Artificial Intelligence for Engineering Design, Analysis and Manufacturing

cambridge.org/aie

Using evolutionary algorithms to select text features for mining design rationale

Miriam Lester[1], Miguel Guerrero[2] and Janet Burge[2]

[1]Department of Mathematics and Computer Science, Wesleyan University, Middletown, CT, USA and [2]Department of Mathematics and Computer Science, Colorado College, Colorado Springs, CO, USA

Research Article

Cite this article: Lester M, Guerrero M, Burge J (2020). Using evolutionary algorithms to select text features for mining design rationale. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **34**, 132–146. https://doi.org/10.1017/S0890060420000037

Received: 30 January 2019
Revised: 23 October 2019
Accepted: 30 October 2019
First published online: 30 January 2020

Key words:
Ant colony optimization; design rationale; feature selection; genetic algorithms; text mining

Author for correspondence: Janet Burge, E-mail: jburge@coloradocollege.edu

Abstract

At its heart, design is a decision-making process. These decisions, and the reasons for making them, comprise the design rationale (DR) for the designed artifact. If available, DR provides a comprehensive record of the reasoning behind the decisions made during the design. Unfortunately, while this information is potentially quite valuable, it is usually not explicitly captured. Instead, it is often buried in other design and development artifacts. In this paper, we study how to identify rationale from text documents, specifically software bug reports and design discussion transcripts. The method we examined is statistical text mining where a model is built to use document features to classify sentences. Choosing which features are most likely to be good predictors is important. We studied two evolutionary algorithms to optimize feature selection – ant colony optimization and genetic algorithms. We found that for many types of rationale, models built with an optimized feature set outperformed those built using all the document features.

Introduction

Designers make many decisions throughout the design process. These decisions, and the reasons behind them, form the design rationale (DR) for the system or artifact created. DR includes potentially useful information such as the alternative designs considered, reasons for and against these alternatives, questions raised during the process and the procedure followed to answer them, and the final decision made. This information has many potential uses. For example, understanding past decisions can help predict the results of new decisions in similar situations (Brazier *et al.*, 1997). The rationale provides a record of what the original designers of a system intended, which helps new designers joining the team remain consistent with that intent. The rationale can also be used to improve design processes (King and Bañares-Alcántara, 1997) and support design reuse (de Medeiros and Schwabe, 2008). If the rationale was available, designers and re-designers would be able to take this information into account when revising earlier decisions or making new ones.

There are many possible uses of DR, but it is often not captured explicitly because of perceptions that it is expensive and time-intensive (Burge and Brown, 2004). It is often captured implicitly, however, and potential sources include e-mail discussions between design contributors, transcripts of design discussions, project specifications, and domain-specific documents, such as software bug reports. DR could be manually extracted from these text documents, but this task is considered tedious and time-consuming. A preferable approach would be to have automated extraction tools to identify and extract the DR from existing documents. The extracted rationale could then be structured and formatted as needed, a process called "incremental formalization" (Shipman and McCall, 1994).

We have been investigating techniques for mining rationale from text documents using text mining techniques to classify sentences as rationale. These techniques use features found in the documents being classified as an input into machine learning classifiers to build models that can be used to classify additional data. Document features that can be used include parts-of-speech (verbs, nouns, adverbs, etc.), word combinations (n-grams), sentence length, and domain-specific terminology. Finding good feature sets for text mining is a critical part of this process. The feature sets available for text mining are numerous, and it is difficult to determine which ones will have the most predictive power. While it is sometimes possible to use all of the features when text mining, this can cause problems when irrelevant features, features that do not relate to whether a sentence contains rationale or not, are used and less, then optimal results are returned. Using all of the features is also very expensive when considering the time and space requirements of a text mining algorithm.

Feature sets can be selected using statistical techniques (such as chi-squared, document frequency, mutual information, and term strength) (Aghdam *et al.*, 2009). Alternative approaches use nature-inspired algorithms such as ant colony optimization (ACO) and genetic algorithms (GAs). These algorithms perform a broader search of the feature space by introducing

© Cambridge University Press 2020

CAMBRIDGE UNIVERSITY PRESS

randomness along with the knowledge from previous iterations to search for more optimal feature sets (Aghdam *et al.*, 2009).

In the work described in this paper, we decided to compare two different evolutionary algorithms – ACO (Dorigo, 1992) and GAs (Holland, 1975/1992). ACO simulates the process that ants take to find food sources, while GAs mimic the process of natural selection by representing solutions as chromosomes and evolving them over multiple generations (iterations).

This work addresses the following research questions:

1. Does using evolutionary algorithms for feature selection improve performance when identifying rationale compared to building classifiers without feature selection?
2. How does the classification performance of the ACO selected feature set compare to the GA selected feature set?

We measure performance using the F-1 measure, the harmonic mean of the precision and recall. The F-1 measure takes into account false positives and false negatives. This makes it a better representation of how successful a classifier is than measuring accuracy, which can give misleading results if a dataset is imbalanced (Skiena, 2017). For example, if 10% of the instances belong to the target category and all instances are given a negative classification, then the accuracy would be 90% even though none of the target instances were found.

The remainder of the paper is structured as follows: The "Related research" section describes related work in feature set selection and rationale extraction. The "Approach for mining rationale" section describes our approach which includes how we prepared our training data, a description of the "pipeline" we used to integrate evolutionary algorithms for feature selection with a classifier to extract rationale, our approach to using ACO for feature selection, and how we used a GA for feature selection. The "Results" section gives the results of our experiments, and the "Conclusions and future work" section summarizes our results and plans for future work.

Related research

DR is an active area of research in many domains, including Human–Computer Interaction (Moran and Carroll, 1996), Software Engineering (Dutoit *et al.*, 2006; Burge *et al.*, 2008), and Engineering Design (Chung and Bañares-Alcántara, 1997; Lee, 1997; Burge and Bracewell, 2008). Here, we are focusing on work in two areas: feature set selection and rationale extraction.

Feature set selection

Text mining models work on looking for patterns of text features (specific words, parts-of-speech, etc.) that indicate if the rationale is present. Features that are irrelevant (those without any correlation to the classification goals) can confuse classifiers that will try to fit to them and lead to sub-optimal performance then if only a subset of the features were used. Feature selection refers to the process of choosing which features are most likely to be predictive.

Many variations of ACO algorithms are having success over nonevolutionary methods when applied to feature selection for classification problems. Al-Ani (2005) used ACO to select feature sets for speech segment and image texture classification. He compared results using ACO with results from a GA. ACO slightly outperformed the GA with a classification accuracy of 0.842 versus the GA accuracy of 0.835. We cannot compare these results to ours since we chose to use the F-1 measure rather than accuracy since accuracy can produce misleading results in imbalanced datasets.

Aghdam *et al.* (2009) used the ACO algorithm to reduce the dimensionality of the search space for a text categorization problem. They compared the performance of their ACO-based approach to GA, chi-squared (CHI), and information gain (IG), on the Reuters-21578 benchmark dataset. In the experiments, ACO outperformed all other methods. They calculated a Macro-F1 score (an F-1 measure that calculates a score for multiclass classification by equally weighting all classes). Their experiments resulted in Macro-F1 scores of 0.784 with ACO, 0.763 with GA, 0.709 with CHI, and 0.698 with IG. They also calculated Micro-F1 scores (a measure that gives the highest weights to more common classes) of 0.891 with ACO, 0.864 with GA, 0.822 with CHI, and 0.809 with IG.

Saraç and Özel (2014) used the ACO for web page classification. They worked with a very high-dimensional feature space that consisted of pairs of "tagged terms" (e.g., <url><term>). For each of the five datasets examined, they achieved better F-measures by using ACO selected features. The ACO had an average F-measure of 0.952, compared to 0.684 without feature selection. They also implemented ACO with an ant feature subset construction method that reduced unnecessary computation by choosing features in groups rather than individually. We integrated this approach into our ACO as well.

GAs are also used in feature selection. Ozyurt (2012) used a GA-based feature selection technique to classify biomedical literature. Mukherjee *et al.* (2010) used a GA to select optimal features for classifying e-mails. They form chromosomes from topics in the document and select the "parents" for the next generation by randomly choosing half of the chromosomes with the highest fitness.

Hybrid algorithms capture the best features of multiple algorithms. Zaiyadi and Baharudin (2010) performed feature subset selection by hybridizing ACO with IG. They used this technique to reduce the dimensionality of feature space for text document categorization by using IG as the heuristic desirability measure for each ant. Ali and Shahzad (2012) combined ACO with symmetric uncertainty (SU) (a variation of IG). In the ACO-SU algorithm, the fitness is calculated by weighting the length of the chosen subset and the sum of the SU for all the features chosen in the subset. We have inspired these two methods and used the IG statistic of chi-squared as our measure of a heuristic value.

Some hybrid algorithms combine GAs and ACO algorithms. Basiri and Nemati (2009) did this by having ants select a population and then applying selection, mutation, and crossover. They then exchanged poor performing individuals with better ones found by the GA. This hybrid algorithm performed better ACO alone. Roeva *et al.* (2013) combined ACO and GAs by using ACO to generate an initial population for the GA, so the GA could start with a population that was nearer to an optimal solution than one selected randomly. This allowed them to reduce computational time by using smaller populations.

Jiang *et al.* (2009) hybridized the GA with a taboo search algorithm for feature subset selection for text categorization. They incorporated the taboo search's memory function into the GA's evolution-based search.

Rationale extraction

Liang *et al.*'s work (2012) extracted DR from patent documents. They used a three-tiered model to capture issues, design solutions,

https://doi.org/10.1017/S0890060420000037 Published online by Cambridge University Press

and artifacts. They started by identifying artifacts using a modified PageRank (Brin and Page, 1998) algorithm on frequently appearing words. Issue summarization was then done by using issue language patterns in the manifold ranking. The final step identified reason sentences and paired them with the remaining solution sentences to create reason-solution pairs. This was done using reason language patterns. They achieved a 0.185 F-measure for artifact identification, a 0.520 F-measure for issue summarization, and a 0.562 F-measure for reason-solution extraction.

López *et al.* (2012) worked on recovering DR from existing software documents, as well as representing the rationale and integrating the rationale with a software tool. They use ontology-based extraction of software design rationale, where the ontologies relate concepts of software architectures and thesauri of relevant terms. They used documents for designs of a security clearing system to evaluate their tool and achieved an F-measure of 0.5 for recovering rationale, beating the F-measure for manual recovery of 0.42 and taking less time.

Mao *et al.* (2014) looked for argument rationale in the Wikipedia article for deletion discussion forums. Their main goal was to identify direct imperative arguments. They achieved an F-measure of 0.7874 on the Wikipedia data.

Kurtanovic and Maalej (2018) used Amazon reviews of software products as a source of user rationale. They trained their classifiers on a set of influential terms, review metadata, and a syntax tree of the review text. They were looking for five categories of rationale: issues, describing problems with the software; alternatives, which could refer to alternative software products, alternative versions of the software, alternative features of a different type of software, and other types of alternatives; criteria, which include usability, reliability, performance, supportability, and other criteria; decisions, which include acquiring software, relinquishing software, switching software, and other decisions; and justifications, which are sentences that justify other sentences that contain purpose clauses, reason clauses, and independent clauses. They evaluated seven different classification algorithms. The algorithms, features, and preprocessing step combinations were combined randomly, and they evaluated between 10,109 and 17,457 different configurations. The evaluation was done using cross-validation rather than holding out data for testing. They were able to get F-1 measures of 0.77 for issues, 0.82 for alternatives, 0.77 for criteria, 0.83 for decisions, and 0.74 for justifications.

Alkadhi *et al.* (2018) used machine learning techniques to classify Internet Relay Chat (IRC) messages into those with rationale and those without. They achieved an F-1 measure of 0.61 for classifying messages with rationale at the message level. This was done using 3-fold cross-validation where each fold consisted of messages from a different project. The classifier with the best results was multinomial Naïve Bayes.

Rogers *et al.* (2012) experimented with using two feature sources – ontologies (vocabularies of potential arguments and domain terminologies) and a small set of linguistic features (modal auxiliaries, adverbial clauses, and projective clauses). They used these features with a large number of different classifiers to identify DR in Chrome bug reports. The best F-1 measure achieved with ontologies was 0.597 for binary classification (rationale/not rationale). This small set of linguistic features resulted in an F-1 measure of 0.336. Expanding to a larger set of linguistic features and a more rigorously annotated dataset produced better results; improving the F-1 measure to 0.676 for binary classification and 0.569 for the argumentation subset (binary classification of rationale excluding the questions, answers, and procedures that commonly occurred along with boilerplate text making them easier to identify) (Rogers *et al.*, 2014). Our earlier work using GAs and WEKA (Waikato Environment for Knowledge Analysis) for feature selection and classification (Rogers *et al.*, 2016) resulted in F-1 measures of 0.576 for the argumentation subset of the Chrome bug reports (using 10-fold cross-validation). This was not the same implementation of the GA described in this paper and used fewer document features.

McCall (2018) also extracted the rationale from design discussions. This was done by building an Argumentative Semantic Grammar, ASGARD (Argumentative, Semantic Grammar for Analysis of Rationale for Design). This grammar contained rules for parsing text and forming it into a version of the PHI (Procedural Hierarchy of Issues) schema (McCall, 1991). This grammar was customized for the particular design transcript, so more testing is needed to see how it generalizes to other documents.

A final set of papers refers to arguments in legal texts, which tend to have a higher density of rationale [about 50% for legal texts (Palau and Moens, 2009) compared to 11% for bug reports (Rogers *et al.*, 2014)]. Prakken *et al.* (2003) explored the formalization of rationale arguments in legal texts, using common legal argumentation structures as a base. Moens *et al.* (2007) explored the automatic extraction of rationale arguments from legal texts, experimenting with a number of different feature sets. They found that the best performing feature sets were unigrams, bigrams, word couples, and combinations including the three, with F-measures ranging from 0.704 to 0.738. Palau and Moens (2009) built on the previous rationale argumentation work by adding a context-free grammar to automatically structure argumentation text, obtaining around 60% accuracy.

Approach for mining rationale

DR could appear in a variety of different types of text documents. Some are domain neutral, such as meeting transcripts, meeting minutes, e-mail messages, discussion boards, and design documentation. Others are more domain-specific, such as the bug reports used in software development projects. Our eventual goal would be to take a potentially large number of documents and extract the rationale, so it would be available to engineers to assist with making and revising decisions.

We followed a text mining process that used a set of data labeled as rationale to train a classifier that could then be used to classify additional data. The following sections describe how we prepared our training data, a description of our general pipeline used to build the classifiers, and then the two specific feature selection algorithms used in this work.

Training data

We annotated documents with rationale using GATE (General Architecture for Text Engineering) (Cunningham *et al.*, 2011) with by having two researchers annotate and a third researcher adjudicate using a process described in Rogers *et al.* (2014). We also used GATE to automatically annotate the parts-of-speech from the Penn Treebank (Marcus *et al.*, 1993) and custom features specific to this project: ontology terms (Burge, 2005; Rogers *et al.*, 2012), sentence length (Rogers *et al.*, 2014), and acronyms (Mathur, 2015).

https://doi.org/10.1017/S0890060420000037 Published online by Cambridge University Press

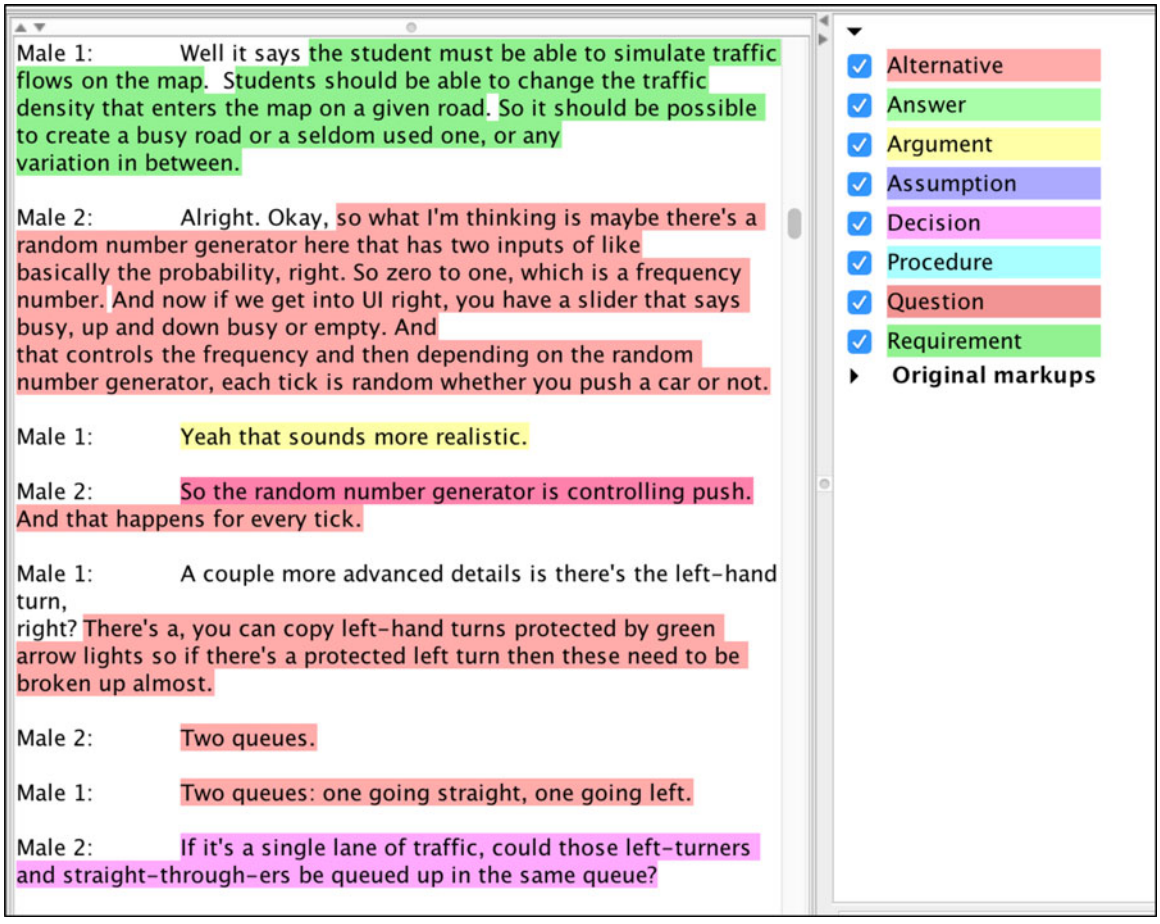


Fig. 1. Annotated document in GATE.

The experiments described here worked with two different datasets. The first consisted of bug reports for the Chrome web browser that were provided for the Mining Software Repositories Mining Challenge (http://2011.msrconf.org/msr-challenge.html). We selected 200 bug reports from this repository. Our second dataset consisted of transcribed design discussions that took place during the Studying Professional Software Designer (SPSD) project. These two datasets provide two different views of rationale sources in the category of design communication (Shipman and McCall, 1997) where one is asynchronous (bug reports) and the other synchronous (discussion transcripts). The bug reports are a type of document specific to a domain (software engineering), while design discussion transcripts are more domain neutral, although these specific transcripts involved discussions between software developers.

The data wasannotated looking for specific types of rationale [adapted from Burge and Brown (2003)]:

- *Requirements* – statements referring to functionality that the software was required to have;
- *Decisions* – statements that indicated what issue had to be resolved;
- *Alternatives* – different options for resolving the issue described by a decision;
- *Arguments* – reasons for or against an alternative;
- *Assumptions* – claims made where the author indicated uncertainty;
- *Questions* – questions posed that indicate where more information is needed to make a decision;
- *Procedures* – description of actions needed to gain information required to answer a question or make a decision;
- *Answers* – answers to questions posed in the rationale.

A sentence can be classified as more than one type of rationale. Figure 1 shows some example annotations inside GATE, the tool that was used to annotate the documents. This example shows a selection from the SPSD dataset.

The experiments described in this paper were designed to look for five different classification targets:

- *Binary rationale* – a sentence was either rationale (any one of the above types) or nonrationale.
- *Argumentation subset* – a sentence was either argumentation (a requirement, decision, alternative, argument, or assumption) or not.
- *Decisions* – a sentence was a decision or not.
- *Alternative* – a sentence was an alternative or not.
- *Arguments-all* – a sentence was a requirement, argument, or assumption or not.

The argumentation subset classification was used to compare results that included questions, procedures, and answers to those that did not. Questions, procedures, and answers frequently appeared in the boilerplate, common text at the start of each bug

https://doi.org/10.1017/S0890060420000037 Published online by Cambridge University Press

report. This made them easier to classify than the argumentation. The arguments-all classification was used because determining if an argument referred to a requirement, assumption, or something else was subjective. What might be a requirement for the author of a bug report (for example) might not actually be a requirement for the system being described.

Pipeline for text processing and model building

For our earlier work (Rogers *et al.*, 2016), we built the GATE_WEKA pipeline, which started with exported GATE XML files and automatically ran WEKA tests as specified by the user. This has a number of limitations, not the least of which were feature set size limitations imposed by WEKA. For this work, we built a new pipeline that was no longer GA-specific, so we could use the same pipeline for both the ACO and GA-based subset selection algorithms.

Figure 2 shows the pipeline for building classifiers. The following paragraphs describe each major component.

Sentence parser

The input to the pipeline is a file (sentences.csv) that contains the features extracted from each sentence along with its rationale annotations. The sentence parser uses the sentence.csv file to create a sentence dictionary data structure for each sentence (Fig. 2). The dictionary structure organizes all the features where each (key, value) pair holding information about an individual feature category. The key is the ID of the feature category, and the value is the set of all feature instances of that feature category for that sentence. With a dictionary structure, each category can be accessed in constant time, allowing the feature set to be created from a simple union of instances from each category in the chosen subset of categories. In addition to discrete feature types, such as verbs and adverbs, we also used n-grams of words and feature types (as shown in Fig. 3). We captured features from adjacent sentences since context is likely to be important in identifying rationale.

We split the data into training (70%) and test (30%) sets, being careful to not have sentences in the same bug report or transcript appear in training and test. The corresponding entries from the sentence dictionary are written into either the Training Sentences File or the Test Sentences File.

Feature subset selection

This component is implemented using either the ACO or GA, as described in "ACO for feature selection" and "GAs for feature selection" sections. The common aspects to these algorithms are that they iteratively search for a feature subset, train a classifier using those features, and then use the results of the training to choose features for the next iteration. This is done using the Training Sentences File.

Classifier building and evaluation

The optimized feature sets are used to train an NLTK Naïve Bayes classifier using 10-fold cross-validation. After training, we test the classification model using the training set (the 30% of the sentences pulled out before training).

We performed a number of preliminary experiments using different classifiers to select the one to use in our approach. Many did not successfully complete training on our data. Of those that did, the NLTK Naïve Bayes classifier achieved the best F-1 measure.

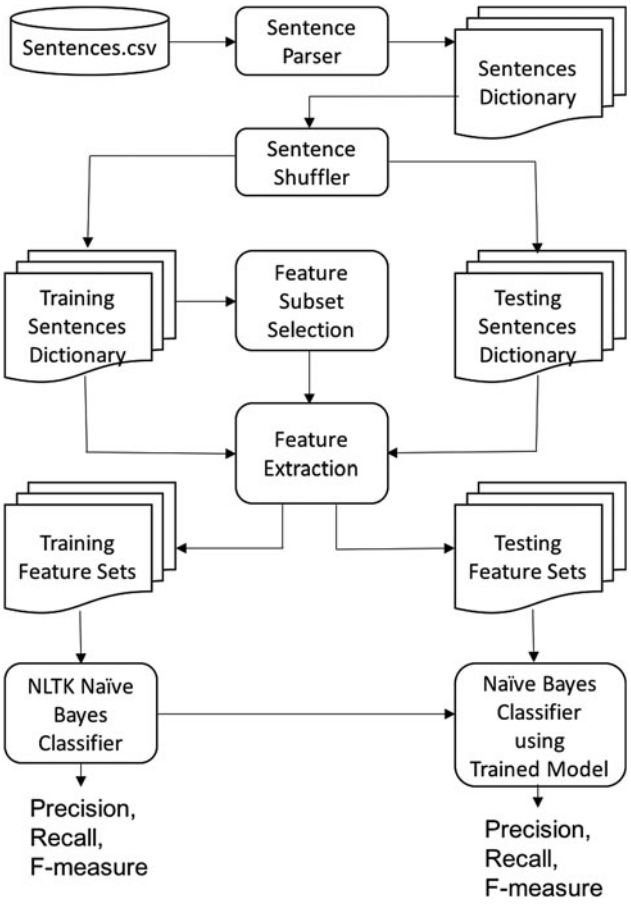


Fig. 2. Pipeline for classifier building (Lester and Burge, 2018).

Example:
For the sentence "I decided to use HTML"

	0	1	2	3	4
with feature categories =	[unigrams,	bigrams,	verbs,	adverbs,	acronyms]

Sentence dictionary representation:

```
{
   0: {"i", "decided", "to", "use", "html"},          # unigrams
   1: {"idecided", "decidedto", "touse", "usehtml"},  # bigrams
   2: {"decided", "use"},                             # verbs
   3: {}                                              # adverbs
   4: {"html"}                                        # acronyms
}
```

Fig. 3. Sentence dictionary representation.

ACO for feature selection

In 1990, Deneubourg *et al.* (1990) showed that foraging ants find the shortest path between their nest and a food source by using pheromone trails. Ants lay down pheromone trails as they forage for food and these are reinforced as they travel to and from food sources. Dorigo (1992) showed that computer simulation of an ant colony can model this process; a method known as the ACO metaheuristic. Many different variations of the ACO were developed, including a version of the ACO for subset problems (Leguizamon and Michalewicz, 1999). We applied ACO to our feature subset selection problem by having each ant construct a subset of features where pheromone trails are applied feature

https://doi.org/10.1017/S0890060420000037 Published online by Cambridge University Press

components instead of connections between components using some measure of heuristic desirability of each feature.

Feature subset problems can be considered combinatorial optimization problems. To do this, we map the combinatorial optimization problem to a problem characterized by a finite set $C = \{c_1, c_2, \ldots, c_n\}$ of components, where n is the number of components. Artificial ants perform randomized walks on a completely connected graph $G_c = (C, L)$ whose nodes are the components C, and the set L fully connects the components of C. The graph G_c is called a *construction graph*, and the elements of L are called *connections* (Dorigo and Stützle, 2004). These walks form the solutions. Artificial ants construct solutions, constrained so that components are only added to the current solution if the resulting solution is feasible (Dorigo and Stützle, 2004). Each component $c_i \in C$ is associated with a pheromone trail τ_i and a heuristic value η_i. Each ant updates the pheromones, which constitute a global, long-term memory of good solutions found throughout the search procedure. Pheromone trails evaporate over time to avoid converging to sub-optimal local extrema. We also use a heuristic value which captures information about the problem obtained from a source other than the ants (such as the cost of adding a component to the solution).

Simulated ants move by applying a probabilistic transition rule, which is a function of the locally available pheromone trails, problem constraints, and heuristic values. After a solution has been built, the pheromone levels of the components along their path are updated based on solution quality. Quality (fitness) is evaluated using the Naïve Bayes algorithm. Dorigo explains his algorithm by saying, "It is important to note that ants act concurrently and independently and that although each ant is complex enough to find a (probably poor) solution to the problem under consideration, good-quality solutions can only emerge as the result of the collective interactions among the ants" (Dorigo and Stützle, 2004).

ACO is traditionally applied to ordering problems such as the traveling salesperson problem, where the goal is to find the shortest path that visits all the cities (Dorigo and Stützle, 1999). The ACO can also be used to solve subset problems (Leguizamon and Michalewicz, 1999). In our problem, we are looking for subsets of features. This means that nodes in the graph represent features and the solution is the set of features chosen by the ant. Solutions are constructed by applying a probabilistic transition rule were ants add available features to their current subset until meeting a stopping criteria. Pheromone trails are associated with components (features), not connections.

ACO has been applied to many different types of subset problems (Dorigo and Stützle, 2010), including the Multiple Knapsack problem (Leguizamon and Michalewicz, 1999), Set Covering problem (Lessing *et al.*, 2004), and Maximum Clique problem (Solnon and Fenet, 2006). ACO has been further extended to a wide range of machine learning problems, including learning the structure of a Bayesian network (de Campos *et al.*, 2002) and learning rules in a Fuzzy system (Casillas *et al.*, 2000).

Implementing the ACO requires defining the following critical components (Dorigo *et al.*, 1996; Aghdam *et al.*, 2009; Basiri and Nemati, 2009; Kanan *et al.*, 2007):

1) *Graphical representation of the problem*: A fully connected graph with nodes and edges between nodes representing the discrete search space of the problem (Basiri and Nemati, 2009). It must be able to represent a solution to the problem. For our problem, each node is a feature category and traversing connections means selecting the category.
2) *Feasible solution construction constraint*: A mechanism to make sure only feasible solutions are built (Basiri and Nemati, 2009). In our case, we stop after a pre-specified number of nodes has been chosen, with each one chosen by satisfying the probabilistic transition rule.
3) *Heuristic desirability*: A "measure of goodness" (Kanan *et al.*, 2007) of adding any component to a partially constructed solution. Because this is a subset problem, the heuristic value is associated with a feature instead of the edge between two features (Leguizamon and Michalewicz, 1999). We defined local importance (LI) as the heuristic measure and calculated LI as the highest chi-squared statistic of all terms in a feature category.
4) *Pheromone update rule*: A rule for updating pheromone levels – a strategy for increasing pheromone concentration for previously successful solutions, and an evaporation rule to globally decrease pheromone levels over time (Kanan *et al.*, 2007) This constitutes the autocatalytic feedback process. Because this is a subset problem, the pheromone value is associated with a feature rather than the edge between two features. The pheromone update rule for an ant is shown below, where pheromone of feature i is being updated from $T_i \rightarrow T_i'$, with ρ representing the pheromone evaporation rate. Our ACO implementation follows the standard method of selecting the k best ants (those whose solutions had the highest fitness) and updating the paths they took (Kanan *et al.*, 2007), to reinforce the success of these best paths.

   For each feature f_i, the new pheromone level for an ant in the set of k best ants is given as follows:

$$\Delta T_i = \frac{\text{fitness(ant)} - \text{worst}_{\text{fitness}}}{\max_{j=1:k} (\text{fitness}(j) - \text{worst}_{\text{fitness}})},$$

$$T_i' = (1 - \rho)^* \; T_i + \Delta T_i,$$

   where ρ is the evaporation rate.

   We chose 20 as the number k, 0.1 as the pheromone evaporation rate (ρ), and our fitness function used the NLTK Naïve Bayes classifier.
5) *Probabilistic transition rule*: A solution construction rule that computes the probability of an ant next moving to a new node in the graph (Basiri and Nemati, 2009). In our implementation, each ant chooses p features that maximize updated selection measurement (USM).

$$\text{USM}_i^{S_j} = \begin{cases} \dfrac{(T_i)^{\eta}(\text{LI}_i^{S_j})^{\kappa}}{\sum_{g \notin S_j} (T_g)^{\eta}(\text{LI}_g^{S_j})^{\kappa}} & \text{if } i \notin S_j \\ 0 & \text{otherwise} \end{cases}.$$

The USM of feature i with respect to S_j (the subset of ant j): T_i is the pheromone level of feature f_i; η is the relative weight of pheromone intensity; LI_i is the local importance of feature f_i; and κ is the relative weight of local importance.

We chose 1 as the initial pheromone level (T_i), and 1 and 2 as the relative weights of pheromone intensity (η) and local importance (κ), respectively. These were determined experimentally to

https://doi.org/10.1017/S0890060420000037 Published online by Cambridge University Press

decrease the speed of convergence, so ants did not get stuck in poor local optima.

Figure 4 shows the diagram of our ACO implementation; based on an algorithm outlined by Al-Ani (2005). We chose 100 as the number of ants (NA) and 40 as the number of iterations. When selecting each ant's features for the next iteration, we chose three-fourths from the features used in the 20 best ants, and the remaining one-fourth from features currently not assigned to the ant that maximize the update selection rule (USM). Most of the ACO parameters were chosen to be consistent with parameters used by the GA (100 ants to be similar to 100 chromosomes, 40 iterations for each algorithm, 20 best ants to be similar to the feature subset size used in the GA).

GAs for feature selection

The GA (Holland, 1975/1992) is a method inspired by natural selection that uses a population of candidate solutions (chromosomes) that are evolved over a number of generations (iterations). During each iteration, their fitness is evaluated, and the successful solutions are combined and/or mutated to create new solutions. This repeats until a stopping criteria is met, and the best solutions are returned as potentially optimal solutions (Martin-Bautista and Vila, 1999). The GA can be applied to the feature selection problem by having chromosomes represent candidate subsets of features and using the F-measure of classifying sentences with only features from the selected subset as a measure of fitness.

In a successful run of a GA, the population of chromosomes converges so that each individual has a very similar genotype, and the chosen chromosome is a near-optimal solution to the central optimization problem (Anderson and Ferris, 1994). GAs are superior to traditional search methods in that they explore more solutions to the optimization problem in parallel, and therefore are less likely to become trapped in local sub-optimal areas of the search space (Atkinson-Abutridy *et al.*, 2004).

A GA implementation must define the following components:

1) Chromosome representation: Chromosomes represent potential solutions to the optimization problem, and each gene represents one feature category. These chromosomes contain a fixed length list of feature categories that represent a candidate feature subset. If a feature category is included as a gene in a chromosome, all instances of features in that category will be considered relevant and included in the resultant classifier (Martin-Bautista and Vila, 1999). Each chromosome represents an unordered set of feature categories; therefore, the order of the genes does not matter. The initial population of chromosomes has randomly selected features and is evolved using selection, crossover, and mutation.

   Figure 5 shows an example chromosome with the feature categories. Some categories include n-grams (denoted by (n) prefix) and instances from adjacent sentences (denoted by {m} prefix).
2) *Fitness function:* The fitness function is used to evaluate how good each solution (chromosome) is. We used the NLTK Naïve Bayes classifier and computed the F-measure.
3) *Selection process*: A process for choosing which chromosomes are used to create new solutions. Parent chromosomes with the highest fitness (for our case, the average Naïve Bayes F-measure over 10-fold cross-validation) are chosen to "reproduce" in every next generation (Goldberg, 1989). This is an example of *truncated selection* (Mukherjee *et al.*, 2010), where only the top chromosomes can be selected for reproduction.
4) *Crossover and mutation algorithms*: The algorithms that decide how to create the next generation of solutions. Crossover (depicted in Fig. 6) is performed by randomly selecting two different parents from the pool of chromosomes chosen to repopulate the next generation. Then, half of the categories from each parent are randomly selected and combined into a new set representing the "child" chromosome with any

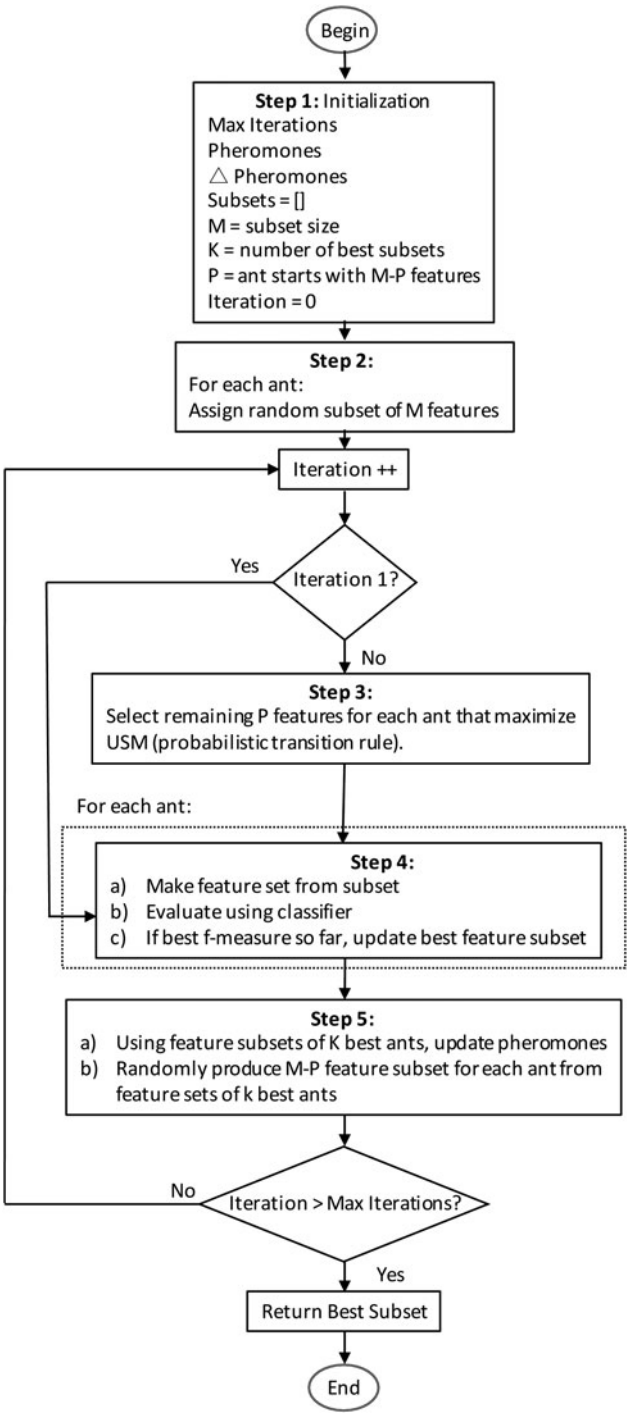


Fig. 4. ACO flow chart (Lester and Burge, 2018).

https://doi.org/10.1017/S0890060420000037 Published online by Cambridge University Press

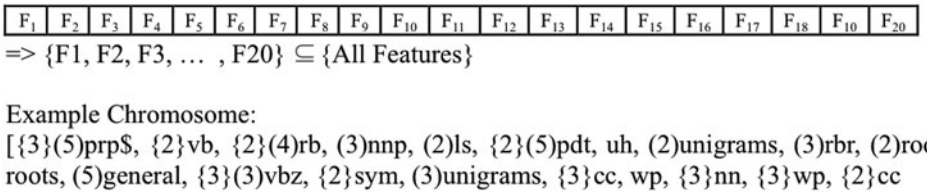


Fig. 5. Chromosome structure.

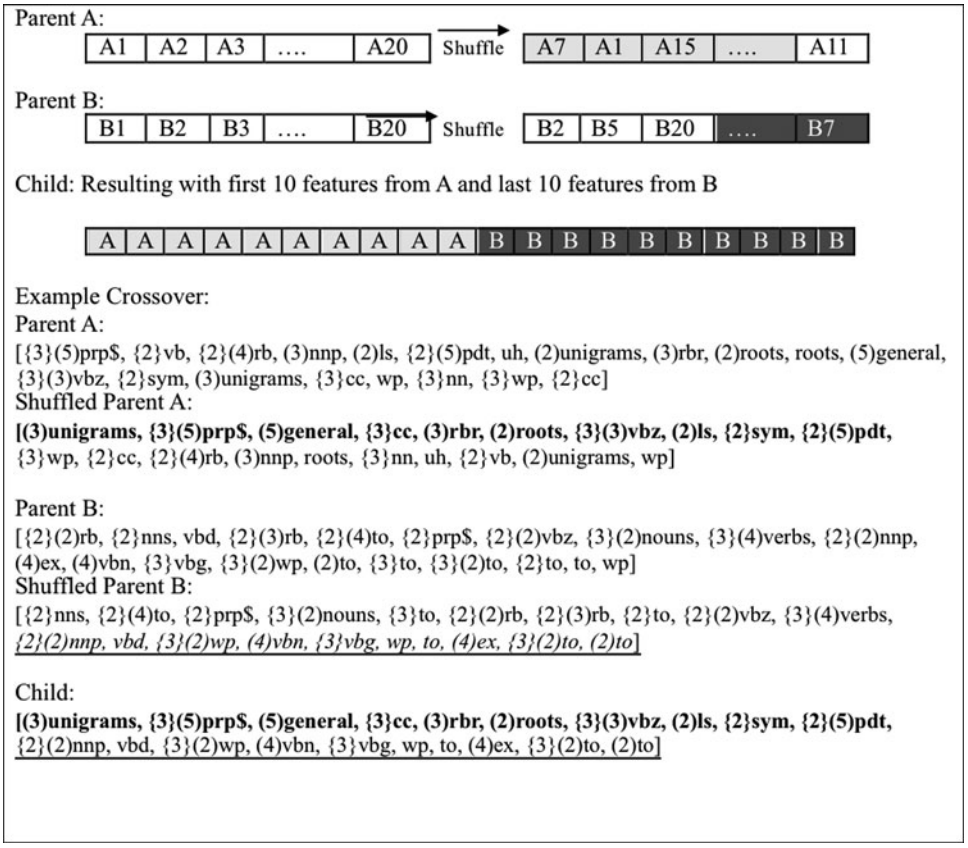


Fig. 6. Crossover example.

duplicate categories removed. If duplicate features are removed, we randomly add features from the set of all features until the pre-determined chromosome length has been reached. We use this method because the feature order is not relevant (Dorigo and Stützle, 2004). Also, if the parents contain too many duplicate features, that indicates that the solutions are converging and no longer evolving, so adding randomly selected features introduces more variation to continue searching the feature space. The mutation was performed by first determining if mutation should occur (with a mutation rate of 10%) and then randomly choosing one feature in the chromosome to be replaced with a randomly selected feature not already in the chromosome.

Figure 7 shows the diagram of the GA feature subset selection algorithm run for the experiments. For our experiments, we used 40 iterations (I), a population size of 100 (P), a feature subset size of 20 (L), a mutation rate of 10%, and a retention rate of 20%.

Results

For this paper, we performed experiment trials on different rationale categories with the two feature selection algorithms. The GA and ACO introduce randomness into the process, so different trials resulted in different classification results. The results reported are the best of the trials for each experiment, based on predictive ability assessed by classifying hold-out test data (Kohavi and John, 1997). The standard deviation of the trial results is also given.

We ran baseline experiments using *all* feature categories, so we could compare those results with those using feature selection. We calculated the inter-annotator agreement by comparing annotations of two researchers. This measure represents an estimate of F-measure achievable manually. Any overlapping annotations were considered a match (Rogers *et al.*, 2014). We used the F-measure instead of the Cohen's kappa because it was better suited for analyzing text with overlapping annotations (Hripcsak and Rothschild, 2005).

https://doi.org/10.1017/S0890060420000037 Published online by Cambridge University Press

Each experiment outputs the best feature subset at each iteration along with their F-measure for training and test. Training scores are the result of 10-fold cross-validation over the training sentences with the Naïve Bayes classifier, using the subset of features generated at the final iteration of feature selection. Validation scores are based on training a Naïve Bayes classifier with the training sentences, and evaluating this classifier over all the test sentences – still using just the subset of features chosen at the final iteration of the feature selection algorithm.

Dataset 1: Chrome bug reports

Binary rationale and alternatives had higher F-1 measure validation scores from the model generated with the ACO than with the GA. All the rest had higher F-1 measure validation scores from the model generated with GA than with ACO. Every rationale subclass had higher validation scores from the model generated with feature selection than with using all features. In the tables below, the highest validation score is bolded, to indicate the best model created. For the features identified by the ACO, the largest difference between the training and validation results was 0.098, with an average difference of 0.052. For the features identified by the GA, the largest difference was 0.215 (for arguments-all), with an average difference of 0.079. Using all features had the largest difference of 0.099 and an average difference of 0.049 (Tables 1–5).

Dataset 2: studying professional software designers

Binary rationale, the argumentation subset, and alternatives had a higher validation F-measure from the model generated with the ACO than with the GA. The rest had a higher validation F-measure from the model generated with the GA than with the ACO. For binary rationale, alternatives, and decisions, using all features gave a better validation F-measure than either the GA or ACO. In the tables below, the highest validation score is bolded, to indicate the most representative model created. For the features identified by the ACO, the largest difference between the training and validation results was 0.258, with an average difference of 0.163. For the features identified by the GA, the largest difference was 0.224 (for arguments-all), with an average difference of 0.158. Using all features had the largest difference of 0.091 and an average difference of 0.018 (Tables 6–10).

Conclusions and future work

The results given above evaluated the rationale identified using the machine learning models against the annotated documents used as test and training data. We would like to add another level of verification by looking at correctly and incorrectly classified sentences manually. This is still in progress since it requires re-building the documents (pre-processing converts them into collections of features, so they are no longer human-readable) and importing them into a tool that can be used by humans.

The results presented here show that rationale identification is a challenging task, even for humans. In some cases, our models did better than the humans, as shown by the examples where it did better than the inter-annotator agreement.

In our study, we answered two research questions:

1. Does using evolutionary algorithms for feature selection improve performance when identifying rationale compared to building classifiers without feature selection?

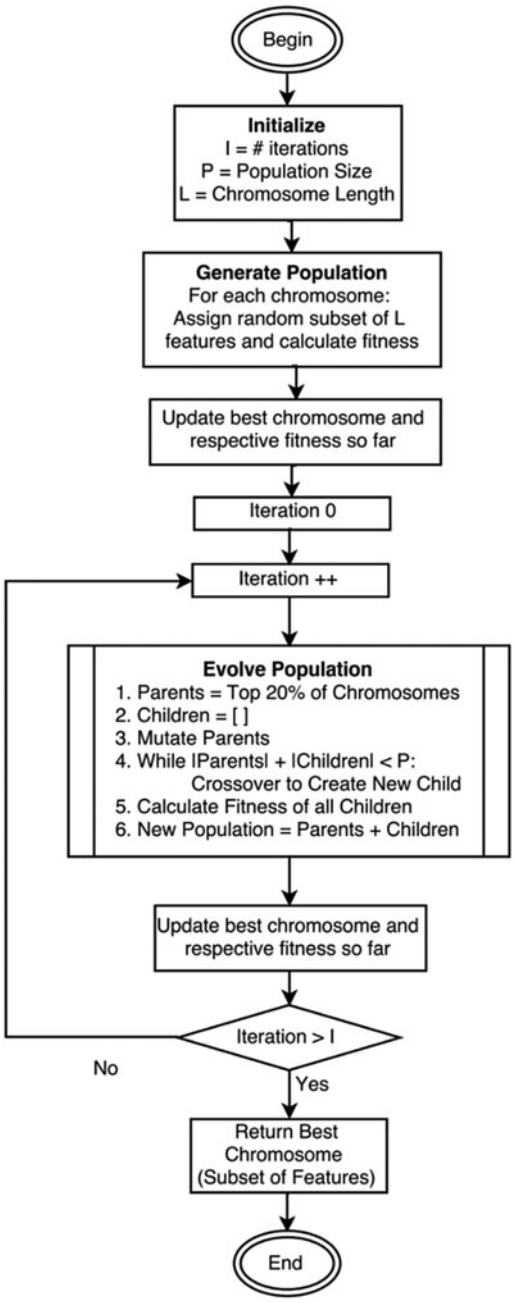


Fig. 7. Genetic algorithm process.

For the larger bug report dataset, the validation F-measures were higher with feature selection than without. Results were less clear for the smaller SPSD dataset. Thus, the hypothesis that there are irrelevant, redundant features that hurt classification is supported for the bug report data but not necessarily for the smaller dataset. Further, we achieved a better F-measure for most inter-annotator agreement F-measures. The exceptions were BR-Decision, BR-Alternative, and SPSD-Decision. For BR-Arguments-all, the ACO was within 0.001 of the AI. This suggests that automatic identification may be as effective as manual identification for the broader categories of rationale.

We noticed that in some experiments, the validation score was higher than the training score. This may be because of

https://doi.org/10.1017/S0890060420000037 Published online by Cambridge University Press

Table 1. Binary rationale (all rationale subclasses)

	Precision	Recall	F-measure	Standard deviation
ACO				
Training	0.943	0.712	0.808	0.00068
Validation	0.950	0.065	**0.772**	0.000513
GA				
Training	0.939	0.723	0.813	0.00079
Validation	0.656	0.936	0.771	0.00095
All features				
Training	0.979	0.635	0.766	
Validation	0.992	0.578	0.730	
Inter-annotator agreement			0.752	

The percentage of sentences with this classification: 17.4%.

Table 2. Argumentation subset (alternative, argument, assumption, decision, and requirement)

	Precision	Recall	F-measure	Standard deviation
ACO				
Training	0.828	0.494	0.614	0.00237
Validation	0.834	0.381	0.523	0.00909
GA				
Training	0.851	0.507	0.631	0.00445
Validation	0.859	0.387	**0.533**	0.01090
All features				
Training	0.981	0.375	0.540	
Validation	0.990	0.291	0.450	
Inter-annotator agreement			0.524	

The percentage of sentences with this classification: 9.5%.

Table 3. Arguments-all (requirement, argument, and assumption)

	Precision	Recall	F-measure	Standard deviation
ACO				
Training	0.750	0.366	0.484	0.00163
Validation	0.788	0.255	**0.386**	0.00449
GA				
Training	0.749	0.378	0.497	0.00082
Validation	0.742	0.250	0.282	0.00590
All features				
Training	0.981	0.212	0.346	
Validation	0.992	0.141	0.247	
Inter-annotator agreement			0.261	

The percentage of sentences with this classification: 5%.

https://doi.org/10.1017/S0890060420000037 Published online by Cambridge University Press

Table 4. Alternative

	Precision	Recall	F-measure	Standard deviation
ACO				
Training	0.572	0.205	0.296	0.00113
Validation	0.661	0.181	0.285	0.01018
GA				
Training	0.447	0.242	0.304	0.00122
Validation	0.683	0.176	**0.306**	0.01126
All features				
Training	0.969	0.090	0.164	
Validation	0.993	0.075	0.139	
Inter-annotator agreement			0.332	

The percentage of sentences with this classification: 2.7%.

Table 5. Decision

	Precision	Recall	F-measure	Standard deviation
ACO				
Training	0.465	0.338	0.374	0.00064
Validation	0.479	0.341	0.399	0.00531
GA				
Training	0.462	0.374	0.397	0.00074
Validation	0.479	0.352	**0.406**	0.00289
All features				
Training	0.991	0.083	0.153	
Validation	0.993	0.865	0.159	
Inter-annotator agreement			0.554	

The percentage of sentences with this classification: 2.5%.

Table 6. Binary rationale (all rationale subclasses)

	Precision	Recall	F-measure	Standard deviation
ACO				
Training	0.947	0.679	0.785	0.00036
Validation	0.970	0.539	0.692	0.00405
GA				
Training	0.951	0.697	0.799	0.00210
Validation	0.932	0.540	0.684	0.00722
All features				
Training	0.797	0.608	0.682	
Validation	0.875	0.587	**0.703**	
Inter-annotator agreement			0.622	

The percentage of sentences with this classification: 53.5%.

https://doi.org/10.1017/S0890060420000037 Published online by Cambridge University Press

Table 7. Argumentation subset (alternative, argument, assumption, decision, and requirement)

	Precision	Recall	F-measure	Standard deviation
ACO				
Training	0.951	0.665	0.778	0.00179
Validation	0.926	0.517	**0.664**	0.00932
GA				
Training	0.947	0.687	0.791	0.00183
Validation	0.909	0.512	0.655	0.06581
All features				
Training	0.831	0.591	0.682	
Validation	0.872	0.531	0.660	
Inter-annotator agreement			0.606	

The percentage of sentences with this classification: 50.8%.

Table 8. Arguments-all (requirement, argument, and assumption)

	Precision	Recall	F-measure	Standard deviation
ACO				
Training	0.485	0.533	0.469	0.02320
Validation	0.386	0.145	0.211	0.02257
GA				
Training	0.563	0.427	0.461	0.00833
Validation	0.523	0.153	**0.237**	0.01726
All features				
Training	0.627	0.201	0.297	
Validation	0.727	0.119	0.200	
Inter-annotator agreement			0.205	

The percentage of sentences with this classification: 11.2%.

Table 9. Alternative

	Precision	Recall	F-measure	Standard deviation
ACO				
Training	0.908	0.502	0.639	0.00364
Validation	0.787	0.342	0.477	0.02893
GA				
Training	0.869	0.555	0.669	0.00621
Validation	0.651	0.268	0.470	0.00684
All features				
Training	0.921	0.411	0.562	
Validation	0.863	0.334	**0.482**	
Inter-annotator agreement			0.391	

The percentage of sentences with this classification: 32.1%.

the number of sentences used to train each model. The training models use 10-fold cross-validation of training sentences, meaning each model is trained with 9/10 of 70% of the sentences – 63% of the sentences. The models used to evaluate the test scores are trained with all of the training sentences – or 70% of the sentences. This may be why the test scores sometimes surpass the training scores since the model learns over more total instances. This is one possible explanation, but it

https://doi.org/10.1017/S0890060420000037 Published online by Cambridge University Press

Table 10. Decision

	Precision	Recall	F-measure	Standard deviation
ACO				
Training	0.461	0.288	0.342	0.00676
Validation	0.277	0.220	0.245	0.02843
GA				
Training	0.549	0.288	0.368	0.00050
Validation	0.308	0.228	0.261	0.02367
All features				
Training	0.847	0.112	0.195	
Validation	0.892	0.155	**0.264**	
Inter-annotator agreement			0.262	

The percentage of sentences with this classification: 9.8%.

is also possible that some of the faults lies in the random split of the training and test sentences.

2. How does the classification performance of the ACO selected feature set compare to the GA selected feature set?

In "Dataset 1: Chrome bug reports" and "Dataset 2: studying professional software designers" sections, we compared the results of the ACO and GA experiments for both datasets. For the BR dataset: binary rationale and arguments-all had higher validation F-measures from the model generated with the ACO than with the GA. For the SPSD dataset: the argumentation subset and alternatives had higher validation F-measure from the model generated with the ACO than with the GA. Thus, for 6 out of 10 different experiments, the GA outperformed the ACO. The results from the ACO and GA were generally very close. For the BR dataset, the average difference in validation F-measure was 0.035. For the SPSD dataset, the average difference in validation F-measure was 0.014.

Since the ACO performed comparably to the GA, and took half the time on average to run on the large BR dataset, the ACO may be preferable to the GA if training time is a concern. For the larger bug report dataset, the F-measure with feature selection was higher than without feature selection; however, this was only the case with some of the rationale categories for the smaller SPSD dataset.

As mentioned above, we are currently in the process of studying the misclassified sentences to see if there might be features in common that we should be adding to our feature sets that might help us improve performance. We are examining both the false negatives and false positives. Another way to evaluate our work would be to have domain experts look at the classified rationale to see if what we did capture, although not complete, might still be sufficient to help a developer. For the broader rationale categories, the precision was better than the recall – is some rationale better than none? Other interesting experiments that could be performed include looking at how features interact and a sensitivity analysis to determine if some features are more important than others.

A potential future extension of this work is integrating automatic design rationale capture algorithms into a tool for software engineers. We are working on importing the classification results into a rationale structuring and presentation tool, C_SEURAT (Collaborative Software Engineering Using Rationale) and will use this to study if partially captured rationale is easier to work with than starting with documents where no rationale is indicated.

Acknowledgements. We thank Miami University graduate students Michelle Flowers, John Malloy, Tanmay Mathur, and Benjamin Rogers and undergraduate students James Gung, and Yechen Qiao for their assistance in annotating the data used in these experiments and Wesleyan student Connor Justice for creating the sentences.csv file. Miriam Lester was supported by a fellowship from the American Association of University Women (AAUW). The SPSD data was produced in design sessions funded by the National Science Foundation (NSF) (award CCF-0845840). We thank the workshop organizers, André van der Hoek, Marian Petre, and Alex Baker for granting access to the transcripts. The data annotation work was supported by NSF CAREER Award CCF-0844638 (Burge). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

References

Aghdam MH, Ghasem-Aghaee N and Basiri ME (2009) Text feature selection using ant colony optimization. *Expert Systems with Applications* **36**, 6843–6853.

Al-Ani A (2005) Feature subset selection using ant colony optimization. *International Journal of Computational Intelligence* **2**, 53–58.

Ali SI and Shahzad W (2012) A feature subset selection method based on symmetric uncertainty and ant colony optimization. *International Conference on Emerging Technologies*. New York, NY: IEEE, pp. 1–6.

Alkadhi R, Nonnenmacher M, Guzman E and Bruegge B (2018) How do developers discuss rationale? *International Conference on Software Analysis, Evolution and Reengineering*, Campobasso, Italy, March 20–23, pp. 357–369.

Anderson EJ and Ferris MC (1994) Genetic algorithms for combinatorial optimization: the assemble line balancing problem. *ORSA Journal on Computing* **6**, 161–173.

Atkinson-Abutridy J, Mellish C and Aitken S (2004) Combining information extraction with genetic algorithms for text mining. *IEEE Intelligent Systems* **19**, 22–30.

Basiri ME and Nemati S (2009) A novel hybrid ACO-GA algorithm for text feature selection. *IEEE Congress on Evolutionary Computation*. New York, NY: IEEE, pp. 2561–2568.

Brazier FMT, van Langen PHG and Treur J (1997) A compositional approach to modelling design rationale. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **11**, 125–139.

Brin S and Page L (1998) The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems* **30**, 107–117.

https://doi.org/10.1017/S0890060420000037 Published online by Cambridge University Press

Burge JE (2005) *Software Engineering Using Design RATionale* (Dissertation). Worcester, Massachusetts: Worcester Polytechnic Institute.

Burge J and Bracewell R (2008) Special issue: design rationale. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **22**, 309–310.

Burge JE and Brown DC (2003) Rationale support for maintenance of large scale systems. *Workshop on Evolution of Large-Scale Industrial Software Applications (ELISA)*, ICSM'03, Amsterdam, Netherlands.

Burge JE and Brown DC (2004) An integrated approach for software design checking using design rationale. In Gero JS (ed), *Design Computing and Cognition '04*. Dordrecht: Springer Netherlands, pp. 557–575.

Burge J, Carroll JM, McCall R and Mistrik I (2008) *Rationale-Based Software Engineering*. Heidelberg: Springer.

Casillas J, Cordón O and Herrera F (2000) Learning fuzzy rules using ant colony optimization algorithms. *Proc. 2nd International Workshop on Ant Algorithms*, Brussels, Belgium, September 8–9, pp. 13–21.

Chung PWH and Bañares-Alcántara R (Guest Eds) (1997) Special issue on representation and use of design rationale. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **11**, 89–167.

Cunningham H, Maynard D and Bontcheva K (2011) *Text processing with GATE (Version 6)*. University of Sheffield Department of Computer Science. 15 April 2011. ISBN 0956599311.

de Campos LM, Fernández-Luna JM, Gámez JA and Puerta JM (2002) Ant colony optimization for learning Bayesian networks. *International Journal of Approximate Reasoning* **31**, 291–311.

de Medeiros AP and Schwabe D (2008) Kuaba approach: integrating formal semantics and design rationale representation to support design reuse. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **22**, 399–419.

Deneubourg J-L, Aron S, Goss S and Pasteels JM (1990) The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior* **3**, 159–168.

Dorigo M (1992) *Optimization, Learning and Natural Algorithms* (PhD Thesis). Politecnico di Milano, Italy.

Dorigo M and Stützle T (1999) ACO algorithms for the traveling salesman problem. In *Evolutionary Algorithms in Engineering and Computer Science: Recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming, Genetic Programming and Industrial Applications*. Chichester: John Wiley & Sons.

Dorigo M and Stützle T (2004) *Ant Colony Optimization*. Cambridge, MA: MIT Press.

Dorigo M and Stützle T (2010) Ant colony optimization: overview and recent advances. In Gendreau M and Yves Potvin J (eds), *Handbook of Metaheuristics*. Boston, MA: Springer, pp. 227–263.

Dorigo M, Maniezzo V and Colorni A (1996) Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* **26**, 29–41.

Dutoit A, McCall R, Mistrik I and Paech B (2006) *Rationale Management in Software Engineering*. Berlin: Springer-Verlag.

Goldberg D (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: AddisonWesley.

Holland JH (1975/1992) *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Cambridge, MA: MIT Press.

Hripcsak G and Rothschild A (2005) Agreement, the F-Measure, and reliability in information retrieval. *Journal of the American Medical Informatics Association* **12**, 296–298.

Jiang P-p, Liu P, Zhu Z and Zhao L (2009) Optimization of text feature subsets based on GATS algorithm. *IEEE International Symposium on IT in Medicine & Education*. New York, NY: IEEE, pp. 924–927.

Kanan HR, Faez K and Hosseinzadeh M (2007) Face recognition system using ant colony optimization-based selected features. *IEEE Symposium on Computational Intelligence in Security and Defense Applications*, Honolulu, HI, pp. 57–62.

King JMP and Bañares-Alcántara R (1997) Extending the scope and use of design rationale records. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **11**, 155–167.

Kohavi R and John GH (1997) Wrappers for feature subset selection. *Artificial Intelligence* **97**, 273–324.

Kurtanovic Z and Maalej W (2018) On user rationale in software engineering. *Requirements Engineering* **23**, 357–379.

Lee J (1997) Design rationale systems: understanding the issues. *IEEE Expert* **12**, 78–85.

Leguizamon G and Michalewicz Z (1999) A new version of ant system for subset problems. *Proceedings of the 1999 Congress on Evolutionary Computation*, Vol. **2**. IEEE, pp. 1459–1464.

Lessing L, Dumitrescu I and Stützle T (2004) A comparison between ACO algorithms for the set covering problem. *International Workshop on Ant Colony Optimization and Swarm Intelligence*. Berlin, Heidelberg: Springer.

Lester M and Burge J (2018) Identifying design rationale using ant colony optimization. *Proc. of the International Conference on Design, Computing, and Cognition*, Lake Cuomo, Italy, pp. 581–600.

Liang Y, Liu Y, Kwong C and Lee W (2012) Learning the 'Whys': discovering design rationale using text mining – an algorithm perspective. *Computer-Aided Design* **44**, 916–930.

López C, Codocedo V, Astudillo H and Cysneiros LM (2012) Bridging the gap between software architecture rationale formalisms and actual architecture documents: an ontology-driven approach. *Science of Computer Programming* **77**, 66–80.

Mao F, Mercer RE and Xiao L (2014) Extracting Imperatives from Wikipedia Article for Deletion Discussions. *Proceedings of the first workshop on Argumentation Mining*, Baltimore, Maryland, ACL, p. 106.

Marcus MP, Marcinkiewicz MA and Santorini B (1993) Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics* **19**, 313–330.

Martin-Bautista MJ and Vila M-A (1999) A survey of genetic feature selection in mining issues. *Proceedings of the 1999 Congress on Evolutionary Computation, CEC 99*, Vol. **2**. New York, NY: IEEE, pp. 1314–1321.

Mathur T (2015) Improving Classification Results Using Class Imbalance Solutions & Evaluating the Generalizability of Rationale Extraction Techniques (Master's Thesis). Miami University.

McCall R (1991) PHI: a conceptual foundation for design hypermedia. *Design Studies* **12**, 30–41.

McCall R (2018) Using argumentative, semantic grammar for capture of design rationale. *Proceedings of the International Conference on Design Computing and Cognition '18*, Lecco Italy, 2–4 July 2018, pp. 571–580.

Moens M-F, Boiy E, Palau RM and Reed C (2007) Automatic detection of arguments in legal texts. *Proceedings of the 11th International Conference on Artificial Intelligence and Law*. New York, NY: ACM, pp. 98–107.

Moran T and Carroll J (Eds) (1996) *Design Rationale Concepts, Techniques, and Use*. Mahwah, NJ: Lawrence Erlbaum Associates.

Mukherjee I, AL-Fayoumi M, Mahanti PK, Jha R and Al-Bidewi I (2010) Content analysis based on text mining using genetic algorithm. *2nd International Conference on Computer Technology and Development*. New York, NY: IEEE, pp. 432–436.

Ozyurt IB (2012) Automatic identification and classification of noun argument structures in biomedical literature. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* **9**, 1639–1648.

Palau M and Moens M (2009) Argumentation mining: the detection, classification and structure of arguments in text. *Proc. of the 12th International Conference on Artificial Intelligence and Law (ICAIL '09)*,Barcelona, Spain, June 8–12, 2009, pp. 98–107.

Prakken H, Reed C and Walton D (2003) Argumentation schemes and generalisations in reasoning about evidence. In *Proceedings of the 9th international conference on Artificial intelligence and law (ICAIL '03). Association for Computing Machinery*, New York, NY, USA, pp. 32–41.

Roeva O, Fidanova S and Atanassova V (2013) Hybrid ACO-GA for parameter identification of an *E. coli* cultivation process model. *International Conference on Large-Scale Scientific Computing*. Berlin, Heidelberg: Springer.

Rogers B, Gung J, Qiao Y and Burge J (2012) Exploring techniques for rationale extraction from existing documents. *New Ideas and Emerging Results Track, International Conference on Software Engineering*, Zurich, Switzerland, June 2012.

Rogers B, Qiao Y, Gung J, Mathur T and Burge J (2014) Using text mining techniques to extract rationale from existing documentation. *International Conference on Design Computing and Cognition,* London, UK 23–25 June, pp. 457–474.

https://doi.org/10.1017/S0890060420000037 Published online by Cambridge University Press

Rogers B, Justice C, Mathur T and Burge JE (2016) Generalizability of document features for identifying rationale. *Design Computing and Cognition '16*. Berlin, Heidelberg: Springer, pp. 633–651.

Saraç E and Özel SA (2014) An ant colony optimization based feature selection for web page classification. *The Scientific World Journal* **2014**, 16.

Shipman F and McCall R (1994) Supporting knowledge-base evolution with incremental formalization. *Proc. CHI'94,* Boston, Massachusetts, April 24–28, pp. 285–291.

Shipman F and McCall R (1997) Integrating different perspectives on design rationale: Supporting the emergence of design rationale from design communication. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **11**, 141–154.

Skiena S (2017) *The Data Science Design Manual*. Berlin, Heidelberg: Springer.

Solnon C and Fenet S (2006) A study of ACO capabilities for solving the maximum clique problem. *Journal of Heuristics* **12**, 155–180.

Zaiyadi M and Baharudin B (2010) A proposed hybrid approach for feature selection in text document categorization. *World Academy of Science, Engineering and Technology* **48**, 111–116.

Miriam Lester is currently working as a Software Engineer at YouTube. She received her MA in Computer Science at Wesleyan University, with support from the AAUW Selected Professions Fellowship. Her research focused on machine learning and natural language processing.

Miguel Guerrero is an undergraduate student at Colorado College. Miguel's interests include the role of software in ideation, turning natural language into formal logic, algorithmic art, and game design.

Janet E. Burge is an Associate Professor in the Colorado College Department of Mathematics and Computer Science. Dr. Burge's major research interests are in Software Engineering and Artificial Intelligence. Her primary research area is in Design Rationale, with a focus on Design Rationale for Software Maintenance. Dr. Burge is a co-author of the book *Rationale-Based Software Engineering*. She has worked in industry as a researcher and software developer for over 20 years. She received her PhD and MS in Computer Science from WPI, and her BS in Computer Science from Michigan Technological University.

https://doi.org/10.1017/S0890060420000037 Published online by Cambridge University Press