Information Extraction and Graph Representation for the Design of Formulated Products

Sagar Sunkle[1(✉)], Krati Saxena[1], Ashwini Patil[1], Vinay Kulkarni[1], Deepak Jain[2], Rinu Chacko[2], and Beena Rai[2]

[1] Software, Systems, and Services Group, TCS Research, Pune, India
{sagar.sunkle,krati.saxena,ab.patil2,vinay.vkulkarni}@tcs.com
[2] Physical Sciences Group, TCS Research, Pune, India
{deepak.jain3,rinu.chacko,beena.rai}@tcs.com

Abstract. Formulated products like cosmetics, personal and household care, and pharmaceutical products are ubiquitous in everyday life. The multi-billion-dollar formulated products industry depends primarily on experiential knowledge for the design of new products. Vast knowledge of formulation ingredients and recipes exists in offline and online resources. Experts often use rudimentary searches over this data to find ingredients and construct recipes. This state of the art leads to considerable time to market and cost. We present an approach for formulated product design that enables extraction, storage, and non-trivial search of details required for product variant generation. Our contributions are threefold. First, we show how various information extraction techniques can be used to extract ingredients and recipe actions from textual sources. Second, we describe how to store this highly connected information as a graph database with an extensible domain model. And third, we demonstrate an aid to experts in putting together a new product based on non-trivial search. In an ongoing proof of concept, we use 410 formulations of various cosmetic creams to demonstrate these capabilities with promising results.

Keywords: Formulated products · Design · Ingredients · Recipe · Information extraction · Conceptual model · Graph database · Neighbourhood · Creams · Cosmetics

1 Introduction

The formulated products industry is an emerging global market of around 1400bn Euro focusing on an array of ubiquitous products used in daily life world over. Despite this scale, the advent of information systems in this sector is still nascent. State of the art in the design of formulated products relies heavily on experiential knowledge of the experts who consult various sources of information perfunctorily [8]. Formulations of organic formulated products contain ingredients that undergo a step-by-step procedure with actions such as heating, cooling, stirring,

S. Dustdar et al. (Eds.): CAiSE 2020, LNCS 12127, pp. 433–448, 2020.
https://doi.org/10.1007/978-3-030-49435-3_27

mixing, and so on to obtain specific target properties, both physical and chemical [5,19]. Suppose a cosmetic company decides to introduce a new face cream. A team of experts on payroll has a general idea that a face cream requires an emulsifier and an emollient. An emulsifier is an ingredient that promotes dispersion of immiscible liquids while forming the cream, and an emollient imparts skin-soothing effect in the end product. The team would proceed by finding similar formulations for a face cream, choose ingredients that are representative of the said functionalities, and put together a recipe using appropriate actions.

This seemingly straightforward process takes months because the information sources are often not well organized and scattered over offline and online media such as handbooks, articles, journals, and websites, respectively. A vast number of ingredients exists representing multiple functionalities, possessing different names depending on different nomenclatures. Similar formulations show the composition of these ingredients, as well as numerous actions performed on them. Still, the synonyms/other names of the ingredients or their functionalities, reside in different sources and must be consulted separately.

There are necessarily two requirements for solving this problem one, the gathering of information from multiple sources, and two, enabling non-trivial searches and analyses of such information for the product design activity. This problem has been recognized in recent years by the EU Formulation Network[1] and the American Chemical Society[2], both of which have come up with roadmaps and offerings featuring more organized information sources.

We propose to cater to the above-stated requirements using specialized information extraction techniques and graph representation, respectively. The research in formulated products, as well as information systems in the chemical sector, often assumes a database of all/majority of the constituents of formulations under consideration as we will discuss in the next section. In comparison, our approach provides the method and tools to create such a database to aid the expert in designing new products.

We demonstrate our approach using 410 cream formulations obtained from the Volumes 1 to 8 of the book cosmetic and toiletry formulations by Flick [6]. The formulated product design activity requires informed access to ingredients and their other names, their weight ranges, their functionalities, knowledge of how to combine them and actions performed on them with specific conditions. Considering this, our specific contributions are as follows:

1. **Extraction.** In Sect. 3, we propose to use state of the art open information extraction (Open IE) as well as dependency parsing techniques augmented with dictionaries and stacking to obtain what we refer to as action-mixture/ingredient-condition (A-M-C) structures from the recipe texts of formulations. We extract product name, ingredients, and their weights using regular expressions. For information such as other names for given ingredients and their functionalities, we extract the information from relevant websites as shown in Fig. 1.

[1] EU Formulation Network https://formulation-network.eu/.
[2] Formulus by American Chemical Society https://www.cas.org/products/formulus.

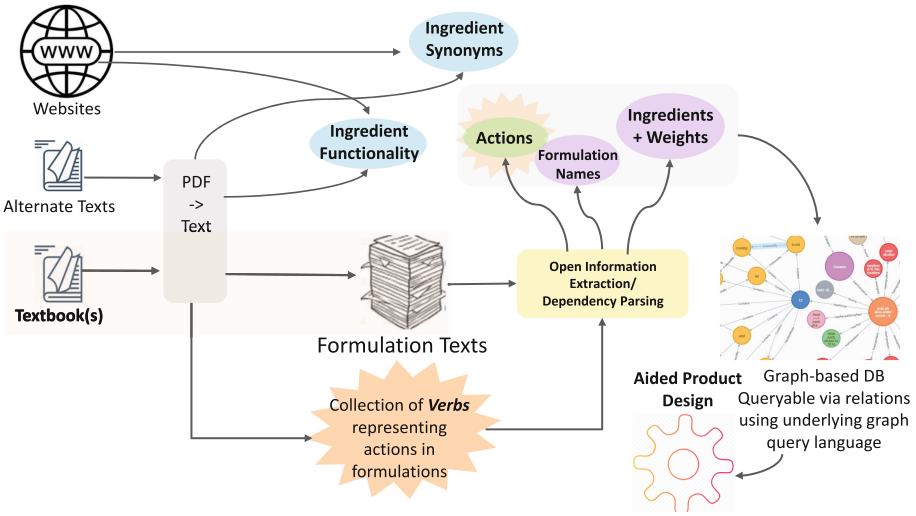


Fig. 1. Extraction, storage, and retrieval of formulations

2. **Storage.** We present a conceptual model for storing the details of each cream as a node in a graph database in Sect. 4. The extensibility offered by graph database means that not only creams as a kind of cosmetic product, but details of other formulated products can also be easily accommodated in this database. As illustrated in Fig. 1, our tool takes the formulation text as input and generates insertion queries on top of the base graph. We store ingredient synonyms and ingredient functionalities separately as simple lists.
3. **Retrieval and Design Aid.** SQL-like queries over the graph database and the A-M-C structures, along with queries on synonym and functionality lists, enable finding relatedness of ingredients, functionalities, and products, resulting in what we refer to as neighbourhoods. In Sect. 5, we show how we use these neighbourhoods step by step to aid the expert in composing variants of the intended products.

We begin by presenting the background on formulated products industry and the use of information systems therein in the next section.

2 Background and Related Work

Formulated Products Industry. We come across many formulated products in our daily lives in the form of cosmetics, personal care, detergents and other household and professional care products, foods, adhesives, fuels and fuel additives, lubricants, paints, inks, dyes, coatings, pesticides, construction materials, and medicines and pharmaceutical products. Individual ingredients used within a formulation may be incorporated to provide active functionality and enhanced

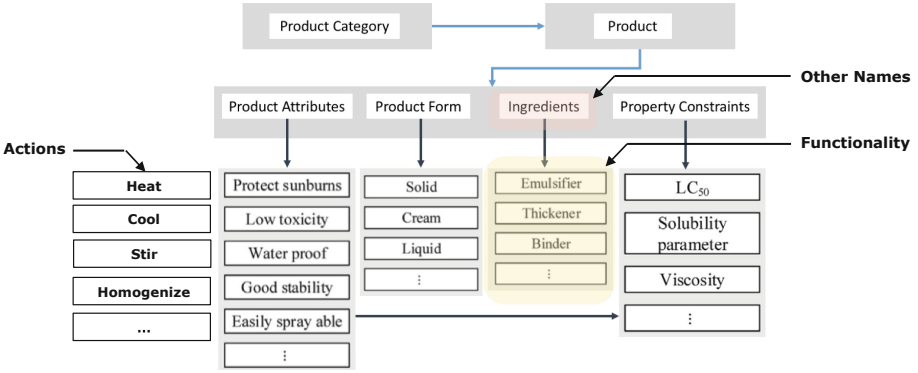


Fig. 2. Information and knowledge required for Formulation Design, adapted from [19]

delivery or as a protective or stabilizing agent. The idea of active or primary ingredients and other or secondary ingredients leads to the notions of mandatory and optional functionality in the design of formulated products as we will discuss in Sect. 5.

Design of Formulated Products. In the search for a new formulation, an expert must refer to the already existing recipes to make rational judgments when choosing the ingredients, their respective quantities and the procedure to follow to get a stable formulation that has the desired chemical functions. Several approaches have been proposed toward optimal design of formulated products [3–5,7,8,13,18,19]. These approaches suggest using knowledge from experience, models or databases to choose a product form such as cream; then select functionality of ingredients such as a solvent; generate candidates for each chosen ingredient functionality and finally combine the ingredients [19]. In most of these approaches, the assumption is that either a relevant database/ knowledge-base is available or created manually. State of the art, therefore, relies mainly on experts finding similar formulations using standard file search and compilation. Without any knowledge support tools, formulated product development becomes iterative and time-consuming without a list of acceptable ingredients and actions to be applied to them [13].

Requisite Information and Sources. Zhang et al. describe the kinds of knowledge and information that is required in formulated product design [19]. As illustrated in Fig. 2, in addition to product attributes, product form, ingredients, and property constraints; functionality of ingredients, actions associated with ingredients and their other names are also required. The product form and the product attributes describing functional requirements, indicate the main reason why a consumer may want to buy a product, e.g., a sunscreen lotion must protect skin from sunburns and skin ageing. On the other hand, the functionality of an ingredient, i.e., whether it is an emulsifier or a thickener, is the crucial factor when designing the product since other ingredients are chosen based on it. An important consideration when synthesizing ingredient-functionality list is

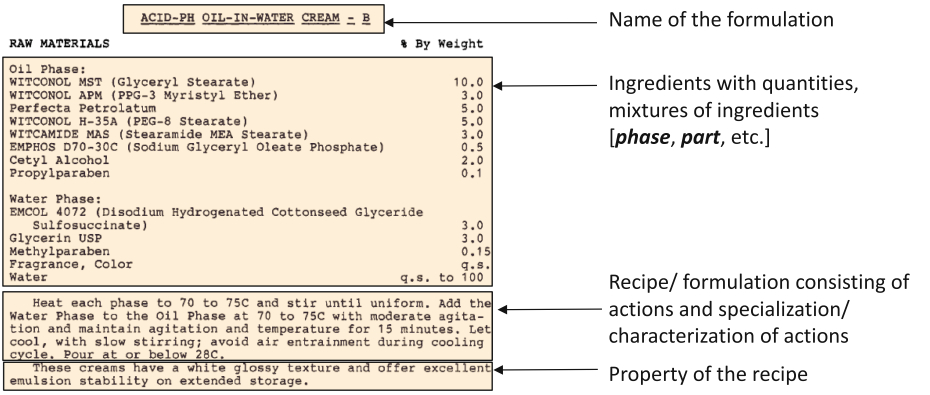


Fig. 3. Structure of a formulation from a textual source [6]

that several different names refer to a single ingredient. In such cases, an ingredients dictionary, such as [14] can be used as one source of collating different names of the same ingredient. Once experts finalize a set of ingredients, then it is possible to use techniques like mixed-integer programming to incorporate heuristics and compute possible variants [2,17].

Extraction of Requisite Details. Previous attempts at extracting formulation constituents have focused on inorganic materials [10,11,15], which are distinct from organic materials in formulated products like creams, meaning that recipe actions are not reactions between chemicals. To the best of our knowledge, ours is the first attempt at applying information extraction techniques to organic formulated products. In the next section, we elaborate our approach in extracting formulation constituents.

3 Extraction of Formulation Constituents

An organic chemical formulation text usually contains the name of the formulation, ingredients, mixtures (if any), weights or proportions of ingredients, and actions to be performed on the ingredients and mixtures, with conditions such as specific temperatures or states as shown in Fig. 3. We refer to these details as constituents of a formulation. We first cover the extraction of ingredient details followed by the processing of recipe text.

3.1 Extraction of Ingredients, Mixtures, and Ingredient Weights

To extract ingredients and ingredient weights, it helps to preserve the layout while transforming the PDF files to text format, which we achieve using the Apache PDFBox API. With a preserved layout, the ingredient and its weight occur in a single line of text. Additionally, the formulation may use the ingredients as a part of a mixture. To recognize the mixture indicators separately, we prepare a small list of mixture phrases. The list we use contains indicator phrases like phase a, phase

b, phase c, oil phase, water phase, part a, part b, part c, and part d. These indicator phrases appear in a line followed by the list of ingredients that are part of that mixture, as shown in Fig. 3. To process the ingredients as part of the mixture, we first identify if a mixture phrase is present. We associate all ingredients with the current mixture obtained from that line until we encounter the next mixture. To ensure that we only consider the part of the text that contains the ingredients for the processing of ingredients, we apply simple sentence (boundary) detection (SBD)[3]. As illustrated in Fig. 3, the ingredients occur in the part of the text that is NOT a set of sentences (whereas the recipe text is).

To recognize the weight fractions of each ingredient, we use a regular expression. The regular expression is `\d+\s*\.\s*\d+|q.s.|as\s*desired` in Python. The `\s*` flag takes care of multiple white spaces between the integer and the fraction part of an ingredient's weight represented by the flag `\d`. The + sign in front of the flag indicates more than one digits in the integer part of the weight. Words such as q.s. (indicating the amount which is needed) can be added as more of such phrases are encountered.

3.2 Extraction of Actions from Recipe Texts

The recipe text describes actions performed a) on the ingredients individually or b) ingredients as a part of a mixture, and c) on the mixtures if the mixtures are present, as shown in Fig. 3.

The critical problems faced in extracting A-M-C structures are that a) the recipes contain instructions which are imperative sentences, and b) objects may be alluded to but could be missing from the sentences [9,15]. Since the sentences are instructions, they begin with an instructional verb and therefore often lack a subject (from the typical subject-verb-object structure of a sentence). Additionally, with the flow of instructions, the previous object acted upon is often implicitly considered without explicitly mentioning it in the next instruction.

Given that we need to associate actions with ingredients, we choose techniques that return set of subject-verb-object* (SVO*) triples from a given sentence. While other similar attempts in inorganic materials synthesis procedure extraction have used dependency parsing, we also use open information extraction or open IE[4]. An open IE implementation returns a triple of subject-verb-object*. Specific implementations may return individual triples, replicating the subject and verb for each object if there are many objects. Open IE models are often trained by bootstrapping on other open IE models which could have been trained on manually extracted triples from sentences.

Our observation is that open IE, as well as dependency parsing[5], fail for imperative or instructional sentences returning an incorrect SVO triple. We solve this problem by prepending "You should" to each instructional sentence. So that

[3] Spacy Sentence Boundary Detection https://spacy.io/usage/spacy-101.
[4] AllenNLP Open IE https://demo.allennlp.org/open-information-extraction.
[5] Spacy Dependency Parser https://spacy.io/usage/linguistic-features/#dependency-parse.

sentence like "Heat each phase to 70 to 75C and stir until uniform." from Fig. 3 reads as "You should heat each phase to 70 to 75C and stir until uniform". To solve the second problem, that of missing objects, we introduce a novel mechanism of using a stack as explained later in this section.

Additionally, we make use of a dictionary of verbs that are representative of actions performed on ingredients and/or mixtures. We compile the list using the 410 files and applying SBD and open IE to identify the verbs. Some of the example verbs are maintain, heat, add, stir, moisturize, cool, extract, demineralize, mix, disperse, blend, emulsify, select, distil, chelate, and so on. We use a total of 129 lemmatized verbs. As we will demonstrate in Sect. 6, we found that using a verb list is critical for accurate mapping of actions to ingredients/mixtures.

Figure 4 shows control flows for both open IE and dependency parsing for the extraction of what we refer to as Action Mixture/Ingredient Condition (henceforth A-M-C) structures.

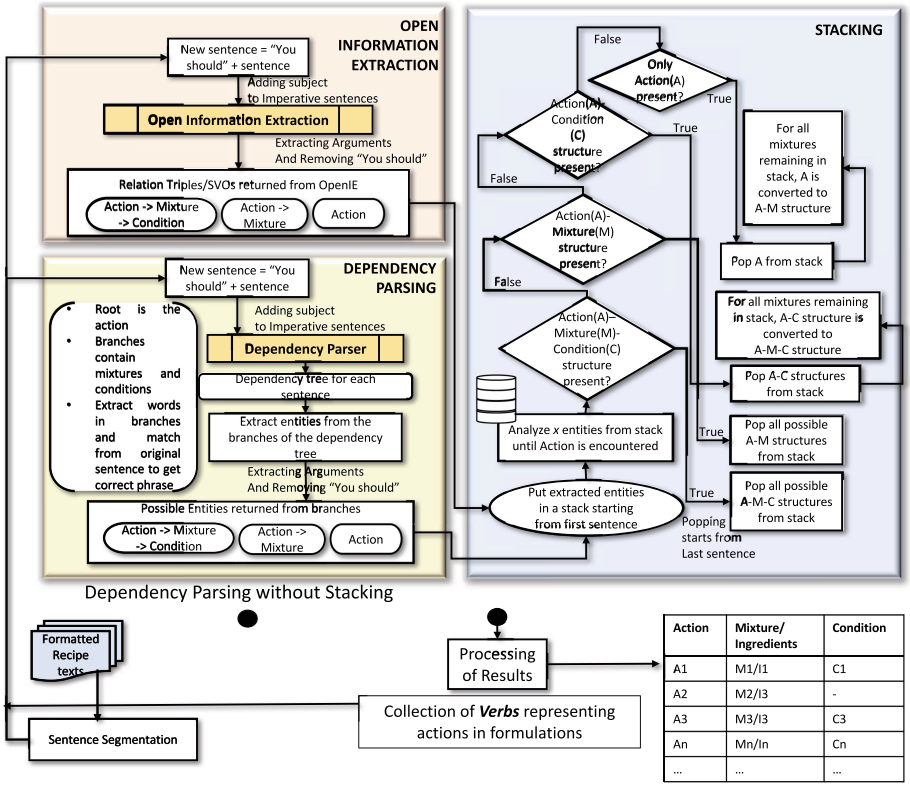


Fig. 4. Extraction of A-M-C structures from formulated product recipe text

Using Open IE with Stacking. Open IE model returns multiple (and possibly overlapping) triples of up to 4 values each; first value is the subject of the

sentence, the second value is the verb of the sentence, the third value is the first object of the sentence, and the fourth value is the second object of the sentence. We illustrate the complete process of applying open IE to extract A-M-C structure in Fig. 4. We process each triple to separate actions and ingredients.

Depending on the SVO triple returned, if action and two arguments are present, then we find all mixtures (from the mixtures dictionary) and ingredients (from earlier processing). If a mixture or an ingredient exists, we push it to a stack, and the action and the two arguments represent the current A-M-C structure. If a mixture or an ingredient does not exist, then the two arguments (values apart from the action verb) contain conditions. In this case, we use all mixtures from the stack and process output as action, mixture or ingredient, and the two arguments as a single entry, which now represents an A-M-C. If a triple only contains one argument, then we use all mixtures from the stack and process output as action, mixture or ingredient, and single argument to represent the A-M-C.

Using Dependency Parser with Stacking. To recognize the mixtures and conditions properly using dependency parsing, we convert multi-word mixtures to a single word using an underscore. In this case, a mixture identifier like *phase a* becomes *phase_a*. Figure 4 also illustrates the process of using a dependency parser. We extract all the branches from the root in the dependency tree and then process each branch based on the following rules:

- The root is the action (when the root is a verb from the verbs dictionary).
- Branches contain mixtures and conditions.
- If a branch contains two actions, we ignore the root action.
- We extract words in branches and match from original sentence to get the correct phrase to obtain the condition.

The rule-based extraction returns an Action-Mixture (A-M), an Action- Condition (A-C) or an Action-Mixture-Condition (A-M-C) structure. We push the structure to a stack starting from the first sentence. Words are popped from the stack until we encounter an action. If an A-M-C structure is present, then we pop it as a result. If an A-M structure is present, then we pop the A-M structure. If an A-C structure is present, then we pop the A-C structure from the stack and convert it into an A-M-C structure for all the unique mixtures remaining in the stack. Otherwise, if we encounter only A, then we pop A and convert it to an A-M structure for all the unique mixtures remaining in the stack.

We carry out the above steps recursively from the last sentence to the first sentence. The extracted results get rearranged according to their occurrence in the text, thus maintaining the order of actions.

Constructing Ingredient Dictionary for Synonyms and Functionalities. We observe that there are scarce offline resources to collect synonyms or other names of an ingredient as well as their functionalities but several online resources including ingredient entries at Wikipedia and specialized databases like the EU Cosmetic Ingredient Database[6]. We apply web scraping to several online resources to construct ingredient-synonyms and ingredient-functionality dictionaries.

[6] EU CosIng https://ec.europa.eu/growth/sectors/cosmetics/cosing_en.

A total of 2633 ingredients exist in 410 formulations, out of which 1086 are unique, and 333 ingredients repeat more than once. Chemical names are more prominently available in public datasets as opposed to an ingredient name occurring in a formulation. We were able to find chemical names for 447 ingredients. We search for sources such as Wikipedia[7], PubChem[8] [12], Chebi[9], and ChemSpider[10] to gather the desired information. The extracted data contains:

- IUPAC (International Union of Pure and Applied Chemistry) name, Synonyms, Chemical formula, Smiles (a representation of the chemical), PubChem CID, and *uses* or *application* section or functionalities from Wikipedia.
- Chemical Formula and PubChem CID from PubChem.
- Link to Chebi and ChemSpider from Wikipedia.

In the following sections, we present the storage and retrieval of the formulation constituents that we have extracted.

4 Storing Formulations as Graphs

Figure 5 shows the graph conceptual/domain model for cosmetic and toiletry formulations. In Fig. 5, the node FormulationType indicates the high-level formulation category. Since all our formulations are of creams which are of the type cosmetic and toiletry, for all 410 formulations under consideration, we set the label name of the FormulationType to *cosmetic and toiletry*.

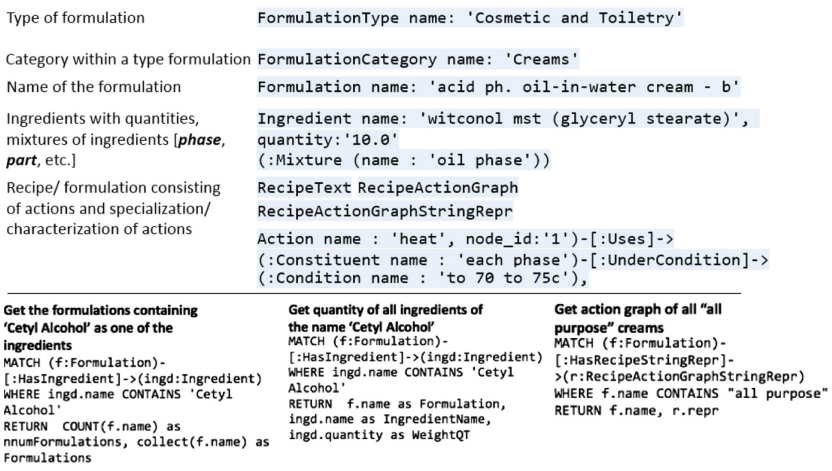


Fig. 5. Graph domain model (with formulation details from Fig. 3) and queries

[7] E.g., cetyl alcohol entry at Wikipedia https://en.wikipedia.org/wiki/Cetyl_alcohol.
[8] at PubChem https://pubchem.ncbi.nlm.nih.gov/compound/1-Hexadecanol.
[9] at Chebi https://www.ebi.ac.uk/chebi/searchId.do?chebiId=16125.
[10] at ChemSpider http://www.chemspider.com/Chemical-Structure.2581.html.

There are two reasons to choose a graph database. First, in contrast to relational databases, where join-intensive query performance deteriorates as the size of dataset increases, while a graph database performance tends to remain relatively constant, even as the dataset grows [1,16].

Second, graphs are also naturally additive, implying that we can add new nodes to represent hierarchies or taxonomies, new kinds of relationships between nodes, new nodes, and new subgraphs to an existing structure without disturbing current queries and application functionality [16].

```
MATCH (a:FormulationCategory) where a.name='Creams'
CREATE (a)-[:HasFormulation]->(f:Formulation {name:'acid-ph oil-in-water cream - b'}),
(f)-[:Source]->(:Source {name:'Vol1-1001.txt'}),
(f)-[:HasIngredient]->(:Ingredient {name:'witconol mst (glyceryl stearate)', quantity:'10.0'})-[:PartOf]->(:Mixture (name:'oil phase')),
(f)-[:HasIngredient]->(:Ingredient {name:'witconol apm (ppg-3 myristyl ether)', quantity:'3.0'})-[:PartOf]->(:Mixture (name:'oil phase')),
(f)-[:HasIngredient]->(:Ingredient {name:'perfecta petrolatum', quantity:'5.0'})-[:PartOf]->(:Mixture (name:'oil phase')),
(f)-[:HasIngredient]->(:Ingredient {name:'witconol h-35a (peg-8 stearate)', quantity:'5.0'})-[:PartOf]->(:Mixture (name:'oil phase')),
(f)-[:HasIngredient]->(:Ingredient {name:'witcamide m a s (stearamide mea stearate)', quantity:'3.0'})-[:PartOf]->(:Mixture (name:'oil phase')),
(f)-[:HasIngredient]->(:Ingredient {name:'emphos d70-30c (sodium glyceryl oleate phosphate)', quantity:'0.5'})-[:PartOf]->(:Mixture (name:'oil phase')),
(f)-[:HasIngredient]->(:Ingredient {name:'cetyl alcohol', quantity:'2.0'})-[:PartOf]->(:Mixture (name:'oil phase')),
(f)-[:HasIngredient]->(:Ingredient {name:'propylparaben', quantity:'0.1'})-[:PartOf]->(:Mixture (name:'oil phase')),
(f)-[:HasIngredient]->(:Ingredient {name:'emcol 4072 (disodium hydrogenated cottonseed glyceride sulfosuccinate) glycerin usp', quantity:'3.0'})
-[:PartOf]->(:Mixture (name:'water phase')),
(f)-[:HasIngredient]->(:Ingredient {name:'methylparaben', quantity:'0.15'})-[:PartOf]->(:Mixture (name:'water phase')),
(f)-[:HasIngredient]->(:Ingredient {name:'fragrance, color', quantity:'q.s.'})-[:PartOf]->(:Mixture (name:'water phase')),
(f)-[:HasIngredient]->(:Ingredient {name:'water', quantity:'q.s.'})-[:PartOf]->(:Mixture (name:'water phase')),
(f)-[:HasRecipe]->(g:RecipeActionGraph),
(g)-[:Contains]->(a1:Action {name:'heat', node_id:'1'})-[:Uses]->(:Constituent {name:'phase'})-[:UnderCondition]->(:Condition {name:'each to 70 to 75c'}),
(g)-[:HasStartNode]->(a1),
(g)-[:Contains]->(a2:Action {name:'stir', node_id:'2'})-[:Uses]->(:Constituent {name:'phase'})->(:Condition {name:'until uniform'}),
(g)-[:Contains]->(a3:Action {name:'add', node_id:'3'})-[:Uses]->(:Constituent {name:'water phase+oil phase'})
-[:UnderCondition]->(:Condition {name:'to the oil phase at 70 75c with moderate agitation'}),
(g)-[:Contains]->(a4:Action {name:'maintain', node_id:'4'})-[:Uses]->(:Constituent {name:'phase+water phase+oil phase'})
-[:UnderCondition]->(:Condition {name:'agitation temperature for 15 minutes'}),
(g)-[:Contains]->(a5:Action {name:'let', node_id:'5'})-[:Uses]->(:Constituent {name:'phase+water phase+oil phase'})
-[:UnderCondition]->(:Condition {name:'with slow stirring'}),
(g)-[:Contains]->(a6:Action {name:'avoid', node_id:'6'})-[:Uses]->(:Constituent {name:'phase+water phase+oil phase'})
-[:UnderCondition]->(:Condition {name:'air entrainment during cooling cycle'}),
(g)-[:Contains]->(a8:Action {name:'pour', node_id:'8'})-[:Uses]->(:Constituent {name:'phase+water phase+oil phase'})
-[:UnderCondition]->(:Condition {name:'at or below 28c'}),
RETURN f.name
```

Fig. 6. Generated query for formulation in Fig. 3

In case we were storing the details of a non-cosmetic and toiletry formulation, we would begin by adding a node of type FormulationType and setting the name property appropriately. Next, the node FormulationCategory captures the specific type of cosmetic and toiletry formulation, in our case, creams (or cream). Typically, for other cosmetic and toiletry formulations like antiperspirants and deodorants, we would set the name accordingly.

We generate and execute the combined MATCH and CREATE query[11] parts as shown in Fig. 6 for each formulation, such that we process a formulation text, generate a query and execute it to add a specific formulation to the graph.

This graph structure lends itself to intuitive queries. We show some example queries at the bottom of Fig. 5. Note the queries to find all the formulations containing the ingredient Cetyl Alcohol, and the weights of Cetyl Alcohol in those formulations. Another query shown in Fig. 5 retrieves ingredients of all creams of the specific kind such as *all purpose* or *skin whitening*.

In theory, having built a database of formulations of a specific type (with FormulationType nodes and the specific instances thereof), it is possible to query the details of similar FormulationCategory nodes and their constituents. This kind of query ability paves the way to the first step of intelligent design of formulated products as we show next.

[11] Cypher Query Language for Neo4j Graph Database https://neo4j.com/developer/cypher-query-language/.

5 Aiding Experts in Design of Formulated Products

Now that we have a database (and in effect a method to create such a database), it is possible to start planning the creation of a design variant as follows:

1. Given a specific kind of FormulationCategory, query the functionalities it usually contains.
2. For each functionality, query all the Ingredient instances associated with it.
3. Query the weight ranges of the ingredients via the quantity attribute.
4. Finalize the set of ingredients and/or mixtures.
5. Query the actions generally performed on each ingredient as a standalone or as a part of a mixture from the A-M-C structures stored as RecipeActionGraph instances.
6. Order the actions suitably to arrive at a complete formulation variant.

We prepare the following set of *neighbourhoods* as an aid to the expert:

– ingredients that never occur together, as well as those that occur together
– bidirectional neighbourhoods for functionality-ingredient and ingredient-actions, enabling to query functionality and actions of an ingredient and vice versa

It is possible to further cluster or rank the neighbourhoods of ingredients in terms of their functionalities, ingredients in terms of actions performed on them, and actions in terms of ingredients to which they apply.

6 Validation and Discussion

We describe validation of extraction, storage, and aided product design below.

Validating the Extraction of A-M-C Structures. To validate the extraction of A-M-C structures from the formulation recipes, we manually label recipes from 175 out of 410 formulation texts in terms of A-M-C structures. Figure 7 shows the F1 scores and score bins for both open IE and dependency parsing with and without a verb list and the stacking mechanism. The techniques, when combined, produce the best scores, as seen in Fig. 7.

For string similarity computation, we use an implementation of Levenshtein distance in Python[12].

We compute a similarity score and consider the prediction correct when the score is above a threshold of 75%. This threshold enables accommodating any small differences in the prediction and truth strings. If the predicted result contains more actions than truth, then we count them as false positives. If an action is missing from the predicted result, we count it as a false negative.

If we do not use the verb list, both open IE and dependency parser tag all possible verbs as actions. If we do not use the stacking mechanism, then the specific implementation misses out on mixtures or ingredients in most recipes,

[12] Fuzzywuzzy String Matching https://github.com/seatgeek/fuzzywuzzy.

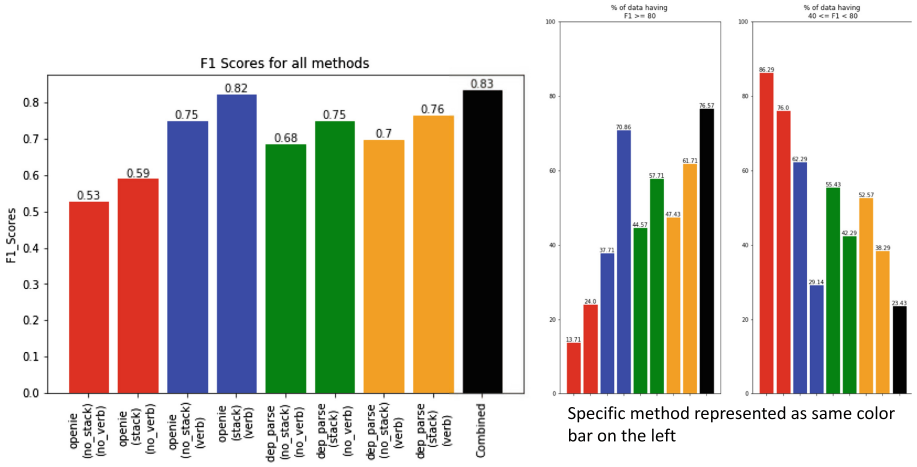


Fig. 7. F1 scores and score bins of extraction methods

where they are not explicitly mentioned. Consequently, when we use both the verb list and the stacking mechanism on top of open IE and dependency parser, we get better F1 scores.

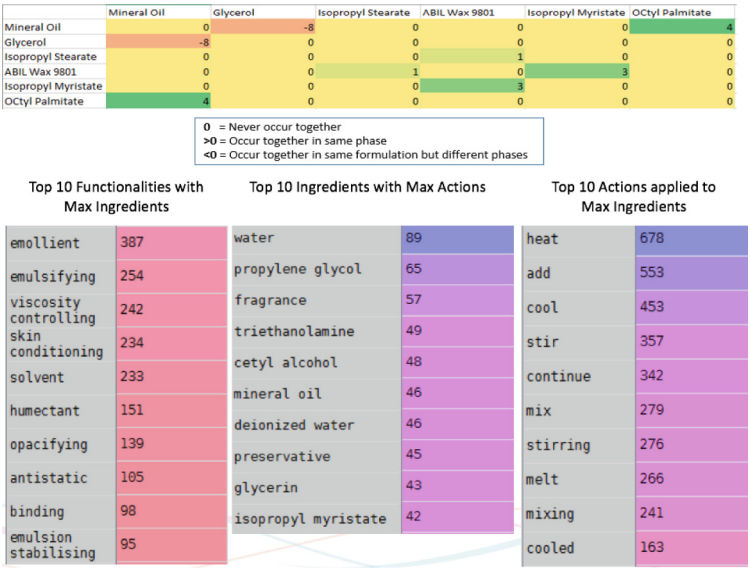
In the combined approach, we use both open IE and dependency parsing with the verb list and the stacking mechanism. Based on observations, we use two rules: a) choose actions from the method that gives more number of actions, and b) for each action, we choose the more descriptive mixture and condition, so that we don't miss out any information that may have been lost when using the specific extraction technique.

The score bins show that the combined method achieves more than 80% F1 score for three fourth of the set of 175 formulations.

Validating Neighbourhood Computation. The top of Fig. 8 shows the ingredients that never occur together, as well as those that occur together, esp. in the same phase or mixture. Such neighbourhoods or clusters of ingredients are useful because using the membership within a specific ingredient-ingredient neighbourhood, the choice of other ingredients can be informed.

At the bottom of Fig. 8, the results already explicate useful insights that tend to be implicit knowledge even if well understood. Functionalities such as emollients and viscosity controlling dominate due to formulations being creams of various kinds. Water, Propylene Glycol, Fragrance, Triethanolamine, and Cetyl Alcohol are the most common ingredients and Heat, Add, and Cool are some of the most frequently occurring actions.

On top of such neighbourhoods, we can also relate functionalities of ingredients to specific kinds of formulations and thereby to formulation categories. For instance, massage cream instances tend to contain ingredients with anti-static, binding, buffering, and denaturant functionalities, among others. Similarly,



	Mineral Oil	Glycerol	Isopropyl Stearate	ABIL Wax 9801	Isopropyl Myristate	OCtyl Palmitate
Mineral Oil	0	-8	0	0	0	4
Glycerol	-8	0	0	0	0	0
Isopropyl Stearate	0	0	0	1	0	0
ABIL Wax 9801	0	0	1	0	3	0
Isopropyl Myristate	0	0	0	3	0	0
OCtyl Palmitate	4	0	0	0	0	0

0 = Never occur together
>0 = Occur together in same phase
<0 = Occur together in same formulation but different phases

Top 10 Functionalities with Max Ingredients

emollient	387
emulsifying	254
viscosity controlling	242
skin conditioning	234
solvent	233
humectant	151
opacifying	139
antistatic	105
binding	98
emulsion stabilising	95

Top 10 Ingredients with Max Actions

water	89
propylene glycol	65
fragrance	57
triethanolamine	49
cetyl alcohol	48
mineral oil	46
deionized water	46
preservative	45
glycerin	43
isopropyl myristate	42

Top 10 Actions applied to Max Ingredients

heat	678
add	553
cool	453
stir	357
continue	342
mix	279
stirring	276
melt	266
mixing	241
cooled	163

Fig. 8. (Top) Ingredient-ingredient neighbourhood (Bottom) Statistics from functionality-ingredient, ingredient-actions and actions-ingredients neighbourhoods

chamomile cream instances tend to contain functionalities such as bulking, humectant, and plasticizer among others.

Validating Product Variant Generation Aid. We show an example of an aided variant generation process in Fig. 9 using a tool based on our approach. Our tool aids the formulator (expert) in making rational decisions regarding ingredient choices and helps him/her arrive at a possible recipe to be followed using the actions generally associated with the chosen ingredient.

For example, if a user wanted to design a variant of a face cream then the tool takes face cream as input product type and returns the count of face cream recipes in the database (11, in this case) along with the various ingredient functionalities associated with the face cream recipes. The user then has the option to specify the functionality to be explored, say emollient. Based on the functionality chosen, the tool returns 33 possible ingredient choices for this functionality from 11 face cream formulations. Given the possible emollient choices, the user then selects one or more ingredients based on experience such as Isopropyl Myristate based on the knowledge that it is a liquid of low viscosity, absorbs quickly and also acts as a permeation enhancer. After selection of the emollient Isopropyl Myristate, the tool outputs 10 ingredients which occurred together with this ingredient and their respective functionalities. The user can opt to choose optional functionality like anti-ageing similarly. We check the novelty of the combination by ensuring that this combination does not occur in any of the 410 cream recipes.

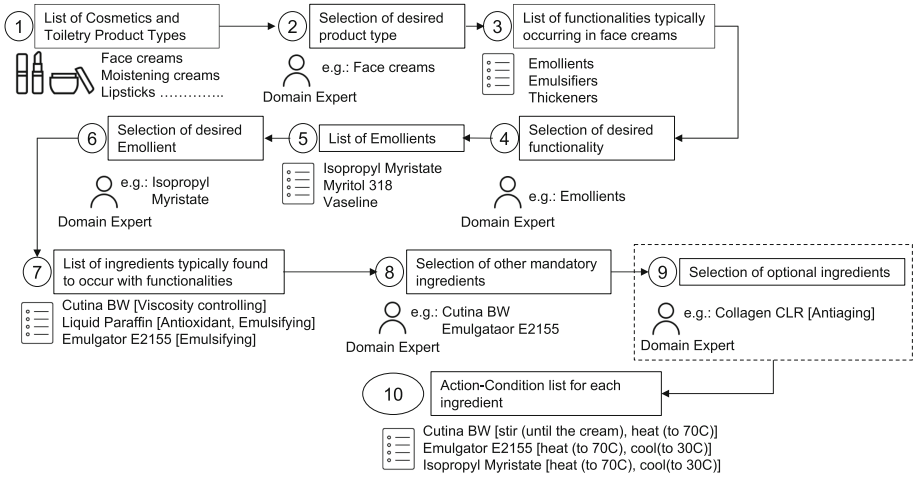


Fig. 9. Product design variant generation example

Once the formulator chooses all required ingredients, the tool outputs a list of A-M-C structures associated with each ingredient. The common A-M-C structures across all the chosen ingredients direct the formulator to the overall recipe steps. A possible recipe obtained as illustrated in Fig. 9 is as below:

```
Phase A: Isopropyl Myristate 10-35%, Cutina BW 0.1-2%,
Emulgator E2155 2-6%
Phase B: Water 55-75%, Collagen CLR 1-10%
Procedure: Heat Phase A and Phase B to about 70c.
Add Phase A to Phase B.
Continue stirring until the cream is emulsified.
Cool down to approximately 30c.
```

In the following, we briefly touch upon the limitations of our approach.

Extraction Limitations. Both open IE and dependency parsing incorrectly process a) gerund verb forms such as *mixing* in "disperse ... using high-speed mixing", and b) passive sentences, resulting in inaccurate triples and therefore, incorrect results. Similarly, if a sentence contains multiple actions as in "melt A and bring to about 70C", the processing fails to separate melting and bringing about as two different actions. We are currently extending annotated data by first creating A-M-C structures with our techniques and then manually correcting them. We plan to use machine learning techniques to overcome these limitations to some extent.

Data Availability Limitations. In spite of these aids, it is not possible to predict the final properties of the designed variant as in how adding or removing an ingredient affects the properties of the formulated product. Additionally, the sequence of recipe steps to be followed cannot be ascertained independently with just the A-M-C structures for the ingredients. We are currently compiling a set of heuristics to tackle these limitations.

7 Conclusion

Formulated products industry presents considerable opportunities for information systems such as one that we have detailed and demonstrated. Although it is tough to obtain and store details of every ingredient, its functionality, and kinds of products where it is useful, our approach provides a step in that direction, mainly if the user focuses on a specific product type like a cream or a coating.

References

1. Angles, R., Gutierrez, C.: Survey of graph database models. ACM Comput. Surv. (CSUR) **40**(1), 1 (2008)
2. Arrieta-Escobar, J.A., Bernardo, F.P., Orjuela, A., Camargo, M., Morel, L.: Incorporation of heuristic knowledge in the optimal design of formulated products: application to a cosmetic emulsion. Comput. Chem. Eng. **122**, 265–274 (2019)
3. Bernardo, F.P., Saraiva, P.M.: A conceptual model for chemical product design. AIChE J. **61**(3), 802–815 (2015)
4. Conte, E., Gani, R., Ng, K.M.: Design of formulated products: a systematic methodology. AIChE J. **57**(9), 2431–2449 (2011)
5. Dionisio, K.L., et al.: The chemical and products database, a resource for exposure-relevant data on chemicals in consumer products. Sci. Data **5**, 180125 (2018)
6. Flick, E.W.: Cosmetic and Toiletry Formulations, vol. 1–8. Elsevier (1989–2014)
7. Gani, R., Ng, K.M.: Product design-molecules, devices, functional products, and formulated products. Comput. Chem. Eng. **81**, 70–79 (2015)
8. Hill, M.: Chemical product engineering–the third paradigm. Comput. Chem. Eng. **33**(5), 947–953 (2009)
9. Kiddon, C., Ponnuraj, G.T., Zettlemoyer, L., Choi, Y.: Mise en place: unsupervised interpretation of instructional recipes. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp. 982–992 (2015)
10. Kim, E., Huang, K., Jegelka, S., Olivetti, E.: Virtual screening of inorganic materials synthesis parameters with deep learning. NPJ Comput. Mater. **3**(1), 1–9 (2017)
11. Kim, E., Huang, K., Saunders, A., McCallum, A., Ceder, G., Olivetti, E.: Materials synthesis insights from scientific literature via text extraction and machine learning. Chem. Mater. **29**(21), 9436–9444 (2017)
12. Kim, S., et al.: Pubchem substance and compound databases. Nucleic Acids Res. **44**(D1), D1202–D1213 (2016)
13. Lee, C., Choy, K.L., Chan, Y.: A knowledge-based ingredient formulation system for chemical product development in the personal care industry. Comput. Chem. Eng. **65**, 40–53 (2014)
14. Michalun, M.V., DiNardo, J.C.: Skin Care and Cosmetic Ingredients Dictionary. Cengage Learning, Boston (2014)

15. Mysore, S., et al.: Automatically extracting action graphs from materials science synthesis procedures. arXiv preprint arXiv:1711.06872 (2017)
16. Robinson, I., Webber, J., Eifrem, E.: Graph Databases. O'Reilly Media, Inc., Newton (2013)
17. Wibowo, C., Ng, K.M.: Product-centered processing: manufacture of chemical-based consumer products. AIChE J. **48**(6), 1212–1230 (2002)
18. Zhang, L., Fung, K.Y., Wibowo, C., Gani, R.: Advances in chemical product design. Rev. Chem. Eng. **34**(3), 319–340 (2018)
19. Zhang, L., Fung, K.Y., Zhang, X., Fung, H.K., Ng, K.M.: An integrated framework for designing formulated products. Comput. Chem. Eng. **107**, 61–76 (2017)