# Solving the Cocktail Challenge with a Case Based Reasoning System

Carlos Berg, Christian Reiser

## Introduction

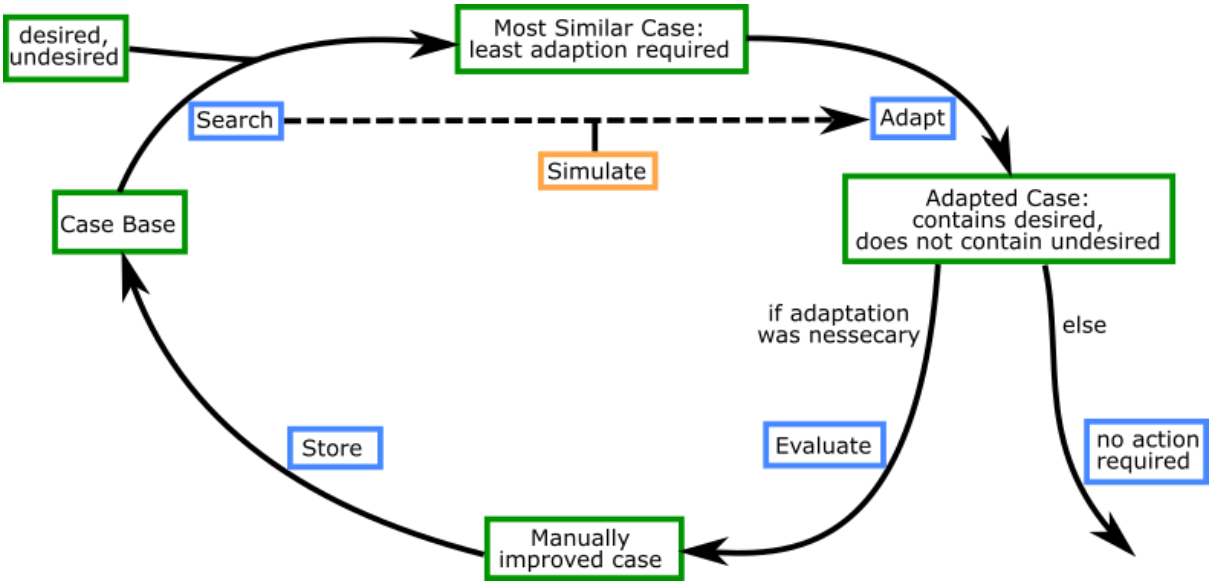
We developed a Case Based Reasoning (CBR) system fine tuned for the Cocktail Challenge (CC) from the Computer Cooking Contest. We put our focus on implementing an adaption function that suits the given application domain. The small number of cases in the challenge allows the use of a flat structure for the Case Library. A more general idea that can be derived from our project is how we compute the similarity measure: The adaption function is simulated for each case in the case library. We log the number of adaptations that had to be made for each candidate case to match the user's query. The case that requires the smallest number of adaptations is chosen. Our case structure follows the vanilla approach given on lecture slide 13.

## Requirements

The requirements of the system are defined by the CC. In the CC the user provides a set of desired ingredients and a set of undesired ingredients (the query). The objective is to return a cocktail that contains all desired ingredients and no undesired ones. The returned cocktail should have a good taste. It should be noted that no quantities are provided by the user. This allows the system to choose arbitrary quantities to achieve the objective of good taste. However, the approach can be easily abstracted. Whenever the applications domain allows a case to be meaningfully described by a set of identifiers and each identifier (here: ingredient) can be annotated by a numerical value (here: quantity) our proposed solution might be used, given that the user does not want to directly specify the numerical values. Further it makes sense to judge generated solutions (here: cocktails) by an expert. Therefore the system should include an option to evaluate generated solutions and in case of a bad solution it should offer a possibility for manual improvement.
Our CBR project only requires little computational power and memory capacity. The system responds instantaneously on average consumer hardware for a small case bases like the CC, even when adding a magnitude (in comparison to the official cases provided by the challenge description) of cases to the case base.

## Functional Architecture

The blue boxes are the components of the CBR engine while the green boxes are the inputs or outputs of those components. The architecture mostly follows the scheme of Aamodt & Plaza, but uses a simulation of the “Adapt” component for the similarity search.




## Implementation details

A query is processed by boing a full search in the flat case base. Hierarchical case bases offer faster query times while potentially degrading the quality of the returned results. Since we noticed that for the case base of the CC the performance with a simple flat structure is sufficient (instantaneous response on consumer hardware) we decided to keep using the flat structure. There exists no function of its own that computes the similarity between the query and a given case. Rather we execute the adaption routine for every case in the case base. The adaption routine makes sure that the case contains every desired identifier and no undesired ones. The resulting adapted case is discarded, but we keep track of the number of necessary modification steps. We then merely define the case that needed the lowest number of modification steps as the most similar one to the query. The number of modification steps define a distance function with the desired property that a case that contains all desired identifiers and no undesired ones will have zero distance to the query.

Before explaining the details of the CC specific adaptation routine, we want to remark that we annotated the case base with ingredient categories. As can be seen in `Data/categories.xml` we divided the ingredients into the four categories nonalcoholic, alcoholic, special and fruit. For that matter we automatically extracted all ingredients from the case base and then manually added category information.

After having obtained a good candidate case we perform once again the actual adaption routine on it. Note that this will just yield the same result as the one obtained during the first step. Yet we decided on not caching the candidates to reduce memory usage. The adaptation of one case takes an inconsiderable amount of processing time. From here on we will use the following terminology for ingredients:

| | |
|---|---|
| missing desired ingredient | Ingredients that are desired, but are not in the candidate cocktail |
| contained undesired ingredient | Ingredients that are in the candidate cocktail, but are undesired |
| optional ingredients | Ingredients that are in the cocktail, but a neither desired nor undesired |

The goal of the adaptation is that the both the sets of missing desired ingredients and contained undesired ingredients are empty. We try to make as little change as possible to the original cocktail, since we assume that the original cocktails taste well, as they were created by human experts. For that we developed four different strategies. We explain the strategies in the order of their favorableness.
The most favorable strategy is to replace a contained undesired ingredient by a missing desired ingredient. With only one replacement we reduce the size of both sets by one. However, we only perform the replacement when the missing desired ingredient and the contained undesired ingredient are in the same ingredient category (as specified by our manual annotation). This is motivated by the assumption that the general composition of a cocktail should be altered as little as possible.
After this strategy was applied as often as possible and the missing desired ingredient and contained undesired ingredient sets are still not empty we employ the next strategy. Try to replace optional ingredients by missing desired ingredients given that their categories match. To get rid of the remaining contained undesired ingredients, they are replaced by random ingredients from the same category. The order of the second and third strategy does not matter since the operations are commutable.
If the first and second strategy could not add all missing desired ingredients, since there were no contained undesired ingredients or optional ingredients of the same category in the candidate cocktail we added a fall back strategy of adding a small amount of the remaining ingredients to the cocktail. Since this operation changes the composition of the cocktail it is the most unfavorable one.
We penalize the last strategy by counting it as two modification steps. Therein also lies one of the advantages of our “search-by-simulate”-approach. We are able to manually specify the kind of cases that we deem fitting in a dynamic way. Note that this defines a rich and complex distance function that also incorporates the information defined in our categories. For other application domains the distance function defined by that approach might be even more complex, considering that we are using a fairly simple one for the CC.

If any adaptations had to be made to the candidate cocktail, i. e. we started with a non empty missing desired ingredients set or a non empty contained undesired ingredients set, we give the user the possibility to judge the adapted cocktail. An expert user that is not pleased with the recommendation by the CBR system has the possibility to alter the recipe manually. As a last step the adapted cocktail is added to the case base.

## Evaluation

hard because taste is subjective
good examples
negative examples