Алгоритм размещения конфликтующих объектов.

Аннотация:

Представлена постановка задачи размещения конфликтующих объектов в одномерном пространстве. Представлен алгоритм решения этой задачи, приведена схема его работы. Приведены результаты аналитических и численных исследований разработанного алгоритма. Приведены рекомендации по применению алгоритмов подобного типа.

Ключевые слова:
оптимальное размещение, жадный алгоритм, метод ветвей и границ, локально-экстремальные методы, реклама, медиа бизнес.

Введение

Современный рынок ТВ рекламы включает в себя множество «игроков» предлагающие схожие по потребительским свойствам товары и услуги, что приводит к вынужденным и всегда взаимно-неприятным соседствам. Это хорошо объяснимо: эффективность рекламы автомобиля падает, когда вслед за рекламным роликом автомобиля следует рекламный ролик еще одного автомобиля. Кроме этого следует учесть растущую раздраженность зрителя, которому скучно смотреть на рекламу одного и того же класса товара. Сложившаяся ситуация, необходимость повышения эффективности рекламного материала привели авторов (а вообще говоря компанию «Видео Интернешнл»???) к необходимости разрешения рекламных конфликтов внутри рекламных блоков.

Постановка задачи

Сформулируем задачу оптимального размещения. Дано множество рекламных роликов $\bar{X} = (x_1, .., x_N)$, (для удобства нумерация начинается с единицы). Ролики из этого множества необходимо разместить в одномерной сети, в таком порядке, который бы минимизировал целевой функционал (ЦФ) P:

$$\boldsymbol{P = \sum_{i=0}^{N-1} \sum_{j=i+1}^{N} \frac{p(x_i, x_j)}{j - i}} \quad (1)$$

где $p(x_i, x_j)$- заданный штраф за соседское размещение роликов x_i и x_j.

Если бы штрафовалось только непосредственное соседство рекламных роликов, то $\boldsymbol{xjj-i}$
(**1**) представляла бы собой задачу коммивояжёра (Travelling salesman problem) [4], которая относиться к классу NP-полных задач. Представленная здесь задача минимизации, скорее

всего так же относиться к классу NP-полных задач, однако доказательства этому факту пока не найдено.

Тем не менее в представленной задаче сложно рассчитывать на алгоритмы полиномиальной сложности и менее. Сложность алгоритма полного перебора составит *O(N!)*. Если для малых N выполнение полного перебора возможно, то с ростом N мы столкнёмся с проблемой нехватки вычислительных ресурсов. Ограничения в вычислительных ресурсах и большие объемы рекламных блоков привели авторов данной статьи к необходимости синтеза экономичного в вычислительном смысле алгоритма поиска оптимального или близкого к оптимальному решения $\bar{X}$.

Синтез алгоритма

Представленная выше задача похожа на задачу коммивояжера, поэтому логично будет применить к ее решению методы, которые рекомендованы для решения задачи коммивояжера. Известно, что в данном классе задач “жадный” алгоритм [3] (greedy alhorithm), оказывается самым быстрым способом получить “хорошее” решение. Жадные алгоритмы – итерационные алгоритмы в которых каждых шаг является локально оптимальным на текущий момент. Рассмотрим работу жадного алгоритма применительно к поставленной выше задаче размещения:

Пусть есть множество роликов $X = (x_1, x_2, x_3, x_4)$. Матрица попарных штрафов имеет вид: $p = \begin{vmatrix} 0 & 1 & 2 & 2 \\ 1 & 0 & 3 & 3 \\ 2 & 3 & 0 & 1 \\ 2 & 3 & 1 & 0 \end{vmatrix}$.

Т.к. в работе жадного алгоритма нам необходимо начинать размещение с самых конфликтных роликов, упорядочим ролики из X, в порядке убывания «конфликтности», которую определим как сумму соответствующей строки в матрице p. Т.о. вектор конфликтности: $\bar{p}$=(5,7,6,6). Далее будем размещать ролики в порядке: (x_2, x_3, x_4, x_1) или (x_2, x_4, x_3, x_1), что неважно т.к. с точки зрения штрафа ролики x_3, x_4 - идентичны.

Сетка размещения изначально пуста:

Ищем оптимальное размещение для x_2, т.к. ни один из вариантов размещения x_2 невозможно предпочесть другому (штраф все равно нулевой) ставим x_2 в первую доступную позицию:

x_2			

Ищем оптимальное размещение для x_3:

x_2			x_3

Ищем оптимальное размещение для x_4:

x_2	x_4		x_3

Ставим x_1 в последнюю доступную позицию:

x_2	x_4	x_1	x_3

Получаем P=9

Основной проблемой «жадного» алгоритма является его… «жадность» - невозможность посмотреть чуть дальше, чем на один ход. Для компенсации этого недостатка авторы предлагают пользоваться эвристическим алгоритмом поиска, который включает в себя достоинства «жадного» алгоритма и метода «ветвей и границ» [3] с ограниченным откатом.

Алгоритм поиска организован следующим образом:

1. Все ролики сортируются в порядке убывания «конфликтности». Пронумеруем их заново $(x_1, x_2, \dots, x_N)$. Смещение S = 0. Сеть размещения – пустая.
2. Берем M роликов $(x_S, \dots, x_{S+M})$ и с помощью полного перебора ищем оптимальное размещение этого подмножества в сетке.
3. Если S+M=N все ролики размещены - Выход.
4. Из всех размещенных M роликов оставляем в сетке только первый x_s, остальные убираем.
5. $S = S + 1$. Возвращаемся к шагу 2.

Отличием данного алгоритма от приведенного выше жадного алгоритма заключается в том что оптимальное размещение x_S ищется не локально оптимально, а с учетом того что после x_S будут размещаться и другие конфликтующие ролики, что позволяет лучше определять оптимальную позицию для x_S.

Проследим по шагам работу данного алгоритма на том же примере:

Сетка:

Отсортированный массив роликов: x_2, x_3, x_4, x_1. Возьмем M = 2.

На первом шаге нам необходимо расставить ролики x_2, x_3, оптимальным образом, получаем:

x_2			x_3

Фиксируем x_2, удаляем x_3:

x_2			

Расставляем следующую пару x_3, x_4:

x_2		x_4	x_3

x_3 можно было бы закрепить и в 3-ей позиции в сетке, но результат от этого не измениться.

Фиксируем x_3, удаляем x_4:

x_2			x_3

На последнем шаге расставляем x_4, x_1, получаем решение:

x_2	x_1	x_4	x_3

Получаем P=7.5

В данном случае нам удалось добиться глобальной оптимальности, хотя понятно, что в общем случае данный алгоритм на такой результат не претендует.

Как показывает практика, предложенный алгоритм удовлетворительно решает задачи размером 7<*N*<30. Действительно, при *N*<7 можно свободно пользоваться полным перебором и не терять в качестве решения, которое при таких малых *N*, часто бывает, заметно «на глаз». При *N*>30 скорость работы алгоритма начинает заметно падать. Для компенсации этого недостатка для больших *N* предлагается разбить блок на подблоки таким образом, что бы длина каждого подблока была менее 30 и проводить оптимизацию в каждом подблоке «почти» отдельно.

Процедуру разделения блока на подблоки предлагается реализовывать следующим образом:

Имеем *N*>30 роликов, они отсортированы в порядке убывания штрафа $(x_1, x_2, \dots, x_N)$. Вычисляем количество подблоков *K*=*N* div 30. Начиная с первого ролика помещаем последовательно все ролики $(x_1, \dots, x_N)$ поочередно по подблокам 1..*K*. Кода подблоки кончатся – начинаем снова с первого подблока, т.о. в каждом подблоке мы получим примерно одинаковый по конфликтности набор роликов.

Замечание: под словом помещаем, понимается не поиск оптимального места для ролика, а всего лишь определение для него подблока.

После такого разбиения предложенный выше алгоритм размещения должен быть применен к каждому подблоку отдельно с одним уточнением: поиск оптимального размещения начиная со 2-ого подблока должен происходить с учетом уже размещенных роликов в предыдущем подблоке, для того что бы избежать размещения конфликтующих роликов на границе подблоков. ~~Для ускорения процедуры размещения в подблоках 2..*K* предлагается учитывать размещение не всего предыдущего подблока, а только последних 5-ти роликов, т.к. именно от них, в основном, зависит штраф ЦФ.~~

С точки зрения авторов данный алгоритм имеет два существенных недостатка:

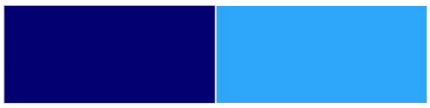
1. Расстановка *n+1*-ого подблока происходит с учетом уже приведенной расстановки *n*-ого подблока, что ведет к не оптимальности решения *n+1*-ого подблока.
2. Разбитие всего множества роликов на подблоки существенно сужает область поиска решения, что может привести к существенному отличию найденного решения от оптимального.

Для избавления от указанных выше недостатков авторы предлагают применить приведенный выше алгоритм для подблоков на границе подблоков. Рассмотрим пример:

Есть блок, который в ходе оптимизации был разбит на 2 подблока:

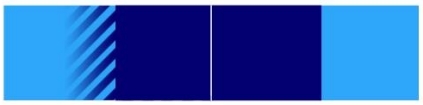
Все множество роликом, соответственно поделено на 2 группы роликов, которые должны быть размещены в подблоках 1 и 2 соответственно.

Сперва выполняется размещение в 1-м подблоке:



Затем размещаем ролики 2-ого подблока с учетом размещения 1-ого подблока:

Затем выделяем промежуточный подблок на границе 1-ого и 2-ого подблока. В данный промежуточный подблок входят $(l1\ div\ 2)$ последних роликов 1-ого подблока и $(l2\ div\ 2)$ первых роликов 2-ого подблока, где $l1$, $l2$ – количество роликов в 1-ом и во 2-ом подблоке соответственно. Выполняем размещение в промежуточном подблоке с учетом расстановки первых $l1 - (l1\ div\ 2)$ роликов 1-ого подблока:



Далее еще раз проводим пере размещение роликов во 2-ом подблоке:

Таким образом нам удалось:

1. Добиться улучшения решения в самом сложном месте (для данного алгоритма) – на границе подблоков.
2. Осуществить частичную «диффузию» (перераспределение) роликов между подблоками.

Очевидно, что данный подход можно обобщить на большее количество подблоков.

Определить на сколько помогает локальная оптимизация.

~~Результат работы предложенного алгоритма не является даже локально-оптимальным, поэтому для конечной «шлифовки» будем использовать алгоритм локальной оптимизации, в котором меняются местами все пары (x_i, x_j), i=1..N-1, j=2..N, если это приводит к уменьшению ЦФ. Пример, в котором был бы действительно нужен локально-экстремальный алгоритм в данной статье не приведен.~~ Общая схема функционирования предложенного алгоритма приведена на рисунке 1.

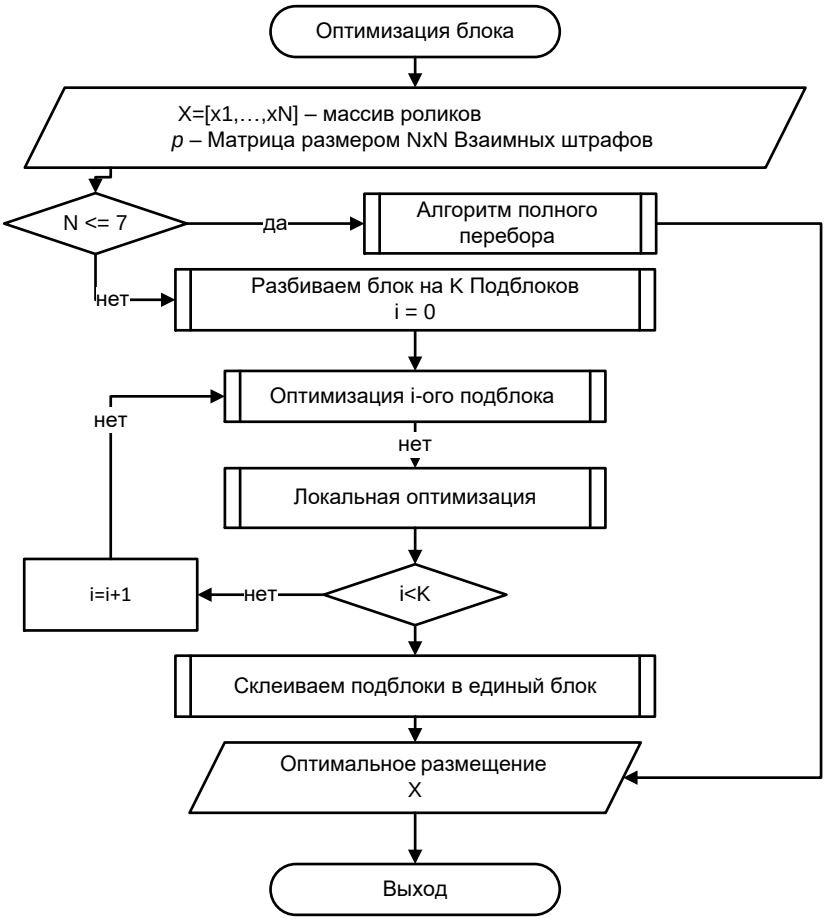


Рисунок 1

Аналитические исследования качеств алгоритма (точность, скорость)

Оценим сложность представленного выше алгоритма. Сложность процедуры 1 (перестановка M объектов по N ячейкам) составляет $C_M^N = \frac{N!}{M!(N-M)!}$ [1] перестановок. Всего алгоритм поиска включает в себя N-M циклов. Таким образом, сложность алгоритма = $O\left(\frac{N!\cdot(N-M)}{M!(N-M)!}\right)$. Воспользовавшись формулой Стирлинга после преобразований получим: $O\big((N-M)\cdot N^M \cdot M^{N-2M}\big)$.

Учитывая, что сложность алгоритма полного перебора равна $O(N!)$, получаем выигрыш в производительности в $M!\,(N-M-1)!$ раз. В то же время, очевидно мы будем иметь потери в качестве решения, которые мы можем оценить численно.

Численные исследования

Оценим численно точность и скорость работы предложенного выше алгоритма. Определить точность работы можно только в сравнении с алгоритмом полного перебора. Ввиду большой ресурсоемкости такого подхода сделать это представляется возможным только для небольших N. Результаты сравнительных исследований для $N \leq 10$ приведены в таблице 1 и на графике 1. В колонке «Ролики» в таблице 1 приведена кодировка рекламного блока на котором проводился замер точности работы алгоритмов. Например, кодировка «1112223» обозначает что в блоке содержатся 7 роликов, в числе которых три одинаковых ролика «111», три одинаковых ролика «222» и ролик «3», который с остальными роликами не конфликтует. При этом здесь и далее: штраф $\begin{cases} p(i,j) = 1, i = j \\ p(i,j) = 0, i \neq j \end{cases}$

Таблица 1. Значения ЦФ для размещения блоков длиной 10 и менее роликов.

	Ролики	Алгоритм полного перебора. Значение ЦФ $P_{полн}$	Предложенный алгоритм. Значение ЦФ P.
7	1112223	2.58	2.58
8	11122334	2.26	2.26
9	111122233	3.82	4.02
10	1111222334	3.35	3.52

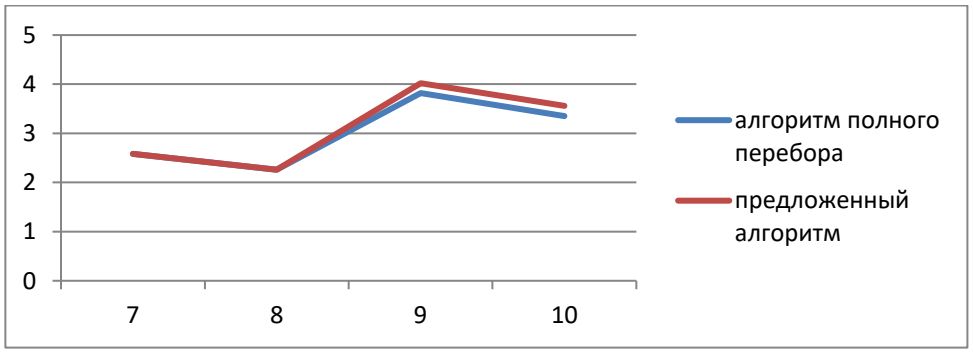


График 1. Значения ЦФ для размещения блоков длиной менее 10 роликов.

К сожалению, авторы не имеют возможности найти оптимальную расстановку с помощью алгоритма полного перебора для N>10, но есть возможность провести специальные тесты, с очевидным решением.

Тест 1. Возьмем блок в 100 роликов, который состоит из двух подмножеств по 50 одинаковых роликов в каждом. Оптимальным размещением в этом случае будет являться чередование роликов из первого подмножества с роликами из второго подмножества.

Оптимальное значение функции штрафа составит P=174.96. Применяя к тому же блоку предложенный алгоритм, получим P=175.86.

Тест 2. Возьмем блок в 100 роликов $x_1, x_2, \dots x_{100}$. Матрица попарных штрафов будет иметь довольно громоздкий вид, поэтому функцию попарных штрафов запишем следующим образом:

$$\begin{cases} p(x_i, x_j) = 1, & \frac{|i-j|}{5} \in Z \\ p(x_i, x_j) = 0, & \frac{|i-j|}{5} \notin Z \end{cases}$$

Другими словами – из первоначального множества X, конфликтуют между собой только элементы из 5-ти подмножеств: $(x_1, x_6, x_{11} \dots x_{96})$, $(x_2, \dots x_{97})$, $(x_3, \dots x_{98})$, $(x_4, \dots x_{99})$, $(x_5, \dots x_{100})$. Оптимальная расстановка для данного теста совпадает с порядком следования роликов $x_1, \dots x_{100}$.

Посчитаем оптимальный штраф для данного размещения. Ввиду того, что конфликтующие подмножества одинаковы с точки зрения штрафа, мы можем подсчитать штраф для одной группы, например $(x_1, \dots x_{96})$ и умножить его на 5. Воспользовавшись ***xjj−i*** (**1**), получаем:

$$\boldsymbol{P = 5 \cdot P(x_1, x_6 \dots x_{96}) = 5 \cdot \begin{pmatrix} \frac{1}{5} + \frac{1}{10} + \frac{1}{15} + \dots + \frac{1}{95} + \\ \frac{1}{5} + \dots + \frac{1}{90} + \\ + \dots + \\ + \frac{1}{5} \end{pmatrix}} \quad (2)$$

Здесь в первой строке записан штраф за конфликты x_1 и всех остальных роликов первого подмножества $(x_1, x_6, x_{11} \dots x_{96})$. Во второй строке штраф за конфликты x_6 и всех *x1*, и т.д. После преобразований (2) получаем:

$$P = 5 \cdot \sum_{i=1}^{19} \frac{(20-i)}{5 \cdot i} = 51{,}95$$

Применим к первоначальному подмножеству предложенный алгоритм, получаем P =53.93.

Тест 3. Возьмем 37 роликов, которые поделены на 2 неконфликтующие группы. 12 роликов первой группы конфликтуют между собой с *p*=10. 25 роликов 2-ой группы с *p*=2. Оптимальное размещение в этом случае будет иметь вид:

x_1	x_2	x_2	x_1	x_2	x_2	x_1	…		x_2	x_2	x_1

где x_1- ролик из 1-ой группы, x_2 - ролик из 2-ой группы.

Значение ЦФ для такого размещения: *P*=206,69.

Предложенный алгоритм так же находит глобальный экстремум: *P*=206,69.

Тест 4. Проведем тест аналогичный тесту 2, только с 3-мя группами по 30 элементов взаимно не конфликтующих объекта. Т.е. имеем ролики $x_1, x_2, \dots x_{90}$. Функция попарных штрафов:

$$\begin{cases} p(x_i, x_j) = 1, & \frac{|i-j|}{3} \in Z \\ p(x_i, x_j) = 0, & \frac{|i-j|}{3} \notin Z \end{cases}$$

Оптимальная расстановка для данного теста совпадает с порядком следования роликов $x_1, \dots x_{90}$.

$$\boldsymbol{P = 3 \cdot P(x_1, x_4 \dots x_{87}) = 3 \cdot \left(\begin{array}{c} \frac{1}{3} + \frac{1}{6} + \frac{1}{9} + \dots + \frac{1}{87} + \\ \frac{1}{3} + \dots + \frac{1}{84} + \\ + \dots + \\ + \frac{1}{3} \end{array} \right)} \quad (3)$$

После преобразований получаем:

$$P = 3 \cdot \sum_{i=1}^{29} \frac{(30-i)}{3 \cdot i} = 89{,}85$$

Применим к первоначальному подмножеству предложенный алгоритм, получаем P =90,38.

При подготовке данной статьи авторы пытались найти худшие примеры, в которых значение ЦФ предложенного алгоритма значительно бы отличался бы от оптимального значения ЦФ, но, это оказалось не так просто и, к счастью авторов, таких примеров найти не удалось.

Сводная таблица примеров

№ примера	P оптимальное	P найденное	% отклонения найденного решения от оптимального
1	174.96	175.86	0,51
2	51,95	53,93 стар	
3	206,69	206,69	0
4	89,85	90,38	0,59
5			
6			

Оценим скорость работы предлагаемого алгоритма в сравнении с алгоритмом полного перебора. Для этого замерим абсолютное время работы в секундах. Результаты замеров приведены в таблице 2. Замерив, абсолютное время работы предложенного алгоритма для больших блоков получим график 2. Заметим, что время обработки блока растет практически линейно относительно объема входных данных.

Таблица 2. Сравнение времени обработки блока алгоритмом полного перебора и предложенным алгоритмом.

	Полный перебор (сек)	Предложенный алгоритм (сек)
7	0.85	0.002
8	0.9	0.002
9	1	0.002
10	2.5	0.002
11	20	0.002
12	126	0.003

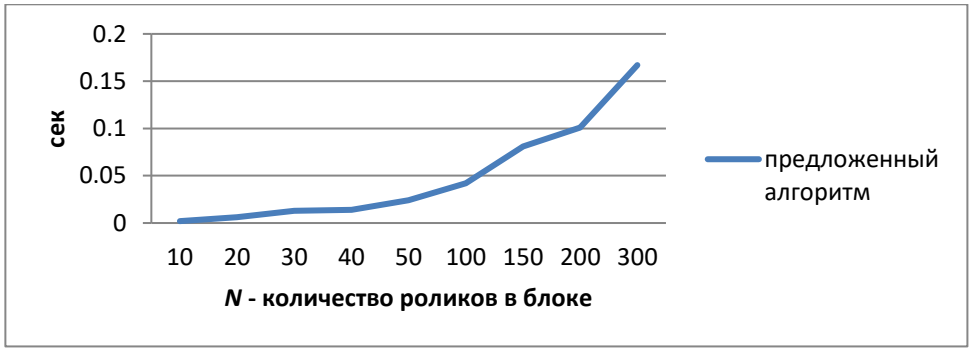


График 2. Зависимость времени работы предложенного алгоритма от количества роликов в блоке

Выводы

Предлагаемый в данной статье алгоритм показал высокое быстродействие при незначительных потерях в качестве решения (менее 5% в худшем случае). Благодаря процедуре разбиения общей задачи на подблоки авторам удалось добиться высокой степени масштабируемости, что позволяет применять предложенный алгоритм на больших объемах данных, при этом сложность алгоритма растет линейно. Данный алгоритм может быть рекомендован к применению для других задач размещения конфликтующих объектов а также для задачи коммивояжера и близких к ней задач обхода графов.

Список литературы

1. Теория вероятности и статистика. В. А. Колемаев,
2. http://edu.nstu.ru/courses/saod/alg_jadn.htm
3. Алгоритмы. Построение и анализ. Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн, Москва, Вильямс 2013.
4. http://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BA%D0%BE%D0%BC%D0%BC%D0%B8%D0%B2%D0%BE%D1%8F%D0%B6%D1%91%D1%80%D0%B0