Creating New Sandwiches From Old

Derek Bridge and Henry Larkin*

Insight Centre for Data Analytics,
School of Computer Science and Information Technology,
University College Cork, Ireland
derek.bridge@insight-centre.org|h.larkin@umail.ucc.ie

Abstract. Earl is a case-based reasoning system which works in the domain of sandwich recipes. It has a case base of 9,507 cases, each being a sandwich recipe which we scraped from the web. We show how Earl can compose new sandwiches from the ingredients of sandwiches that it retrieves from its case base. On each iteration, it inserts into the sandwich the candidate ingredient which results in greatest coherence. Earl computes coherence with reference to the cases in the case base. A preliminary experiment with human participants shows that Earl's sandwiches are competitive with human-authored sandwiches.

1 Introduction

The Computer Cooking Contest has its roots in Case-Based Reasoning (CBR). Most entrants to the Contest are systems that retrieve and adapt recipes from a case base of human-authored recipes. To the best of our knowledge, no entrant to the Contest has ever been capable of creating new recipes, rather than making small substitutions to existing recipes. This is surprising, considering that one of the most famed early CBR systems, CHEF, used case-based planning to create new recipes [3].

In this paper, we describe Earl. Like CHEF, Earl is case-based and it creates new recipes. Unlike CHEF, it is not a planner. It uses a form of constructive adaptation [5]. It composes a new recipe from the ingredients of multiple recipes that it retrieves from its case base. On each iteration, Earl inserts the candidate ingredient to the sandwich which results in greatest coherence. Earl computes coherence with reference to the cases in the case base. The design of Earl owes much to Baccigalupo and Plaza's case-based system for creating music playlists [2].

It is important to note, however, that Earl is not a general-purpose recipe creation tool. It works in a highly circumscribed domain: that of sandwiches. Creating sandwich recipes is obviously much easier than creating recipes for other meals. Nevertheless, Earl represents a first step in a different direction for the Computer Cooking Contest.

* This publication has emanated from research supported in part by a research grant from Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289.

Section 2 describes Earl's recipe case base; Section 3 presents Earl's algorithm for creating new sandwich recipes; Section 4 reports a simple experiment; Section 5 considers avenues for future inquiry.

2 The Sandwich Case Base

To build Earl, we needed a large case base of sandwich recipes. We entered the search term 'sandwich' into five recipe web sites: allrecipes.com, allthecooks.com, tastykitchen.com, food2fork.com and yummly.com.[1] We scraped 10,721 recipes from the search results. Even though some of the search engines allowed us to exclude desserts, cakes and so on from the search, the search results nevertheless contained many recipes with 'sandwich' in the title that were not sandwiches (e.g. "Victoria sponge sandwich" and "ice-cream cookie sandwich"). We manually removed these as best we could, resulting in a case base of 9,507 sandwich recipes. We stored each recipe's name, star rating and list of ingredients. Over 61% of recipes had a rating greater than or equal to four stars. Earl uses only these recipes, meaning that it makes use of about 5,800 recipes.

Earl relies on identifying sets of ingredients that recur across recipes (see Section 3). But the recipes we had scraped often described the same ingredient in different ways. Recipe authors often use alternative names for an ingredient (e.g. "arugula" and "rocket"); they might use brand names (e.g. "Hellman's"); they might misspell or mistype ingredients (e.g. "mayonaise"); and they often include measurements and preparation details (e.g. "two tablespoons cream cheese, softened").

To enable Earl to identify recurring ingredients, we linked each ingredient in each recipe to a canonical ingredient; for example, "mayonnaise", "mayonaise", "Hellman's" and "one tablespoon of mayonnaise" were all linked to "mayonnaise". We had around 1,500 canonical ingredients in a taxonomy that we had scraped from the Cook's Thesaurus[2], which is a gastronomy web site. We provide more details of how we built this taxonomy in [4].

We linked each recipe ingredient to a canonical ingredient as follows. We tokenized recipe ingredients and singularized the tokens. We sought matches and close matches (using Levenshtein distance) between an ingredient's tokens and the foodstuff names in the taxonomy. In the event of multiple good matches, we used heuristics to select one foodstuff from the taxonomy. In essence, the heuristics selected the most specific ingredient for which we had good evidence. We automatically canonicalised just over 85% of the ingredients in this way. The remainder (mostly brand names and gross misspellings), we canonicalised manually where we could. There were a few ingredients that we could make no sense of at all, and we simply removed these from the sandwiches in which they appeared.

[1] `allrecipes.com`, `www.allthecooks.com`, `tastykitchen.com`, `www.food2fork.com`, `www.yummly.com`

[2] `www.foodsubs.com`

3 Making Sandwiches

As we have explained, Earl takes inspiration from Baccigalupo and Plaza's work on case-based music playlist generation [2].

3.1 Generating candidate playlists and sandwiches

Here we present Baccigalupo and Plaza's algorithm. However, we generalise it so that it can apply also to sandwich creation. Their algorithm uses a case base that contains human-authored music playlists, and it incrementally builds a new playlist by adding songs. Earl uses a case base that contains human-authored sandwich recipes, and the algorithm incrementally builds a new sandwich by adding ingredients. Both systems work as follows:

1. Take in a seed item s (a song or an ingredient) and a desired (playlist or recipe) length from the user.
2. Create an initial partial solution p which contains just s.
3. From the case base, retrieve cases (playlists or recipes) that contain the seed item.
4. Repeat the following until p is of the desired length:
   (a) For every item s' in the cases that were retrieved, create a candidate extension of p by adding s' into p.
   (b) Calculate the coherence of each candidate extension (discussed below).
   (c) Choose the candidate extension with highest coherence, and this becomes p on the next iteration.

In fact, Baccigalupo and Plaza's playlist creation algorithm is more complicated in two ways. First, they retrieve only the k highest-scoring cases (playlists) that contain the seed item (song) from the case base, whereas we retrieve all cases (recipes) that contain the seed item (ingredient). Hence, they calculate coherence at both retrieval time and when judging candidate extensions, whereas we do not use coherence scores at retrieval time. The main effect is one of efficiency: they retrieve fewer cases and thus create fewer candidate extensions.

Second, on each iteration, for each song s', Baccigalupo and Plaza create *two* candidate extensions of p: one in which s' is added to the front of p, and one in which it is added to the back of p. What this means is that they treat song order as important. We, on the other hand, treat ingredient order as unimportant: we treat sandwiches as sets, rather than as ordered lists. Hence, on each iteration, for each ingredient s' we need to create only one extension, simply by inserting s' into the set p.

In reality, ingredient order in a sandwich *is* important; for example, one might place moist ingredients away from the bread; one might place crunchy ingredients between softer ones; and so on. But our recipe case base does not give us usable ordering information. Baccigalupo and Plaza assume that each playlist in their case base is a meaningfully ordered sequence of songs; indeed, they prune from their case base any playlists that appear to violate this assumption (e.g. ones that are alphabetically ordered). We could not make a similar assumption about the recipes in our case base. In the recipes we have scraped, ingredients are listed in all sorts of ways: rarely, if ever, are the ingredients presented in the order in which one might layer them in the final sandwich. Typically, the main ingredients are first —the meats or cheeses. Salad ingredients usually follow, and then condiments. But many sandwiches ignore this ordering completely and list ingredients alphabetically or by some other criteria that is not clear at all. Since we cannot rely on our cases to have an ordering based on sandwich construction, it does not make sense to use ordering in the sandwich creation algorithm.

3.2 Coherence

On each iteration, Earl must calculate the coherence of partial sandwich p extended by candidate ingredient s', and here we define this coherence score, again adapting the ideas in [2].

In advance, we use the Apriori algorithm to mine frequent item sets from the cases in the case base [1]. We keep only those frequent item sets whose support (frequency) exceeds a threshold (in our case, 0.02). The score of p extended by s' depends on the frequent item sets that both (a) contain s', and (b) are a subset of p extended by s'. In symbols, if we let FIS be the frequent item sets, then the coherence of p extended by s' is the sum of the relevances of those frequent item sets that satisfy conditions (a) and (b):

$$\text{coherence}(p, s') = \sum_{\mathit{fis} \in \mathit{FIS} \wedge s' \in \mathit{fis} \wedge \mathit{fis} \subseteq p} \text{relevance}(\mathit{fis}, s')$$

The relevance of a frequent item set to one of its member items s' is based primarily on the support of that item set, i.e. its frequency across the cases in the case base. However, there are two factors that can incorrectly inflate the support and hence the relevance of a frequent item set: its length and the popularity of the members of the item set. Smaller item sets are more likely to appear more often, and popular items are more likely to appear. The definition of relevance must penalize smaller item sets and item sets whose support is boosted by containing popular items. The penalties are controlled by two parameters, α and β:

$$\text{relevance}(\mathit{fis}, s') = \text{support}(\mathit{fis}) \times \frac{\alpha^{\theta - |\mathit{fis}|}}{(\text{support}(\mathit{fis} \setminus s'))^{\beta}}$$

θ is the size of the largest frequent item set that we mined. Hence, if a frequent item set is small, then $\theta - |\mathit{fis}|$ will be large and, for α in $(0, 1]$, $\alpha^{\theta - |\mathit{fis}|}$ will be small, thus penalizing the small item set. In the denominator, $\text{support}(\mathit{fis} \setminus s')$ is the support of an item set that comprises the members of fis excluding s'. Dividing by this amount (raised to the β in $[0, 1]$) decreases the relevance of item sets that contain popular items.

This scoring function is a simplified version of the one used by Baccigalupo and Plaza. Their scoring function differs from ours in two ways. First, they must again handle the ordering of songs in playlists. Hence, they refer to ‘patterns’ instead of frequent items sets. But a pattern is simply an ordered version of a frequent item set.

Second, Baccigalupo and Plaza reward a playlist if it exhibits variety. The idea is that a user might dislike a playlist in which songs from the same author or the same album are repeated. This is particularly important for music playlists for at least two reasons: (a) playlists are ordered lists, not sets, and so the variety measure discourages the addition of duplicate songs to a playlist, and (b) some of the cases in their case base might deliberately lack variety (e.g. a playlist that is in fact an album track listing, or a playlist that contains all of a person’s favourite songs by a single artist). Thus Baccigalupo and Plaza score a partial playlist using the product of coherence and variety, where they define variety using music meta-data. Earl, on the other hand, does not need a definition of variety: (a) sandwich recipes are sets, not ordered lists, and sets cannot contain duplicates, so this at least avoids a sandwich with two helpings of radishes, for example; and (b) we can assume that almost all of the sandwich recipes in our case base, if they are genuine sandwich recipes (which is likely because we are only using sandwiches with high ratings), already exhibit sufficient variety to justify their high rating.

3.3 Discussion

Preliminary testing of Earl revealed that after seven iterations many candidate extensions would have the same coherence score. There are enough ingredients present in the sandwich that the same frequent item sets match all candidates. For this reason, we often run Earl with a default length of seven ingredients.

An interesting observation is that, for a wide variety of seed ingredients, Earl adds bread to the sandwich in one of the first few iterations. Although nearly all of the sandwiches in our case base assume the other ingredients will be placed between slices of bread, only about 50% of them explicitly list the bread as an ingredient. So it was not a foregone conclusion that Earl would create sandwiches that include bread, and it is pleasing that it automatically does so.

Earl has a major weakness: the sandwiches that it produces are often very similar. Starting with any common ingredient as the seed, all-too-often it generates a sandwich with bread, butter, salt, tomatoes, peppers, onion, and the seed. The ingredients that it chooses are so ubiquitous across the case base that they all score highly and appear very relevant for any common seed ingredient. This is despite the fact that the sandwiches in the case base are quite inventive, exhibiting a wide range of ingredients, including watercress, walnuts, artichoke hearts and apple among more than a thousand others.

Note that this is not a question of the variety of ingredients *within* the sandwich (which we discussed above). The problem is not that Earl is including three types of lettuce in a sandwich. The problem is one of coverage of the ingredient space, resulting in sandwiches which are too similar to each other.

We tried to overcome this problem by modifying the values of α and β. The effect was not very pronounced on the sandwiches produced. One of the typical ingredients might be replaced by a different one, but this change would be reflected across all the sandwiches produced, so the issue remained.

Another thing we tried was using local support instead of global support. Originally, support is computed over the entire case base. We changed this to calculate support only on the recipes retrieved from the case base, i.e. those with the seed ingredient in them. This was combined with some modification of α and β. Again there was little effect. Overcoming this weakness is obviously one avenue for future research. One avenue is to favour medium values for support instead of high values. This would avoid rewarding 'mainstream' combinations that occur most often in the case base, without rewarding bizarre combinations that occur only once or twice.

4 A Preliminary Experiment

We have run a simple online experiment to obtain a preliminary evaluation of Earl's sandwich creation algorithm. We asked participants to compare human-authored sandwiches (from the case base) with system-generated sandwiches.

In advance, we selected nine different seed ingredients. We used meats (beef, ham, bacon, chicken), tuna and cheese. But we also used lettuce, corn and avocado. We had previously found that Earl would generate meat-free sandwiches for these three seeds, thus ensuring inclusion of vegetarian sandwiches in the experiment. To get the human-authored sandwiches, we used the seed to query the case base and randomly chose from the results a sandwich with a star rating of 4 or higher. For the three seeds chosen to get vegetarian options, if the sandwich contained meat, we chose randomly from the vegetarian results. For the system-generated sandwiches, we ran Earl using the seed and a desired length of 7. The sandwiches were all pre-generated to reduce the chance that waiting times would cause participants to quit the experiment.

The web page displayed two sandwiches for the same seed ingredient: one human-authored sandwich and one system-generated. We asked participants to rate both sandwiches on a scale of 1 to 10, with 1 being low and 10 being high. What we wanted the participants to tell us was how well the ingredients worked as a sandwich, irrespective of their personal tastes. Pilot testing revealed that many people had difficulty being objective in this way. If someone did not like olives and the sandwich contained olives, participants awarded the sandwich a low rating, regardless of how well the ingredients went together. Therefore, in the experiment itself, we tried to encourage participants to be objective by asking for two ratings for each sandwich: a personal rating (where they could tell us about their preferences) and an impersonal rating (where we hoped they would be more objective). We hoped that, by asking both questions, we would make participants more aware of the difference between them.

A total of 56 people completed the experiment. Hence, each of the 9 pairs of sandwiches was shown to at least 6 participants. Figure 1 shows the average rating each sandwich received by method (Human-authored or System-generated) and rating type (Personal or Impersonal). The Figure shows that

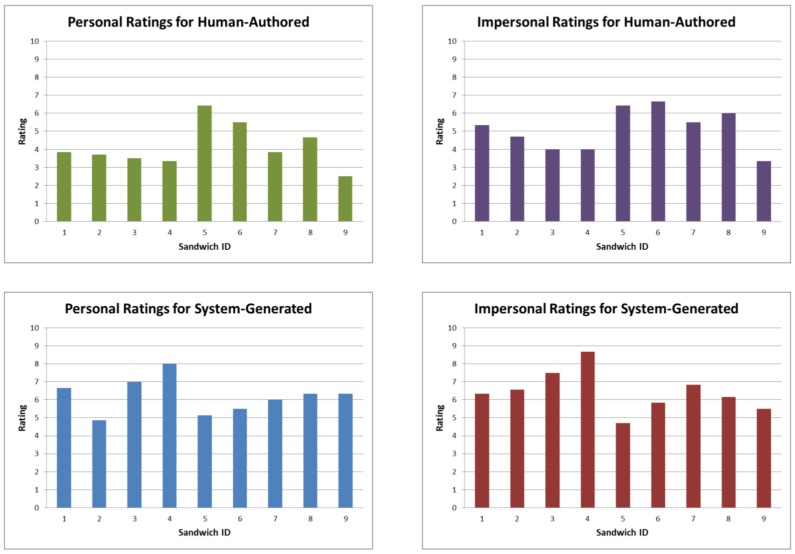

Fig. 1: Average ratings by sandwich

certain sandwiches were better received than others. Human-authored sandwich 5 and system-generated sandwich 4 in particular are the two that participants seemed to prefer most. There are some sandwiches where the personal rating is higher than the impersonal, and vice-versa. System-generated sandwiches seem to score higher overall but on one of the weaker system-generated sandwiches, sandwich 5, the human-authored method produces one of the higher scoring human sandwiches across both personal and impersonal ratings. This is the only sandwich where the human method has a higher personal rating and there is only one other sandwich where the human method has a higher impersonal rating. Across the rest, the system-generated sandwiches score higher, some by a considerable margin.

The average personal ratings were 4.4 and 6.2 for human-authored and system-generated sandwiches, respectively. The average impersonal ratings are better, 5.3 and 6.6. On average, participants found the system-generated sandwiches to be more to their tastes (statistically significant at the 1% level, using a paired, two-tailed t-test), and they thought these sandwiches were better designed (also statistically significant at the 1% level).

5 Conclusions

Our evaluation surprised us! On average, system-generated sandwiches scored higher than human-authored ones. This conclusion comes with the caveat that the experiment is of very small scale. Furthermore, the majority of participants are 18–23 year old students. Earl does tend to suggest sandwiches with very familiar ingredients, while the sandwiches in the case base are often more adventurous. It is possible that many of the participants were conservative in nature. Participants with different demographics might have different tastes or have different experiences to draw on, making for very different results.

In an ideal world, we would have a deli that would produce all the recommended sandwiches for the participants to taste. Without actually tasting the sandwiches, it is hard to judge them. An unusual combination of ingredients might not sound very appealing but might actually taste good.

There are many avenues for future work. We have already noted that Earl creates sandwiches that lack diversity; remedying this is a priority. We noted too that we treat sandwiches as unordered sets of ingredients, but this is not how a sandwich works in reality. There are rules that are generally followed, such as not putting the tomato beside the bread so the bread does not get soggy. It would be interesting to use ordering knowledge, either mined from sandwich recipes that contain this information or using heuristics that work on knowledge about the texture of the ingredients.

At present, Earl does not offer any way to customise what kind of sandwich gets created. If a user wants a vegetarian sandwich, simply starting with a non-meat ingredient as the seed does not guarantee that she will get something that is vegetarian. Offering more control is a desirable improvement.

More challenging is to consider whether the ideas in Earl can be extended to recipes for something more complicated than sandwiches. We would need to lift assumptions that we have made. For example, in using canonical foodstuffs, Earl ignores the measurements of the original recipe ingredients. We took the view that, once it is known that there is 'cream cheese' in a sandwich recipe, a suitable amount can be put in according to personal taste. This does not work with other recipes; for example, it might be crucial to use two teaspoons of baking powder, neither more nor less. Furthermore, our coherence score is primarily determined by recurrence of sets of ingredients across the case base. But to create cooking recipes, a system will often need to know something about the chemistry of the ingredients, i.e. the way they react together.

References

1. Rakesh Agrawal and Ramakrishman Srikant. Fast algorithms for mining association rules. In *Procs. of the 20th International Conference on Very Large Data Bases*, pages 487–499, 1994.
2. Claudio Baccigalupo and Enric Plaza. Case-based sequential ordering of songs for playlist recommendation. In *Proceedings of the 8th European Conference on Case-Based Reasoning*, pages 286–300. Springer, 2006.
3. Kristian J. Hammond. Explaining and repairing plans that fail. *Artificial Intelligence*, 45(1–2):173–228, 1990.
4. Henry Larkin and Derek Bridge. Subs and sandwiches: Replacing one ingredient by another. In *Workshop Programme of the 22nd International Conference on Case-Based Reasoning*, (this volume) 2014.
5. Enric Plaza and Josep Lluís Arcos. Constructive adaptation. In *Proceedings of the 6th European Conference on Case-Based Reasoning*, pages 306–320. Springer, 2002.