Probabilistic Graph Model

Takenori Sato

March 29, 2009

Contents

List of Figures

Chapter 1

Abstract

ODBMS has been proved to work quite well with Graph. The intrinsic advantages of ODBMS to interact with an algorithm makes it possible.

While in Information Retrieval, many studies have been done to improve search engines. From the domain modeling point of view, they look quite impressive especially because they find latent similarities between documents.

In this paper, Probabilistic Graph Model is introduced, in which edges are represented in probabilities, not 1 or 0, but between 0 and 1. And then, its possible application is shown, which models latent relationships of Resources under REST(Representational State Transfer) concept in WWW with an extended model by Composite design pattern. At last, through discovery of latent links between resources, the way to achieve the best possible optimizations are shown.

This is made for **Common Persistent Model Patterns for Performance and/or Scalability Optimization** by ODBMS.ORG in a couple of days. Actually written in a day. There're many to be studied further, but many points are shown from a new point of view.

I hope this would help you to inspire.

Chapter 2

Background

2.1 Graph Basics

Graph denoted by G(V,E) consists of a set V of *vertices* and a set E of *edges*. Each edge is a set of two vertices from V.

$$\begin{aligned} V &= \{v_1, ..., v_m\} \\ E &= \{e_1, ..., e_n\} \\ e_k &= \{v_i, v_j\} \\ e_k &\in E, v_i, v_j \in V \end{aligned}$$

If $\{v_i, v_j\} \neq \{v_j, v_i\}$, G is a *directed* graph. In UML, it is represented as shown in Figure 2.1.

For ODBMS, *no cyclic* directed Graph is highly recommended. This is not often mentioned, but very important practical rule. Suppose you make a query denoted as q, then retrieve a sub set of G denoted as G_q. After n queries, you have a subset of G denoted as G(n)

$$G(n) = G_{q1} \cup ... \cup G_{qn}$$

On G(n), how do you define unused subset $G^{'}$? In some advanced environment like Java, how does GC determin such a subset? If G is not no cyclic directed Graph, that is likely a difficult question. Then, you might

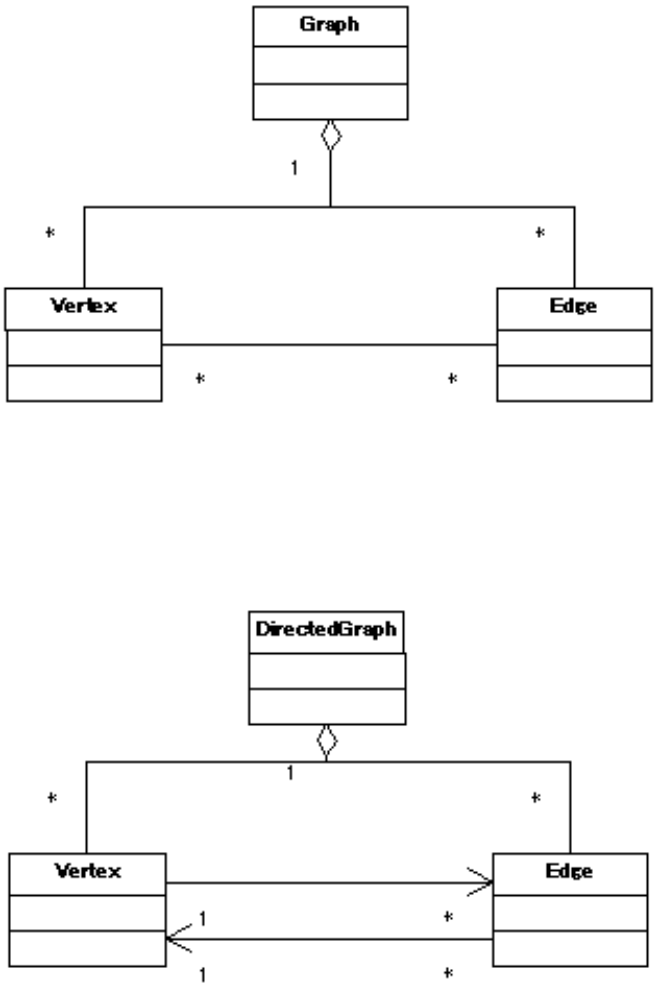

Figure 2.1: Graph and DirectedGraph in UML

end up with loading entire graph G on memory.

Given a query q, a probability if an edge e_k of a vertex v_i is retrieved or not is denoted as $p(e_k|v_i, q) \in (0, 1)$. It is 1 if v_j matches q, 0 otherwise. A graph that is retrieved by query q is $G_q(V_q, E_q)$.

$$\begin{aligned} V_q &= \{v_1, ..., v_x\} \\ E_q &= \{e_1, ..., e_y\} \\ p(e_k|v_i, q) = 1, k \in (1...y), i \in (1...x) \end{aligned}$$

In other words, on a path, which is made by two vertices picked up from V_q, from v_f to v_t, all the probabilities of vertices involved are 1.

As mentioned above, both in terms of query and memory management, non-cyclic directed graph is highly recommended. And a graph retrieved by query q is a sub set of G, which can be represented with vertices of the probability 1. This means that runtime behaviors are constrained by its static model. In some problem domains emerged recently, edges between vertices can not be formed statically in boolean manner, but dynamically in probability. It'll be introduced later in this paper.

2.2 REST

REST is an abbreviation of Representational State Transfer. According to Wikipedia, it is defined as "a style of software architecture for distributed hypermedia systems such as the World Wide Web". And systems that follow REST is called *RESTful*. As XML web services has been turned out to be quite complex, RESTful web service has become major with the successful data structure calle *JSON*.

The central principle of REST is *Resource*(s), each of which can be accessed by its unique identifer as one of possible *Representation*(s). On World Wide Web, a resource is accessed by its URI by HTTP. URI is models as whole-part relationship, with the successful design pattern Composite, and Strategy for available representations as shown in Figure 2.2.

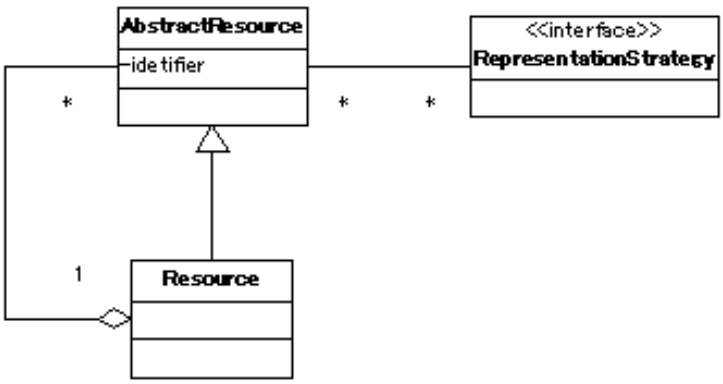

Figure 2.2: Resource in UML

Chapter 3

Probabilistic Graph Model

A graph Ω(V, E) is defined as *Probabilistic Graph* if $p(e_k|v_i, q) \in \{0...1\}$. This definition is an extention of conventional Graph G(V,E), which is the special case when $p(e_k|v_i, q) \in \{0, 1\}$. If such an edge, $e_k = \{v_i, v_j\} \in E$, exists that is $p(e_k|v_i, q)! = p(e_k|v_j, q)$, it is considered *directed*.

There's no such an notation that shows probabilities in UML. So I used dot line to mean such a *might have* replationship as shown in Figure 3.1.

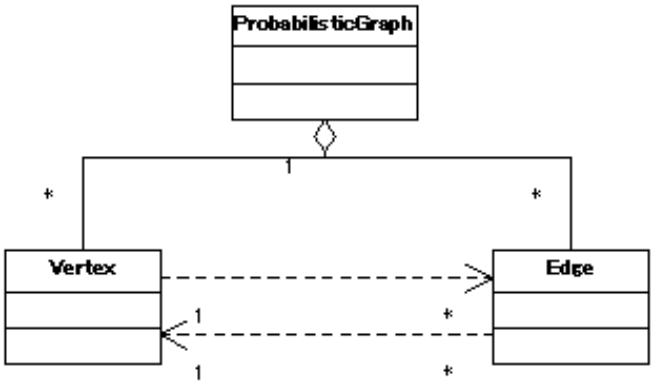

Figure 3.1: Probabilistic Graph in UML

Chapter 4

Latent Resource Link Discovery

4.1 Data Model

Latent Resource Link Discovery is a model to find latent links between Resources as shown in Figure 4.1. The main concern for Information Retrieval has been on a single retrieval task. But in the real world, Information Retrieval is considered as a collection of sequencial tasks. This model is to picture such sequences as Probability Graph.

With directed whole-part relationship, Composite is considered as directed Graph. By extending relationships with probabilities, Latent Resource Link Discovery is modeled as no-cyclic directed Probabilistic Graph.

Aspect is "a part of a program that cross-cuts its core concerns, therefore violating its separation of concerns" according to Wikipedia. In ODBMS, such core concerns are modeled as data structure defined by a set of *Class*, which consists of a set of *Member*. Aspect is weaved with a set of Members. A Resource has one or more Aspect(s).

LatentLink is a utility that holds one pair of possibility value and Resource.

A Link between Resources is implemented with an ordered array of number storing probability values and its corresponding ordered array of AbstractResource objects. In ODBMS, it worths to pay some attentions for their differences. In general, an array belongs to the class that defines the array as its member. While a list is considered independent. In the case

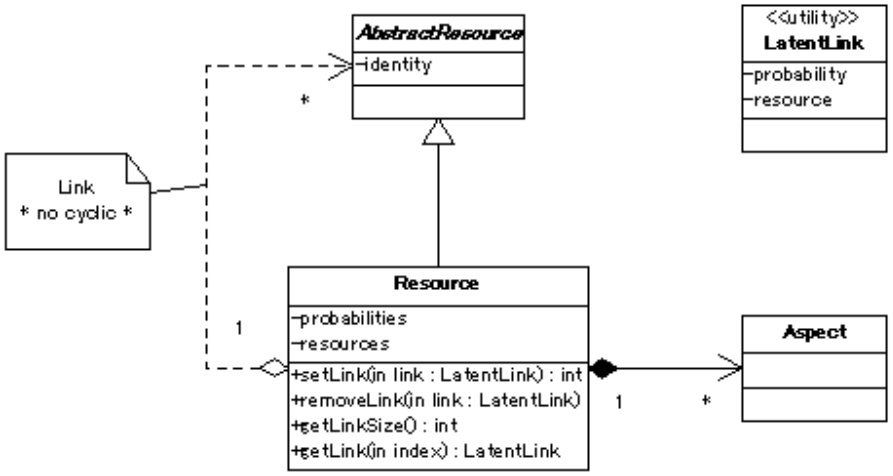

Figure 4.1: Latent Resource Link Discovery in UML

of Link, only a sub set of linked Resources, whose probabilities are over a certain threshold, are accessed. So list seems better, especially when its size is quite large. But I have an assumption that its number is limited to reasonable size that makes sense to load entirely. It is a duty of algorithm to pick up k important factors. Another reason is that Latent Resource Link Discovery is not cyclic, where sharing objects are not likely common.

4.2 Important Factor Analysis and Probability Calculation

I suppose an important factor analysis is done as shown below. The relation between Y and X may be linear or not.

$$Y = \Lambda X$$

Y is a matrix, whose j^{th} column is a vector representing a single sample measured during timespan S_j. And Y_{ij} is access count of i^{th} Resource during S_j.

X is a matrix, whose j^{th} column is a vector representing j^{th} Resource. And X_{ij} is access count prior to i^{th} Resource from j^{th} Resource.

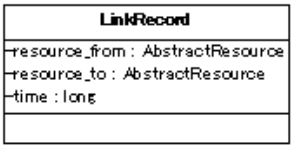

Figure 4.2: LinkRecord in UML

Λ is a matrix, whose j^{th} column is a vector representing a single sample measured during time span S_j. And Λ_{ij} is access probability to i^{th} Resource during time span S_j.

An algorithm will find such Λ to discover Latent Links between Resources. To form such an analysis, access history is stored as LinkRecord shown in Figure 4.2.

Chapter 5

For The Best Possible Performance Through Latent Link Discovery

Once you find latent links between resources, you are ready to get the best possible performance.

To evaluate performance, a model is required. Here, consider Risk Minimization Model. A risk is cost like disk access, file system access, query, networking, weave into aspect. The purpose of the model is to minimize sum of such costs for a certain time span.

- Locality On Low Level I/O
  - disk access cost
- Improved Data Structure
  - file system access cost
  - query cost
  - weave cost
- Improved Client Caching
  - networking

Related Objects mean those that belong to Resources, which are tied with high probability.

5.1 Locality On Low Level I/O

Disk seeks matter. By storing Related Objects contiguously on disk, the most expensive disk access cost is minimized. But changing physical locations are even more expensive. So it is not realistic to change often.

Another way is to read Related Objects all together. In Linux 2.6, even if they are on different blocks, they could be cached in a single cache page if they are accessed at the same time. So, depending on platform, it leads to fewer disk access by improved file system cache hits.

5.2 Improved Data Structure

From latent links through related aspects, you may find Related Objects are defined separately. By grouping them into the fewest possible Classes, overall database cache hits are improved.

Or you may find some of Related Objects belong to large objects. In this case, by putting apart such classes, only necessary objects are read. It leads to better database cache hits.

5.3 Improved Client Caching

The number of network round trips is critical. By fetching Related Objects at once, it can be minimized.