ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ № 4(22), 2014, с. 257–267

С. В. Знаменский

Моделирование задачи оптимального выравнивания последовательностей

Аннотация. Выравнивание последовательностей широко используется в различных компьютерных системах для анализа и оценки близости данных, выделения изменений и родственных задач. Интуитивные требования к постановке задачи оптимального выравнивания последовательностей формализованы в приводимых тестах. Тесты показали, что ни один из широко используемых подходов к близости и выравниванию не соответствует этим требованиям. Описана новая модель минимизации конфликтов при слиянии изменений. Модель приводит к простой постановке задачи оптимального выравнивания последовательностей, удовлетворяющей рассмотренным требованиям.

Ключевые слова и фразы: сходство строк, выравнивание последовательностей, расстояние редактирования, diff, LCS, метрика Левенштейна, разработка ПО, непрерывная интеграция.

1. Введение

Качественное выделение изменений позволяет обеспечить компактное хранение информации об истории изменений. Научный интерес к тематике выделения изменений иллюстрирует запрос к электронной библиотеке Google Scholar о научных публикациях 2014 года с фразой «sequence alignment», в ответ на который возвращается 42000 научных статей и книг. При этом базовые подходы и алгоритмы сформировались более 20 лет тому назад, канонизированы и реализованы в разнообразном ПО. К сожалению, эффективность работы ПО, выделяющего изменения не соответствует разумным ожиданиям. В частности, широко распространённая программа diff, десятилетиями стандартно использующаяся для сопоставления текстовых файлов, не может правильно сопоставить логи с трассировкой работы программы, а при сопоставлении сильно переработанных исходных текстов программ находит «неизменную» часть, состоящую из пустых строк.

Косвенным свидетельством этого факта является популярность нестандартных улучшений diff. Важнейшие из них отмечены в [1] со ссылками на источники.

Порочность ситуации не в общепризнанном выигрыше адаптированных алгоритмов, а в том, что ощутимый выигрыш в качестве достигается заменой оптимального алгоритма [2], тщательно выверенного за десятки лет широкого применения, на эвристический и по сути не опубликованный.

Адекватен ли критерий LCS(Longest Common Subsequence), заложенный в основу diff, тем областям, где он применяется? Этот вопрос особенно актуален в связи с тем, что этот критерий, как и родственная ему метрика Левенштейна, широко применяются для решения разнообразных задач:

- слияние изменений в исходных файлах в системах сборки ПО [1],
- синхронизация файловых систем [3],
- распознавание речи и голосов птиц и животных [4, 5],
- оценка объёмов и исследование структуры заимствований из других работ или документов [6],
- поиск мелодий в базе музыкальных записей [7]
- анализ генетических данных в биоинформатике [8],
- выделение различий между версиями при совместной подготовке текстов [9, 10],
- выделение сходных наименований сущностей при интеграции баз данных [11],
- диагностика сбоев и локализация ошибок в ПО [12].

Гипотеза о неадекватности математической постановки критерия максимальности длины LCS, на которую нацелен классический алгоритм, важнейшим её приложениям, косвенно подтверждается описанными ниже примерами.

2. Базовые классические подходы

Во всех упомянутых ситуациях сопоставление строк базируется на трёх основных подходах:

2.1. Выделение длиннейшей общей подстроки

Алгоритм Ратклифа–Мезнера Bdiff [13] (основа difflib, python, Revlog, Mercurial) производит поиск длиннейшей общей подстроки. Найденная подстрока делит каждую из строк на три части, средняя из которых

с ней совпадает. Далее применяется рекурсивно к начальным и к завершающим частям.

2.2. Расстояние редактирования

Нахождение кратчайшего описания того, как результирующая строка получена по исходной, восходит к работе В. И. Левенштейна [14]. Обычно длина такого описания называется метрикой Левенштейна и выражается через количество вставок и удалений элементов. Алгоритм Смита–Ватермана или Нидлмана–Вунша кроме количества вставок, замен и удалений учитывает длины цепочек соседних вставок или удалений. Часто отдельно учитываются замены. Используются различные зависимости размеров штрафов от длин цепочек. Могут дополнительно учитываться длины заменяемых/удаляемых участков (gaps). Суммарный штраф за длины цепочек минимизируется.

2.3. Выделение длиннейшей общей подпоследовательности LCS (Longest Common Subsequence)

В основе этой группы алгоритмов лежит цель сопоставить максимально возможное количество элементов заданных строк с сохранением порядка следования. Так действуют diff, GNU diffutils, пакеты Perl и ряд других программ. Подход напрямую связан с расстоянием редактирования простой формулой: длина LCS — это половина разности суммарной длины и суммарного количества удалений и замен.

2.4. Композитные подходы

Нередко используются вариации и комбинации этих трёх основных подходов: Так, например, Алгоритм Кохена, известный по bzrlib и patiencediff.py, выделяет [1] уникальные элементы и строится LCS для них, дальше производится рекурсия. Общим для подобных комбинаций и расстояния редактирования является слабость или полное отсутствие теоретического обоснования преимуществ. Положительный практический опыт не может заменить ясную теоретическую обоснованность, поскольку граница, за которой он станет резко отрицательным, может быть пересечена незаметно для пользователей.

3. Сравнительный анализ подходов

Автор не смог найти ни одного обзора или упоминания о сопоставлении базовых подходов к выделению различий в строках. Всюду сравниваются конкретные алгоритмы на основе статистической обработки результатов экспериментов или оценок сложности. Правильность выбора подхода поддержит комплект тестов, выявляющий особенности различных подходов.

На рисунках представлены тесты в виде сопоставления строк. Для наглядности результат правильного сопоставления показан зелёными стрелками, результат неправильного сопоставления — красными. Ниже это же сопоставление представлено как результат редактирования. Зачёркнутый текст — это удаление, подчёркнутый — это вставка.

3.1. Недостаток LCS и метрики Левенштейна

Основным хорошо известным недостатком diff и других наивно базирующихся на LCS или метрике Левенштейна программ является полная нечувствительность к фрагментации выделяемых изменений. Она приводит к нежелательным выделениям, показанным на рис. 1 и 2. Для наглядности сравниваются последовательности букв и пробелов, но сходный эффект наблюдается и при сравнении последовательностей иной природы. Например, для последовательностей строк исходного кода компьютерных программ он проявляется в смешивании фрагментов разных процедур на основе выявления общих строк с пробелами и скобкой.

3.2. Проблематичность альтернативных подходов

Пытаясь исправить этот бросающийся в глаза недостаток LCS и метрики Левенштейна, авторы альтернативных подходов впадают в противоположную крайность, концентрируя внимание на длиннейшем совпадении или на уникальных строках, что нередко помогает, но может и сильно помешать. На рис. 3 показан пример, в котором классический LCS явно срабатывает заметно лучше. Для более длинных строк разница может оказаться намного более значимой.

3.3. Слабость технологий биоинформатики

В биоинформатике метрика Левенштейна корректируется эмпирически подобранным штрафом за удаление или вставку последова-

(a) Лучшее сопоставление строк

(b) Сильная фрагментация

Рис. 1. Сравнение строк «определение перемещений» и «перемещение сбережений»

(a) Лучшее сопоставление строк

(b) Излишняя фрагментация

Рис. 2. Сравнение строк «практика математика» и «математика практика»

(a) Лучшее сопоставление строк

(b) Длиннейшая подстрока из уникальных элементов слабо значима

Рис. 3. Сравнение строк
«$ab + cd + ef + ad + cf + eb + x + y$» и
«$1 + x + y + a \cdot b + c \cdot d + e \cdot f + a \cdot d + c \cdot f + e \cdot b$»

тельно идущих символов. Штраф зависит от длины удаляемой или добавляемой строки. Зависимость подбирается эмпирически.

Представленный на рис. 4 пример однозначно доказывает, что никакая зависимость от длины удалений или вставок не может исправить вышеописанный недостаток метрики Левенштейна. Если все символы равнозначимы, то совпадение нескольких пар может быть случайным, в то время как совпадение длинной строки abcdef менее вероятно, и это может оказаться значимым. Используя более длинные строки, можно неограниченно усилить этот эффект. Никакой алгоритм, анализирующий только различия, такой тест не пройдёт.

4. Минимизация вероятности конфликтов при слиянии версий

Определение 1. Элементарное редактирование строки A непосредственно или косвенно затрагивает строку длины k в A. Эту строку назовём *контекстом элементарного редактирования*, а k — *размером контекста*.

В результате сопоставления исходной строки A длины $l(A)$ со строкой B длины $l(B)$ выделена общая для A и B подпоследователь-

Рис. 4. Сравнение строк «abcdefxgxhxixjxkxlxABCDEFGHIJKL» и «ABXCDXEFXGHXIJXKLXabcdefghijkl»

ность C. Пусть C состоит из m связных компонент $c_1, \ldots, c_m$ длиной $l(c_1), \ldots l(c_k)$ соответственно. Вероятность конфликта определяется размером k контекста.

Обозначим S_k количество подстрок A длины k, целиком лежащих в C:

$$S_k = \sum_{i=1}^{m} \phi(l(c_i) - k + 1), \text{ где } \phi(x) = \frac{x + |x|}{2}.$$

Вероятность отсутствия конфликта с контекстом размера k определяется как

$$p_k = \frac{S_k}{l(A) - k}.$$

4.1. Фиксированная длина контекста

По формуле полной вероятности вероятность избегания конфликта выражается формулой

$$p = \sum_k r_k \cdot p_k,$$

где r_k — относительная вероятность заданного размера контекста.

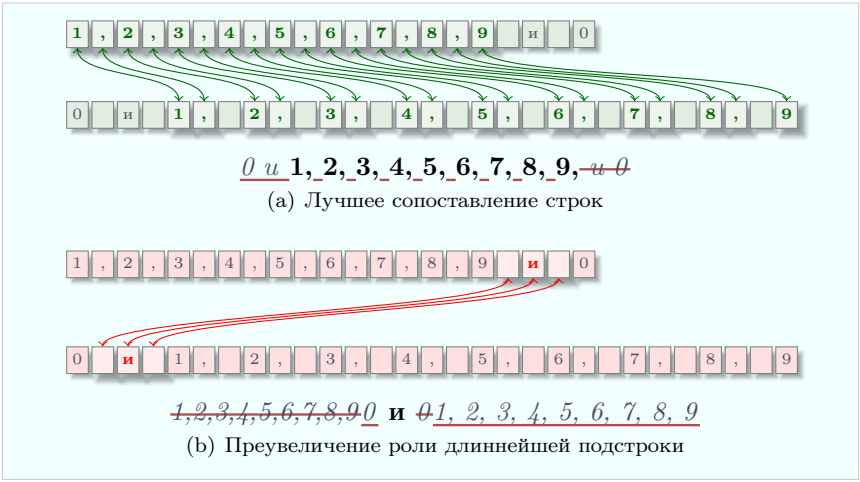

Рис. 5. Сравнение строк «1,2,3,4,5,6,7,8,9 и 0» и «0 и 1, 2, 3, 4, 5, 6, 7, 8, 9»

При $k = 1$ эта сумма пропорциональна длине LCS. Для $k = 2$ или $k = 3$ результаты лучше, чем при использовании LCS, но дополнительный пример на рис. 5 показывают необходимость включения в рассмотрение контекстных строк различной длины.

В предположении равновероятности всех возможных контекстных строк $r_k = 2\frac{l(A)-k}{k(k+1)}$ и справедлива

Теорема 4.1. *Вероятность избежания конфликта со случайной строкой выражается формулой*

$$p = \frac{1}{2k(k+1)}\left(\sum_{i=1}^{m}\left(l(c_i)+\frac{1}{2}\right)^2 + \frac{m}{4}\right).$$

Из этой теоремы легко получается численная характеристика качества выделенной подпоследовательности:

$$K = \sum_{i=1}^{m}(2l(c_i)+1)^2 + 1.$$

Таблица 1. Результаты тестирования подходов

№	Ключевой критерий модели	Номер рисунка				
		1	2	3	4	5
1	Количество вставок, замен и удалений	−	−	+	?	+
2	Длина S_1 общей подпоследовательности C	−	−	+	?	+
3	Длина общей подпоследовательности уникальных элементов	+	+	−	?	+
4	Наличие длиннейших совпадающих строк	+	+	−	+	−
5	Количество вставок, замен и удалений с учётом длин цепочек изменений	±	±	+	?	+
6	Количество S_2 подстрок в C длины 2	+	+	−	−	+
7	Количество S_3 подстрок в C длины 3	+	+	+	+	−
8	Оптимальный критерий K	+	+	+	+	+

4.2. Итоги тестирования

В таблице 1 представлена релевантность подходов рассмотренным тестам по следующим градациям:

+ — правильный вариант однозначно предпочтителен;
− — неправильный вариант однозначно предпочтителен;
± — правильный ответ на тест возможен при специфическом выборе параметров;
? — результат непредсказуем: критерии близости, на которых основана модель, не могут различить хороший и плохой вариант.

5. Заключение

Многообразие различных конкурирующих подходов с варьирующимися параметрами и сомнительной универсальностью вызвано отсутствием убедительной математической модели сопоставления строк. Предложенный набросок такой модели может стать ориентиром для создания качественных алгоритмов выделения изменений и сопоставления строк.

Список литературы

[1] Baudiš P., *Current concepts in version control systems*, 2014, arXiv: 1405.3496 ↑ 258, 259.

[2] Hunt J. W., McIlroy M. D., *An algorithm for differential file comparison*, Bell Laboratories, 1976, 7 pp. ↑ 258.

[3] MacDonald J., *Versioned file archiving, compression and distribution*, U.C. Berkeley, 1998, URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.9429 ↑ 258.

[4] Wieling M., Bloem J., Mignella K., Timmermeister M., Nerbonne J., "Measuring foreign accent strength in English. Validating Levenshtein Distance as a Measure", *The Mind Research Repository (beta)*, 2013, no. 1, URL http://openscience.uni-leipzig.de/index.php/mr2/article/view/41/30 ↑ 258.

[5] Wu X., Wu Zh., Jia J., Meng H., Cai L., Li W., "Automatic speech data clustering with human perception based weighted distance", *The 9th International Symposium on Chinese Spoken Language Processing*, ISCSLP 2014 (12–14 September 2014, Singapore), IEEE, pp. 216–220 ↑ 258.

[6] Luo L., Ming J., Wu D., Liu P., Zhu S., "Semantics-based obfuscation-resilient binary code similarity comparison with applications to software plagiarism detection", *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014 (November 16–21, 2014, Hong Kong, China), ACM, 2014, pp. 389–400, URL http://faculty.ist.psu.edu/wu/papers/cop-fse2014.pdf ↑ 258.

[7] Rho S., Hwang E., "FMF: Query adaptive melody retrieval system", *Journal of Systems and Software*, **79**:1 (2006), pp. 43–56 ↑ 258.

[8] Durbin R., Eddy S., Krogh A., Mitchison G., *Biological sequence analysis: probabilistic models of proteins and nucleic acids*, Cambridge University Press, 1998, 356 pp. ↑ 258.

[9] Dohrn H., Riehle D., "Fine-grained change detection in structured text documents", *Proceedings of the 2014 ACM symposium on Document engineering*, DocEng 2014 (September 16–19, 2014, Denver, Colorado, USA), ACM, 2014, pp. 87–96 ↑ 258.

[10] Oita M., Senellart P., "Deriving dynamics of web pages: A survey", TWAW 2011: Temporal Workshop on Web Archiving (March 28, 2011, Hyderabad, India), 2011, URL http://pierre.senellart.com/publications/oita2011deriving.pdf ↑ 258.

[11] Hall P. A. V., Dowling G. R., "Approximate string matching", *ACM computing surveys*, **12**:4 (1980), pp. 381–402 ↑ 258.

[12] Diamantopoulos T., Symeonidis A., "Localizing Software Bugs using the Edit Distance of Call Traces", *International Journal on Advances in Software*, **7**:1–2 (2014), pp. 277–288 ↑ 258.

[13] Mackall M., "Towards a Better SCM: Revlog and Mercurial", *Proceedings of Linux Symposium.* v. 2 (July 19–22, 2006, Ottawa, Ontario, Canada), 2006, pp. 83–90, URL http://mercurial.selenic.com/wiki/Presentations?action=AttachFile&do=get&target=ols-mercurial-paper.pdf ↑ 258.

[14] Левенштейн В. И., «Двоичные коды с исправлением выпадений и вставок символа 1», *Пробл. передачи информ.*, **1**:1 (1965), с. 12–25 ↑ 259.

Рекомендовал к публикации *к.т.н. Е. П. Куршев*

Об авторе:

Сергей Витальевич Знаменский

Автор критерия разрешимости уравнений свёртки в пространстве функций, голоморфных на множестве, понятий выпуклости в направлении и $\mathbb{C}$-выпуклости, русификации TEX для журналов Отделения математики РАН, графического пакета mfpic3d и ретроспективного подхода к построению информационных систем.

e-mail: svz@latex.pereslavl.ru

Образец ссылки на эту публикацию:

С. В. Знаменский. *Моделирование задачи оптимального выравнивания последовательностей* // Программные системы: теория и приложения: электрон. научн. журн. 2014. Т. 5, № 4(22), с. 257–267.

URL http://psta.psiras.ru/read/psta2014_4_257-267.pdf

Sergej Znamenskij. *Modeling of the optimal sequence alignment problem.*

Abstract. The sequence alignmet is widely used in variouse computer systems for data similarity measure and analisys, changes detection and relative tasks. Some intuitive requirements for string alignmet are formalised in a test suite. The tests shows that none of existing approaches to string similarities and alingment meet the requirements. A new model of minimizing conflicts when merging changes is described. The model leads to a simple formulation of new optimization problem which meet the requirements. (*In Russian.*)

Key Words and Phrases: similarity of strings, sequence alignmet, edit distance, diff, LCS, Levenshtein metric, software development, continuous integration.