ИСПРАВЛЕНИЕ ОРФОГРАФИЧЕСКИХ ОШИБОК С ПОМОЩЬЮ ПЕРЕБОРА, УПРАВЛЯЕМОГО МОРФОЛОГИЧЕСКИМ СЛОВАРЕМ

А. Гельбух

Научно-техническая информация, серия 2, номер 5, стр. 23–30, 1993. PDF.

Рассматривается задача исправления одиночной ошибки в слове, взятом вне контекста, с помощью морфологического словаря. Обсуждаются методы сокращения перебора. Основное внимание уделяется сокращению числа обращений к дисковой памяти и к алгоритму морфоанализа в процессе поиска, а также получению почти всех вариантов исправления на самой ранней стадии работы алгоритма. Приводится статистика числа обращений к словарю и алгоритму морфоанализа до первого, правильного, последнего предлагаемого варианта и до завершения поиска. Предлагаемый алгоритм по основным показателям превосходит известные.

Spelling Correction with Morphological Dictionary. *Nauchno-Tekhnicheskaya Informatsiya*, series 2, vol. 5, pp. 23–30, 1993.

ВВЕДЕНИЕ

В настоящее время каждая система обработки текста должна быть оснащена, кроме средств орфографической проверки текста, также средствами автоматизированного или даже автоматического исправления выявленных ошибок. Ввиду большой вычислительной сложности алгоритмов автоматизированной коррекции ошибок, задача повышения эффективности таких алгоритмов и улучшения полноты поиска вариантов исправления является весьма актуальной [1]. Алгоритмы исправления ошибок, пригодные для работы с текстом на русском языке, должны учитывать особенности русского языка как высоко флективного. Как и в ряде иных работ, под ошибкой ниже понимается вхождение в текст слова, отсутствующего в словаре. Таким образом, не рассматриваются стилистические ошибки, неправильные согласование или пунктуация, замена слова на цепочку букв, случайно совпавшую с существующей в языке словоформой.

Ошибки делятся на грамматические ошибки и опечатки. Между ними имеется существенная разница. Грамматические ошибки часто затрагивают несколько букв слова: **хочут*, **ребенки*, в то время как опечатки обычно затрагивают одну букву. Грамматические ошибки допускают объяснение и требуют выдачи его пользователю, поскольку он не знает правильного написания слова; опечатки не требуют объяснений. Грамматические ошибки указанного типа совершают, как правило, только учащиеся

или лица, плохо владеющие русским языком, опечатки же делаются всеми. В связи с этим задача исправления и объяснения грамматических ошибок представляется несколько специфической [2]. Данная работа посвящена только исправлению опечаток.

Формально под задачей исправления опечатки понимается следующее. Дано некоторое конечное множество слов конечной длины — множество допустимых слов, или словарь, — в некотором конечном алфавите и некоторое отношение близости на множестве всех слов в данном алфавите. Задача исправления опечатки сводится к нахождению для данного исходного слова всех допустимых слов, близких к нему. Заметим, что от критерия близости не требуется ни транзитивности, ни рефлексивности, ни симметричности. Конечно, реализация алгоритма коррекции опечаток в сильнейшей степени зависит от конкретного критерия близости. Не будем вдаваться здесь в подробное описание желательных его свойств; как правило, критерии, обычно применяемые в задаче коррекции опечаток, нужными свойствами обладают.

Наиболее важным типом опечаток считаются так называемые однобуквенные ошибки. Это четыре типа ошибок: вставку буквы (**опечаптка*), пропуск буквы (**опечтка*), замену буквы (**опечятка*) и перестановку двух соседних букв (**опечтака*). Наш алгоритм будет прежде всего ориентирован на исправление однобуквенных опечаток. Кроме однобуквенных, известно несколько типов часто встречающихся неоднобуквенных опечаток [1], например, перестановка согласных через гласную (**очепатка*). Наш алгоритм обрабатывает ряд неоднобуквенных опечаток, однако их конкретные типы не обсуждаются, поскольку они могут выбираться сравнительно произвольно при практической реализации алгоритма. Следует заметить, что большая часть грамматических ошибок (**карова*, **возьмуться*, **сонце*) подпадает под понятие однобуквенной ошибки и, таким образом, исправляется нашим алгоритмом.

Как правило, обсуждаемые в литературе алгоритмы исправления ошибок работают по следующей схеме. Для текстуального подозрительного слова порождаются гипотезы — буквенные цепочки, близкие к нему в смысле выбранного критерия близости. Затем каждая гипотеза независимо подвергается морфологическому анализу и проверке по словарю.

В работе [1] рассматривается набор эвристик, использующий специфические для конкретного (русского) языка соображения и экспериментальные данные, позволяющий сократить такой перебор, упорядочив определенным образом множество порождаемых гипотез и априорно отсеяв некоторые из них. Показана высокая эффективность приведенного метода.

В данной работе исследуется альтернативный подход - использование для отсева и упорядочения гипотез информации, имеющейся в лексикографически упорядоченном морфологическом словаре, при отказе от языково-специфических эвристик и от использования экспериментальных данных. Показана еще большая эффективность предлагаемого метода.

Вместе с тем, в работе [1] исследовался только вопрос, как ускорить получение правильного варианта исправлени Однако для практических целей важен также

вопрос сокращения общего времени работы алгоритма, при условии, что среднее время получения правильного варианта и даже всех вариантов не ухудшится. Вопрос минимизации общего количества гипотез без ухудшения показателей, приведенных в [1], также рассматривается в данной работе.

Дополнительно учитывается специфика работы со словарем, расположенным на жестком диске ЭВМ. Как правило, наиболее медленной операцией морфологического анализатора является не сампроцесс разбора слова, а считывание из словаря необходимого фрагмента. Поэтому для сокращения реального времени работы процедуры исправления существенно не столько общее число проверяемых гипотез, сколько степень их рассеяния по словарю. Сокращение количества дисковых операций в процессе работы алгоритма является объектом нашего особого внимани

КРИТЕРИИ ОПТИМИЗАЦИИ

Пусть имеется машинная процедура, на вход которой подается произвольная цепочка букв, а на выходе генерируется булево значение — допустимо или недопустимо. В процессе работы этой процедуры используется словарь — структура данных, особенности физической организации которой учитываются при оценке эффективности алгоритма. Такая процедура в дальнейшем называется процедурой сопоставления со словарем, или процедурой анализа. Задана еще некоторая простая процедура, вырабатывающая булево значение по двум буквенным цепочкам. Любые два слова, на которых эта процедура выдает значение истина, называются близкими.

По данному слову, не являющемуся допустимым, требуется найти все близкие к нему допустимые слова. Эти слова называются вариантами исправления исходного слова. Правильного варианта, т.е. слова, которое на самом деле имел в виду человек, среди них может и не быть. Заметим, что сам алгоритм не может установить, какой вариант является «на самом деле» правильным, поскольку все они равно допустимы по словарю.

Предполагается, что общая схема работы алгоритма является циклической. На каждом шаге формируется буквенная цепочка, отличающаяся от исходного слова, но близкая к нему в смысле рассматриваемого критерия близости (это очередная гипотеза). Гипотеза предъявляется процедуре сопоставления со словарем и, если она оказывается допустимым словом, добавляется к выходному набору найденных вариантов исправлени Затем на основании дополнительной информации, полученной в процессе работы процедуры сопоставления со словарем, принимается решение о целесообразности продолжения работы алгоритма, формируется следующая гипотеза, и процесс повторяется.

Будем рассматривать два основных показателя эффективности алгоритма: количество обращений к процедуре сопоставления слова со словарем и количество обращений к дисковой памяти, производимых этой процедурой.

Очевидно, число обращений к процедуре анализа равно числу порожденных гипотез. Второй критерий не допускает столь четкого определени Идея состоит в том, чтобы, зная физическую организацию словаря и стратегию его использования

процедурой анализа, помочь ей реже обращаться к дисковой памяти. Это может быть достигнуто, например, путем предъявления ей гипотез в определенной последовательности.

Общие предположения о структуре словаря таковы. Словарь представляет собой большой массив, хранимый на диске (или другом устройстве с блочным доступом, скажем, в расширенной памяти типа XMS IBM-совместимых персональных компьютеров). В каждый момент в оперативной памяти может находиться только относительно малый участок словаря, называемый далее блоком, причем считывание нового блока в память является самой медленной операцией алгоритма [3]. Предполагается, что при очередном обращении к процедуре анализа новый блок считывается только в том случае, если он не остался в памяти от предыдущего обращени Даже если сама процедура не обеспечивает такой проверки, подобная оптимизация обычно выполняется операционной системой, драйверами диска или XMS, либо распространенными специальными программами кеш-буферизации.

Предполагается, что словарь упорядочен лексикографически, так что слова, близкие в алфавитном порядке, находятся в одном и том же блока словар Таким образом, требование минимизации числа обращений к диску состоит в требовании при циклическом обращении к процедуре анализа группировать вместе гипотезы, близкие по алфавиту.

Рассмотрим теперь, как оценивать число гипотез и обращений к диску. Будем исходить из того, что в потенциальном приложении каждый найденный вариант исправления слова может быть немедленно предъявлен пользователю. Если пользователь получает правильный вариант, он вправе остановить работу алгоритма. При этом с чисто психологической точки зрения нежелательно, чтобы алгоритм слишком долго «думал» до выдачи хотя бы одного варианта. Необходимо также особо отметить возможность автоматической проверки того, является ли найденный вариант правильным, с помощью, например, синтактико-семантического анализатора, работающего синхронно с алгоритмом поиска вариантов.

Таким образом, кроме статистики, подсчитывающей время завершения работы алгоритма, не менее, а, возможно, и более важными представляются показатели, рассчитанные до нахождения первого, правильного и последнего варианта. Как видно из приведенного выше описания алгоритма, моментом окончания его работы является не момент нахождения последнего из вариантов, а момент, когда установлено, что других вариантов быть не может. Далее будет показано, что время нахождения всех вариантов обычно оказывается на порядок меньше, чем время установления того, что они найдены исчерпывающе.

УЛУЧШЕНИЕ СТАТИСТИКИ ДОСТИЖЕНИЯ КОНЦА ПОИСКА

Начнем рассмотрение с простого алгоритма, работающего на словаре упрощенной структуры. В дальнейшем структура словаря будет уточняться и усложняться.

Отвлекаясь от деталей, предположим, что словарь представляет собой просто лексикографически упорядоченный набор слов. Задача состоит в том, чтобы для

данной цепочки букв, не содержащейся в словаре, найти все слова из словаря, близкие к ней. В качестве критерия близости рассмотрим, снова отвлекаясь от деталей, только однобуквенные замены: два слова считаем близкими, если их длины одинаковы и они различаются ровно одной буквой.

Предполагается, что процедура поиска в словаре использует, например, хранящийся в памяти индексный массив файла словаря, по которому она находит блок словаря, содержащий потенциальное алфавитное место предъявленной ей буквенной цепочки.

Рассмотрим подлежащее исправлению ошибочное слово, поданное на вход нашего алгоритма. Оно не совпадает ни с каким словом словар Обозначим через D_1 номер позиции, по которой слово отличается от непосредственно предыдущего по алфавиту слова словаря, через D_2 - номер позиции несовпадения со следующим словом словаря, через d_1 и d_2 - буквы на этих позициях во входном слове. Будем называть позицией несовпадения D для данного слова большее из чисел D_1 и D_2, и ближайшей большей буквой — букву d_2, если $D = D_2$, а иначе - специальный признак конца алфавита. Для дальнейшего существенно, чтобы, кроме ответа на вопрос, содержится ли слово в словаре, процедура анализа могла выдать номер позиции D и букву d.

Уточним, что если несовпадения не произошло, т.е. предъявленное слово оказалось просто короче или длиннее соответствующего слова из словаря, позицией несовпадения D считается последня буква слова.

Например, если словарь содержит словоформы *мена, меню*, то для цепочки **мень* процедура вернет номер позиции несовпадения $D = 4$ и ближайшую букву d = "*ю*". Идея алгоритма состоит в том, чтобы при этом избежать испытания заведомо неправильной цепочки **менэ*. Однако алгоритму все равно придется испытывать слова *день, лень, пень, медь, мель, мена* и т.д. При методе полного перебора пришлось бы подставлять каждую букву алфавита вместо одной буквы на каждой позиции по очереди. Как уже отмечалось, оптимальным порядком предъявления гипотез процедуре анализа является алфавитный порядок. Поэтому будем перебирать гипотезы в алфавитном порядке, используя имеющуюся информацию для того, чтобы пропускать заведомо неверные гипотезы. Кроме алфавитно упорядоченного словаря, на практике применяются словари с древесной структурой. Процедура поиска в таком словаре перебирает буквы слова по одной и для каждой буквы делает попытку продвинуться по дереву к следующей букве. В этом случае также предполагается, что процедура может вернуть номер D позиции, на которой продвижение по дереву оказалось неуспешным, и ближайшую следующую букву d на этом уровне дерева.

Алгоритм 1.

Шаг 1. В качестве первой варьируемой позиции выбираем первую позицию цепочки.

Шаг 2 начинает цикл. Фиксируем варьируемую позицию V цепочки. В качестве первой буквы для замены v выбираем первую букву алфавита.

Шаг 3 начинает внутренний цикл. Заменяем букву на варьируемой позиции V на выбранную букву v и предъявляем полученную гипотезу процедуре анализа.

Если гипотеза найдена в словаре, выдаем ее в качестве варианта. Следующей буквой для замены v считаем букву, следующую по алфавиту после текущей, если текущая была не последней.

Если гипотеза не была найдена в словаре, рассмотрим возвращенные процедурой анализа номер D и букву d. Если $D > V$, мы ничего не узнали; выбираем следующего кандидата на замену v, как в случае найденной гипотезы. Если $D = V$, выбираем $v = d$, что приводит к уменьшению числа гипотез. Заметим также, что процедура анализа может вместо буквы возвратить признак достижения конца алфавита. Если $D < V$, v остается неопределенным.

Шаг 4. Для обеспечения правильного алфавитного порядка перебора гипотез следует организовать перебор текущей варьируемой позиции в две фазы — прямой и обратный проход. В момент, когда v сравняется с буквой, стоящей на данной позиции в исходном слове, вместо испытания гипотезы, совпадающей с исходным словом, следует рекурсивно применить алгоритм к оставшейся части слова, т.е. продвинуть текущую варьируемую позицию V вперед и перейти к шагу 2. Когда $D < V$, следует выйти из рекурсии путем продвижения текущей варьируемой позиции V назад, восстановить следующую букву для замены v на этой позиции и продолжить цикл переходом к шагу 3. Этим начинается обратный проход по позициям слова. Когда $V = 0$, алгоритм заканчивает работу.

Итак, рассмотрим букву v, выбранную для замены. Если она по алфавитному порядку меньше буквы, стоявшей на данной позиции в исходной неправильной цепочке, или если выполняется обратный проход по позициям слова, переходим к шагу 3. Если на прямом проходе по позициям буква для замены оказалась по алфавитному порядку больше буквы в исходном слове, входим в рекурсию. Запоминаем v в стеке, увеличиваем номер позиции и переходим к шагу 2. Если же v не было определено, поскольку был достигнут конец алфавита, либо признак конца алфавита был возвращен в качестве d из процедуры анализа, либо было $D < V$, то выходим из рекурсии: уменьшаем варьируемую позицию V, выбираем из стека ранее запомненную букву v и переходим к шагу 3, продолжая перебор на данном уровне рекурсии, прерванный ранее на прямом проходе.

Заметим, что если d не определена, вместо следующей по алфавиту буквы на шаге 3 можно взять $v = d_2$, если $D_2 = V$. Для работы алгоритма 1 необходим один стек из $N - 1$ байта для хранения запоминаемых при входе в рекурсию букв, где N не превышает как число букв в слове, так и число букв в самом длинном слове словар Дополнительной памяти для хранения проверяемой гипотезы не требуется, поскольку все изменения можно производить прямо в предъявленной цепочке, каждый раз возвращая измененную букву на место при смене текущей позиции.

Не имея возможности привести здесь пример работы алгоритма, мы предлагаем читателю вручную проследить его работу на произвольном слове, используя любой имеющийся под рукой словарь.

Следует особо подчеркнуть, что данный алгоритм, в отличие от [1] или [4], не требует абсолютно никаких априорных знаний о языке, составе словаря и механизме совершения ошибок в словах. Фактически не требуется даже знания алфавита данного

языка. Легко заметить, что на шаге 2 в качестве первой буквы алфавита может быть выбран просто первый символ кодового набора ЭВМ (0 или –1). В этом случае уже на следующей итерации цикла шага 3 будет найдено правильное значение, и только в редких случаях количество итераций шага 3 увеличится на 1. Таким образом, алгоритм правильно работает и в том случае, когда в словаре есть слова, содержащие небуквенные символы, например, OS/2, или когда состав и язык словаря вообще не известны заранее.

Легко видеть, что алгоритм фактически варьирует буквы только в пределах той начальной части слова, которая совпадает с начальной частью какого-либо слова из словар Часто это позволяет сократить исследуемый участок слова приблизительно вдвое. Кроме того, для работы алгоритма не требуется априорно определять конец слова в предъявленном потоке букв. Это может оказаться важным, когда в словаре есть слова, содержащие пробел и/или знаки препинания, например, и т.д. или во что бы то ни стало [2]. Как только что отмечалось, потребность в памяти при этом априорно ограничена максимальной длиной слова в словаре.

При полном переборе число гипотез равно $N \times L$, где N число букв в слове, а L — число букв в алфавите [1]. Алгоритм позволяет сократить примерно вдвое число просматриваемых позиций слова и в несколько раз – среднее число испытываемых букв алфавита. Более того, он не требует априорного знания ни N, ни L.

Подобный алгоритм, с учетом очевидных модификаций, обсуждающихся ниже и касающихся усложнения структуры словаря и необходимости искать ошибки разных типов, а не только типа замены, следует применять, когда найденные варианты используются только после того, как сформирован полный набор вариантов и алгоритм их поиска закончил работу. Данный алгоритм оптимизирует как количество обращений к процедуре анализа, так и количество обращений к дисковой памяти, рассчитанные до конца работы алгоритма.

УЛУЧШЕНИЕ СТАТИСТИКИ ДОСТИЖЕНИЯ ПРАВИЛЬНОГО ВАРИАНТА

Рассмотрим теперь число обращений к дисковой памяти, т.е. фактически время работы алгоритма, рассчитанное не до конца работы, а до начала и конца «видимой» ее части — до получения первого и последнего вариантов. Будет также рассматриваться момент получения правильного варианта. Улучшение этих показателей не должно существенно сказаться на общем времени работы алгоритма. Алгоритм 1 выдает варианты в произвольный момент в течение всего времени работы. Наша задача состоит в такой модификации этого алгоритма, чтобы он выдавал почти все варианты в начале своей работы, а уж потом занимался доказательством того, что других вариантов нет. Существует два способа такой организации работы. Первый состоит в том, чтобы сначала искать там, где есть наибольшая вероятность найти, а уж потом просматривать остальные участки пространства поиска «для очистки совести». Второй — в том, чтобы сначала искать там, где искать проще и быстрее, а потом там, где труднее и дольше. Первая стратегия, основанная на учете особенностей языка и

клавиатуры, подробно анализируется в [1]. В данной работе рассматривается случай, когда никакой априорной информации нет, поэтому выбирается второй путь: гипотезы должны подаваться на вход процедуры анализа в порядке возрастания времени, необходимого для их обработки.

Легко заметить, что все слова, близкие к исходному и отличающиеся от него по далеким позициям (т.е. имеющие с ним общий начальный участок существенной длины), расположены по алфавитному порядку близко друг от друга и, весьма вероятно, в одном с ним или близких блоках словар Напротив, слова, отличающиеся от исходного по первым буквам, расположены каждое в своем блоке, и анализ каждого из них требует считывания блока словар Искомая модификация алгоритма состоит в том, чтобы перебирать позиции букв слова не от начала к концу, а от конца к началу.

Алгоритм 2.

Шаг 1. Поскольку исходное слово было проверено и отвергнуто процедурой поиска в словаре, известен номер позиции несовпадения *D*. Очевидно, что никакие исправления на участке слова за этой позицией не приведут к допустимому слову. Поэтому ограничим перебор варьированием букв на позициях не больше данной. Во многих случаях это сразу отсекает большое число заведомо ложных гипотез, как, например, в слове **эликтрификация*. В качестве первой варьируемой позиции *V* выбираем, в отличие от алгоритма 1, именно эту максимальную позицию *D*. Тем самым максимизируется вероятность того, что первая проверяемая гипотеза будет найдена в блоке словаря, уже находящемся в памяти.

Шаг 2 начинает цикл. Фиксируем варьируемую позицию *V*. Будем перебирать буквы алфавита не от начала к концу алфавита, как в алгоритме 1, а от буквы, стоящей на данной позиции в исходном слове, до конца алфавита, и затем от начала алфавита до этой буквы. Такой порядок перебора также способствует оптимальному использованию уже имеющегося в памяти блока словар Таким образом, в качестве первой буквы для замены v выбираем букву, следующую по алфавиту за стоящей на текущей позиции в исходной цепочке.

Шаг 3 — обращение к процедуре анализа — почти совпадает с шагом 3 алгоритма 1. Единственное отличие состоит в том, что следующей буквой за последней буквой алфавита считаем первую букву алфавита, а концом перебора считаем момент достижения буквы, стоящей на данной позиции в исходном слове, с которой перебор и начиналс Таким образом, если процедура анализа возвратила в качестве *d* признак конца алфавита, следующей буквой для замены становится первая буква алфавита. Когда на этой второй фазе *d* больше буквы на данной позиции в исходной цепочке, перебор букв на данной позиции заканчиваетс

Шаг 4. Никакой рекурсии, в отличие от алгоритма 1, не требуется. Если перебор букв на шаге 3 не закончился, переходим к шагу 3. Если закончился, уменьшаем номер варьируемой позиции *V* и переходим к шагу 2. Если $V = 0$, заканчиваем работу.

Заметим, что если на шаге 3 $D_1 < V$, то перехода на первую букву алфавита не требуется.

Алгоритм не требует дополнительной памяти, кроме фиксированного количества переменных.

Поскольку алгоритм 2 не оптимален по отношению к общему числу обращений к диску, следует ожидать, что по сравнению с алгоритмом 1 он приводит к лишним обращениям. Действительно, такие лишние обращения могут возникать в момент нарушения алфавитного порядка гипотез, т.е. в момент обращения к процедуре анализа с гипотезой, лексикографически меньшей предыдущей. Из алгоритма 2 видно, что на тех его участках, где алфавитный порядок соблюдается, лишних обращений не происходит. Алфавитный порядок нарушается в алгоритме один раз на каждую позицию, при переходе от конца алфавита к началу в переборе заменяемых букв. Заметим, что при переходе от текущей позиции к предыдущей алфавитный порядок следования гипотез не нарушаетс

Итак, плата за отступление от оптимального порядка — не более $N-1$ лишних обращений к диску, где N — длина слова. На самом деле их гораздо меньше. В качестве N здесь можно взять среднюю длину общего участка всех слов одного блока, которая даже для очень крупного морфологического словаря обычно не превышает 3 – 4. Легко видеть, что слова, отличающиеся по позициям с большими номерами, находятся все в одном блоке и не требуют дополнительных обращений к диску.

В экспериментах, выполненных автором с использованием словаря [5], количество обращений к диску, производимое алгоритмом при поиске всех вариантов до выдачи последнего из них, оказалось в 7.5 раза в среднем и в 59 раз по медиане меньше общего количества обращений до конца его работы. Другими словами, обычно все варианты «выскакивали» сразу, после чего алгоритм «задумывался» и затем сообщал, что больше вариантов нет. В отличие от данного алгоритма, алгоритм 1 затрачивал столько же времени, но выдавал допустимые варианты иногда в начале, иногда в середине, а иногда в конце работы. Легко видеть, что моментом наибольшей продуктивности алгоритма 1 является момент наибольшей глубины рекурсии, очевидным образом зависящий от входного слова. В алгоритме 2 мы как раз начинаем с этого момента, независимо от вида исходного слова.

Первый найденный вариант «выскакивает» в среднем в 18.1 раза быстрее, чем алгоритм заканчивает работу, в то время как в алгоритме 1 среднее время ожидания - почти половина общего времени работы. Количество обращений к диску до нахождения правильного варианта в 13.1 раз меньше общего количества.

ПОИСК В СЛОВАРЕ

В предыдущих разделах мы ограничивались словарем в виде простого списка словоформ. Рассмотрим теперь работу с реальным морфологическим словарем. Пересматривать схемы алгоритмов 1 и 2 не нужно, уточним только логику работы процедуры морфоанализа. А именно, эта процедура должна возвращать точно такое же значение, как при использовании словаря полных словоформ.

Поскольку процедуры морфоанализа часто в конечном счете имитируют поиск по дереву алфавитно упорядоченного списка словоформ, это не представляет труда.

Проиллюстрируем данную идею на примере словаря, рассмотренного в [3], и процедуры поиска, рассмотренной в [2].

В [2] слово рассматривается как конкатенация своих морфов: *чит-а-ющ-его-ся*. Первая часть слова (*чит-*) называется основой, однако формально рассматривается как один из морфов, что, конечно, несколько не согласуется с принятой лингвистической терминологией, но удобно при обсуждении алгоритма. Морфы всех слов языка, стоящие на определенной позиции по порядку в словах, собраны в списки; так, четвертый список содержит морфы *-ий-*, *-ой-*, *-его-*, *-ого-*, *-ему-*, *-ому-* и т.д., соответствующие падежным окончаниям разных классов слов. Первый из списков содержит основы всех слов языка; он называется словарем основ и размещается на диске. Остальные называются списками морфов и размещаются в памяти. Все списки упорядочены по алфавиту (или, что то же самое, организованы в виде дерева). Процедура анализа ищет совпадающую с начальным участком слова цепочку в словаре основ (*чит-*), затем совпадающую с начальным участком оставшейся части слова цепочку в первом списке морфов (*-а-*) и т.д. до конца слова (*-ся*).

Допустимыми по данному словарю словами являются слова из некоторого подмножества множества цепочек, получающихся конкатенацией элементов разных списков во всех возможных сочетаниях, при условии выбора первого элемента из первого списка, второго — из второго и т.д. Это последнее множество назовем множеством, порождаемым по данному словарю. Например, цепочка *чита-ющ-ого-ся* принадлежит к такому множеству, но не принадлежит к подмножеству допустимых слов. В данной работе не рассматривается способ выделения допустимых слов; на самом деле при этом учитывается имеющаяся в словаре разнообразная морфологическая информаци Для работы алгоритма исправления опечаток процедура анализа должна возвратить номер позиции несовпадения *D* и букву *d* на этой позиции в слове, следующем за алфавитным местом предъявленного ей слова в упорядоченном множестве всех словоформ. Пусть вместо этого процедура просмотра отдельного списка морфов, включая словарь основ, может возвратить такую информацию для каждого отдельного списка. Пусть слово успешно прошло сравнение с несколькими первыми списками и не прошло сравнения с очередным списком (возможно, со словарем основ). Рассмотрим позицию несовпадения, считая от начала слова, а также букву на ней в следующем элементе этого списка.

Очевидно, это та позиция и та буква, которые были бы выбраны при поиске предъявленного слова в множестве, порождаемом данным словарем. Поскольку множество допустимых слов является его подмножеством, найденная позиция не меньше, а буква не больше искомых для множества допустимых слов. Следовательно, как видно из алгоритмов 1 и 2, найденную позицию и букву можно использовать вместо требуемых в этих алгоритмах. Конечно, при этом происходит некоторая потеря эффективности, но это, по-видимому, неизбежная плата за усложнение структуры словаря.

Если в последнем просмотренном списке нет элементов, больших, чем поступившая на сравнение с ним часть входного слова, в качестве следующей буквы (как и в случае словаря словоформ) процедура возвращает признак конца алфавита.

Перейдем теперь к организации поиска по большому дисковому файлу — словарю основ. Как было только что показано, задача нахождения информации о позиции несовпадения в сложно организованном морфологическом словаре сводится к исходной задаче нахождения такой информации в простом упорядоченном списке. Рассмотрим поиск в таком упорядоченном списке буквенных цепочек, а именно — в словаре основ, учитывая, однако, блочную структуру его хранени В [3] показано, что для обычного морфологического анализа одного слова достаточно считывания одного блока диска. Теперь, в контексте задачи исправления опечаток, рассмотрим хотя и не очень частый, но неприятный случай, когда алфавитное место предъявленной цепочки приходится между блоками словаря. В этом случае можно все же избежать считывания обоих блоков.

В качестве примера рассмотрим организацию доступа к файлу словаря, приведенную в [3]. Словарь разбит на блоки размером 512 байт. Для поиска нужного блока используется индексный массив, расположенный в памяти. В индексный массив включено первое слово каждого блока. При этом оно усечено до минимальной длины, позволяющей отличить последнее слово предыдущего блока от первого слова данного. Для единообразия доступа этот индексный массив также разбит на блоки, по которым составлен индексный массив второго уровня. При необходимости может быть построен индексный массив третьего уровня, и т.д. При поиске в единственном блоке индексного массива верхнего уровня ищется наибольшее слово, не превышающее данного, затем аналогично просматривается соответствующий найденному слову блок следующего индексного массива, и в конце концов считывается с диска блок словаря и производится поиск в нем. Специальная оптимизация, подробно рассмотренная в [3], позволяет найти всю необходимую для анализа слова информацию в одном считанном блоке.

Для алгоритма исправления опечаток необходимо знать букву d_2 на позиции несовпадения D_2 в следующем слове, которое может оказаться в следующем блоке. Однако для нахождения нужной буквы достаточно информации, имеющейся в индексном массиве. Заметим, что в нашем случае следующее слово — это как раз первое слово следующего блока, т.е. следующее слово индексного массива первого уровня. Обозначим последнее слово данного блока через W_1, первое слово следующего блока через W_2, и следующее слово индексного массива через W'_2. Пусть W_1 совпадает с W_2 по первым $k - 1$ позициям, тогда слово W'_2 является начальным отрезком слова W_2 длины k. Поскольку входное слово процедуры анализа расположено между W_1 и W_2, оно совпадает с W_1, W_2 и W'_2 по (k – 1)-й позиции, и k = D_2 - позиция несовпадения. Таким образом, в качестве буквы d_2 следует взять соответствующую букву индексного массива. Заметим, что процедура поиска в индексном массиве не меняется, поскольку D_2 и d_2 — это как раз позиция и буква несовпадения, найденные обычным образом при просмотре индексного массива.

Конечно, ситуация, когда слово попадает на границу блоков, может встретиться и при работе с индексным массивом. Однако, поскольку обсуждаемая процедура является рекурсивной, она автоматически поступит правильно: в качестве D_2 и d_2 снова будут использованы значения, найденные на предыдущем шаге рекурсии, при поиске

в индексном массиве второго уровня. Итак, при поиске в словаре, разбитом на блоки, рекурсивная процедура, рассмотренная в [3], должна дополнительно выдавать либо букву d_2 и позицию D_2 несовпадения со следующим словом списка, если такое слово имеется в данном блоке, либо значение, найденное на предыдущем шаге рекурсии — при поиске в индексном массиве предыдущего уровня. Считывания дополнительного блока не требуется.

ИСПРАВЛЕНИЕ ОПЕЧАТОК РАЗНЫХ ТИПОВ

До сих пор рассматривалось, для простоты, исправление опечаток только одного типа - замена буквы. Конечно, реальный алгоритм должен, как минимум, исправлять также опечатки типа пропуска, вставки и перестановки соседних букв. Кроме того, в [1] рассматривается несколько типов неоднобуквенных опечаток, например, перестановка гласных букв через согласную или согласных через гласную, которые, как показывают исследования, встречаются относительно часто.

Заметим сначала, что для работы алгоритмов 1 и 2 не требуется, чтобы гипотезы отличались ровно одной буквой от исходного слова. Фактически, эти алгоритмы могут быть обобщены на любой тип опечаток. В алгоритмах 1 и 2 первым действием шага 3 является порождение гипотезы, отличающейся от исходного слова по текущей варьируемой позиции V на выбранную букву для замены v. Порожденная гипотеза не обязательно должна совпадать с исходным словом по позициям, которые больше V. На самом деле, вместо замены буквы можно производить вставку, что сразу дает алгоритм исправления опечаток типа пропуска буквы. Если буква исходного слова на позиции $V + 1$ совпадает с v, можно проверить гипотезу перестановки букв. Можно также проверить любые гипотезы с неоднобуквенным исправлением, которое затрагивает только позиции слова начиная с V и дает на позиции V букву v.

Алгоритм 3, являющийся модификацией алгоритма 2, иллюстрирует параллельное исправление опечаток разных типов. Основными требованиями, предъявляемыми к порядку перебора гипотез, остаются алфавитный порядок перебора гипотез для одной варьируемой позиции и перебор варьируемых позиций от конца слова к началу. Гипотезой для данной позиции V и буквы v может быть выбрана любая буквенная цепочка, совпадающая с исходным словом по позициям, которые меньше V, и имеющая на позиции V букву v. Конечно, множество таких гипотез должно быть разумно ограничено выбором типов неоднобуквенных опечаток, которые должен исправлять алгоритм. О выборе типов исправляемых неоднобуквенных опечаток см. в [1].

Приведем лишь отличия усовершенствованного алгоритма от алгоритма 2.

Алгоритм 3.

Шаг 1 - выбор начальной варьируемой позиции — тот же.

Шаг 2. Выбор очередной варьируемой позиции V производится как в алгоритме 2. После того, как она выбрана, составляется список простых гипотез для этой позиции: из слова удаляется одна буква, переставляются соседние буквы, а также порождаются неоднобуквенные гипотезы всех необходимых типов, с изменением буквы на позиции

V и без изменения букв левее V. В среднем в таком списке должно быть две-три гипотезы.

Шаг 3. Фиксируется буква для замены v. Порождаются две гипотезы путем замены буквы позиции V на v и вставки буквы v на позицию V. Из списка, составленного на шаге 2, выбираются гипотезы, имеющие на позиции V данную букву v. Затем гипотезы проверяются в алфавитном порядке. В качестве новой буквы d оставляется последняя полученная.

Шаг 4. Следующая буква или позиция выбираются как в алгоритме 2, и, если нужно, производится переход к шагу 2 или шагу 3.

Алгоритм 3 требует дополнительной памяти для хранения списка простых гипотез. Размер дополнительной памяти в среднем составляет несколько более $3N$, где N — минимум из длины предъявленного слова и длины самого длинного слова словар Обычно в списке бывает две гипотезы, поскольку неоднобуквенные гипотезы встречаются нечасто.

Однако, если все необходимые гипотезы вычислять каждый раз заново на шаге 3, дополнительной памяти не потребуется. Поскольку сложность такого вычисления невелика, подобная модификация алгоритма может оказаться предпочтительной.

Количество гипотез, подлежащих словарной проверке, всего лишь примерно вдвое превышает аналогичный показатель для алгоритма 2. Действительно, вместе с каждой гипотезой типа замены буквы теперь проверяется и гипотеза об ошибке типа пропуска буквы. Два других типа гипотез — перестановка и вставка — дают дополнительно не более $(2N - 1)$ проверок, что составляет небольшую часть от общего числа. Неоднобуквенные гипотезы также не увеличивают существенно числа проверок, поскольку их число также пропорционально N.

Как показывают эксперименты, число обращений к дисковой памяти увеличивается менее чем вдвое по сравнению с алгоритмом 2.

Экспериментальная проверка алгоритма 3 проводилась с использованием морфологического анализатора [2] с алгоритмом доступа к словарю [3]. Использовался морфологический словарь, составленный на основе словаря [5]. Проверка производилась на массиве реально встретившихся в текстах опечаток, любезно предоставленном автору для этой цели проф. И. А. Большаковым. Данный массив является в точности тем массивом, для которого приводится статистика в [1], что позволяет сравнить эффективность предлагаемого подхода с методами, рассмотренными в [1].

При контрольном прогоне программы для чистоты эксперимента рассматривались только однобуквенные гипотезы — замены, вставки, пропуски, перестановки соседних букв. В контрольном массиве ошибочных слов было 507 слов. Из них для 482 слов был найден правильный вариант исправлени Правильность варианта устанавливалась экспертом на основе анализа контекста. Для 21 слова программа не смогла найти ни одного варианта исправления — это неоднобуквенные опечатки типа **выячвлялясь* вместо *выявлялась*. В семи случаях программа, наоборот, не признала слово ошибочным — например, *исковых* вместо *искомых*. В 4 случаях среди предложенных

вариантов не было правильного — это также неоднобуквенные ошибки типа **коет* вместо *койот*.

Медианы и средние значения показателей								
	до первого варианта		до правильного варианта		до последнего варианта		до конца поиска	
	медиана	среднее	медиана	среднее	медиана	среднее	медиана	среднее
Гипотезы	16 15	26.65 25.37	19 18	33.25 30.76	26 26	47.97 42.24	215 149	215 150
Диск	1 1	6.61 5.69	2 2	9.21 7.64	2 2	16.78 13.17	118 76	120 77
Варианты			1	1.40	1	2.15		
Время	0.27 0.16	0.59 0.40	0.38 0.22	0.74 0.49	0.49 0.33	1.06 0.70	3.96 2.53	4.70 2.88

В таблице показаны средние значения и медианы распределения следующих параметров: количества проверенных по словарю гипотез, количества считанных с диска блоков словаря, количества найденных вариантов исправления и затраченного времени. Каждый параметр подсчитывался до выдачи программой первого найденного варианта, до выдачи правильного варианта, до выдачи последнего из найденных вариантов, и до конца поиска, т.е. до момента установления того, что других вариантов быть не может. К предмету данного параграфа статьи имеет отношение первая строка каждого пункта таблицы. Первое число в каждой паре представляет собой медиану соответствующей величины, второе среднее значение.

Время работы программы (в секундах) приводится только для сравнения. Эксперименты проводились на сравнительно медленной машине класса PC/XT. Не ставилось цели оптимизировать программу в рамках данного алгоритма.

В работе [1] приводится только один из наших показателей - число гипотез, испытанных до получения правильного варианта. Наилучший достигнутый показатель — медиана 23 и среднее 49.5. Как видно из таблицы, наличие словаря действительно приводит к лучшим цифрам — 19 и 33.25. Однако основным показателем эффективности метода мы считаем количество обращений к диску. Как видно из таблицы, для нахождения всех вариантов по медиане достаточно двух обращений к диску, а 118 обращений нужно, чтобы убедиться, что других вариантов нет.

Недостатком нашего подхода, по сравнению с подходом [1], является отсутствие ранжирования вариантов по вероятности быть правильным. Однако, возможно, лучше быстро получить все варианты, чем долго ждать самый вероятный. Поскольку среднее число вариантов равно приблизительно двум, и они выдаются нашим алгоритмом с ничтожной задержкой, такое ранжирование, можно провести апостериорно, на основании методов, рассмотренных в [1].

В проведенных автором экспериментах по апостериорному ранжированию вариантов подсчитывались показатели только для тех слов, для которых было найдено более одного варианта. Без ранжирования в среднем порядковый номер правильного варианта был 1.80, и лишь в 43.3% случаев правильный вариант выдавался первым и в 34.6% - вторым. Ранжирование производилось на основании методов, изложенных

в [1, 6, 7], а также разработанных автором. После ранжирования средний номер правильного варианта составил 1.37, правильный вариант был первым в 78.0%, вторым в 16.6% и третьим в 2.0% случаев. Однако необходимость такого ранжирования не представляется нам очевидной, поскольку без искусственного отсева слов, для которых был выдан только один вариант, т.е. при учете всего массива искаженных слов, правильный вариант выдавался первым в 83.3% случаев до ранжирования и в 93.3% — после.

В практическом приложении, по-видимому, стоит применить следующую тактику. После получения первого варианта задержать его выдачу пользователю на время, примерно равное времени его ожидания, и, если за это время будет найден второй вариант, провести апостериорное ранжирование. Как показывает анализ данных, приведенных в таблице, потери времени при этом будут относительно невелики, а варианты получатся почти всегда ранжированными.

БЕССЛОВАРНЫЙ ОТСЕВ ГИПОТЕЗ

Вновь рассмотрим задачу улучшения показателей до завершения поиска. Ситуация парадоксальна: обычно все варианты исправления бывают получены практически сразу, а основное время тратится на установление того, что других вариантов нет.

Как было показано, статистику, связанную с получением первого, правильного и даже последнего варианта исправления, вряд ли можно существенно улучшить какими-либо дополнительными методами отсева гипотез. Однако стратегия выбора следующей буквы v в алгоритме 1, весьма эффективная при работе с большей частью слова, становится крайне неэффективной при варьировании первых двух-трех его букв, поскольку процедура анализа в этом случае почти всегда возвращает значение *d*, равное текущей букве v, и в качестве следующей буквы приходится брать просто следующую по алфавиту. Вместе с тем, на фазу варьирования первых двух-трех букв слова падает основная часть затрат общего времени работы алгоритма 2: каждая новая буква из первых двух-трех букв слова, как правило, требует считывания нового блока словаря.

Таким образом, для сокращения общей продолжительности работы алгоритма крайне желательно отсеять часть гипотез, варьирующих первые буквы слова, до обращения к словарю. Для этой цели могут быть применены бессловарные методы отсева, такие, как диграммный и триграммный контроль [1].

Модификация алгоритма 1 состоит в том, что на шаге 3, перед обращением к процедуре анализа, гипотеза подвергается три- или диграммному контролю. Если в ней найдено недопустимое сочетание букв, обращения к процедуре анализа не происходит, а в качестве следующей буквы v берется следующая по алфавиту буква.

Однако имеются два соображения в пользу того, чтобы применять такой отсев не на всех этапах работы алгоритма, а только на этапе варьирования первых двух-трех букв слова.

Во-первых, получение массива триграмм по крупному морфологическому словарю требует синтеза всех словоформ. Гораздо проще собрать статистику по первым буквам основ, хотя синтезировать словоформы с короткими основами все же приходится.

Во-вторых, что гораздо существеннее, массивы триграмм и особенно диграмм, собранные только по первым буквам слов, получаются более разреженными, и отсев по ним происходит более интенсивно. Существенным преимуществом общепринятого метода перед этим последним является возможность в некоторых случаях сузить интервал поиска опечатки, что позволяет в таких случаях не варьировать первые буквы слова вовсе. По данным, приведенным в [1], диграммный контроль позволяет сузить интервал поиска в 11% случаев. Конечно, оптимальным является двойной контроль: на всем протяжении слова — по триграммам, собранным по всей длине слов словаря, и на начальном участке слова — по триграммам, собранным по первым буквам слов словаря.

Проводились эксперименты с триграммным контролем первых букв слова. Результаты даны в таблице во второй строке каждого пункта. Как и ожидалось, существенно изменились только показатели, рассчитанные до конца поиска, зато в этом случае улучшение значительно.

Недостатком метода является как дополнительный расход памяти, так и необходимость получения и хранения дополнительной информации, кроме обычного морфологического словаря, и обновления ее при каждом изменении словаря.

ЗАКЛЮЧЕНИЕ

В работе рассмотрены методы исправления опечаток на основе применения морфологического словар Недостатком такого подхода по сравнению с подходом, изложенным в [1], является наличие дополнительных, правда, весьма простых, требований к процедуре морфологического анализа. Достоинством является отказ от использования эмпирических языково-зависимых данных. Показана большая эффективность предложенного метода. При типовых условиях предложенный метод позволяет найти правильный вариант исправления, произведя в 1,21 раза по медиане и в 2,12 раза в среднем меньше испытаний. Общее время исчерпывающего поиска допустимых вариантов, по-видимому, значительно меньше, чем для алгоритма [1] (в [1] данный показатель не приводится). Время нахождения первого, правильного и последнего вариантов исправления однобуквенной опечатки представляется близким к возможному минимуму, в особенности если время работы алгоритма пропорционально не числу проверяемых гипотез, а числу обращений к дисковой памяти.

Метод оптимизирует количество гипотез и обращений к диску как до конца поиска, так и до нахождения правильного варианта. Особенностью предложенного алгоритма является то, что он почти всегда очень быстро выдает все возможные варианты исправления, и только после этого тратит основную часть времени на доказательство того, что других вариантов нет. Иными словами, при том же общем времени работы варианты, как правило, выдаются не в середине работы алгоритма, а в самом начале.

Мы видим перспективы дальнейшего совершенствования изложенного подхода в комбинировании его с методами, рассмотренными в [1, 4], в особенности при переборе вариантов замены первых двух-трех букв слова. Наш алгоритм может исправлять неоднобуквенные ошибки заранее заданных типов, однако не рассматривался вопрос выделения конкретных типов таких ошибок и не определялась эффективность работы алгоритма при исправлении неоднобуквенных опечаток.

Автор выражает глубокую благодарность проф. И. А. Большакову, осуществлявшему общее научное руководство работой и любезно предоставившему материалы для сбора статистики, а также проф. Г. Г. Белоногову за внимание к предмету исследования и ряд ценных замечаний. Автор благодарит А. Антонова за полезное обсуждение проблемы апостериорного ранжирования вариантов.

Литература

1. Большаков И. А. Минимизация перебора альтернатив при автоматизированном исправлении искаженных слов // Семиотика и информатика. 1990. — вып. 31. — С. 124–149.
2. Гельбух А. Ф. Эффективно реализуемая модель морфологии флективного языка // НТИ. Сер. 2.— 1992. — N 1.
3. Гельбух А. Ф. Минимизация количества обращений к диску при словарном морфологическом анализе // НТИ. Сер. 2. — 1991. — N 6.
4. Kernighan M. D., Church K. W., Gale W. A. A spelling correction program based on a noisy channel model // Proc. of COLING'90. vol. 2. — Helsinki, 1990. — P. 205–210.
5. Зализняк А. А. Грамматический словарь русского языка. Словоизменение.- М.: Русский язык, 1987. — 878 с.
6. Большаков И. А. Проблемы автоматической коррекции текстов на флективных языках // Итоги науки и техники. Теория вероятностей. Математическая статистика. Техническая кибернетика. Т.28. — М.: ВИНИТИ, 1988. — С.111–139.
7. Партыко З.В. Анализ искажений, возникающих при вводе текстов в ИИС «Ассистент» // НТИ. Сер.1. — 1982. — N 1. — С. 21–26.