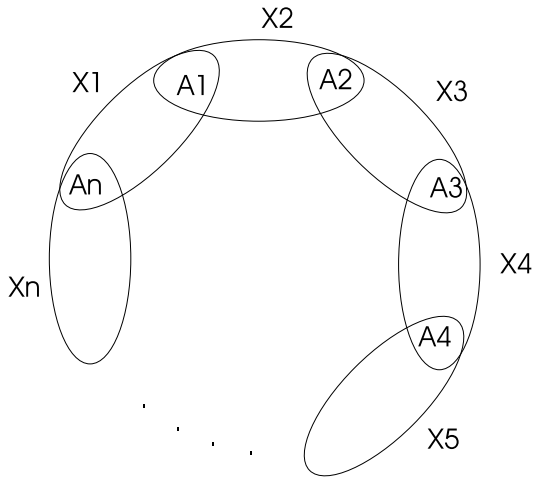


Figura 3.2: Ciclo puro

Teorema 3.2.1 *La Full Disjunction è unica*

Dimostrazione. Tenendo presente la definizione di Full Disjunction data precedentemente, si può vedere come l'unica relazione R che la soddisfi consista di tuple t aventi le seguenti caratteristiche:

- t è data dal join di alcune tuple join-consistenti provenienti da un sottoinsieme connesso delle relazioni $R_1, R_2, \ldots, R_n$ a cui è stato successivamente applicato il padding.

- non esiste un'altra relazione fra le R_i contenente una tupla che sia join-consistente con t.

□

3.2.2 La γ-*ciclicità*

In questa sede verrà riportato solo lo stretto indispensabile per riuscire a comprendere il seguito dello studio. Per definizioni più puntuali ed approfondite e per un numero maggiore di casi di γ-*ciclicità* e per definizioni equivalenti a quella fornita, si faccia riferimento a [35] o a [36] .

Un **ciclo puro** è una collezione di $n \geq 3$ iperarchi e nodi come quello mostrato in figura 3.2;

in particolare:
Ogni coppia di iperarchi X_i ed X_{i+1} (stesso discorso vale anche per la coppia X_n ed X_1) hanno almeno un nodo A_i (A_n, nel caso di X_n ed X_1) in comune. Inoltre nessuno dei nodi condivisi A_j appare in più di due iperarchi del ciclo puro. Questi nodi possono comunque apparire in iperarchi al di fuori del ciclo puro.
Ci può essere più di un nodo nelle intersezioni, ma questi nodi non devono apparire in nessun altro iperarco del ciclo puro.

Un γ-3-$ciclo$ è un insieme di tre iperarchi come quello mostrato in figura 3.3.

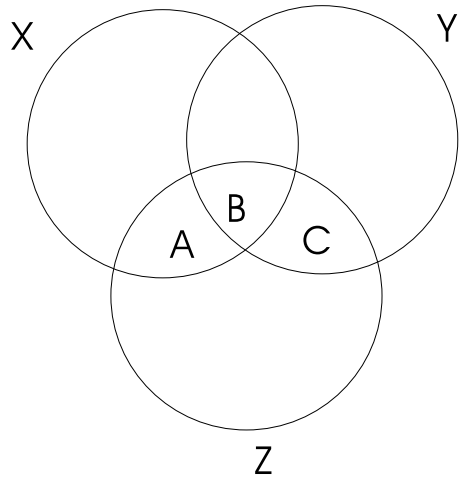


Figura 3.3: γ-3-ciclo

In particolare, i nodi A, B e C devono esistere; una qualsiasi altra regione del diagramma può essere vuota o meno e le regioni contenenti A, B e C possono contenere o meno altri nodi.
In altre parole, ci devono essere alcuni nodi in tutti e tre gli iperarchi: uno che sia solo in X e Z e uno che sia solo in Y e Z. Per esempio, il più piccolo nei tre iperarchi $\{ABC, AB, BC\}$.

Un γ-$ciclo$ è un ciclo con almeno tre archi, avente struttura simile a quella presentata in figura 3.2. A differenza del ciclo puro, un γ-$ciclo$ permette ad A_n di comparire in ognuno degli iperarchi $X_2, \ldots, X_{n-1}$ oltre che in X_1 ed X_n. Tutti gli altri nodi $A_1, \ldots, A_{n-1}$ nelle intersezioni appaiono solamente nelle X_i mostrate nel grafico.

Un ipergrafo si dice γ-$aciclico$ se e solo se non ha γ-$cicli$. Equivalentemente,

un ipergrafo è γ-$aciclico$ se e solo se non contiene cicli puri o γ-3-$cicli$.
Un ipergrafo si dice γ-*ciclico* se e solo se non è γ-$aciclico$.
Un altro modo per caratterizzare la γ-*ciclicità* risiede nei *diagrammi di Bachman*. Dato un ipergrafo $\mathcal{H}$, è possibile aggiungergli come iperarchi addizionali tutte le intersezioni di due o più iperarchi di $\mathcal{H}$. Sia $\mathcal{G}$ l'ipergrafo risultante. Si crea poi un grafo (non un ipergrafo) i cui nodi sono gli iperarchi di $\mathcal{G}$ e per il quale esiste un arco dal nodo E al nodo F se e solo se $E \subset F$ ed il contenimento è il più vicino possibile; ossia non esiste un nodo G tale che $E \subset G \subset F$. Il grafo risultante è il diagramma di Bachman di $\mathcal{H}$, denotato con $BD\mathcal{H}$.
Il diagramma di Bachman è aciclico se, trattato come un grafo non orientato, non ci sono cicli. Si può provare che un ipergrafo $\mathcal{H}$ è γ-$aciclico$ se e solo se $BD\mathcal{H}$ è aciclico. In figura 3.4 è riportato il diagramma di Bachman relativo all'ipergrafo che ha come iperarchi $\{AB, BC, CD\}$.

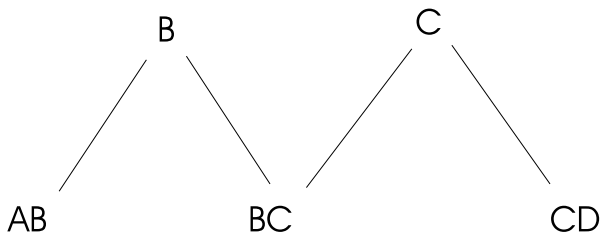


Figura 3.4: Esempio di diagramma di Bachman

3.2.3 L'importanza della γ-*ciclicità* per la Full Disjunction

Lo scopo dello studio dei due autori è quello di dimostrare che la γ-$aciclicità$ è il limite invalicabile al cacolo di un ordinamento corretto degli outerjoin per la Full Disjunction .
La Full Disjunction può essere calcolata infatti per ogni tipo di ipergrafo da una espressione di qualche tipo, ma non rispetterà tutte le regole di ordinamento corretto. Ciò significa che una relazione potrà comparire in più di un join, o che sarà necessario utilizzare altri operatori oltre a quello di outerjoin.
Soprattutto questo risultato è da considerarsi importante per gli algoritmi che verranno in seguito. Infatti, grazie a questa prova è possibile affermare che, sia l'algoritmo che utilizza l'OR, sia l'algoritmo di Konstantin, non producono un ordinamento corretto, in quanto violano il vincolo posto sulla γ-*ciclicità*.

Nel seguito verrà provato separatamente che non esiste un ordinamento corretto per i γ-3-$cicli$ e per i cicli puri.

Lemma 1 *Siano R, S e T relazioni tali che:*

- *esista un attributo A negli schemi di S e T che non sia nello schema di R.*
- *esista un attributo B sia in R che in T (e forse in S).*

Allora $(R \overset{\circ}{\bowtie} S) \overset{\circ}{\bowtie} T$ non è un ordinamento corretto.

Dimostrazione. Si supponga che R e T contengano una sola tupla, r e t rispettivamente, e che siano join-consistenti (ossia concordano in B e negli altri attributi comuni). Sia $S = \emptyset$. Allora $R \overset{\circ}{\bowtie} S$ è r con il padding per gli attributi di $S - R$ e quindi anche per A. A causa di ciò, la tupla $R \overset{\circ}{\bowtie} S$ non andrà in join con t perché non concordano su A. Quindi il risultato sarà formato da due tuple e la Full Disjunction non viene calcolata correttamente. □

Lemma 2 *Nessun γ-3-$ciclo$ ha un ordinamento corretto.*

Dimostrazione. Si consideri il diagramma di figura 3.3. Se ci fosse un ordinamento corretto delle tre relazioni X, Y e Z, una di queste dovrebbe essere l'ultima. Se Z fosse l'ultima, allora l'attributo B in figura può essere visto come l'attributo B del lemma 1 e A o C in figura come A del lemma, portando così ad affermare che non esiste un ordinamento corretto.
Se X fosse l'ultima, si può scrivere l'espressione come $(Y \overset{\circ}{\bowtie} Z) \overset{\circ}{\bowtie} X$. Poiché l'outerjoin è commutativo, l'espressione scritta è equivalente a $(Z \overset{\circ}{\bowtie} Y) \overset{\circ}{\bowtie} X$, sia Y che Z possono essere visti come la S del lemma. Tenendo valida la prima scelta, l'attributo A nell'espressione può essere visto come A del lemma e B come B del lemma, dimostrando che l'ordinamento non è valido.
Se Y fosse l'ultima, vale simmetricamente quanto detto per X. □ Si può inoltre dimostrare che:

Lemma 3 *nessun ipergrafo con un γ-3-$ciclo$ ha un ordinamento corretto.*

Segue ora la parte riguardante i cicli puri.

Lemma 4 *Nessun ciclo puro ha un ordinamento corretto.*

Dimostrazione. Si consideri l'ipergrafo in figura 3.2. Nel seguito si assuma inoltre di operare in modulo n. Ossia $X_0 = X_n$, $X_1 = X_{n+1}$ e così via. Stesso discorso vale per gli A_i.
Si supponga di avere un ordinamento corretto E. In E l'ultimo outerjoin ha due

argomenti, costituiti da due insiemi S_1 e S_2, contenenti una partizione degli X_i. Si supponga che S_2 abbia almeno due membri; il fatto che $n \geq 3$ assicura che almeno un insieme possa assumere il ruolo di S_2.
Così si possono trovare sequenze di uno o più iperarchi n S_1 come ad esempio $X_i, \ldots, X_j$, tali che in S_1 non siano presenti X_{i-1} e X_{j+1}. Si consideri ora un database dove $X_i, \ldots, X_{j+1}$ contegono ognuna una tupla e che queste tuple siano join-compatibili. Tutte le altre relazioni sono vuote. Allora la Full Disjunction di queste relazioni contiene quella tupla risultante dai join di $X_i, \ldots, X_{j+1}$ completa di padding.
Per quanto riguarda l'altra partizione, il risultato dell'outerjoin su S_2 non può contenere una tupla che abbia valore non nullo per l'attributo A_{i-1} (attributo di intersezione fra X_{i-1} e X_i). Questo perché S_2 ha almeno due membri, e quindi $i \neq j - 2$ (modulo n). Quindi X_{i-1} non può essere X_j o X_{j+1} e quindi A_{i-1} non è in X_{j+1}.
Di conseguenza, quando si applica l'ultimo outerjoin di E, non si può ottenere una singola tupla. Qualsiasi siano le tuple (o tupla) risultante dai join di $X_i, \ldots, X_j$, almeno una di queste ha un valore non nullo per A_{i-1} (si parla di più tuple per ammettere che già S_1 possa aver fallito nel creare la Full Disjunction). Dal momento che questa tupla non può andare in join con nessuna tupla di S_2, E non può contenere una sola tupla. Quindi E non può contenere la Full Disjunction del grafo. □ Si può poi provare che:

Lemma 5 *nessun ipergrafo che contiene un ciclo puro ha un ordinamento corretto.*

Dai lemmi 3 e 5 si ottiene che:

Teorema 3.2.2 *Se un ipergrafo ha un ordinamento corretto degli outerjoin, allora deve essere* γ-ciclico.

A questo punto bisogna dimostare l'inverso del teorema 3.2.2.
Prima di tutto è bene fare notare che, nel caso di ipergrafo sconnesso, non si può ottenere la Full Disjunction a meno che non si utilizzi anche l'operatore di outerunion.

Si definisce γ-*decomposizione* di un ipergrafo connesso $\mathcal{H}$ una coppia $(\mathcal{H}_1, \mathcal{H}_2)$ tale che:

- $\mathcal{H}_1, \mathcal{H}_2$ sono insiemi non-vuoti e disgiunti di $\mathcal{H}$ che ne includono tutti gli iperarchi.
- i nodi di $\mathcal{H}_1$ e $\mathcal{H}_2$ sono formati dall'unione degli iperarchi degli ipergrafi corrispondenti. (ossia, $\mathcal{H}_1$ e $\mathcal{H}_2$ sono generati dagli iperarchi).

- Sia X l'insieme di nodi che sono sia in $\mathcal{H}_1$ che in $\mathcal{H}_2$; allora:
  - X non è vuoto
  - ogni iperarco di $\mathcal{H}$ contiene X oppure è disgiunto da esso.

Data questa definizione, si può esporre il seguente lemma:

Lemma 6 *sia $(\mathcal{H}_1, \mathcal{H}_2)$ una γ-decomposizione di un ipergrafo connesso $\mathcal{H}$ e si supponga $R_1 = FD(\mathcal{H}_1)$ e $R_2 = FD(\mathcal{H}_2)$. Allora $R_1 \overset{\circ}{\bowtie} R_2 = FD(\mathcal{H})$.*

Dimostrazione. Sia X l'intersezione dei nodi di $\mathcal{H}_1$ e $\mathcal{H}_2$. Si dimostra che la tupla t è in $R_1 \overset{\circ}{\bowtie} R_2$ se e solo se è in $FD(\mathcal{H})$

SE: Si consideri una tupla t in $FD(\mathcal{H})$. Sia $\mathcal{G}$ il sottoinsieme connesso di iperarchi di $\mathcal{H}$ tale che t sia formata dal join delle tuple corrispondenti ad ogni iperarco di $\mathcal{G}$ con il padding, se necessario. La situazione è simile a quella mostrata in figura 3.5. Sia t_i la restrizione di t ai nodi di $\mathcal{H}_i$, con $i = 1, 2$. Allora t_i è data dal

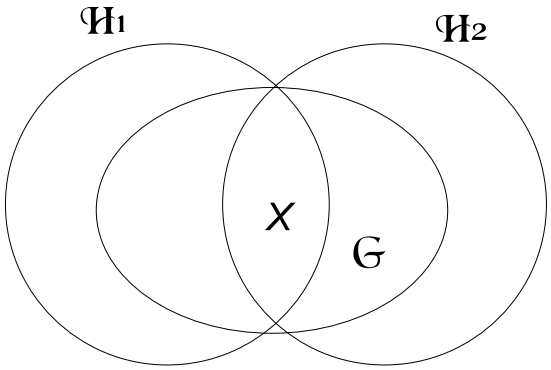


Figura 3.5: Grafico per il lemma 6

join delle relazioni corrispondenti agli iperarchi di $\mathcal{G} \cap \mathcal{H}_i$.
Per ipotesi t_i (con eventuale padding) è in relazione con R_i.
Supponendo che t_1 sia nulla, allora $\mathcal{G} \subseteq \mathcal{H}_\in$. Quindi $t = t_2$ (con padding) e prendendo il risultato di $R_1 \overset{\circ}{\bowtie} R_2$, a t_2 è applicato il padding, così da creare t. Discorso simmetrico vale se t_2 fosse nulla.
Se invece t_1 e t_2 non sono nulle, allora $\mathcal{G}$ interseca sia $\mathcal{H}_1$ che $\mathcal{H}_2$. Dal momento che $\mathcal{G}$ è connesso, ci deve essere almeno un iperarco E_1 di $\mathcal{G} \cap \mathcal{H}_1$ ed almeno un E_2 di $\mathcal{G} \cap \mathcal{H}_2$ tale che ognuno contenga un insieme di attributi di X. Così, sia t_1 che t_2 sono non nulli negli attributi di X. Ovviamente, dal momento che entrambi provengono da t, sono concordi in tali attributi, così, in $R_1 \overset{\circ}{\bowtie} R_2$ vanno in Join per dare t. Quindi: $FD(\mathcal{H}) \subseteq R_1 \overset{\circ}{\bowtie} R_2$

SOLO SE: Si supponga che le tuple t_1 e t_2 (rispettivamente provenienti da R_1 e R_2) possano andare in join. Allora t_i è in $FD(\mathcal{H}_i)$, per $i = 1, 2$. Cioè t_i è il join di qualche insieme di tuple massimale scelto dalla relazione corrispondente all'insieme connesso di iperarchi di $\mathcal{H}_i$. Dal momento che questi insiemi di iperarchi devono includerne uno che contenga X (altrimenti t_1 et_2 non potrebbero fare join in $R_1 \overset{\circ}{\bowtie} R_2$), anche l'unione di questi insiemi è connesso.
Sia t la tupla che concorda con t_1 e t_2 in tutti gli attributi. Allora t deve essere massimale, nel senso che non si possono estendere i suoi componenti non nulli facendo join con tuple appartenenti ad altre relazioni. Se potessimo fare ciò, allora o t_1 o t_2 non sarebbero massimali e quindi non sarebbero nelle rispettive Full Disjunction . Quindi $t \in FD(\mathcal{H})$, provando $R_1 \overset{\circ}{\bowtie} R_2 \subseteq FD(\mathcal{H})$. □ Da ciò discende che è possibile risalire alla Full Disjunction applicando ricorsivamente la γ-decomposizione finché ciò è possibile. Questo processo è detto γ-riduzione e porta ad avere una collezione di ipergrafi con solo iperarco.

Lemma 7 *ogni ipergrafo γ-aciclico con più di un iperarco ammette una γ-decomposizione.*

Dimostrazione. schizzo
Si costruisca il diagramma di Bachman. Attribuendo a X lo stesso significato che ha nella definizione di γ-disgiunzione, si ha che assume il ruolo di nodo minimale nel contesto del diagramma di Bachman. A questo punto, si possono presentare due casi:

caso 1) se X ha un solo nodo adiacente: $\mathcal{H}_1 = \{X\}$ e $\mathcal{H}_2$ è composto da tutti gli altri nodi,

caso 2) altrimenti X è adiacente a più componenti connesse, allora: come $\mathcal{H}_1$ si prenda una qualsiasi delle componenti e come $\mathcal{H}_2$ si assuma X e le componenti rimanenti.

□

3.2.4 L'algoritmo SOJO

A questo putno è possibile definire un algoritmo per il calcolo di un ordinamento corretto di outerjoin.

INPUT: un ipergrafo $\mathcal{H}$ connesso e γ-aciclico.
OUTPUT: un ordinamento corretto per $\mathcal{H}$.
METODO: Si costruisce l'algoritmo di Bachman per $\mathcal{H}$ e lo si partziona ricorsivamente come indicato nel lemma 7.

CASO BASE: si ha un solo iperarco e quindi la relazione restituita è chiaramente la Full Disjunction di $\mathcal{H}$.
INDUZIONE: $\mathcal{H}$ ha più di un iperarco:

1. si trovi una γ-decomposizione per $\mathcal{H}$. Sia $(\mathcal{H}_1, \mathcal{H}_2)$ questa decomposizione, ottenibile come indicato nel lemma 7.

2. si partizioni $BD(\mathcal{H})$ in due parti corrispondenti a $\mathcal{H}_1$ e $\mathcal{H}_2$ Nel primo caso del lemma 7 si prende come $\mathcal{H}_1$ X e come $\mathcal{H}_2$ gli altri nodi, nel secondo caso si rimuove X, si separano i componenti del diagramma e si aggiunge X a $\mathcal{H}_2$ solo se è necessario. Cioè: se X è un iperarco di $\mathcal{H}$ o se $\mathcal{H}_2$ è composto da più di un componente allora lo si include.

3. Si trovano ricorsivamente le espressioni di E_1 e E_2, ordinamenti corretti per $\mathcal{H}_1$ e $\mathcal{H}_2$.

4. si restitiusce l'espressione $E_1 \overset{\circ}{\bowtie} E_2$

Per induzione è possibile dimostrare facilmente il seguente teorema:

Teorema 3.2.3 *l'algoritmo SOJO calcola un ordinamento corretto degli outerjoin per $\mathcal{H}$.*

Infine è facilmente verificabile anche il seguente teorema:

Teorema 3.2.4 *Un ipergrafo ha un ordinamento corretto degli outerjoin se e solo se è γ-aciclico e connesso. Inoltre, l'ordinamento può essere ottenuto tramite γ-riduzione.*

3.3 Confronto fra gli studi di Galindo-Legaria e di Ullman

Teoricamente lo studio di Ullman appena esposto è perfetto, anche perché supportato da numerose dimostrazioni, ma non risolve i problemi lasciati in sospeso da Galindo-Legaria, anche se così appare ad un primo esame. Consideriamo infatti l'esempio che viene riportato in [34] ed utilizzato per provare l'esattezza delle tesi degli autori: sono disponibili tre relazioni (che chiameremo, per semplicità, L_1, L_2, L_3), con schema UDF, UDS e UA. Viene dimostrato che un'ordinamento corretto è dato dalla sequenza: $(L_1 \overset{\circ}{\bowtie} L_2) \overset{\circ}{\bowtie} L_3$. Infatti, supponendo che UDF e UA contengano l'unica tupla udf e ua rispettivamente e che UDS sia vuota, con questo ordinamento si ottiene $udf \perp a$.

Del resto però, provando ad utilizzare un semplice full outerjoin, come quelli utilizzati da Galindo-Legaria nel suo studio, utilizzando la sequenza $(UDF \overset{p^{UDF-UDS}}{\leftrightarrow} UDS) \overset{p^{UDF-UA}}{\leftrightarrow} UA$ si ottiene come risultato proprio $udf \perp a$, esattamente come nel caso precedente. Quindi, a differenza di quanto affemato da Ullman, la full disjunction si ottiene anche utilizzando il metodo di Galindo-Legaria. In realtà questo esempio è un caso particolare; Ullman ha semplicemente scelto i dati in modo infelice. Infatti, supponendo che le relazioni contengano le seguenti tuple:

L_1

U	D	F
U_1	D_1	F_1
U_2	D_2	F_2

L_2

U	D	S
U_2	D_2	S_2
U_3	D_3	S_3

L_3

U	A
U_3	A_3
U_1	A_1

in questo caso, il risultato finale con il metodo di Galindo-Legaria sarebbe:

$(UDF \overset{p^{UDF-UDS}}{\leftrightarrow} UDS) \overset{p^{UDF-UA}}{\leftrightarrow} UA$

$UDF.U$	$UDF.D$	$UDF.F$	$UDS.U$	$UDS.D$	$UDS.S$	$UA.U$	$UA.A$
U_1	D_1	F_1	$NULL$	$NULL$	$NULL$	$NULL$	$NULL$
U_2	D_2	F_2	U_2	D_2	S_2	$NULL$	$NULL$
$NULL$	$NULL$	$NULL$	U_3	D_3	S_3	U_3	A_3
$NULL$	$NULL$	$NULL$	$NULL$	$NULL$	$NULL$	U_1	A_1

Come si può ben vedere, manca una connessione fra la prima e l'ultima tupla della relazione. Invece, utilizzando quanto indicato da Ullman:

$(UDF \overset{\circ}{\bowtie} UDS) \overset{\circ}{\bowtie} UA$

U	D	F	S	A
U_1	D_1	F_1	$NULL$	A_1
U_2	D_2	F_2	S_2	$NULL$
U_3	D_3	$NULL$	S_3	A_3

Quindi è vero che lo studio di Ullman riesce a calcolare la full disjunction anche in casi in cui Galindo-Legaria riteneva l'operazione impossibile, ma questo è dovuto unicamente al fatto che Ullman utilizza un operatore diverso e non standard (secondo SQL92): il natural outerjoin. Per rendere chiara la differenza, si potrebbe dire che il full outerjoin sta al natural outerjoin come il thata join sta al natural join; il metodo di Ullman funziona meglio solo perché il natural outerjoin "compatta" gli schemi automaticamente dopo ogni join, fondendo le colonne rappresentanti lo stesso attributo provenienti da sorgenti differenti; in questo modo non si verifica più il problema sollevato da Galindo-Legaria tramite l'esempio a pagina 56. Vediamo infatti dall'esempio precedente come risulterebbe la relazione in uscita dal primo join in entrambi i casi: prima secondo Galindo-Legaria:

$(UDF \overset{p^{UDF-UDS}}{\leftrightarrow} UDS) \overset{p^{UDF-UA}}{\leftrightarrow} UA$

$UDF.U$	$UDF.D$	$UDF.F$	$UDS.U$	$UDS.D$	$UDS.S$
U_1	D_1	F_1	$NULL$	$NULL$	$NULL$
U_2	D_2	F_2	U_2	D_2	S_2
$NULL$	$NULL$	$NULL$	U_3	D_3	S_3

come è ben chiaro, qualsiasi sia il join che segue, o con UDF o con UDS, si ha che viene persa una connessione. Invece secondo Ullman non si ha questo problema:

$(L_1 \overset{\circ}{\bowtie} L_2)$

U	D	F	S
U_1	D_1	F_1	$NULL$
U_2	D_2	F_2	S_2
U_3	D_3	$NULL$	S_3

Quindi Ullman non affronta il problema, bensì lo aggira; inoltre evidentemente l'autore si cura solamente dei pregi di questo operatore, senza considerarne alcuni difetti. L'utilizzo del natural join porta forzatamente ad eseguire l'equijoin su tutti gli attributi comuni delle relazioni; mentre ciò in alcuni casi può bastare, in generale è visto come un vincolo troppo forte, perciò la sequenza prodotta dall'algoritmo SOJO diventa inutilizzabile in molti casi; anche quando l'utilizzo del natural outerjoin viene accettato, rimane comunque il problema di come tradurre la sequenza prodotta in un'unica query SQL: quale operatore standard si dovrà utilizzare? Quindi Ullman fornisce sì una sequenza, ma poi non è chiaro come utilizzarla.
In realtà quello di Ullman è quasi uno studio indipendente da quello di Galindo-Legaria piuttosto che una sua continuazione. Ciò che voleva affermare l'autore al termine del primo studio era che il limite della full disjunction calcolata utilizzando *un'unica* query costituita *esclusivamente* da operatori *standard* della famiglia degli outerjoin è rappresentato dalla ciclicità dello schema. Ullman ha sì esteso il problema, ma partendo da presupposti diversi.

Precedentemente è stato mostrato come un caso di grafo ciclico sia stato risolto tramite il metodo di Galindo-Legaria, nonostante lui avesse affermato che tale operazione non era possibile. In realtà ci sono alcuni casi, anche con grafi ciclici, che possono essere risolti senza ricorrere al metodo di Ullman. Tali casi però sono dipendenti dal valore delle tuple, e quindi non predicibili a priori; quindi quanto detto da Galindo-Legaria è corretto. Comunque, per completezza, tali casi possono essere riconosciuti ridefinendo il concetto di ipergrafo: si mantiene il precedente significato per gli iperarchi (che rappresentano quindi le relazioni) ma il concetto di nodo viene modificato: ora ogni nodo rappresenta un oggetto,

non più un attributo. Partendo da questo presupposto si può definire una sorta di "γ-ciclicità degli oggetti". In particolare si può affermare che il metodo di Galindo-Legaria funziona quando, in caso di ciclicità del grafo, non c'è un ciclo puro nell'ipergrafo così ridefinito.
La figura 3.6 mostra l'ipergrafo dell'esempio precedente risolvibile solo tramite il metodo di Ullman, mentre la figura 3.7 quello corrispondente ai seguenti dati:

L_1			L_2			L_3	
U	D	F	U	D	S	U	A
U_1	D_1	F_1	U_2	D_2	S_2	U_3	A_3
U_2	D_2	F_2	U_3	D_3	S_3	U_4	A_4

Per comodità si è considerato come oggetto il numero accanto agli attributi (ad esempio, per indicare l'oggetto che ha come valore degli attributi U_1, D_1, F_1, A_1 si è utilizzato semplicemente il numero 1). Con facilità si può vedere come il risul-

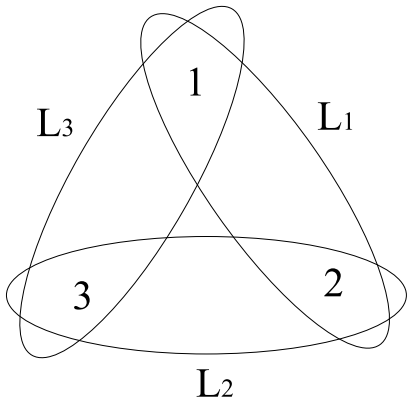


Figura 3.6: Ipergrafo di un ciclo puro

tato dell'esempio corrispondente al grafo senza cicli puri sia realmente risolvibile da Galindo-Legaria, utilizzando la sequenza: $(L_1 \overset{p^{UDF-UDS}}{\leftrightarrow} L_2) \overset{p^{UDF-UA}}{\leftrightarrow} L_3$. Si ottiene infatti:

$(L_1 \overset{p^{L_1-L_2}}{\leftrightarrow} L_2) \overset{p^{L_2-L_3}}{\leftrightarrow} L_3$

$L_1.U$	$L_1.D$	$L_1.F$	$L_2.U$	$L_2.D$	$L_2.S$	$L_3.U$	$L_3.A$
U_1	D_1	F_1	$NULL$	$NULL$	$NULL$	$NULL$	$NULL$
U_2	D_2	F_2	U_2	D_2	S_2	$NULL$	$NULL$
$NULL$	$NULL$	$NULL$	U_3	D_3	S_3	U_3	A_3
$NULL$	$NULL$	$NULL$	$NULL$	$NULL$	$NULL$	U_4	A_4

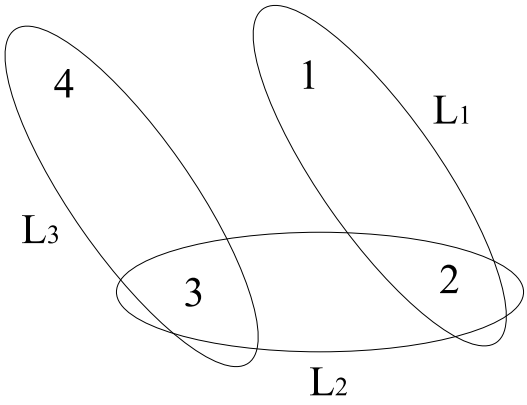


Figura 3.7: Ipergrafo senza cicli puri

3.4 L'algoritmo con l'OR

Questo algoritmo, come gli altri che seguono, è stato ideato all'interno dell'Università di Modena e Reggio Emilia. Il principio di funzionamento è molto semplice, ma nonostante ciò, l'algoritmo è in grado di calcolare la Full Disjunction di un qualsiasi insieme di relazioni avente schema anche γ-ciclico. Purtroppo, il prezzo da pagare è quello dalla pesantezza computazionale. Inoltre, ben poco si può fare per cercare ottimizzazioni, in quanto il problema risiede proprio nell'operatore OR.
Nel generico caso di n relazioni viene eseguita la seguente query:

select *
from (L_1 full join L_2 on JM(L_1,L_2))
full join L_3 on (JM(L_1,L_3) OR JM(L_2,L_3))
...
full join L_n on (JM(L_1,L_n) OR JM(L_2,L_n) OR ... OR JM(L_{n-1},L_n))

Dove JM(L_i,L_j) è una generica condizione di join fra le relazioni L_i ed L_j, quale potrebbe essere ad esempio $L_i.A_1 = L_j.A_1$. Per il momento si passi sopra a tale simbologia (adottata in ambito SEWASIE) e si consideri solamente il caso restrittivo del semplice equijoin come condizione di join. Il significato di tale notazione verrà chiarito in sezione 4.1.
Si può intuire facilmente come sia elevata la complessità computazionale di tale interrogazione: una fonte viene coinvolta anche in n join ed in totale sono presenti $\sum_{i=1}^{n-1} i$ operatori ($n-1$ join e $\sum_{i=1}^{n-2} i$ OR).
Ma non ci sono solo cattive notizie; infatti a differenza dell'algoritmo SOJO pre-

sentato in sezione 3.2.4, utilizzando l'algoritmo con l'OR è possibile definire una qualsiasi clausola di join, poiché l'operatore a cui si ricorre è il full outerjoin, non il natural outerjoin come nel caso precedente. In più, questo metodo permette l'implementazione su un qualsiasi database che supporti lo standard SQL92.
Pur utilizzando lo stesso operatore di Galindo-Legaria, è possibile compattare i risultati solo alla fine e non dopo ogni join. Grazie a questa caratteristica la Full Disjunction può essere calcolata utilizzando una sola query. Questo è possibile appunto perché nell'interrogazione non viene utilizzato esclusivamente l'operatore di outerjoin ed ogni relazione compare più di una volta.

Ben lungi dall'essere una trattazione completa, è comunque doveroso mostrare come questo algoritmo riesca a calcolare la Full Disjunction anche in presenza di cicli puri, di γ-3-cicli e di γ-cicli tramite semplici esempi.
Ammettiamo di avere le seguenti tre relazioni:

L_1

A_1	A_2
1	10
2	20

L_2

A_2	A_3
10	100
20	200

L_3

A_3	A_1
100	1
200	2

Facendo riferimento alla sezione 3.2.2 a pagina 60 si può vedere come le tre relazioni diano origine ad un ipergrafo in configurazione di ciclo puro. In particolare, stando alla definizione, è il ciclo puro più semplice possibile. La query da eseguire sarebbe la seguente:

select *
from (L_1 full join L_2 on $L_1.A_2 = L_2.A_2$)
full join L_3 on (($L_1.A_1 = L_3.A_1$) OR ($L_2.A_3 = L_3.A_3$))

Il risultato del primo full join è il seguente:

$L_1.A_1$	$L_1.A_2$	$L_2.A_2$	$L_2.A_3$
1	10	10	100
2	20	20	200

mentre il risultato finale è:

$L_1.A_1$	$L_1.A_2$	$L_2.A_2$	$L_2.A_3$	$L_3.A_3$	$L_3.A_1$
1	10	10	100	100	1
2	20	20	200	200	2

Avendo invece situazioni più complesse, come:

L_1	
A_1	A_2
1	10
2	$NULL$

L_2	
A_2	A_3
10	100
20	200

L_3	
A_3	A_1
100	$NULL$
200	2

la query produrrebbe:

$L_1.A_1$	$L_1.A_2$	$L_2.A_2$	$L_2.A_3$	$L_3.A_3$	$L_3.A_1$
1	10	10	100	100	$NULL$
2	20	$NULL$	$NULL$	200	2
$NULL$	$NULL$	$NULL$	200	200	2

che, basandosi sulla definizione di pagina 59, è proprio la Full Disjunction cercata. Quindi anche in presenza di un ciclo puro l'algoritmo funziona correttamente. Si vede come l'algoritmo SOJO fallisca in base alla definizione di ordinamento corretto data a pagina 59. Per rispettare il vincolo secondo il quale una relazione può comparire una sola volta all'interno dell'ordinamento si esclude la possibilità di avere quel join che permetterebbe di "chiudere il ciclo", quindi il risultato perde una connessione.

Veniamo ora ai γ-3-cicli; come prima, prendiamo come riferimento il caso più semplice. Stando a quanto affermato a pagina 61, la configurazione cercata corrisponde a:

L_1		
A_1	A_2	A_3
1	10	100
2	20	$NULL$
3	30	300

L_2	
A_1	A_2
1	10
2	$NULL$

L_3	
A_2	A_3
10	$NULL$
20	200
30	300

Utilizzando la query:

select *
from (L_1 full join L_2 on ($L_1.A_1 = L_2.A_1$ AND $L_1.A_2 = L_2.A_2$))
full join L_3 on (($L_2.A_2 = L_3.A_2$) OR ($L_1.A_2 = L_3.A_2$ AND $L_1.A_3 = L_3.A_3$))

si ha per risultato:

$L_1.A_1$	$L_1.A_2$	$L_1.A_3$	$L_2.A_1$	$L_2.A_2$	$L_3.A_2$	$L_3.A_3$
1	10	100	1	10	10	$NULL$
2	20	$NULL$	$NULL$	$NULL$	$NULL$	$NULL$
3	30	300	$NULL$	$NULL$	30	300
$NULL$	$NULL$	$NULL$	2	$NULL$	$NULL$	$NULL$
$NULL$	$NULL$	$NULL$	$NULL$	$NULL$	20	200

che costituisce la Full Disjunction cercata.

Per quanto riguarda i γ-cicli, dalla definizione data a pagina 61, si vede come il caso più semplice degeneri in un γ-3-ciclo. Infatti, poiché un γ-ciclo è definito come minimo per tre iperarchi, ammettiamo che questi siano L_1, L_2, L_3; inoltre ammettiamo di avere tre nodi: A_1, A_2, A_3. In particolare assumiamo che L_1 sia costituito dai nodi A_1 e A_3, L_2 da A_1 e A_2 e L_3 da A_2 e A_3. Inoltre, poiché la proprietà fondamentale che differenzia i γ-cicli dai cicli puri è la possibilità di fare apparire un generico attributo in più di due iperarchi, facciamo assumere a A_2 questo ruolo. Così, includendo nell'iperarco L_1 anche il nodo A_2, abbiamo ricostruito la situazione precedente. Le immagini 3.8 e 3.9 mostrano intuitivamente quanto descritto.

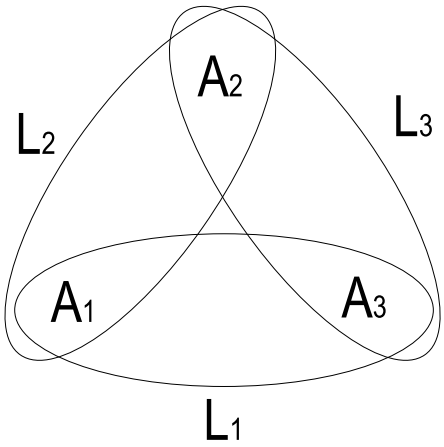


Figura 3.8: Situazione iniziale

Con gli esempi precedenti si è mostrato come questo algoritmo riesca a risolvere problemi causati da ipergrafi in configurazione di γ-ciclo, γ-3-ciclo e ciclo puro. Questa è ben lontana dall'essere una dimostrazione rigorosa, poiché bisognerebbe mostrare che la Full Disjunction viene calcolata correttamente anche in caso di ipergrafi che non sono costituiti unicamente dalle configurazioni prese in esame.
Inoltre va ribadito che tali Full Disjunction sono state ottenute utilizzando una sola query, sebbene molto pesante.

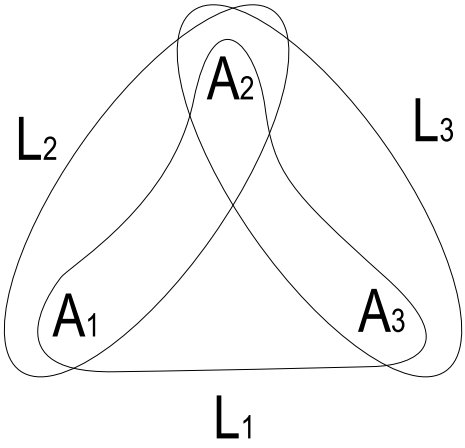


Figura 3.9: Situazione dopo l'inserimento di A_2 in L_1

3.5 L'algoritmo di Kostenarov

Questo algoritmo dovrebbe prendere il posto, quando ultimato, di quello con l'OR nell'ambito del progetto SEWASIE. Come precedentemente affermato, questo algoritmo vuole realizzare l'operazione di full-disjunction anche nel caso di γ-*ciclicità* dello schema del database.
Idealmente si vuole generare una pseudo-sequenza di outerjoin (PSOJ) in grado di essere implementata in un qualsiasi DBMS che supporti lo standard SQL92.

3.5.1 Algoritmo per la generazione della pseudo-sequenza di outerjoin

Il seguente esempio ha lo scopo di introdurre l'algoritmo in modo il più intuitivo possibile.

Si tenga in considerazione che il seguente esempio parte dal presupposto che i risultati delle esecuzioni sulle sorgenti siano già presenti e che quindi ci si concentri solo sulla generazione del risultato globale della query.

Si consideri la classe globale G rappresentativa di quattro classi locali (relazionali) R_1, R_2, R_3, R_4 con i seguenti schemi: R_1 = (AEF), R_2 = (ABC), R_3 = (CDE) ed R_4 = (ACE). Ognuna di queste classi comprende le istanze presenti nelle seguenti tabelle:

R_1		
A	**F**	**E**
1	1	3
4	2	6
10	3	12
18	4	20

R_2		
A	**B**	**C**
1	2	7
4	5	1
7	11	4
13	19	10

R_3		
C	**D**	**E**
1	7	3
7	9	6
13	11	8
14	16	14

R_4		
A	**C**	**E**
4	1	8
7	4	3
10	14	20
1	19	7

Si consideri di avere una tabella di join JT che esprima le condizioni di join come natural join, ossia tramite gli attributi comuni fra gli schemi delle relazioni. Più precisamente, nel nostro caso:

1. $F_{12} \Longrightarrow R_1$.A = R_2.A
2. $F_{23} \Longrightarrow R_2$.C = R_3.C
3. $F_{13} \Longrightarrow R_1$.E = R_3.E
4. $F_{14} \Longrightarrow R_1$.AE = R_4.AE
5. $F_{24} \Longrightarrow R_2$.AC = R_4.AC
6. $F_{34} \Longrightarrow R_3$.CE = R_4.CE

Seguendo la definizione di Full Disjunction , il risultato che si deve ottenere è:

A	**B**	**C**	**D**	**E**	**F**
1	2	7	-	3	1
4	5	1	-	6	2
10	-	-	-	12	3
18	-	-	-	20	4
7	11	4	-	3	-
13	19	10	-	-	-
-	-	13	11	8	-
10	-	14	-	20	-
1	-	19	-	7	-
4	5	1	-	8	-
1	2	7	9	6	-
4	5	1	7	3	-
7	11	4	16	14	-

Come si può osservare, ricorrendo alle definizioni sulle differenti ciclicità, il grafo in Figura 3.10 è γ-*ciclico*, quindi non è possibile utilizzare i risultati di Galindo-Legaria o Ullman per calcolare la Full Disjunction delle relazioni in esame. Tale grafo ha come nodi gli attributi delle relazioni R_i e come archi le relazioni stesse e viene costruito nel seguente modo:

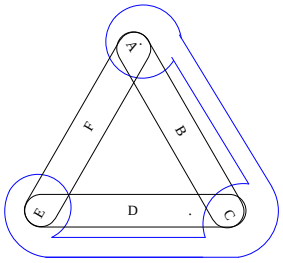


Figura 3.10: Grafo γ-ciclico

- Per ogni relazione R_i si crea un iperarco che contiene tutti gli attributi (rappresentanti i nodi) appartenenti alla relazione stessa.

- Per ogni $F_{k,q}$ del tipo R_k.A = R_q.A viene generato un solo nodo A, incluso sia in R_k che in R_q.

- Per ogni F_k del tipo $F_k(A, B, C)$, dove A,B e C sono degli attributi globali per cui esiste una Join Table, vengono create tante relazioni quanti sono le classi coinvolte nella Join Table e per ognuna vengono definiti i predicati di outerjoin corrispondenti estrapolando l'informazione neccessaria dalla Join Table.

- L'ipergrafo ottenuto ripetendo i due passi precedenti è completo e spesso sovrabbondante di informazioni, perciò viene ridotto eliminando gli iperarchi che sono propriamente inclusi in un qualsiasi altro iperarco che fa parte dell'ipergrafo.

Per illustrare il funzionamento dell'algoritmo che calcola la Full Disjunction si farà riferimento al diagramma in Figura 3.11. Tale diagramma viene costruito a partire dal grafo di Figura 3.10 ed ha come nodi le relazioni (ossia gli iperarchi del grafo) mentre i lati rappresentano le condizioni di outerjoin $F_{i,j}$. Nella rappresentazione supponiamo che eventuali predicati di join indiretto siano stati già trasformati durante la fase di integrazione. Ciò vuol dire che per ogni due classi S_1 e S_2 (a cui corrispondono due relazioni R_1 e R_2) a cui andrebbe applicato un predicato di join indiretto (rappresentato da una JoinTable $F_j(R_j)$) sono state create due relazioni indotte R_1' e R_2" che sono il risultato del join naturale tra ognuna delle relazioni interessate R_1 e R_2 e la relazione di "passaggio" R_j.

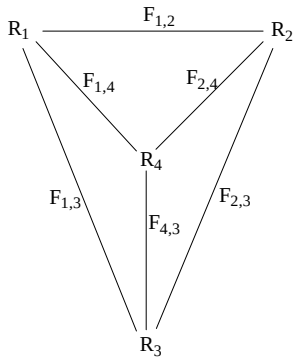


Figura 3.11: Diagramma rappresentativo

- si comincia l'elaborazione del diagramma scegliendo un nodo di partenza, per esempio la relazione R_2 (questa operazione sarà discussa in dettaglio più avanti);

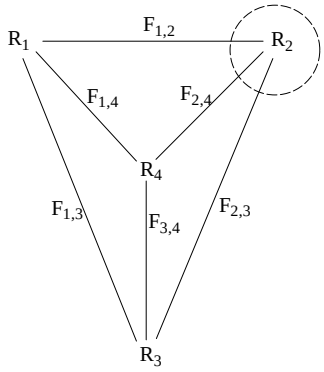


Figura 3.12: Diagramma rappresentativo alla prima iterazione

- si genera il taglio che comprende solo il nodo R_2 e si verifica quali sono i lati intersecati dal taglio. Nel nostro caso tali lati risultano ($R_2 \longleftrightarrow R_1$),($R_2 \longleftrightarrow R_4$) e ($R_2 \longleftrightarrow R_3$) (vedi Figura 3.12);
- si procede eliminando i lati intersecati effettuando una operazione di inglobamento del nodo (relazione) selezionato verso i nodi (relazioni) a cui è

collegato effettuando anche l'operazione di natural outerjoin espressa dai lati intersecati. Il risultato di ogni operazione di NOJ è sempre una relazione. Tali relazioni formano i nuovi nodi del diagramma rappresentato in Figura 3.13.

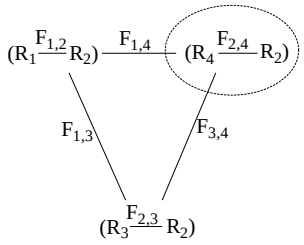


Figura 3.13: Diagramma rappresentativo alla seconda iterazione

- a partire dalla Figura 3.13 si genera il secondo taglio su un nodo arbitrario, per esempio ($R_2 \longleftrightarrow R_4$) e si ripetono le operazioni descritti sopra.

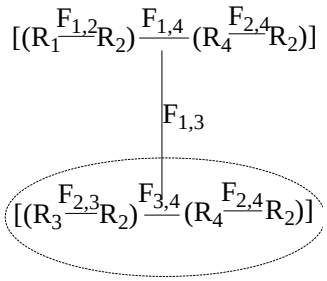


Figura 3.14: Diagramma rappresentativo alla terza iterazione

- in riferimento alla Figura 3.14 si genera il terzo taglio su ($R_2 \longleftrightarrow R_4$)$\longleftrightarrow$($R_2 \longleftrightarrow R_3$), si esegue l'operazione di inglobamento e alla fine si ottiene l'espres-

sione di Figura 3.15

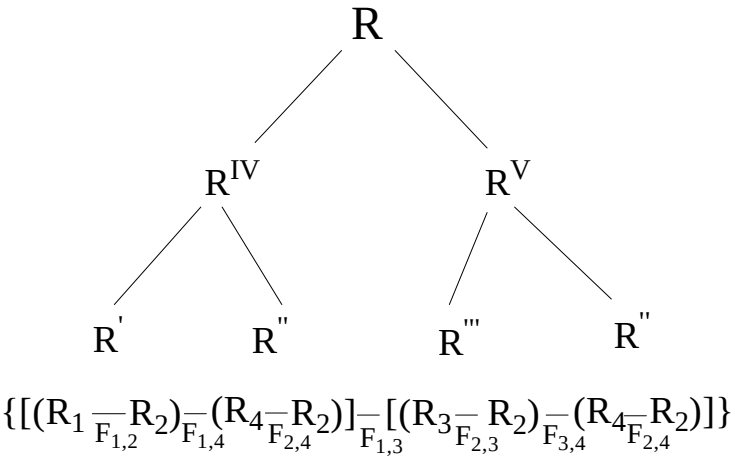


Figura 3.15: Diagramma rappresentativo della quarta iterazione

Riassumendo si ottiene la peudo-sequenza di join:

$$\{[(R_1 =\bowtie_{F_{1,2}}= R_2) =\bowtie_{F_{1,4}}= (R_4 =\bowtie_{F_{2,4}}= R_2)] =\bowtie_{F_{1,3}}= [(R_3 =\bowtie_{F_{2,3}}= R_2) =\bowtie_{F_{3,4}}= (R_4 =\bowtie_{F_{2,4}}= R_2)]\}$$

A questo punto presentiamo l'algoritmo in modo più formale . Per l'applicazione dell'algoritmo PSOJ abbiamo bisogno delle seguenti informazioni:

- Schema globale **S($\mathcal{G}$)**, Mapping Table, Join Table.
- L'insieme delle relazioni locali $\mathcal{R} = \{R_1, \ldots, R_n\}$ trascritti sullo schema globale **S($\mathcal{G}$)**.
- $\mathcal{F} = \{F_{i,j} \mid i < j, i = 1 \ldots n-1; j = 2 \ldots n\}$

Si noti che nel nostro caso i diagrammi indotti sono sempre completamente connessi. Questo fatto è ottenuto ponendo alcuni $F_{i,j} = false$. Ciò significa che non c'è intersezione tra le calssi L_i ed L_j quindi in questo caso il full outerjoin si riconduce al "prodotto cartesiano" con un successivo padding delle tuple con valori nulli per gli attributi disgiunti. Cioè le tuple del risultato sono del tipo

$R_i \rightarrow \{R_i.\star, \underbrace{null, \ldots, null}_{n}\}$ e $R_j \rightarrow \{\underbrace{null, \ldots, null}_{m}, R_j.\star\}$ se la cardinalità delle relazioni R_i ed R_j è rispettivamente n ed m.
Presentiamo l'algoritmo PSOJ in pseudo-codifica in Tabella 3.1.

Calcolo della Full Disjunction

Input: * $\mathcal{R} = \{R_1, \ldots, R_n\}$: relazioni definite sullo schema globale.
* $\mathcal{F} = \{F_{i,j}\}$: Join Table per tutte le relazioni di $\mathcal{R}$.
Output: pseudo-sequenza per calcolare la Full Disjunction
Procedimento:
• Poni $F_0 = \mathcal{F}$, $R_0 = \mathcal{R}$ ed $NP = (n-1)$ numero dei passi.
• for ($i = 0; i < NP; i++$)
  • seleziona $R_i \in \mathcal{R}_i$
    $\mathcal{R}_{i+1} = \{R_i =\bowtie_f= R'_i \mid R'_i \in \mathcal{R}_i; R'_i \neq R_i; f = OJC_i(R_i, R'_i)$ non è null$\}$
• Restituisci $\mathcal{R}_{NP}$.

Tabella 3.1: Algoritmo PSOJ

La funzione di outerjoin Condition ($OJC_i(R_i, R'_i)$) restituisce la JC da applicare tra R_i ed R'_i sulla base delle relazioni usate per definire le R_i ed R'_i ed è definita nel seguente modo:

$$OJC_i(R_i, R'_i) = \begin{cases} JC(R_0, R'_0) \in \mathcal{F} & \text{se } i = 0 \\ OJC_{i-1}(R^a_{i-1}, R^b_{i-1}) & \text{tale per cui} \\ & \exists R^c_{i-1} \in \mathcal{R}_{i-1} \mid \\ & R_i = R^c_{i-1} =\bowtie= R^a_{i-1} \\ & R'_i = R^c_{i-1} =\bowtie= R^b_{i-1} \end{cases} \quad (3.1)$$

Verifichiamo l'algoritmo sul nostro esempio di riferimento:
$\mathcal{R}_0 = \{R_1, R_2, R_3, R_4\}$
al passo $i = 0$: selezioniamo $R_2 \in \mathcal{R}_0$
otteniamo: $\mathcal{R}_1 = \{R^a_1 = R_4 =\bowtie_{F_{4,2}}= R_2$
$R^b_1 = R_3 =\bowtie_{F_{3,2}}= R_2$
$R^c_1 = R_1 =\bowtie_{F_{1,2}}= R_2\}$
al passo $i = 1$: selezionaiamo $R^a_1 \in \mathcal{R}_1$
otteniamo: $\mathcal{R}_2 = \{R^a_2 = R^c_1 =\bowtie_{F_{1,4}}= R^a_1$
$R^b_2 = R^b_1 =\bowtie_{F_{3,4}}= R^a_1\}$
al passo $i = 2$: selezionaiamo $R^b_1 \in \mathcal{R}_2$

otteniamo: $\mathcal{R}_3 = \{R_3^a = R_2^b =\bowtie_{F_{1,3}}= R_2^a\}$

Mostriamo anche il calcolo delle OJC. Per esempio al passo $i = 2$ abbiamo:
$OJC_2(R_2^b, R_2^a) = OJC_1(R_1^c, R_1^b) = OJC_0(R_3, R_1) = F_{1,3}$

Si noti che l'algoritmo, ad ogni iterazione opera su coppie di relazioni riducendo il numero dei lati del grafo di un numero pari al numero di lati interseccati dal arco generato. Dunque l'algoritmo si arresta quando rimane soltanto un nodo. Questo nodo in pratica rappresenta la pseudo-sequenza da noi cercata.

Si noti inoltre che dalla pseudo sequenza generata si può osservare che gli operatori di NOJ sono applicati a delle coppie di relazioni e che alcune coppie si ripetono. D'altro canto alcune operazioni di join possono essere eseguite in parallelo senza alterare il risultato finale poiché sono del tutto indipendenti. Dunque se noi sostituissimo al risultato di ogni operazione di join una relazione indotta essa andrà in join con un'altra operazione indotta e cosi via fino alla creazione di un albero, la radice del quale rappresenterà la risposta della query. L'albero risultante è un albero binario perfettamente bilanciato. Dunque le operazioni di join che si trovano sulle foglie possono essere eseguite in parallelo, i risultati delle quali diventeranno foglie nell'albero ridotto.

Infine, l'operatore di natural outerjoin è implementato nel SQL92 e per adesso ci appogiamo sugli algoritmi di esecuzione implementati nelle diverse DBMS commerciali.

Teorema 3.5.1 (Validità dell'algoritmo PSOJ.) *Dato un insieme R=$\{R_1, \ldots, R_n\}$ di relazioni definite sullo schema globale **S($\mathcal{G}$)** e dati P = $\{P_1, \ldots, P_m\}$ predicati di ugualianza del tipo A=B oppure $R_i.A = R_j.B$ e/o predicati di selezzione $\mathcal{P}$ = $\{\mathcal{P}_i, \ldots, \mathcal{P}_k\}$ definiti da altretanti Join Table, allora: L'algoritmo PSOJ genera una pseudo-sequenza di operatori di natural outerjoin che produce la completa disgiunzione(FD).*

Dimostrazione. Dimostriamo il teorema per induzione.

a) Definiamo il caso base cosi:
Date due relazioni R' e R'' che formano un diagramma con un solo la-

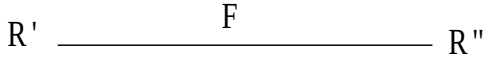


Figura 3.16: Caso base.

to(Figura 3.16), applicando l'algoritmo PSOJ il diagramma si trasforma in

due relazioni collegate da un operatore di natural outerjoin. Per la definizione del natural outerjoin questo produce la full disjunction $FD(R^{'}, R^{''})$. Dunque nel caso n = 1 lato l'algoritmo è valido.

b) Supponiamo che date R = $\{R_1, \ldots, R_{k-1}\}$ relazioni sullo schema globale **S($\mathcal{G}$)** con i rispettivi predicati di join ed il corrispondente diagrama connesso, l'algoritmo PSOJ genera la $FD(R_1, \ldots, R_{k-1})$.
Dobbiamo dimostrare che l'algoritmo è valido per R = $\{R_1, \ldots, R_{k-1}\} \bigcup \{R_k\}$, cioè per un diagramma di k lati.

c) Secondo la costruzione dell'algoritmo dati n nodi il massimo numero di lati che gli collegano(denotato con m) si hanno quando il grafo generato a partire dagli n nodi è completamente connesso ed è $m_{max} = \frac{n(n-1)}{2}$, mentre il minimo numero si ha quando il grafo è connesso, ma aciclico ed è $m_{min} = (n-1)$. In ogni caso ad ogni passo viene generato un taglio che include al suo interno solo un nodo e dunque dal grafo viene eliminato solo questo nodo. Da questo segue che ad ogni passo il numero di nodi diminuisce di uno. Dunque al k-esimo passo si ha $m_{max} = \frac{(n-k)(n-(k-1))}{2}$ e $m_{min} = (n-(k-1))$.
D'altro canto per costruzione ogni lato interseccato dal arco generato viene sostituito dall'operatore di natural outerjoin tra due relazioni e ciò come abbiamo detto genera la FD tra questa coppia di relazioni. Dunque ad ogni passo viene applicato il caso base.
Allora supponiamo di aggiungere un'altra relazione R_n e di voler calcolare la FD($\{R_1, \ldots, R_{n-1}\} \bigcup \{R_n\}$). Per quanto detto sopra questa modifica non avrà nessuna influenza sul procedimento di applicazione dell'algoritmo, poiché ad ogni passo noi trattiamo solo due relazioni alla volta. Dunque l'algoritmo PSOJ genera la FD di un numero arbitrario di relazioni.

□

Il Teorema precedente afferma che per un numero arbitrario di nodi l'algoritmo PSOJ genera la FD a partire dal grafo connesso che rappresenta le relazioni coinvolte e le connessioni tra di loro. Se il grafo corrispondente alle relazioni risultasse non connesso l'algoritmo deve essere applicato a tutte le parti connesse.

Corollario 1 *L'algoritmo PSOJ genera la FD per un qualsiasi grafo connesso.*

Facciamo alcune considerazioni nel caso in quale il grafo corrispondente alle relazioni in esame non risulta completamente connesso. Questo può essere utile quando vogliamo applicare la nostra tecnica per il calcolo della Full Disjunction in qualche altro contesto. In particolare possiamo dire che si possono verificare due casi generali:

a) Il diagramma è completamente connesso (questo è il caso generale nel nostro contesto);

b) Il grafo non è completamente connesso.

Nel primo caso possiamo affermare che la scelta del nodo è completamente arbitraria, mentre nel secondo bisogna scegliere un nodo opportuno. Più pecisamente il taglio che dobbiamo individuare è quello di costo minore.
Definiamo come costo di un taglio il numero di archi appartenenti al diagramma che il taglio stesso interseca. Il numero dei passi per ottenere la pseudo-sequenza di NOJ è proporzionale al numero degli archi del diagramma. Dunque se ad ogni passo scegliamo il taglio che interseca il maggior numero di archi, cioè il taglio di costo maggiore, avremo il minor numero di iterazioni. Nel caso in cui il diagramma risultasse completamente connesso tutti nodi sono collegati con tutti gli altri allora il costo di tutti i tagli è lo stesso. Proprio per questo fatto in questo caso la scelta del nodo di partenza è arbitraria. Nel secondo caso invece quando il diagramma risulta non completamente connesso non possiamo seguire lo stesso procedimento perché si rischia di staccare parti del grafo dividendolo in due parti non connessi e dunque l'algoritmo di generazione della PSOJ si interrompe prima di determinare la pseudo-sequenza. Per ovviare questo problema basta procedere nel modo opposto del primo caso, cioè ad ogni passo si sceglie il taglio di costo minore. L'algoritmo per determinare il taglio di costo minimo è rappresentato in Tabella 3.2.

In questo caso il costo dell'algoritmo aumenta, ma comunque in tutti i due casi il costo totale di esecuzione dell'algoritmo PSOJ è minore dal costo totale del metodo di calcolo della Full Disjunction proposto da Galindo-Legaria nella sezione precedente.

Il discorso sulla scelta del nodo di partenza o più in generale sul nodo su cui effettuare il taglio può ritornare utile anche nel nostro contesto.
Infatti il nostro grafo è sempre completamente connesso, ma vi possono essere degli archi la cui condizione è "$false''$. È intuibile che un full outerjoin su una condizione "$false''$ restituisce un numero massimo di tuple pari alla cardinalità $\mid R_1 \mid \times \mid R_2 \mid$ quindi è ovvio "ritardare" queste operazioni il più possibile in modo da realizzarle su insiemi più piccoli di tuple.
Intuitivamente questo si può ottenere selezionando un nodo che ha un minor numero di OJC "$false''$. Per questo scopo è applicabile l'algoritmo presentato in Tabella 3.2 con una piccola modifica: nella condizione di controllo su $F_{i,j}$ basta sostituire $null$ con $false$.

Input: * $\mathcal{R} = \{R_1, \ldots, R_i, \ldots, R_n\}$: relazioni definiti sullo schema globale.
* $\mathcal{F} = \{F_{i,j}\}, i = 1, \ldots, n; j = 1, \ldots, m$: Join Table per tutte le relazioni di $\mathcal{R}$.
* Diagramma D indotto.

Output: nodo di costo minimo.

Procedimento:

• Poni $old = \infty$, $R_s = \emptyset$, $i = 0$
• While $\mathcal{R} \neq \emptyset$
  • seleziona $R_i \in \mathcal{R}$
  poni $\mathcal{R} = \mathcal{R} - \{R_i\}$
  $new = 0$
    for $j = 0$ to m do
      if $F_{i,j} \neq null$ then
        $new = new + 1$
      endif
    endfor
    if $new < old$ then
      $R_s = R_i$
      $old = new$
    endif
 endwhile
• Restituisci R_s che rappresenta la relazione a cui corrisponde il nodo su cui effettuare il taglio di costo minimo.

Tabella 3.2: Algoritmo per il calcolo del taglio di costo minimo.

3.5.2 Critiche all'algoritmo

Seguendo la dimostrazione sembrerebbe che l'algoritmo funzioni correttamente. Ammettendo per il momento che ciò sia vero, soffermiamoci inizialmente sulla sua implementazione su di un DBMS relazionale, che presenta dei problemi e delle imprecisioni.

- Prima di tutto è bene soffermare la propria attenzione sull'operatore utilizzato: il natural outerjoin. Come già affermato in precedenza, vi sono numerosi svantaggi nell'utilizzo di tale operatore:
  - per prima cosa, la condizione di join è da ritenersi fissata; se in alcuni casi questa restrizione può essere accettata, così non è nel nostro ambito. L'ambiente su cui si basa il sistema SEWASIE si basa su condizioni di join che in molti casi possono essere diverse da quella che confronta tutti gli attributi comuni.
  - l'uso di tale operatore nel nostro ambito rende inutile la join table, il cui scopo dovrebbe essere quello di indicare quali attributi devono essere presi in considerazione per il join ed in che modo devono essere confrontati.
  - si afferma che il natural outerjoin è uno compatibile con SQL92, ma così non è, dato che secondo quello standard la clausola "natural" esclude l'utilizzo della clausola "outer". Tale restrizione è stata rimossa dallo standard SQL99.
  - questo approccio, fondendo automaticamente gli attributi con lo stesso nome, non rende possibile la risoluzione dei conflitti. Ciò è accettabile sotto l'ipotesi di omogeneità semantica, ma diventa inadeguato non appena tale vincolo viene rimosso e, quindi, nella realtà.

- l'autore afferma che un ordinamento non è necessario nel caso di grafo completamente connesso, come nel nostro ambito sempre succede. Purtroppo così non è, dato che nel caso di fonti vuote (anche in assenza di γ-ciclicità) un ordinamento è necessario, così come affermava Ullman in [34]. Per mostare l'errore verrà proprio utilizzato l'esempio base che appare in quello studio.
  Supponiamo di avere le seguenti relazioni:

L_1

U	D	F
U_1	D_1	F_1

L_2

U	D	S

L_3

U	A
U_1	A_1

Il grafo corrispondente è riportato in figura 3.17 mentre l'ipergrafo corrispondente (non γ-ciclico), è riportato in figura 3.1.

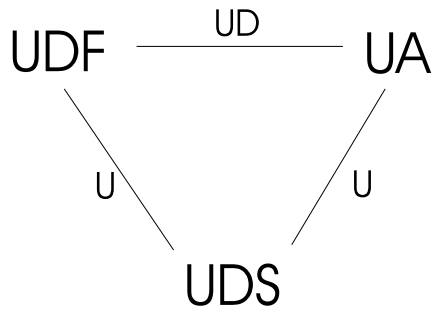


Figura 3.17: Grafo di esempio

Come si può notare, la relazione UDS è vuota. Adesso applichiamo l'algoritmo PSOJ, prendendo come nodo iniziale la relazione L_1. Dopo il primo passo risultano due relazioni che per semplicità chiameremo L_{12} ed L_{13}:

L_{12}

U	D	F	S
U_1	D_1	F_1	$NULL$

L_{13}

U	D	F	A
U_1	D_1	F_1	A_1

Dopo il secondo passo si avrà chiaramente un'unica relazione con schema $UDFSA$ e contenente la sola tupla $U_1D_1F_1\perp A_1$, che effettivamente corrisponde alla full disjunction.
Proviamo invece a prendere come nodo iniziale la relazione L_3. Si ha un risultato parziale di:

L_{23}

U	D	S	A
U_1	$NULL$	$NULL$	A_1

L_{13}

U	D	F	A
U_1	D_1	F_1	A_1

a questo punto si vede che eseguendo un join su queste due relazioni si ottiene una relazione con schema $UDFSA$ contenente due tuple: $U_1\perp\perp\perp A_1$ e $U_1D_1F_1\perp A_1$. Nonostante la seconda tupla contenga le stesse informazioni del caso precedente, la presenza di una tupla sussunta (la prima) non permette di parlare di full disjunction.
Quindi, in realtà, un ordinamento è necessario. In linea di principio una possible soluzione sarebbe quella di escludere tutte le sorgenti vuote e di eseguire poi l'algoritmo solo su quelle significative.

- il problema delle fonti vuote rende inutile anche l'approccio proposto per i grafi non completamente connessi. Se infatti si verificasse una situazione come quella in figura 3.18 e se il nodo indicato con la lettera A rappresentasse una fonte vuota, allora non si potrebbe procedere come nel caso precedente, ossia escludendo la fonte, dato che in questo caso il grafo ottenuto dall'eliminazione della sorgente si presenterebbe disconnesso.

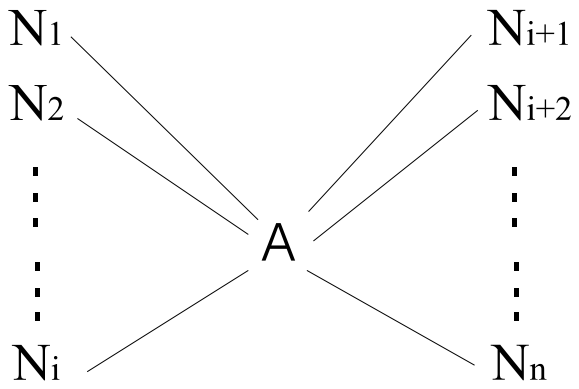


Figura 3.18: Grafo non completamente connesso

- per quanto riguarda i grafi non completamente connessi si dice che è necessario prendere il nodo avente il taglio di costo minore (ossia un numero minore di connessioni) per evitare una disconnessione. Questo è vero solamente nel caso banale di nodi con una sola connessione, mentre in tutti gli altri casi il ragionamento non è corretto. Supponiamo di avere un grafo come quello in figura 3.19, dove un singolo nodo funge da ponte tra due grafi completamente connessi: seguendo l'approccio indicato da Konstantin il nodo da scegliere dovrebbe essere L_x, in quanto ha solo due connessioni, mentre tutti gli altri nodi ne hanno n-1 o m-1 (a parte i due nodi collegati anche con L_x, che hanno una connessione in più). Con questa scelta però si ottiene una disconnessione. Quindi il criterio indicato è errato: in realtà non ci si può basare unicamente sul numero di connessioni per evitare la disconnessione. Infatti, guardando alla figura 3.20, ottenuta dalla figura 3.19 con m=n=3 si nota come gli unici nodi che evitano problemi siano quelli indicati con L_y, aventi due connessioni esattamente come L_x. Quindi dal punto di vista delle connessioni L_x e L_y sono equivalenti, anche se danno risultati diversi.

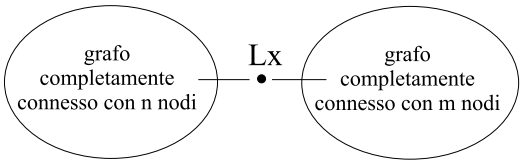


Figura 3.19: esempio di disconnessione

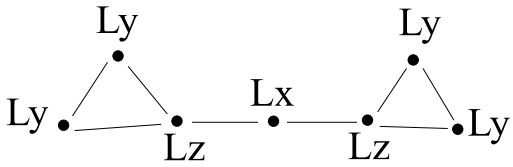


Figura 3.20: altro esempio di disconnessione

Per ottenere un risultato affidabile è necessario, una volta scelto il nodo da eliminare, che esista un percorso che colleghi tutti i nodi a lui connessi. Per esempio, facendo riferimento alla figura 3.20:

- scegliendo il nodo L_x non si riesce a trovare un percorso che riesca a congiungere i nodi a lui connessi (ossia gli L_z), quindi si otterrebbe una disconnessione.
- scegliendo uno dei nodi L_z non si riesce a trovare un percorso che riesca ad unire i nodi a lui connessi (ossia due L_y e L_x), quindi si otterrebbe una disconnessione.
- scegliendo uno dei nodi L_y si riesce sempre a trovare un percorso che unisca i nodi a lui connessi (ossia un L_z ed un L_y) e quindi il grafo rimane connesso.

Questa proprietà vale anche per il caso di nodo con un'unica connessione, in quanto in questo caso è sempre possibile trovare un percorso che degenera nell'unico nodo di connessione. Ciò comunque non significa che i nodi con una sola connessione debbano per forza essere trattati in maniera prioritaria. Ai fini dell'algoritmo potrebbe essere più conveniente trattarli in diversi momenti.
Nel nostro ambito è possibile affrontare questo problema utilizzando i valori memorizzati nella join table. Formalmente:
Dato un grafo G ed un nodo $L_i \in G$, l'eliminazione del nodo L_i non disconnette G se e solo se:

$$\forall L_j \mid F_{ij} \neq NULL \;\; \exists L_k \mid F_{jk} \neq NULL, F_{ik} \neq NULL \quad \text{con } i \neq j \neq k$$

dove F_{ij} indica la condizione di join tra la fonti L_i ed L_j.

- per ultimo è necessario criticare l'approccio che segue l'autore nel caso di un grafo completamente connesso, quando però sono presenti delle condizioni di join "false". Nel nostro ambito una condizione "false" non ha lo stesso significato che una condizione mancante.

  – Se una condizione di join manca, si intende che pur non potendo legare direttamente le istanze delle due classi, è possibile che tali istanze possano essere collegate tramite join con altre relazioni. Ad esempio, due classi con schema AB e CD non hanno attributi in comune e quindi non hanno possibilità di essere collegate, ma se è presente una relazione con schema BC a fare da ponte, allora le istanze possono essere collegate.

  – Se una condizione di join è "false" si intende che le classi non contengono istanze degli stessi oggetti. Questa assunzione implica una conoscenza estensionale delle fonti e dei dati in esse contenuti.

Nei casi in cui sia presente una condizione "false" viene consigliato di occuparsi di tali condizioni il più tardi possibile, per calcolare il prodotto cartesiano su un numero di connessioni il più basso possibile.
Vediamo cosa succederebbe volendo realizzare la full disjunction delle due seguenti relazioni.

L_1

A	B
A_1	B_1
A_2	B_2

L_2

C	D
C_3	D_3
C_4	D_4

Calcoliamo la full disjunction sia seguendo la definizione classica sia applicando l'algoritmo di Kostenarov.

FD classica: $L_1 \oplus L_2$

A	B	C	D
A_1	B_1	$\perp$	$\perp$
A_2	B_2	$\perp$	$\perp$
$\perp$	$\perp$	C_3	D_3
$\perp$	$\perp$	C_4	D_4

FD secondo Kostenarov: $L_1 \Rightarrow\!\!\!\!\Leftarrow L_2$

A	B	C	D
A_1	B_1	C_3	D_3
A_1	B_1	C_4	D_4
A_2	B_2	C_3	D_3
A_2	B_2	C_4	D_4

L'errore in questo caso sta nell'utilizzo dell'operatore di outerjoin: la produzione di un prodotto cartesiano porta infatti alla creazione di connessioni in realtà non esistenti e quindi ad una full disjunction errata. In questo caso l'utilizzo di un operatore di unione è più adeguato. In particolare si consiglia l'uso dell'outerunion ($\uplus$): questo operatore prima proietta entrambe le

relazioni sull'unione degli schemi, eventualmente riempiendo gli attributi non comuni con valori nulli, poi esegue l'unione; in questo modo si ottiene la vera full disjunction.
Per evitare l'introduzione di un ulteriore operatore nell'algoritmo d'ora in avanti una connessione etichettata con "false" verrà considerata come assente; quindi un grafo completamente connesso per essere tale dovrà servirsi solamente di condizioni di join significative.

Da quanto affermato si può desumere che l'algoritmo, oltre a presentare alcuni problemi in determinati casi, non è implementabile in un DBMS commerciale; quindi è necessario trovare una soluzione alternativa, a partire dal tipo di operatore utilizzato. A questo scopo vediamo cosa succede nella pratica, ossia facendo ricorso al linguaggio SQL, se si tenta di sostituire all'operatore di natural outerjoin quello di full outerjoin.
Ammettiamo di avere le sorgenti locali L_1, L_2, L_3 (si noti come il numero delle sorgenti sia il minimo per il quale abbia senso parlare di full disjunction) aventi internamente la seguente struttura:

L_1		L_2	L_3	
a	**b**	**b**	**a**	**b**
1	1000	1000	1	1000
2	2000	2000	3	3000
3	3000	6000	6	6000
4	4000	5000	7	7000

Come si può desumere dai valori delle tuple, è stata fatta l'ipotesi di omogeneità semantica.
La seguente query SQL mostra il risultato della full disjunction su L_1, L_2, L_3 secondo l'algoritmo che utilizza l'OR che, come già detto, è testato e funzionante.

select
L_1.a as L1a, L_1.b as L1b, L_2.a as L2a, L_3.a as L3a, L_3.b as L3b
from (L_3 full join L_1 on (L_3.a = L_1.a))
full join L_2 on (L_1.b = L_2.b or L_3.b = L_2.b)

La tabella 3.3 mostra i risultati della query. Come si può rapidamente verificare, data la semplicità dell'esempio, si ottiene realmente la full disjunction.

Ovviamente i risultati qui esposti, prima di essere presentati all'utente finale, dovranno essere sottoposti ad un'altra funzione, in grado di fondere le colonne rappresentanti attributi locali che sono mappate sullo stesso attributo globale. Una possibile soluzione è quella di utilizzare una funzione di risoluzione come quelle

L1a	L1b	L2a	L3a	L3b
NULL	NULL	NULL	7	7000
NULL	NULL	5000	NULL	NULL
NULL	NULL	6000	6	6000
1	1000	1000	1	1000
2	2000	2000	NULL	NULL
3	3000	NULL	3	3000
4	4000	NULL	NULL	NULL

Tabella 3.3: Risultato esatto della query

a	b
1	1000
2	2000
3	3000
4	4000
NULL	5000
6	6000
7	7000

Tabella 3.4: Risultato esatto compattato

in definizione 2.3.1 e 2.3.2 introdotte nel capitolo 2. Il risultato finale sarà del tipo riportato in tabella 3.4.

Tornando all'esempio, passiamo ora all'applicazione dell'algoritmo di Kostenarov utilizzando però il full outerjoin. Idealmente la query SQL che realizza il risultato dovrebbe essere questa:

select
L_1.a as L1a, L_1.b as L1b, L_2.b as L2b, L_3.a as L3a, L_3.b as L3b
from L_3 full join L_1 on (L_3.a = L_1.a))
full join
(L_2 full join L_3 on (L_2.b = L_3.b))
ON (L_1.b = L_2.b)

A prima vista la query riportata sembra corretta; in effetti è così, ma solo nella teoria. Provando infatti a fare eseguire la query ad un DBMS che supporti lo standard SQL92 ci si accorge di un problema: la query non viene eseguita e viene

L1a	L1b	L2a	L3aX	L3bX	L3aY	L3bY
NULL	NULL	NULL	NULL	NULL	3	3000
NULL	NULL	NULL	NULL	NULL	7	7000
NULL	NULL	NULL	7	7000	NULL	NULL
NULL	NULL	NULL	6	6000	NULL	NULL
NULL	NULL	5000	NULL	NULL	NULL	NULL
NULL	NULL	6000	NULL	NULL	6	6000
1	1000	1000	1	1000	1	1000
2	2000	2000	NULL	NULL	NULL	NULL
3	3000	NULL	3	3000	NULL	NULL
4	4000	NULL	NULL	NULL	NULL	NULL

Tabella 3.5: Risultati errati

restituito un errore, causato dalla presenza in due outerjoin della stessa tabella. Per questo motivo, il sistema non sa come comportarsi quando deve riportare in output le colonne L_3.a e L_3.b. A quale delle due tabelle L_3 deve fare riferimento? Quella in outerjoin con L_1 o con L_2?

Si potrebbe pensare di aggirare il problema assegnando degli *alias* alle due tabelle che causano l'errore e producendo tutto in output. Continuiamo quindi l'esempio adottando questa soluzione. La query verrà quindi riscritta nel seguente modo:

select
L_1.a as L1a, L_1.b as L1b, L_2.b as L2b, **X**.a as L3aX, **X**.b as L3bX, **Y**.a as L3aY, **Y**.b as L3bY
from L_3 **X** full join L_1 on (**X**.a = L_1.a))
full join
(L_2 full join L_3 **Y** on (L_2.b = **Y**.b))
ON (L_1.b = L_2.b)

Il risultato prodotto è riportato nella tabella 3.5.

Come si può facilmente notare, il risultato differisce da quello riportato nella tabella 3.3 e quindi è errato. A questo punto, per ricondursi al risultato esatto, è necessario fondere le colonne di X e Y ed eliminare le tuple sussunte. Quindi, per fare funzionare correttamente l'algoritmo, bisognerebbe introdurre un ulteriore outerjoin fra le colonne X ed Y. Per fare ciò è possibile modificare così la query:

select
L_1.a as L1a, L_1.b as L1b, L_2.b as L2b, X.a as L3aX, X.b as L3bX, Y.a as L3aY, Y.b as L3bY
from L_3 X full join L_1 on (X.a = L_1.a))
full join
(L_2 full join L_3 Y on (L_2.b = Y.b))
ON (L_1.b = L_2.b **or Y.a = X.a**)

Il join aggiunto può riguardare arbitrariamente uno qualsiasi dei campi della relazione L_3. In questo modo il risultato torna, ma è necessario ricorrere all'operatore OR; poiché questo algoritmo è nato proprio per evitarlo, questa soluzione non è accettabile. In più, questo approccio è utilizzabile solamente sotto la condizione di omogeneità semantica.

3.5.3 Il nuovo algoritmo

Quindi non basta sostituire solamente l'operatore per ottenere un buon risultato. Inoltre l'applicazione del metodo è vincolato all'ipotesi troppo restrittiva di omogeneità semantica. Per questo motivo proviamo ora a mantenere l'operatore di full disjunction, affiancandogliene un altro che permetta di rimuovere tale vincolo. La scelta più ovvia ricade sulla funzione di risoluzione in definizione 2.3.1 o 2.3.2 introdotte nel capitolo 2. L'estrema generalità di tale funzione rende possibile la sua applicazione in un numero altissimo di ambiti, fra i quali anche quello SEWASIE. Nonostante ciò, per una maggiore chiarezza e coerenza rispetto a quanto fatto fino ad oggi, nella sezione 4.1 verrà presentata la funzione di risoluzione secondo la simbologia già utilizzata in [23].

Osservando la query si può notare come questa sia idealmente in due passi. Il primo che produce gli outerjoin fra L_1 ed L_3 e fra L_2 ed L_3 ed il secondo, che esegue un outerjoin fra i due risultati ottenuti nel primo passo. Per rendere più chiaro l'approccio che verrà adottato si tengano presenti le due tabelle che seguono, contenenti i risultati del primo passo.

L1a	**L1b**	**L3aX**	**L3bX**
1	1000	1	1000
2	2000	NULL	NULL
3	3000	3	3000
4	4000	NULL	NULL
NULL	NULL	6	6000
NULL	NULL	7	7000

L2b	**L3aY**	**L3bY**
1000	1	1000
2000	NULL	NULL
NULL	3	3000
5000	NULL	NULL
6000	6	6000
NULL	7	7000

Ammettiamo adesso di applicare la funzione di risoluzione alle relazioni parziali appena ottenute. Si avrebbero queste due nuove relazioni:

L_{1-3}	
a	**b**
1	1000
2	2000
3	3000
4	4000
6	6000
7	7000

L_{2-3}	
a	**b**
1	1000
NULL	2000
3	3000
NULL	5000
6	6000
7	7000

Si tenga presente che i nomi attribuiti alle relazioni e agli attributi sono solamente indicativi. Nella realtà si utilizzeranno quelli adottati nella relazione globale.
Applichiamo ora il secondo passo della query alle relazioni date in output dalla funzione di risoluzione. La query diverrebbe:
select
L_{1-3}.a as L13a, L_{1-3}.b as L13b, L_{2-3}.a as L23a, L_{2-3}.b as L23b
from L_{1-3} full join L_{2-3} ON (L_{1-3}.b = L_{2-3}.b)

Si otterrebbe quindi la relazione:

L13a	**L13b**	**L23a**	**L23b**
1	1000	1	1000
2	2000	NULL	2000
3	3000	3	3000
4	4000	NULL	NULL
NULL	NULL	NULL	5000
6	6000	6	6000
7	7000	7	7000

Da cui, sottoponendo l'output nuovamente alla funzione di risoluzione:

a	**b**
1	1000
2	2000
3	3000
4	4000
NULL	5000
6	6000
7	7000

Come si può notare, il risultato è identico a quello in tabella 3.4 ottenuto dall'algoritmo che utilizza l'OR; quindi è da considerarsi corretto.

Da questo risultato si può desumere che utilizzando il full outerjoin insieme ad una funzione di risoluzione è possibile arrivare ad un risultato corretto come utilizzando il natural outerjoin, ma in più si risolvono anche i conflitti. È quindi accettabile modificare l'algoritmo prevedendo l'uso di questi operatori.
Le modifiche da apportare sono riportate in Tabella 3.6.

Calcolo della Full Disjunction

Input: * $\mathcal{R} = \{R_1, \ldots, R_n\}$: relazioni definite sullo schema globale.
* $\mathcal{F} = \{F_{i,j}\}$: Join Table per tutte le relazioni di $\mathcal{R}$.
* $\mathcal{F}_r = \{f_{r1}, f_{r2}, \ldots, f_{rm}\}$, insieme delle funzioni di risoluzione; una per oni attributo.

Output: Full Disjunction
Procedimento:
• Poni $F_0 = \mathcal{F}$, $R_0 = \mathcal{R}$ ed $NP = (n-1)$ numero dei passi.
• for $(i = 0; i < NP; i++)$
  • seleziona $R_i \in \mathcal{R}_i$
    $\mathcal{R}_{i+1} = \{R_i =\bowtie_f= R'_i \mid R'_i \in \mathcal{R}_i; R'_i \neq R_i; f = OJC_i(R_i, R'_i)$ non è null$\}$
    • $\forall$ **join presente in** $\mathcal{R}_{i+1}$
      calcola il join
      • $\forall$ **tupla del join**
        • $\forall$ **attributo** j **della tupla**
          applica la funzione di risoluzione f_{rj}
• Restituisci $\mathcal{R}_{NP}$.

Tabella 3.6: Algoritmo PSOJ modificato

In conclusione quindi l'algoritmo funziona, almeno in alcuni casi, nella sua versione originale, ma solo in teoria. Questo perché non è stata tenuta abbastanza in considerazione la sua implementazione pratica.
Infatti, analizzando il teorema 3.5.1 ed in particolare la dimostrazione seguente, si può notare come questa rimanga sempre su un piano prettamente teorico, ipotizzando l'operatore di natural outerjoin come compatibile con lo standard SQL92.
Inoltre si è visto come la semplice sostituzione del natural outerjoin con il full outerjoin non permetta di arrivare ad un risultato corretto, poiché non permette la fusione di colonne provenienti da relazioni diverse in ingresso in un'unica colonna in uscita. Questo porta ad avere relazioni "sporche", nelle quali sono presenti due o più colonne che, idealmente, dovrebbero essere una sola e di conseguenza al fallimento dell'algoritmo.

3.5.4 Il problema delle sorgenti vuote

Come visto in precedenza, sarebbe necessario un ordinamento per risolvere il prolema delle sorgenti vuote. Purtroppo però, per quanto detto da Rajaraman e Ullman, per i grafi γ-ciclici *non è possibile* trovare a priori un ordinamento delle sorgenti, quindi questo problema deve essere affrontato in un modo differente: come soluzione provvisoria si è pensato di adottare la soluzione che prevede di non considerare nell'algoritmo queste sorgenti particolari. Però e necessario trovare un metodo più strutturale per poter utilizzare questo algoritmo anche al di fuori del nostro ambito. Si prenda il solito esempio utilizzato da Ullman. Oltre alle solite relazioni UDF, UDS ed UA supponiamo di avere anche la relazione UAX; inoltre siano UDS ed UAX vuote, UDF ed UA contenenti rispettivamente le tuple udf ed ua. Per comodità assumiamo di utilizzare il natural join come operatore, quindi le condizioni di join saranno poste su tutti gli attributi comuni. A pagina 86 viene mostrato come, partendo da UDF, l'algoritmo PSOJ riesca a calcolare il risultato corretto, quindi anche ora sceglieremo lo stesso nodo. In questo caso, dopo il primo passo la situazione è quella riportata in figura 3.21. I contenuti delle relazioni sono invece quelli riportati in tabella:

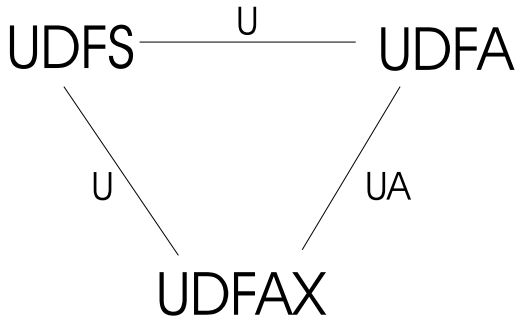


Figura 3.21: Grafo dopo un passo

$UDFS$				$UDFA$				$UDFAX$				
U	D	F	S	U	D	F	A	U	D	F	A	X
u	d	f	$\perp$	u	d	f	a	u	d	f	$\perp$	$\perp$

Già a questo punto è chiaro come le tuple delle relazioni $UDFA$ ed $UDFAX$ non abbiano la possibilità di venire fuse, compromettendo il calcolo della full disjunction, composta chiaramente solo da una tupla.
Quindi l'approccio adottato precedentemente non funziona; utilizzando l'algoritmo SOJO è possibile calcolare una sequenza anche in presenza di relazioni vuote, a patto di non avere ipergrafi γ-ciclici; nel nostro caso questa assunzione è molto

forte, quindi è necessario un metodo alternativo.
In linea di massima è possibile trovare un ordinamento anche nel caso che alcune relazioni siano vuote; indicativamente si ha che:

- quando c'è una sola relazione vuota, è necessario scegliere come primo nodo quello per cui la condizione di join con la sorgente vuota stessa interessa il maggior numero di attributi. Infatti nell'esempio a pagina 86 si opera una scelta in questo senso, dato che UDF esegue il join tramite due attributi con UDS, mentre UA lo fa solo su di uno.
  In alternativa, si raggiungono gli stessi risultati anche prendendo come primo nodo la relazione vuota stessa. Infatti, riferendoci sempre a quell'esempio ed utilizzando come prima scelta la relazione UDS, le due relazioni risultanti $UDFS$ ed $UDSA$ hanno come tuple rispettivamente $udf\perp$ e $u\perp\perp a$. Poiché la condizione di join che verrà utilizzata successivamente sarà sull'attributo u, si vede come le due tuple verranno fuse.

- quando il numero di relazioni vuote è superiore ad uno, perché sia possibile calcolare la full disjunction tramite il nostro algoritmo è necessario scegliere una relazione vuota come nodo da utilizzare. Facendo riferimento all'esempio riportato sopra, è possibile utilizzare come primo nodo sia UDS che UAX; per esempio prendiamo inizialmente UDS. Risultano così le seguenti relazioni:

  $UDFS$

U	D	F	S
u	d	f	$\perp$

  $UDSAX$

U	D	S	A	X

  $UDSA$

U	D	S	A
u	$\perp$	$\perp$	a

  A questo punto prendiamo il nodo $UDSAX$. Come risultato si hanno le due relazioni $UDFSAX$ e $UDSAX$ contenenti rispettivamente $udf\perp\perp\perp$ e $u\perp\perp a\perp$. Poiché la condizione di join è espressa solo su u si ha che le tuple vengono fuse, dando la full disjunction.

Perciò, generalmente, per calcolare la full disjunction anche nei casi in cui vi siano m fonti vuote è sufficiente utilizzare nei primi m passi un nodo corrispondente ad una relazione vuota.
Ma perché utilizzando prima le sorgenti vuote si ottiene il risultato corretto? Per rispondere a questa domanda bisogna ritornare all'idea che sta dietro all'algoritmo SOJO e quindi riferirsi ai diagrammi di Bachman. I diagrammi di questo tipo evidenziano gli attributi di join in comune tra diverse relazioni; l'algoritmo SOJO li sfrutta dando la precedenza alle condizioni di join che si basano su un numero maggiore di attributi. Quindi i join fra le relazioni andrebbero eseguiti in ordine

decrescente di attributi. Utilizzando questo particolare ordinamento si può verificare che si elimina il problema delle sorgenti vuote; purtroppo questo metodo non può essere applicato così com'è nel nostro caso, in quanto noi, una volta scelto un nodo, ne eseguiamo il join con tutte le relazioni possibili, a prescindere dalla struttura delle condizioni di join. È però possibile specializzare ulteriormente il ragionamento, applicandolo solo al caso delle sorgenti vuote. Ciò che importa non è tanto eseguire *tutti* i join nell'ordine specificato, quanto solo quelli riguardanti le fonti vuote. In particolare, per la buona riuscita del metodo è necessaria un sola condizione: siano L_1 ed L_2 due fonti, sia F_{ij} una generica condizione di join fra le relazioni L_i ed L_j e sia $card(F_{ij})$ una funzione che restituisca il numero di attributi sul quale si basa F_{ij}, sia L_1 vuota e sia $card(F_{12}) \geq card(F_{1j})$; allora per evitare il problema delle fonti vuote è necessario che il primo join nel quale la relazione L_2 è coinvolta sia proprio quello con la relazione L_1.
A questo punto dovrebbe essere chiara la scelta evidenziata precedentemente nel caso di una sola relazione vuota: se si prende la relazione vuota L_1, allora c'è la certezza che il primo join eseguito su L_2 sia proprio quello richiesto, dato che verrebbero eseguiti tutti i join del tipo F_{1j}; scegliendo invece L_2, il principio vale ugualmente in quanto tutti i join del tipo F_{i2} vengono eseguiti contemporaneamente e quindi si può assumere che F_{12} sia eseguito per primo. Quest'ultimo approccio non vale in generale nel caso di più sorgenti vuote, in quanto L_2 dovrebbe avere valore di $card()$ massimo per ogni sorgente vuota contemporaneamente e non solamente per L_1. La tecnica che vede prendere le sorgenti vuote prima di tutte le altre funziona invece anche in questo caso, poiché quanto detto per una sola relazione vuota può essere reiterato perfettamente.

Nonostante quanto detto sembri in contrasto con quanto affermato da Rajaraman e Ullman, in realtà così non è. Infatti a priori, ossia conoscendo solo gli schemi, non è realmente possibile calcolare la full disjunction; per applicare quanto trovato sarà necessario procedere, prima di ogni esecuzione, ad un esame delle fonti, per stabilire quali sono vuote.

Questo importante risultato permette di utilizzare l'algoritmo anche in presenza di grafi non completamente connessi; questa limitazione non interessava quello che è l'ambito principale per cui è stato ideato, ma comunque ne minava la generalità applicativa. In questi ultimi però bisogna fare attenzione a mantenere il grafo connesso, quindi è necessario ritardare l'uso di una sorgente vuota quando questa porterebbe ad una disconnessione.

3.5.5 I limiti dell'algoritmo

Il modo in cui è stato affrontato il problema delle sorgenti vuote fa sorgere spontaneo un dubbio: un ordinamento delle fonti si rende necessario se e solo se fra

queste sono presenti sorgenti vuote?
In altri termini: i risultati dello studio di Ullman si rendono utili solamente in questi limitati casi?
In generale la risposta è no, come si può vedere dal seguente esempio:
prendiamo gli schemi del solito esempio usato da Ullman, ipotizzando clausole di join di tipo natural ed ottenendo quindi un ipergrafo *non γ-ciclico*, ma modifichiamo il contenuto delle singole relazioni:

UDF

U	D	F
U_1	D_1	F_1
U_3	D_3	F_3

UDS

U	D	S
U_1	D_1	S_1
U_2	D_2	S_2

UA

U	A
U_2	A_2
U_3	A_3

A questo punto realizzando il join utilizzando la sequenza calcolata dall'algoritmo SOJO

$$(UDF \overset{\circ}{\bowtie} UDS) \overset{\circ}{\bowtie} UA$$

si ottiene:

$(UDF \overset{\circ}{\bowtie} UDS) \overset{\circ}{\bowtie} UA$

U	D	F	S	A
U_1	D_1	F_1	S_1	NULL
U_2	D_2	NULL	S_2	A_2
U_3	D_3	F_3	NULL	A_3

Che corrisponde alla vera full disjunction. Utilizzando però la sequenza

$$(UA \overset{\circ}{\bowtie} UDS) \overset{\circ}{\bowtie} UDF$$

il risultato diventa:

$(UA \overset{\circ}{\bowtie} UDS) \overset{\circ}{\bowtie} UDF$

U	D	F	S	A
U_1	D_1	F_1	S_1	NULL
U_2	D_2	NULL	S_2	A_2
U_3	NULL	NULL	NULL	A_3
U_3	D_3	F_3	NULL	NULL

che è evidentemente errato; quindi l'ordinamento è necessario anche quando non ci sono sorgenti vuote.

Per quanto riguarda il nostro ambito, il discorso diventa più complesso. Data la necessità di andare oltre il limite posto dai γ-cicli, nel nostro caso trovare un ordinamento diventa impossibile; d'altra parte Konstantin afferma che utilizzando l'algoritmo da lui proposto non è necessario nessun ordinamento. Come visto in precedenza ciò non è vero nel caso di fonti vuote, ma ci sono altri casi in cui questo errore si fa sentire? Purtroppo la risposta è positiva.
Supponiamo di avere a che fare con lo stesso schema dell'esempio precedente ed ammettiamo di utilizzare l'algoritmo PSOJ prendendo come primo nodo UDF. Allora dopo un passo si avrebbe:

$UDFS$

U	D	F	S
U_1	D_1	F_1	S_1
U_2	D_2	NULL	S_2
U_3	D_3	F_3	NULL

$UDFA$

U	D	F	A
U_1	D_1	F_1	NULL
U_2	NULL	NULL	A_2
U_3	D_3	F_3	A_3

Poiché l'ultimo passo vedrebbe le due relazioni in join sull'attributo U (condizione di join fra UDS ed UA), il risultato corrisponderebbe alla full disjunction corretta calcolata precedentemente.
Prendendo come nodo iniziale UA il risultato non dovrebbe cambiare, invece dopo un passo si ha:

$UDSA$

U	D	S	A
U_1	D_1	S_1	NULL
U_2	D_2	S_2	A_2
U_3	NULL	NULL	A_3

$UDFA$

U	D	F	A
U_1	D_1	F_1	NULL
U_2	NULL	NULL	A_2
U_3	D_3	F_3	A_3

Poiché l'ultimo passo prevede il join fra le due relazioni basandosi sulla coppia di attributi UD (clausola di join tra UDF ed UDS), il risultato diviene:

$UDFSA$

U	D	F	S	A
U_1	D_1	F_1	S_1	NULL
U_2	D_2	NULL	S_2	A_2
U_2	NULL	NULL	NULL	A_2
U_3	D_3	F_3	NULL	A_3
U_3	NULL	NULL	NULL	A_3

Che contiene tutte le informazioni possibili riguardo ai tre oggetti, ma presenta anche due tuple sussunte (la terza e la quinta) e quindi non può essere considerata una full disjunction corretta.
Ricordo che l'ipergrafo degli schemi non è neppure γ-ciclico, quindi l'algoritmo

di Konstantin non riesce a trattare neppure casi risolubili col precedente algoritmo di Ullman.
Eppure le cose sembravano funzionare bene; ora sorge spontaneo chiedersi quali siano realmente i limiti di questo algoritmo: per fare questo è necessario richiamare un concetto presentato per la prima volta in [37]: la β-ciclicità.
Per facilitare la comprensione, nonostante le definizioni (anche se informali) siano già state date in sezione 3.2, è bene reintrodurne alcune utilizzando la stessa notazione.

Definizione 3.5.1 (ciclo puro) *Sia* $(S_1, \ldots, S_m, S_{m+1})$ *una sequenza di insiemi, dove* $S_1, \ldots, S_m$ *sono distinti e* $S_1 = S_{m+1}$*. Chiamiamo* S_i *ed* S_{i+1} $(1 \leq i \leq m)$vicini*; in particolare* S_m *ed* S_1 *sono vicini.* $(S_1, \ldots, S_m, S_{m+1})$ *si dice* ciclo puro *se* $m \geq 3$ *e, con* $i \neq j$*,* $S_i \cap S_j$ *è non vuoto se e solo se* S_i *e* S_j *sono vicini.*

Cioè un coppia è non disgiunta se e solo se è una coppia di vicini. Se $m = 3$ è necessario che $S_1 \cap S_2 \cap S_3 = \emptyset$.
Nel nostro caso, ovviamente, quando si parla di "sequenza di insiemi" si assume che gli insiemi abbiano come elementi degli iperarchi. Per un esempio di ciclo puro si faccia riferimento alla figura 3.2, dove gli insiemi sono indicati con X_i invece che S_i.

Definizione 3.5.2 (β-ciclo) *Un* β-ciclo *in un ipergrafo* $\mathcal{H}$ *è una sequenza* $(S_1, \ldots, S_m, S_{m+1})$ *di iperarchi tale che se* $X = S_1 \cap \cdots \cap S_m$ *e* $S'_i = S_i - X$ $(1 \leq i \leq m)$ *allora* $(S'_1, \ldots, S'_m, S'_{m+1})$ *è un ciclo puro.*

Perciò ogni β-ciclo è nella forma $(S'_1 \cup X, \ldots, S'_m \cup X, S'_{m+1} \cup X)$, dove $(S'_1, \ldots, S'_m, S'_{m+1})$ è un ciclo puro. Da ciò discende che ogni ciclo puro è un β-ciclo. Per un esempio di β-ciclo si faccia riferimento alla figura 3.22.

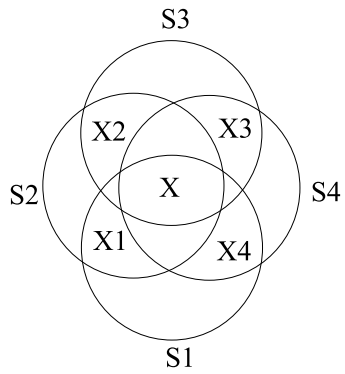


Figura 3.22: Esempio di β-ciclo

Definizione 3.5.3 (β-ciclicità) *Un ipergrafo si dice β-ciclico se contiene un sottoinsieme in forma di β-ciclo.*

Esistono anche altre definizioni equivalenti per la β-ciclicità, ma questa è quella che serve meglio al nostro scopo.

Definizione 3.5.4 (β-ciclo debole) *Un* β-ciclo debole *in un ipergrafo $\mathcal{H}$ è una sequenza $(S_1, x_1, S_2, x_2, \ldots, S_m, x_m, S_{m+1})$ tale che:*

1. *$x_1, \ldots, x_m$ sono nodi distinti di $\mathcal{H}$,*
2. *$S_1, \ldots, S_m$ sono iperarchi distinti di $\mathcal{H}$ e $S_1 = S_{m+1}$,*
3. *$m \geq 3$, ossia ci sono almeno tre iperarchi interessati,*
4. *x_i è in S_i ed in S_{i+1} $(1 \leq i \leq m)$ ma in nessun altro S_j.*

Denotando un β-ciclo debole con $(S_1, \ldots, S_m, S_{m+1})$ (omettendo i nodi) è chiaro come ogni β-ciclo sia anche un β-ciclo debole; comunque il contrario non è vero: per un esempio di β-ciclo debole si faccia riferimento alla figura 3.23. Come si vede, il nodo X non è in S_4, quindi la figura non rappresenta un β-ciclo.
Con questa definizione si può estendere il concetto di β-ciclicità introdotto prima.

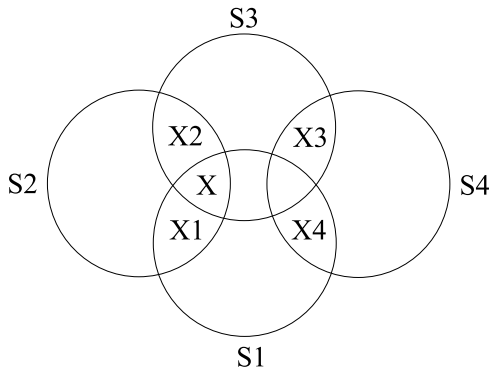


Figura 3.23: Esempio di β-ciclo debole

Definizione 3.5.5 (β-ciclicità) *Un ipergrafo si dice β-ciclico se contiene un sottoinsieme in forma di β-ciclo debole.*

Definizione 3.5.6 (γ-ciclo) *Un* γ-ciclo *in un ipergrafo $\mathcal{H}$ è una sequenza $(S_1, x_1, S_2, x_2, \ldots, S_m, x_m, S_{m+1})$ tale che:*

1. *$x_1, \ldots, x_m$ sono nodi distinti di $\mathcal{H}$,*

2. $S_1, \ldots, S_m$ *sono iperarchi distinti di* $\mathcal{H}$ *e* $S_1 = S_{m+1}$,

3. $m \geq 3$, *ossia ci sono almeno tre iperarchi interessati,*

4. x_i *è in* S_i *ed in* S_{i+1} $(1 \leq i \leq m)$,

5. *se* $1 \leq i < m$ *allora* x_i *non è in nessun* S_j *eccetto* S_i *e* S_{i+1}.

Poiché l'unica differenza tra un γ-ciclo ed un β-ciclo debole risiede in $1 \leq i < m$ al posto di $1 \leq i \leq m$, è chiaro che ogni β-ciclo debole è anche un γ-ciclo. Il contrario non è vero: in figura 3.24 viene mostrato un γ-ciclo che però non è un β-ciclo.
Un γ-ciclo composto da m iperarchi distinti si dice *di dimensione* m.

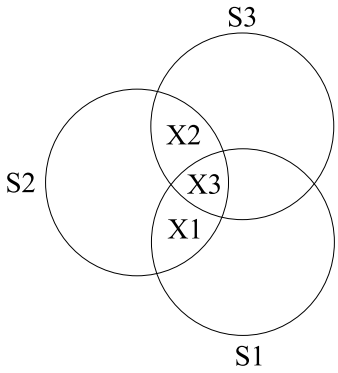


Figura 3.24: Esempio di γ-ciclo

Definizione 3.5.7 (γ-ciclicità) *Le seguenti definizioni sono equivalenti:*

- *Un ipergrafo si dice γ-ciclico se contiene un sottoinsieme in forma di γ-ciclo.*

- *Un ipergrafo si dice γ-ciclico se contiene un sottoinsieme o in forma di γ-ciclo di dimensione 3 o in forma di ciclo puro.*

A questo punto, date le definizioni, è facile comprendere che, poiché ogni β-ciclo è un β-ciclo debole e che ogni β-ciclo debole è un γ-ciclo, si ha:

$$\gamma\text{-aciclicità} \Rightarrow \beta\text{-aciclicità}$$

Quindi l'insieme delle configurazioni in forma di β-ciclo è un sottoinsieme dell'insieme delle configurazioni in forma di γ-ciclo. Da ciò risulta chiaro che l'algoritmo, trattando tutte le configurazioni γ-cicliche dovrebbe riuscire a risolvere senza problemi anche le configurazioni β-cicliche.
Con il seguente esempio proverò che non è proprio così. Per comodità si assuma che la condizione di join fra le relazioni di base coinvolga tutti gli attributi comuni tra loro (come avviene per il join naturale).
Supponiamo di avere i seguenti schemi:

- Schema 1

L_1		L_2		L_3	
A	B	B	C	A	C
A_1	B_1	B_2	C_2	A_1	C_1
A_2	B_2	B_3	C_3	A_3	C_3

- Schema 2

L_1		L_2			L_3	
A	B	A	B	C	A	C
A_1	B_1	A_2	B_2	C_2	A_1	C_1
A_2	B_2	A_3	B_3	C_3	A_3	C_3

- Schema 3

L_1			L_2			L_3		
A	B	D	A	B	C	A	C	D
A_1	B_1	D_1	A_2	B_2	C_2	A_1	C_1	D_1
A_2	B_2	D_2	A_3	B_3	C_3	A_3	C_3	D_3

Gli ipergrafi corrispondenti sono rappresentati in figura 3.25, 3.26 e 3.27.

Il primo schema risulta un ciclo puro formato da tre iperarchi. Si nota subito una certa simmetria nell'ipergrafo; intuitivamente si capisce come questa simmetria renda il risultato finale indipendente dal nodo che viene scelto al primo passo. Per questo motivo partiamo da L_1. Dopo il primo passo si avrebbe:

$L_1 \bowtie_B L_2$			$L_1 \bowtie_A L_3$		
A	B	C	A	B	C
A_1	B_1	$\perp$	A_1	B_1	C_1
A_2	B_2	C_2	A_2	B_2	$\perp$
$\perp$	B_3	C_3	A_3	$\perp$	C_3

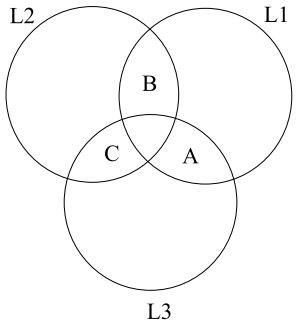


Figura 3.25: Schema 1: ciclo puro (schema γ-ciclico e β-ciclico)

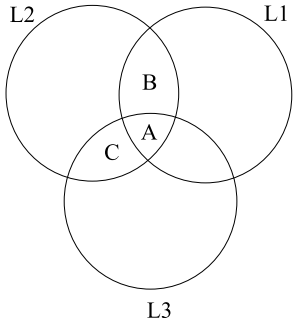


Figura 3.26: Schema 2: γ-ciclo di dimensione 3 (schema γ-ciclico e β-aciclico)

Il join successivo si basa sulla clausola di join fra le relazioni base L_2 ed L_3 e quindi avviene su C:

FD?

A	B	C
A_1	B_1	C_1
A_1	B_1	$\perp$
A_2	B_2	C_2
A_2	B_2	$\perp$
A_3	B_3	C_3

Si vede chiaramente che la seconda e la quarta tupla sono sussunte rispettivamente dalla prima e dalla terza, quindi ciò che è stato ottenuto non può essere considerato come la full disjunction.

Passiamo adesso al secondo schema: supponiamo di voler prendere come relazione di partenza L_1. Dopo un passo si avrebbe:

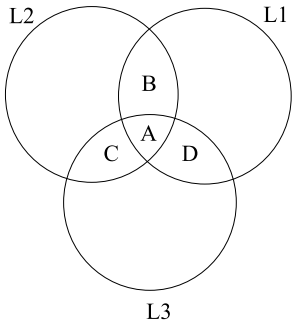


Figura 3.27: Schema 3: γ-ciclo di dimensione 3 (schema γ-ciciclico e β-ciclico)

$L_1 \bowtie_{AB} L_2$

A	B	C
A_1	B_1	$\perp$
A_2	B_2	C_2
A_3	B_3	C_3

$L_1 \bowtie_{A} L_3$

A	B	C
A_1	B_1	C_1
A_2	B_2	$\perp$
A_3	$\perp$	C_3

il join successivo si basa sulla clausola di join fra le relazioni base L_2 ed L_3 e quindi avviene su AC:

FD?

A	B	C
A_1	B_1	C_1
A_1	B_1	$\perp$
A_2	B_2	C_2
A_2	B_2	$\perp$
A_3	B_3	C_3

La seconda e la quarta tupla vengono rispettivamente sussunte dalla prima e dalla terza, quindi quella ottenuta non è la full disjunction. Proviamo invece a partire da L_2:

$L_1 \bowtie_{AB} L_2$

A	B	C
A_1	B_1	$\perp$
A_2	B_2	C_2
A_3	B_3	C_3

$L_2 \bowtie_{AC} L_3$

A	B	C
A_1	$\perp$	C_1
A_2	B_2	C_2
A_3	B_3	C_3

il join successivo si basa sulla clausola di join fra le relazioni base L_1 ed L_3 e quindi avviene su A:

FD

A	B	C
A_1	B_1	C_1
A_2	B_2	C_2
A_3	B_3	C_3

Questa è davvero la full disjunction, quindi per ottenere un risultato corretto è necessario prevedere un ordinamento opportuno. Intuitivamente, guardando il rispettivo ipergrafo di figura 3.26, si vede come sia presente una certa asimmetria dell'ipergrafo: la necessità di un ordinamento deriva proprio da questo aspetto.

Passiamo ora al terzo schema: guardando l'ipergrafo corrispondente (figura 3.27) si vede come manchi l'asimmetria presente nell'ipergrafo precedente. In particolare è facile verificare come il risultato finale sia completamente indipendente dalla scelta del nodo iniziale. Per completare l'esempio scelgo come nodo iniziale L_1. Dopo il primo passo si ottiene:

$L_1 \bowtie_{AB} L_2$

A	B	C	D
A_1	B_1	$\perp$	D_1
A_2	B_2	C_2	D_2
A_3	B_3	C_3	$\perp$

$L_1 \bowtie_{AD} L_3$

A	B	C	D
A_1	B_1	C_1	D_1
A_2	B_2	$\perp$	D_2
A_3	$\perp$	C_3	D_3

il join successivo si basa sulla clausola di join fra le relazioni base L_2 ed L_3 e quindi avviene su AC:

FD?

A	B	C	D
A_1	B_1	C_1	D_1
A_1	B_1	$\perp$	D_1
A_2	B_2	C_2	D_2
A_2	B_2	$\perp$	D_2
A_3	B_3	C_3	D_3

Anche in questo caso la seconda e la quarta tupla risultano sussunte rispettivamente dalla prima e dalla terza, quindi non siamo di fronte alla full disjunction. A differenza del problema precedente però, ora non è possibile trovare un ordinamento in grado di portare ad un risultato differente da quello esposto; questo è dovuto alla simmetria dell'ipergrafo, la stessa simmetria che nel primo schema impediva di arrivare al risultato corretto.
Prendendo in esame i tre ipergrafi si nota subito che la principale differenza tra loro è costituita dal fatto che, mentre il secondo ipergrafo è β-aciclico, gli altri due sono β-ciclici. Proprio in questo risiede la fonte dell'errore. L'algortimo è in

grado di trattare solamente un sottoinsieme degli schemi γ-ciclici, per la precisione quelli β-aciclici. Questo è dovuto alla simmetria intrinseca nella struttura dei β-cicli: riprendendo la definizione, si nota infatti che ogni β-ciclo è un ciclo puro con l'aggiunta di un insieme di nodi X, comune a tutti gli iperarchi. Perciò non si può ricercare un ordinamento vero e proprio, in quanto tutti gli iperarchi hanno le stesse caratteristiche.
Si potrebbe pensare che la situazione migliori quando il grafo non è totalmente connesso: quindi consideriamo un ciclo puro costituito da quattro nodi.
Ciò non interessa l'ambito SEWASIE, in quanto considerando gli schemi sempre e comunque completamente connessi ha senso parlare di cicli puri solamente se la loro dimensione è minima. Modifichiamo quindi lo schema 1 nel seguente modo:

L_1		L_2		L_3		L_4	
A	B	B	C	C	D	D	A
A_1	B_1	B_2	C_2	C_3	D_3	D_1	A_1
A_2	B_2	B_3	C_3	C_4	D_4	D_4	A_4

Per questo caso vale lo stesso discorso di prima, ossia la scelta del nodo sul quale eseguire il primo passo è da considerarsi completamente arbitraria; quindi scelgo come primo nodo L_1, che andrà in join solamente con L_2 ed L_4.

$L_1 \bowtie_B L_2$			$L_1 \bowtie_A L_4$			L_3	
A	B	C	A	B	D	C	D
A_1	B_1	$\perp$	A_1	B_1	D_1	C_3	D_3
A_2	B_2	C_2	A_2	B_2	$\perp$	C_4	D_4
$\perp$	B_3	C_3	A_4	$\perp$	D_4		

Al secondo passo, per evitare di sconnettere il grafo, il nodo L_3 non può essere preso. Perciò scelgo il nodo $L_1 \bowtie_B L_2$, che va in join solamente con L_3.

$(L_1 \bowtie_B L_2) \bowtie_C L_3$				$L_1 \bowtie_A L_4$		
A	B	C	D	A	B	D
A_1	B_1	$\perp$	$\perp$	A_1	B_1	D_1
A_2	B_2	C_2	$\perp$	A_2	B_2	$\perp$
$\perp$	B_3	C_3	D_3	A_4	$\perp$	D_4
$\perp$	$\perp$	C_4	D_4			

Il join rimasto viene eseguito sull'attributo D.

FD?

A	B	C	D
A_1	B_1	$\perp$	D_1
A_1	B_1	$\perp$	$\perp$
A_2	B_2	C_2	$\perp$
A_2	B_2	$\perp$	$\perp$
$\perp$	B_3	C_3	D_3
A_4	$\perp$	C_4	D_4

Quindi si ripresenta il solito problema delle tuple sussunte.

Finora si è parlato solo di β-cicli, ma non di β-cicli deboli; questi ultimi causano qualche tipo di problema? La risposta è positiva. Si prenda infatti l'ipergrafo di figura 3.23: può essere visto come l'insieme di un γ-ciclo di dimensione tre (formato dagli iperarchi S_1, S_2 ed S_3) e di un ciclo puro di dimensione tre (composto da S_1,S_3 ed S_4). In questo caso non si presenta la simmetria tipica dei β-cicli, quindi la scelta del nodo iniziale influenza il risultato finale. Ad un certo punto ci si troverà comunque ad affrontare un ciclo puro, con le conseguenze del caso. Si può provare che non esiste un ordinamento in grado di dare la full disjunction.
Quindi l'insieme delle configurazioni trattabili si riduce ulteriormente, dato che i β-cicli sono un sottoinsieme dei β-cicli deboli.

Ricapitolando, per ora abbiamo ridotto i casi nei quali l'algoritmo funziona ai seguenti:

1. con schemi di database aciclici
2. con schemi di database γ-ciclici ma non β-ciclici

3.5.6 L'errore della dimostrazione

Poiché l'algoritmo di Konstantin fallisce, evidentemente anche la dimostrazione ha qualcosa che non va.
Dato che la full disjunction per definizione preserva tutte le connessioni tra i fatti è ovvio che:

$$FD(L_i, L_j) \quad = \quad L_i =\bowtie_{F_{ij}}= L_j \tag{3.2}$$

dove F_{ij} rappresenta la condizione di join tra L_i e L_j.
La full disjunction non è altro che una relazione e quindi si può estendere l'uguaglianza appena esposta nel seguente modo:

$$FD(FD(L_i, L_j), FD(L_i, L_k)) \quad = \quad FD(L_i, L_j) =\bowtie_{F_{jk}}= FD(L_i, L_k) \tag{3.3}$$

Facendo riferimento allo studio di Galindo-Legaria si può verificare che le precedenti uguaglianze si basano sul concetto di disgiunzione e quindi perché siano valide deve essere possibile riscrivere gli outerjoin come disgiunzioni. In particolare dovrebbe valere l'uguaglianza:

$$L_1 =\bowtie_{F_{12}}= L_2 \quad = \quad L_1 \bowtie_{F_{12}} L_2 \oplus L_1 \oplus L_2 \tag{3.4}$$

ma ciò è vero se e solo se L_1 e L_2 non presentano tuple sussunte. Proprio questo è il punto: noi possiamo assumere l'assenza di tuple sussunte sulle relazioni locali, ma chi ci assicura che tali tuple non si vengano poi a formare durante la computazione dell'algoritmo? Il caso base della dimostrazione mostra che la full disjunction tra due relazioni viene calcolata tramite un full outerjoin (in realtà tramite un natural outerjoin, ma il ragionamento vale lo stesso), proprio come mostrato in equazione 3.2; i casi successivi invece fanno affidamento sull'equazione 3.3. Entrambe però valgono se e solo se le relazioni che vanno in join non contengono tuple sussunte, cosa che in generale non può essere verificata. Perchè tale assunzione sia valida è necessario, oltre che avere relazioni base prive di tuple sussunte, che sia verificata su ogni join l'ipotesi di consistenza dei valori di join. Sia Ullman che Konstantin operano in presenza di tale ipotesi, ma solo il primo fornisce un algoritmo corretto. Questo perchè l'ipotesi è fatta solamente sulle relazioni di base, quindi niente si può dire sui successivi join; Ullman però riesce a reiterare la validità dell'ipotesi join dopo join, tramite un opportuno ordinamento. In fondo proprio questa è la motivazione di un ordinamento e la causa principale della difficoltà di calcolo della full disjunction. La mancanza di un ordinamento nell'algoritmo di Konstantin rende inutile, a parte casi eccezionali, l'ipotesi sulle relazioni di base, perchè questa non viene conservata fino alla fine.

Questo è il motivo che porta al fallimento dell'algoritmo di Konstantin, ma come mostrato nelle precedenti sezioni anche seguendo un approccio differente i problemi non si risolvono, sempre per lo stesso motivo.
Dimostrare la validità del nuovo algoritmo equivale a dimostrare che

$$FD(L_1, L_2, L_3 \ldots, L_n) = FD(FD(L_1, L_2), FD(L_1, L_3) \ldots, FD(L_1, L_n))$$

e quindi al momento del join vero e proprio si ripresenta il problema delle tuple sussunte come in precedenza. Comunque in assenza di tali tuple si può dimostrare che l'uguaglianza vale, semplicemente applicando le uguaglianze 3.2, 3.3 e 3.4, riconducendosi al concetto di disgiunzione.

3.5.7 L'ordinamento da utilizzare

Come ormai risulta più che chiaro, è necessario servirsi di un opportuno ordinamento per giungere ad un risultato corretto.

Prima di tutto è necessario comprendere su quali basi si debba poggiare l'ordinamento da cercare. Osservando le sezioni precedenti si nota come, sia nel caso delle fonti vuote sia in quello dell'ipergrafo γ-ciclico e β-aciclico, ciò che viene messo in risalto sono le condizioni di join: in particolare la loro cardinalità. Sembra quindi che utilizzando un ordinamento che assegni maggiore priorità alle sorgenti con clausole di join costituite da un numero maggiore di attributi si possa arrivare alla full disjunction. Ma se davvero fosse questa l'unica condizione richiesta allora nei precedenti esempi riguardanti i cicli puri non ci sarebbe stato problema, dato che tutte le condizioni di join sono costituite da un singolo attributo. Quindi è necessaria una clausola aggiuntiva, che per ora non è stata considerata.
Per capirla è necessario risalire all'esempio a pagina 100; prendendo per prima la relazione avente i join con maggiore cardinalità si ottiene il risultato corretto, ma analizzando attentamente tutti i passaggi si può arrivare anche ad un altro risultato. Per arrivare alla causa dell'errore è necessario mettere a confronto il caso in cui si prenda UA come prima relazione (caso che porta ad avere tuple sussunte nel risultato) e quello in cui la prima scelta ricada su UDF (dove si arriva alla vera full disjunction). In particolare osserviamo ciò che accade dopo la prima iterazione:

- **caso** UA

$UA \overset{\circ}{\bowtie} UDF$

U	D	F	A
U_1	D_1	F_1	$\perp$
U_2	$\perp$	$\perp$	A_2
U_3	D_3	F_3	S_3

$UA \overset{\circ}{\bowtie} UDS$

U	D	S	A
U_1	D_1	S_1	$\perp$
U_2	D_2	S_2	A_2
U_3	$\perp$	$\perp$	A_3

  Il join successivo avviene su UD

- **caso** UDF

$UDF \overset{\circ}{\bowtie} UA$

U	D	F	A
U_1	D_1	F_1	$\perp$
U_2	$\perp$	$\perp$	A_2
U_3	D_3	F_3	S_3

$UDF \overset{\circ}{\bowtie} UDS$

U	D	F	S
U_1	D_1	F_1	S_1
U_2	D_2	$\perp$	S_2
U_3	D_3	F_3	$\perp$

  Il join successivo avviene su U.

Si nota subito come nel primo esempio siano i valori nulli a causare il prolema, non permettendo la fusione delle tuple che rappresentano lo stesso oggetto reale, dato che: $U_2D_2 \neq U_2\perp$ e $U_3D_3 \neq U_3\perp$.
Anche nel secondo caso esistono dei valori nulli, ma non influenzano il successivo

join, dato che l'attributo di join presenta valori significativi per tutte le tuple.
Una volta stabilita la fonte del problema, resta da chiedersi perché questo avvenga solo in determinati casi. Per rispondere a questa domanda non è sufficiente basarsi sulle condizioni di join: è necessario fare riferimento anche agli schemi delle relazioni e a come questi vengono "estesi" dai join. Con il termine esteso si vuole intendere il fatto che, a seguito di un join, una relazione acquisisca una o più colonne facenti parte dello schema dell'altra relazione. Ad esempio, facendo riferimento al secondo caso riportato sopra, lo schema UDF acquisisce l'attributo S, mentre lo schema UDS acquisisce l'attributo F. Come si può notare, sono proprio in queste colonne acquisite che si verificano eventuali valori nulli indesiderati.
Mentre questo fatto nel secondo caso non crea problemi, in quanto il join successivo utilizza un altro attributo presente in entrambe le relazioni originali, nel primo può causare delle noie; infatti la relazione UA viene estesa a $UDFA$ e $UDSA$, acquisendo gli attributi DFS e causando su questi alcuni valori non significativi. Il problema sta nel fatto che l'attributo D viene utilizzato in un join successivo, quindi i valori nulli rendono impossibile una corretta fusione.
Da questo esempio è chiaro che per ottenere la full disjunction è necessario che gli attributi acquisiti non rientrino nelle successive condizioni di join. Con "successivi" si vogliono intendere solo i join che il nodo in esame deve affrontare forzatamente, ossia quelli con i nodi a cui è collegato direttamente. Ad esempio, il grafo in figura 3.28 non può essere risolto in quanto partendo da UDF o UDS (dove con questi nomi si intende sia il nome della relazione che lo schema della stessa) si estenderebbe lo schema con A, attributo del prossimo join, mentre scegliendo UA o UAX vale lo stesso discorso, ma per l'attributo D. Il grafo in figura 3.29-a può invece essere risolto. Prendiamo prima il nodo UDF: lo schema viene esteso con l'attributo A, possibile fonte d'errore, in quanto nel grafo è presente un join su UA (si veda la situazione nella figura 3.29-b). Se al secondo passo scegliamo UAX si ha la scomparsa dell'unica fonte d'errore, in quanto il join su UA viene eseguito (si veda la situazione nella figura 3.29-c). A questo punto non ci sono più difficoltà fino al termine dell'algoritmo.

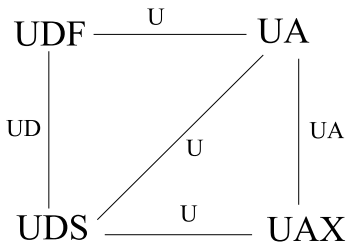
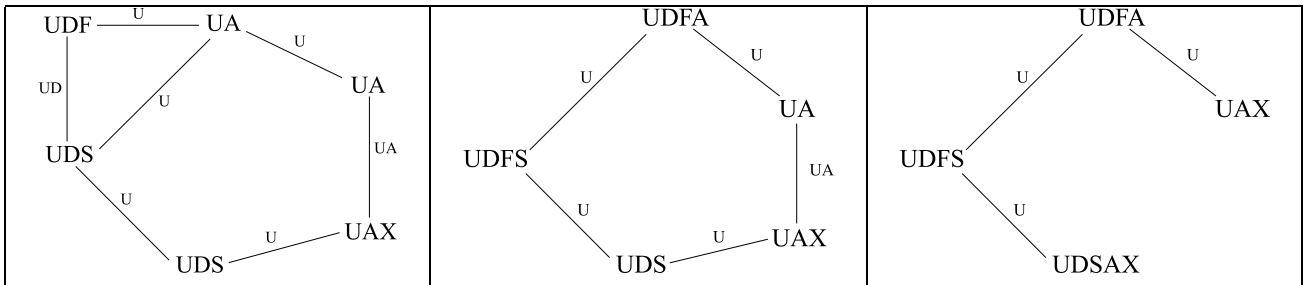


Figura 3.28: Grafo non risolubile



3.29-a) grafo iniziale | 3.29-b) dopo un passo | 3.29-c) dopo due passi

Figura 3.29: Grafo risolubile

Utilizzando questo ragionamento ora è facile capire perché i β-cicli non possano essere trattati da un algoritmo di questo tipo; facciamo riferimento all'ipergrafo 3.25 ed al relativo schema di relazioni: poiché, come affermato in precedenza, il nodo di partenza non influenza il risultato finale, ammettiamo senza quindi perdere in generalità di prendere come relazione iniziale L_1. Sia nel join con L_2 che in quello con L_3 lo schema della relazione iniziale viene esteso dall'attributo C, attributo sul quale si basa il join successivo. Si può vedere facilmente che lo stesso accade iniziando con L_2 o L_3, rispettivamente a causa degli attributi A e B.
Nel caso generale di β-ciclo, mostrato in figura 3.27, l'insieme di attributi X è completamente ininfluente poiché, essendo comune a tutte le relazioni, non estende nessuno schema.
Pe questo motivo in ogni β-ciclo ogni relazione di partenza viene estesa con un attributo che poi sarà utilizzato in un join successivo.
D'ora in avanti chiameremo *corretto* un ordinamento in grado di portare alla full disjunction. Ora mostriamo, limitatamente al caso di tre sorgenti, che è realmente possibile trovare un ordinamento corretto. Supponiamo di avere le sorgenti L_1, L_2 e L_3 e consideriamo le condizioni di join F_{12}, F_{13} e F_{23} tra le relazioni:

- se le condizioni di join contengono solo attributi comuni alle tre relazioni, allora il problema non sussiste, in quanto nessun join provoca un'estensione degli schemi su attributi di join. Un ordinamento non è necessario.
  Esempio: $S(L_1) = \{AC\}$ $S(L_2) = \{AB\}$ $S(L_3) = \{AD\}$ $F_{12} = F_{13} = F_{23} = \{A\}$.

- se uno ed un solo join, ad esempio F_{12}, provoca l'estensione degli schemi su un attributo di join, ciò significa che $(S(L_1) \cup S(L_2)) - (S(L_1) \cap S(L_2)) \neq \emptyset$ e che quindi gli schemi delle due relazioni differiscono in almeno un attributo che nel passo successivo andrà in join. Allora il join tra L_1 ed L_2 dovrà essere trattato per ultimo, in modo tale da ritardare il più possibile l'estensione degli schemi, così da non causare problemi. Quindi il primo passo dell'algoritmo verrà eseguito su L_3.

Esempio: $S(L_1) = \{AB\}$ $S(L_2) = \{AC\}$ $S(L_3) = \{ABC\}$ $F_{12} = \{A\}$ $F_{13} = \{B\}$ $F_{23} = \{A\}$

- se due join, ad esempio F_{12} e F_{13}, provocano un estensione degli schemi, allora si deve forzatamente avere $F_{12} = F_{13}$. Infatti ammettiamo che sia $F_{12} \neq F_{13}$, allora deve esistere un insieme di attributi $X_{12} \subseteq F_{12}$ tale che $X_{12} \subseteq L_1,\ X_{12} \subseteq L_2,\ X_{12} \nsubseteq L_3$ ed un insieme $X_{13} \neq X_{12}$ tale che $X_{13} \subseteq L_1,\ X_{13} \nsubseteq L_2,\ X_{13} \subseteq L_3$. Il join F_{13} non deve estendere gli schemi con attributi di join, ma ciò è assurdo in quanto indipendentemente dagli altri attributi L_2 viene esteso con X_{13}, mentre L_3 con X_{12}.
  Quindi supponiamo che $F_{12} = F_{13}$. Per questo motivo è ovvio che gli attributi che compongono i due join sono comuni a tutti gli schemi e quindi non causano problemi se lasciati per ultimi. Perciò per ottenere un risultato corretto è sufficiente non prendere come primo nodo quello che ha i due join uguali; in questo caso L_1.
  Esempio: $S(L_1) = \{A\}$ $S(L_2) = \{ABC\}$ $S(L_3) = \{ABC\}$ $F_{12} = F_{13} = \{A\}$ $F_{23} = \{B\}$

- se tutti e tre i join provocano estensioni degli schemi su attributi di join allora se $F_{12} \cap F_{23} \cap F_{13} = X$ con $\emptyset \subseteq X$ e:

  - $\nexists(L_i, L_j) \mid F_{ij} = X$. Allora l'ipergrafo corrispondente è un β-ciclo e quindi non trattabile dall'algoritmo.
    Esempio: $S(L_1) = \{AB\}$ $S(L_2) = \{BC\}$ $S(L_3) = \{AC\}$ $F_{12} = \{B\}$ $F_{13} = \{C\}$ $F_{23} = \{A\}$ $X = \emptyset$
  - $\exists!(L_i, L_j) \mid F_{ij} = X$. Allora siamo in presenza di un γ-ciclo di dimensione 3, per il quale è facile vedere che, prendendo per prima la relazione $L_k,\ k \neq i, j$ si ottiene il risultato cercato, in quanto l'ultimo join avviene su X.
    Esempio: $S(L_1) = \{AB\}$ $S(L_2) = \{BC\}$ $S(L_3) = \{ABC\}$ $F_{12} = \{B\}$ $F_{13} = \{AB\}$ $F_{23} = \{BC\}$ $X = \{B\}$
  - In ogni altro caso si ha in un ipergrafo aciclico, ma comunque è impossibile che tutti e tre i join provochino estensioni su attributi di join. Prendiamo ad esempio il caso in cui $\exists(L_i, L_j), (L_i, L_k) \mid F_{ij} = F_{ik} = X$: perché siano presenti tutti e tre i join è necessario che X non sia un insieme vuoto. Dato che $F_{23} \supset X$ è ovvio che l'insieme di attributi X è contenuto in ogni clausola di join e in ogni relazione; quindi il join F_{23} non estende nessuna relazione sugli attributi di join, dato che entrambi gli altri join sono costituiti unicamente da X.
    Esempio: $S(L_1) = \{A\}$ $S(L_2) = S(L_3) = \{AB\}$ $F_{12} = \{A\}$ $F_{13} = \{A\}$ $F_{23} = \{AB\}$ $X = \{A\}$

Quindi tra i casi di interesse per l'algoritmo esiste un ordinamento corretto per:

- ipergrafi γ-ciclici e β-aciclici
- ipergrafi γ-aciclici con tre sorgenti

Mancano all'appello gli ipergrafi γ-aciclici con più di tre sorgenti. Questo perchè purtroppo alcuni ipergrafi di questo tipo non possono essere risolti con il nostro algoritmo. Si prenda per esempio il grafo di figura 3.28: l'ipergrafo corrispondente è aciclico ed è stato mostrato come sia impossibile arrivare alla full disjunction.

Ora il problema è come trovare l'ordinamento. Non è possibile affrontare il problema passo dopo passo, in quanto, a causa delle scelte precedenti, ci si potrebbe trovare in un vicolo cieco. Per questo è necessario calcolare l'intera sequenza all'inizio dell'esecuzione: quindi non si può fare altro che procedere per tentativi. Fortunatamente esiste un modo per ridurre il numero di configurazioni da provare: infatti si può dimostrare che condizione necessaria perchè un ordinamento sia corretto è che il nodo scelto ad ogni passo sia quello con i join di cardinalità maggiore. Quanto detto fino a questo momento sembrerebbe in contrasto con i ragionamenti fatti per le sorgenti vuote, ma così non è.
Riprendendo il solito esempio di Ullman di pagina 86, abbiamo visto più volte che prendendo come primo nodo UA il risultato era errato. Supponiamo ora di avere UA vuoto:

L_1			L_2			L_3	
U	D	F	U	D	S	U	A
U_1	D_1	F_1	U_1	D_1	S_1		

e supponiamo per comodità le condizioni di join su tutti gli attributi comuni.
Prendendo L_3 come primo nodo ci dovremmo aspettare un errore, in quanto L_3 viene esteso con D, attributo di prossimo join.
Invece dopo il primo passo si ottiene:

$L_1 =\bowtie= L_3$				$L_2 =\bowtie= L_3$			
U	D	F	A	U	D	S	A
U_1	D_1	F_1	NULL	U_1	D_1	S_1	NULL

Poichè il join successivo avviene su UD si ha effettivamente la full disjunction.
Ma perchè questo risultato? Si è detto che ciò che causa problemi sono le estensioni degli schemi, ma entrando più in dettaglio possiamo dire che sono le istanze di oggetto che, a seguito di un'estensione, provocano errori; infatti una tupla potrebbe acquisire valori nulli sugli attributi necessari per prossimi join.
Con le sorgenti vuote questo problema non si pone, in quanto non hanno tuple e quindi una eventuale loro estensione non provoca nessun effetto negativo. È

proprio per questo motivo che nel caso appena esposto si arriva ad un risultato corretto: L_3 viene effettivamente esteso con D, attributo del join successivo, ma nessuna tupla viene infuenzata da tale estensione.
Bisogna invece fare attenzione alle relazioni che vengono estese dalla sorgente vuota: per queste valgono le normali regole esposte precedentemente. Nell'esempio L_1 ed L_2 vengono estese da L_3 con A, che non è attributo di prossimo join e quindi si ottiene un risultato corretto.
Per questo motivo le relazioni vuote non devono sottostare ad un ordinamento. Infatti supponiamo che in figura 3.30 L_1 sia vuota e che $S(L_1) \neq S(L_2)$: scegliendo L_1 al primo passo dell'algoritmo avremmo che L_2 verrebbe estesa con l'insieme di attributi $X = \{S(L_1) - S(L_2)\}$, ma dato che i join di L_2 sono fomati da attributi contenuti in $S(L_2)$ abbiamo che nessuno di questi comprende attributi di X, dato che $X \nsubseteq S(L_2)$.

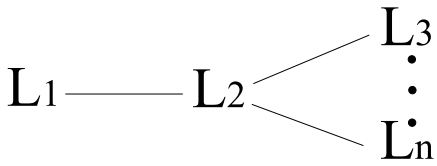


Figura 3.30: Grafo d'esempio

Ora riprendiamo l'algoritmo proposto in tabella 3.6 a pagina 96: alla luce di quanto detto in questa sezione il passo "seleziona $R_i \in \mathcal{R}_i$" non è più così banale; per questo è opportuno introdurre una funzione di ricerca che svolga tale compito.
Prima di tutto è necessario definire un insieme dei nodi che, se utilizzati, non portano ad una disconnessione del grafo. Chiameremo "ammissibili" tali nodi.

Definizione 3.5.8 (Insieme di nodi ammissibili) *Dato un grafo $\mathcal{G} = (V, E)$ con $V = \{R_1, \ldots, R_n\}$ e $E = \{(R_i, R_j) \mid R_i, R_j \in V, F_{ij} \neq NULL\}$ si definisce nodo ammissibile un nodo $R_i \in V$ se, essendo $RJ = R_{j1}, \ldots, R_{jn}$ l'insieme di tutti i nodi collegati direttamente a R_i, si riesce sempre a trovare un percorso in $\mathcal{G}' = (V', E')$, con $V' = V - \{R_i\}$ e $E' = \{(R_i, R_j) \mid R_i, R_j \in V', F_{ij} \neq NULL\}$, che colleghi tutti i componenti di RJ tra loro. Si definisce $\varphi_{\mathcal{G}}$ l'insieme di tutti i nodi ammissibili del grafo $\mathcal{G}$.*

La funzione dovrebbe, dato un grafo, selezionare i nodi del grafo R_i tali che per ogni nodo R_j con cui questi debbano andare in join si abbia che per ogni nodo R_k con cui R_j deve andare in join l'insieme degli attributi di join sia disgiunto dall'insieme degli attributi estesi di R_i ed R_j. Inoltre deve essere impedita una disconnessione del grafo. Più formalmente:

Definizione 3.5.9 (Funzione di ricerca) *Dato un grafo $\mathcal{G} = (V, E)$ con $V = \{R_1, \ldots, R_n\}$ e $E = \{(R_i, R_j) \mid R_i, R_j \in V, F_{ij} \neq NULL\}$ e dato l'insieme $\varphi_{\mathcal{G}}$ di tutti i nodi ammissibili di $\mathcal{G}$, si definisce funzione di ricerca $f_{ric}(\mathcal{G})$ una funzione che, dato $\mathcal{G}$, restituisce l'insieme $V' \subseteq V$ tale che:*

$$V' = \{R_i \in \varphi_{\mathcal{G}} \mid \forall (R_i, R_j) \in E, R_j \mid \forall (R_j, R_k) \in E, F_{jk} \cap ((S(R_i) \cup S(R_j)) - (S(R_i) \cap S(R_j))) = \emptyset\}$$

con $R_i, R_j, R_k \in V$

In più è necessario definire un algoritmo di selezione, che passo dopo passo scelga uno dei nodi restituiti dalla funzione di ricerca. Un algoritmo che realizza l'obiettivo è riportato in tabella 3.7.
Ora il passo "seleziona $R_i \in \mathcal{R}_i$" dell'algoritmo a pagina 96 si realizza semplicemente utilizzando la sequenza seq. Si tenga presente che, una volta fissate le relazioni e le condizioni di join tra esse, f_{sel} dovrà essere applicata solo una volta.

3.5.8 Confronto fra i due algoritmi

Ovviamente, l'output dell'algoritmo cambia. Ora non si ha più una pseudo-sequenza, bensì la Full Disjunction già calcolata.
Utilizzando questo metodo, i join vengono calcolati ad ogni iterazione, realizzando anche un'ottimizzazione rispetto all'algoritmo precedente.
Si ricorda infatti, come risulta chiaro dalla Figura 3.15, che alcuni join vengono inevitabilmente ripetuti.
Vediamo di calcolare quante sono queste ripetizioni nel caso più generale possibile: quello con n sorgenti. Si ricorda che l'algoritmo considera, in ambito SEWASIE, il grafo completamente connesso, quindi la scelta del nodo da utilizzare per ogni iterazione può essere considerata arbitraria (escludendo le sorgenti vuote).
All'inizio (si può dire di essere all'iterazione 0) si hanno n nodi, corrispondenti alle relazioni, e ovviamente nessuna ripetizione.
La situazione iniziale è illustrata in figura 3.31.

Dopo una iterazione si hanno $n - 1$ nodi e nessuna ripetizione, come si vede in figura 3.32.

Al termine della seconda iterazione, come risulta chiaro dalla figura 3.33, vi sono tante ripetizioni del join (*)$R_{n-1} =\bowtie= R_n$ quanti sono i nodi, ossia $n - 2$.

$f_{sel}(\mathcal{G}, R_i, seq)$

Input: * $\mathcal{G} = (V, E)$ grafo delle relazioni.
* $V = \{R_1, \ldots, R_n\}$: nodi del grafo (relazioni definite sullo schema globale).
* $E = \{(R_i, R_j) \mid R_i, R_j \in V, OJC(R_i, R_j) \neq NULL\}$: archi del grafo (join).
* $R_i =$ nodo selezionato.

Output: sequenza corretta seq **di nodi**

Procedimento:

- Poni $V_0 = V$, $E_0 = E$, $R_i = NULL$, seq=$NULL$
- se $R_i \neq NULL$
  - calcola tutti i join tra R_i ed R_j, con $R_j \in V$
  - elimina R_i da V
- se $\mid V \mid = 1$
  - return OK
- calcola $V' = f_{ric}(G)$
- ordina V' per cardinalità di join decrescenti
- while $V' \neq \emptyset$
  - prendi il primo $R_i \in V'$
  - se $f_{sel}(\mathcal{G}, R_i, seq)$=OK
    - aggiungi in testa a seq il nodo R_i
    - return OK
  - altrimenti
    - elimina R_i da V'
- return KO

Tabella 3.7: Algoritmo di selezione

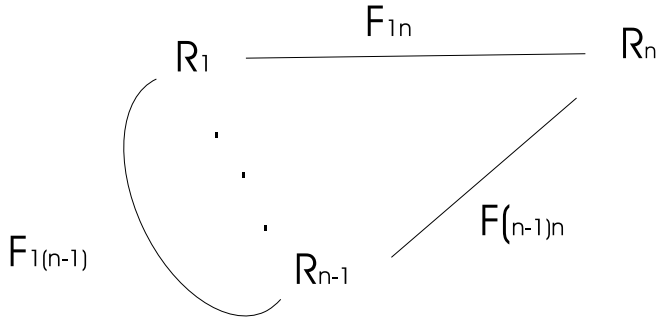


Figura 3.31: Stato iniziale

Dopo la terza iterazione, illustrata in figura 3.34, vi sono $n-3$ ripetizioni (sempre quante i nodi) del blocco (**)$(R_{n-2} =\bowtie= R_n) =\bowtie= (R_{n-1} =\bowtie= R_n)$, compo-

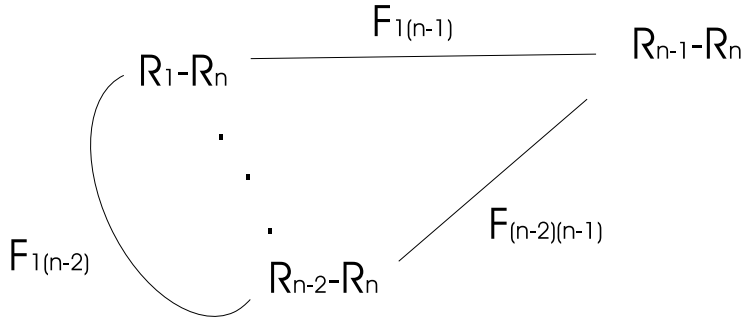


Figura 3.32: Dopo iterazione 1

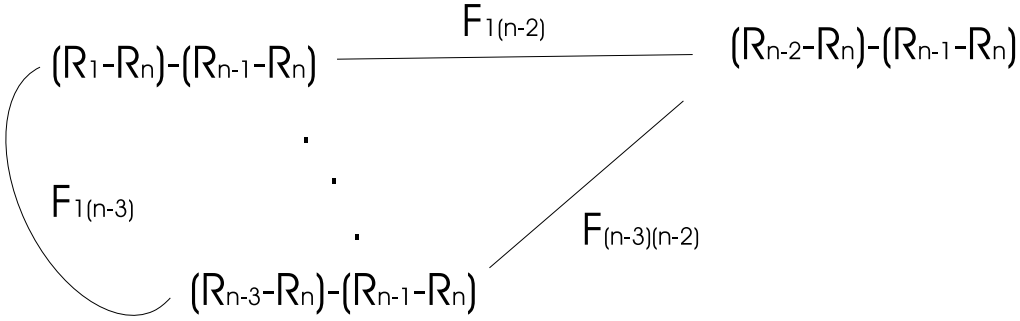


Figura 3.33: Dopo iterazione 2

sto da due join. Inoltre si ripete $n-3$ volte anche (*). Quindi si aggiungono altre $(n-3)*3$ ripetizioni.

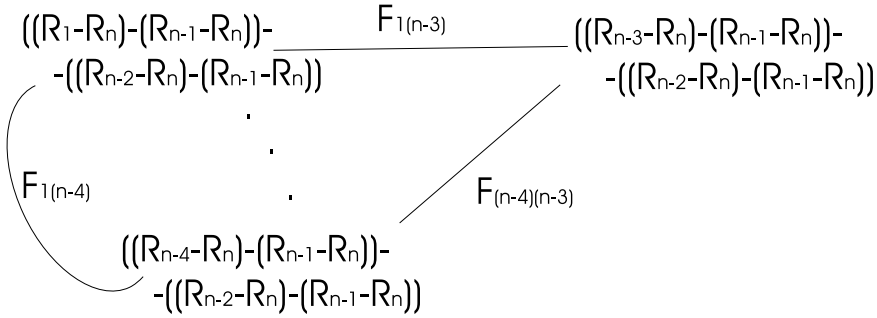


Figura 3.34: Dopo iterazione 3

Al termine della quarta iterazione, risultano $(n-4)*7$ ripetizioni, dovute al blocco (***)$[(R_{n-3} =\bowtie= R_n) =\bowtie= (R_{n-1} =\bowtie= R_n)] =\bowtie= [(R_{n-2} =\bowtie= R_n) =\bowtie=$

$(R_{n-1} =\bowtie= R_n)]$ presente per ognuno dei $n-4$ nodi e formato da quattro join e al fatto che (*) e (**) si ripetono $n-4$ volte, per un totale di altri tre join.
Per non appesantire troppo la figura 3.35, rappresentativa di questa iterazione, vengono utilizzati i simboli (*) e (**), con lo stesso significato utilizzato finora.

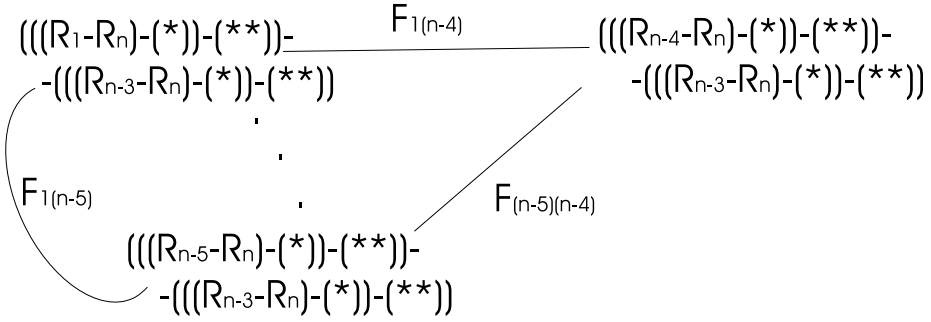


Figura 3.35: Dopo iterazione 4

Dopo la $i-sima$ iterazione, si ha che il numero di ripetizioni è:
$rip = (n-i) * (2^{i-1} - 1)$.
Dopo la penultima iterazione, ossia la numero $NP-1$, si hanno quindi
$rip = [n-(NP-1)] * (2^{(NP-1)-1} - 1)$ ripetizioni.
In figura 3.36 il simbolo (****) è generato dalla quinta iterazione ed ha stesso significato dei simboli (*), (**) e (***).

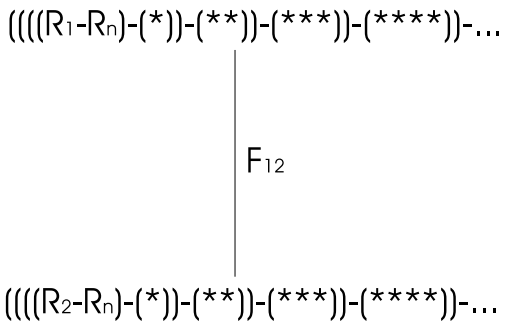


Figura 3.36: Dopo iterazione $NP-1$

Come si può intuitivamente dedurre dalla figura 3.36, l'ultima iterazione non è altro che un join fra due pseudo-sequenze. Per questo non vengono aggiunte altre

ripetizioni e quindi il numero delle ripetizioni finale è esattamente uguale a quello della penultima iterazione.

Da ciò che si è visto, nè la prima nè l'ultima iterazione producono delle ripetizioni. Ciò è in linea con quanto accade quando $n = 3$. In questo caso, come è facilmente verificabile, non vi sono ripetizioni proprio perché si hanno due sole iterazioni: la prima e l'ultima.
Formalmente, si ottiene che il numero di ripetizioni rip è pari a:

$$rip = \begin{cases} 0 & \text{se } NP < 3 \\ [n - (NP - 1)] * (2^{(NP-1)-1} - 1) & \text{se } NP \geq 3 \end{cases}$$

Dove n è il numero delle relazioni coinvolte nella Full Disjunction e NP è il numero di passi necessari all'algoritmo per ricavare la pseudo-sequenza.
Nella formula è stato lasciato il termine $n-(NP-1)$ nonostante sia fisso e pari a 2 per renderla più comprensibile. Alla luce di questo, la formula può essere vista come non dipendente esplicitamente dal numero delle sorgenti. Quindi si ha che $rip = rip(NP)$.

Utilizzando l'algoritmo modificato non si provocano tutte queste ripetizioni, in quanto i join vengono calcolati ad ogni passo e non al termine dell'algoritmo. Nonostante questa nota positiva, bisogna tenere in considerazione che alla generica iterazione i dell'algoritmo, la funzione di risoluzione va applicata $n - i$ volte, anche se su di un numero di colonne ridotto rispetto all'algoritmo precedente, secondo il quale la risoluzione andava attuata solamente al termine dell'algoritmo. In quest'ultimo caso infatti, le relazioni coinvolte nella risoluzione erano n, mentre ora se ne considerano due per volta. Il vantaggio risulterà sicuramente maggiore quando verrà rimossa la condizione di omogeneità semantica. Grazie a questa limitazione è necessario controllare i valori di tutte le colonne solamente nel caso abbiano tutte valore *NULL*; in caso contrario ci si può fermare non appena si trova un valore significativo, dato che i successivi saranno per ipotesi uguali a quello trovato.
Comunque il peso della funzione di risoluzione non è calcolabile, in quanto, con riferimento alla definizione 2.3.1, dipende in gran parte dalla funzione $g_A(X_1, X_2, \ldots, X_n)$, la cui complessità è legata al tipo di funzione che si intende attuare. Mantenendo l'ipotesi di omogeneità semantica (quindi non considerando la funzione g) si può supporre un netto miglioramento, data la semplicità della funzione di risoluzione.

Capitolo 4

La fusione dei dati in un contesto a mediatore

In questo capitolo si specializzeranno i concetti introdotti nei capitoli precedenti per un sistema a mediatore. Per comodità nel seguito si utilizzerà la stessa notazione presentata in [23] per l'ambiente MOMIS-SEWASIE, ma si tenga presente che i concetti esposti sono validi per un qualsiasi sistema a mediatore con le stesse caratteristiche (una classe globale e più classi locali, approccio GAV).

4.1 La funzione di risoluzione in ambito SEWASIE

Da questo punto in poi due attributi locali appartenenti a due relazioni diverse verranno detti *duplicati* se mappati sullo stesso attributo globale.
Come si era detto precedentemente, dato il risultato esatto è necessaria una ulteriore funzione per presentare all'utente un risultato più leggibile.
Allo stato attuale, il progetto SEWASIE fa l'ipotesi di omogeneità semantica delle tuple ed utilizza quindi per questo scopo una semplice funzione che, data in input una relazione con colonne duplicate, dà in output una relazione senza colonne duplicate, ma con lo stesso contenuto informativo. Più precisamente, la funzione controlla il contenuto di ogni colonna duplicata per ogni riga e:

- se viene trovato un valore non nullo, viene riportato nella relazione in output. Si noti che questa semplificazione è possibile proprio grazie all'ipotesi di omogeneità semantica.
- se tutti i valori sono nulli, verrà riportato NULL anche nella relazione in output.

Qui viene definita una versione più generale della funzione, anche in vista di una futura rimozione del vincolo di omogeneità semantica e quindi di una variazione

del criterio di selezione appena esposto.
La seguente funzione prende in input un generico insieme di valori per un determinato attributo. Per ottenere il risultato desiderato, sarà sufficiente applicarla un numero di volte pari al numero degli attributi.

Ora però sorge un problema all'applicazione di alcune delle funzioni di risoluzione proposte. La complicazione è dovuta alla struttura dell'algoritmo modificato in tabella 3.6.
Poiché gli outerjoin vengono eseguiti ad ogni passo, su due relazioni per volta, non si hanno informazioni di insieme sui valori degli attributi possibilmente conflittuali, quindi l'insieme delle funzioni utilizzabili, nel nostro caso, si riduce solamente a quelle che soddisfano la proprietà *associativa*.
L'associatività delle funzioni permette infatti di calcolare il risultato in maniera iterativa, aggiornando il valore passo dopo passo ma assicurando comunque la correttezza del valore finale. Questa, in situazioni normali, sarebbe da considerarsi solamente una proprietà auspicabile, ma non necessaria. Il discorso cambia nel nostro caso, per via della particolare struttura dell'algoritmo. Funzioni come la media, la varianza, la mediana o la deviazione standard non possono quindi essere applicate correttamente senza ricorrere a degli espedienti.
La media, ad esempio, può essere calcolata utilizzando ad ogni passo anche le funzioni SUM e COUNT (funzioni che invece sono associative) ed immagazzinando, in due attributi aggiuntivi, i risultati delle due funzioni. In questo modo, anche nei passi successivi, è possibile stabilire con esattezza quanti siano i valori che, fino a quel momento, sono stati considerati nella media. In alternativa è possibile utilizzare solamente due funzioni, dato che la terza può essere ricavata tramite la relazione

$$AVG = \frac{SUM}{COUNT}$$

Ben poco si può invece fare per tentare di implementare la mediana, in quanto la correttezza della funzione richiede necessariamente la conoscenza di tutti i valori in gioco.

A questo punto, dopo aver introdotto in maniera adeguata la funzione di risoluzione ed aver spiegato le problematiche relative alla fusione dei dati, è necessario mettere in luce un difetto presente nel precedente algoritmo, che risiede però non nell'implementazione dello stesso, quanto nell'operatore di full outerjoin.
Questo operatore da solo non permette la fusione di attributi provenienti da relazioni differenti in un'unica colonna. Quindi non si preoccupa della risoluzione dei

conflitti, poiché, di fatto, non ne può generare.
In più, come mostrato nel capitolo precedente, non offre in output una vista della fusione delle relazioni. Proprio per questo motivo è necessario affiancare all'operatore di full outerjoin una funzione di risoluzione che supplisca a questa lacuna.
In ambito di ricerca esiste già un approccio simile a quello adottato in questa sede, presentato in [4]. In quell'ambito si introduce un operatore detto di *Union-merge*; per definirlo è necessario introdurre un altro operatore:

Definizione 4.1.1 (Join-Merge $\sqcap$) *Siano $R = (A_1, \ldots, A_m)$ e $S = (A_1, \ldots, a_n)$ due relazioni con attributo identificativo A_1 in comune. Gli attributi $A_i, \ldots, A_m$ sono in comune ad entrambe le relazioni e sono mappati sullo stesso attributo sullo schema globale. Allora*

$$
\begin{aligned}
R \sqcap S := \{&\text{tupla } t \mid \exists r \in R, s \in S \text{ con} \\
&t[A_1] = r[A_1] = s[A_1], && (4.1)\\
&t[A_j] = r[A_j], j = 2, \ldots, i-1 && (4.2)\\
&t[A_j] = f_{j-i}(r[A_j], s[A_j]), j = 1, \ldots, m && (4.3)\\
&t[A_j] = s[A_j], j = m-1, \ldots, n\} && (4.4)
\end{aligned}
$$

Dove (4.1) è la condizione di join, (4.2) sono i valori forniti solo da R, (4.3) sono i valori potenzialmente in conflitto e (4.4) sono i valori forniti solo da S. Le $f_i(), i = 0, \ldots, m-i$ sono le funzioni di risoluzione, una per ogni attributo.
L'operatore di *Union-Merge* è un'estensione del join-merge. Questo operatore garantisce che ogni tupla di ogni fonte entri in join.

Definizione 4.1.2 (Union-Merge $\sqcup$) *Siano $R = (A_1, \ldots, A_m)$ e $S = (A_1, \ldots, a_n)$ due relazioni con attributo identificativo A_1 in comune. Gli attributi $A_i, \ldots, A_m$ sono in comune ad entrambe le relazioni e sono mappati sullo stesso attributo sullo schema globale. Allora:*

$$
\begin{aligned}
R \sqcup S := (&R \sqcap S)\\
&\cup (R \setminus (R \sqcap S)[R] \times \{(\perp_{m+1}, \ldots, \perp_n)\}\\
&\cup (S \setminus (R \sqcap S)[S] \times \{(\perp_2), \ldots, \perp_{i-1}\}
\end{aligned}
$$

Il seguente semplice esempio mette in luce quali siano i risultati delle esecuzioni dei due operatori su due tabelle.

r

A_1	A_2	A_3
1	2	$\perp$
2	5	$\perp$
3	$\perp$	z

s

A_1	A_3	A_4
1	x	g
2	y	$\perp$
3	x	i

$r \sqcap s$

A_1	A_2	A_3	A_4
1	2	x	g
3	$\perp$	$f_0(z, y)$	$\perp$

$r \sqcup s$

A_1	A_2	A_3	A_4
1	2	x	g
2	5	$\perp$	$\perp$
3	$\perp$	$f_0(z, y)$	$\perp$
4	$\perp$	x	i

Quindi, mentre il join-merge prende in considerazione solo le tuple che entrano in un natural join basato sull'attributo identificatore, l'union-merge porta nel risultato anche le tuple di una relazione che non hanno una corrispondenza nell'altra, esattamente come farebbe l'operatore di outerjoin. Si noti l'operazione di fusione delle colonne, non operata dal semplice outerjoin, e quella di risoluzione dei conflitti, non possible con il natural outerjoin.
Dalle definizioni 4.1.1 e 4.1.2 emerge una limitazione a questi due operatori: entrambi infatti assumono a priori che le tabelle da unificare abbiano un identificatore in comune, cosa estremamente rara soprattutto in ambiti come quello SEWASIE, dove le sorgenti locali non possono essere modificate e sono autonome. Quindi il join-merge e l'union-merge hanno un'applicazione minata da una condizione molto stringente, mentre la Full Disjunction non presenta questa limitazione e anzi il suo utilizzo ha senso soprattutto se le relazioni implicate non hanno un identificatore comune.

Come precedentemente affermato, per completezza è necessario ridefinire la funzione di risoluzione utilizzando la terminologia utilizzata fino ad oggi in ambito SEWASIE. Per favorire la comprensione, di seguito verranno riportate le terminologie necessarie per capire l'analogia fra le funzioni precedentemente introdotte e quelle che verranno di seguito. Per informazioni più approfondite, fare riferimento a [23] o ad altri documenti pubblicati precedentemente.

Si denota con $\mathcal{G}$ una classe globale con schema $S(\mathcal{G})$, con $\mathcal{L}$ l'insieme di classi locali di $\mathcal{G}$ e con $L \in \mathcal{L}$ una classe locale. Ogni schema di classe locale, denotato con $S(\mathcal{L})$, è l'insieme delle informazioni disponibili nella classe L, rappresentato come un insieme di attributi. Infine, si denota con $\mathcal{M}$ il mapping fra lo schema globale e i vari schemi locali. In particolare, si denota con $\mathcal{M}[A, L]$ il mapping dell'informazione $A \in S(\mathcal{G})$ nella classe locale L; si assume il valore *null* se A non è mappato in L. Il mapping soddisfa l'approccio global as view (GAV), ossia:

per ogni $A \in S(\mathcal{G})$ esiste almeno un $L \in \mathcal{L}$ tale che $\mathcal{M}[A, L] \neq null$. Data la classe locale $L \in \mathcal{L}$ si definisce lo schema di L rispetto a $\mathcal{G}$, denotato con $S^{\mathcal{G}}(L)$, come $S^{\mathcal{G}}(L) = \{A \in S(\mathcal{G}) \mid \mathcal{M}[A, L] \neq null\}$.
Un SINode rappresenta la conoscenza disponibile sugli oggetti del mondo reale istanziati nelle sorgenti locali. Si denota con $\mathcal{O}$ l'insieme di oggetti del mondo reale disponibili e si introduce una funzione ε, detta *estensione*, definita come $\varepsilon : \mathcal{L} \rightarrow 2^{\mathcal{O}}$, in grado di associare ogni classe locale L in $\mathcal{L}$ con il sottoinsieme di oggetti di $\mathcal{O}$ rappresentati in L.

Prendendo come riferimento un modello relazionale, si assume che ogni L sia il nome di una relazione con schema $S(L)$, mentre con l si denota una relazione locale. Data una estensione ε, l si dice legale rispetto a ε se e solo se esiste una funzione biunivoca $t_L : \varepsilon(L) \rightarrow l$, dove ogni oggetto o istanziato in L è rappresentato dalla tupla $t_L(o)$ e l corrisponde a $\{t_L(o) \mid o \in \varepsilon(L)\}$.

Per ogni classe locale L si considera un operatore unario $(r)^{\mathcal{G}}_L$ tale che data una relazione l con schema $S(L)$ produca una relazione $l^{\mathcal{G}} = (r)^{\mathcal{G}}_L$ con schema $S^{\mathcal{G}}(L)$ ottenuto da:

1. *Traduzione dello schema*: si rinominano gli attributi di r in attributi di $\mathcal{G}$ tramite il mapping $\mathcal{M}$

2. *Conversione dei dati*: si convertono le tuple di r in tuple di $r^{\mathcal{G}}$ tramite apposite funzioni, come ad esempio la concatenazione di stringhe.

Una tupla di $l^{\mathcal{G}}$ viene denotata con $t^{\mathcal{G}}_L(o)$. Dato l'insieme di classi locali $\mathcal{L} = \{L_1, \ldots, L_n\}$, si introduce la nozione di *database di sorgenti legale rispetto a* ε come $db = \langle l^{\mathcal{G}}_1, \ldots, l^{\mathcal{G}}_n \rangle$.

Si definisce una *condizione di join* fra ogni coppia di relazioni che presentano sovrapposizione tale da poter identificare e fondere tuple corrispondenti allo stesso oggetto reale. La condizione di join è definita su $S(\mathcal{G})$.
Ad esempio, sia $L_1.A_1 = L_2.A_1$ la condizione indicata dal progettista nel mapping fra L_1 ed L_2; dati $o_i \in \varepsilon(L_1)$ e $o_j \in \varepsilon(L_2)$, si ha che $t^{\mathcal{G}}_{L_1}(o_i)[A_1] = t^{\mathcal{G}}_{L_2}(o_j)[A_2]$ se e solo se o_i ed o_j sono lo stesso oggetto reale.
Più formalmente:

Definizione 4.1.3 (Condizione di join) *Sia $\mathcal{G}$ una classe globale e sia $\mathcal{L}$ il corrispondente insieme di classi locali. Una* condizione di join *fra $L_1, L_2 \in \mathcal{L}$ è una condizione ϕ espressa su $S^{\mathcal{G}}(L_1) \cup S^{\mathcal{G}}(L_2)$ tale che $\forall \varepsilon$, $\forall t_1 \in l^{\mathcal{G}}_1$, $\forall t_2 \in l^{\mathcal{G}}_2$, $\phi(t_1, t_2)$ è vera se e solo se $\exists o \in \varepsilon(L_1) \cap \varepsilon(L_2)$ tale che $t_1 = t_{L_1}(o)$ e $t_2 = t_{L_2}(o)$.*

Analizzando la definizione, si nota come la condizione di join abbia una connotazione molto generale. Infatti questa è definita come una generica funzione in grado di fondere tuple rappresentanti lo stesso oggetto reale; questo significa che

sono ammesse anche soluzioni diverse dal semplice equijoin.
La necessità di una funzione generica è dettata dalla necessità di dover riconoscere gli oggetti. Infatti, come visto in sezione 2.5.1, alcune tuple possono presentare valori di attributi diversi anche rappresentando lo stesso oggetto reale.
D'altra parte, il vincolo che stabilisce quando il risultato della funzione è vero equivale a dire: $\phi(t_1, t_2)$ è vera se e solo se t_1 e t_2 rappresentano lo stesso oggetto. Questa assunzione pone alcune limitazioni:

1. Per prima cosa, dall'assunzione deriva che la relazione in uscita non deve presentare tuple separate rappresentanti lo stesso oggetto reale; per realizzare ciò è necessaria una fase di riconoscimento degli oggetti. Nell'ambito SEWASIE questa fase ha luogo con il join tra le relazioni, dato che si è operata la scelta di eseguire un riconoscimento basato esclusivamente sui valori degli attributi. Dal momento che nella stessa relazione di base potrebbero esistere due tuple separate rappresentanti lo stesso oggetto, il join da solo può non bastare e quindi sulla relazione in uscita è necessario applicare una ulteriore fase di riconoscimento per eliminare questi duplicati.

2. In secondo luogo, l'assunzione non ammette falsi positivi o falsi negativi. Questo significa che la ricerca di tuple duplicate deve essere effettuata perfettamente. Ovviamente ciò è solo un punto di riferimento ideale, dato che, come precedentemente mostrato, questa condizione è estremamente difficile da realizzare.

A questo punto è possibile ridefinire le funzioni di risoluzione secondo la terminologia appena introdotta. Per comodità e per coerenza riguardo a quanto mostrato nel nuovo algoritmo, le seguenti funzioni prevedono il join solamente fra due relazioni.

Definizione 4.1.4 (Funzione di risoluzione in SEWASIE) *Siano L_1, L_2 due classi locali e siano l_1, l_2 relazioni con schema $S(L_1), S(L_2)$ rispettivamente. Sia $\mathcal{M}$ il mapping tra le classi locali e la classe globale. Sia $A_j \in S(l_1^{\mathcal{G}}) \cup S(l_2^{\mathcal{G}})$. Sia D_j il dominio di A_j e sia $D_j^+ = D_j \cup \perp$. Sia $t_{L_i}(o)$ una generica tupla di l_i, con $i = 1, 2$, facente riferimento all'oggetto reale o. Sia $t_{L_i}(o)[\mathcal{M}[A_j, L_i]]$ il valore assunto dalla relazione l_i, con $i = 1, 2$, per l'attributo A_j e sia $t^{\mathcal{G}}_{L_i}(o)[A_j]$ la rispettiva tupla di $l_i^{\mathcal{G}}$, con $i = 1, 2$. Allora si definisce* funzione di risoluzione *$f_{rA_j} = f_{rA_j}(t^{\mathcal{G}}_{L_i}(o)[A_j]) : {D_j^+}^2 \rightarrow D_j^+$ con $i = 1, 2$:*

$$t^{\mathcal{G}}(o)[A_j] = \begin{cases} \perp & \text{se } \forall i \mid A_j \in S^{\mathcal{G}}(L_i)\ \ t^{\mathcal{G}}_{L_i}(o)[A_j] = \perp \\ t^{\mathcal{G}}_{L_i}(o)[A_j] & \text{se } \exists i \mid t^{\mathcal{G}}_{L_i}(o)[A_j] \neq \perp \wedge \\ & \quad \forall k \neq i (t^{\mathcal{G}}_{L_k}(o)[A_j] = \perp \vee t^{\mathcal{G}}_{L_k}(o)[A_j] = t^{\mathcal{G}}_{L_i}(o)[A_j]) \\ & \text{con } L_i, L_k \mid A_j \in S^{\mathcal{G}}(L_i) \\ g_{A_j}(t^{\mathcal{G}}_{L_i}(o)[A_j]) & \text{altrimenti} \end{cases}$$

Dove $g_{A_j}(t^{\mathcal{G}}_{L_i}(o)[A_j]) : D_j^{+^2} \rightarrow D_j^+$

La definizione è stata introdotta in modo da mantenerne la struttura il più possibile vicina a quella in definizione 2.3.1. È però possibile introdurre alcune semplificazioni per rendere la funzione ancora più vicina al nostro contesto:

- si può evitare di porre ogni volta la condizione $\forall i \mid A_j \in S^{\mathcal{G}}(L_i)$ assumendo che le entrambe le L_i abbiano come schema $S(\mathcal{G})$ con gli attributi di $S(\mathcal{G}) - S^{\mathcal{G}}(L_i)$ posti a $\perp$.
- in linea di principio, la seconda condizione potrebbe essere meno articolata; si potrebbe prevedere di fare ricorso alla funzione di risoluzione ogni volta che si riscontrino due valori significativi, senza considerare se questi siano uguali o meno.

La prossima funzione verrà ridefinita tenendo conto di queste due osservazioni; inoltre, per comodità, si ipotizza che siano necessari tutti gli altri attributi per risolvere i conflitti, anche se ciò non rispecchia la realtà dei fatti.

Definizione 4.1.5 (Funzione di risoluzione generalizzata in SEWASIE) *Siano* L_1, L_2 *due classi locali, siano* l_1, l_2 *relazioni con schema* $S(L_1), S(L_2)$ *rispettivamente. Sia* $\mathcal{M}$ *il mapping tra le classi locali e la classe globale. Sia* $A_j \in S(l_1^{\mathcal{G}}) \cup S(l_2^{\mathcal{G}})$. *Sia* D_j *il dominio di* A_j *e sia* $D_j^+ = D_j \cup \perp$. *Sia* $t_{L_i}(o)$ *una generica tupla di* l_i, *con* $i = 1, 2$, *facente riferimento all'oggetto reale o. Sia* $t_{L_i}(o)[\mathcal{M}[A_j, L_i]]$ *il valore assunto dalla relazione* l_i, *con* $i = 1, 2$, *per l'attributo* A_j *e sia* $t^{\mathcal{G}}_{L_i}(o)[A_j]$ *la rispettiva tupla di* $l_i^{\mathcal{G}}$, *con* $i = 1, 2$. *Sia* $E^+ = \times_k E_k^+$ *il prodotto cartesiano dei domini di altri attributi di dominio* E_k *e sia* $t^{\mathcal{G}}_{L_i}(o)[A_k]$ *con* $i = 1, 2\ k = 1, \ldots, m\ k \neq j$ *il valore dell'attributo* A_k *per la tupla* $t_{L_i}(o)$ *della relazione* L_i. *Allora si definisce* funzione di risoluzione $f_{rA_j} = f_{rA_j}(t^{\mathcal{G}}_{L_i}(o)[A_j], t^{\mathcal{G}}_{L_i}(o)[A_k]) : (D_j^+ \times E^+)^2 \rightarrow D_j^+$

$$t^{\mathcal{G}}(o)[A_j] = \begin{cases} \perp & \textit{se } \forall i \ \ t^{\mathcal{G}}_{L_i}(o)[A_j] = \perp \\ t^{\mathcal{G}}_{L_i}(o)[A_j] & \textit{se } \exists i \mid t^{\mathcal{G}}_{L_i}(o)[A_j] \neq \perp \wedge \forall j \neq i \ t^{\mathcal{G}}_{L_i}(o)[A_j] = \perp \\ g_{A_j}(t^{\mathcal{G}}_{L_i}(o)[A_j], t^{\mathcal{G}}_{L_i}(o)[A_k]) & \textit{altrimenti} \end{cases}$$

Dove $g_{A_j}(t^{\mathcal{G}}_{L_i}(o)[A_j], t^{\mathcal{G}}_{L_i}(o)[A_k]) : (D^+ \times E^+)^2 \rightarrow D^+$

La funzione ridefinita accetta in input relazioni del tipo $l_i^{\mathcal{G}}$; questo perché si assume che i problemi concernenti l'eterogeneità degli schemi siano stati precedentemente risolti.
È ancora possibile introdurre un miglioramento nella funzione di risoluzione generalizzata. Supponiamo di dover risolvere i conflitti relativi all'attributo *prezzo*. Supponiamo anche che a tale attributo sia stata associata una funzione di risoluzione che necessiti del valore dell'attributo *data* per giungere ad un risultato

corretto. Seguendo l'algoritmo modificato si ha che per trovare il *prezzo* corretto è necessario controllare i valori di *data* presenti nelle due tuple, prendendo poi per buono il valore di *prezzo* corrispondente, ad esempio, alla *data* più recente. Questo in realtà equivale a risolvere anche i conflitti relativi alla *data*; perciò, se l'attributo *data* fosse già stato trattato in precedenza, tutto il procedimento si rivelerebbe solamente una ripetizione. Perciò è più appropriato ricorrere agli attributi già risolti piuttosto che a quelli mantenuti sulle relazioni locali. In questo modo la funzione diverrebbe:

$$f(t^{\mathcal{G}}_{L_i}(o)[A_j], t^{\mathcal{G}}_{L_i}(o)[A_k], t^{\mathcal{G}}(o)[A_h]) : (D^+_j \times E^+)^2 \rightarrow D^+_j$$
$$con \quad i = 1, 2 \quad h = 1, \ldots, j-1 \quad k = j+1, \ldots, m$$

Supponendo che gli attributi vengano risolti in ordine.
Rimane comunque il problema inverso: utilizzando gli stessi attributi di prima, se si risolve prima *prezzo* e poi *data*, quest'ultimo attributo viene risolto due volte, ottenendo una ripetizione. Per ovviare all'inconveniente si potrebbe pensare di risolvere *data* mentre si risolve *prezzo*. Ciò è possibile, genericamente, richiamando ricorsivamente la funzione di risoluzione quando un attributo necessario alla decisione non è ancora stato risolto.
Nonostante nella pratica ciò si verifichi molto raramente, in teoria si potrebbero verificare delle dipendenze cicliche fra le funzioni di risoluzione degli attributi; cioè: l'attributo A_l necessita del valore dell'attributo A_m per essere risolto e viceversa. In questo caso è necessario rompere il ciclo, tentando di risolvere uno dei due attributi in un modo alternativo.

Sotto l'ipotesi di omogeneità semantica, data un'estensione ε ed un database di sorgenti semanticamente omogeneo $db = \langle l^{\mathcal{G}}_1, \ldots, l^{\mathcal{G}}_n \rangle$ legale rispetto a ε, l'*istanza globale* g si derivava nel seguente modo: $g = \{t_{\mathcal{G}}(o) \mid o \in \mathcal{O}\}$ dove $t_{\mathcal{G}}$ è ottenuto applicando la proprietà di omogeneità semantica:

$$\forall A \in S(\mathcal{G}) : t_{\mathcal{G}}(o)[A] = \begin{cases} t^{\mathcal{G}}_L(o)[A] & se \exists L \in \mathcal{L} : A \in S^{\mathcal{G}}(\mathcal{L}), o \in \varepsilon(L) \\ null & altrimenti \end{cases}$$

Per quanto riguarda il concetto di full disjunction , dalla definizione originale si ha che lo schema è $S^{\mathcal{G}}(L_1) \times \cdots \times S^{\mathcal{G}}(L_n)$. Sulla base della proprietà di omogeneità semantica si ha che:

$$\forall t \in fd, \forall A \in S^{\mathcal{G}}(L_i) \cap S^{\mathcal{G}}(L_j) \;\; se \;\; t[L_i.A] \neq null \wedge t[L_j.A] \neq null \;\; then \;\; t[L_i.A] = t[L_j.A]$$

tale proprietà vale per ogni join map di $\mathcal{G}$.
Dalla full disjunction si può derivare una relazione con con schema $S(\mathcal{G})$ chiamata *full disjunction globale*, denotata con $GFD(l^{\mathcal{G}}_1, \ldots, l^{\mathcal{G}}_n)$, nel modo seguente:

$\forall t \in fd$, una tupla t' di $GFD(l_1^{\mathcal{G}}, \ldots, l_n^{\mathcal{G}})$ è definita da:

$$\forall A \in S(\mathcal{G}), t'[A] = \begin{cases} t[L.A] & \text{se } \exists L \in \mathcal{L} : A \in S^{\mathcal{G}}(L) \wedge t[L_j.A] \neq null \\ null & \text{altrimenti} \end{cases} \tag{4.5}$$

Si può dimostrare che

$$g = GFD(l_1^{\mathcal{G}}, \ldots, l_n^{\mathcal{G}})$$

Con l'introduzione della funzione di risoluzione i precedenti concetti devono essere ridefiniti:

Definizione 4.1.6 (Struttura di mapping della classe globale) *Una* struttura di mapping della classe globale *é una quadrupla* $(\mathcal{G}, \mathcal{L}, \mathcal{M}, JM)$ *dove* $\mathcal{G}$ *è la classe globale con schema* $S(\mathcal{G})$*,* $\mathcal{L}$ *è l'insieme di classi locali di* $\mathcal{G}$*,* $\mathcal{M}$ *il mapping di* $\mathcal{G}$ *e* JM *la join map di* $\mathcal{G}$*.*

Definizione 4.1.7 (Istanza globale) *Data una struttura di mapping della classe globale* $(\mathcal{G}, \mathcal{L}, \mathcal{M}, JM)$ *e dato un database di sorgenti* $db = \langle l_1^{\mathcal{G}}, \ldots, l_n^{\mathcal{G}} \rangle$*, sia* $fd = FD(l_1^{\mathcal{G}}, \ldots, l_n^{\mathcal{G}})$*. L'istanza globale* g *di* $\mathcal{G}$ *è una relazione con schema* $S(\mathcal{G})$ *definita come segue:*

$$g = \{t \mid t' \in fd, t[A] = f_{rA}(t'), \forall A \in S(\mathcal{G})\}$$

Anche l'algoritmo può essere ridefinito per il nostro ambito; il risultato è mostrato in tabella 4.1.
La funzione di ricerca f_{ric} di pagina 118 può essere così ridefinita:

Definizione 4.1.8 (Funzione di ricerca in ambito SEWASIE) *Siano* $L_1, \ldots, L_n$ *classi locali e siano* $l_1, \ldots, l_n$ *relazioni con schema rispettivamente* $S(L_1), \ldots, S(L_n)$*. Dato un grafo* $\mathcal{G} = (V, E)$ *con* $V = \{l_1, \ldots, l_n\}$ *e* $E = \{(l_i, l_j) \mid l_i, l_j \in V, JM(l_i, l_j) \neq NULL\}$ *e dato l'insieme* $\varphi_{\mathcal{G}}$ *di tutti i nodi ammissibili di* $\mathcal{G}$*, si definisce funzione di ricerca* $f_{ric}(\mathcal{G})$ *una funzione che, dato* $\mathcal{G}$*, restituisce l'insieme* $V' \subseteq V$ *tale che:*

$$V' = \{l_i \in \varphi_{\mathcal{G}} \mid \forall (l_i, l_j) \in E, l_j \mid \forall (l_j, l_k) \in E, F_{jk} \cap ((S^{\mathcal{G}}(L_i) \cup S^{\mathcal{G}}(L_j)) - (S^{\mathcal{G}}(L_i) \cap S^{\mathcal{G}}(L_j))) = \emptyset\}$$

con $L_i, L_j, L_k \in V$

mentre la funzione di selezione f_{sel} non viene riportata perchè la sua ridefinizione è molto simile a quella dell'algoritmo per il calcolo della full disjunction.

Calcolo della Full Disjunction

Input: * $db = \{l_1^{\mathcal{G}}, \ldots, l_n^{\mathcal{G}}\}$: database di sorgenti legali rispetto a ε.
* $JM = \{F_{i,j}\}$: Join Table per tutte le relazioni di db.
* $\mathcal{F}_r = \{f_{r1}, \ldots, f_{rm}\}$, insieme delle funzioni di risoluzione; una per ogni attributo.
Output: Full Disjunction
Procedimento:
• Poni $F_0 = JM$, $db_0 = db$ ed $NP = (n-1)$ numero dei passi.
• for ($i = 0; i < NP; i++$)
  • seleziona $l_i^{\mathcal{G}} \in db_i$
    $db_{i+1} = \{l_i^{\mathcal{G}} =\bowtie_f= l_i'^{\mathcal{G}} \mid l_i'^{\mathcal{G}} \in db_i; l_i'^{\mathcal{G}} \neq l_i^{\mathcal{G}}; f = OJC_i(l_i^{\mathcal{G}}, l_i'^{\mathcal{G}})$ non è null$\}$
    • $\forall$ **join presente in** db_{i+1}
      calcola il join
      • $\forall$ **tupla del join**
        • $\forall$ **attributo** j **della tupla**
          applica la funzione di risoluzione f_{rj}
• Restituisci db_{NP}.

Tabella 4.1: Algoritmo PSOJ in ambito SEWASIE

4.1.1 I conflitti sugli attributi di join

Ora che è stata definita la condizione di join in ambito SEWASIE è necessario parlare di un problema di difficile risoluzione; fino a questo momento tutti gli attributi sono stati considerati possibili fonti di conflitti e quindi sono stati trattati tutti allo stesso modo. Purtroppo però alcuni di questi necessitano di un trattamento particolare, poiché possono causare un numero maggiore di problemi: si tratta degli attributi di join.
Supponiamo di essere nella seguente situazione:

Persona (L_1)

Matr	CF	Nome
α	CFZ	Mario
β	CFW	Luigi

Tasse_Universitarie (L_2)

Matr	Cod_T	Tax_Uni
α	4	100

Tasse_Statali (L_3)

Cod_T	CF	Tax_Stat
4	CFW	50

supponiamo per comodità che gli attributi locali siano mappati su attributi globali con lo stesso nome e supponiamo anche che le condizioni di join siano:
L_1.Matr = L_2.Matr
L_1.CF = L_3.CF
L_2.Cod_Tax = L_3.Cod_Tax
Per vedere dov'è esattamente il problema tentiamo di fondere le relazioni: la prima tupla di L_1 andrebbe in join con quella di L_2, che a sua volta si potrebbe congiungere con quella di L_3. Quindi sembrerebbe che queste tuple rappresentino diverse istanze dello stesso oggetto reale. Se però il primo join ad essere eseguito fosse quello tra L_1 ed L_3 sarebbe la seconda tupla di L_1 ad entrare in join. Quindi, a seconda del join che viene eseguito per primo, le stesse istanze vengono riconosciute come parti di diversi oggetti reali.
Per capire quali sarebbero le vere connessioni tra i fatti è necessario ricorrere alla full disjunction, quindi ne applichiamo pedissequamente la definizione: prima eseguiamo un unione tra le relazioni e poi eliminiamo le tuple sussunte.

$L_1 \uplus L_2 \uplus L_3$

L_1			L_2			L_3		
Matr	CF	Nome	Matr	Cod_Tax	Tax_Uni	Cod_Tax	CF	Tax_Stato
α	CFZ	Mario	α	4	100	$\perp$	$\perp$	$\perp$
$\perp$	$\perp$	$\perp$	α	4	100	4	CFW	50
β	CFW	Luigi	$\perp$	$\perp$	$\perp$	4	CFW	50

Dopo l'unione non risultano tuple sussunte. Ora fondiamo le colonne che rappresentano gli stessi attributi:

FD

Matr	CF	Nome	Cod_Tax	Tax_Uni	Tax_Stato
α	CFZ	Mario	4	100	$\perp$
α	CFW	$\perp$	4	100	50
β	CFW	Luigi	4	$\perp$	50

Corrispondendo alla full disjunction, questo risultato contiene tutte le possibili connessioni tra i fatti.
Vediamo cosa succede con il nostro algoritmo nei tre casi che hanno origine scegliendo sempre un diverso nodo di partenza.

- iniziando da L_1

FD

Matr	CF	Nome	Cod_Tax	Tax_Uni	Tax_Stato
$g(\alpha, \beta)$	g(CFW,CFZ)	g(Mario,Luigi)	4	100	50
α	CFZ	Mario	$\perp$	$\perp$	$\perp$
β	CFW	Luigi	$\perp$	$\perp$	$\perp$

- iniziando da L_2

FD

Matr	CF	Nome	Cod_Tax	Tax_Uni	Tax_Stato
α	CFZ	Mario	4	100	$\perp$
$g(\alpha, \beta)$	CFW	Luigi	4	100	50

- iniziando da L_3

FD

Matr	CF	Nome	Cod_Tax	Tax_Uni	Tax_Stato
α	g(CFW,CFZ)	Mario	4	100	50
β	CFW	Luigi	4	$\perp$	50

dove $g(x, y)$ indica che è necessaria una funzione di risoluzione tra i valori x e y per stabilire il valore finale dell'attributo.
Come si può chiaramente vedere i tre risultati differiscono tra di loro, a volte anche nel numero di tuple ottenute; in più, e questa è la notizia peggiore, nessuno rispecchia la full disjunction calcolata precedentemente. L'ipergrafo degli schemi è β-ciclico e quindi non trattabile dal nostro algoritmo, ma non è questa la fonte del problema. Come detto in precedenza ciò che disturba sono le inconsistenze sui valori degli attributi di join. In particolare se nell'esempio l'attributo CF di

L_3 avesse avuto il valore CFZ invece che CFW l'attribuzione delle istanze agli oggetti reali sarebbe stata univoca.
Seguendo la definizione di condizione di join in ambito SEWASIE è chiaro come due istanze possano andare in join se e solo se rappresentano lo stesso oggetto reale; avendo basato il riconoscimento solamente sul valore degli di join non siamo in grado di superare questo problema. Anche l'applicazione della funzione di risoluzione su attributi di join non avrebbe senso: se due attributi di join sono diversi è lecito supporre che si tratti di istanze di oggetti diversi e non di istanze dello stesso oggetto ma con errori sull'attributo di join.
Per questo motivo nel nostro ambito supporremo che gli attributi di join siano esenti da conflitti e che quindi un'applicazione della funzione di risoluzione su tali attributi sia superflua.
Con questa assunzione stiamo supponendo una consistenza dei join definita sulle istanze. In altre parole: date tre tuple t_1, t_2 e t_3 appartenenti rispettivamente alle relazioni L_1, L_2 e L_3 sia $JC(L_i, L_j)$ la condizione di join tra le relazioni L_i e L_j. Se $JC(L_1, L_2)$ è vera per t_1 e t_2 e $JC(L_1, L_3)$ è vera per t_1 e t_3 allora la join map si dice join consistent se e solo se $JC(L_2, L_3)$ è vera per t_2 e t_3.
Meno formalmente: se t_1 e t_2 sono istanze dello stesso oggetto e t_1 e t_3 sono istanze dello stesso oggetto, allora anche t_2 e t_3 devono essere istanze dello stesso oggetto.

Per tentare di risolvere il problema dell'inconsistenza si può agire come in [14], dove le inconsistenze non vengono risolte e vengono portati in output solamente i dati consistenti.
Se invece si vuole tentare di risolvere le inconsistenze è necessario ricorrere ad informazioni aggiuntive, come attributi comuni ma non di join, considerando approcci come quelli in [20], [8], [38] o [24]. In particolare quest'ultimo (già visto a pagina 50) sembra adattarsi bene al nostro approccio. Il compito della funzione di conversione è svolto nel nostro ambito dalla mapping table, quindi rimangono solo due passi da compiere. Per la fase del confronto può essere usato ogni tipo di funzione, mentre è necessaria la classificazione in due classi: "vero" e "falso". Poichè le tuple sono confrontate due a due si può applicare il concetto appena esposto della consistenza dei join. Se t_1 e t_2 vengono classificate nella classe "vero" e t_1 e t_3 vengono classificate nella classe "vero", allora per avere consistenza t_2 e t_3 devono essere classificate nella classe "vero". In caso contrario si può decidere se non presentare all'utente i dati inconsistenti o tentare di risolvere i conflitti.

4.1.2 Una possibile ottimizzazione

L'introduzione della funzione di risoluzione in ambito SEWASIE porta sicuramente dei vantaggi, dato che la rimozione del vincolo di omogeneità semantica

avvicina alla situazione reale, ma c'è anche il rovescio della medaglia, costituito dalla maggiore complessità computazionale richiesta. Infatti, prendendo l'algoritmo così com'è è necessario applicare la funzione di risoluzione ad ogni passo su ogni attributo non di join; questa però è una soluzione semplicistica, che può essere migliorata.
Prima di tutto è necessario ricordare perché la funzione di risoluzione deve essere applicata ad ogni passo, invece che solo alla fine. Questo è dovuto alla necessità di unire le colonne che rappresentano lo stesso attributo in vista di un possibile join che le coinvolga. Nel nostro ambiente però abbiamo escluso l'applicazione della funzione su attributi di join e quindi di fatto questa esigenza non si pone più. Per questo motivo è possibile servirsi della funzione di risoluzione solamente quando serve, ossia solamente una volta alla fine del processo. In questo modo non solo l'algoritmo diventa più veloce, in quanto ci sono meno richiami alla funzione di risoluzione, ma diviene anche possibile utilizzare direttamente una funzione di risoluzione *non associativa* senza ricorrere ad altri artifici. Questo perché tutti i valori possibili vengono mantenuti fino al termine dell'algoritmo ed utilizzati tutti contemporaneamente.
Nell'eventualità che in prossimi sviluppi si decida di applicare la funzione di risoluzione anche sugli attributi di join, può essere ipotizzata anche per questi ultimi una miglioria: non tutti infatti vengono utilizzati ad ogni passo, quindi è possibile applicare su di loro la funzione di risoluzione solamente appena prima di un join che li riguardi e non ad ogni passo come precedentemente veniva fatto. In più è necessaria un'ultima applicazione al termine del procedimento.
Con queste ottimizzazioni il corpo dell'algoritmo cambia leggermente nella parte iterativa; il risultato (comprendente anche l'ottimizzazione per gli attributi di join) è mostrato in tabella 4.2.

Calcolo della Full Disjunction

Input: * $db = \{l_1^{\mathcal{G}}, \ldots, l_n^{\mathcal{G}}\}$: database di sorgenti legali rispetto a ε.
* $JM = \{F_{i,j}\}$: Join Table per tutte le relazioni di db.
* $\mathcal{F}_r = \{f_{r1}, \ldots, f_{rm}\}$, insieme delle funzioni di risoluzione; una per ogni attributo.
Output: Full Disjunction
Procedimento:
• Poni $F_0 = JM$, $db_0 = db$ ed $NP = (n-1)$ numero dei passi.
• for ($i = 0; i < NP; i++$)
  • seleziona $l_i^{\mathcal{G}} \in db_i$
    $db_{i+1} = \{l_i^{\mathcal{G}} =\bowtie_f= l_i'^{\mathcal{G}} \mid l_i'^{\mathcal{G}} \in db_i; l_i'^{\mathcal{G}} \neq l_i^{\mathcal{G}}; f = OJC_i(l_i^{\mathcal{G}}, l_i'^{\mathcal{G}})$ non è null$\}$
    • $\forall$ join presente in db_{i+1}
      • if($i > 0$)
        • $\forall$ tupla del join
          • $\forall$ attributo j della tupla coinvolto nel join
            applica la funzione di risoluzione f_{rj}
      calcola il join
• $\forall$ tupla della relazione
  • $\forall$ attributo j della tupla
    applica la funzione di risoluzione f_{rj}
• Restituisci db_{NP}.

Tabella 4.2: Algoritmo PSOJ in ambito SEWASIE

4.2 La riscrittura della query in ambito SEWASIE

L'introduzione della funzione di risoluzione e la conseguente rimozione del vincolo di omogeneità semantica rendono necessari alcuni interventi sulla tecnica di riscrittura della query finora utilizzata in ambito SEWASIE. Per permettere una maggiore comprensione, prima di introdurre le modifiche e le osservazioni viene introdotto l'attuale metodo in uso. Per ulteriori approfondimenti si rimanda a [23].

Si assume di avere un'espressione globale di tipo SQL sullo schema globale $\mathcal{G}$ avente la forma:

$$\text{select } AL \text{ from } \mathcal{G} \text{ where } Q \tag{4.6}$$

dove $AL \subseteq S(\mathcal{G})$ è una lista di attributi e Q è una condizione posta come un'espressione booleana di predicati atomici positivi aventi la forma ($A\ op\ valore$) o ($A\ op\ A'$) con $A, A' \in S(\mathcal{G})$. Per esempio considerando la seguente query:

select Nome from G
where Nome like 'P*' and (Anno='1' or Dip='Dip1')

si ha AL={Nome} e Q="Nome like 'P*' and (Anno='1' or Dip='Dip1')".
Data un'estensione ε, un database di sorgenti semanticamente omogenee $db = \langle l_1^{\mathcal{G}}, \ldots, l_n^{\mathcal{G}} \rangle$ ed una relazione globale g legale rispetto a ε, si può definire la risposta ad una query nella forma di 4.6 come

$$\pi_{AL}(\sigma_Q(g))$$

Dal momento che $g = GFD(l_1^{\mathcal{G}}, \ldots, l_n^{\mathcal{G}})$ si può scrivere

$$\pi_{AL}(\sigma_Q(GFD(l_1^{\mathcal{G}}, \ldots, l_n^{\mathcal{G}})))$$

Nel contesto SEWASIE il problema della riscrittura della query consiste nel riscrivere questa espressione nella seguente forma equivalente:

$$\pi_{AL}(\sigma_{Q_r}(GFD((lq_1)_{L_1}^{\mathcal{G}}, \ldots, (lq_n)_{L_n}^{\mathcal{G}})))$$

dove $lq_i = \pi_{AL_i}(\sigma_{LQ_i}(l_i))$ è la risposta alla query locale per la classe locale L_i e Q_r è la condizione residua.
Dal momento che un SINode non possiede i dati, le query locali vengono eseguite alle sorgenti locali, quindi è importante ridurre la dimensione delle risposte lq_i. Per ottenere questo risultato è necessario massimizzare la selettività della condizione della query LQ_i e minimizzare la cardinalità della select-list AL_i. Con il seguente metodo entrambe queste proprietà sono soddisfatte.
Per ottenere gli adeguati $LQ_i, 1 \leq i \leq n$ e Q_r è necessario:

1. **Normalizzare la query**
   Per prima cosa si converte Q in una query Q^d in forma *disgiuntiva normale* (DNF) dove i predicati sono atomici. La query DNF sarà nella forma $Q^d = C_1 \vee \cdots \vee C_m$, dove ogni termine di congiunzione C_i è nella forma $C_i = P_1 \wedge \cdots \wedge P_n$, dove P_i è un predicato atomico.
   Nell'esempio si ha:
   Q_d= (Nome like 'P*' and Anno='1') or (Nome like 'P* and Dip='Dip1')

2. **Scrivere la condizione** LQ_i
   In secondo luogo ogni predicato atomico P viene riscritto in uno equivalente supportato dalla classe locale L.
   Un predicato atomico P si dice *ricercabile* nella classe locale L se gli attributi globali usati in P sono presenti in L, ossia se gli attributi globali sono mappati in L con un mapping non nullo.
   Viene fatta l'ipotesi che ogni predicato atomico P ricercabile in L sia pienamente supportato da L, cioè che esista una condizione di query Q_L^P espressa rispetto allo schema $S(L)$ della classe locale tale che, per ogni l, se l' è la relazione ottenuta come risposta alla valutazione di Q_L^P su l, allora

   $$(l')_L^{\mathcal{G}} = P(l^{\mathcal{G}})$$

   Un predicato atomico P di Q^d è riscritto in P_L rispetto a L nel modo seguente:

   - se P è ricercabile in L: $P_L = Q_L^P$
   - se P non è ricercabile in L: $P_L =$true

   nell'esempio si consideri la seguente riscrittura dei predicati:

	Nome like 'P*'	Anno='1'	Dip='Dip1'
L_1	cognome like 'P*'	anno='1'	true
L_2	nome like 'P*'	true	cod_dip='Dip1'

   La condizione LQ_i della query locale si ottiene riscrivendo ogni predicato atomico P di Q^d in P_{L_i} rispetto a L_i.
   Nell'esempio:
   LQ_1=cognome like 'P*' and anno='1'
   LQ_2=nome like 'P*' and cod_dip='Dip1'

3. **Scrivere la condizione residua** Q_r
   Il calcolo di Q_r è diviso nei seguenti passi:

a) Trasformare Q in una query Q^c in forma *congiuntiva normale* (CNF), ossia nell'AND logico di clausole C che sono l'OR logico di predicati atomici P.
Nell'esempio:
Q^c=Nome like 'P*' and (Anno='1' or Dip='Dip1')

b) Ogni clausola C di Q^c contenente predicati atomici non ricercabili in una o più classi locali è una *clausola residua*; nell'esempio:

   * Dip='Dip1' non è ricercabile in L_1, allora (Anno='1' or Dip='Dip1') è una condizione residua per L_1
   * Anno='1' non è ricercabile in L_2 allora (Anno='1' or Dip='Dip1') è una condizione residua per L_2

c) La condizione residua Q_r è uguale all'AND logico delle clausole residue. Nell'esempio:
Q_r=(Anno='1' or Dip='Dip1')

4. **Scrivere la select list di una query locale** LQ
La select list della query LQ per la classe locale L, denotata con LAL, è ottenuta considerando l'unione dei seguenti insiemi di attributi:

   (a) gli attributi della select list globale AL

   (b) l'insieme degli attributi di join $JA(L)$ per la classe locale L

   (c) gli attributi nella condizione residua A_r

   e trasformando questi attributi sulla base della mapping table:

$$LAL = \{A \in S(L) \mid \exists AG \in AL \cup JA(L) \cup A_r, A \in M[AG][L]\}$$

   Nell'esempio:
   A_r={Anno,Dip}
   AL={Nome}
   $JA(L_1)$={Nome}
   $JA(L_2)$={Nome}
   allora:
   LAL_1={cognome, nome, anno}
   LAL_2={nome,cod_dip}

In conclusione, si hanno le seguenti query locali LQ_1 e LQ_2:

select cognome,nome,anno
from L1
where (nome like'P*' and anno='1'

select nome, cod_dip
from L2
where (nome like 'P*' and cod_dip='Dip1')

La query locale LQ_i è spedita alla sorgente relativa alla classe locale L_i; la sua risposta q_i è trasformata dall'operatore $(.)_{\mathcal{G}}^{L_i}$ ed il risultato è mantenuto in una relazione temporale T_i, definita dall'equazione 4.5 a pagina 131 e dalla condizione residua Q_i. In seguito si calcola la full disjunction delle T_i ed infine si ottiene la risposta alla query globale applicando la trasformazione definita in equazione 4.5 a pagina 131.

4.2.1 Le modifiche da apportare

Per prima cosa viene a cadere l'ipotesi di omogeneità semantica e quindi non si parla più di db come di un database di sorgenti semanticamente omogenee e perciò possono essere possibili dei conflitti fra i valori nei dati. Ciò può causare degli inconvenienti per quanto riguarda l'applicazione della funzione di risoluzione. Ammettiamo infatti di avere le seguenti sorgenti:

L_1

prezzo	data	ID
40	9/2	ID1
43	1/1	ID2

L_2

prezzo	data	ID
50	10/2	ID1
42	10/1	ID3

e la seguente mapping table:

	Prezzo	Data	ArticoloID
L_1	prezzo	data	ID
L_2	prezzo	data	ID

Per semplicità e per non spostare l'attenzione dal problema ammettiamo che i due database mantengano informazioni riguardanti lo stesso negozio, quindi è ammissibile supporre la presenza di un identificatore comune: "ArticoloID"; in più chiamiamo $\mathcal{G}$ la relazione globale.
A questo punto supponiamo che venga posta la seguente query:

select *
from $\mathcal{G}$
where Prezzo< 45 and Data$> 1/2$

Supponiamo inoltre che la funzione su cui si basa la risoluzione dei conflitti per l'attributo "prezzo" si basi sulla ricerca del valore più aggiornato e quindi scelga quello corrispondente al valore di "data" più recente, mentre per quanto riguarda l'attributo "data" si sceglie semplicemente il valore più aggiornato.
Seguendo i passi riportati precedentemente, ad ogni fonte viene spedita la query:

select *
from L_1 (o L_2)
where prezzo< 45 and data$> 1/2$

che porta ai seguenti risultati:

tuple risultanti da L_1

prezzo	data	ID
40	9/2	ID1

tuple risultanti da L_2

prezzo	data	ID
$\perp$	$\perp$	$\perp$

e quindi il risultato mostrato all'utente sarà costituito dalla prima tupla di L_1.
Adesso invece proviamo ad eseguire la seguente query:

select *
from $\mathcal{G}$

Come è ovvio aspettarsi, questa volta i risultati delle query locali coincideranno con le sorgenti stesse, ossia:

tuple risultanti da L_1

prezzo	data	ID
40	9/2	ID1
43	1/1	ID2

tuple risultanti da L_2

prezzo	data	ID
50	10/2	ID1
42	10/1	ID3

poiché abbiamo ipotizzato la presenza di un identificatore globale si vede come vi siano, tra le tuple risultanti, due che rappresentano lo stesso oggetto; le relative informazioni però presentano dei conflitti, in particolare una tupla è più recente dell'altra. Quindi all'atto della fusione dei risultati si rende necessario l'utilizzo della funzione di risoluzione riportata precedentemente; al termine del processo si ottiene la seguente relazione:

$\mathcal{G}$

Prezzo	Data	ArticoloID
50	10/2	ID1
43	1/1	ID2
42	10/1	ID3

Soffermiamo l'attenzione sulla prima tupla, quella riguardante l'oggetto ID1. Rispetto al caso precedente i due risultati sono diversi, nonostante rappresentino lo stesso oggetto; quello riportato nella prima esecuzione è errato, dato che riporta un risultato obsoleto; il risultato invece da considerarsi esatto consisterebbe nella tupla vuota.
Questo errore è causato dalla funzione di risoluzione, o meglio da un suo utilizzo ritardato; per eliminarlo dovrebbe essere possibile risolvere le tuple direttamente sulle fonti locali. Il vero problema di questo approccio consiste nel risolvere conflitti su tuple appartenenti a relazioni diverse: a parte l'esecuzione in sè, alquanto onerosa, l'assenza nella maggior parte dei casi pratici di un identificatore globale richiede alla funzione di risoluzione di agire a monte dell'identificazione degli oggetti; ciò, in altri termini, significa delegare alla funzione di risoluzione il compito di determinare se due istanze rappresentano lo stesso oggetto. Questo, sempre se realizzabile, esula dalla definizione di funzione di risoluzione, estendendone il concetto.
In realtà non tutte le funzioni di risoluzione provocano questo tipo di problemi, ma solo quelle generalizzate (ossia quelle che si servono dei valori degli altri attributi per fornire il risultato); quindi sarebbero solo queste a dover subire un trattamento particolare.
Invece che anticipare la funzione di risoluzione, eseguendola sulle fonti locali, in alternativa è possibile posticipare le clausole di selezione riguardanti attributi che fanno uso di funzioni di risoluzione generalizzate, eseguendole esclusivamente come condizione residua, a prescindere dal fatto che le condizioni siano o meno eseguibili su tutte le classi locali. Quindi, riprendendo l'esempio precedente, come prima cosa si inviano alle classi locali le seguenti query:

select *
from L_1 (o L_2)
where data$> 1/2$

si noti come la clausola where contenga una restrizione su "data": questo perché l'attributo in questione si basa su una funzione di risoluzione non generalizzata. Il risultato restituito è:

tuple risultanti da L_1

prezzo	data	ID
40	9/2	ID1

tuple risultanti da L_2

prezzo	data	ID
50	10/2	ID1

a questo punto, durante la costruzione della vista globale, è possibile applicare la funzione di risoluzione generalizzata, ottenendo:

$\mathcal{G}$

Prezzo	Data	ArticoloID
50	10/2	ID1

Ora è possibile eseguire la query comprendente i predicati residui:

select *
from $\mathcal{G}$
where prezzo< 45

ottenendo come risultato (corretto) una relazione vuota.
Utilizzando questo metodo il problema viene risolto, al costo di rilassare le clausole di selezione da spedire alle query locali e quindi di ottenere un maggior numero di tuple in risposta. Va ricordato comunque che solamente le funzioni di risoluzione generalizzate creano questo tipo di inconveniente, quindi per tutte le altre è utilizzabile il metodo di riscrittura presentato precedentemente.
In ogni caso, in futuro si studieranno metodi alternativi per ovviare a questo problema, probabilmente estendendo il concetto di “funzione di risoluzione”.

4.3 La γ-ciclicità in ambito SEWASIE

A seguito di tutti i concetti esposti precedentemente, può sorgere spontanea una domanda: ma la γ-ciclicità negli schemi è un problema così importante?
La risposta può essere incerta considerando gli ambiti nei quali agisocno gli algoritmi di Konstantin e di Rajaramann e Ullman, mentre è sicuramente positiva nell’ambiente in cui verrá applicato il nuovo algoritmo proposto. Questo perché se degli ipergrafi γ-ciclici si verificano con una certa frequenza in ambiti diversi dal nostro, in SEWASIE la probabilità di ritrovarsi a dover gestire questi casi aumenta di molto.
La ragione di questo incremento è molto semplice e va ricercata nella struttura dei grafi. Nel nostro caso si considerano solamente grafi competamente connessi; mentre ciò non costituisce condizione sufficiente per la γ-ciclicità degli ipergrafi (è addirittura possibile trovare grafi completamente connessi che corrispondono ad

ipergrafi aciclici), ne aumenta considerevolmente la probabilità, soprattutto quando il numero di nodi è maggiore di tre.

Oltre a questa considerazione basata sul contesto, se ne possono muovere altre basate sull'algoritmo, o meglio sulle condizioni di join e quindi di riflesso anche negli operatori a cui si ricorre.
A differenza dei precedenti algoritmi di Ullman e Konstantin, l'operatore che si utilizza nel nuovo algoritmo è il full outerjoin; ció porta ad un aumento delle condizioni di join ammissibili, che non sono più vincolate a basarsi su tutti gli attributi comuni tra le relazioni. Ció in alcuni casi può portare a vantaggi, allargando anche il campo d'azione del nostro algoritmo.
Come detto in sezione 3.5.5 il nostro algoritmo funziona solamente con ipergrafi in configurazione aciclica o γ-ciclica ma β-aciclica; ciò è vero solamente se le condizioni di join sono espresse su tutti gli attributi comuni, ma nel nostro caso questi limiti possono essere oltrepassati. Questo è dovuto alla definizione stessa di ipergrafi data da Fagin in [37], dato che tutto lo studio dell'autore si basa sul presupposto che i join avvengano su tutti gli attributi comuni.
Come mostrato in sezione 3.5.5 ciò che porta all'errore è l'estensione degli schemi con attributi facenti parte di prossimi join, quindi la discriminante è costituita unicamente dagli attributi di join, lasciando piena libertà per quanto riguarda gli altri; ció significa che gli attributi comuni ma non di join possono assumere qualsiasi configurazione senza sconvolgere l'esito dell'algoritmo.
Riprendiamo la definizione di β-ciclo data a pagina 102: ogni ciclo di questo tipo ha un insieme di attributi X comuni a tutte le relazioni, in assenza del quale ci si trova in presenza di un ciclo puro. Supponiamo che il join tra tutte le relazioni si basi solamente su attributi facenti parte dell'insieme X: il ciclo puro sarebbe quindi formato solamente da attributi non di join, non influenti per il risultato dell'algoritmo. Il ciclo puro non sarebbe più un problema e il risultato dell'esecuzione dell'algoritmo porterebbe alla full disjunction. Ciò deriva dal fatto che essendo gli attributi di join comuni a tutte le relazioni non si ha in nessun caso un'estensione degli schemi basata sugli attributi di join.
Quindi le definizioni dei vari gradi di ciclicità degli ipergrafi sono adeguate solamente se le condizioni di join si basano esclusivamente su tutti gli attributi comuni e quindi nel nostro caso devono essere modificate.
Per rendere l'idea si puó usare una notazione che usi come nodi gli attributi di join (non più tutti gli attributi in generale) e come iperarchi sempre le relazioni (d'ora in avanti ci si potrà riferire a questi come *ipergrafi ridefiniti*). In linea di principio è possibile tracciare l'ipergrafo classico e poi eliminare tutti quegli attributi che non sono di join; questo tipo di ipergrafo funziona bene soprattutto per un numero di nodi ed iperarchi poco elevato, mentre in caso contrario tende a diventare caotico. Per questo motivo per il futuro è bene cercare una notazione alternativa.

Gli ipergrafi utilizzati negli altri contesti hanno la proprietà di essere in relazione biunivoca con lo schema del database e quindi è possibile dire che "uno schema è γ-ciclico (o β-ciclico) se il rispettivo ipergrafo lo è"; utilizzando una notazione basata sulle condizioni di join è chiaro che una proprietà del genere perde di significato, in quanto ad uno schema possono corrispondere diversi ipergrafi, a seconda delle condizioni di join mantenute nella join table: perció ad uno schema γ-ciclico può corrispondere un ipergrafo γ-aciclico, in base alle condizioni di join da applicare.
In realtà a questo punto è facile vedere che può avvenire anche il caso inverso: ad uno schema γ-aciclico può corrispondere un ipergrafo γ-ciclico. Supponiamo ad esempio di avere le tre sorgenti L_1, L_2, L_3, tutte con schema $A_1A_2A_3$. Utilizzando la vecchia notazione si ha un unico ipergrafo determinato univocamente: quello in figura 4.1. Come si può palesemente vedere, l'ipergrafo non è γ-ciclico.

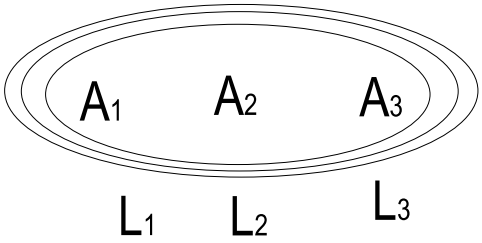


Figura 4.1: Ipergrafo tradizionale

Per quanto riguarda invece il nostro caso, supponiamo di avere la seguente join table:

	L_1	L_2	L_3
L_1	X	A_1	A_3
L_2	A_1	X	A_2
L_3	A_3	A_2	X

l'ipergrafo corrispondente è riportato in figura 4.2. In questo caso, si vede chiaramente che siamo in presenza di un ciclo puro, ossia di un β-ciclo.

Ció non deve spaventare, dato che uno schema γ-aciclico secondo la definizione classica degli ipergrafi può originare un ipergrafo γ-ciclico (e, nell'esempio, anche β-ciclico) in casi ben determinati. Prima di tutto è necessario che le relazioni abbiano tra di loro un gruppo X di attributi di join in comune. Oltre a questo, alcuni di essi possono avere anche un ulteriore gruppo di attributi in comune Y; quindi ci sono al massimo due tipi di schemi di relazione: X e $X \cup Y$. Se così non fosse, allora si ricadrebbe in un caso di γ-ciclicità.
Trattiamo i due casi separatamente:

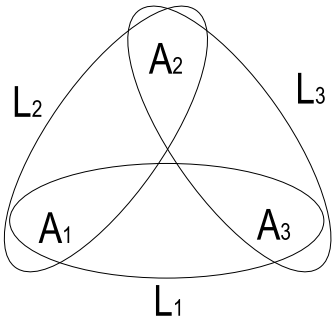


Figura 4.2: Ipergrafo ridefinito

- nel caso in cui tutti gli schemi contengano solo gli attributi in X, allora l'algoritmo riesce ad arrivare alla full disjunction senza problemi, poiché nessun join estende gli schemi.
- nel caso in cui alcuni schemi contengano anche gli attributi in Y, allora per arrivare il risultato è sufficiente applicare la regola generale e quindi prendere per primi quei nodi che corrispondono a relazioni con schema $X \cup Y$

Quindi anche situazioni di questo tipo non creano problemi. D'altra parte, essendo lo schema non γ-ciclico, anche l'algoritmo di Ullman è in grado di risolvere questi problemi.

Abbiamo quindi che il campo d'azione dell'algoritmo ora comprende:

- ipergrafi γ-aciclici
- ipergrafi γ-ciclici ma β-aciclici
- ipergrafi β-ciclici che originano ipergrafi ridefiniti β-aciclici

e quindi si allarga leggermente.

4.4 Le tuple sussunte in ambito SEWASIE

La definizione originaria di tupla sussunta è estremamente generale:

Definizione 4.4.1 *una tupla si dice sussunta se esiste un'altra tupla che fornisce un contenuto informativo maggiore.*

Essendo una definizione molto vaga, gli autori degli studi precedenti ([33], [34], [36]) l'hanno specializzata per renderla più consona al proprio contesto:

Definizione 4.4.2 *una tupla sussume un'altra tupla se coincide con essa in tutti gli attributi con valore significativo e in più ha almeno un attributo con valore significativo dove l'altra ha un valore nullo.*

La precedente definizione si può adattare al nostro contesto basato sul concetto di oggetto semplicemente parlando di classi, oggetti e stati degli oggetti piuttosto che di relazioni, tuple ed attributi.
Nonostante ciò il concetto di fondo presenta ancora quello che può essere considerato come un errore nel nostro contesto. Ammettiamo per esempio di avere le seguenti relazioni:

A	B
1	10
2	$\perp$

A	B
1	10
2	20

Volendone calcolare la full disjunction, ammettiamo di confrontare tutti gli attributi comuni e che gli attributi locali abbiano lo stesso nome di quelli globali; il risultato sarebbe:

A	B
1	$\perp$
1	10
2	20

considerando la definizione di tupla sussunta appena introdotta il risultato diviene palesemente errato, in quanto la seconda tupla corrisponde alla prima per quanto riguarda gli attributi con valore significativo (A) e in più fornisce valori significativi in corrispondenza ai valori nulli della prima (attributo B).
Ma quanto affermato è da considerarsi davvero corretto? Dipende da quello che viene considerato come identificatore. Poniamo che sia possibile stabilire univocamente l'identità di un oggetto facendo riferimento solamente ad A. In questo caso allora si può davvero affermare che la seconda tupla sussume la prima in quanto entrambe forniscono informazioni sullo stesso oggetto, però la seconda ne fornisce una quantità maggiore.
Se però l'identificatore è costituito dalla coppia AB, allora le cose possono assumere diverse sfumature; in particolare ci possono essere due punti di vista:

1) si assume che la seconda tupla sussuma la prima, esattamente come nel caso precedente. Questo approccio è in linea con quanto affermato in [33], [34] e [36] e pertanto segue anche la definizione più specializzata enunciata precedentemente.

2) si assume che la seconda tupla non sussuma la prima, ritenendo i due identificatori diversi o comunque non confrontabili, data la presenza del campo nullo. In un certo senso adottando questo approccio si può vedere, in determinati casi, il valore nullo non come mancanza di informazione, bensì come informazione vera e propria. Per mostrare il senso di quest'ultima affermazione prendiamo un esempio concreto: supponiamo di avere una relazione con attributi "nome" e "tessera#" e che l'identificatore scelto sia composto da entrambi gli attributi. A questo punto si potrebbe verificare un caso come il seguente:

nome	tessera#
Mario Rossi	123
Mario Rossi	$\perp$

In questo caso il null può essere visto come mancanza di informazione, e quindi le due tuple si riferiscono alla stessa persona, oppure può semplicemente significare che l'utente non possiede ancora una tessera; in questo caso si avrebbe un caso di omonimia, dove il primo utente possiede una tessera, mentre il secondo ancora no. In questo caso le due tuple rappresenterebbero due entità distinte e quindi il concetto di tupla sussunta non viene applicato.

Perciò il valore nullo può essere considerato un valore significativo se fa parte dell'identificatore; quindi si vede come sia necessaria una nuova ridefinizione di tupla sussunta da utilizzare nel nostro ambito:

Definizione 4.4.3 *un'istanza di oggetto sussume un'altra istanza di oggetto se entrambe rappresentano lo stesso oggetto, forniscono gli stessi dati per gli stati di valore significativo ed in più il primo fornisce informazioni relative almeno ad un ulteriore stato per il quale la seconda ha valore non significativo.*

In realtà la sola differenza tra questa definizione e quelle date precedentemente risiede nel fatto che ora si fa riferimento esplicitamente al concetto di identità della tupla (o dell'oggetto). I lavori precedenti non hanno mai preso in considerazione l'identificatore e quindi il valore nullo era visto solo come una mancanza di informazione.
Nel nostro ambito non sono solo i parziali valori nulli sugli identificatori a creare dei problemi: ammettiamo di avere le seguenti sorgenti locali:

L_1		L_2	
A	B	A	B
2	9	1	9
4	6	3	$\perp$
5	7	5	$\perp$

Ammettiamo che gli attributi siano stati mappati su attributi aventi stesso nome anche nella classe globale, inoltre supponiamo che la condizione di join fra L_1 ed L_2 sia espressa tramite l'espressione $L_1.A = L_2.A + 1$. Allora la full disjunction risulta:

$L_1.A$	$L_1.B$	$L_2.A$	$L_2.B$
2	9	1	9
4	6	3	$\perp$
5	7	$\perp$	$\perp$
$\perp$	$\perp$	5	$\perp$

Soffermiamoci adesso sulla seconda tupla: è stata ottenuta dal join delle due seconde tuple delle relazioni locali. Utilizzando le precedenti definizioni di tupla sussunta le due tuple risultano indipendenti, invece ricorrendo a quella appena introdotta (che fa uso del concetto di identità) si ottiene che le due tuple sono in realtà sussunte. Infatti la seconda tupla di L_2 non aggiunge alcuna informazione a quella di L_1, anche se prima di conoscere la condizione di join (o di sapere che entrambe rappresentano lo stesso oggetto) ciò sarebbe stato impossibile da intuire. Specularmente, mentre utilizzando le tradizionali definizioni di tupla sussunta si ottiene che la terza tupla di L_1 sussume la corrispondente di L_2, in realtà così non è, in quanto nella full disjunction si vede chiaramente che queste due tuple rappresentano oggetti distinti.

Questo è dovuto al fatto che la condizione di join che viene utilizzata nel nostro ambito non è costituita da una semplice uguaglianza, ma da una espressione vera e propria e quindi i concetti e le assunzioni dei precedenti studi non sono più adeguate.

Capitolo 5

Conclusioni

Il lavoro svolto con questa tesi può essere diviso in due parti. Nella prima sono stati esaminati i vari aspetti riguardanti la fusione di dati provenienti da diverse fonti e le varie tecniche finora utilizzate per tentare di risolverle. Come spesso accade nel mondo dell'informatica, non esiste una soluzione completamente priva di difetti. La vastità del problema rende difficile prendere in esame tutte le possibili cause di errore e quindi ogni metodo si rivela particolarmente efficace nel controbattere solo alcune di queste.

Fra le varie problematiche che riguardano la fusione dei dati, quella che sembra essere la più complessa da risolvere è il riconoscimento degli oggetti. Del resto, forse proprio per questo motivo, le ricerche riguardanti la fusione si sono concentrate proprio su questo aspetto, lasciando scoperte altre problematiche (ad esempio, gli studi che riguardano la funzione di risoluzione o la full disjunction si contano sulle dita di una mano). Nonostante ciò non si è ancora trovata una soluzione definitiva; questo perché negli ultimi anni si è supposta anche la presenza di eventuali errori di immissione nei dati. Quest'ultima assunzione ha causato una complicazione enorme del problema; la domanda che bisognerebbe porsi è se sia o meno possibile occuparsi anche di questo aspetto. Il fatto è che, senza informazioni aggiuntive, risulta molto difficile, se non impossibile, stabilire quando una tupla contiene valori erronei. Utilizzando le funzioni di distanza presentate precedentemente si ottengono risultati approssimativi, ma anche disponendo a priori di informazioni il compito risulta complesso. Per evitare di dover memorizzarne un numero troppo elevato si potrebbe pensare di integrarle con un metodo basato sulla distanza; in questo modo sarebbe possibile mantenere informazioni indicanti l'uguaglianza (o la diversità) solamente delle coppie di tuple con distanza inferiore alla soglia scelta. Comunque, a seconda dei dati con cui si ha a che fare, anche tale metodo può risultare molto dispendioso o inutile. In alcuni casi queste informazioni aggiuntive possono essere ricavate direttamente dalle tuple, confrontando

i valori di attributi comuni ma non di join. Perchè questo sia possibile è necessario che gli schemi presentino una estesa sovrapposizione. Come si è visto anche un approccio di questo tipo può presentare delle lacune, a seconda del contesto e delle informazioni mantenute.
Comunque non solo il riconoscimento dei dati viene disturbato dagli errori; infatti anche supponendo la correttezza dei campi identificatori, errori sugli altri attributi portano a presentare all'utente risultati sbagliati, anche se con gli attributi in comune a più fonti il problema può essere almeno mitigato con l'uso di opportune tecniche (come le funzioni di risoluzione).

Dopo una breve analisi si può concludere che le soluzioni più promettenti sono quelle che prevedono l'impiego di informazioni aggiuntive. Possedere un buon numero di informazioni può rendere molto più semplice e preciso ogni passo della fusione dei dati, a partire dall'integrazione degli schemi fino ad arrivare alla risoluzione dei conflitti; del resto anche tutto il discorso della qualità si basa proprio su una conoscenza delle fonti, anche se non così approfondita.

Nella seconda parte del lavoro si sono corretti gli errori e mostrati i limiti dell'algoritmo da implementare nell'ambito SEWASIE, introducendo anche il problema della risoluzione dei conflitti tra i dati e fornendo una possibile soluzione con l'implementazione della funzione di risoluzione nell'algoritmo.
L'algoritmo é stato prima proposto in un ambiente più generale e solo in seguito specializzato per l'ambito SEWASIE, mostrandone le modifiche necessarie.

Entrambi gli ambiti trattati sono in piena fase di ricerca, ma purtroppo una soluzione definitiva ai problemi trattati non é ancora stata trovata.
In particolare risulta difficile l'identificazione degli oggetti ed il trattamento delle inconsistenze sugli attributi di join. L'operazione, già complessa di per sè, è resa ancor più complicata nel nostro ambito dal riconoscimento effettuato utilizzando solo gli attributi di join. Nemmeno l'algoritmo basato sugli or (quello più costoso, ma più efficace) riesce a portare in uscita la full disjunction. Perché questo sia possibile è necessario ricorrere a metodi alternativi che facciano uso di informazioni aggiuntive. Comunque questo aspetto merita un futuro studio più approfondito.
L'algoritmo di Konstantin, presentato inizialmente come un enorme passo avanti nel calcolo della full disjunction, si è poi rivelato molto meno efficace di quanto sperato, riuscendo a risolvere solo una piccola parte dei problemi lasciati in sospeso da Rajaramann e Ullman e a che prezzo! La complessità computazionale è davvero elevata, molto più vicina al pesante algoritmo basato sull'or che non al SOJO.
I limiti dell'algoritmo ora pongono ulteriori problemi: come trattare i casi nei quali questo fallisce? Come ridurre la complessità computazionale il più possibile? Basandosi sull'analisi degli schemi e delle condizioni di join si potrebbe utilizza-

re l'algoritmo con il costo minore e che è in grado di portare alla full disjunction. Perciò dato uno schema γ-aciclico si utilizza la sequenza di Ullman, se invece lo schema è γ-ciclico e β-aciclico ed esiste un ordinamento corretto si applica l'algoritmo qui proposto mentre in ogni altro caso si ricorre al metodo basato sull'or. Volendo sviluppare un algoritmo alternativo a quello proposto in questa sede è necessario prescindere completamente dai risultati raggiunti da Konstantin, in quanto *non è possibile* risolvere ipergrafi β-ciclici partendo da un concetto simile al suo.

Bibliografia

[1] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey D. Ullman, and Jennifer Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *16th Meeting of the Information Processing Society of Japan*, pages 7–18, Tokyo, Japan, 1994.

[2] Yannis Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom. Object exchange across heterogeneous information sources. In P. S. Yu and A. L. P. Chen, editors, *11th Conference on Data Engineering*, pages 251–260, Taipei, Taiwan, 1995. IEEE Computer Society.

[3] Felix Naumann and Matthias Hussler. Declarative data merging with conflict resolution.

[4] Felix Naumann and Johann C. Freytag. Completeness of information sources. Technical Report Informatik Bericht 135, 2000.

[5] Gösta Grahne and Alberto O. Mendelzon. Tableau techniques for querying information sources through global schemas. *Lecture Notes in Computer Science*, 1540:332–347, 1999.

[6] Jarek Gryz. Query folding with inclusion dependencies. In *14th Int. Conference on Data Engineering*, pages 126–133, Orlando, Florida, 1998.

[7] Oliver M. Duschka. Query optimization using local completeness. In *AAAI/IAAI*, pages 249–255, 1997.

[8] Alon Y. Levy. Obtaining complete answers from incomplete databases. In *The VLDB Journal*, pages 402–412, 1996.

[9] Ke Wang and Weining Zhang. Detecting data inconsistency for multidatabases.

[10] M. Scannapieco A. Virgillito R. Baldoni C. Marchetti, M. Mecella. Data quality notification in cooperative information systems. In *Proceedings of International Workshop on Data Quality in Cooperative Information Systems*, 2003.

[11] Felix Naumann. From databases to information systems - information quality makes the difference. In *Proceedings of the International Conference on Information Quality (IQ)*, Cambridge, MA, 2001.

[12] H. J. Lenz F. Naumann M. Neiling, S. Jurk. Object identification quality. In *Proceedings of International Workshop on Data Quality in Cooperative Information Systems*, 2003.

[13] Laure Berti-Equille. Quality extended query processing for distributed sources. In *Proceedings of International Workshop on Data Quality in Cooperative Information Systems*, 2003.

[14] L. Bertossi and J. Chomicki. Query answering in inconsistent databases.

[15] Felix Naumann. Data fusion and data quality. In *Seminar on New Techniques and Technologies for Statistics*, Sorrento, Italy, 1998.

[16] Mauricio Antonio Hernandez. A generalization of band joins and the merge/purge problem, 1996.

[17] Mauricio A. Hernandez and Salvatore J. Stolfo. The merge/purge problem for large databases. In *SIGMOD Conference*, pages 127–138, 1995.

[18] D. DeWitt, J. Naughton, and D. Schneider. An evaluation of non-equijoin algorithms. In *Proceedings of the 17th Conference on Very Large Databases, Morgan Kaufman pubs. (Los Altos CA), Barcelona*, 1991.

[19] C.I. Ezeife Timothy E. Ohanekwu. A token-based data cleaning technique for data warehouse systems. In *Proceedings of International Workshop on Data Quality in Cooperative Information Systems*, School of Computer Science, University of Windsor, Ontario, 2003.

[20] Ee-Peng Lim, Jaideep Srivastava, Satya Prabhakar, and James Richardson. Entity identification in database integration. *Information Sciences*, 89(1):1–38, 1996.

[21] I. P. Felligi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Society*, (64):1183–1210, 1969.

[22] Cheng Hian Goh, Stéphane Bressan, Stuart Madnick, and Michael Siegel. Context interchange: new features and formalisms for the intelligent integration of information. *ACM Transactions on Information Systems*, 17(3):270–270, 1999.

[23] D. Beneventano, S. Bergamaschi, F. Guerra, and M. Vincini. Synthesizing an integrated ontology. *IEEE Internet Computing Magazine*, pages 42–51, 2003.

[24] Mattis Neiling and Hans-Joachim Lenz. Data integration by means of object identification in information systems. In *In Proceedings of European Conference on Information Systems*, Vienna, Austria, 2000.

[25] Mattis Neiling and H.-J. Lenz. Data fusion and object identification. In *SSGRR2000*, l'Aquila, Italy, August 2000.

[26] Mattis Neiling and Steffen Jurk. The object identification framework. In *KDD03 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, 2003.

[27] R. Hull. Managing Semantic Heterogeneity in Databases: A Theoretical Perspective. In *ACM Symp. on Principles of Database Systems*, pages 51–61, 1997.

[28] T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The information manifold. In *In Working Notes of the AAAI Spring Symposium on Information Gathering from Heterogeneous*, 1995.

[29] H. Garcia-Molina et al. The TSIMMIS Approach to Mediation: Data Models and Languages. In *NGITS workshop*, 1995. Available ftp://db.stanford.edu/pub/garcia/1995/tsimmis-models-languages.ps.

[30] M. Scannapieco P. Bertolazzi, L. De Santis. Automatic record matching in cooperative information systems. In *Proceedings of International Workshop on Data Quality in Cooperative Information Systems*.

[31] Y. Yang T. Lee. Combining information extraction and verification using semistructured constraints. In *Proceedings of International Workshop on Data Quality in Cooperative Information Systems*, Department of Operations and Information Management, The Wharton School, University of Pennsylvania, 2003.

[32] B. Pernici P. Plebani M. Scannapieco C. Cappiello, C. Francalanci. data quality assurance in cooperative information systems: a multi-dimension quality certificate. In *Proceedings of International Workshop on Data Quality in Cooperative Information Systems*, 2003.

[33] C. Galindo-Legaria. Outerjoins as disjunctions. *CWI*, 1993.

[34] Anand Rajaraman and Jeffrey D. Ullman. Integration information by outerjoins and full disjunction. *ACME*, 1997.

[35] Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM, 30:3, pp 514-550,* 1983.

[36] K. E. Kostenarov. Elaborazione di interrogazioni in un sistema multidatabase. Tesi di Laurea, Dipartimento di Scienze dell'Ingegneria, Università degli Studi di Modena, 2002.

[37] E. F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems (TODS)*, 4(4):397–434, 1979.

[38] D. Lembo, M. Lenzerini, and R. Rosati. Source inconsistency and incompleteness in data integration. In *proceedings of KRDB*, 2002.