Substructure Similarity Measurement in Chinese Recipes

Liping Wang[1] Qing Li[1] Na Li[1] Guozhu Dong[2] Yu Yang[1]

[1] Department of Computer Science, City University of Hong Kong

[2] Dept of Computer Sci & Engr, Wright State University

stella.wang@gmail.com; {qing.li,nali25}@cityu.edu.hk; guozhu.dong@wright.edu; pauly.yang@gmail.com

ABSTRACT

Improving the precision of information retrieval has been a challenging issue on Chinese Web. As exemplified by Chinese recipes on the Web, it is not easy/natural for people to use keywords (e.g. recipe names) to search recipes, since the names can be literally so abstract that they do not bear much, if any, information on the underlying ingredients or cooking methods. In this paper, we investigate the underlying features of Chinese recipes, and based on workflow-like cooking procedures, we model recipes as graphs. We further propose a novel similarity measurement based on the frequent patterns, and devise an effective filtering algorithm to prune unrelated data so as to support efficient on-line searching. Benefiting from the characteristics of graphs, frequent common patterns can be mined from a cooking graph database. So in our prototype system called *RecipeView*, we extend the subgraph mining algorithm *FSG* to cooking graphs and combine it with our proposed similarity measurement, resulting in an approach that well caters for specific users' needs. Our initial experimental studies show that the filtering algorithm can efficiently prune unrelated cooking graphs without affecting the retrieval performance and the similarity measurement gets a relatively higher precision/recall against its counterparts.

Categories and Subject Descriptors

H.2.8 **[Database Management]**: Database Applications – Data Mining; H.3.3 **[Information Storage and Retrieval]**: Information Search and Retrieval – Information filtering

General Terms

Algorithms, Measurement

Keywords

Recipes, cooking graph, filtering, similarity measurement, subgraph mining.

1. INTRODUCTION

How to improve the precision of information retrieval has always been a challenging task for Chinese searching and recommendation systems, which is due to the rich yet implicit phrases/expressions in Chinese culture. On one hand, Chinese phrases/expressions can be very complicated thus rendering much burden on the searching systems, and on the other hand, the way local users interact with a Chinese website is quite different from that on an English website. To explore and elaborate this type of problems, in this paper we narrow down our focus on a particular domain, namely, Chinese recipes. Nevertheless, the methodology in this paper can also be applied to other domains with similar problems.

1.1 Motivation

Chinese cuisines are famous for their delicious taste resulted from delicate cooking skills and variant adaptation. The need of learning those cooking skills on the Web has been increasing in China; yet current on-line recipe search or recommendation systems only allow users to query by text, usually the names of the recipes. Unlike the names of western cuisines, which are typically constructed by 'past participle (cooking action) + main/minor ingredients', Chinese recipes' names are often injected with more meanings and/or bear no relationship with underlying cooking actions and ingredients:

- Some recipes are given auspicious names in hope that people who eat those dishes will be auspicious. An example of this type of recipes is 'Jin Yu Man Tang' (金玉滿堂)[ζ], which can actually refer to several recipes as long as their dish appearances mainly possess both colors of yellow and white: yellow represents gold and white represents jade, both mean wealthy. However, if searching this item on Google Search[Ψ], many words of blessings are retrieved, only few of these are related to recipes of this name.
- Some recipes are given names by anecdotes, because they are said to be originated from some folktales. For example, 'Kung-Pao Chicken' (宮保雞丁) — a popular dish of Sichuan style — is said to be first made by Ding Baozhen's chef and Ding was a late Qing Dynasty official. This dish was later named after Ding's official title 'Kung Pao' (宮保) when it was handed down from generation to generation.
- Some other recipes are given names simply by their appearance. For example, a recipe called 'Hu Die Gu' (蝴蝶骨) in Chinese means something like 'Butterfly Bone' in English. The reason for such an amazing expression is that the dish looks like several butterflies stopping on the plate. However, the correct name for this recipe in English is 'Braised Spare Ribs'. If searching this recipe in Chinese on Google Search, among the top 10 results, nearly no results are related to the recipe.

The above examples reveal the incapability of the traditional search method when it comes to finding results under such a special domain. Although, local government has noticed this cultural issue, and recently has taken the step to conform most

WWW 2008, April 21–25, 2008, Beijing, China.
ACM 978-1-60558-085-2/08/04.

[ζ] Our work is based on Chinese Web, but we translate some terms into English for readers' easier understanding.

[Ψ] http://www.google.com/

Chinese recipes' English translation [5], this problem is still prevalent currently on the Web with no good solution. In observation of the above situation, we advocate a recipe model for recipes, through which several structure features of graph can be exploited and utilized for similarity measurement, thereby improving the performance of recipe search or recommendation systems.

1.2 Related Work

The graph model has already been widely used in many domains. In the domain of chem-informatics and bioinformatics, ChemIDplus [6], a free data service offered by the National Library of Medicine (NLM), provides access to searching similar molecules by their substructure besides the traditional query by text. Cook *et al.* [2] focus on pattern recognition, and later to CAD circuit domain [4] in which structure similarity measure is used to discover similar designs. The DDIS group in Zurich [7] initiates the structure similar measure in ontology and workflows from the Web using their SimPack package. All those applications indicate the importance and wide usage of a graph model and its accompanied similarity measure sheds some light on similar search issues with respect to implicit structure similarity upon Chinese Web.

However, it is a commonly known fact that graph models have a big problem of computation with high complexity in terms of time and space; therefore, tailoring the graph model to the need of different applications is the first concern when building a graph database. Meanwhile, efficient indexing and filtering mechanisms can prune some unrelated graphs, thereby reducing the computational cost. Dennis *et al.* [16] build a path-based annotation by extracting subgraphs for graph matching, while Yan *et al.* [21] further apply feature-based indexing method on substructure patterns, in which a key issue was to generate optimal feature sets that could maximize the filtering capability.

Much work has been done to find similar parts between graphs. One approach is the *graph edit distance* [13, 15], in which a set of edit operations are introduced (e.g. deletion, insertion and substitution of vertices and edges) and the similarity of two graphs is defined in terms of the shortest (or least cost) sequence of edit operations which transforms one graph into the other. However, the problem on how those edit costs are obtained is still unsolved. Another approach for similarity search can be summarized as a subgraph isomorphism problem. A popular line of subgraph isomorphism is the well-known Maximum Common Subgraph algorithms [1, 3]. A comparison of three Maximum Common Subgraph (MCS) algorithms on a synthetic dataset is included in [3]. Kuramochi and Karypis [10] compare their method with another computationally efficient algorithm called AGM [8] on a real chemical dataset. Though efficient enough as they claimed, they still need days to compute on relative simple graphs, even though the dataset is of a medium size.

Similarity measures are commonly calculated after those similar parts are found. There are several similarity measures as described in [22], which can be classified into three categories: (1) physical property-based, e.g., toxicity and weight; (2) feature-based; and (3) structure-based. For the feature-based measure, domain-specific elementary structures are stored. Whether two graphs are similar or not is determined by the number of common features they share. For the structure-based similarity measure, direct comparison on the topology of two graphs is conducted. Several works have explored efficient similarity measures belonging to one of the above categories to adapt the data features. For example, the feature-based measure is used in [22] to roughly filter dissimilar sets as the first step, then the structure-based measure is used for more accurate search; that is because the latter measure can take structure connectivity into full consideration, leading to thus more accurate search than the former one does. Karakoc *et al.* [9] review the commonly used similarity measures for small molecules, and propose an improved algorithm on top of their study.

1.3 Our Contributions

In this paper, we explore the features of a Chinese recipe database, in which recipes are crawled from the Web by our *RecipeCrawler* [11]. To solve the long-standing retrieval problem on Chinese Web, we propose a framework to consider documents by their inner relationship or, in other words, inter-structural similarity.

A close study of the crawled recipes reveals that we can model each recipe as a *cooking graph* [12]. This cooking graph is different from a common workflow, since it includes multi-edges between each pair of vertices representing action-flows and ingredient-flows. This graph model is thus of a very general form that can be used in describing very complex processes in the real world.

In response to the demand of on-line performance, we propose in this paper a filtering algorithm to firstly prune dissimilar cooking graphs from the cooking graph database before doing further match. The filtering algorithm extracts representative features from graph databases, stores them in an inverted index and then prunes dissimilar graphs according to the index. We show that this filtering algorithm is efficient and scalable, therefore can be used for on-line search. Furthermore, compared with the feature extraction of [22] which is dependent on a specific knowledge base, our method is domain-independent, therefore can be applied to other similar structure (including workflows) involving directed graphs.

We use this filtering algorithm in our *RecipeView* system [12], upon which the similarity is computed between each pair of recipes. A new similarity measurement is introduced and we combine it with a subgraph mining algorithm (viz., FSG [10]) to conduct common subgraph detection. We make experiments in evaluating the filtering algorithm as well as our similarity measurement upon real-world Chinese recipes crawled from the Web, and highlight the importance of examining an increasingly popular search problem in a graph database exemplified by cooking graphs. To our best knowledge, this is the first piece of work to evaluate different similarity measures on this kind of graph data.

The rest of the paper is organized as follows. Section 2 defines the preliminary terminologies to be used in the rest of this paper; algorithms to be evaluated and our system framework are also summarized there. Section 3 and section 4 represent our filtering algorithm and a new similarity measurement in order. Then in section 5, we conduct several experiments to demonstrate the performance of our filtering algorithm as well as the new method for similarity measurement. A conclusion is made in section 6, along with some suggested directions for future work.

2. TERMINOLOGY AND CONCEPT

In this section, we introduce some necessary terminologies as well as the recipe model. Then a framework is described to provide an overview of the system and its mechanisms.

2.1 Basic Terminology

A graph consists of vertices which are connected by edges. In a labeled graph, vertices and edges are associated with attributes, called *labels*. The attributes in our cooking graph can be classified into two types, namely, actions and ingredients. A *directed edge* in a directed graph connects a source vertex and a target vertex which can be determined by the edge itself. In our filtering algorithm, we use the definition of this directed edge as a feature of an action order.

More formally, a graph $G = (V, E, L)$ consists of three sets *V, E, L,* denoting vertices, edges and labels of vertices, respectively. $V = \{v_i \mid i = 1...n\}$ is a set of vertices. E is a set of edges on V, and an edge (v_i, v_j) indicates a directed edge from vertex v_i to v_j, with v_i being called the predecessor of v_j, or in other words, v_j being the successor of v_i.

2.2 Recipe Model Simplified

It is a common sense that cooking involves a complex procedure, which may concern several aspects and their relationships. To describe this procedure, the system should give succinct while adequate knowledge of how to prepare ingredients and how to cook step by step, as well as the specific skill in each step. To this end, we present a simplified recipe model (in comparison with [12]) to concentrate on the procedural aspect.

Definition 2-1. A recipe R is defined by a tuple of two elements:

$$R = <RP, SP>$$

where:

- *RP* is a set of recipe-level properties (and functions) applied on *R* itself, such as cooking style, region and images of the dish of the recipe;
- *SP* = (*V, E, Time, Cons*) is a "Cooking Graph" which is a labeled directed graph describing the whole cooking procedure of making a dish out of recipe *R*. More specifically,
  - In *V*, each vertex v_i represents either a cooking action or a raw ingredient, associated with a unique timestamp $Time(v_i)$ (indicating the start time of v_i), and a set of constraints $Cons(v_i)$. For an action vertex, $Cons(v_i)$ defines constraint conditions that should be satisfied when the action v_i takes place. These constraints are called the 'cooking action constraints', such as the shape of an ingredient (parameter) or the temperature of the action that should be held. For example, for an action vertex 'cut', material can be cut into roasts, chops, cubes, strips or cuties. For an ingredient vertex v_i, $Cons(v_i)$ mainly defines the amount of the raw ingredient. A label $L(v_i)$ refers to the action/ingredient name of vertex v_i.
  - In *E*, the directed edges represent 'action flows' or 'ingredient flows'; the former describe the temporal execution sequence of cooking actions and the latter keep track of ingredient sources.

Example 2-1. Table 1 shows a cooking procedure of recipe 'Hu Die Gu' (蝴蝶骨) in Chinese (translated as 'Braised Spare Ribs' in English) crawled by our *RecipeCrawler* [11]. The cooking procedure is parsed into basic actions and their related properties as discussed in [19].

Table 1. Cooking procedure of 'Hu Die Gu' (in Chinese) —'Braised Spare Ribs' (in English)

Step #	Recipe cooking procedure in steps
1	Cut spare ribs into butterfly shape. Marinate with black pepper, soy sauce, sugar and cornstarch for 30 minutes.
2	Mix black pepper, honey, soy sauce, cornstarch and water.
3	Heat oil. Deep-fry the spare ribs when the oil is hot.
4	Remove the spare ribs when they turn brown.
5	Heat oil. When the oil is hot, add the mixed sauce and stir quickly. Boil the sauce.
6	Then add the spare ribs and stir briefly. Simmer for 1-2 minutes. Then remove.

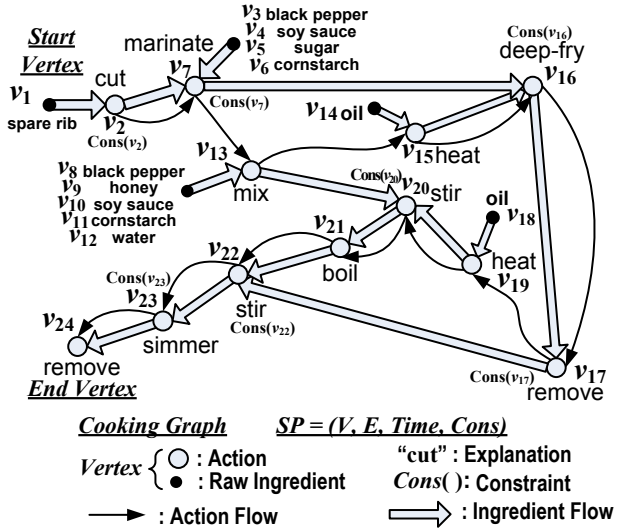


Figure 1. Cooking graph of 'Hu Die Gu' (in Chinese) —'Braised Spare Ribs' (in English)

In Figure 1, the corresponding cooking graph of "Hu Die Gu" is illustrated according to the recipe model. To differentiate the two types of vertices, actions such as 'cut' or 'deep-fry' are represented by white nodes and raw ingredients such as 'spare rib' or 'black pepper' are represented by black nodes. Here a raw ingredient means the ingredient has not been processed (in any action) before. For simplicity, only one ingredient vertex is shown when there is more than one raw ingredient used in a certain action. The detailed information of time and constraints for each vertex is not shown due to space limit. Here we assume $Time(v_i)<Time(v_j)$ for any $i<j$. From ingredient flows, we can see that the ingredients needed in a certain action are the ingredient outputs generated from its preceding actions or directly from some raw ingredients.

Figure 2 shows a simplified substructure of a cooking graph where two types of edges are unified into one type. Here a vertex v's predecessor p_i (i = 1, 2, …, m) points to v ($Time(p_i)<Time(v)$) and the latter is followed by a v's successor s_j (j = 1, 2, …, n) (i.e., $Time(v)<Time(s_j)$). Only the edge (p_m, v) is selected as the forward edge of v since p_m has the nearest timestamp to v among all v's predecessors.

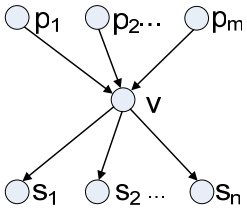


Figure 2. Substructure with predecessors and successors

According to Figure 2, we further define two notions below:

Definition 2-2. $Pred(v) = \{p_i \mid i = 1...m\}$ is the predecessor set of vertex v and $|Pred(v)| = m$ is defined as the number of predecessors in $Pred(v)$. If $m>1$, then for any adjacent pair p_i and p_{i+1} ($i = 1...m-1$), we have $Time(p_i)<Time(p_{i+1})$.

Definition 2-3. $Succ(v) = \{s_i \mid i = 1...n\}$ is the successor set of vertex v and $|Succ(v)| = n$ is defined as the number of successors in $Succ(v)$. If $n>1$, then for any adjacent pair s_i and s_{i+1} ($i = 1...n-1$), we have $Time(s_i)<Time(s_{i+1})$.

2.3 The Overall Framework

In our *RecipeView* system, the user first inputs a query by the name or description of a recipe, he/she can also quickly browse the recipe list in our system. The system then retrieves the result using the traditional method. If the user wishes to find recipes which have similar cooking skill or ingredient(s), he/she can choose 'similar recipe' to obtain the result.

At the backend of the system, we pre-compute the similarity between each recipe pairs. If new recipes are crawled from the web, additional indexes will be built accordingly, which facilitates fast computation of recipe similarities. The overall framework of our system can be best described through the following three processes, each of which is to be further elaborated in subsequent sections.

1. **Recipe Filtering** — to build the vocabulary of substructures of all cooking graphs: when a new cooking graph is added, simply extract the featured subgraph and add to the vocabulary; construct an inverted index to store the whole vocabulary; filter the most dissimilar recipe pairs using a threshold p.
2. **Recipe Similarity Measurement** — to compute each pair of recipes' similarity in the filtered graph set. In this work, we use the FSG [10] algorithm to compute common subgraph, followed by using our newly proposed measure to calculate the similarity.
3. **Recipe Retrieval** — to retrieve from the database similar recipes ranked according to the target/query recipe and evaluate the performance.

3. RECIPE FILTERING

In this section, we present our filtering algorithm to prune out dissimilar cooking graphs from the recipe (cooking graph) database. A string with q characters is called a q-gram. For two strings S_1 and S_2, Ukkonen [17] prove that the edit distance between S_1 and S_2 can be bounded by the difference within the number of q-grams. Rui [23] further apply the q-gram idea to tree structure, based on the observation that the number of edges in a tree is equal to the number of vertices minus one, so it can make various transformations to reorganize the tree-structure into a string-like form for q-gram analysis. However, graph structure is different from string or tree structure, as the number of edges in a graph may be several times larger than the number of vertices. For instance, a complete graph has n vertices and n*(n-1)/2 edges, with an exponential increase of the internal relationship information among the edges. Thus, unless we can tolerate losing some edge relationship information, it is very difficult to draw a succinct bound on graph edit distance using q-gram like algorithms designed for the strings or trees.

Nevertheless, our filtering algorithm adopts the idea of q-gram algorithm based on a careful observation of the relationship on the cooking graphs. We use *Predecessor ReciSets* and *Successor ReciSets* to retain the information of action/ingredient set, and *Forward Edge ReciSet* to record the most important preprocessing actions leading towards the next action/ingredient. Later experiments further show that, our system has a better pruning power than other representation models, and the system is also more sensitive towards the understanding of cooking graphs.

3.1 Cooking Subgraph Transformation

We observe that steps with multiple actions and/or ingredients are always considered as distinguished steps in a cooking graph. For example, action 'cut' is followed by action 'stir-fry' with 'sauce' added. Therefore, in the cooking graph, both the 'cut' and the 'sauce' are the predecessors of 'stir-fry' and 'cut' appears before adding 'sauce'.

To better abstract the cooking graph and retain the internal timeline relationship, we further simplify the graph by combining action flows with ingredient flows. In particular, if there exists an action edge along with an ingredient edge, in the same direction, between any two vertices, we just unify these two edges into one directed edge. More specifically, we use *ReciSet* to represent the temporal characters of the graph. Though it may look like feature-based at the first glance, *ReciSet* is in fact domain-independent, hence can be applied to other graphs in the indexing step, especially to graphs with interrelated time sequence.

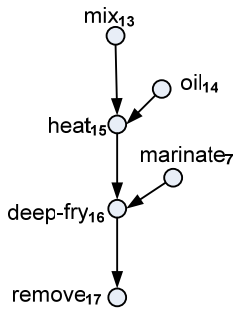
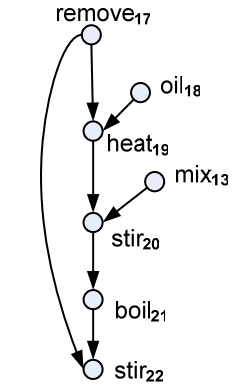
Definition 3-1. (*ReciSet Representation*) A *ReciSet* is in the format of $<T, v_1, v_2>$, where T is the property indicator that represents predecessor (P), successor (S) or forward edge (F) and v_1, v_2 are vertices.

For *Predecessor / Successor ReciSet*, v_1 and v_2 forms an adjacent pair (cf. Definition 2-2 / 2-3). For *Forward Edge ReciSet*, v_1 has the nearest timestamp to v_2 among all v_2's predecessors. For a certain cooking graph G, the total number of all *ReciSets* (including all three types—*Predecessor / Successor /Forward Edge ReciSet*) is at most twice the total number of edges in G. This proves the scalability of our filtering algorithm, the proof of which is omitted and readers can find the details from [19].

Table 2 provides two simplified graphs G_1 and G_2 with *ReciSet* representations against the example 'Hu Die Gu' in Figure 1 in which both graphs are the subgraphs derived from the example cooking graph. The subscript number in each vertex's label denotes the timestamp order which conforms to the vertex number in Figure 1. According to Definition 3-1, we can see that for vertex '$heat_{15}$' in the graph G_1, the predecessor vertex 'oil_{14}' has the nearer timestamp to '$heat_{15}$' than 'mix_{13}', thus we have the *Forward Edge ReciSet* <F, 'oil_{14}', '$heat_{15}$'> to indicate that the action 'heat' performs immediately after adding 'oil'. The *Predecessor ReciSet* <P, '$marinate_7$', '$heat_{15}$'> means that both 'marinate' and 'heat' are the predecessors of 'deep-fry' and 'marinate' appears earlier than 'heat'.

The *ReciSets* of all cooking graphs in the recipe database constitute the whole *ReciSet* vocabulary, and the *ReciSet* distance is defined by Definition 3-2.

Table 2. *ReciSet* representation

 Graph 1 (G_1)	RS_0: <P, 'marinate', 'heat'> RS_1: <P, 'mix', 'oil'> RS_2: <F, null, 'marinate'> RS_3: <F, null, 'mix'> RS_4: <F, null, 'oil'> RS_5: <F, 'oil', 'heat'> RS_6: <F, 'heat', 'deep-fry'> RS_7: <F, 'deep-fry', 'remove'>
 Graph 2 (G_2)	RS_8: <P, 'mix', 'heat'> RS_9: <P, 'remove', 'oil'> RS_{10}: <P, 'remove', 'boil'> RS_{11}: <S, 'heat', 'stir'> RS_3: <F, null, 'mix'> RS_{12}: <F, null, 'remove'> RS_4: <F, null, 'oil'> RS_5: <F, 'oil', 'heat'> RS_{13}: <F, 'heat', 'stir'> RS_{14}: <F, 'stir', 'boil'> RS_{15}: <F, 'boil', 'stir'>

Definition 3-2. (*ReciSet Space*) The *ReciSet Space* *RSS*(G) of a graph G is the set of all *ReciSets* that appear at least once in G. $|RSS(G_1)|_{G_2}$ denotes the number of *RSS*(G_1) found in G_2 and $\|RSS(G_1)\|_{G_2}$ denotes the number of the occurrence of *RSS*(G_1) found in G_2.

Definition 3-3. (*ReciSet Distance*) The *ReciSet Distance* from graph G_1 to G_2 is calculated on the *ReciSet Space* of G_1. Let RS_i be the i-th *ReciSet* in *RSS*(G_1), then r_i and r_i' are the numbers of occurrences of RS_i in G_1 and G_2, respectively. Then the *ReciSet Distance* is obtained by:

$$RDist(G_1, G_2) = \sum_{i=1}^{|RSS(G_1)|_{G_1}} \left| r_i - r_i' \right|.$$

Definition 3-4. (*Shared Percentage*) For any two cooking graphs G_1 and G_2, the *Shared Percentage* *Per*(G_1, G_2) is defined as the percentage of the number of the occurrence of *RSS*(G_1) found in G_2 to the total number of occurrence of the *ReciSets* in G_2, i.e.,

$$Per(G_1, G_2) = \frac{\|RSS(G_1)\|_{G_2}}{\|RSS(G_2)\|_{G_2}}.$$

Obviously, *RDist*(G_1, G_2) may not be equal to *RDist*(G_2, G_1), as for different G_1 and G_2, their corresponding *ReciSet Spaces* may not contain the same *ReciSets*. Also, the triangular inequality may not hold, so that *ReciSet Distance* is not a metric on graph data.

In the cooking graphs, given the same distance value, having a small *Per*(G_1, G_2) value indicates that G_1 is only a small portion of G_2. The graphs themselves are not similar to each other, hence should not be given a low distance value. In Section 5 we shall demonstrate that the value of *Per*(G_1, G_2) can reflect the percentage of retrieved data quite well.

3.2 Cooking Graph Indexing

As mentioned earlier, an inverted index is constructed to store the whole vocabulary of all *ReciSets* of the recipe database. For each *ReciSet*, an inverted list is built to store the number of its occurrences in the corresponding graph. For example, Figure 3 shows the inverted index of G_1 and G_2 on the *ReciSet Space* of G_1 (*RSS*(G_1)) according to Table 2. There are 8 *ReciSets* of G_1 and 3 *ReciSets* of G_2 on *RSS*(G_1) in which each *ReciSet* occurs only once. We can see that three *ReciSets* (RS_3, RS_4 and RS_5) occur in both graphs G_1 and G_2. So the *ReciSet Distance* *RDist*(G_1, G_2)= 8–3=5 and the *Shared Percentage* *Per*(G_1,G_2)=3/11=27.3%.

Based on the inverted index, the filtering algorithm can be applied to search, after which similarity measure can be easily implemented on the candidate answer set. We will examine the actual performance in Section 5.

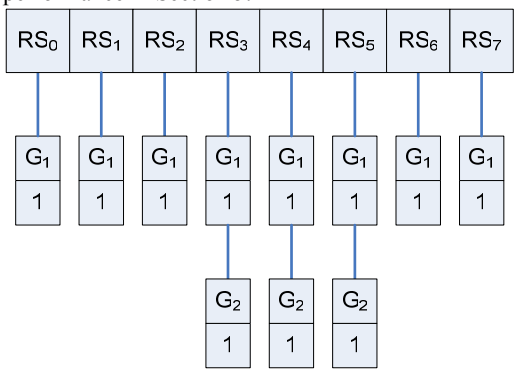


Figure 3. Inverted index representation

3.3 The Filtering Algorithm

Algorithm 1. Filtering algorithm

Input: The inverted index I,
The query Q.
Output: Candidate answer set C_Q.

```
1:  Get the ReciSet Space of Q;
2:  S_q = ||RSS(Q)||_Q;
3:  for each ReciSet R_i in the ReciSet Space of Q do
4:    occ_i = the number of occurrence of R_i in Q;
5:    if R_i is in the inverted index then
6:      for each Graph G_j in the inverted list of R_i do
7:        S_j=||RSS(G_j)||_Gj ;
8:        occ_j = the number of occurrence of R_i in G_j;
9:        if G_j is not in C_Q then
10:         Record G_j into C_Q;
11:         Record RDist(Q, G_j)=S_q-occ_i + |occ_i - occ_j|;
12:         Record Per(Q, G_j)= occ_j/S_j;
13:       else
14:         Get RDist(Q, G_j), Per(Q, G_j) from C_Q;
15:         Update RDist(Q, G_j) -= occ_i - |occ_i - occ_j|;
16:         Update Per(Q, G_j) += occ_j/S_j;
17: return C_Q;
```

Given a query graph Q and the inverted index I built for the recipe database, **Algorithm 1** is formed to generate the candidate answer set C_Q. Each graph in the answer set is associated with its *ReciSet Distance* and *Shared Percentage*. First the *ReciSet Space* of the query Q is acquired. Before a graph G_j in the dataset is visited in the algorithm, we consider it as an empty graph. The initial distance $RDist(Q, G_j)$ is therefore set to $\sum_{i=1}^{|RSS(Q)|_Q} |r_i - 0| = \|RSS(Q)\|_Q$, and the initial *Shared Percentage* $Per(Q, G_j)$ is set to 0. For each *ReciSet* R_i that is in the *ReciSet Space* of Q and each graph G_j that is found in the inverted list of R_i, if G_j is not in the candidate answer set C_Q, the initial distance from Q to G_j on R_i should be cleared. At each round $RDist(Q, G_j)$ is updated to add the difference of the occurrence of R_i in Q and G_j, and the *Shared Percentage* $Per(Q, G_j)$ is updated to add the number of the occurrence of R_i in Q divided by $\|RSS(G_j)\|_{G_j}$. The final values of *ReciSet Distance* and *Shared Percentage* should be derived once all the related *ReciSets* are processed.

4. RECIPE SIMILARITY MEASUREMENT

Experienced cooks always follow certain cooking patterns to prepare their meals, including some common actions and ingredients. If a certain cooking process frequently appears in many recipes, this process can be considered as a cooking technique/skill. Once a user learns how to handle those techniques, he/she will be able to handle many recipes or even to create a new dish by himself/herself. Observing this, we believe that the function to find similar recipes *structurally* is important. Though the goal is not easy to achieve, our *RecipeView* attempts to reveal all frequently used cooking processes for users to access and learn.

4.1 Discovery of Cooking Subgraphs

We define a *Cooking Pattern* as a subgraph G' such that (i) G' occurs in more than k cooking graphs, where k is a threshold, and (ii) G' has at least one action/ingredient edge.

However, existing subgraph mining algorithms such as FSG [10] or gSpan [20] do not support directed graphs in which multiple edges can exist between a given pair of vertices. As it is a distinct characteristic of our cooking graphs, we develop an extended approach to make existing subgraph mining algorithm applicable to our cooking graphs.

The extensions and the process of discovering common cooking subgraphs are described as follows:

1. A "dummy" vertex v_{A1} (cf. Figure 4) is added between two vertices v_1 and v_2, if there is an action edge from v_1 to v_2. We set the label $L(v_{A1})=L(v_1)+'\_'+L(v_2)$.

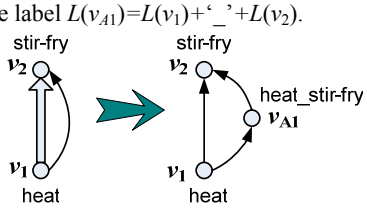


Figure 4. Dummy vertex insertion

Therefore, all edges belong to one type of edges. If there is an edge directly from v_1 to v_2, this edge represents an ingredient flow from v_1 to v_2. For a dummy vertex v_{A1}, if there exists an edge from v_1 to v_{A1} and an edge from v_{A1} to v_2, then there is an action flow from v_1 to v_2.

2. The subgraph mining algorithm such as FSG is applied to the 'modified' cooking graphs to find cooking patterns.
3. After graph mining, three steps are needed for restoration:
   (1). As the derived patterns from FSG are undirected, graph matching using Ullmann's algorithm [18] is conducted and edge direction check (as shown in Figure 5) needs to be processed.
   For explanation, we take one of the derived patterns and its candidate cooking graphs which are known to contain this pattern as an example. First we apply the derived pattern to map a candidate cooking graph. Once all the vertices and edges can be mapped one by one in the candidate cooking graph, we assign the directions of all edges in the pattern same as those of the mapped edges in the cooking graph. We take this pattern as a *recorded pattern*. Next we use the *recorded pattern* to map to another candidate cooking graph. If we can't get an exact mapping, it means that the directions of the edges in the *recorded pattern* are not totally the same as those in this candidate cooking graph (e.g., the *recorded pattern* as in Figure 5(b) and the candidate cooking graph as in Figure 5(c)). Then the original undirected pattern should be used to make a mapping to the candidate graph. The result pattern is considered as another *recorded pattern*.
   If all candidate cooking graphs get mapped, we remove any *recorded pattern* if it does not satisfy a given minimal support threshold. Then the remaining record patterns are what we need.

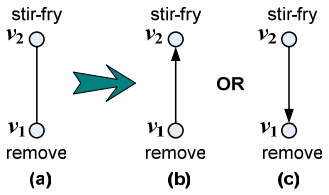


Figure 5. Edge direction check

   (2). Remove dummy vertices: to remove all the dummy vertices so as to restore the action flows and ingredients flows in the pattern.
   (3). Remove duplicated patterns: after we remove all the dummy vertices, some duplicated patterns may occur. So we have to check and remove them.

Note that there could be more than one cooking technique used in a particular recipe, and our system is capable of identifying all of them so as to enable the users to know how many and what patterns are contained in the recipe. By learning the combined sequence of different patterns in the same cooking graph, a user can get a rough sense of the structure and the style of the recipe/dish.

4.2 Recipe Similarity Calculation

Based on the structure of cooking graphs, we proceed to propose a novel graph-based similarity calculation method which is radically different from normal text-based or content-based approaches. Using this method, users can perform similarity search over the graph structure, shared characteristics, and distinct characteristics of each recipe.

Suppose SP_1 is the cooking graph of recipe R_1 and SP_2 is the cooking graph of recipe R_2. Assuming SP_1 and SP_2 share m

subgraphs SP_{Si} (i = 1 .. m), then the structure similarity of recipes R_1 and R_2 is calculated as follows:

$$sim(SP_1, SP_2) = \left[\left(\sum_{i=1}^{m} |E_{Si}|\left(\mu|E_{SAi}| + \gamma|E_{SIi}|\right)\cdot \log_2 \frac{N}{d_{Si}}\right)\cdot Per(SP_1, SP_2)\right]^{1/2} \quad (1)$$

where $|E_{Si}|$ is the number of edges (including both action and ingredient edges) of the subgraph SP_{Si}; $|E_{SAi}|$ and $|E_{SIi}|$ are the numbers of action and ingredient edges in SP_{Si} respectively (i.e. $|E_{Si}| = |E_{SAi}|+|E_{SIi}|$); N is the total number of recipes; d_{Si} is the number of cooking graphs that contain subgraph SP_{Si}; $\log_2(N/d_{Si})$ is the inverse subgraph frequency, the purpose of which is to make rare common subgraphs more important than frequent ones. Since different user queries may emphasize more on either the flavor (affected by ingredient flow) or the cooking skill (affected by action flow), we use μ and γ as the weights for action and ingredient edges, which are adjustable. $Per(SP_1, SP_2)$ is pre-calculated in graph filtering; intuitively, the larger $Per(SP_1, SP_2)$ is, the more similar SP_1 and SP_2 look like to each other.

In the above formula, the similarity of two cooking graphs is calculated based on the common subgraphs shared by them[ξ]. It takes both local and global views into consideration. $Per(SP_1, SP_2)$ shows the similarity based on the paired graphs. If the two graphs are the same, their $Per(SP_1, SP_2)$ is one. On the other hand, $\log_2(N/d_{Si})$ considers the importance of the subgraph in the whole graph database. A popular action (e.g. heat oil) in two graphs is not distinguishing enough to conclude if the two recipes are really similar to each other. In sum, the higher the score is, the more similar two cooking graphs resemble each other. By using this method, users are able to retrieve similar recipes in terms of cooking processes even though the names of the recipes may be totally different.

5. EXPERIMENT

As part of our research, we have conducted a series of four experiments upon our *RecipeView* system. The first two belong to performance study, in which we examine the efficiency of our filtering algorithm, as well as comparing it to other ones. The next two actually run two implemented algorithms comprehensively on the *RecipeView* system to examine their retrieval effectiveness and efficiency.

The judgement of determining whether a recipe is relevant to the queried recipe is somewhat different from traditional ones. For the vector space model [14], only word frequency is mainly considered. But in our judgement, not only ingredients are considered but also the way of cooking (i.e. cooking procedure) is taken into account. For example, 'Chengdu Young Chicken' and 'Fried Spareribs in Orange Juice' are regarded as relevant because they share most of the cooking procedures (cf. Figure 8).

5.1 Filtering Algorithm Performance Study

In this part of the experiments, we conduct the empirical studies to examine the properties of our filtering algorithm. We use a recipe database containing 103 Chinese cooking graphs (with 51 Guangdong style dishes and 52 Sichuan style dishes), which are some of the most representative and popular recipes in Chinese cuisine. These cooking graphs consist of 34.4 vertices on average (ranging from 15 to 56), and 45.2 edges on average (ranging from 20 to 80). After segmentation, a vocabulary of 5739 *ReciSets* is generated.

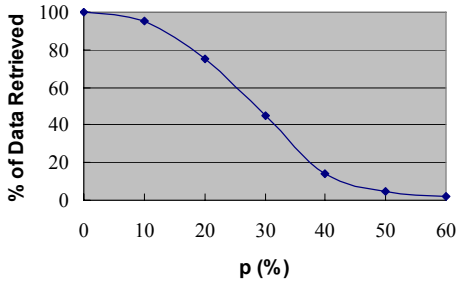


Figure 6. Percentage of data retrieved for *Per(Q, *)*≥ *p*

First let's check the performance of *Shared Percentage Per*(*Q, **) where * denotes the graphs to be compared with the query graph *Q*. Here we temporarily exclude out *ReciSet Distance* (*RDist*) from consideration, but we will add this factor in the later experiment. Figure 6 shows that the percentage of data (cooking graphs) is retrieved when the condition $Per(Q, *) \geq p$ holds with the variable p. When p equals to 0 (i.e. *Per*(*Q, **) does not have any effect), the candidate set contains a large number of related cooking graphs of *Q*. We can see that in this data set, all cooking graphs share some common parts with one another. When p is increased from 20% to 40%, the size of the candidate set decreases sharply from 75% down to 14% of the original set. When p reaches 60%, the candidate set is almost empty. The result shows that p has a strong pruning power in filtering out graphs.

We further calculate the precision, recall and F-measure to see if the retrieval performance is improved by our filtering algorithm. Here:

$$\text{Precision } P = \frac{\text{Total no. of relevant cooking graphs retrieved}}{\text{Total no. of cooking graphs retrieved}}$$

$$\text{Recall } R = \frac{\text{Total no. of relevant cooking graphs retrieved}}{\text{Total no. of relevant cooking graphs}}$$

$$\text{F - measure} = \frac{2 \cdot P \cdot R}{(P + R)}$$

Note that the F-measure is the weighted harmonic mean of precision and recall (i.e. precision and recall are evenly weighted); the larger the F-measure is, the better the performance is.

Table 3. Precision and recall for *Per(Q, *)*≥ *p*

p (%)	0	10	20	30	40	50
Precision	0.207	0.217	0.274	0.425	0.585	0.515
Recall	1	1	0.977	0.882	0.453	0.126
F-measure	0.343	0.356	0.428	0.574	0.511	0.203

Table 3 shows that the retrieval performance is obviously improved (p=20%~ 40%) after filtering.

In the following experiment, we add the factor of *ReciSet Distance RDist*(*Q, **). The cooking graphs are retrieved under the conditions $Per(Q, *) \geq p$ and $RDist(Q, *) \leq (1-x)MaxRDist(Q, *)$ where *MaxRDist*(*Q, **) is the maximum value of *RDist*(*Q, **) corresponding to the query graph *Q*. Table 4 shows the values of F-measure when p ranges from 0 to 50% and x ranges from 10% to 25%.

[ξ] Note that each part is normalized before multiplication.

Table 4. F-measure for both *Per*(*Q*, *)≥ *p* and *RDist*(*Q*, *)≤(1−*x*)*MaxRDist*(*Q*,*)

x(%) \ *p*(%)	0	10	20	30	40	50
10	0.438	0.439	0.457	0.577	0.512	0.203
15	0.496	0.497	0.497	0.583	0.505	0.203
20	0.541	0.542	0.537	0.594	0.513	0.203
25	0.560	0.560	0.562	0.581	0.468	0.205

The values of F-measure are generally larger than those in Table 3, which means the retrieval gets better performance when *ReciSet Distance* is taken into consideration. The optimal value of *p* and *x* can be found in the shaded area (*p*=20%~40% and *x*=15%~25%) to get the largest value of F-measure. After calculating all the shaded area, the largest value of F-measure (=0.623) is found when *p*=35% and *x*=18% (precision=0.576, recall=0.677). Under this condition, about 28% cooking graphs have been retrieved.

This demonstrates the efficiency and suitability of our algorithm for on-line graph-matching applications, particularly Chinese recipe retrieval.

5.2 Further Evaluation

In this second part of experiments, we first try to determine the weights μ and γ for Formula (1) by conducting a set of experiments on our recipe data. First we fix μ to 1 and change the value of γ so that we can see the performance of precision and recall under different values of γ. We calculate the precision and recall based on the top 10 retrieved recipe results because users usually view the top 10 results with interests and may not be patient for the later results.

Table 5. Comparison of precision and recall under different γ (μ=1)

γ	0.1	0.5	1	2	10
Precision@10	0.716	0.734	0.778	0.771	0.747
Recall@10	0.313	0.321	0.362	0.354	0.326
F-measure	0.436	0.447	0.494	0.485	0.454

According to the experiment results (Table 5), the precision is over 0.7 under each γ (γ varies from 0.1 to 10 where μ=1), which means that this similarity calculation method gets a good retrieval performance in general for top 10 results. When γ=1, F-measure gets the best result. So we use μ=1 and γ=1 and Formula (1) becomes the following:

$$sim(SP_1, SP_2) = \left[\left(\sum_{i=1}^{m} |E_{Si}|^2 \cdot \log_2 \frac{N}{d_{Si}} \right) \cdot Per(SP_1, SP_2) \right]^{1/2} \quad (2)$$

Then we further compare Formula (2) with Formulas (3) and (4) below:

$$sim(SP_1, SP_2) = \left[\left(\sum_{i=1}^{m} |E_{Si}| \cdot \log_2 \frac{N}{d_{Si}} \right) \cdot Per(SP_1, SP_2) \right]^{1/2} \quad (3)$$

$$sim(SP_1, SP_2) = 1 - \frac{|\mathrm{mcs}(SP_1, SP_2)|}{\max(|SP_1|, |SP_2|)} \quad (4)$$

The difference between Formula (2) and (3) is that $|E_{Si}|$ is squared in Formula (2). Formula (4) is the typical graph distance measure based on the maximal common subgraph (MCS) [1], for which the smaller the graph distance is, the more similar two graphs look to each other. $|\mathrm{mcs}(SP_1,SP_2)|$ denotes the number of vertices in the maximal common subgraph of two graphs SP_1 and SP_2. $|SP_1|$ and $|SP_2|$ are the numbers of vertices in SP_1 and SP_2, respectively. Then $\max(|SP_1|,|SP_2|) = \begin{cases} |SP_1| & \text{if } |SP_1| > |SP_2| \\ |SP_2| & \text{if } |SP_2| > |SP_1| \end{cases}$.

Table 6. Comparison of precision/recall for Formulas (2) – (4)

Formula	(2)	(3)	(4)
Precision@10	0.778	0.712	0.600
Recall@10	0.362	0.334	0.281
F-measure	0.494	0.455	0.382

In Table 6, we list the precision and recall over the top 10 retrieved recipe results by using Formulas (2) – (4) separately. Obviously Formula (2) performs significantly better than either Formula (3) or Formula (4), so we adopt Formula (2) as our recipe similarity measure.

Next we compare our graph-based similarity measurement with text similarity measurement (using baseline vector space model [14]). We evaluate the performance of both algorithms using traditional precision and recall. At the end, we present a comprehensive case to facilitate a better understanding of our whole system.

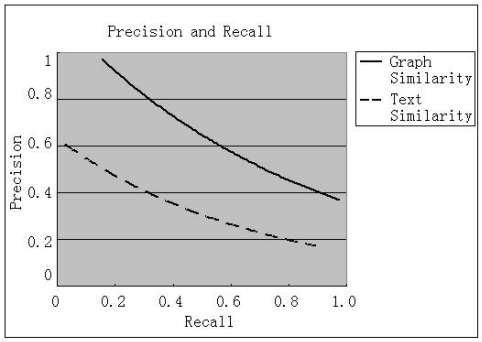


Figure 7. Precision and recall curve for graph similarity measurement and text similarity measurement

From Figure 7, we can conclude that our graph-based similarity measurement gains a far better improvement in retrieval performance than text similarity measurement. Under the same recall rate, the former has an approximately twice precision rate as the latter in the interval from 0.2 to 0.9. The reason for this improvement mainly attributes to the cooking graph structure. Taking the advantage of abstracting the workflow of a recipe document into a graph, it contains more semantic information describing the procedure of data, when compared with simple statistical text representation of the vector space model. Consider, e.g. the two recipes 'Kung Pao Pork' and 'Kung Pao Chicken' which are very similar (by our graph-based similarity measure in terms of cooking procedure) albeit with different main ingredients, the vector space model may not give this pair a high similarity value because different terms/main ingredients are used in each recipe; it may also suffer from correlation problems. In addition, another reason may be due to the different pre-processing we conduct. For the vector space model, we only remove stemming words, such as 'a', 'the', as well as html tag. While for graph conversion, our (currently) manual process can cleanse most (if not all) of the dirty data. Consequently, the retrieval result (which is ranked by similarity score) shows that the top results from the graph similarity method are very similar to the query recipe, which is not the case in the text similarity

measurement. This result further confirms that using cooking graphs is the major reason for the performance improvement.

To provide a more complete picture of our *RecipeView* system, we give an example in Figure 8 which shows a query 'Fried Spareribs in Orange Juice' submitted by a user and the results derived from the system. In particular, the system finds the recipe upon receiving the query string, and then the particular recipe graph is retrieved and displayed in the left window. Meanwhile, similar recipes are listed on the right column with 10 recipes per page. The corresponding cooking graph is shown in the 2nd (from left) window if the user clicks on one of them. Common parts between the two recipes are highlighted in orange color. If a user moves the mouse across the vertices or edges, some popup information is displayed indicating the associated constraints. For the two cooking graphs in Figure 8, it can be seen that the two recipes are structurally rather similar to each other (except for the main ingredient and some minor ingredients), even though their names are totally unrelated. This kind of 'substructure similarity' search would not be possible by using a traditional method. Moreover, by carefully examining the top 10 recipes returned, we find that they all belong to the same cooking style, and have some important actions/ingredients in common. This demonstrates the effectiveness of using the frequent common graph-based graph matching, based on which interesting and semantically meaningful results are obtained. However, the action 'deep-fry' in the left window is not regarded the same as 'stir-fry' in the right window (without an orange color in the vertex) by FSG. Therefore, more work could be done to make FSG more error-tolerant.

6. CONCLUSION

Improving the precision of information retrieval has been a challenging issue on Chinese Web. As exemplified by Chinese recipes on the Web, it is not easy/natural for people to use keywords (e.g. recipe names) to search recipes, since the names can be literally so abstract that they do not bear much, if any, information on the underlying ingredients or cooking methods. In this paper we have explored and elaborated this type of problems in the context of *RecipeView*, aiming to solve complicated semantic computing problems as exemplified by retrieving Chinese recipes on Chinese Web. In particular, we draw our effort on translating recipes into directed graphs, and making use of a graph mining approach to extract useful features (patterns). To accommodate powerful and semantically meaningful search that can better cater for specific user needs, we have introduced and incorporated into our *RecipeView* system a filtering algorithm and a new similarity measurement suitable for complex graphs embodying cooking graphs. It has been demonstrated that our filtering algorithm is efficient in facilitating the process of similarity search; in addition, its scalability allows an application to update similarity records on the fly. Our newly proposed similarity measurement features graph structure well, and can be combined with frequent subgraph mining to handle graph-based similarity search. Based on the *RecipeView* prototype system, we have tested the precision /recall based on our method compared to another graph matching approach (MCS). The results also shed light on the suitability and usability of different algorithms for applications involving complex graph data.

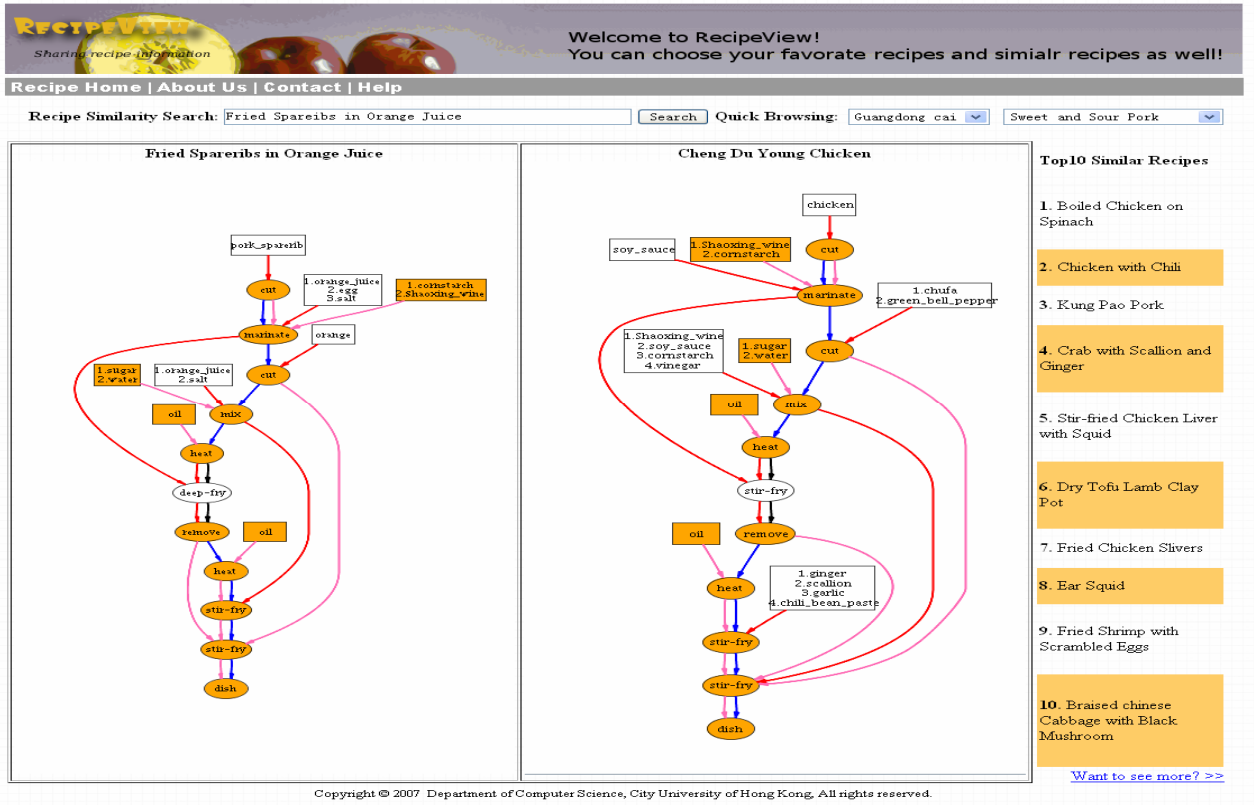


Figure 8. *RecipeView* screenshot: the target recipe (left window) with its most similar recipe (right window), and other top 10 similar recipes (shown on the right column)

In addition to recipes, any other types of data with operational sequence features can make use of the modeling framework we have proposed, including for example Web services and medical care domains. Our future work lies in data quality assurance for such application domains, in particular (semi-)automatic dirty data identification and correction. Another interesting issue for future research is to support (personalized) user adaptations and provide recommendations for error handling in the course of recipe execution.

7. ACKNOWLEDGMENT

Part of the work was done while Guozhu Dong was visiting the Zhejiang Normal University, where he is a guest professor. The work described in this paper has been supported, substantially, by a grant from the Research Grants Council of the HKSAR, China [Project No. CityU 117405], and a Strategic Research Grant of City University of Hong Kong [No. 7002212].

8. REFERENCES

[1] Bunke, H., and Shearer, K. A graph distance metric based on the maximal common subgraph. Pattern Recogn. Lett. 19, 3-4 (1998), 255-259.

[2] Cook, D. J., and Holder, L. B. Substructure discovery using minimum description length and background knowledge. Journal of Artificial Intelligence Research 1 (1994), 231-255.

[3] Conte, D., Guidobaldi, C., and Sansone, C. A comparison of three maximum common subgraph algorithms on a large database of labeled graphs. In Proc. of the 4th IAPR International Workshop on Graph Based Representations in Pattern Recognition (GbRPR), York, UK, 2003, pp. 589-607.

[4] Djoko, S., Cook, D. J., and Holder, L. B. An empirical study of domain knowledge and its benefits to substructure discovery. IEEE Transactions on Knowledge and Data Engineering 9, 4 (1997), 575-586.

[5] Government News. http://www.cq.xinhua.org/food/200801/15/content_12221389.htm

[6] Homepage ChemIDPlus. http://chem.sis.nlm.nih.gov/chemidplus/.

[7] Homepage Simpack. http://www.ifi.unizh.ch/ddis/simpack.html.

[8] Inokuchi, A., Washio, T., and Motoda, H. An apriori-based algorithm for mining frequent substructures from graph data. In Proc. of the 4th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD), London, UK, 2000, pp. 13-23.

[9] Karakoc, E., Cherkasov, A., and Sahinalp, S. C. Novel approaches for small biomolecule classification and structural similarity search. SIGKDD Explor. Newsl. 9, 1 (2007), 14-21.

[10] Kuramochi, M., and Karypis, G. Frequent subgraph discovery. In Proc. of the IEEE International Conference on Data Mining (ICDM), San Jose, USA, 2001, pp. 313-320.

[11] Li, Y., Meng, X., Wang, L., and Li, Q. *RecipeCrawler*: collecting recipe data from www incrementally. In Proc. of the 7th International Conference on Web-Age Information Management (WAIM), Hong Kong, China, 2006, pp. 263-274.

[12] Wang, L., Li, Q. A personalized recipe database system with user-centered adaptation and tutoring support. In ACM SIGMOD Ph.D. workshop on Innovative database research (IDAR), 2007.

[13] Messmer, B. T., and Bunke, H. A new algorithm for error-tolerant subgraph isomorphism detection. IEEE Trans. Pattern Anal. Mach. Intell. 20, 5 (1998), 493-504.

[14] Salton, G., Wong, A., and Yang, C. S. A vector space model for automatic indexing. Commun. ACM 18, 11 (1975), 613-620.

[15] Sanfeliu, A., and Fu, K. S. A distance measure between attributed relational graphs for pattern recognition. IEEE Transactions on Systems, Man and Cybernetics 13, 5 (1983), 353-362.

[16] Shasha, D., Wang, J. T. L., and Giugno, R. Algorithmics and applications of tree and graph searching. In Proc. of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS), New York, USA, 2002, pp. 39-52.

[17] Ukkonen, E. Approximate string-matching with q-grams and maximal matches. Theor. Comput. Sci. 92, 1 (1992), 191-211.

[18] Ullmann, J. R. An algorithm for subgraph isomorphism. J. ACM 23, 1 (1976), 31-42.

[19] Wang, L. *CookRecipe* - towards a versatile and fully-fledged recipe analysis and learning system. Ph.D. thesis, Department of Computer Science, City University of Hong Kong, Hong Kong (Jan. 2008).

[20] Yan, X., and Han, J. gSpan: Graph-based substructure pattern mining. In Proc. of the IEEE International Conference on Data Mining (ICDM), Washington DC, USA, 2002, p. 721.

[21] Yan, X., Yu, P. S., and Han, J. Graph indexing based on discriminative frequent structure analysis. ACM Trans. Database Syst. 30, 4 (2005), 960-993.

[22] Yan, X., Yu, P. S., and Han, J. Substructure similarity search in graph databases. In Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD), New York, USA, 2005, pp. 766-777.

[23] Yang, R., Kalnis, P., and Tung, A. K. H. Similarity evaluation on tree-structured data. In Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD), New York, USA, 2005, pp. 754-765.