

Generative Mixology: An Engine for Creating Cocktails

Johnathan Pagnutti and Jim Whitehead

Computer Science Department
University of California, Santa Cruz
{jpagnutt, ejw}@ucsc.edu

Abstract

This paper details an expert cocktail generation system. After using expert knowledge to break down cocktails into eight categories, the system generates cocktails from a particular category using a context-free stochastic grammar. These cocktails were then evaluated by human participants in a research setting. Participants evaluated the cocktails on the basis of quality, novelty and typicality to check the creative potential of the generator's output.

Introduction

Some domains, such as music and visual art, have been studied in depth by the computational creativity and procedural content generation (PCG) communities. Yet other domains, such as preparing food, have not. Part of this is that food preparation is a complex task, not only in dealing with which particular combinations of ingredients should be used, but also how those ingredients should be prepared and transformed into a finished product. Even simple domains, such as chocolate chip cookies, can have many ingredient and preparation step permutations (Kenji López-Alt 2013). However, there is a strong interest in artificial chefs, servers and bartenders, as evidenced by the steady rise of restaurants featuring robotic servers and bartenders (Sloan 2014; Kross et al. 1976) as well as home meal serving and bartending robots (Glass 2014; Monsieur, LLC 2015). The next step for this niche mechanization of the food and beverage industry is to implement an AI system that can create new dishes or drinks to prepare for patrons.

A factor analysis on the Creative Achievement Questionnaire (CAQ), a creativity assessment test, revealed three categories of creative achievement: Expressive (Visual Arts, Writing, Humor), Performance (Dance, Drama, Music), and Scientific (Invention, Scientific, Culinary). This result shows that culinary creativity falls into a similar domain as scientific and innovative creativity (Carson, Peterson, and Higgins 2005). This implies that techniques used in creative recipe generators have applications in problem solving and research direction. Therefore, the development of creative recipe generation and other culinary arts may have applications for more general-purpose problem solving AI.

We can break recipes into two parts: the static ingredient list and the dynamic preparation instructions. The in-

redient list is composed of the ingredients that the recipe will use; the preparation instructions are how those ingredients are transformed into a final dish. However, there are a very large number of potential ingredients that could go into any dish, and even more ways those ingredients can be combined to become a final product. Therefore, work with smaller, less complex domains is needed to gain insight into the problem of an artificial chef. One such useful domain is mixed drinks, as the potential ingredient space for cocktails is smaller than that of culinary dishes and the mixing instructions are far simpler, while still retaining a lot of the interesting complexity. As such, we developed an expert system for cocktail generation and evaluated the artifacts it generated to start understanding the nature of computational cooking.

Related Work

PIERRE (Morris et al. 2012) uses a genetic algorithm to generate crock-pot recipes from a corpus gathered from various websites. The fitness function is based around novelty, trying to maximize the number of rare n-grams in a recipe. Recipes have also gotten attention from case-based AI planners, such as CHEF (Hammond 1986). Both these generators have a high chance to output a 'bad' recipe. PIERRE makes no claims about the quality of its output, and CHEF needs to learn from bad examples in order to create good ones. A similar branch of research to this is JULIA (Hinrichs 1992), which uses case-based design techniques such as case adaptation to determine how to best design and present a meal. Our work does not need to learn from 'bad' examples and attempts to always produce a believable drink.

Pinel and Varshney have worked on a recipe generator (Pinel and Varshney 2014), which unlike PIERRE or CHEF does not deal with a particular style or type of cooking. Using a cognitive model of creativity and a large knowledge base built from scraping recipe wikis, they created a mixed initiative generator that produces ingredient lists and rough steps to completing a recipe. This work is part of a larger system by Pinel, Varshney and Bhattacharjya (Pinel, Varshney, and Bhattacharjya 2015) that generates recipes by mining data from the Wikia recipe repository and Wikipedia to build an extensive knowledge base of recipes. From there, the system uses a mixed initiative approach, in which a new recipe is generated with user-selected categories. Varshney

et al. (Varshney et al. 2013) discuss many of the difficulties in working with recipe generation, emphasizing that how something tastes is actually the result of all five classical senses working together, plus several psychological, neurological and social phenomena.

Cocktail generation has been done, although not on any formal level. The Mixilator (Haigh 2004) is an online cocktail generator based on the writing of mixologist David Embury. The Mixilator picks a random ingredient from each of the three categories defined by Embury, and makes a predefined cocktail from it, with mixing instructions hardcoded. Although it uses an impressive amount of ingredients, the generator is highly constrained—Embury believed all drinks should contain at least three ingredients, so the Mixilator can never create a gin and tonic, for example. The Mixilator also has no knowledge about how combinations of ingredients function. It assumes that, as long as it picks from each correct category, the resultant cocktail will be good. Yet, as the authors point out, no considerations for quality went into its development. While investigating the Mixilator in writing this paper, ingredient combinations like lime sherbet and maple syrup were suggested for cocktails. Another drink called for “2 drops of liqueur” without ever specifying which flavor of liqueur. This makes the Mixilator’s output appear more whimsical than structured cocktail generation. While related investigations like the Mixilator are hobbyist projects, and some large scale recipe generators give a passing glance to the cocktail domain, we aim to be the first to take a domain sensitive, computational creativity approach to cocktail generation, and maintain a critical eye towards drink quality and expressive potential.

Mixologists have been inventing new drinks in popular literature. One of the first books on mixology as an art, *The Fine Art of Mixing Drinks* (Embury 1953), by David Embury, details a basic ratio to follow for cocktails, as well as several ingredient categories to use. More recently, *DIY Cocktails* (Simmons 2011), details several basic ratios for a wide variety of drinks. However, mixology books commonly focus on presentation or are just a compiled list of cocktail recipes (such as (Regan 2003; Joseph 2012)). Sadly, mixologist blogs ((Bovis 2015; English 2015; Jamieson 2015), for example) also tend to focus on recipe compilation or product review rather than cocktail theory.

Computational creativity is a vibrant field, with a plethora of definitions, theories and evaluation methods for creativity in computer programs. Much modern work stems from three techniques (Boden 1998) and their formalisms (Wiggins 2006) for establishing creativity in AI: by producing novel combinations of familiar ideas, by exploring potential conceptual spaces or by making transformations that allow the generation of previously impossible ideas. These relate to creativity in the process of artifact generation. The other side of the coin refers to creativity as a quality in generated artifacts (e.g. the difference between “this painting was made by a creative person” vs. “this painting is creative”). In these terms, metrics for evaluating the creativity of generated artifacts have been proposed (such as (Pease, Winterstein, and Colton 2001)), and we evaluate our cocktails on the categories of quality, novelty and typicality as defined in

(Ritchie 2007).

Expert System Constraints

The cocktail generation system defined here is derived from the rules and opinions of two primary texts: *The Fine Art of Mixing Drinks* (Embury 1953) and *DIY Cocktails* (Simmons 2011). Both texts treat cocktail creation as a process, and outline several basic rules to follow in the terms of ratios and ingredient categories. In addition, they provide mixing instructions for various categories.

Multiple source texts were used to try to minimize the amount of author bias in the system. *The Fine Art of Mixing Drinks* is an older text. Several common modern cocktails are impossible to create by following its rules alone, and today there are far more popular cocktail ingredients than there were in Embury’s time. By augmenting Embury’s rules with a more modern text, the generator can be more expressive and better reflect modern cocktail design aesthetics.

In addition, *DIY Cocktails* (Simmons 2011) gives a theoretical basis for which ingredients work well together. This helps the cocktail generator avoid various pitfalls in ingredient choice (such as combining a citric acid and a cream, which will curdle the cream), and also be smarter in selecting which ingredients to use to create a cocktail.

Finally, there were some constraints set at the discretion of the authors. Shooters and shots are not considered cocktails, and are ignored. In addition, the generator does not use overproof spirits (those that contain more alcohol than proof spirit), as they can be difficult to acquire.

Cocktail Properties

We divide a recipe into two parts: the mixing instructions (dynamic instruction) and the ingredient list (static elements). Cocktail mixing instructions are either derived from the ingredients used in the cocktail or previously decided steps in the mixing process. Lighter ingredients (juices and spirits) only require stirring; heavier ingredients (syrups and purees) may require shaking or rolling in a cocktail shaker. There are a few generally uncommon preparation instructions that are more common to cocktails, such as muddling (mashing the ingredient in the bottom of the glass). As it makes no sense to muddle an ingredient in a shaker for mixing and/or rolling (the straining head of the shaker would keep the muddled ingredients in the shaker and not in the glass), step order occasionally matters. However, someone could shake various juices and pour them into an ingredient they had muddled, so keeping track of what process is being applied to which ingredient is important. There are several other ways to mix a drink that deal with spectacle: floating a high proof liquor on the top of a drink before setting the liquor on fire, or floating several ingredients on top of each other to provide a layering effect. These techniques do not have a strong bearing on flavor, so they are not considered by the cocktail generator.

The ingredient list is more complicated. Depending on the source, the raw ingredients of a cocktail can either have many very fine qualities (such as undertone, notes or hints) or be very basic (sweet, sour). This makes it difficult to

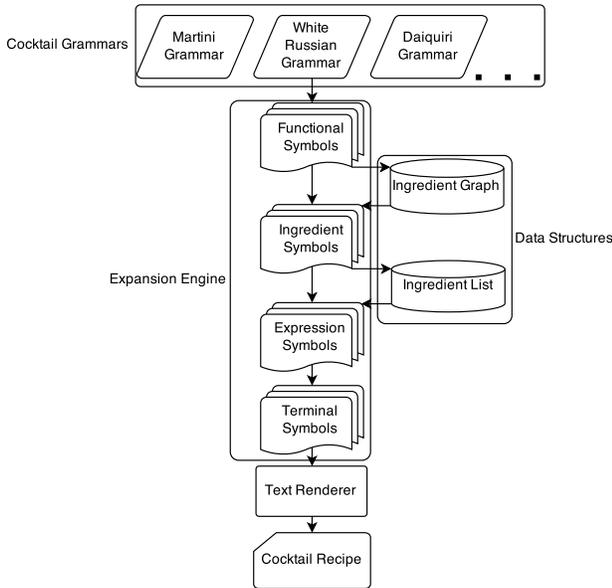


Figure 1: System Architecture. A grammar is chosen from a list of various cocktail grammars and then expanded as a set of symbols, from functional to terminal. For some expansions, symbols are built on the fly by requesting information from an external data structure. Once the grammar has expanded to terminal symbols, it is rendered as a human readable recipe and presented to a user.

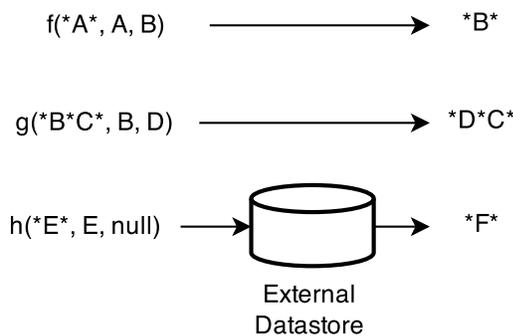


Figure 2: Three general categories of grammar expansion.

ascertain a good top-down or bottom-up model of how a particular ingredient tastes—do notes of elderberry work well with sour? Should undertones be sweet or smoky or smoky-sweet, and does any of that work well with strawberries? However, this also is not how common sense reasoning about taste functions. When someone hears the ingredients in a drink (or dish), they recall what each ingredient tasted like in the past, and make some guesses as to what they might taste like together. The cocktail generation system was built on this basic reasoning concept. Rather than try to accurately model how each ingredient tastes, the generator keeps track, on an abstract level, which ingredients work well together, and then creates drinks with combina-

tions of good ingredients. These ingredient pairings were built based on expert knowledge, rather than a database or chemical hypothesis, as in (Ahn et al. 2011).

In addition, the cocktail generation system breaks cocktails into eight categories. Each of these categories is based around a particular set of exemplars in cocktail literature. These exemplars either all share an ingredient category (such as the use of cream for drinks derived from a White Russian) or a particular ratio (2 to 1 ratios for drinks derived from a Gin and Tonic). It is important to note that all of the International Bartenders Association official cocktails roughly fall into these eight categories. Although this categorical system does not perfectly cover every potential drink, it encapsulates most of the space of potential cocktails.

Generator Architecture

The cocktail generation system has four main components: a set of stochastic, context-sensitive cocktail grammars, an engine to expand the grammars, a set of outside data structures used to build grammar symbols and a text rendering system to present generated recipes, as seen in Figure 1. All four of these components work in a serial fashion to generate a new cocktail; no part of the system runs in parallel. One of the main difficulties when designing the cocktail generation system was the amount of symbols in the grammar. There are at least 260 symbols, so writing rules out directly would have taken a large amount of human authoring time.

Cocktail Grammars

Following the research on mixology, most cocktails can be broken down into eight categories of drinks. The categories are all based on exemplar drinks; the Old Fashioned category uses the same ratios present in an Old Fashioned, for example. Sometimes a drink is considered an exemplar because it has a unique and useful ratio (the margarita's 3 parts strong : 2 parts sweet : 1 part sour), or a particularly important ingredient (the cream in a White Russian). Usually, a category also has a trend: Old Fashioned based drinks always have muddled ingredients or syrups, while Margarita-like drinks always use a liqueur as one of their sweetening agents. As such, to capture these trends, a unique grammar needs to be built for each drink category. The categories are Old Fashioned, Martini, White Russian, Margarita, Daiquiri, Mai Tai, Gin and Tonic, and Mojito.

All the grammars use the same set of symbols, but each category has its own unique production rules and constraints. Several rules were reusable (a single context free replacement rule, for example), however, each grammar has several custom, unique rules.

There are three basic ways that a grammar expands, as seen in Figure 2. The first two examples shown here are deterministic, although they both have stochastic variants where a random choice is made from a list of potential symbols. The last example is always stochastic. First, grammar symbols can expand without considering what other symbols are currently in the grammar, commonly referred to as context-free expansion. This expansion is shown at the top

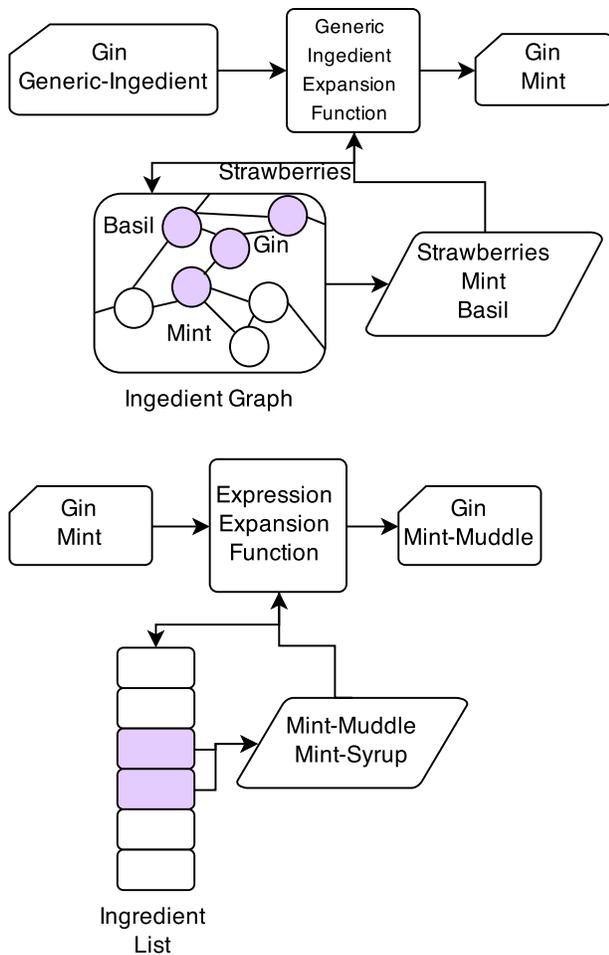


Figure 3: Overview of database expansion.

of the figure, where an expansion function, $f()$, takes an input grammar, the symbol to be expanded ('A') and the symbol to expand to ('B') and returns an output grammar where 'A' has been replaced with 'B'. Second, grammar symbols sometimes do care about context, and look at the other symbols in the current string before expanding. If certain symbols are present, then that symbol expands differently. This is referred to as context-sensitive expansion. This expansion is shown in the middle of Figure 2. The expansion function, $g()$, takes an input grammar, the symbol to be expanded ('B') and the symbol to expand to ('D'). $g()$ scans the input string, and since the grammar contains both B and C, transforms B to D. Unlike other context sensitive grammars, the C's location in the string is unimportant—if the string contains a C, B will transform to D. Finally, some rules, instead of going from one symbol to the next, instead request an external structure to supply the next symbol. These rules can be context free or context sensitive. This expansion is shown at the bottom of Figure 2. The expansion function, $h()$, takes an input grammar, the symbol to be expanded ('E') but does not have a symbol to expand to. Instead, $h()$ makes a request

of an external data store as to what 'E' should expand into. The data store returns 'F', and the function replaces 'E' with 'F' and returns the grammar.

External data structures make no promises about being able to fulfill a request. When a database cannot fulfill a request, grammar expansion is restarted from the axiom. External database calls are outlined in Figure 3. The top graph in Figure 3 shows expansion using the ingredient graph. When queried, the expansion function either passes a symbol that has a node in the graph (in this case, gin) or polls the graph randomly. The node's neighbors are returned, and the function chooses one to use for expansion. The bottom graph shows expansion using the ingredient list. When used, the list is supplied a symbol that needs to be expressed (in this case, mint). The list returns all the possible expressions of the symbol, and then the expansion function chooses which one to use.

Symbols also have a flag that is set to 'false' for context-free symbols and 'true' for context-sensitive symbols. This flag is used to allow for context-free symbols to be expanded before context sensitive symbols, so that symbols that need context will have as much information as possible before expanding.

To help keep track of how a grammar is expanding, symbols are actually a (symbol, type) tuple. The symbol is what gets replaced or used for replacement, while the type helps various context sensitive rules determine the right time in the expansion to execute. Types work like walls, all symbols need to be of a particular type before the next set of rules can apply. The types used are functional, ingredient, expression, and terminal. Functional symbols are qualifiers like "strong", "sweet" or "sour". They describe the function of a particular ingredient, according to a ratio. So, a margarita can be described in rough terms as 3 parts strong : 2 parts sweet : 1 part sour. Ingredient level symbols fill in the functional symbols with high level ingredient qualifiers, so, "lemons" could fill in for "sour". The next type of symbols, expression symbols, tell us how each ingredient is going to be expressed in a cocktail. So, "lemons" could become "lemons-juice" or "lemons-muddled". Most expression symbols are also terminal symbols, however, occasionally the grammar needs to add a few more details to a symbol before it can get rendered to text.

Symbols keep track of what they replaced, which allows us to trace a symbol's lineage. This commonly happens when we divide up the ingredients into parts. If a drink calls for 2 parts sour, and both lime and lemon juice are being used, then the cocktail generation system checks that the juices both come from the same original sour symbol. It then correctly divides the parts equally among the juices.

It is also possible for a rule to rewrite a symbol's lineage, as in Figure 4. Lineage rewrites perform abstraction and recategorization within a set of rules. This increases the expressive potential of a particular category, so that it can still accurately represent the drinks that fall into that category without resorting to having a collection of starting axioms. This also allows for axiomatic change based on how a current grammar is expanding. If it suddenly makes more sense for a particular expansion of symbols to have been derived

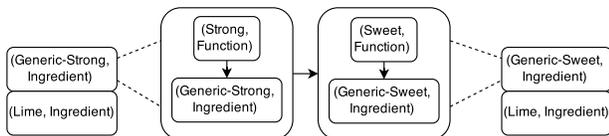


Figure 4: Overview of a lineage rewrite rule. The rule transforms a strong symbol into a sweet symbol. Normally, new symbols are appended as children to the symbol they expanded from, but for lineage rewrites, we replace the symbol and the symbols it descended from.

from a similar axiom, the axiom can shift to reflect that.

Each grammar expands until it hits a set of terminal symbols. Then, the set of symbols is passed off to the text renderer to generate a human readable recipe. Axioms for a cocktail grammar start with a set of functional symbols, and the amount of each symbol that should be in the final drink expressed in parts. An example grammar (the Old Fashioned grammar) is presented in Figure 5.

Ingredient Representation

There are two data structures that the expansion engine can query for information to build new symbols. How both of these structures are used is outlined in Figure 3. The first structure is used primarily for expanding ingredient symbols; the second structure is used for expanding expression symbols. The first data structure is the ingredient graph, a bidirectional graph where each node corresponds to an ingredient such as chocolate or strawberries. Adjacent ingredients on the graph work well together in a drink, according to experts. So, if we already know we are going to use one ingredient, we can get that ingredient's neighbors in order to see what other ingredients should be used with it. There are some nodes that are connected to every other node, but it is, in general, a sparse graph.

The other main data structure is a list of ingredient expressions, organized by ingredient. This lets us look up how lemons can expand, and pick an expression that fits a rule. The list makes no promises about having the right entry for a rule. If a rule is looking for a puree and is trying to expand lemons, the query on the list will return nothing (as lemon puree was not considered a valid ingredient by experts).

Expansion Engine

The expansion engine takes a cocktail grammar axiom and expands it in turn, from functional symbols to ingredient symbols, then to expression symbols, then to terminal symbols, as seen in Figure 1. The engine also makes requests of the outside data structures to get information needed to create a symbol when required. The engine has a hard rule: expand context free symbols before context sensitive symbols. In addition, when two or more context sensitive symbols can expand, and no context free symbols can expand, a random one is chosen to expand first.

The best way to go over the expansion engine is to go through a sample run. A user has asked for the system to generate something based off of a White Russian. The grammar starts with three symbols: (strong, function), (sweet,

function) and (mild, function). Now, the system looks through the current rules it has and there are two that can potentially apply: perform a lineage rewrite to transform the strong symbol into a sweet symbol, or expand the strong symbol into an ingredient level base spirit. As these are both equally valid rules, the system picks one at random, and decides to transform the strong symbol into a sweet symbol. The grammar now reads like (sweet', function), (sweet, function), (mild, function). The grammar now has several context-free rules it can apply, expanding sweet' into a ingredient level sweet symbol, expand sweet into an ingredient level sweet symbol and expand mild into an ingredient level mild symbol. The grammar randomly picks among these three rules, as all of them are context-free. After those rounds of expansion, the grammar now looks like (generic-sweet, ingredient), (generic-sweet, ingredient), (generic-mild, ingredient). Now, there is a context-free rule for expanding the generic-mild symbol, whereas expanding generic-sweet is context-sensitive. So, generic-mild gets expanded next and the grammar now looks like (generic-sweet, ingredient), (generic-sweet, ingredient), (cream, expression). At this point, we start to expand one of the generic-sweet symbols and the grammar needs outside help, as there are many symbols that it can expand into. The ingredient graph is queried, looking for neighbors of the cream symbol. A list of neighbors is returned. This is stored, in case the grammar needs to expand another ingredient from the graph. One is picked from them: mint. The first generic sweet symbol is expanded, and now the grammar looks like (mint, expression), (generic-sweet, ingredient), (cream, expression). The grammar then checks the stored symbols from the last graph query and selects another one to expand the last generic-sweet symbol into. The grammar now looks like (mint, expression), (chocolate, expression), (cream, expression). The last round of expansion has the grammar query the ingredient list three times, once for each of these symbols to look for valid ways to express them. The end grammar string is (mint-creme, terminal), (chocolate-liqueur, terminal), (heavy cream, terminal).

Text Rendering

The last part of the system is the text renderer. After expanding out the grammar, we have ingredients and the amount of each ingredient expressed in parts. This still needs to be converted to a human readable recipe. For the most part, this means replacing the dash in the symbol with a space. Some symbols are important to a cocktail, but are not given a part amount because they are used for garnishes, taste or in such small amounts it makes no sense to display them as a part. A prime example would be bitters, used in cocktails based on the Old Fashioned. In this case, amounts given in dashes are used (or other garnishing terms, like a "twist of lemon").

The mixing instructions are appended to the ingredient list. The mixing instructions come directly from the original category of drink and the ingredients used, with some simple replacement (such as ingredient names rather than functional terms) to make the recipe easy to follow. Finally, a name (currently an adjective, noun pair) is added to the cocktail. An example final recipe is presented in Figure 6

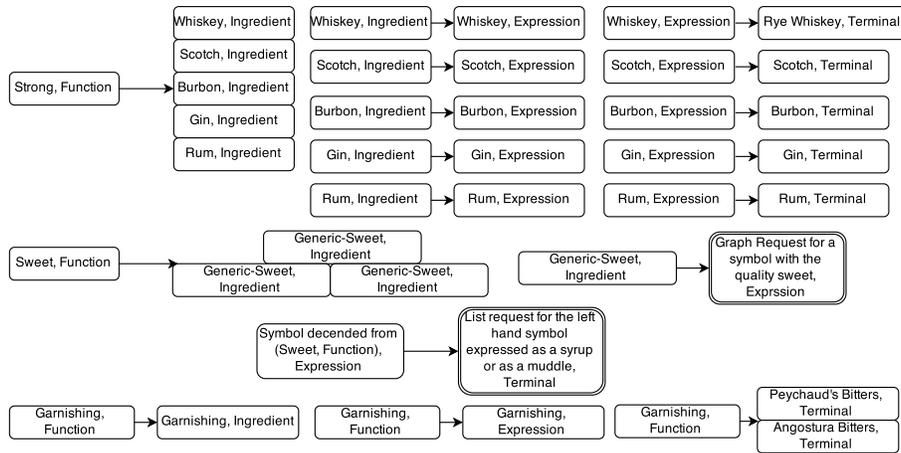


Figure 5: Diagram of the Old Fashioned grammar

brave crocodile
 4 parts tequila
 1.5 parts strawberry liqueur
 1.5 parts passion fruit puree
 1 part lemon juice
 Pour all ingredients into a cocktail shaker with ice and shake for about 15 seconds.
 Strain into an ice-filled highball glass.

Figure 6: A cocktail generated with the Mai Tai grammar.

Expressivity

In both PCG and computational creativity, the expressive range of a generator is a strong consideration for how well that generator performs. Expressive range can be thought of as the range of parameters that change the kind of content the generator can produce (Smith and Whitehead 2010). For a cocktail grammar, expressive range is tied with the connectivity of the ingredient graph (the more connections the graph has, the more symbols a particular grammar can access). In addition, we can look at how 'open' a particular grammar is to various ingredient expressions. If a grammar can use a lot of ingredient expressions, then it can generate many more combinations.

To measure this, each grammar generated 1,000 cocktails, and the amount of times a particular terminal symbol occurred was counted. As we have a list of all potential terminal symbols, if a symbol was never used, it was given a use count of zero. The result of this count is shown in Figure 7. Each cocktail grammar is not equally expressive. Some categories are more restrictive than others, and lean more heavily on particular ingredients. However, each grammar seems to focus on different parts in the potential ingredient space, and when looked at all together, the entire system does a good job of making sure that all provided ingredients get used.

Evaluation

To see if a particular generated artifact is creative, three metrics were used: quality (the measure of how well an artifact

performs a particular purpose), novelty, (the measure of how unique an artifact is to an evaluator), and typicality (the measure of how well the artifact fits in a particular class of artifacts). For cocktails, quality is how well the cocktail tastes as compared to other cocktails the taster has drank. Novelty is how different a cocktail tastes as compared to other cocktails the taster has drank. Typicality is how much like a cocktail a current cocktail tastes like. This forms an evaluation space, where differing rating triples have meaning. High ratings in quality but low ratings in novelty imply that a cocktail was good, but very similar to what the taster usually orders. High in novelty and low in quality implies an interesting cocktail, but one that does not taste very good. A low score in typicality implies that the cocktail does not taste like a cocktail at all, and tastes closer to a non-alcoholic drink or straight base spirit. In order to be considered creative, a generated artifact needs to perform highly in all three categories, as per artifact-focused definitions of creativity.

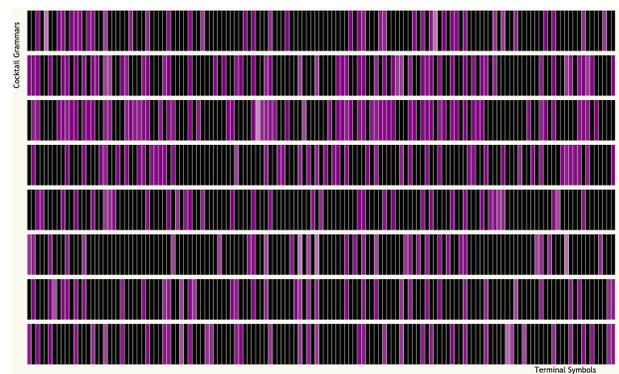


Figure 7: Ingredient use heatmap. The x-axis graphs individual ingredients, the y-axis graphs the grammars using them. As squares get lighter, they were used more times in the generated run.

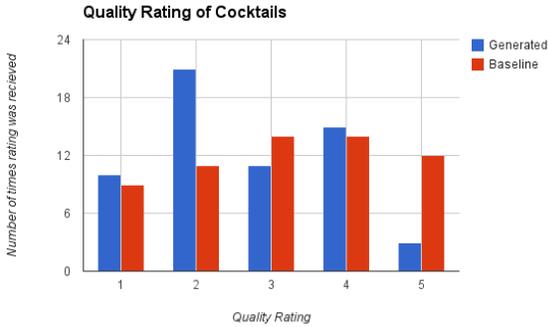


Figure 8: Quality ratings for the generated and baseline drinks. Quality in the generated drinks appears to be more polarizing than quality in the baseline drinks.

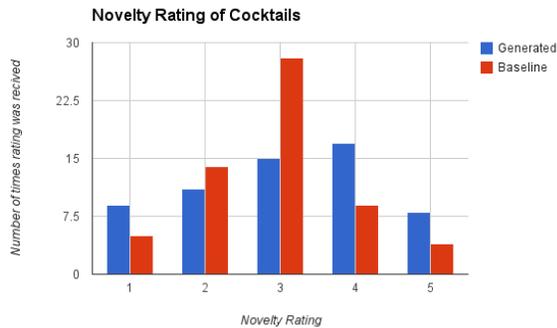


Figure 9: Novelty ratings for the generated and baseline drinks.

Table 1: P-Values

Category	Quality P-Values	Novelty P-Values
Overall	0.048	0.344
Margaritas	0.001	0.003
Martinis	0.505	0.118

Margaritas have detectable differences in both quality and novelty, martinis have no detectable differences and there is a detectable difference in quality but not novelty overall.

To this end, we had two of the eight grammars evaluated for quality, novelty and typicality by human tasters. Two cocktails from both the Margarita grammar and the Martini-based grammar were generated. In addition, two established cocktails that followed the rules for each grammar were chosen as a baseline to compare the generated cocktails against. Participants tasted each cocktail and then evaluated the cocktail based on their sip. Cocktails were presented in a random order, and participants were told that all eight cocktails were generated.

The use of a baseline to compare the generated drinks

against also helped reduce taste effects—if a particular participant did not like martinis, for example, they were expected to rate both the generated martinis and the baseline martinis low.

One third of the participants had prior experience mixing drinks. All participants had at least two cocktails over the past year, with 20% having had two to four cocktails, 33% having had five to seven cocktails and the rest having had more than seven. All participants were at least 21 years old. 60% of participants were 25-29, $\approx 27\%$ were 21-24, and the rest were 30 or older. 40% of participants were female, the rest were male. The tastings occurred in an office environment.

Participants were asked to evaluate the cocktails on quality and novelty using a five point scale, with a score of one being low and a score of five being high. To rate the cocktails for typicality, participants were asked if they believed what had been served was a cocktail, and to try to classify which exemplar the cocktail was based on. Table 1 contains the p-values from an unpaired t-test between the baseline and the generated cocktails. There was not a detectable difference in the novelty metric, overall. However, the generated drinks, overall, did perform slightly detectably worse in the quality metric. These results are captured in Figures 8 and 9.

For typicality, the generator performed well, with very few participants believing that they were not served a cocktail. However, when asked to try and identify which exemplar drink the cocktail had come from, participants did poorly. Participants only correctly identified the exemplar cocktail 26.67% of the time. This can imply two things: 1) that a general audience does not have enough skill in cocktails to taste where particular drinks came from and/or 2) the classification scheme used by the generator is not how the average person classifies cocktails.

Threats to Validity

With no detectable difference in novelty between the baseline cocktails and the generated drinks, we can not conclude anything about the generated drinks compared to the baseline. In addition, the generator and evaluation did not take into account the environment the cocktail should be consumed in. It is possible that bar ambiance could impact the perception of flavor. Garnish selection is not considered in the current generator, and garnishes can strongly impact how people perceive cocktails.

There are weaknesses in any expert system—how well did the experts describe their process, and how well was that process encapsulated in the system? The majority of the cocktail generation system came from expert knowledge, from the structure of the ingredient graph to the types and numbers of grammars used. This still leaves out certain cocktails. A Cement Mixer, for example, breaks one of the cardinal rules of the system (citric acid and cream should not be mixed) to create a novel texture.

There are several weaknesses with the open loop of generating, then evaluating with human evaluators. The generator itself cannot react to the evaluations of its own output and make adjustments to its internal drink mixing philosophy. As pointed out by Stokes(2011) as well as others, this

implies that the current generator is not creative, regardless of how highly its output is scored. In addition, the generator makes no attempt to account for any sort of taste. It blindly puts ingredients together without understanding why those ingredients might work well together. This generator will also never modify either ingredient representation to discover new cocktails.

Discussion and Future Work

Other computational ways to evaluate the system could be employed. Output recipes could be compared to existing rated recipes from online websites and databases, and a quality, novelty and typicality metrics could be derived from this comparison. However, the use of human evaluators, at least in the current state of the field, is important, because there are aspects of taste not captured in an ingredient bill or preparation steps.

Data driven cocktail generators are a strong next step. This, generally, would mean scraping various lists of cocktails (either from the web or from popular literature) and attempting to derive some heuristic for cocktails from the data. Online databases are particularly attractive, as they may have both good and bad examples to learn from.

As alluded to earlier, typicality can be a tricky part of the generator to evaluate. A way around this problem would be to have evaluators rate several well-known variations, to establish a baseline for 'cocktail recognition' that the generator's output can be compared to.

Finally, there is a need to evaluate cocktails on the merits of taste. In turn, we need a computational model of taste to see how potential drinks might taste. This lets us truly close the loop so the generator can evaluate its own work. This sort of model would allow for the use of modification or repair to a poorly evaluating dish, like several case-based reasoning techniques modify plans to best fit the current scenario. In addition, such an evaluator could evaluate many recipes, far faster than a human could.

References

- Ahn, Y.-Y.; Ahnert, S. E.; Bagrow, J. P.; and Barabási, A.-L. 2011. Flavor network and the principles of food pairing. *Scientific reports* 1.
- Boden, M. A. 1998. Creativity and artificial intelligence. *Artificial Intelligence* 103(1):347–356.
- Bovis, N. 2015. The liquid muse. <http://theliquidmuse.com/>.
- Carson, S. H.; Peterson, J. B.; and Higgins, D. M. 2005. Reliability, validity, and factor structure of the creative achievement questionnaire. *Creativity Research Journal* 17(1):37–50.
- Embury, D. A. 1953. *The Fine Art of Mixing Drinks*. Faber.
- English, C. 2015. Alcademics. <http://www.alcademics.com/>.
- Glass, C. 2014. *Cocktail Automation Management System*. Ph.D. Dissertation, Cornell University.
- Haigh, T. 2004. The mixilator. <http://www.cocktaildb.com/mixilator>.
- Hammond, K. J. 1986. Chef: A model of case-based planning. In *AAAI*, 267–271.
- Hinrichs, T. R. 1992. *Problem solving in open worlds: A case study in design*. Lawrence Erlbaum Hillsdale, NJ.
- Jamieson, J. 2015. Liquor snob. <http://www.liquorsnob.com/>.
- Joseph, P. 2012. *Boozy Brunch: The Quintessential Guide to Daytime Drinking*. Rowman & Littlefield.
- Kenji López-Alt, J. K. 2013. The food lab: The science of the best chocolate chip cookies. <http://sweets.serious-eats.com/2013/12/the-food-lab-the-best-chocolate-chip-cookies.html>.
- Kross, R. W.; Fiel, L. D.; Crockett, C. R.; Neely, G. B.; and Benton, L. J. 1976. Automatic mixed drink dispensing apparatus. US Patent 3,940,019.
- Monsieur, LLC. 2015. monsieur. <http://monsieur.co/>.
- Morris, R. G.; Burton, S. H.; Bodily, P. M.; and Ventura, D. 2012. Soup over bean of pure joy: Culinary ruminations of an artificial chef. In *Proceedings of the 3rd International Conference on Computational Creativity*, 119–125.
- Pease, A.; Winterstein, D.; and Colton, S. 2001. Evaluating machine creativity. In *Workshop on Creative Systems, 4th International Conference on Case Based Reasoning*, 129–137.
- Pinel, F., and Varshney, L. R. 2014. Computational creativity for culinary recipes. In *CHI'14 Extended Abstracts on Human Factors in Computing Systems*, 439–442. ACM.
- Pinel, F.; Varshney, L. R.; and Bhattacharjya, D. 2015. A culinary computational creativity system. In *Computational Creativity Research: Towards Creative Machines*. Springer. 327–346.
- Regan, G. 2003. *The Joy of Mixology*. Random House LLC.
- Ritchie, G. 2007. Some empirical criteria for attributing creativity to a computer program. *Minds and Machines* 17(1):67–99.
- Simmons, M. 2011. *DIY Cocktails: A simple guide to creating your own signature drinks*. Adams Media.
- Sloan, G. 2014. Robot bartenders? this new cruise ship has them. <http://www.usatoday.com/story/cruiselog/2014/11/01/quantum-robot-bar-cruise/18308319/>.
- Smith, G., and Whitehead, J. 2010. Analyzing the expressive range of a level generator. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, 4. ACM.
- Stokes, D. 2011. Minimally creative thought. *Metaphilosophy* 42(5):658–681.
- Varshney, K. R.; Varshney, L. R.; Wang, J.; and Myers, D. 2013. Flavor pairing in medieval european cuisine: A study in cooking with dirty data. *arXiv preprint arXiv:1307.7982*.
- Wiggins, G. A. 2006. A preliminary framework for description, analysis and comparison of creative systems. *Knowledge-Based Systems* 19(7):449–458.