УДК 519.8.А

**Специализированный алгоритм ветвей и границ для обобщённой задачи коммивояжера с ограничениями предшествования**

М.Ю. Хачай, д. ф.-м.н., mkhachay@imm.uran.ru; С.С. Уколов, s.s.ukolov@urfu.ru; А.А. Петунин, д.т.н., aapetunin@gmail.com

**Аннотация**: Обобщенная задача коммивояжера (Generalized Traveling Salesman Problem, GTSP) – это хорошо известная задача комбинаторной оптимизации, имеющая множество важных практических приложений при исследовании операций. В GTSP с ограничением предшествования (PCGTSP) накладываются дополнительные ограничения на порядок посещения кластеров в соответствии с некоторым заранее заданным частичным порядком. В отличие от классической GTSP, PCGTSP все еще слабо исследована с точки зрения разработки и реализации алгоритмов. Насколько нам известно, все известные алгоритмические подходы для этой проблемы исчерпываются общей структурой ветвления Салмана, несколькими моделями MILP и недавно предложенной авторами метаэвристикой PCGLNS. В данной работе представлен первый специализированный алгоритм ветвей и границ, разработанный с расширением подхода Салмана и использующий PCGLNS в качестве мощной первичной эвристики. Используя общедоступную тестовую библиотеку PCGTSPLIB, мы оцениваем производительность предложенного алгоритма по сравнению с классической схемой динамического программирования Хелда-Карпа, дополненный стратегией ветвления и границ, и современным решателем Gurobi, использующим нашу недавнюю моделью MILP и горячий старт на основе PCGLNS.

**Ключевые слова**:

**Problem-Specific Branch-and-Bound Algorithms for the Precedence Constrained Generalized Traveling Salesman Problem**

M.Yu. Khachay, [mkhachay@imm.uran.ru](mailto:mkhachay@imm.uran.ru); S.S. Ukolov, [s.s.ukolov@urfu.ru](mailto:s.s.ukolov@urfu.ru); A.A. Petunin, [aapetunin@gmail.com](mailto:aapetunin@gmail.com)

**Abstract**: The Generalized Traveling Salesman Problem (GTSP) is a well-known combinatorial optimization problem having numerous valuable practical applications in operations research. In the Precedence Constrained GTSP (PCGTSP), any feasible tour is restricted to visit all the clusters according to some given partial order. Unlike the common setting of the GTSP, the PCGTSP appears still weakly studied in terms of algorithmic design and implementation. To the best of our knowledge, all the known algorithmic results for this problem can be exhausted by Salmans's general branching framework, a few MILP models, and the PCGLNS meta-heuristic proposed by the authors recently. In this paper, we present the first problem-specific branch-and-bound algorithm designed with an extension of Salman's approach and exploiting PCGLNS as a powerful primal heuristic. Using the public PCGTSPLIB testbench, we evaluate the performance of the proposed algorithm against the classic Held-Karp dynamic programming scheme with branch-and-bound node fathoming strategy and Gurobi state-of-the-art solver armed by our recently proposed MILP model and PCGLNS-based warm start.

**Keywords**:

1. Введение

Обобщенная задача коммивояжера (GTSP) – это хорошо известная задача комбинаторной оптимизации, представленная в основополагающей статье [1] Сриваставы и др. и привлекшая внимание многих исследователей (см. обзор в [2]). В GTSP для данного взвешенного орграфа и разбиения набора узлов на непустые взаимно непересекающиеся кластеры требуется найти замкнутый тур с минимальной стоимостью, который посещает каждый кластер ровно один раз. В этой статье мы рассматриваем обобщенную задачу коммивояжера с ограничением предшествования (PCGTSP), в которой кластеры следует посещать в соответствии с некоторым заданным частичным порядком. Эта расширенная версия GTSP имеет множество практических применений, включая

* оптимизацию траектории инструмента для станков с числовым программным управлением (ЧПУ) [3]
* минимизацию времени *холостого хода* при раскрое листового металла [4; 5]
* координатно-измерительное оборудование [6]
* оптимизацию траектории при многоствольном бурении [7].
  1. Связанные работы

GTSP – это расширение классической задачи коммивояжера (TSP). Следовательно, если оценивать размер задачи количеством кластеров, задача становится NP-сложной даже на евклидовой плоскости [8]. Сдругой стороны, хорошо известная схема динамического программирования Хелда и Карпа [9], адаптированная к GTSP, имеет временную сложность, то есть этот алгоритм принадлежит FPT, будучи параметризован количеством кластеров. Следовательно, оптимальное решение GTSP может быть найдено за полиномиальное время при условии. Обзор литературы показывает, что алгоритмическое проектирование GTSP развивалось по нескольким направлениям.

Первый подход основан на сведении исходной задачи к некоторой задаче асимметричной TSP, после чего этот вспомогательный экземпляр может быть решен с помощью алгоритмов, разработанного для ATSP ([10; 11]). Несмотря на математическую элегантность, этот подход страдает несколькими недостатками:

1. полученные экземпляры ATSP устроены довольно необычно, что затрудняет их решение даже для современных решателей MIP, таких как Gurobi и CPLEX.
2. близкие к оптимальным решения задачи ATSP могут соответствовать недопустимым решениям исходной задачи [12].

Другой подход связан с разработкой точных алгоритмов для частных случаев и алгоритмов аппроксимации с теоретическими гарантиями производительности. Среди них есть алгоритмы ветвей и границ и ветвей и разрезов (см., например, [13; 14]) и приближенные схемы полиномиального времени (PTAS) для некоторых специальных случаев [15; 16].

Наконец, третий подход заключается в разработке различных эвристик и метаэвристик. Так, Г. Гутин и Д. Карапетян [17] предложили эффективный меметический алгоритм, в [18] знаменитый эвристический решатель Лина-Кернигана-Хельсгауна был расширен до GTSP, а в [19] была разработана мощная метаэвристика Adaptive Large Neighborhood Search (ALNS), которая на сегодняшний день является наиболее эффективной. К сожалению, в случае PCGTSP алгоритмические результаты все еще остаются довольно малочисленными. Насколько нам известно, в открытых источниках доступны только

1. эффективные алгоритмы для специальных ограничений предшествования типа Баласа [20—22] и ограничений предшествования, приводящих к квази- и псевдопирамидальным оптимальным обходам [23]
2. общие идеи о специализированном (PCGTSP) алгоритме ветвей и границ [24]
3. недавняя эвристика PCGLNS, предложенная авторами [25] как расширение результатов [19].

В этой статье мы пытаемся восполнить этот пробел.

* 1. Новизна данной работы
* расширяя идею, предложенную в [24], мы разрабатываем и реализуем первый специализированный алгоритм для PCGTSP
* расширяя классический подход к ветвлению [26], мы реализуем схему динамического программирования Хелда и Карпа, дополненную оригинальной ограничивающей стратегией
* проведенные численные эксперименты показывают, что производительность предложенных алгоритмов сравнима как в смысле скорости, так и точности получаемых решений с современным решателем Gurobi, использующим лучшую в настоящее время MILP-модель и стартовое решение MIP

2. Постановка задачи

Мы рассматриваем общую постановку обобщённой задачи коммивояжера с ограничениями предшествования (PCGTSP). Задача определяется тройкой , где

* взвешенный ориентированный граф определяет веса для всех путей
* разбиение делит множество вершин графа на непустых попарно непересекающихся *кластеров*
* ориентированный ациклический граф определяет частичный порядок (*ограничение предшествования*) на множестве кластеров .

Для каждой вершины, за обозначим (единственный) кластер , такой что . Далее, без ограничения общности, полагаем транзитивно замкнутым (то есть из и следует ) и что для каждого .

Замкнутый -тур называется *допустимым решением* задачи PCGTSP, если он

* начинается и заканчивается в некоторой вершине
* посещает каждый кластер
* каждое ребро в (кроме ребра ) удовлетворяет ограничению предшествования, то есть .

Каждому решению , мы назначаем стоимость

Требуется найти допустимый тур с минимальной стоимостью .

3. Общие соображения

Оба алгоритма, разработанные и реализованные в данной работе, используют общие основные идеи.

3.1. Разбиение задачи

В каждой вершине дерева поиска мы разделяем исходную задачу на две (более простых) подзадачи следующим образом:

1. Рассмотрим подмножество , такое что , зафиксируем некоторый кластер и вершины и соответственно
2. Пусть – нижняя граница стоимости -путей, проходящих через все кластеры и удовлетворяющих ограничению предшествования[[1]](#footnote-1)
3. Исключая из все внутренние кластеры и соединяя с напрямую ребром нулевого (0) веса, мы тем самым создаём подзадачу , имеющую все те же остальные веса путей, разбиение на кластеры и ограничения предшествования, что и исходная
4. Принимая

(1)

за нижнюю границу мы отсекаем все узлы, для которых . Здесь это вес некоторого эффективно находимого решения упрощённой задачи и – стоимость наилучшего известного допустимого решения исходной задачи.

3.2. Нижние границы

В этом разделе мы сравним разные способы получения нижних границ для вспомогательной задачи . Рассмотрим несколько способов упростить , используя двухэтапный подход, предложенный в [24].

На первом этапе мы упрощаем , превращая ее в задачу ATSP одним из следующих способов:

1. Ослабляя исходное ограничение предшествования, исключаем все ребра , для которых . Затем, сводим полученную задачу к ATSP, используя классическую трансформацию Нуна-Бина [11].
2. Тем же способом ослабив исходное ограничение предшествования, сводим полученную задачу к ATSP, определённой на вспомогательном *графе кластеров* , где
3. Сводим исходную задачу к ATSP, определённой на ориентированном графе , где

то есть, для любого , упорядоченная пара , если существует и вершины и , такие, что путь не запрещён исходным ограничением предшествования . Далее,

На втором этапе мы находим приближенное решение полученной задачи ATSP, путем нахождения либо минимального остовного дерева (Minimum Spanning Arborescence Problem, MSAP), либо решения задачи о назначениях (Assignment Problem, AP) и тем самым получаем значение нижней границы по формуле (1). Кроме того, мы можем посчитать ещё более точную нижнюю границу, прямо решая вспомогательную задачу ATSP при помощи солвера Gurobi (на практике только для задач, полученных способом 2). Для удобства все способы получения нижних границ сведены в табл. 1a). Её столбцы – разные способы сведения задачи к ATSP на первом этапе, строки – способы решения полученной задачи ATSP на втором.

Таблица 1

Нижние границы

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | |  | Нун-Бин |  |  | | AP | E1 | **L1** | **L2** | | MSAP | E2 | E3 | E4 | | Gurobi | E5 | **L3** | E6 | | |  |  |  | | --- | --- | --- | | E1 | E2=E3 | E4 | | 0.48±0.03 | 0.54±0.01 | 0.60±0.002 | | L1 | L2 | L3 | | 0.91±0.02 | 0.97±0.02 | 1.00 | |
| а) Обозначения | б) Оценки по сравнению с L3 |

На основе результатов численных экспериментов мы сократили полный список методов оценки нижней границы, см. табл. 1б). На практике оценки L1–L3 оказываются почти всегда наиболее строгими, статистически значимо с доверительным интервалом 95%. Мы также отказались от использования оценок E5 и E6 ввиду того, что они требуют гораздо большего времени счета. Таким образом, в разделе 6 мы ограничились использованием оценок

4. Алгоритм ветвей и границ

Для решения задачи PCGTSP , мы обходим дерево поиска в ширину (Breadth First Search), см. Алгоритм 1. Каждый узел этого дерева связан с префиксом , где , , и . Кластеры посещаются в порядке , а все остальные – в произвольном порядке (с соблюдением ограничений предшествования ), тем самым образуя вспомогательную задачу из раздела 3.1.

|  |  |
| --- | --- |
| **Алгоритм 1**. BnB :: Главная процедура | |
| Вход: | орграф , кластеры , частичный порядок |
| Выход: | маршрут и его стоимость |
|  | Инициализация empty queue |
|  | Начинаем с |
|  |  |
|  | **while** **not** .empty() |
|  | Берем следующий префикс для обработки |
|  |  |
|  | **if** **not** **then** |
|  | Префикс отсекается; **continue** |
|  | **end** **if** |
|  |  |
|  | for all **do** |
|  | Помещаем префикс в очередь на обработку .push() |
|  | **end** **for** |
|  | **end while** |

К каждому узлу дерева поиска мы применяем процедуру отсечения (Алгоритм 2), которая выполняет следующие действия:

* для префикса мы находим кортеж
* на шаге 4, мы вычисляем матрицу минимальных попарных весов по формуле:

Эта матрица удобно вычисляется инкрементально на основе матрицы родительского узла дерева поиска

* если, для некоторого , и

то префикс имеет веса в матрице больше, чем для префикса и подлежит отсечению

* на шаге 11, мы рассчитываем оценки L1 и L2, см. табл. 1 и сохраняем их в глобальной переменной , используя формулу
* для текущего узла , рассчитываем на шаге 13 нижнюю границу по формуле
* наконец, узел отсекается, если

|  |  |
| --- | --- |
| **Алгоритм 2**. BnB :: Bound | |
| Вход: | префикс |
| Выход: | признак того, что префикс подлежит обработке1 |
|  | **global** |
|  | **global** |
|  | вычисляем кортеж |
|  |  |
|  | **if** **then** |
|  | **return** **false** |
|  | **end** **if** |
|  | обновляем веса маршрутов |
|  |  |
|  | **if** **then** |
|  | вычисляем нижнюю границу |
|  | **end** **if** |
|  |  |
|  | **if then** |
|  | **return false** |
|  | **end if** |
|  | **return true** |

Префиксы, которые избежали отсечения, обрабатываются процедурой (Алгоритм 3), которая пытается удлинить префикс на один кластер, соблюдая при этом ограничение предшествования .

|  |  |
| --- | --- |
| **Алгоритм 3**. BnB :: Branch | |
| Вход: | префикс |
| Выход: | список потомков префикса для обработки |
|  | Инициализация empty queue |
|  | **for** **all** **do** |
|  | **true** |
|  | **for all do** |
|  | **if** **or** **then** |
|  | **false** |
|  | **break** |
|  | **end** **if** |
|  | **end for** |
|  | **if** **then** |
|  | добавляем новый префикс .push() |
|  | **end if** |
|  | **end for** |
|  | **return** |

1. В нашем варианте динамического программирования эта граница будет точной [↑](#footnote-ref-1)