Journal of Bioinformatics and Computational Biology
Vol. 5, No. 6 (2007) 1233–1259

Imperial College Press
www.icpress.co.uk

RAPID PATTERN DEVELOPMENT FOR CONCEPT RECOGNITION SYSTEMS: APPLICATION TO POINT MUTATIONS

J. GREGORY CAPORASO*

Department of Biochemistry and Molecular Genetics
Center for Computational Pharmacology
University of Colorado Health Sciences Center
Aurora, CO, USA
gregcaporaso@gmail.com

WILLIAM A. BAUMGARTNER, JR.

Center for Computational Pharmacology
University of Colorado Health Sciences Center
Aurora, CO, USA

DAVID A. RANDOLPH

Department of Computer Science
University of Colorado at Boulder
Boulder, CO, USA

Motorola Mobile Devices, Libertyville, IL, USA

K. BRETONNEL COHEN† and LAWRENCE HUNTER‡

Center for Computational Pharmacology
University of Colorado Health Sciences Center
Aurora, CO, USA

†*Department of Linguistics*
University of Colorado at Boulder
Boulder, CO, USA
‡*larry.hunter@uchsc.edu*

Received 1 May 2007
Revised 30 July 2007
Accepted 23 August 2007

The primary biomedical literature is being generated at an unprecedented rate, and researchers cannot keep abreast of new developments in their fields. Biomedical natural language processing is being developed to address this issue, but building reliable systems often requires many expert-hours. We present an approach for automatically developing collections of regular expressions to drive high-performance concept recognition systems

*Corresponding author.

with minimal human interaction. We applied our approach to develop MutationFinder, a system for automatically extracting mentions of point mutations from the text. MutationFinder achieves performance equivalent to or better than manually developed mutation recognition systems, but the generation of its 759 patterns has required only 5.5 expert-hours. We also discuss the development and evaluation of our recently published high-quality, human-annotated gold standard corpus, which contains 1,515 complete point mutation mentions annotated in 813 abstracts. Both MutationFinder and the complete corpus are publicly available at http://mutationfinder.sourceforge.net/

Keywords: Pattern learning; information extraction; biomedical natural language processing; mutations; corpus construction; text mining; concept recognition.

1. Background

It is frequently noted that biological researchers are unable to keep pace with the rate of publication of the biomedical literature pertaining to their field.[1–4] Text mining and biomedical natural language processing (NLP) are techniques aimed at addressing this issue. Strong evidence for the necessity of text mining and other automated methods of genome and proteome curation assistance comes from recent work by Baumgartner *et al.*,[5] who found that the current rates of biological database curation will not provide full coverage of even just the currently sequenced genomes for the foreseeable future. However, in almost all cases, developing good systems requires many person-hours of manual intervention, in terms of both corpus annotation and system development. Methods that accelerate the development of useful text mining and biomedical language processing systems have the potential to facilitate access to data currently available only as free text in the biomedical literature. To pursue these points, this paper investigates two hypotheses: (1) that automatic methods can be used to learn patterns of sufficient quantity and quality in order to match or outperform a manually built rule set; and (2) that annotation standards and text collections sufficient for the building of high-quality corpora of mutation mentions with good interannotator agreement can be developed.

We have recently published MutationFinder,[6] a high-performance system for extracting mentions of point mutations from the text, and a high-quality corpus containing 1,515 hand-annotated mutation mentions in 813 Medline abstracts. MutationFinder relies on the collection of regular expressions to extract mutation mentions from the text. A unique feature of our system and its development process is that its top-performing collection of regular expressions was generated by an almost fully automated process which required only 5.5 hours of human interaction, outperforming a baseline system which used only manually constructed regular expressions. Our corpus is unique in that it is larger (in terms of number of mutations and abstracts annotated) and more reliable (because of its development around clearly defined annotation guidelines, and evaluation by interannotator agreement) than existing mutation corpora. Corpus annotation required 54 person-hours, outside of system development.

Our recent paper[6] briefly described MutationFinder in comparison to a baseline system. Here, we describe a generalizable methodology for designing

high-performance, rule-based, biomedical concept recognition (CR) systems with minimal human effort. Using MutationFinder as a case study, we demonstrate the utility of our methodology for streamlining CR system development. We additionally present a description of a manually curated corpus containing mutation mentions, and describe the annotation process and guidelines used in its construction and evaluation. MutationFinder and the complete corpus are publicly available at http://mutationfinder.sourceforge.net/

1.1. *Point mutation recognition*

We define "point mutation recognition" as the process of extracting a relationship between three entities from free text: the position where a substitution occurred in a biological sequence (i.e. a nucleic acid or polypeptide sequence), and the wild-type and mutant entities (i.e. bases or residues). We refer to point mutation recognizers as concept recognition systems, and point out that point mutation recognition is similar to both named entity recognition and information extraction. Point mutation mentions in abbreviated formats [e.g. "the A42G mutant"; Table 2(a)] look like data typically targeted by named entity recognizers, while mentions in natural language formats [e.g. "alanine 42 replaced with glycine"; Table 2(c)–2(e)] look like data typically targeted by information extraction (IE) systems. Jackson and Moulinier[7] note that IE systems attempt to fill in forms which describe events involving several entities. All concepts recognized by point mutation recognizers are exactly that: even mutation mentions in the abbreviated formats, although they look very much like named entities, are references to events that relate three entities — a pre-event state (the wild-type entity), a postevent state (the mutant entity), and an event location (the sequence position).

Point mutation recognition systems have been the subject of several recent publications.[6,8–12] MuteXt,[8] MEMA,[9] and Mutation GraB[10] attempt to extract mentions of mutations paired with a specific gene or gene product from input texts. OSIRIS[11] is a web-based information retrieval system for compiling the mutation literature using a concept-driven, mutation-recognition approach. MutationMiner[12] is a system which uses mutation data from the literature to annotate protein structures.

The MuteXt system, when tested against a manually compiled database of GPCR mutations, was reported to achieve a precision of 87.9% and a recall of 49.5%. MEMA, when judged on 100 human-annotated abstracts, was reported to achieve a precision of 99% and a recall of 35% for extracting gene–mutation pairs, and a precision of 98% and a recall of 75% when extracting mutations only (i.e. when no attempt was made to connect them with a specific gene or protein). Mutation GraB, when tested on 295 unseen, manually annotated articles, was reported to achieve a mean precision of 74% and a mean recall of 81%. When extracting mutations alone (and tested on the MEMA corpus), Mutation GraB was reported to achieve a precision of 97.7% and a recall of 77.3%. Because the testing schemes

(i.e. test corpus and computed performance metrics) differ for each system, these values are not directly comparable, except for the comparison between Mutation GraB and MEMA.

1.2. *Automatic pattern generation for concept recognition*

Several previous attempts[13–19] have been made to automatically generate patterns for concept recognition systems. The AutoSlog/AutoSlog-TS systems[13,14] are classic early examples, and the approach of Hovy and his colleagues[15,16] has been quite influential in recent years. These systems differ from ours in that they require either annotated corpora,[13] hand-classified documents,[14] or multiple rounds of iteration and mandatory tuning.[15,16]

Recently, there has also been domain-specific work on pattern learning for biomedical texts.[17–19] These approaches differ from ours in that they typically overgenerate considerably, producing large numbers of patterns whose utility is often underevaluated and questionable.

1.3. *MutationFinder*

MutationFinder[6] focuses on extracting mentions of point mutations without trying to connect them with their gene or protein source. While at first glance the fact that MutationFinder does not attempt to associate mutations with their gene or gene product might appear to be a limitation, we deliberately chose this approach in order to create modular software. Rather than simultaneously attacking three unsolved problems (i.e. extracting mutation mentions, extracting gene/protein names, and associating the appropriate pairs), MutationFinder's design is based on the UNIX philosophy of software development: "Write programs that do one thing and do it well."[20] Our approach gives MutationFinder users the ability to combine the output of MutationFinder with the output of, for example, the currently top-performing protein name identifier (Hakenberg *et al.*,[21] as determined by the BioCreAtIvE 2 evaluation). Rather than being locked into a single problem or approach, MutationFinder users may mix and match independent tools. Source code modifications are not required to, for example, use an alternative protein name recognition technique.

Like the earlier mutation recognition systems,[8–10] MutationFinder applies a set of regular expressions to identify mutation mentions in input texts. Our currently top-performing collection of regular expressions results in a precision of 98.4% and a recall of 81.9% when extracting mutation mentions from completely blind test data. (We cannot make a direct comparison of our performance with those of the earlier systems, but the publication of our gold standard data set and a script for scoring the output of mutation recognition systems make direct comparisons of mutation recognition systems readily performable for the first time.)

Unlike the earlier systems, the 759 regular expressions driving MutationFinder were automatically generated using a generalizable bootstrap approach, which is

the primary subject of this article. This approach minimizes the time investment typically associated with developing high-performance rule-based systems in the form of manual annotation or pattern construction. Only 5.5 hours were required by a domain expert (Caporaso) to refine an initial collection of patterns into the input for an automated regular expression generator.

We also developed a gold standard data set, which is an additional subject of this article, at a cost of approximately 54 person-hours split among three annotators. It is split into development data and blind test data, and is used for optimizing and evaluating MutationFinder. While extremely helpful for system optimization and testing, a human-annotated gold standard data set is not a prerequisite for developing high-performance concept recognizers by our method.

2. Performance Metrics

In our previous work,[6] we used precision, recall, and F-measure on three separate tasks for evaluating mutation recognition systems: Extracted Mentions, Normalized Mutations, and Document Retrieval. Briefly, the Extracted Mentions task requires that all mentions of all mutations be extracted from input text, while the Normalized Mutations task requires that at least one mention of each mutation be extracted from input text, and the Document Retrieval task requires that at least one mention of any mutation be extracted from input text.

3. Algorithm and Application

Our approach for achieving high-performance concept recognition is based on a six-step process for regular expression generation. Like Webclopedia[15] and the Huang *et al.*[18] text alignment technique, we do not rely on annotated training data. Annotated development and test data are useful for system optimization and testing, but are not required for rule set development. The six steps, followed by an in-depth discussion with examples (see Table 2 and Secs. 3.1–3.6), are as follows:

(1) Automatically compile a collection of raw patterns, or patterns likely to represent the information to be extracted. This step aims to achieve high recall while paying little attention to precision.
(2) Refine the collection of raw patterns to eliminate false positives. This step is intended to address the precision ignored in the first step.
(3) Process the patterns to assign terms from a semantic grammar to the entities to be extracted. In the case of mutation extaction, the semantic classes to assign are wild-type residue (`WRES`), mutant residue (`MRES`), and sequence position (`SPOS`). We refer to the semantically annotated patterns resulting from this step as mutation patterns.
(4) Generate regular expressions from the mutation patterns resulting from step 3.
(5) Perform post hoc analyses to optimize system precision. This does not require human-annotated development data. Optionally, optimize system recall on

Table 1. Person-hours required in each system development step. Step numbers correspond with subsection numbers in Sec. 3.

Step	Person-hours
1: Raw pattern compilation	0.0
2: Raw pattern refinement	2.0
3: Mutation pattern generation	2.5
4: Regular expression generation	0.0
5.1: System optimization: precision	1.0
5.2: System optimization: recall (optional)	18.0*
6: System testing (optional)	36.0*
Total required time	5.5

*Steps 5.2 and 6 require time for corpus development; these steps are optional, and therefore are not included in the total time for system development.

development data. General changes to the system can be tested or in-depth error analyses can be performed to inform design and implementation details.

(6) Optionally,[a] test the system on previously unseen test data. It is critical that these data be different from the development data.

Table 1 details the amount of time devoted to each step. Using MutationFinder as a case study, the subsequent sections demonstrate the development of a high-performance rule set for concept recognition using our methodology.

3.1. *Step 1: raw pattern compilation (automatic)*

To bootstrap our system, we began with a simple rule to identify blocks of text that were likely to describe point mutations. These text blocks were then converted into raw patterns. Example input texts and the resulting raw patterns are presented in Table 2 and discussed throughout this section. This step requires no human interaction.

3.1.1. *Bootstrap rule*

The development of the bootstrap rule for automatically generating raw patterns was informed by domain knowledge. The bootstrap rule states that a block of text might describe a point mutation if it

(1) contains at least two mentions of amino acid residues;
(2) mentions at least one positive integer in numeric characters; and
(3) does not span a sentence break.

[a]While optional for system development, this step is, however, required if one wishes to publish performance data in their system.

Table 2. Example output from steps 1–4, illustrating the transition from input text to regular expression. Source PubMed identifiers for rows (f) and (g) are 11210138 and 2198714, respectively; all other examples are hypothetical instances of common patterns. For space, the output of step 4 is presented only for row (a); all regular expressions are available as supplementary material. The regular expression used to match residues is abbreviated with `RES_PATTERN` for clarity, but is available as supplementary material. Line breaks in all texts, patterns, and the regular expression are present for clarity. Patterns eliminated in step 2 contain cross-references to the section describing the triggered rule. `RES` and `POS` signify undifferentiated mentions of possible amino acid residues and sequence positions, respectively; `WRES`, `MRES`, and `SPOS` signify wild-type residue, mutant residue, and sequence positions, respectively, referring to the mutation mentioned. Step numbers correspond with subsection numbers in Sec. 3.

	Step 1.1 result: text containing mentions identified (Sec. 3.1.1)	**Step 1.2 result:** raw patterns generated (Sec. 3.1.2)	**Step 2 result:** raw patterns refined (Sec. 3.2)	**Step 3 result:** mutation patterns generated (Sec. 3.3)
(a)	*"the **Ala64Gly** mutation"*	`RESPOSRES`	`RESPOSRES` *(no change)*	`WRESSPOSMRES`
(b)	*"the **Ala64→Gly**"*	`RESPOS-->RES`	`RESPOS-->RES` *(no change)*	`WRESSPOS-->MRES`
(c)	*"we mutated **Ala64** to **glycine**"*	`RESPOS to RES`	`RESPOS to RES` *(no change)*	`WRESSPOS to MRES`

(d)	*"we mutated **Ala64** to **glycine**, **leucine**, and **valine**"*	`RES-POS to RES,`
		`RES, and RES`
			`RES-POS to RES,`
			`RES, and RES`
			(no change)
				`WRES-SPOS to MRES`
				`WRES-SPOS to RES,`
				`MRES`
				`WRES-SPOS to RES,`
				`RES, and MRES`

(e)	*"**alanine** was mutated to **glycine** at positions **64** and **72**"*	`RES was mutated to`
		`RES at positions`
		`POS and POS`
			`RES was mutated to`
			`RES at positions`
			`POS and POS`
			(no change)
				`WRES was mutated to`
				`MRES at positions`
				`SPOS`
				`WRES was mutated to`
				`MRES at positions`
				`POS and SPOS`

(f)	*"the catalytic residues (**Asp325**, **Glu354**, and **Asp421**) are necessary"*	`RESPOS, RESPOS,` `and RESPOS`	False positive, manually eliminated (Sec. 3.2.2)	*Not applicable*

(g)	*"A **pro**-memoria on the occasion of the **200**th anniversary of **his** death"*	`RES-memoria on the`
		`occasion of the POSth`
		`anniversary of RES`
			False positive, automatically eliminated (one Medline occurrence) (Sec. 3.2.1)	*Not applicable*

(h)	*"the **Ala64Gly** and **Ala72Thr** mutations"*	`RESPOSRES and` `RESPOSRES`	Manually eliminated (two occurrences of row (a) pattern) (Sec. 3.2.3)	*Not applicable*

	Step 4 result: regular expressions generated *(row (a) example only)* (Sec. 3.4)	`(^|[\s\(\[\'"/,\-])(?P<wt_res>(RES_PATTERN))`
		`(?P<pos>[1-9][0-9]*)`
		`(?P<mut_res>(RES_PATTERN))(?=([\.,\s)\]\'":;\-?!\]|\$))`		

This rule was implemented by splitting input text at periods and applying two regular expressions to each resulting "sentence." (While more effective methods exist for splitting sentences,[22] a review of the raw patterns and an error analysis following system development suggested that this simplistic approach is acceptable for this application.) Two regular expressions were applied to determine whether a sentence matched the bootstrap rule. First, an amino acid pattern (ignoring case) matched mentions of amino acids in their three-letter abbreviations or full names; this was required to match twice. Second, a sequence position pattern, designed to identify mentions of specific positions in a gene or protein, matched numerically expressed positive integers less than 10,000; this was required to match once.

3.1.2. *Raw pattern generation*

In a given sentence, if the amino acid pattern matched at least twice and the position pattern matched at least once, then a raw pattern was generated by selecting the shortest span of text that contained all of the amino acid residue and integer mentions. Amino acid mentions were replaced with `RES` and integer mentions were replaced with `POS`. Example input texts and the resulting raw patterns are presented for both true-positive [Table 2(a)–2(e) and 2(h)] and false-positive [Table 2(f) and 2(g)] matches.

We applied this rule to a local installation of Medline in its entirety (in October 2006). A total of 89,108,055 sentences from 16,333,215 Medline-record title and abstract fields were provided as input for creating raw patterns. A total of 262,157 unique raw patterns were generated from 295,618 sentences complying with the bootstrap rule. The most commonly occurring patterns included `RESPOSRES`, `RESPOS-->RES`, and `RESPOS to RES` (Table 2(a), 2(b), and 2(c), respectively). Of these raw patterns, 258,067 occurred only once in Medline; the remaining 4,090 patterns occurred at least twice. Since raw patterns were generated from the shortest span of text containing all of the `RES` and `POS` entities, each sentence could generate only a single raw pattern. The complete set of patterns generated by this process is available as supplementary material.[b]

As mentioned earlier, this step is intended to achieve high recall and is very prone to false positives. Of these raw patterns, 99.7% were not used for generating regular expressions because the vast majority represented false positives.

3.2. *Step 2: raw pattern refinement (2.0 person-hours)*

Many irrelevant patterns are obtained by applying the bootstrap rule to Medline. Through a partially automated process of refinement, we reduced the number of raw patterns from 262,157 to 634. This required 2 hours of work by an expert.

[b]All supplementary material is available at http://mutationfinder.sourceforge.net/

3.2.1. *Automatic raw pattern refinement*

First, to eliminate raw patterns likely to be unimportant, all raw patterns which occurred only once in Medline were filtered out. This reduced the number of raw patterns by 98.44%, from 262,157 to 4,090.

3.2.2. *Manual raw pattern refinement: false positives*

Second, the remaining 4,090 patterns were manually reviewed to determine their relevance. If it was intuitively obvious that a pattern represented a mutation mention, it was retained; otherwise, the pattern was deleted. (Questionable patterns tended to have a low number of occurrences and, since an objective of this project is to minimize the effort of humans in pattern development, it was decided that they should be deleted without investing time to investigate their validity.) Following this process, all false-positive (i.e. nonmutation) patterns were deleted.

3.2.3. *Manual raw pattern refinement: nonunique patterns*

Finally, raw patterns which would not provide novel mutation patterns were manually deleted. For example, the raw pattern `RESPOSRES and RESPOSRES` [Table 2(h)] would not provide a novel mutation pattern because it contained two mentions of the simpler pattern `RESPOSRES` [Table 2(a)]. These manual refinement steps further reduced the number of raw patterns from 4,090 to 634, and took a total of 2 hours.

3.3. *Step 3: mutation pattern generation (2.5 person-hours)*

Since raw patterns could contain more than one mutation mention [e.g. Table 2(d) and 2(e)] and since step 1 did not assign wild-type and mutant identities to each residue mention, we needed to tag the elements to be extracted as the wild-type residue, mutant residue, and sequence position in each raw pattern. In this step, each remaining raw pattern was manually reviewed and edited to assign semantic tags indicating whether a given term was the wild-type residue, mutant residue, or sequence position entity. The resulting patterns are referred to as mutation patterns, since they can be used to represent point mutation mentions unambiguously. Each mutation pattern will match, at most, a single mutation.

For raw patterns containing only a single mutation mention [e.g. Table 2(a)–2c)], the `RES` entry was replaced with `WRES` or `MRES`, and the `POS` entry with `SPOS`. The decision of which `RES` term to be converted to `WRES` versus `MRES` was informed by domain knowledge. In 83.13% of the mutation patterns, the first residue mention was assigned `WRES`.

One hundred raw patterns contained more than one mutation mention [e.g. Table 2(d) and 2(e)]. Semantic tag assignment was slightly more complicated for these. The raw pattern was duplicated to appear once for each mutation mention, and each raw pattern was edited to assign identities. This duplication of patterns

containing more than one mutation mention resulted in an addition of 125 patterns, from 634 raw patterns to 759 mutation patterns. In Table 2(d) and 2(e), the raw patterns are duplicated to yield three and two mutation patterns, respectively, corresponding to the number of mutation mentions in each.

In duplicated patterns [e.g. Table 2(d) and 2(e)], extraneous text remained that would result in the overspecialization of a pattern. These were edited so that only the shortest piece of text containing the `WRES`, `MRES`, and `SPOS` entities was retained; and leading or trailing text was removed. (For example, in the first mutation pattern in Table 2(e), the trailing text `and POS` were removed when the pattern was duplicated.)

3.3.1. *Automatic mutation pattern generation*

We explored automating this process, and found that a large development corpus would be required to achieve acceptable recall. Given a collection of raw patterns, mutation patterns with greater than two occurrences of `RES` or greater than one occurrence of `POS` were deleted. In the remaining raw patterns, the first `RES` mention was replaced with `WRES`, the second with `MRES`, and the position with `SPOS`; this resulted in an automatically generated collection of mutation patterns. Beginning with a simple mutation finder that would match only *wNm*-formatted mutation mentions using single-letter amino acid abbreviations, each mutation pattern was tested to determine if it increased the F-measure of the system on the development data. If so, that pattern was retained; otherwise, it was deleted.

Due to the infrequent occurrence of many mutation patterns in Medline, the patterns actually occurring in our corpus are limited. Very few patterns end up being retained by this method, and the resulting system is overfit to the development data. This approach was not incorporated into our final system, but illustrates a way to automate mutation pattern generation. Manually generating mutation patterns, which took approximately 2.5 hours, is far more time-efficient than developing a larger corpus to support the process.

3.4. *Step 4: automatic regular expression generation (no manual intervention)*

Translation from mutation patterns to regular expressions is a computationally trivial task. Each of the entity types in a raw pattern was replaced with a regular expression to match the text strings which might appear in that slot. Residues were matched in their three-letter abbreviation or full name, and sequence positions were matched as positive integers between 1 and 9,999. The entities to be extracted from the text (`WRES`, `MRES`, and `SPOS`) were matched using symbolic grouping expressions. In addition to the regular expressions contained in the collection of mutation patterns, an additional pattern was always added to match *wNm*-formatted mutation mentions using single-letter abbreviations. This pattern was case-sensitive, and is the only pattern that matches single-letter-abbreviated residue mentions.

3.4.1. *Automatic linguistic processing*

A minor processing step was applied at this stage to generalize regular expressions. Occurrences of either `a` or `an` were replaced with the regular expression `(a|an)`, so the texts "*Ala42 to a glycine*" and "*Ala42 to an isoleucine*" would both be matched by the same regular expression.

3.5. *Step 5: system optimization*

After the set of regular expressions was generated and refined, it was possible to optimize the rule set and individual rules by testing their performance on annotated and unannotated corpora. For MutationFinder, two types of system optimization were performed: optimization of precision and optimization of recall. Our experience suggests that optimizing precision is essential, but that optimizing recall is optional. Optimization of precision was performed via a post hoc error analysis and took very little time; optimization of recall required the use of our human-annotated development data. While the error analysis itself did not take long, approximately 18 person-hours were devoted to corpus annotation. (Until this point, no annotated data has been required for system development.) Because optimization of system recall is not required (as we will show), we do not include the time for corpus development in our estimate of time for system development.

3.5.1. *Optimization of precision with post hoc error analysis on unannotated data (approximately 1 person-hour)*

We used an unannotated corpus of GeneRIFs[23] to optimize MutationFinder's precision. Before precision optimization, we applied MutationFinder to the complete collection of GeneRIFs (through June 2006). We randomly selected 100 GeneRIFs which MutationFinder identified as containing mutations. We then manually scored MutationFinder's Document Retrieval precision on these GeneRIFs. False positives were categorized, and adjustments were made to avoid the most significant error types. Three major adjustments to MutationFinder (of the six briefly presented in our previous publication[6]) were made as a result of this error analysis. These are now presented in more detail.

Prior to optimization, certain gene/protein names were commonly mistaken for mutation mentions. For example, *H4A* and *E2F* were commonly mistaken as representing mutations. MutationFinder's regular expression for matching *wNm*-format mutation mentions with single-letter abbreviations only was modified to require that N be greater than 9. This led to a 30% increase in precision on the same collection of GeneRIFs [Table 3(b)]. Next, to further avoid confusion with gene names, we required that the same regular expression be case-sensitive, only matching uppercase letters. This led to an 11% increase in precision [Table 3(c)]. Third, this error analysis led us to implement a postprocessing rule to ignore "mutations"

Table 3. Post hoc precision estimates on a sample of 100 GeneRIFs determined to mention point mutations by MutationFinder prior to precision optimization. Preoptimized MutationFinder refers to an implementation of MutationFinder that does not incorporate these three precision-optimizing rules. These data show the relative contribution of each individual rule.

Description	Precision (%)
(a) Preoptimized MutationFinder	0.58
(b) Preoptimized MutationFinder + $N > 9$ rule	0.88
(c) Preoptimized MutationFinder + case-sensitivity rule	0.69
(d) Preoptimized MutationFinder + wild-type $\neq$ mutant rule	0.61
(e) MutationFinder	0.97

where the wild-type and mutant entities were identical. For example, the mention of a homozygous genotype, *C1298C*, was initially extracted as a point mutation. This rule led to a 3% increase in precision [Table 3(d)].

This post hoc error analysis required only a minimal time commitment from a human domain expert. Scoring the precision of the 100 random GeneRIFs took approximately 10 minutes, and the resulting error analysis led to a large increase in system precision. Prior to the implementation of these rules, the precision achieved on GeneRIFs was 58% [Table 3(a)]; after optimization, when using the same data set, the precision was 97% [Table 3(e)] and no false negatives were incurred. (Note that the precision improvements displayed in Table 3(b)–3(d) are for these changes in isolation. The optimizations are not mutually exclusive. For example, a false positive resulting from the text "*crucial for C1q binding to ligands*"[c] is avoided by both the $N > 9$ and case-sensitivity rules.)

A danger in a precision-oriented optimization step such as this is the overfitting of rules to the test collection. Care should be taken to only incorporate rules that make intuitive sense and are supported by the analysis. For example, rather than requiring that sequence positions always be greater than 9, we tailored this rule specifically to what we observed and what made intuitive sense: many gene names appear similar to single-letter-abbreviated *wNm*-formatted mutation mentions, but we do not expect to see gene names that conform to many other mutation patterns. We tailored this rule based on domain knowledge and the results of this error analysis. Incidentally, the remaining false positives were all cell line names that were extracted as mutation mentions. We revisit the problem of mistaking gene and cell line names for mutations in our discussion below.

Since we had a human-annotated corpus available for optimization, we tested these rules on that data set. Comparison of the preoptimized and postoptimized MutationFinder on our development data set showed an approximate 11% precision increase with only a 1% decrease in recall for extracting Normalized Mutations.

[c]Source Entrez Gene ID: 714.

3.5.2. *Optimization of recall — optional (approximately 18 person-hours for corpus annotation)*

Recall optimization of the complete rule set generally took one of two forms. First, the overall utility of general rules was tested (e.g. replacing occurrences of `a` or `an` with `(a|an)` in regular expressions) by testing the performance of MutationFinder on the development data with the rule alternately incorporated and not incorporated. Second, error analyses performed by looking at false negatives incurred by the system were used to inform future decisions about system design.

The methods described in our six-step approach are biased toward high recall. By applying the bootstrap rule to a very large corpus of relevant documents, important patterns can be compiled without the need for much human interaction. We found that recall optimization on our annotated development data did little to improve the performance of our system on these same data.

Table 4 compares different approaches we applied. Table 4(a) presents the performance of our best collection of regular expressions on the development data. We now discuss the implementation details we explored in the context of their effect on performance.

3.5.3. *Utility of single-letter abbreviations*

As described in Sec. 3.4, when automatically generating regular expressions we matched amino acid residues by their three-letter or full name abbreviations, but not by their single-letter abbreviations. As illustrated by comparing rows (a) and (b) in Table 4, matching on single-letter abbreviations is associated with an Extracted Mentions recall increase of 3.3 percentage points and a precision decrease of 34.2 percentage points. While this decrease in precision could likely have been reduced by allowing one-letter abbreviations only in certain patterns (rather than globally), the recall increase presented is the maximum that could be achieved. Because a major emphasis of this study is to minimize the amount of time invested in manual pattern generation and refinement, it was determined that hand-selecting patterns would take too long to be worth a maximum of a 3.3-percentage-point increase. It is also important to note that the Normalized Mutations recall increase was only 0.6 percentage points (data not shown), suggesting that in many practical applications of MutationFinder the recall difference would be minimal.

Table 4. Extracted Mention performances achieved on development data by systems built with various collections of regular expressions during system optimization.

Description	TP	FP	FN	P	R	F
(a) Top-performing system	456	8	94	0.983	0.829	0.899
(b) One-letter abbreviations included	474	265	76	0.641	0.862	0.736
(c) No linguistic processing of `a`, `an`	456	8	94	0.983	0.829	0.899

TP, true positive; FP, false positive; FN, false negative; P, precision; R, recall; F, *F*-measure.

3.5.4. *Utility of linguistic processing*

As noted in Sec. 3.4.1, some regular expressions were generalized by substituting the words "a" and "an" with the regular expression `(a|an)`. This processing step resulted in no change in performance on our development corpus [see Table 4(a) and 4(c)]. Since we observed no precision degradation, but expect a recall improvement on a larger corpus, we opted to incorporate this processing step into our system.

3.6. *Step 6: system testing — optional (approximately 36 person-hours for corpus annotation)*

After optimization, MutationFinder was applied to our previously unseen test data set. Performance in terms of our three performance metrics is presented in Table 5. As noted earlier, system testing on unseen data is not required for rule development. We therefore do not include the time for corpus development in our estimate of time for system development.

MutationFinder was evaluated on the blind test subset of the gold standard corpus, which consists of 910 mutation mentions in 508 documents annotated by annotators 1 and 2. MutationFinder achieved a precision of 0.984 and a recall of 0.819 on Extracted Mentions on this test collection. Complete test results are presented in Sec. 5.1 and Table 5.

3.6.1. *Automatically generated versus manually generated rules*

To test our hypothesis that automatically generated patterns can achieve equivalent or improved performance when compared to manually generated patterns, we compared MutationFinder with a collection of manually generated rules. These manually generated rules were constructed by annotator 3 prior to the automatically generated rules in an initial attempt to build a high-performance mutation

Table 5. Comparison of manually and automatically generated rule sets on the blind test subset of the gold standard corpus. Counts of true positives (TP), false positives (FP), and false negatives (FN) are presented, along with precision (P), recall (R), and F-measure (F) for the three different performance metrics. These data can be recreated by applying versions `0_2-beta` (for manually generated rules) and `0_3-beta` (for automatically generated rules) of MutationFinder to the blind test data, and scoring with the provided performance script. All code and data are available at http://mutationfinder.sourceforge.net.

	TP	FP	FN	P	R	F
Manually generated rules						
Extracted Mentions	686	4	221	0.994	0.756	0.859
Normalized Mutations	352	2	124	0.994	0.740	0.848
Document Retrieval	146	0	36	1.000	0.802	0.890
Automatically generated rules						
Extracted Mentions	743	12	164	0.984	0.819	0.894
Normalized Mutations	384	10	92	0.975	0.807	0.883
Document Retrieval	162	1	20	0.994	0.890	0.939

recognition system. The test collection was blind with respect to both the automatically generated and manually generated rules. The manually generated system is available at http://mutationfinder.sourceforge.net as `MutationFinder-0_2-beta`.

4. Gold Standard Corpus

We constructed a gold standard data set consisting of 1,515 annotated complete point mutation mentions in the title and abstract fields of 813 Medline records (the gold standard corpus). To compile a corpus likely to contain many mutation mentions, documents were randomly selected from the collection of articles cited as primary citations for mutant Protein Data Bank[24] (PDB) entries.

Three mutation types were annotated: point mutations, insertions, and deletions.[d] The structure of each mutation annotation was based on a simple mutation event ontology that was developed for this purpose (Supplementary Fig. 1). The vast majority of mutations annotated were point mutations; this corpus proved to contain too few insertion and deletion mentions to be useful for testing insertion/deletion extraction systems.

Mutation annotations could be complete or partial. For example, the text "*mutation at alanine-64*" only mentions a wild-type residue and a sequence position. If the associated mutant residue was not present in the same document (i.e. title and abstract), a partial point mutation annotation would be created that contained a blank `Mutant Element` field. If a mention contained the information necessary to fill the `Sequence Position`, `Wild-type Element`, and `Mutant Element` fields (for a `Substitution Event`), the resulting annotation was complete. Comprehensive information on the annotation process is described in our annotation guidelines, which are provided as supplementary material.

4.1. *Preprocessing*

To reduce the time needed to annotate the corpus, all of the documents (with the exception of 25, which will be discussed shortly) were preprocessed to automatically annotate obvious mutation mentions. Mutations described in the *wNm* format were automatically annotated. All automatic annotations were manually inspected for correctness during the annotation process.

4.2. *Annotation*

Knowtator,[25] a text annotation tool, was used for corpus annotation. Annotators marked up mentions of mutations that were not identified in the preprocessing step, and validated mutation mentions that were labeled during the preprocessing step.

[d]Although insertions and deletions were annotated, MutationFinder does not attempt to extract these mutation types. These may be addressed in a future version.

Preprocessing errors (e.g. the text "*the E2F protein*" erroneously annotated as a point mutation) were repaired or deleted, as necessary.

The corpus was divided into five subcollections for the annotation process:

(1) 25 abstracts, preprocessed, annotated by all annotators;
(2) 25 abstracts, not preprocessed, annotated by all annotators;
(3) 254 abstracts, preprocessed, annotated by annotator 1;
(4) 254 abstracts, preprocessed, annotated by annotator 2; and
(5) 255 abstracts, preprocessed, annotated by annotator 3.

Subcollections 1 and 2 were used to calculate interannotator agreement (IAA); by preprocessing only subcollection 1, we could determine what effect, if any, preprocessing had on the annotation process. The 50 documents in subcollections 1 and 2 were randomly selected from the full corpus and added (randomly dispersed) to each annotator's annotation set (subcollections 3, 4, and 5). Some bias may have been introduced by not preprocessing subcollection 2 (because these documents were sometimes readily identifiable), but we considered it important to understand the effect (if any) of preprocessing on interannotator agreement.

4.3. *Annotation guidelines*

To guide the annotation process and promote continuity between annotators, a draft set of annotation guidelines was constructed prior to the annotation process. Meetings among the annotators were periodically conducted, whereby difficult cases were discussed and the annotation guidelines adjusted. Final annotation guidelines are provided as supplementary material.

4.4. *Annotation quality assurance*

After the completion of the annotation process, each of the annotators independently performed a review and consistency check of their entire annotation set based on the finalized set of annotation guidelines.

4.5. *Interannotator agreement*

Interannotator agreement (IAA) was calculated in a pairwise manner using the same metrics that were used to evaluate MutationFinder. Each annotator's data set was alternately treated as the gold standard; and the average F-measure (harmonic mean of precision and recall) was calculated for Extracted Mentions, Normalized Mutations, and Document Retrieval.[e] IAA was calculated over all complete point

[e]An alternative approach to calculating interannotator agreement in the corpus construction literature is the Kappa metric.[26] Kappa attempts to take into account the likelihood of chance agreement between annotators. We do not use it here for a variety of reasons. A number of authors have pointed out various deficiencies of Kappa for annotation tasks like ours, ranging

Table 6. Mean pairwise interannotator agreement (IAA). Mean pairwise F-measure is presented, using three evaluation metrics, on the 25 documents in subcollection 1 (PREPROCESSED), the 25 documents in subcollection 2 (NOT PREPROCESSED), and the 50 documents in the union of subcollections 1 and 2 (ALL IAA).

Task	PREPROCESSED	NOT PREPROCESSED	ALL IAA
Extracted Mentions	0.949	0.951	0.950
Normalized Mutations	0.916	0.967	0.941
Document Retrieval	0.926	1.000	0.961

mutations, and the mean pairwise IAA was 95.0% for Extracted Mentions. Complete IAA data is presented in Sec. 5.2 and Table 6.

4.6. *Division of gold standard into development and test data*

Annotator 3 (Caporaso) had the role of both corpus annotator and primary developer of MutationFinder. To avoid overfitting MutationFinder to the gold standard corpus, the corpus was divided into development and test subsets.

The development subset of the gold standard was composed of documents from subcollections 1, 2, and 5 (i.e. the 305 documents that were annotated by annotator 3). The test subset was composed of documents from subcollections 3 and 4 (i.e. the 508 documents that were not annotated by annotator 3). For a detailed breakdown of the gold standard corpus and the division between development and test data, see Table 7.

Table 7. Corpus contents. Contents of the development (DEV) and test (TEST) sets derived from the generated gold standard corpus. For the three mutation types that were annotated, the number of complete annotations is presented, followed by the number of partial annotations in parentheses. The vast majority of annotations were point mutations.

	DEV	TEST	TOTAL
Documents	305	508	813
Documents containing mutation mentions	111	212	323
Documents not containing mutation mentions	194	296	490
Point mutation annotations (partial)	605 (56)	910 (150)	1515 (206)
Insertion annotations (partial)	0 (0)	0 (3)	0 (3)
Deletion annotations (partial)	4 (0)	10 (5)	14 (5)

from the difficulty in deciding how to calculate chance agreement[27] to the widespread difficulty in interpreting it.[28] More salient to the evaluation of the work described here is the fact that, as Hripcsak and Rothschild[29] have demonstrated, when the number of negative instances in a set of data far outnumbers the number of positive instances, chance agreement approaches zero and Kappa is essentially equivalent to the pairwise F-measure between arguments. This characterizes very well the situation in this task, where the number of tokens (roughly equivalent to that of words) in the corpus that are instances of mutation mentions is far outnumbered by the number of tokens that are not instances of mutations.

The development corpus was used for informing decisions about which patterns and rules should be included in the system. The test set served as a completely blind test for MutationFinder. Evaluation was only performed using the test data following system development and optimization.

5. Results

We address two hypotheses in the article: (1) that automatic methods can be used to learn patterns of sufficient quantity and quality in order to match or outperform a manually built rule set, and (2) that annotation standards and text collections sufficient for the building of high-quality corpora of mutation mentions with good IAA can be developed. We now present data supporting both of these hypotheses.

5.1. *Manually versus automatically constructed rules*

Our initial attempt at building a high-performance concept recognition system involved manually compiling rules informed during annotation of the development corpus by annotator 3. This annotation process required approximately 18 hours to complete, and was an important step in the development of the manually compiled rule set. Our automatic approach required only 5.5 person-hours for rule development, and achieved a higher F-measure than the manually constructed rule set when tested on data that were blind with respect to both rule sets (Table 5). Since statistical tests have not yet been performed, we conclude that automatically generated rule sets can achieve performance that is at least equivalent to manually generated rule sets. In our case, the automatically constructed rule set took less than 1/3 of the time required for compiling the manually constructed rule set.

In an additional test of MutationFinder's automatically constructed rule set, we applied MutationFinder to the full-text corpus recently published with MutationGraB[10] (Table 8). For this analysis, we combined their three development and three test corpora into a single test corpus. Our Normalized Mutations metric is equivalent to their "*Cited Mutation*" metric. Since Lee *et al.*[10] only present performance data on this corpus for the more ambitious task of extracting associated pairs of mutations and genes/proteins, a direct comparison with their system cannot be

Table 8. Performance of MutationFinder on the Mutation GraB corpus. Counts of true positives (TP), false positives (FP), and false negatives (FN) are presented, along with precision (P), recall (R), and F-measure (F) for the three different performance metrics. This corpus did not annotate mutation mentions, so Extracted Mentions cannot be calculated. All code and data, including a translation of the corpus that is compatible with our performance script, are available at http://mutationfinder.sourceforge.net.

Automatically generated rules	TP	FP	FN	P	R	F
Normalized Mutations	3572	177	837	0.952	0.808	0.874
Document Retrieval	503	1	26	0.998	0.951	0.974

performed at this time. The nearly identical performance of MutationFinder on the Mutation GraB corpus and on our blind test data (Table 5) further suggests that MutationFinder's performance will scale well to unseen data.

Our automated pattern generation approach yielded 759 patterns. The frequency of these patterns in the literature roughly matched a power-law (Zipfian) distribution, and the resulting regular expressions in our `regex.txt` file (available on the project web site) are listed in order from most commonly matching to least commonly matching. While most of the very high-frequency regular expressions were included in the manually generated pattern set, it is the compilation of the lower-frequency patterns that is truly valuable. These patterns could be compiled manually, but the time required would be prohibitive.

5.2. *Interannotator agreement (IAA)*

A total of 1,515 complete point mutation mentions were annotated in the 813 documents. The mean pairwise IAA, when comparing Extracted Mentions calculated on the fifty documents annotated by all three annotators, was 95%. Similar IAA results were obtained when calculating over the preprocessed versus non-preprocessed data; preprocessing does not appear to have an effect on IAA. We expect that the observed differences are not statistically significant (see Table 6). The resultant gold standard corpus is available at http://mutationfinder.sourceforge.net.

6. Discussion and Future Directions

6.1. *Error analysis*

6.1.1. *Precision limitations*

MutationFinder achieved very high precision on all of the data sets on which it was tested. There were, however, several categories of residual errors. In some cases, MutationFinder extracted other entities, such as genes, proteins, or cell lines, whose names appeared similar to mutation mentions. Additionally, MutationFinder occasionally incurred false positives due to promiscuous pattern matching; for example, in the text "*the transfer of a proton from the catalytic cysteine to a His 207-Asp 205 diad via a system of ordered water molecules,*"[f] `H207D` was extracted as a point mutation because MutationFinder's pattern `WRES SPOS-MRES` was matched.

To quantify the problem of extracting gene or cell line names as mutations, we generated lists of gene names and cell line names to provide as input to MutationFinder. A list containing 2,706,089 unique gene names was constructed by parsing out the symbol, official symbol, full name, and synonyms for each gene record in the Entrez Gene `gene_info`[g] file. A collection of 8,102 unique cell line names was compiled from three online sources.[30–32] We supplied these lists as input to

[f]Source PMID: 10841779.
[g]ftp://ftp.ncbi.nih.gov/gene/DATA/gene_info.gz.

MutationFinder, and counted the number of gene and cell line names which MutationFinder erroneously identified as mutation mentions. Only 679 (0.025%) of the gene names and 30 (0.370%) of the cell line names were identified as mutations by MutationFinder, leading us to conclude that these false positives will be rare in practice.

The `H207D` example presented above represents a false-positive mutation mention extracted from our development corpus. Promiscuous patterns do not appear to be overly problematic for us at this stage, in part due to the post hoc error analysis presented in Sec. 3.5.1. Since post hoc error analyses of precision are not dependent on the availability of annotated data, we expect that in most cases problematic patterns can be easily identified and removed or modified, as we did with MutationFinder. An explicit fault model[33] is helpful for the sometimes surprisingly difficult task of error analysis. Cohen *et al.*[34] described the use of a fault model in designing test suites for a biomedical named entity recognition system, and Johnson *et al.*[35] described the design of a fault model for post hoc error analysis of ontology linking systems. A strength of our approach is that it is very easy to review, modify, and delete patterns. (We retained the pattern that caused the `H207D` error, but modified other patterns as discussed earlier.)

6.1.2. *Recall limitations*

MutationFinder did not achieve the full recall that we believe it is capable of (with high precision maintained). We performed an error analysis by reviewing all false negatives incurred by MutationFinder on our development corpus at the Document Retrieval level. Errors fell into four categories, and in some circumstances a single error fell into more than one category. First, in five cases, mutations were missed due to the presence of intervening text in the patterns. For example, the text "*His-554 of IIAMtl was mutated to glutamine*"[h] describes the mutation `H554Q`. The first mutation pattern presented in Table 2(d) comes close to matching this mention, except that there is intervening text in the input: "*of IIAMtl*". Second, in four cases, mentions were missed because patterns did not comply with the bootstrap rule (e.g. a mutation mention was split across sentence boundaries). Third, two errors arose due to the presence of compound complex coordinations[i] being used to describe several mutations at once. For example, if the patterns in Table 2(e) were applied to the text "*Ala-42 was mutated to Gly, Phe, Leu, and Trp*", we would only extract two of the four mutations (i.e. `A42G` and `A42F`). Finally, two more errors arose due to our exclusion of single-letter-abbreviations when matching mutation mentions in all patterns except `WRESSPOSMRES`.

The first three of these limitations could be avoided by modifying MutationFinder's rule set, while the fourth represents an underlying difficulty in concept

[h]Source PMID: 16443929.
[i]Phrases where a noun phase is meant to be associated with each element in a list.

recognition. The problem of intervening text can be addressed by applying the method presented by Huang and colleagues[18] (by attempting to align patterns with text), or by syntactically analyzing the text and allowing optional extensions to the patterns. In the example provided above, we might try specifying that a prepositional phrase may be optionally present in the pattern. While syntactic processing may be necessary to address these few remaining problems, the complexities introduced by its application (perhaps even increasing the complexity of the pattern language beyond regular expressions) can at best address a relatively small number of errors. Limitations arising from the bootstrap rule could also be addressed. In our error analysis, two of these four errors arose because a mutation mention was split across two sentences. By rebuilding raw patterns, but allowing for occurrences of residues and sequence positions in neighboring sentences, we could obtain patterns that matched these mentions. Next, by adding an additional syntactic processing step to step 4, compound complex coordinations could be generalized to allow a variable number of intermediate residue mentions. The mentions missed due to single-letter-abbreviated residue mentions, however, represent a more inherent problem: the trade-off between precision and recall that must be made when developing a concept recognizer. As discussed in Sec. 3.5.3, a large precision hit is incurred by matching single-letter-abbreviated residue mentions, while only a small increase in recall is achieved. Without investing a large amount of time to investigate when single-letter abbreviations should be accepted, mentions of this type would always be missed by MutationFinder to maintain precision.

If higher recall is required than is initially achieved when applying our rule-learning methodology to other tasks, we recommend performing error analyses to identify false negatives and addressing those that cause the most significant problem first (at the expense of additional person-hours for system development). Recall optimization, however, will often require some sort of annotated test data. If resources are unavailable for corpus development, Craven and Kumlien[36] recommend approaches for rapidly developing weakly annotated data. Based on our error analysis, we conclude that false negatives, in many cases, can be avoided through additional syntactic and other processing in step 4 of our approach. We expect that recall optimization will not be necessary in many applications, and discuss this further in Sec. 6.3.

6.2. *Generation of patterns for other concept recognition tasks*

To apply our methodology of developing patterns for extracting different types of events or relationships, raw patterns would need to be compiled with an alternate bootstrap rule. For example, we will examine the problem of extracting descriptions of protein transport events[37] from the biomedical literature. Imagine that we would like to design a system to extract a `TRANSPORTED-PROTEIN`, a `SOURCE`, and/or a `DESTINATION` from unseen text. Our bootstrap rule could specify that if a protein and a cellular location are mentioned in a single sentence, then a

Table 9. Hypothetical patterns which might be generated for a protein transport event extraction system. These examples were manually obtained from GeneRIFs describing protein transport events.

Input text (Entrez Gene ID)	Pattern
(a) ***Bcl-xL*** *from the* ***cytosol*** *to the* ***outer mitochondrial membrane*** (598)	`TRANSPORTED-PROTEIN` from the `SOURCE` to the `DESTINATION`
(b) ***MDMX*** *is transported to the cell* ***nucleus*** *upon DNA damage* (4194)	`TRANSPORTED-PROTEIN` is transported to the cell `DESTINATION`
(c) ***Stau2(59)*** *is exported from the* ***nucleus*** *by two distinct pathways* (171500)	`TRANSPORTED-PROTEIN` is exported from the `SOURCE`

protein transport event may have been described. Named entity recognizers could be applied to identify sentences complying with this rule. Both a protein-name recognizer and a cellular-location recognizer would be required to match at least once. Raw patterns could then be generated from complying sentences and manually converted into transport patterns. Example texts and patterns are presented in Table 9. This example differs from MutationFinder to illustrate how the system could be modified to handle the extraction of incomplete events [see Table 9(b) and 9(c)].

Recognizing mutations in text is relatively simple compared to recognizing protein transport events, and applying our approach to more complex concept recognition problems may require more work or result in lower performance. We plan to apply our methodology to generate more complex patterns and report on its utility in future publications.

6.3. *Optionality of recall optimization*

We optimized MutationFinder's precision and recall to construct a high-performance mutation recognition system. Our findings are consistent with the hypothesis that the construction of rule sets for mutation recognition is largely automatable. We conclude that our approach, being recall-oriented, does not require recall optimization, although it can be helpful. Precision optimization, on the other hand, is necessary. This is convenient: precision optimization can be performed inexpensively via post hoc analyses, while recall optimization typically requires more costly human-annotated data.

MutationFinder achieves high recall prior to optimization [Table 4(a) and 4(c)]. While an error analysis has suggested four causes of false negatives, we expect that for many practical applications, expending additional effort to optimize recall may not be worthwhile. Our tests of MutationFinder involved collections of abstracts or GeneRIFs as corpora. We expect that, if MutationFinder were evaluated on annotated full-text, the Normalized Mutation recall would be higher than if it

were evaluated on abstracts alone (although precision may be adversely effected).[j] We reason that when a mutation is the subject of an article, it is likely to be mentioned more than once. With each additional mention of a specific mutation, the chances of it being matched by a pattern increase. As full text continues to become more available, we expect that if higher recall is required, MutationFinder could be applied to full text versus abstracts alone. Additionally, as we have previously noted,[6] for most practical concept recognition applications, extracting at least one instance of each targeted event is sufficient (i.e. the Normalized Mutations task is more important than the Extracted Mentions task).

Taken together, the already high recall achieved by our approach, the increasing availability of full text, and the generally greater significance of extracting normalized information as opposed to individual mentions support our claim that recall optimization (with human-annotated data) of systems developed by our approach is not required.

6.4. *Association of mutation mentions with genes or proteins*

MutationFinder extracts mentions of point mutations from the literature with high precision and recall. An even more useful system would associate mutations with genes or proteins. We plan to address this next, and we predict that our pattern learning approach will scale well to this problem. By using MutationFinder as an entity recognizer and by employing a high-performance gene/protein name recognizer, we can learn patterns for extracting these associations from the literature. In addition to providing a useful concept recognition system, this future work will provide insight on automatically learning patterns to extract associated entity mentions from text.

7. Conclusions

An unprecedented quantity of biomedical text is available via Medline, and full-text journal publications are becoming increasingly available to the public. Biomedical language processing offers hope for managing this ever-increasing literature base, but it has been noted that "[k]nowledge-based NLP systems will be practical for real-world applications only when their domain-dependent dictionaries can be constructed automatically."[13] We have presented one mechanism, along with a case study, illustrating how Medline can be utilized to make data currently available only in the biomedical literature more accessible to researchers.

We have presented the methodology employed to develop MutationFinder, our high-performance system for extracting point mutation mentions from the

[j]In our test on the Mutation GraB corpus, we did notice a slight decrease in precision when MutationFinder was tested on full text, compared with the precision on abstracts in our test collection. We plan to perform a post hoc error analysis of MutationFinder using the Mutation GraB corpus to identify the cause of the additional false positives. Since abstracts were not annotated separately in this corpus, we were not able to test our abstract versus full-text hypothesis on this data set.

biomedical literature; our publicly available mutation corpus, which should prove valuable for enabling direct comparisons between mutation recognition systems; and several measures of MutationFinder's performance, beyond what was presented in our previous MutationFinder publication. Our methodology is extensible to other concept recognition goals, and provides a means for achieving high precision and recall while minimizing the investment of time in corpus development (when training and testing machine-learning–based systems) or system development (when manually developing rule-based systems). This work additionally highlights the significance of studying mutation recognition from free text: the mutation recognition problem is a useful model for text mining from biomedical text in general, and it appears to provide insights that can be generalized to other biomedical domains.

Acknowledgments

The authors would like to thank Jim Martin for helpful discussion and guidance in the initial stages of this project. Additionally, we would like to thank Philip Ogren for providing insight with regard to the relationship between the kappa statistic and F-measure, Zhiyong Lu for providing GeneRIFs describing protein transport events, and the four anonymous reviewers for their helpful comments and suggestions during the preparation of this article. This work was partially funded by NLM grants R01-NLM-009254 and R01-LM008111 to L. Hunter.

References

1. Hunter L, Cohen KB, Biomedical language processing: What's beyond PubMed?, *Mol Cell* **21**(5):589–594, 2006.
2. Cohen AM, Hersh W, A survey of current work in biomedical text mining, *Brief Bioinform* **6**(1):57–71, 2005.
3. Rebholz-Schuhmann D, Kirsch H, Couto F, Facts from text — Is text mining ready to deliver?, *PLoS Biol* **3**(2):e65, 2005.
4. Yeh A, Morgan A, Colosimo M, Hirschman L, BioCreAtIvE task 1A: Gene mention finding evaluation, *BMC Bioinformatics* **6**(Suppl 1):S2, 2005.
5. Baumgartner WA Jr, Cohen KB, Fox LM, Acquaah-Mensah G, Hunter L, Manual curation is not sufficient for annotation of genomic databases, *Bioinformatics* **23**(13): i41–i48, 2007.
6. Caporaso JG, Baumgartner WA Jr, Randolph DA, Cohen KB, Hunter L, MutationFinder: A high-performance system for extracting point mutation mentions from text, *Bioinformatics* **23**(14):1862–1865, 2007.
7. Jackson P, Moulinier I, *Natural Language Processing for Online Applications: Text Retrieval, Extraction, and Categorization*, John Benjamins, Amsterdam, 2002.
8. Horn F, Lau AL, Cohen FE, Automated extraction of mutation data from the literature: Application of MuteXt to G protein-coupled receptors and nuclear hormone receptors, *Bioinformatics* **20**(4):557–568, 2004.
9. Rebholz-Schuhmann D, Marcel S, Albert S, Tolle R, Casari G, Kirsch H, Automatic extraction of mutations from Medline and cross-validation with OMIM, *Nucleic Acids Res* **32**(1):135–142, 2004.
10. Lee LC, Horn F, Cohen FE, Automatic extraction of protein point mutations using a graph bigram association, *PLoS Comput Biol* **3**(2):e16, 2007.

11. Bonis J, Furlong LI, Sanz F, OSIRIS: A tool for retrieving literature about sequence variants, *Bioinformatics* **22**(20):2567–2569, 2006.
12. Baker CJO, Witte R, Mutation mining — A prospector's tale, *J Inf Syst Front* **8**(1):47–57, 2006.
13. Riloff E, Automatically constructing a dictionary for information extraction tasks, in *Proceedings of the Eleventh National Conference on Artifical Intelligence*, AAAI Press/MIT Press, Menlo Park, CA, pp. 811–816, 1993.
14. Riloff E, Automatically generating extraction patterns from untagged text, in *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, AAAI Press/MIT Press, Menlo Park, CA, pp. 1044–1049, 1996.
15. Hovy EH, Hermjakob U, Lin CY, The use of external knowledge in factoid QA, *Text REtrieval Conference*, 2001.
16. Ravichandran D, Hovy E, Learning surface text patterns for a question answering system, in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pp. 41–47, 2002.
17. Tanabe L, *Text Mining the Biomedical Literature for Genetic Knowledge*, Ph.D. thesis, George Mason University, Washington, DC, 2003.
18. Huang M, Zhu X, Hao Y, Payan DG, Qu K, Li M, Discovering patterns to extract protein–protein interactions from full texts, *Bioinformatics* **20**(18):3604–3612, 2004.
19. Hao Y, Zhu X, Huang M, Li M, Discovering patterns to extract protein–protein interactions from the literature: Part II, *Bioinformatics* **21**(15):3294–3300, 2005.
20. Salus PH, *A Quarter Century of UNIX*, Addison-Wesley UNIX and Open Systems Series, Addison-Wesley Professional, New York, 1994.
21. Hakenberg J, Royer L, Plake C, Strobelt H, Schroeder M, Me and my friends: Gene mention normalization with background knowledge, in *Proceedings of the Second BioCreative Challenge Evaluation Workshop*, 2007.
22. http://www.alias-i.com/lingpipe/
23. Mitchell JA, Aronson AR, Mork JG, Folk LC, Humphrey SM, Ward JM, Gene indexing: Characterization and analysis of NLM's GeneRIFs, *Proceedings of the AMIA 2003*, 2003.
24. Deshpande N, Addess KJ, Bluhm WF, Merino-Ott JC, Townsend-Merino W, Zhang Q, Knezevich C, Xie L, Chen L, Feng Z, Green RK, Flippen-Anderson JL, Westbrook J, Berman HM, Bourne PE, The RCSB Protein Data Bank: A redesigned query system and relational database based on the mmCIF schema, *Nucleic Acids Res* **33**(Database issue):D233–D237, 2005.
25. Ogren PV, Knowtator: A plug-in for creating training and evaluation data sets for biomedical natural language systems, in *Proceedings of the 9th International Protégé Conference*, pp. 73–76, 2006.
26. Carletta J, Assessing agreement on classification tasks: The Kappa statistic, *Comput Linguist* **22**(2):249–254, 1996.
27. Schubert L, Tong M, Extracting and evaluating general world knowledge from the Brown corpus, in *Proceedings of the HLT-NAACL 2003 Workshop on Text Meaning*, Association for Computational Linguistics, Morristown, NJ, pp. 7–13, 2003.
28. Kilgarriff A, No-bureaucracy evaluation, *Proceedings of the EACL 2003 Workshop on Evaluation Initiatives in Natural Language Processing*, Budapest, Hungary, 2003.
29. Hripcsak G, Rothschild AS, Agreement, the F-measure, and reliability in information retrieval, *J Am Med Inform Assoc* **12**(3):296–298, 2005.
30. http://www.iclc.it/Listanuova.html/
31. http://www.biotech.ist.unige.it/cldb/cname-1c.html/
32. http://www.cabri.org/HyperCat/cells/CellLines.html/

33. Binder RV, *Testing Object-Oriented Systems: Models, Patterns, and Tools*, The Addison-Wesley Object Technology Series, Addison-Wesley Professional, Boston, 1999.
34. Cohen KB, Tanabe L, Kinoshita S, Hunter L, A resource for constructing customized test suites for molecular biology entity identification systems, *BioLINK 2004: Linking Biological Literature, Ontologies, and Databases: Tools for Users*, Association for Computational Linguistics, East Stroudsburg, PA, pp. 1–8, 2004.
35. Johnson HL, Cohen KB, Hunter L, A fault model for ontology mapping, alignment, and linking systems, *Pac Symp Biocomput* **12**:233–268, 2007.
36. Craven M, Kumlien J, Constructing biological knowledge bases by extracting information from text sources, *ISMB 1999*, 1999.
37. Lu Z, *Text mining on GeneRIFs*, Ph.D. thesis, University of Colorado Health Sciences Center, Denver, CO, 2007.

J. Gregory Caporaso is a Ph.D. candidate in the Department of Biochemistry and Molecular Genetics at the University of Colorado Health Sciences Center in Denver, CO. In 2001, he received a Bachelor's degree in Computer Science from the University of Colorado at Boulder, and shortly thereafter began research in bioinformatics, studying the evolution of the genetic code. After completing a second Bachelor's degree in Biochemistry in 2004, also from the University of Colorado at Boulder, Mr. Caporaso began his Ph.D. He is now studying coevolution in proteins, and is developing and applying text mining techniques to compile data for these studies. Mr. Caporaso's primary research interests include molecular evolution, structural biology, and biomedical text mining.

William A. Baumgartner, Jr. is a member of the Center for Computational Pharmacology at the University of Colorado School of Medicine, where he currently serves as Lead Software Engineer for the Biomedical Text Mining Group. He received his M.S.E. in Biomedical Engineering from Johns Hopkins University in 2001. His current research focuses on the application and evaluation of natural language processing technologies in the biomedical domain.

David A. Randolph is Senior Staff Software Engineer at Motorola Mobile Devices and an M.E. candidate in the Department of Computer Science at the University of Colorado at Boulder. His research interests include configuration management, computer-aided instruction, data mining, natural language processing, and bioinformatics. He has been with Motorola since 1999. Previously, he was Staff Software Engineer at IBM Printing Systems and a teacher in Illinois public schools. Mr. Randolph

earned a B.S. in Secondary Education from the University of Illinois at Urbana–Champaign in 1989, a B.S. in Computer Sciences from the University of Wisconsin–Madison in 1996, and a Postgraduate Certificate in Bioinformatics from the University of Illinois at Chicago in 2006.

K. Bretonnel Cohen leads the Biomedical Text Mining Group at Lawrence Hunter's Center for Computational Pharmacology.

Lawrence Hunter is Director of both the Computational Bioscience Program and the Center for Computational Pharmacology at the University of Colorado School of Medicine, and an Associate Professor in the Departments of Pharmacology, Computer Science (Boulder), and Preventive Medicine and Biometrics. He received his Ph.D. in Computer Science from Yale University in 1989, and then spent more than 10 years at the National Institutes of Health, ending as the Chief of the Molecular Statistics and Bioinformatics Section at the National Cancer Institute. He inaugurated two of the most important academic bioinformatics conferences, ISMB and PSB, and was the Founding President of the International Society for Computational Biology. Dr. Hunter's research interests span a wide range of areas, from cognitive science to rational drug design. His primary focus recently has been the integration of natural language processing, knowledge representation, and machine learning techniques, and their application to interpreting data generated by high-throughput molecular biology.