Natural Language Processing to Quantify Security Effort in the Software Development Lifecycle

Constantine Aaron Cois
Carnegie Mellon University (CMU)
Software Engineering Institute (SEI)
Pittsburgh, PA, USA
cacois@andrew.cmu.edu

Rick Kazman
University of Hawaii and
SEI/CMU
Honolulu, HI, USA
kazman@hawaii.edu

***Abstract*—Addressing security in the software development lifecycle is an ever-present concern for software engineers and organizations. From a management and monitoring perspective, it is difficult to measure 1) the amount of effort being focused on security concerns during active development and 2) the success of security related design and development efforts. Such data is simply not recorded. If reliable measurements were available, software project leaders would have a powerful tool to assess risk and inform decision making. This would enable managers to direct development and testing to assure a desired level of security in their software products, to protect both their organizations and customers. To fill this need and provide such data, we propose a technique for performing topic detection on data commonly available in most software development projects: text artifacts from issue tracking and version control systems. We apply machine learning and natural language processing techniques to create classifiers capable of accurately detecting whether a given text snippet is related to the topic of security. Realization of such a capability will give software teams the ability to analyze current and past levels of security effort, revealing immediate project focus and the long-term impacts of security tasking. We validate our approach via experiments on data from the large-scale open source Chromium software project. Our results show that a Naïve Bayes classification scheme using an n-gram feature-space is an appropriate and effective approach to automated topic detection of software security text snippets, and that effective training data can be derived from public data sources without the need for manual intervention.**

Keywords-natural language processing; machine learning; software security; security; topic defection; classification; naïve bayes

I. INTRODUCTION

Adequately injecting security into the software development lifecycle (SDLC) to ensure the creation of secure systems is a growing concern. Recent high profile security breaches in major industry and government organizations [1, 2, 3] have shown the unsettling vulnerability of modern software systems, even those developed by mature technology firms with experienced software engineering teams. Industry has taken note, and is responding with initiatives such as Microsoft's Security Development Lifecycle (SDL), a framework for formalizing and monitoring regular security effort throughout the SDLC [4]. In the open source community, equally prolific security holes have recently been exposed, including the high profile Heartbleed [5, 6] and Shellshock [7] vulnerabilities, which contribute to a broad threat faced by many technology sectors and industries.

It is widely accepted that to achieve a high level of quality with regards to security in a software application, security as a quality attribute must be consistently addressed through all phases of the SDLC [4, 8, 9]. Techniques such as threat modeling, attack surface reduction, and penetration testing have been developed and put into practice in an attempt to meet this condition of secure software development [4, 10]. However, these techniques represent the injection of isolated activities at specific phases of the SDLC, not an overarching, consistent amount of concern for and effort towards security throughout software development. The goal of consistent security thought and effort in software development is hampered by the lack of robust means of identifying and measuring security effort within a software project or team.

Despite the efforts to regularize the reporting of security bugs (e.g., the CVE classification [11]), software development teams rarely record security bugs, tasks, or effort specifically. For this reason, accurate measurement of security effort is virtually impossible. In fact, our analysis of over 400,000 project repositories hosted on Github, a popular open source project management system, showed that only 1.4% of projects using a labeling system for tasks made available to developers a label for security related issues. The ability to identify and measure, at any point in time, the amount and type of effort being dedicated to security would give software developers and project managers powerful new capabilities. For example, they could plan and track the levels of effort expended towards software security throughout the SDLC. Additionally, such capability would allow teams to recognize early when projects are at risk of lowering security quality through inattention, thus avoiding the unintentional injection of vulnerabilities into their software product. Further, if applied continuously (or even in hindsight), such a quantitative capability would allow architects and project managers to identify points of introduction of design flaws, process failures, architectural inconsistencies, or weakness in security process enforcement, enabling isolation of areas of code developed during these periods for more rigorous testing and maintenance. This data would also allow the project to make informed decisions about when the technical debt of

Sponsored by the U.S. Department of Homeland Security Science and Technology Directorate

such flaws had accumulated sufficiently that it was economically advantageous to refactor the affected portions of the code base.

For example, in other work we have shown how certain types of design flaws are consistently highly correlated with high bug and change rates [12, 13]. If these design flaws are not fixed, no amount of bug-finding and bug-fixing effort will result in higher quality software. We are interested to know whether the same patterns hold specifically for security bugs. If this were true then, armed with this information, a project manager could focus project resources on refactoring, to remove the design flaws, and hence systematically increase system security. But such an analysis is almost impossible today: most projects do not indicate which bugs in their issue-tracking system or which commits in their Version Control System (VCS) are security related. We simply lack the raw information to perform this analysis. To address this shortcoming we have focused on filling this gap—that is, providing the missing information—in software development project situational awareness. In this way we can provide insight to software architects and project managers as to the ongoing security efforts within their projects based on a data-driven assessment of their project repositories.

To achieve this goal, two common sources of incontrovertible data were identified for use: 1) text from tasks entered into trackers and 2) text messages accompanying source code commits to a VCS. It is expected that most organized software development teams use both issue trackers and some form of VCS that provides the opportunity for commit messages, and as such that these data sources will be readily available in the vast majority of operational contexts. To quantify security effort using these sources of data, natural language processing (NLP) methods were applied to derive feature sets from the unstructured text data and machine learning classification methods were applied to determine whether each text snippet was likely to represent a security-related topic. While others have attempted to use NLP techniques to detect specific security information within full text documents [14], our approach represents a generalized topic detection scheme with broad potential utility in informing software development processes.

II. BACKGROUND

NLP techniques have been used by researchers in a number of ways to analyze artifacts of the software development process, often focused on commit messages or defect reports [15, 16, 17]. Other researchers have noted the challenges and potential rewards of mining process and requirements data from software project artifacts [18, 19]. Our work builds on this research by attempting to create a generalized classifier capable of identifying security-focused text artifacts from data residing in standard software development tools.

The classification method that we have been exploring, Naïve Bayes, simplifies statistical learning processes by adopting an assumption of independence between features of a given classification. Though this assumption can be questioned for many applications, in practice Naïve Bayes competes well against more sophisticated techniques, and is therefore a common starting point for exploration of a new problem space [20].

III. DATA

To train classifiers to identify software security-focused text snippets, reliable gold-standard data was required. The most accessible source of large quantities of such training data comes in the form of issue tracker items from open source projects in which contributors specifically label security-related issues with a "security" tag. The Chromium projects [21] hosted on the Google Code platform were identified as one such source. The issue-tracking repository for these projects included 875 issues labeled as "security", dating from 3/10/13 to 1/9/15. The summary statements of these issues were extracted as positive examples of software security-related text snippets, while the summary statements of additional Chromium issues not tagged as "security" were used as negative examples. The full data set used in this study contained 1874 text samples (875 security related, 999 not security related). Table I shows examples of both positive and negative text snippets used for training.

Data were randomized and divided into training and testing sets for performance validation, discussed in detail in the next section.

IV. EXPERIMENTAL METHODS

This initial study applied Naïve Bayes classification to the problem of determining whether snippets of text are related to software security. To perform feature selection and classification, the Python programming language and the associated Natural Language Toolkit (NLTK) were used. Methods of Naïve Bayes classifier training, testing, accuracy calculation, and confusion matrix generation used were all unmodified NLTK implementations [22].

TABLE I. TRAINING DATA SAMPLES

Chromium Project Training Data	
Summary Message (text snippet)	***Correct Classification***
Clicking "Safe Browsing diagnostic page" link broken on malware interstitial	security
Block chrome-extension:// pages from importing script over non-HTTPS connections	security
Security: XSS issue in the FTP parser	security
Heap-use-after-free in WebCore::RenderLayer::repaintBlockSelectionGaps	security
Rendering glitch when switching windows	not security
Regression: Default cursor not seen in Sign in page of chrome after navigating back from any other tab.	not security
Status Bar fails to hide	not_security
Separators on column header disappear when display language is RTL.	not_security

A. Feature Extraction

Features for text classification were derived from the presence or absence of n-grams in tokenized text snippets. For example, the text snippet "I am a rock, I am an island" is tokenized into the following set of tokens: ['i', 'am', 'a', 'rock', 'i', 'am', 'an', 'island']. Prior to feature detection, stop words (words too common to indicate any semantic meaning for our classification) were removed. Removing stop words (including 'i', 'am', and 'an', as defined in [23]) from our example yields the following remaining tokens: ['rock', 'island']. These tokens indicate two unigram (or n-gram of size one) features for our text snippet, namely *contains(rock) = True* and *contains(island) = True*. It is also relevant to know if a text snippet does *not* contain certain words, such as *contains(snowblower) = False*, as the presence of the word "snowblower" would likely impact the meaning or topic of our text. N-gram feature analysis often goes beyond unigram features to also include bigram (adjacent word pair) features [e.g., *contains(rock, island)*], trigram (adjacent word triplet) features, etc.

To determine the set of all potential features relevant to the domain of interest (i.e., software security), a training set of summary messages from issues in the open source Chromium project was analyzed, as depicted in Fig. 1.

Text from the summary message of each issue was tokenized and stop words were removed. The remaining tokens were used to generate feature spaces S_1, S_2, and S_3, representing a unigram-only feature space, a (unigram + bigram) feature space, and a (unigram + bigram + trigram) feature space, respectively. These feature spaces were used to extract features from the text for classifier training and testing, as described in the next section.

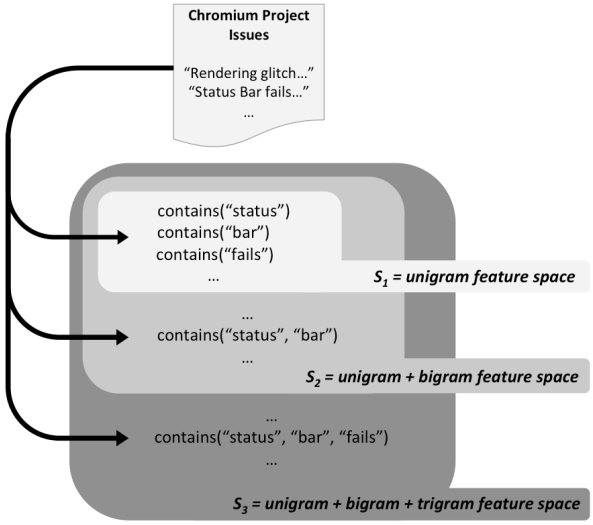

Figure 1: Generation of Experimental Feature Spaces

B. Naïve Bayes Classification

Once the full feature space was determined, text samples in a training set with known classifications could be analyzed to determine their feature vectors. For our text samples, a feature vector is a representation of the presence or absence of all features in the full feature space. Thus, when running an experiment using feature space S_1, the feature vector $V_1(t)$ for a given tokenized text snippet t is an array containing a Boolean value for each token feature represented in S_1, indicating whether the text contained or did not contain the token. See Fig. 2. These feature vectors and their correct classifications were used to train a Naïve Bayes classifier, yielding a statistical model of features and their statistical contributions to the classification of software security related text.

It should be expected that the words comprising highly informative features in models generated from training will be independent of words found by prior studies to be common to text snippets from many areas of software projects, such as those reported in [24]. For example, words like "html", "add", or "feature" would be common to almost any software project text, and thus would not be expected to yield highly informative features to our security classifier after training. Table II presents an example of the most informative features of a model from a single training run, illustrating unigram features and their likelihood ratios in classifying software security related text. True to expectations, the highly informative features discovered have no overlap with high-frequency words common across many software systems [24], lending confidence to the domain-centric training of the classifiers.

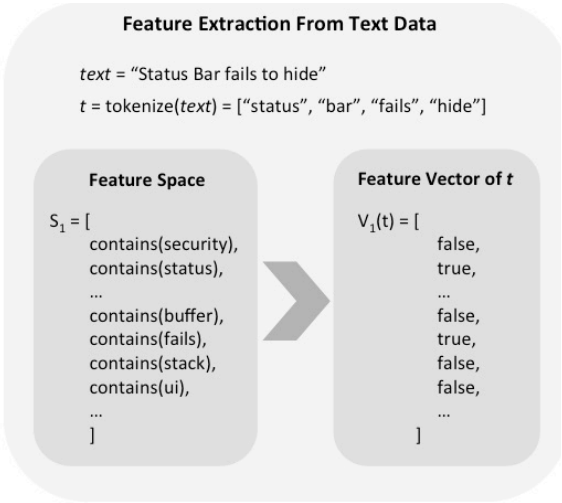

Figure 2. Feature Vector Generation

TABLE II. UNIGRAM CLASSIFIER MODEL EXAMPLE

Informative Features for Classifying Software Security Text	
Feature	***Likelihood Ratio (security : not_security)***
contains(heap) = True	25.3 : 1.0
contains(corruption) = True	18.1 : 1.0
contains(security) = True	17.9 : 1.0
contains(bad) = True	16.2 : 1.0
contains(integer) = True	15.2 : 1.0
contains(overflow) = True	13.9 : 1.0
contains(pointer) = True	12.0 : 1.0
contains(doesn) = True	1.0 : 11.4
contains(stack) = True	11.3 : 1.0
contains(buffer) = True	11.2 : 1.0
contains(seen) = True	1.0 : 10.5
contains(ui) = True	1.0 : 9.4

A number of words highly relevant to the topic of software security appear in the models derived from classifier training. As might be expected, we see that words such as "corruption", "overflow", and "security" are strong indicators that a text snippet should be classified as a software security message. Strong negative indicators also appeared, such as the presence of the word "ui" (user interface) indicating that the message was likely not related to software security. It is worth noting that while the top 15 indicators for each classifier training run were stored, no strong indicators based on the absence of a word were observed. This shows no indication, for example, that any single feature is so common in security-related text that its absence alone conveys strong semantic meaning.

Further experiments expanded the feature space by allowing bigram (two-word) and trigram (three-word) features to be used in classification. Table III shows an example feature model for a classifier derived using a feature space containing unigrams, bigrams, and trigrams.

It can be seen that new strong indicators were introduced by the addition of larger n-grams, including the (buffer, overflow) bigram and (heap, buffer, overflow) trigram. These phrases are consistent with expected terms highly relevant to software security, and provide positive anecdotal evidence for the classifier training data and methodology.

C. Validation

Repeated random sub-sampling validation [25] was performed to validate the approach to text classification. This validation method was chosen to demonstrate the efficacy of training functional classifiers from many possible selected data sets in the hopes of proving the approach robust without specifically selected training data. Repeated random sub-sampling is performed by repeatedly splitting gold standard data into two randomly distributed partitions of pre-defined proportions, training and testing the classifier for each split, and recording all performance results.

TABLE III. UNIGRAM + BIGRAM + TRIGRAM CLASSIFIER MODEL EXAMPLE

Informative Features for Classifying Software Security Text	
Feature	***Likelihood Ratio (security : not_security)***
contains(heap) = True	38.7 : 1.0
contains(overflow) = True	27.3 : 1.0
contains(security) = True	24.9 : 1.0
contains(cast) = True	23.4 : 1.0
contains(pointer) = True	21.8 : 1.0
contains(bad) = True	18.3 : 1.0
contains(buffer, overflow) = True	17.8 : 1.0
contains(doesn) = True	1.0 : 17.7
contains(corruption) = True	17.3 : 1.0
contains(fails) = True	1.0 : 14.3
contains(integer) = True	13.9 : 1.0
contains(buffer) = True	12.6 : 1.0
contains(heap, buffer) = True	12.3 : 1.0
contains(heap, buffer, overflow) = True	12.3 : 1.0

In this study, three experiments were performed, using different feature sets. The first feature set included only unigram (single word) tokens observed in data from the Chromium project. The second experiment used a feature set containing unigrams and bigrams, and the third experiment used a feature set containing unigrams, bigrams, and trigrams. In each experiment, we performed 50 repetitions of random sub-sampling on our aforementioned issue tracker data from the Chromium project. The full experimental data set was distributed into 80/20% training/test distributions, resulting in random training sets containing 1499 samples and random test sets containing 375 samples.

Classifier performance from the experiments can be seen in Table IV.

TABLE IV. RESULTS OF N-GRAM FEATURE STUDIES

Classifier Performance for Various n-gram Feature Spaces			
Feature Space	***Average Precision***	***Average Recall***	***Average F-Measure***
S_1 (Unigrams, 12674 total features)	0.91 ± 0.026	0.89 ± 0.023	0.90 ± 0.019
S_2 (Unigrams + Bigrams, 25347 total features)	0.92 ± 0.022	0.88 ± 0.022	0.90 ± 0.015

Classifier Performance for Various n-gram Feature Spaces			
Feature Space	***Average Precision***	***Average Recall***	***Average F-Measure***
S_3 (Unigrams + Bigrams + Trigrams, 38019 total features)	0.93 ± 0.017	0.88 ± 0.020	0.91 ± 0.016

Overall, the trained classifier performed well at software security topic detection. The average precision (or positive predictive value), which measures the fraction of text snippets classifier as "security" by our classifier that were proven to be correct classifications, was observed between 91% and 93% in our experiments. This data not only represents a promising classifier, but show that the addition of more complex features (bigrams and trigrams) increases performance of a classifier for this domain. Recall (or true positive rate), which measures the fraction of correctly classified security text snippets out of the total number of security text snippets in the data set, was measured at between 88% and 89% in our experiments. Finally, the average f-measure (the harmonic mean of precision and recall) increased from 90% to 91% as more (and more complex) features were added.

V. Conclusions

This paper has presented the results of an initial study investigating the potential of applying NLP and machine learning techniques to extract information from data residing in VCSs and issue tracking systems. The information that we extracted in this study was a classification of issues as security related or not security related. Using this information we can create measures of, and get insights into, software process and software quality.

We have shown here that we can fully automate the process of extracting semantically meaningful information from issue-tracking systems and that this information has both high precision and high recall. Our belief is that the precision and recall are high enough that this technique will open up many possibilities for post-hoc analysis of project repositories and communications, enabling insights that were hitherto impossible, due to the dearth of data. While our goal here was to identify security-related issues, we believe that this technique has the potential to "mine" many other kinds of data from project repositories.

It is expected that more sophisticated feature extraction and classification techniques may further improve on these results. The initial success of this methodology using token-derived features also indicates that the lexicon of security within software development is sufficiently common and consistent that this domain is ripe for the sort of analysis presented here. As training data was extracted, unaltered, from active open source projects with no interaction with developers generating the text, we remain confident that the language used naturally within this domain will yield successful training data from other software development projects in the future.

VI. Future Work and Limitations

We recognize a number of limitations and areas for fruitful expansion of this work, including (1) a broader number of data sets, spanning a wider range of software project types and teams, (2) more sophisticated classification methods such as Support Vector Machines or ensemble methods, and (3) application of these techniques to detect software quality attributes other than security.

It is hoped that versions of the classifiers demonstrated here can be proven effective in classifying not only text snippets from tasks in issue-tracking systems, but also to classify other text snippets common to the software development lifecycle, such as commit messages in version control systems. Validation of this will be explored in future studies.

Additionally, the authors plan to explore the efficacy of this approach in autonomously measuring and monitoring a variety of software quality attributes from data derived from standard software process management and DevOps systems. For example, it would be valuable to monitor when the occurrences of issues related to other quality attributes—such as usability or availability or safety—was spiking.

It is anticipated that the achievement of autonomous quality monitoring can be leveraged into real-time alerting and predictive capabilities suitable for providing expert decision support to software development management, and proactively improving the quality of software developed by organizations employing the envisioned data-driven process optimization techniques.

Acknowledgment

We would like to acknowledge the support of Carol Woody, Bob Ellison, Yuanfang Cai, and Qiong Feng for this research. In addition we gratefully acknowledge the support of the U.S. Department of Homeland Security.

This material is based upon work funded and supported by Department of Homeland Security under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

REFERENCES

[1] Target Puts Data Breach Cost at $148 Million, and Forecasts Profit Drop. *New York Times*. August 2014, retrieved March 2015. http://www.nytimes.com/2014/08/06/business/target-puts-data-breach-costs-at-148-million.html

[2] JPMorgan Hack Exposed Data of 83 Million, Among Biggest Breaches in History. Reuters. October 2014, retrieved March 2015. http://www.reuters.com/article/2014/10/03/us-jpmorgan-cybersecurity-idUSKCN0HR23T20141003

[3] U.S. Postal Service Says It Was Victim of Data Breach. *The Wall Street Journal*. November 2014, retrieved March 2015. http://www.wsj.com/articles/u-s-postal-service-says-it-was-victim-of-data-breach-1415632126

[4] M. Howard and S. Lipner. *The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software*. Microsoft Press, 2006.

[5] The Heartbleed Bug, 2014. http://heartbleed.com/

[6] Zakir Durumeric, James Kasten, David Adrian, J. Alex Halderman, Michael Bailey, Frank Li, Nicolas Weaver, Johanna Amann, Jethro Beekman, Mathias Payer, and Vern Paxson. 2014. The Matter of Heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference* (IMC '14). ACM, New York, NY, USA, 475-488.

[7] MITRE. 2014b. CVE-2014-7169. Common Vulnerabilities and Exposures. November 1, 2014: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-7169

[8] OWASP, Comprehensive, lightweight application security process, http://www.owasp.org, 2006.

[9] G. McGraw, *Software Security: Building Security*, Addison Wesley (2006).

[10] Bart De Win, Riccardo Scandariato, Koen Buyens, Johan Grégoire, Wouter Joosen, On the secure software development process: CLASP, SDL and Touchpoints compared, *Information and Software Technology*, Volume 51, Issue 7, July 2009, Pages 1152-1171.

[11] MITRE, Common Vulnerabilities and Exposures, https://cve.mitre.org/, 2015.

[12] L. Xiao, Y. Cai, and R. Kazman, "Design Rule Spaces: A New Form of Architecture Insight", *Proceedings of the International Conference on Software Engineering (ICSE) 2014*, (Hyderabad, India), June 2014.

[13] R. Mo, Y. Cai, R. Kazman, and L. Xiao, "Hotspot Patterns: The Formal Definition and Automatic Detection of Architecture Smells", *Proceedings of The Working IEEE/IFIP Conference on Software Architecture (WICSA 2015*), (Montreal, Canada), May 2015, in press.

[14] Xusheng Xiao, Amit Paradkar, Suresh Thummalapenta, and Tao Xie. 2012. Automated extraction of security policies from natural-language software documents. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering* (FSE '12).

[15] Abram Hindle, Neil A. Ernst, Michael W. Godfrey, and John Mylopoulos. 2011. Automated topic naming to support cross-project analysis of software maintenance activities. In *Proceedings of the 8th Working Conference on Mining Software Repositories* (MSR '11). ACM, New York, NY, USA.

[16] Runeson, P.; Alexandersson, M.; Nyholm, O., "Detection of Duplicate Defect Reports Using Natural Language Processing,". *ICSE 2007. 29th International Conference on Software Engineering, 2007*, pp.499-510, 20-26 May 2007.

[17] Rubén Prieto-Díaz. 1991. Implementing faceted classification for software reuse. *Communications of the ACM* 34, 5 (May 1991), 88-97.

[18] Poncin, W.; Serebrenik, A.; van den Brand, M., "Process Mining Software Repositories," *15th European Conference on Software Maintenance and Reengineering (CSMR)*, pp.5,14, 1-4 March 2011.

[19] Cleland-Huang, J.; Settimi, R.; Xuchang Zou; Solc, P., "The Detection and Classification of Non-Functional Requirements with Application to Early Aspects," *Requirements Engineering, 14th IEEE International Conference*, pp. 39,48, 11-15, Sept. 2006.

[20] Rish, Irina. "An empirical study of the naive Bayes classifier." *IJCAI 2001 workshop on empirical methods in artificial intelligence*. Vol. 3. No. 22. IBM New York, 2001.

[21] The Chromium Projects, 2015, http://www.chromium.org/

[22] Bird, Steven, Edward Loper and Ewan Klein (2009), *Natural Language Processing with Python*. O'Reilly Media Inc.

[23] Porter, M. F. 1980. "An Algorithm for Suffix Stripping." *Program*, 14(3), 130-37.

[24] Alali, A.; Kagdi, H.; Maletic, J.I., "What's a Typical Commit? A Characterization of Open Source Software Repositories," *International Conference on Program Comprehension (IPCP) 2008,* 10-13 June 2008, pp.182-191.

[25] Ron Kohavi. 1995. "A study of cross-validation and bootstrap for accuracy estimation and model selection." In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2* (IJCAI'95), Vol. 2. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.