Проект по курсу "Сложности вычислений". Алгоритм составления кроссворда.

Жарков Андрей, МФТИ

8 апреля 2017 г.

1 Введение.

В данной работе рассматривается задача составления кроссворда, то есть необходимо для фиксированного поля и словаря сделать кроссворд или сообщить о невозможности составления. В работе объясняется почему эта задача является сложной и как её можно решать за разумное время.

2 Формулировка задачи.

Условие: Заданы конечное множество слов $W \subset \Sigma^*$ и матрица A из нулей и единиц размером $n \times n$.

Вопрос: Можно ли составить кроссворд размером $n \times n$ из слов множества W так, чтобы слова записывались в клетки, соответствующие нулям в A? Другими словами, если E - множество пар (i, j), таких, что $A_{ij} = 0$, то существует ли отображение $f : E \to \Sigma$, такое, что буквы, приписываемые любой горизонтальной или вертикальной последовательности (максимальной длины) элементов из E, образуют слово из W? Образуемые слова при этом не должны повторяться.

3 Анализ сложности.

Теорема. Задача является NP-полной.

Для доказательства будем пользоваться NP-сложностью задачи о точном покрытии 3-множествами, которая формулируется следующим образом:

Условие: дано конечное множество X, $|X| = 3q$ и семейство C его трёхэлементных подмножеств.

Вопрос: содержит ли C точное покрытие X, т. е. такое подсемейство $C' \subset C$, что каждый элемент из X содержится ровно в одном элементе из C'

Доказательство теоремы.

Сведём к задаче составления кроссворда задачу о точном покрытии 3-множествами, NP-сложность которой общеизвестна.

► Итак, дано конечное множество X, $|X| = 3q$ и семейство C его трёхэлементных подмножеств.

Возьмём $\Sigma := X$. Таким образом, элементы X будут буквами алфавита Σ.

Каждое множество $\{x_{i_1}, x_{i_2}, \dots x_{i_k}\}$ из букв X будем отождествлять со словом из этих букв (для определённости, в алфавитном порядке).

В W включим:

1. все слова из 3х различных элементов X, т. е. все слова вида $x_i x_j x_k$, где $i < j < k$;
2. для каждого множества из C добавим слово, ему соответствующее, т. е. $\forall \{x_i, x_j, x_k\} \in C$ -> $x_i x_j x_k$ (i < j < k);
3. все слова вида $x_i x_i$, то есть все слова из двух одинаковых букв.

В качестве матрицы A выберем матрицу размера $4q \times 2$, в столбцах, номер которых делится на 4, стоят единицы, а в остальных клетках нули.

Все эти преобразования, очевидно, можно сделать за полином.

Теперь, если кроссворд на полученных A и W удалось составить, значит в этом заполнении использованы все двубуквенные слова (ибо таких слов в словаре 3q как и мест для них в кроссворде). Так как все двубуквенные слова состоят из разных букв, множества из C, соответствующие словам в блоках, не пересекаются. Так как были использованы все буквы, объединение всех множеств, соответствующих словам в блоках есть X. Таким образом, найдено точное покрытие X из C.

Обратно, пусть есть точное покрытие X - обозначим его $C' \in C$. Тогда в каждый блок поставим слова, соответствующие разным множествам из C'. Другие точно такие же слова были добавлены в W на шаге 1. Из выбранных слов, с добавлением двубуквенных можно составить кроссворд.

Пример заполнения кроссворда при $q = 3$:

x_1	x_2	x_3	0	x_4	x_5	x_6	0	x_7	x_8	x_9	0
x_1	x_2	x_3	0	x_4	x_5	x_6	0	x_7	x_8	x_9	0

Таким образом, задача о точном покрытии 3-множествами сведена к задаче составления кроссворда. Следовательно, задача составления кроссворда, NP-трудная. В то же время, так как правильность составления кроссворда можно проверить за полином, она лежит в NP. Итак, задача составления кроссворда NP-полная. ■

4 Алгоритм нахождения решений.

Решения можно найти перебором. Ставим первое слово, затем все последующие, проверяя, что буквы на пересечениях совпадают с поставленными ранее. Однако это будет очень долго. Положим, кроссворд состоит из 50 слов и слов в словаре любой длинны 1000. Тогда, для установки любого слова, у которого хотя бы одна буква фиксированна, потребуется в среднем около 1000/33 итераций. Таким образом, для заполнения 50 слов, даже если всегда будут находиться варианты для установки (то есть не придётся откатываться и перевыбирать предыдущии слова) потребуется порядка 30^{49} итераций, что непозволительно много. Приведённый ниже алгоритм поможет сократить время рассчёта на много порядков. Алгоритм рассчитан на сетки большой сложности, такие как канадские кроссворды, где каждое слово имеет пересечение на каждой букве, или близкие к ним.

Во-первых, составление кроссворда разобъём на 2 этапа:

1)Анализ - определение последовательности, в которой будут генерироваться слова, а также вычисление различных вспомогательных данных.

2)Генерация - собственно процесс заполнения сетки кроссворда, методом полного перебора всех возможных вариантов, с учётом данных, полученных на этапе анализа.

Итак, начнём.

Простейшие соображения

РЕШЕНИЕ 0. Словарь храниться как отображение (длинна слова -> массив таких слов). Все массивы случайно перемешиваются до начала генерации. На этапе генерации сначала выбирается 1й элемент массива, когда выясняется, что это слово не подходит - следующий, и т. д.

РЕШЕНИЕ 1. Слова будут генерироваться в последовательности, зависящей в первую очередь от их длины. В случае совпадения длин, сначала будет генерироваться слово с большим количеством пересечений. Это решение уменьшает вероятность отката на поздних стадиях генерации, ведь намного легче выбрать короткое слово или слово пересекающееся с небольшим числом других слов, чем наоборот. Решение используется на этапе анализа.

РЕШЕНИЕ 2. С учётом решения 1 не гарантируется пересечение двух соседних слов в последовательности генерации. Поэтому если мы не нашли слово для установки стоит возвращаться не к выбору предыдущего слова, а к выбору последнего слова, пересекающего данное. Решение используется на этапе анализа.

Фрагменты

Если отследить установку слов, в порядке, основанном на решении 1, можно наблюдать образование локально независимых фрагментов. Посмотрите на рисунок ниже:

На рисунке первым будет установлено слово, помеченное красным, затем слово, помеченное жёлтым. Как видим, после установки этих слов, образуется фрагмент, помеченный синим, установка слов в котором никак не связана с установкой оставшихся (белых) слов.

При нахождении фрагмента можно идти разными путями - либо сначала сгенерировать все слова фрагмента, проверяя правильность установки слова, после установки которого он образовался, либо продолжать генерировать слова в ранее заданной последовательности. И в том и в другом случае есть свои преимущества и недостатки.

РЕШЕНИЕ 3. В случае обнаружения фрагмента, все слова принадлежащие одному фрагменту будут генерироваться последовательно друг за другом (разумеется, порядок определяется дальнейшим поиском фрагментов и решением 1). Решение используется на этапе анализа.

Проверено, на несложных сетках алгоритм действительно работает. Однако с повышением сложности кроссворда время генерации становится совершенно не приемлимым.

Шаблоны

Хотя полученный промежуточный алгоритм работает намного быстрее примитивного перебора, он всё ещё обладает существенными недостатками. Во-первых, мы по-прежнему должны проходить по всем словам данной длины в поисках слова с нужными буквами на нужном месте. И во-вторых, устанавливая новое слово, мы не проверяем, остаются ли варианты для установки слов, которые пересекают текущее. Если бы мы сразу узнали, что установка данного слова заведомо не оставляет вариантов установки для некоторого другого слова, это бы сэкономило десятки или даже сотни тысяч итераций или более.

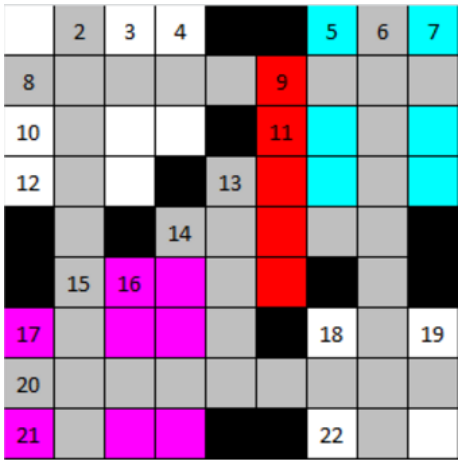
Это и наводит на мысль о шаблонах. Будем хранить отображение ставящее в соответствие каждому регулярному выражению (например ’З..ЛЯ’ - регулярное выражение, задающее все слова из пяти букв, начинающиеся с З и заканчивающиеся на ЛЯ) набор слов словаря (например, в виде бора), удовлетворяющих этому регулярному выражению. Это позволит преодолеть озвученные выше недостатки.

Таким образом мы пришли к

РЕШЕНИЕ 4. Для каждого устанавлимаемого слова рассчитываются шаблоны всех пересекающихся с ним слов (если такие шаблоны ещё не рассчитаны). Если для какого-то из рассчитанных шаблонов не окажется слов, ему соответствующих - это слово не подходит для установки.

Акселераторы

Во время заполнения поля, встречаются слова, пересекающиеся с фрагментом только один раз. Посмотрите на рисунок:



Серым выделены уже заполненные слова. Красным - текущее заполняемое слово, при его установке, как видим, образуются два новых фрагмента (выделены фиолетовым и голубым). Заметим, что с фиолетовым фрагментом устанавливаемое слово пересекается лишь по одной букве. Допустим теперь, что фиолетовый фрагмент не получилось заполнить. Значит, какое бы красное слово мы не устанавливали, в нём не должна быть та же буква, что и установленная в данный момент - иначе решения точно не будет. Таким образом, можно значительно ускорить генерацию и сэкономить миллионы итераций.

Назовём слово, после установки которого появляется фрагмент, и пересекающее этот фрагмент только одной буквой *акселератором*.

РЕШЕНИЕ 5. Для каждого устанавливаемого слова будем проверять (на этапе анализа), является ли оно акселератором и если да - для каких фрагментов. На этапе генерации в случае отката к слову-акселератору, запоминаем букву, стоящую на месте пересечения с фрагментом. При следующих выборах слова-акселератора будем проверять, что на местах пересечения не стоит та же буква.

Все перечисленные выше решения были повзаимствованы из [2]

Отсечение вариантов

Все перечисленные выше решения позволяют добиться неплохой скорости в среднем, однако достаточно часто генерация может идти очень долго из-за неудачного выбора одного из первых слов. Это наталкивает на мысль не устанавливать слова, оставляющие слишком мало вариантов на установку другим словам.

Данное решение заведомо обрежет некоторые варианты перебора, в которых могло быть решение. Однако, с практической точки зрения, лучше достаточно быстро сказать, что решения нет, чем несколько лет пытаться получить ответ, которого может и не быть.

РЕШЕНИЕ 6. Во время установки слова, будем проверять, что всем неустановленным словам, пересекающим данное, оставлено достаточно вариантов для установки. Если это не выполняется

- слово не подходит для установки. Решение используется на этапе генерации.

Тут следует понимать, что если некоторое слово имеет пересечения на каждой букве, то всего несколько установленных букв могут оставить всего 1-2 варианта для установки. Однако, если каждый раз их отсекать только из-за этого, вполне вероятно, слово вообще не получиться установить. Поэтому в РЕШЕНИИ 6 для каждого пересекающего слова проверяется ещё и как много букв в нём уже установлено. Если это количество букв также меньше некоторой константы - то отсечение вариантов действительно применяется.

Борьба с зависанием

При установке первых нескольких слов, особенно трудно отсечь варианты с низким шансом на успех (всё-таки мы ограничиваем только одну-две буквы, и, как правило, остаётся много слов, отсечение не происходит). Из-за этого в при генерации может происходить 'зависание' - при попытке установить некоторое слово (назовём это слово *словом зависания*) для одного и того же слова его пересекающего (назовём его *проблемным*) не находится вариантов для установки, и мы возвращаемся на предыдущую итерацию, однако затем ситуация повторяется снова и снова.

Если в проблемном слове небольшое количество заполненных букв - значит, скорее всего, одно из предыдущих слов установленно неверно. И пока мы его не заменим, мы так и не сможем установить проблемное слово.

Несложно понять, что при случайном порядке слов в словаре, на месте пересечения слова зависания и проблемного слова успеют побывать все (или почти все) буквы алфавита в среднем за достаточно небольшое количество итераций (около $|\Sigma|$), в то время как количество возможных вариантов для установки слова зависания может достигать нескольких тысяч. Таким образом, уже через некоторое количество итераций становится понятно, что проблема не в слове зависания, а в одном из предыдущих слов, пересекающих проблемное.

РЕШЕНИЕ 7. В случае зависания, возвращаемся к выбору последнего слова, пересекающего проблемное. Если мы возвращались к данному слову из-за зависания уже некоторое число раз - возвращаемся к выбору предыдущего слова, пересекающего проблемное. Решение используется на этапе генерации.

Решения 6 и 7 могут выглядеть спорными, однако действительно значительно сокращают время работы алгоритма в худшем случае на порядок.

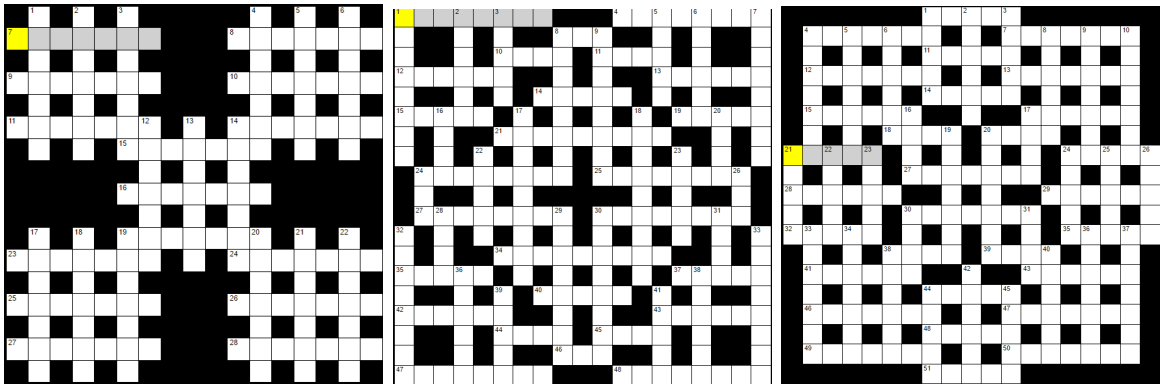
5 Тестирование.

Описанный выше алгоритм был реализован в [3], словарь был повзамиствован из [4] (около 100 тысяч слов). На каждой сетке проводилось 50 тестов. Измерялось время генерации.

В во всех тестах стояло ограничение на количество итераций - 1.000.000. В случае его превышения, перевыбиралось первое слово.

Тест 0:

Сетки:

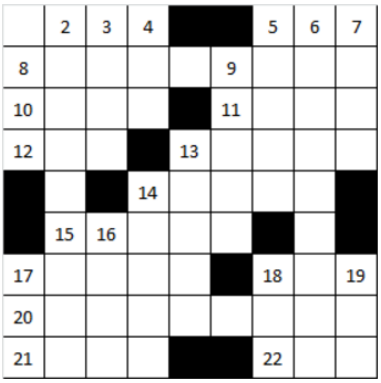


Среднее время работы, секунд: 0 (на всех тестах много меньше одной секунды)

Подитог: алгоритм рассчитывался на сетки с большой сложностью, поэтому такие "простые"сетки генерируются почти мгновенно.

Тест 1:

Сетка:

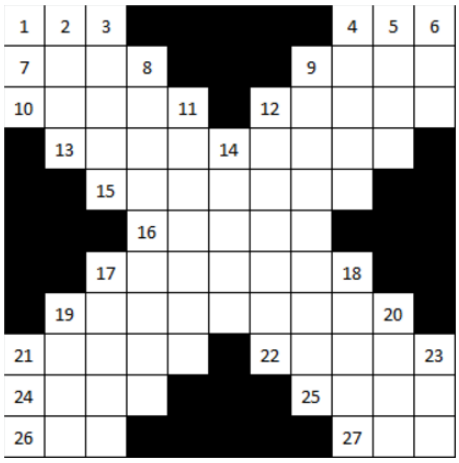


Среднее время работы, секунд: 1,32 (от 0 до 35)

Подитог: Как правило алгоритм отрабатывает крайне быстро (меньше 2 секунд), но иногда работа программы затягивается.

Тест 2:

Сетка:

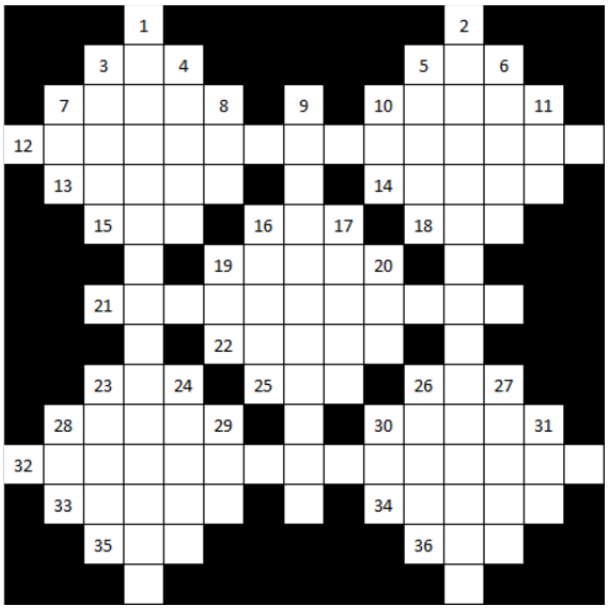


Среднее время работы, секунд: 1,1 (от 0 до 13)

Подитог: На этой, казалось бы, более сложной сетке такой хороший результат был достигнут благодаря последним двум решениям (шестому и седьмому). Время генерации в худшем случае значительно сократилось.

Тест 3:

Сетка:

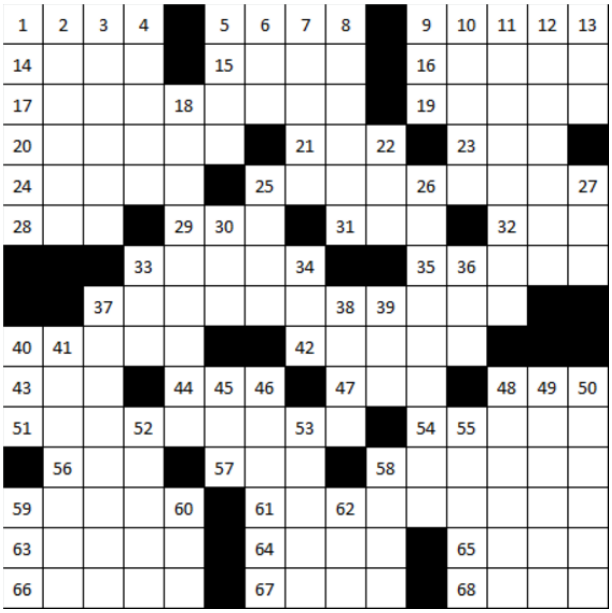


Среднее время работы, секунд: 0,2 (от 0 до 2)

Подитог: данная сетка очень хорошо разбивается на фрагменты, поэтому, несмотря на большее, по сравнению с предыдущими, количество слов, генерация происходит быстрее.

Тест 4:

Сетка:



Среднее время работы, секунд: 29 (от 0 до 392)

Подитог: В данной сетке достаточно много слов (77) и она не так хорошо фрагментируется. Как правило, сетка заполняется быстро (меньше 7 секунд в 90% случаев), однако иногда из-за неудачного выбора первых слов, который не удаётся быстро распознать, генерация заметно затягивается. Однако даже в худшем случае генерация укладывается в 7 минут.

6 Возможные оптимизации.

Безусловно, возможности улучшения данного алгоритма не исчерпаны. Что же можно сделать?

1. Оптимизация констант.

   В данной работе константы (например, минимальное количество слов в шаблоне, при котором не производится обрезание вариантов; количество итеаций, после которого мы возвращаемся к предыдущему слову при зависании и т. п.) подбирались руками. Для нахождения наилучших значений можно воспользоваться каким-нибудь алгоритмом оптимизации. Можно пойти по другому пути и изменять их во время работы (или даже для каждого устанавливаемого слово хранить своё значение констант).

2. Распараллеливание.

   Можно порядок слов сделать не линейным, а лишь частично упорядочить, т. е. после слова, установка которого создаёт фрагменты, будут идти слова фрагментов, но слова разных фрагментов не сравнимы друг с другом. Данный порядок легко представить в виде дерева. Собственно, каждый раз, когда у вершины дерева, соответствующего слову, будет несколько детей - каждое дочернее поддерево будет обрабатываться в отдельном потоке. Если одно из поддеревьев невозможно установить, откат производится до вершины-родителя и все поддеревья пересчитываются заново.

7 Резюме

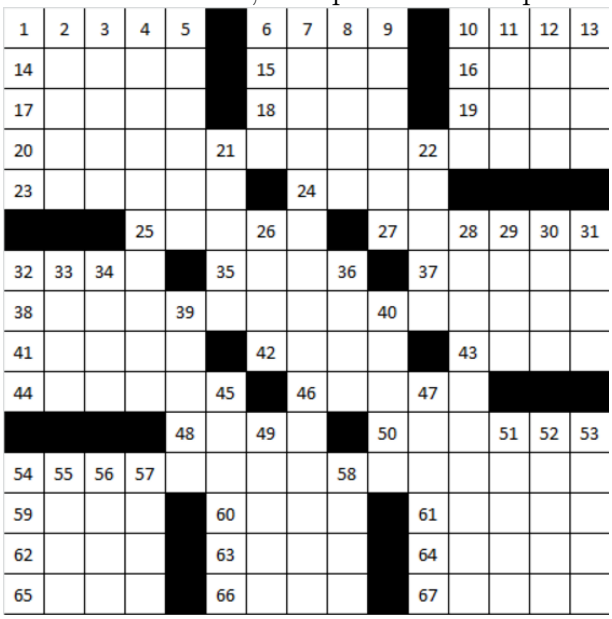
С достаточно простыми сетками (такими, как в тестах 1-3) алгоритм разбирается в пределах 5 секунд в 90% случаев и 2-3 минуты в худшем случае. С сетками средней сложности (такими, как в тесте 4) алгоритм также стабильно справляется в пределах 7 минут.

На тестах использовался достаточно большой словарь, поэтому при многих первых словах удавалось заполнить кроссворд успешно. И на всех тестах такое первое слово удавалось выбрать за 4 попытки в худшем случае. Вполне возможно, что при небольшом словаре из-за последних решений мы отсечём единственный действительно подходящий нам вариант заполнения. В таком случае (при небольших размерах словаря) следует подобрать другие константы или вообще не использовать последние решения, ибо количество вариантов перебора и так значительно сокращается.

Однако с более сложными сетками (большое количество слов, пересечений, плохая фрагментируемость), например, той, что изображена ниже, должно сильно повезти, чтобы генерация была успешна.

Для сетки, изображённой ниже я пробовал произвести генерацию с самыми разными параметрами с ограничением по времени в 10 часов. Однако ни разу результата добиться не удалось.

Сложная сетка, алгоритм её не берёт:



Таким образом, алгоритм применим для сеток

- без (или почти без) очень длинных слов с большим количеством пересечений
- количество слов < 70 или хорошая фрагментируемость

8 Ссылки

- [1] М. Гэри, Д. Джонсон "Вычислительные машины и труднорешаемые задачи"
- [2] `https://habrahabr.ru/post/249899/`
- [3] `https://github.com/andreyzharkov/crossgen` - source code
- [4] `http://crossword.ucoz.ru/download.htm` - словарь