Федеральное государственное бюджетное учреждение науки
Институт математики и механики им. Н. Н. Красовского
Уральского отделения Российской академии наук

На правах рукописи

Салий Ярослав Витальевич

Некоторые методы решения маршрутных задач с условиями предшествования

Специальность 05.13.18 —
Математическое моделирование, численные методы и комплексы программ

Диссертация на соискание учёной степени
кандидата физико-математических наук

Научный руководитель:
д.ф.-м.н., член-корр. РАН
Ченцов А. Г.

Екатеринбург — 2018

Содержание

Введение

Общая характеристика работы

Степень разработанности. Актуальность

Рассматриваемые в диссертации задачи допустимо полагать обобщениями широко известной *задачи коммивояжера* (Traveling Salesman Problem), далее TSP: требуется посетить города из множества $1\,..\,n$, каждый по одному разу, так, чтобы суммарная стоимость перемещений, заданных некоторой функцией стоимости $\mathfrak{c}\colon 1\,..\,n\times 1\,..\,n \to \mathbb{R}$ (обычно представлена в виде матрицы) оказалась минимальной. Решение этой задачи — перестановка индексов $1\,..\,n$, называемая *маршрутом*, откуда «маршрутные задачи»[1]. Сравнительно недавний обзор TSP и некоторых ее обобщений представлен в [2]; см. также другие обзоры, посвященные TSP либо ее конкретным обобщениям [3–13]. Со времен первых работ [14–17], рекордная размерность решаемых задач существенно выросла — с 49 городов в 1954 году до 85 900 в 2006 (*симметричные расстояния*), см. подробнее в [11, § 1.7].

Замечание. В зависимости от того, фиксируются ли начальная и конечная точки, требуется или нет возвращение в начальную точку по завершении обхода, задачи могут называться несколько по-разному. Мы используем форму задачи «о кратчайшей гамильтоновой цепи (ii)» (Shortest Hamiltonian Chain Problem (ii), SHCP (ii) [18]): начало фиксировано в некоторой особой точке 0 («базе»), и, по завершении обхода $1\,..\,n$ (гамильтонов путь по $1\,..\,n$), агент должен попасть в фиксированную конечную точку $\mathfrak{t} = n+1$ («терминал»), *сохраняя* обозначение TSP и далее не оговариваем особо подобные отличия в постановках задачи.

Замечание (О труднорешаемости TSP). Труднорешаемость — NP-трудность[2] TSP — широко известный, классический результат; см., напр., [19]. Принадлежность рассматриваемых нами задач (как правило, содержащих «TSP» в обозначении) к этому же классу сложности обуславливается как минимум одним из аргументов ниже:

(a) TSP является *частным случаем*: TSP-PC (допустимы *пустые*, тривиальные условия предшествования), Generalized TSP (GTSP, см. [2, Ch. 14]), Clustered TSP [20]: достаточно рассмотреть мегаполисы (кластеры), содержащие один и только один город

(b) известно полиномиальной сложности сведение к TSP: Bottleneck TSP (BTSP), см. [2, Ch. 15]

Далее мы не оговариваем особо труднорешаемость обобщений TSP.

[1] здесь «маршрутные задачи» трактуются *узко*, исключая вопрос распределения заданий по нескольким исполнителям, характерный для «задачи маршрутизации транспортных средств» Vehicle Routing Problem (VRP) и ее обобщений (см. [1]).

[2] здесь следует сделать оговорку, что NP-*полна* не сама TSP, а ее формулировка в виде *задачи распознавания* (decision problem), где задается вопрос не об оптимальном гамильтоновом пути/цикле, а о том, есть или нет цикл заданного веса

В диссертации преимущественно рассматриваются задачи, в которых движение агента дополнительно ограничено *условиями предшествования*[3]: например, требуется некоторый город a непременно посетить ранее города b и т. д.; в TSP-подобной интерпретации можно говорить о том, что агент должен *сначала* забрать посылку в городе a и лишь затем *передать* ее в город b (см. интерпретации, связанные с теорией расписаний в библиографии обзора [21]). Обычно условия предшествования предполагаются *транзитивными*: если город a нужно посетить прежде города b, а город b — прежде города c, то никак не возможно посетить город c прежде города a; автору неизвестны постановки TSP-подобных задач, в которых транзитивность отрицается.

Распространены три варианта *формализации* условий предшествования: *отношение порядка* [22,23], *ациклический орграф* [24] и «фундированное» *бинарное отношение* [25,26]. В случае конечного числа городов, все три представления эквивалентны (например, в смысле совпадения допустимых маршрутов), что следует, в частности, из того, что отношение покрываемости определяет частичный порядок [27, гл. I, § 2, лемма 1]; см. также теоретико-графовую интерпретацию [28]. Для перехода от представления в виде множества пар отправитель–получатель («адресные пары») к представлению в виде частичного порядка требуется взять *транзитивное замыкание* (см., напр., [29, Def. 3.2.1]), которое достаточно быстро отыскивается классическим алгоритмом Руа–Уоршолла [29, § 3.2.8].

В диссертации условия предшествования формализуются в виде некоторого частичного порядка $P = (1\,..\,n, <_P)$; предполагается, что если для двух городов $a, b \in 1\,..\,n$ известно, что $a <_P b$, то в любом *допустимом* маршруте город a должен стоять *прежде* b.

В русскоязычной литературе задача коммивояжера с условиями предшествования обыкновенно называется *задачей курьера*; термин впервые введен в [30], упомянут в обзоре [5]. В англоязычной литературе наиболее распространенным представляется Sequential Ordering Problem (SOP), введенный в [24]; также встречается Precedence Constrained TSP (TSP-PC); аббревиатура взята из [31]; и другие сходные аббревиатуры: PCATS [32] (где «A» обозначает *асимметричность* функции стоимости перемещений), PCTSP[4] [33, 34]. Мы будем использовать аббревиатуру TSP-PC.

Уже «чистая» TSP-PC имеет немало приложений:

логистика оптимизация маршрута курьера, собирающего и доставляющего заказы, вертолета, осуществляющего инспекцию нефтяных вышек [35] и др.

оптимизация производственных процессов выбор оптимального порядка операций на конвейере и др. [24, 36, 37]; условия предшествования связаны с технологическими особенностями операций — например, отверстие сначала требуется просверлить и лишь затем можно будет развертывать; другой пример — конвейер покраски автомобилей, где при переход от более *темного* оттенка краски к более *светлому* требует существенно более затратных процедур переналадки (вплоть до запретительно дорогих), чем в обратном случае [38]; в задачах маршрутизации инструмента в машинах листовой резки условия предшествования отражают необходимость, при наличии вложенных контуров, сначала вырезать внутренние [39] (однако в таких задачах предпочтительнее рассматривать *обобщенные* задачи с конгломератами городов, см. далее)

оптимизация работы крана-штабелера (stacker crane), автоматизированной системы хранения [40]; условия предшествования могут быть связаны, например, с хрупко-

[3] в англоязычной литературе встречаются в форме «precedence constraints», реже «precedence relations» и еще реже «precedence conditions»

[4] также обозначает другую задачу — Prize Collecting TSP [2, Ch. 14], где за посещение каждого города начисляются «очки», которых нужно набрать не менее заранее заданного порога, минимизировав стоимость посещения — не требуется посещать *все* города.

стью единиц хранения: нельзя допускать, чтобы, при наборе заказа из системы хранения, что-то ложилось *сверху* на *хрупкие* грузы

Наиболее исследованной, разработанной и распространенной следует признать методику решения TSP-PC с помощью линейной релаксации постановки в виде задачи целочисленного программирования (MILP, Mixed Integer-Linear Programming), как правило, включающей схему ветвей и границ, см., напр., [32, 34, 41–43]. За небольшим исключением (решение задачи ry48p.3.sop из TSPLIB [44]), наивысшие достижения в решении задач TSP-PC из библиотеки TSPLIB получены с помощью MILP в [34].

Кроме MILP следует отметить решения методом ветвей и границ, не задействующие этот аппарат [45, 46] и решения на основе «многозначных диаграмм решений» [47] (multivalued decision diagrams, см. подробнее [48]).

Динамическое программирование (ДП), работа с которым представляет основное содержание диссертации, трудно назвать характерным методом решения TSP-PC (в формулировке без временных окон): насколько известно автору, за исключением [49], где рассматривался особый случай TSP-PC на прямой, а также работ А. Г. Ченцова и соавторов (одна из первых статей, посвященных задаче с условиями предшествования, вышла в 2004 году [50]), наиболее новой можно считать публикацию 1992 года [31]; следует отметить, что в последней статье применена схема ветвей и границ в динамическом программировании по [51]. Впервые ДП для TSP-PC было сформулировано, соответственно, в [52] — в прямой постановке, подобной [16], — и в [50] — в *попятной* постановке, подобной [15].

Модель с *абстрактной* функцией агрегирования затрат позволяет единообразно с задачами с *аддитивным* агрегированием стоимости перемещений рассматривать задачи *на узкие места* (то же *минимакс*) BTSP-PC[5], в которых, вместо последовательного суммирования затрат на перемещение, к ним применяется операция взятия наибольшего max; таким образом, стоимостью маршрута полагаются затраты на *наиболее дорогостоящее* перемещение. Для этой модели корректность динамического программирования для задачи с условиями предшествования доказана, например, в [53]; впервые, для задачи без условий предшествования, подобная модель рассматривалась, по-видимому, в [54].

Прообразом модели с абстрактной функцией агрегирования затрат допустимо полагать описание динамического программирования через общее понятие *разложимости* целевой функции, происходящее, в частности, из [55, 56] и обсуждаемое, например, в [57, Ch. 8, §§ 2.5,2.6].

Несмотря на малое[6] внимание к B(G)TSP-PC[7] в литературе, следует отметить, что для многих приложений TSP-PC такая постановка тоже актуальна:

- Там, где TSP-PC предполагает минимизацию *суммарных затрат* агента, в рамках BTSP-PC можно изучить *выполнимость* задания исходя из ограниченных ресурсов агента, пополняемых при каждом посещении города; например, можно ставить задачу о минимальное емкости *аккумулятора* электромобиля, электрокара или беспилотного летательного аппарата (дрона), выполняющего доставку заказов либо некоторые работы по обслуживанию. Стоимость и вес аккумуляторов являются одним из факторов, затрудняющих широкое применение таких устройств. В свою очередь, в задачах теории расписаний, max-агрегирование обеспечивает оптимизацию *циклической скорости*, определяющей, например, максимальную производительность конвейера.

[5] обсуждение ее *обобщенного* варианта BGTSP-PC см. ниже и в гл. 2

[6] автору не удалось отыскать работ, в которых одновременно было бы и max-агрегирование стоимости перемещений и условия предшествования, исключая серию работ А. Г. Ченцова и соавторов (см., напр., [58–62])

[7] т. е. задача с max-агрегированием стоимости перемещений и условими предшествования или ее обобщенный (с кластерами/мегаполисами) вариант

- Кроме того, эти постановки могут рассматриваться как элемент *робастных* методов дискретной оптимизации [63], призванных, в первую очередь, давать «надежные» решения в случае *неопределенности*[8]. В частности, можно понимать их как эвристическое приближение к «истинно робастным» задачам, см. подробнее о минимаксных вариантах классических задач комбинаторной оптимизации в [65]. В продолжение разговора о *робастности* следует упомянуть двойственную постановку: задачу на максимум с *min*-агрегированием, в TSP-подобной постановке известную как Maximum Scatter TSP [66]. Одно из приложений этой постановки связано с избежанием *термической* деформации, например, при установке заклепок; среди других приложений, связанных с избежанием термической деформации, отметим задачи о маршрутизации инструмента в машинах листовой резки (о технических ограничениях в таких задачах см. [39]).

Важным обобщением TSP является задача с зависимостью стоимости перемещений от времени (Time-Dependent TSP, далее TD-TSP[9]); см. обзор [13]. Практическая сложность решения TD-TSP зависит от конкретной разновидности зависимости от времени; по настоящее время остаются нерешенными экземпляры задачи, содержащие порядка 100 городов [13]. Особо упомянем постановки TD-TSP, где зависимость от времени связана с тем, что города, либо множества городов, если рассматривается обобщенная задача, перемещаются в пространстве согласно заданной системой дифференциальных уравнений динамике, что требует использования подходов теории управления для определения стоимости перемещений: [70, 71]; см. также [72, 73].

За редкими исключениями, зависимость от времени не исследуется в комбинации с другими обобщениями TSP. Из этих редких и важных исключений упомянем [74], где применен комбинированный подход, задействующий *программирование в ограничениях* (constraint programming; см. обзор [75]), «диаграммы решений» (decision diagrams, см. подробнее [48]) и линейное программирование для решения TD-TSP, TD-TSP с временными окнами (TD-TSP with Time Windows: для каждого города посещение допустимо только в определенном временном интервале), и TD-TSP с условиями предшествованиями (в [74] названа «TD-SOP», далее обозначается TD-TSP-PC); в частности, предложен набор тестовых экземпляров TD-TSP-PC на основе известной библиотеки экземпляров TSPLIB [76] с функцией стоимости типа «traveling deliveryman» (стоимость перемещения — расстояние, помноженное на номер перемещения в маршруте *с конца*: последнее перемещение домножается на 1, предпоследнее — на 2, и так далее). Отметим, что концептуально сходная с *traveling deliveryman* функция стоимости перемещений рассматривалась в обобщенной задаче с условиями предшествования в [73].

В [77] автору диссертации удалось превзойти некоторые результаты [74] в части решения TD-TSP-PC: точным ДП закрыты 7 экземпляров, включая экземпляры свыше 100 городов, размерность которых представлялась в [74] как слишком высокая; показано, что эвристика усеченного ДП (restricted dynamic programming, впервые предложена в [78]) достигает все известные оптимальные решения (включая полученные точным ДП) кроме одного экземпляра (p43.1.sop) на подходящем значении параметра и улучшены либо достигнуты *все* оценки сверху для открытых экземпляров (по сравнению с [74]), см. подробнее в гл. 1.

Дальнейшим обобщением TD-TSP можно считать задачи, где стоимости перемещений «зависят от предыстории»: стоимость перемещения из города a в город b зависит от множества городов *уже посещенных* на момент перемещения из a в b. По аналогии с зада-

[8] отметим также игровую постановку GTSP с неопределенностью в выборе городов [64]

[9] также *динамическая* задача коммивояжера [5]; в англоязычной литературе «Dynamic TSP» может относится к не связанной разновидности задач, где *динамичность* заключается в постепенном раскрытии общего списка заданий, что имитирует динамическое поступление заявок, см. [67–69]

чами теории расписаний с похожей зависимостью [79][10], будем называть соответствующий вариант задачи коммивояжера Past Sequence Dependent TSP-PC (далее PSD-TSP-PC)[11]

В TSP-подобной постановке зависимость от списка заданий, по-видимому, впервые описана в 2010 году. В [80] решается одна специфическая задача монтажа печатных плат[12], в которой зависимость от списка заданий возникает в связи с тем, что подвижная плита, на которой закреплена монтируемая печатная плата, должна «подставить» (смещаясь в плоскости) место для очередного компонента под аппарат, непосредственно осуществляющий монтаж; оптимизируя размещение компонентов во вращающемся магазине монтажного аппарата, можно уменьшить расстояние, проходимое подвижной плитой, что приводит к ускорению всего процесса. Второй аспект зависимости от списка заданий связан с тем, что, в зависимости от веса компонентов в магазине монтажного аппарата, изменяется скорость его вращения. Дается постановка в форме задачи целочисленного программирования, которая решается эвристически (точные методы не рассматриваются). Отметим также [66], где рассматривается Maximum Scatter TSP, фактически, с зависимостью от предыстории: предполагается поиск наибольшего удаления от m городов, посещенных агентом прежде текущего.

В [53] рассматривалась более общая постановка задачи, связанная с минимизацией облучения сотрудников атомной электростанции, демонтирующих выведенный из эксплуатации энергоблок; в качестве стоимости перемещения рассматривалалась дозовая нагрузка, получаемая в процессе перемещения и работ на «городах», и зависимость от списка (невыполненных) заданий возникала в связи с предположением о том, что источниками излучения являются только *еще не демонтированные* объекты. Постановка совмещала несколько обобщений TSP обычно рассматриваемых по-отдельности: (а) зависимость стоимости от списка (невыполненных) заданий, (б) обобщенность (перемещение по комплексам городов — мегаполисам, аналог GTSP [2, Ch. 13]), (в) условия предшествования на перемещения между мегаполисами; также, функция агрегирования стоимости перемещений рассматривалась в *абстрактной* форме, что позволило доказать корректность *динамического программирования* (точный метод решения) для произвольной операции агрегирования, монотонно неубывающей по второму аргументу, таким образом, могут быть использованы как обычная сумма, характерная для классической TSP, так и, например, max, характерный для задачи на узкие места (Bottleneck TSP, см. [2, Ch. 15]). Авторы приводят оптимальное решение одной задачи с 27 мегаполисами (по 14–20 городов) с 20 парами в фундированном отношении, задающем условия предшествования; время счета составило 29 минут 29 секунд.

Также отметим приложение задач с зависимостью от списка заданий к маршрутизации инструмента в машинах листовой резки (см., напр., [85]): в них стоимость перемещений совмещает характеристики времени холостого хода (менее важная часть) и риска брака в процессе резки (более важная) за счет учета, в частности, «условий жесткости листа» — эмпирических соображений, призванных предотвратить, в частности, тепловую деформацию деталей в процессе резки; подробнее о дополнительных условиях в задаче маршрутизации листовой резки см. [39].

[10] роль «функции стоимости перемещений» в них играет *время переналадки*

[11] в [44, 62, 77] мы использовали ту же аббревиатуру без «past» (SD-TSP-PC), вслед за [80]

[12] Задача названа «Sequence-Dependent TSP», что не совсем точно согласуется с аналогичным термином из теории расписаний (см. обзоры [21, 81, 82]), обозначающим лишь зависимость времени переналадки (аналог функции стоимости перемещений в TSP) от *предыдущего* задания, более длинная предыстория не учитывается; в TSP такая зависимость есть *по определению*: стоимость перемещения зависит от пары городов. Для обозначения зависимости от предыстории в *теории расписаний* использовались термины «state dependent» [83], «past-sequence-dependent» [79] (представляется наиболее распространенным), «history-dependent» [84].

Обобщенная задача

Выше, начиная с [53], уже упомянуты *обобщенные*-кластерные[13] варианты TSP-PC. Мы опустим обсуждение «обычной» (без условий предшествования) *обобщенной задачи коммивояжера*, ограничившись ссылкой на обзор [2, Ch.13, §1.6 Generalized Traveling Salesman Problem], и перейдем к более редкой и более важной для настоящего исследования *обобщенной* задаче коммивояжера *с условиями предшествования*, которую обозначим GTSP-PC, по аналогии с GTSP и TSP-PC; эта задача сравнительно мало изучена и общепринятого обозначения пока не выработано.

По-видимому, первое формальное описание GTSP-PC (строго говоря, CTSP-PC, см. сноску) представлено в [87] в применении к автоматизации обработки листа металла (сверление, развертывание отверстий, нарезка резьбы — различного диаметра), где *кластерный* характер связан с необходимостью, поместив определенный инструмент в патрон станка, провести все операции с этим инструментом (формирующие кластер), и лишь затем переходить к операциям с другим инструментом, а условия предшествования связаны с характером операций: отверстие нужно *сначала* просверлить и лишь *затем* развертывать. Для решения задачи предложена вариация метода ветвей и границ.

В [88] GTSP-PC применялась в контексте маршрутизации инструмента в машинах листовой резки, для решения предложен эвристический алгоритм; смотрите дальнейшее обсуждение этих задач в предыдущем разделе. В [50] предложено решение GTSP-PC, см. [25]; одно из приложений — минимизация радиационного облучения персонала АЭС при работах по демонтажу энергоблока АЭС, выведенного из эксплуатации; в [89] представлена постановка задачи с *внутренними работами*, позволяющая единообразно рассматривать GTSP-PC и CTSP-PC. В [90] рассматривается GTSP-PC для *двух* коммивояжеров, каждый из которых представляет отдельный кран-штабелер. В [90] отмечено, что параллельное использование нескольких кранов на контейнерных терминалах внедрено сравнительно недавно, в коммерческих программных пакетов для оптимизации их операций используется *жадный алгоритм*. В магистерской диссертации [91] предложена математическая модель GTSP-PC средствами линейной релаксации задач целочисленного программирования (MILP) и несколько эвристических алгоритмов на ее основе; также представлено [92] приложение GTSP-PC к оптимизации работы координатно-измерительных машин (coordinate-measuring machine, CMM) на сборочном конвейере для автомобилей; наилучшие результаты были показаны гибридной эвристикой муравьиной колонии (Hybridized Ant Colony System).

«Обычная», необобщенная задача коммивояжера на узкие места (Bottleneck TSP, BTSP) *без условий предшествования* описана, например, в обзоре [2, Ch. 15]; впервые была поставлена в [93]; в [94] представлены различные методы решения этой задачи и актуальная библиография. Мы не будем далее обсуждать необобщенную BTSP без условий предшествования.

Обобщенная BTSP (назовем ее Bottleneck Generalized TSP, BGTSP), по-видимому, впервые поставлена в [95]; отметим, что формулировка в этой статье на самом деле несколько шире того, что предполагает BGTSP: рассматривалась задача обхода *непрерывных объектов*[14], для решения которой применялось динамическое программирование, а непрерывные множества определенным образом дискретизировались; эта работа полу-

[13] Generalized TSP предполагает посещение *одного* города из каждого мегаполиса; Clustered TSP [20] предполагает посещение *всех* городов в кластере — прежде перехода к следующему. Далее под GTSP-PC мы понимаем оба варианта задачи, с учетом их единого представления через формализм *внутренних работ* [86], см. пример представления в [62] и гл. 2.

[14] задачи об обходе *непрерывных множеств* называются, в частности, TSP with Neighborhoods (TSP с окрестностями), первыми статьями, посвященными таким задачам были [96, 97], см. также подобную задачу с зависимостью от времени [72]; о приложениях см., напр., [98]

чила развитие в [99], где рассмотрено приложение к трассировке проводников на многослойных печатных платах. Условия предшествования в BGTSP (которую с ними будем называть BGTSP-PC) впервые добавлены в [58], где на BGTSP-PC распространен вариант динамического программирования, предложенный в [50]. Затем, в [59] постановка дополнена *зависимостью от списка невыполненных заданий* (следует называть такую задачу PSD-BGTSP-PC).

Качественное исследование TSP-PC. Вопросы сложности

Как уже упоминалось выше, TSP-PC, формально, NP-трудна, поскольку содержит обычную TSP как частный случай. Между тем, очевидно, что сложность конкретного экземпляра связана с разновидностью условий предшествования, хотя и не совсем очевидно, как именно; ясно, что на практическую сложность каждого экземпляра влияет и конкретный метод решения задачи.

В одной из первых работ, посвященных решению задачи, близкой к TSP-PC [45], применялся метод ветвей и границ, пространство решений которого — все *допустимые* по предшествованию маршруты; в этой работе рассматривался особый, узкий класс условий предшествования, «pickup and delivery constraints», которые можно описать как «параллельную композицию двухэлементных цепей» (см. анализ сложности этого класса в контексте ДП [22]) или «независимые адресные пары» (рассматривалось в [100]). Поскольку для метода ветвей и границ наихудшей оценкой производительности является перебор всех маршрутов, было рассчитано их количество — для $2n$ городов-заказчиков получается $(2n)!/2^n$ допустимых маршрутов. Нам неизвестны другие попытки аналогичным образом рассчитать количество допустимых маршрутов в работах по *дискретной оптимизации*, тем не менее, отметим что в целом теория, позволяющая это делать, хорошо разработана в рамках *теории порядков*; в контексте последней, допустимые маршруты — это *линейные продолжения* (linear extension) данного частичного порядка; см., напр., [101], хотя сама по себе задача определения количества линейных продолжений труднорешаема (#P-полная, см. [101]).

Следует отметить, что специфическая для методов решения, использующих линейно-целочисленную релаксацию (MILP), «сложность» TSP-PC — размерность политопа — описана в [41].

В работах по теории расписаний с условиями предшествования была введена *плотность*[15] частичного порядка [102] — отношение мощности данного частичного порядка к мощности линейного на том же множестве — которая затем широко использовалась для оценки сложности экземпляров задач в смысле *ограничительности* условий предшествования, и продолжает использоваться по сей день [103, 104].

В контексте *динамического программирования* сложность решения задачи *полиномиально* зависит [44][16] от количества существенных списков заданий — порядковых идеалов (прямое ДП [16, 52]) или фильтров (попятное ДП [15, 25]), см. определения в разделе 1.1.2. Вопрос о количестве порядковых идеалов представляется менее разработанным чем вопрос о количестве линейных продолжений; в контексте дискретной оптимизации он рассматривался в [22, 23]; кроме того, в [105] напрямую оценивалась сложность ДП-решения (включая число состояний) для простого случая условий предшествования (одна пара сравнимых мегаполисов); также этот вопрос исследовался в статье [100]. Поскольку в общем случае задача подсчета числа порядковых идеалов *труднорешаема* (#P-полная, в связи с полиномиальной сводимостью к задаче о перечислении всех *антицепей*, см. [22, 106]), большой интерес представляют *оценки*, см. [22, 23, 44].

[15] см. формальное определение в разделе 1.4.5

[16] один из основных результатов диссертации; см. раздел 1.4

Отметим также статьи в которых, благодаря особым условиям предшествования, сложность динамического программирования становится в некотором смысле *линейной*: [107, 108].

Цели и задачи работы.

Наша цель состоит в разработке и анализе основанных на динамическом программировании методов и алгоритмов решения задач *маршрутизации* перемещений *с ограничениями* и функциями стоимости перемещений, допускающими зависимость от списка невыполненных заданий, для достижении которой потребовалось решить следующие **задачи**:

1. Провести качественный анализ модели перемещений с условиями предшествования, охватывающей разнообразные практические задачи
2. Построить конструкции решения на основе широко понимаемого *динамического программирования*
3. Оценить влияние *условий предшествования* на вычислительную сложность настоящих методов
4. Разработать и реализовать алгоритмы решения, провести вычислительные эксперименты

Основные положения, выносимые на защиту:

1. Проведено качественное исследование комбинаторной модели перемещений с ограничениями в виде условий предшествования: получены гарантии вычислительной сложности, на основе которых выработан критерий практической разрешимости.
2. Реализован программный комплекс, выполняющий решение задач, соответствующих модели перемещений, точным (динамическое программирование) и эвристическим (усеченное динамическое программирование) методами.
3. Реализован экономичный вариант динамического программирования для обобщенной задачи курьера на узкие места с зависимостью от списка невыполненных заданий (Past Sequence Dependent Bottleneck Generalized TSP-PC, PSD-BGTSP-PC); предложен подход к параллелизации этого варианта динамического программирования для систем с общей памятью; исследована эффективность экономичного динамического программирования в сравнении с другими методами для TSP-PC в классической постановке и в одной постановке с зависимостью функции стоимости перемещений от времени, на примере экземпляров из TSPLIB.
4. Исследовано качество эвристики усеченного динамического программирования для обобщенной задачи курьера на узкие места с зависимостью от списка невыполненных заданий (Past Sequence Dependent Bottleneck Generalized TSP-PC, PSD-BGTSP-PC), получена оценка временно́й сложности для этой задачи; исследовано качество эвристики усеченного динамического программирования в сравнении с другими методами для TSP-PC в обычной постановке и в одной постановке с зависимостью функции стоимости перемещений от времени на примере экземпляров из TSPLIB на различных значениях параметра эвристики.

Научная новизна

В настоящем разделе *римскими* цифрами даны номера работ, опубликованных автором по теме диссертации (см. список далее).

Точное динамическое программирование для обобщенной задачи курьера на узкие места [I,III,IV,VI,VIII–X]. В [X] описана разработанная автором оригинальная схема программной реализации динамического программирования, использовавшаяся в [III,IV,VI,VIII–X], а также оригинальная схема параллелизации вычислений на основе OpenMP. Новизна схемы программной реализации динамического программирования состоит в генерации существенных списков заданий *снизу вверх* [X, Eq. 23], что позволяет и сгенерировать все существенные списки заданий и расчитать все значения функции Беллмана «одновременно», в прямом ходе алгоритма снизу вверх, в отличие от процедуры, описываемой, например, в [25], где предполагается сначала сгенерировать все существенные списки заданий посредством оператора **I** *сверху вниз* и затем, в обратном ходе алгоритма *снизу вверх* рассчитывать значения функции Беллмана.

Кроме того, новым является *иерархическое* устройство основной структуры данных, предназначенной для хранения значений функции Беллмана: каждому существенному списку заданий ставится в соответствие множество пар «база-экстремум состояния», что позволяет сэкономить память за счет того что для всех состояний с одинаковым списком заданий последний хранится в единственной копии, а не повторяется каждый раз в отличие от, например, *плоской* реализации [78, Section 3]; подобная экономия становиться особенно важной в рассматриваемом случае *обобщенной* задачи. Иерархия реализована в виде вложенной хеш-таблицы, с целью обеспечить быстрый доступ к значениям функции Беллмана.

В [Section 5, X] также описана схема параллелизации расчета значений функции Беллмана в рамках одного слоя пространства состояний для систем с общей памятью на основе OpenMP. Новизна схемы заключается в применении конструкций `task`, введенных в OpenMP 3.0, для параллелизации обхода вложенной хеш-таблицы, которую, в отличие от *массива*, невозможно обработать параллельно через директиву `parallel for`, поскольку *хеш-таблица* не гарантирует доступ к элементам за *постоянное* время.

Качественное исследование модели перемещений с условиями предшествования: пространственная и временная сложность [V,XII]. Автором получены оценки пространственной и временной сложности решения TSP-PC динамическим программированием и их приближения сверху и снизу. Метод получения оценок распространяет результаты [22, 23] на TSP-PC; принципиальные отличия между оценками из [XII] и [22, 23] проистекают из различий в пространстве состояний динамического программирования, вызванных различием рассматриваемых задач. Известные оценки [41] применимы исключительно к решениям, использующим линейно-целочисленного программирование, соответственно, не релевантны для рассматриваемого варианта динамического программирования. В [45] представлены оценки только для одного класса условий предшествований (параллельная композиция *двух*элементных цепей).

Результаты о «количестве существенных списков заданий» [V] повторяют часть результатов о количестве идеалов в *последовательно-параллельных* частично упорядоченных множествах (см., напр., [22, Section 3.1]), но получены независимо и выражены на языке *фундированных* отношений (см., напр. [25]), а не частично упорядоченных множеств. Кроме того, в [V] впервые формально описан класс условий предшествования, порождаемый требованием всегда «вырезать *внутренний* контур прежде *внешнего*» в задачах маршрутизации инструмента в машинах листовой резки (лес с исходящей степенью не более 1).

Ультраметрическая задача коммивояжера на узкие места [II]. Доказана тривиальность ультраметрической задачи коммивояжера на узкие места: целевая функция, цена маршрута *на узкие места*, постоянна на всем пространстве решений и совпадает с весом наиболее дорогого возможного перемещения. В диссертации результат обобщен до свойства «если две точки входят в маршрут, в маршруте найдется ребро, равное расстоянию между этими точками [включая самое тяжелое ребро]».

Следует отметить, что последнее свойство было доказано в [109] как характеризация ультраметрик [109, Proposition 3.1], однако, (а) доказательство в [II] получено независимо и несколько проще (в связи с менее емкой формулировкой) чем [109, Proposition 3.1], (б) в [109] не рассматривалась задача коммивояжера на узкие места.

Теоретическая значимость

- Определены пространственная и временная сложность динамического программирования для задачи коммивояжера с условиями предшествования и основанной на нем эвристики *усеченного* динамического программирования; потребность в определении сложности последней связана с тем, что эвристика «в пределе» — при приближении параметра эвристики к мощности наибольшего слоя пространства состояний — переходит в *точное* динамическое программирование.

Практическая значимость

- Экспериментально показано, что динамическим программированием можно эффективно решать определенный класс задач с условиями предшествования; выявлять попадание произвольной данной задачи в этот класс можно априорно, с существенно меньшими затратами времени и ресурсов чем потребовалось бы на попытку запустить точный алгоритм (который, в случае превышения допустимого расходы ресурсов был бы остановлен).

Методология и методы исследования

В статьях, посвященным задачам дискретной оптимизации, можно выделить четыре ключевых методологических аспекта:

1. Строгая постановка задачи, обычно комбинаторная (см., напр., [25,31,74]) или в виде задачи целочисленного программирования (см., напр., [33,34])
2. Формальное описание метода решения, обоснование его корректности
3. Исследование сложности решения задачи указанным методом
4. Испытание метода на общепринятом наборе тестовых экземпляров (напр., задачи TSPLIB [76,110]), на нескольких задачах из практики (например, [88,92,111]), либо на модельных задачах, как правило, случайно сгенерированных с учетом специфики [26]; последние два варианта наиболее характерны для статей, рассматривающих специфические прикладные задачи, содержащие множество особенностей и ограничений

Этой же методологии придерживался и автор диссертации, исследуя варианты динамического программирования и основанной на нем эвристики *усеченного* динамического программирования (restricted dynamic programming [78]) для маршрутных задач с условиями предшествования.

Степень достоверности

Сформулированные в диссертации научные положения, выводы и рекомендации обоснованы теоретическими построениями и экспериментальными данными, полученными в работе, корректны математически и согласуются с известным опытом создания моделей и алгоритмов решения маршрутных задач.

Апробация работы

Основные результаты работы представлялись в нижеследующих докладах:

- Ченцов А. Г., Салий Я. В. *Об одной задаче на узкие места* // Всероссийская конференции СМО-2011 (Россия, Челябинск, 2011 г)
- Салий Я. В. *Об ультраметрической задаче коммивояжера на узкие места* // Современные проблемы математики. Международная (43-я Всероссийская) молодёжная школа-конференция (Россия, Екатеринбург, 2012)
- Салий Я. В., Ченцов А. Г. *О маршрутной задаче на узкие места с внутренними работами и условиями предшествования* // Международная конференция «Дискретная оптимизация и исследование операций»: Материалы конференции (Россия, Новосибирск, 2013)
- Ченцов А. Г., Салий Я. В. *Неаддитивная задача маршрутизации с условиями предшествования* // Алгоритмический анализ неустойчивых задач: тез. докл. Всерос. конф. с междунар. участием, посвящ. памяти В. К. Иванова (Россия, Челябинск, 2014)
- Салий Я. В. *О прямом и попятном динамическом программировании в маршрутных задачах с условиями предшествования и алгоритмах генерации допустимых подзадач* // XV Всероссийская конференция «Математическое программирование и приложения» (Россия, Екатеринбург, 2015)
- Ченцов А. Г., Салий Я. В. *Задача маршрутизации «на узкие места» с ограничениями и усложненными функциями стоимости* // 8-й Всерос. мультиконф. (Россия, Дивноморское, 2015)
- Salii Y. V. *Restricted Dynamic Programming Heuristic for Precedence Constrained Bottleneck Generalized TSP* // 1st Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists (Russia, Yekaterinburg, 2015)
- Salii Y. V., Ivanko E. E. *Dynamic programming and greedy heuristic in Camera Trap Traveling Salesman Problem* // Международная (46-я Всероссийская) конференция «Современные проблемы математики и ее приложений» (Россия, Екатеринбург, 2016).
- Салий Я. В. *Испытание эвристики усеченного динамического программирования на известных экземплярах задачи коммивояжера с условиями предшествования* // Международная (47-я Всероссийская) конференция «Современные проблемы математики и ее приложений» (Россия, Екатеринбург, 2017).
- Salii Y. V. *Order-theoretic characteristics and dynamic programming for Precedence Constrained Traveling Salesman Problem* // Fourth Russian–Finnish Symposium on Discrete Mathematics (Finland, Turku, 2017)

- Салий Я. В. *Задача коммивояжера с условиями предшествования: точное динамическое программирование и эвристика на его основе, зависимость размерности экземпляра задачи от условий предшествования.* // Семинар «Дискретные экстремальные задачи», ИМ им. С. Л. Соболева СО РАН (Россия, Новосибирск, 2017)

- Салий Я. В. *Представление диссертации «Некоторые методы решения маршрутных задач с условиями предшествования»*// Семинар «Математическое моделирование и дискретная оптимизация» ОФ ИМ им. С. Л. Соболева СО РАН (Россия, Омск, 2017)

- Салий Я. В. *Представление диссертации «Некоторые методы решения маршрутных задач с условиями предшествования»* // Научный семинар по информационным технологиям ЮУрГУ (Россия, Челябинск, 2017)

Список работ, опубликованных автором по теме диссертации

Основные результаты по теме диссертации изложены в 12 печатных и электронных публикациях, 3 из которых изданы в журналах, рекомендованных ВАК [III,V,VIII], 2 — в виде статьей в трудах конференций, входящих в список рекомендованных ВАК [X,XI] (индексируемых системой Scopus).

I Ченцов А. Г., Салий Я. В., Об одной задаче на узкие места // Сборник трудов Всероссийской конференции СМО-2011 (Челябинск, 28 ноября – 3 декабря 2011 г.). Челябинск: Издательский центр ЮУрГУ, 2011. С. 85–92.

II Салий Я. В., Об ультраметрической задаче коммивояжера на узкие места // Тезисы Международной (43-й Всероссийской) молодежной школы-конференции (Екатеринбург, 29 января – 5 февраля 2012 г.). Екатеринбург: Изд. УМЦ УПИ, 2012. С. 287–289.

III Ченцов А. Г., Салий Я. В., Об одной маршрутной задаче на узкие места с внутренними работами // Вестник Тамбовского Университета (сер. Естественные и технические науки), 2012. Т. 17. № 3. С. 827–847.

IV Салий Я. В., Ченцов А. Г., О маршрутной задаче на узкие места с внутренними работами и условиями предшествования // Материалы конференции «Международная конференция Дискретная оптимизация и исследование операций» (Новосибирск, 24 – 28 июня 2013 г.), Новосибирск: Изд-во Ин-та математики, 2013. С. 134.

V Салий Я. В., Влияние условий предшествования на вычислительную сложность решения маршрутных задач методом динамического программирования // Вестник Удмуртского университета. Математика. Механика. Компьютерные науки, 2014. № 1. С. 76–86.

VI Ченцов А. Г., Салий Я. В., Неаддитивная задача маршрутизации с условиями предшествования // Тезисы докладов Всероссийской конференции с международным участием, посвященной памяти В. К. Иванова Алгоритмический анализ неустойчивых задач (Челябинск, 10 – 14 ноября 2014 года). Челябинск: Издательский центр ЮУрГУ, 2014. С. 166–167.

VII Салий Я. В., О прямом и попятном динамическом программировании в маршрутных задачах с условиями предшествования и алгоритмах генерации допустимых подзадач // Инф. бюллетень Ассоциации математического программирования, 2015. № 13. С. 168–169.

VIII Chentsov A. G., Salii Ya. V., A model of "nonadditive" routing problem where the costs depend on the set of pending tasks // Vestnik YuUrGU. Ser. Mat. Model. Progr., 2015. V. 8. no. 1. P. 24–45.

IX Ченцов А. Г., Салий Я. В., Задача маршрутизации «на узкие места» с ограничениями и усложненными функциями стоимости // Материалы 8-й Всерос. мультиконф. (МКПУ-2015): в 3-х т. Ростов-на-Дону: Изд. ЮФУ, 2015. Т. 1. С. 214–216.

X Salii Ya., Restricted Dynamic Programming Heuristic for Precedence Constrained Bottleneck Generalized TSP // Proceedings of the 1st Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists (Yekaterinburg, November 17th, 2015), CEUR Workshop Proceedings. V. 1513. P. 85–108. http://ceur-ws.org/Vol-1513/#paper-10.

XI Salii Ya. V., Ivanko E. E., Dynamic programming and greedy heuristic in Camera Trap Traveling Salesman Problem // Proceedings of the 47th International Youth School-conference "Modern Problems in Mathematics and its Applications" (Yekaterinburg, Russia, January 31–February 6, 2016), CEUR Workshop Proceedings. V. 1662. P. 215–219. http://ceur-ws.org/Vol-1662/mpr5.pdf.

XII Salii Ya., Order-theoretic characteristics and dynamic programming for Precedence Constrained Traveling Salesman Problem // Proceedings of the Fourth Russian Finnish Symposium on Discrete Mathematics (Turku, May 16–19 2017), Juhani Karhumäki, Yuri Matiyasevich, Aleksi Saarela (Eds.), — Turku Centre for Computer Science. V. 26. P. 152–164. http://urn.fi/URN:ISBN:978-952-12-3547-4

Свидетельства о государственной регистрации программ для ЭВМ.

1. Я. В. Салий, Программа вычисления оптимального решения обобщенной задачи курьера на узкие места с условиями предшествования и зависимостью от списка невыполненных заданий методом динамического программирования // ФЕДЕРАЛЬНАЯ СЛУЖБА ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ, (12) ГОСУДАРСТВЕННАЯ РЕГИСТРАЦИЯ ПРОГРАММЫ ДЛЯ ЭВМ. — № 2015661435. — дата регистрации 27.10.2015.

2. Я. В. Салий, Программа расчета сложности задачи о маршрутизации инструмента в машинах листовой резки с целью сокращения холостого хода // ФЕДЕРАЛЬНАЯ СЛУЖБА ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ, (12) ГОСУДАРСТВЕННАЯ РЕГИСТРАЦИЯ ПРОГРАММЫ ДЛЯ ЭВМ. — № 2015661434. — дата регистрации 27.10.2015.

3. Я. В. Салий, Модульная программа вычисления оптимальных и эвристических решений задачи курьера и ее обобщений методом динамического программирования // ФЕДЕРАЛЬНАЯ СЛУЖБА ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ, (12) ГОСУДАРСТВЕННАЯ РЕГИСТРАЦИЯ ПРОГРАММЫ ДЛЯ ЭВМ. — № 2018614740. — дата регистрации 17.04.2018.

Разделение вкладов в публикациях в соавторстве.

В работах [I,III,IV,VI,VIII,IX] доказательство корректности уравнения Беллмана принадлежит А. Г. Ченцову. Во всех этих работах Я. В. Салий писал реализацию алгоритма и генератор экземпляров задачи, проводил вычислительный эксперимент; в работах [IV,VI,VIII,IX] разрабатывал и реализовывал схему параллелизации алгоритма для систем с общей памятью; в работе [IX] Я. В. Салию также принадлежит модель функции стоимости перемещений ремонтной бригады, допускающей экстренный вызов на не пройденный объект.

В работе [XI] Я. В. Салий разработал обновленный, более эффективный вариант динамического программирования для рассматриваемой задачи, реализовал его на C++ и испытал на наборе тестовых экземпляров; кроме того, реализовал на Haskell программу расчета количества состояний этого алгоритма и реализовал на C++ генератор экземпляров рассматриваемой задачи. Е. Е. Иванко реализовал эвристику (жадный алгоритм) и испытал ее на наборе тестовых задач.

Структура работы

введение дает общую характеристику работы как того требует ГОСТ Р 7.0.11—2011 «Диссертация и автореферат диссертации. Структура и правила оформления».

глава 1 посвящена вопросам решения TSP-PC и ее варианта с зависимостью от времени, TD-TSP-PC, точным методом (динамическое программирование) и эвристическим, допускающим переход к точному (усеченное ДП, restricted dynamic programming) на примере задач из TSPLIB — результаты [77], и о методике качественного, априорного анализа сложности конкретных экземпляров (TD)-TSP-PC, в частности, гарантиях на пространственную и временную сложность [44, 100], а также о вычислительном эксперименте на программном комплексе, реализующем точное и усеченное ДП и описанном в главе 4. Соответствует пунктам 3–4 из «основных результатов диссертации».

глава 2 посвящена вопросам точного и эвристического вариантов динамического программирования для PSD-BGTSP-PC, а также параллельной реализации точного метода. Соответствует пунктам 3–4 «основных результатов», на материале работ [60–62, 112, 113].

глава 3 содержит то, что не поместилось в первые две главы — результаты, связанные с задачами *без* условий предшествования; в частности, о тривиальности ультраметрической BTSP, по результатам [114] и о решении «задачи о камерах-ловушках»[17] и ее пространственной сложности, по результатам [116].

глава 4 описывает программный комплекс [117], продолжающий и расширяющий [118][18], применявшийся [77] в вычислительном эксперименте главы 1

заключение представляет изложение итогов исследования, рекомендаций и перспектив дальнейшей разработки темы.

[17] очень слабый вариант 1-Pickup and Delivery TSP [115]

[18] различные версии применялись в [61, 62] и ранее

Глава 1

Задача коммивояжера с условиями предшествования с зависимостью от списка невыполненных заданий / PSD-TSP-PC

1.1 Определения и обозначения

1.1.1 Общие определения

Здесь и далее символ $\triangleq$ обозначает *равенство по определению*; *неупорядоченные пары* объектов записываются в фигурных скобках $\{a, b\}$, *упорядоченные пары* (УП) обозначаются круглыми скобками, (a, b). Пусть $z = (a, b)$ — некоторая УП; введем ее *проекции*: обозначим $\mathrm{pr}_1(z)$ первый элемент ($\mathrm{pr}_1(z) = a$), а $\mathrm{pr}_2(z)$ — второй элемент. Более двух упорядоченных объектов также записываются в круглых скобках, например, $(a_1, \ldots, a_k)$, либо в виде кортежа $(a_i)_{i=1}^k$; *множество* элементов кортежа вводим следующим образом:

$$\overline{(a_i)_{i=1}^k} \triangleq \bigcup_{i \in 1..k} \{a_i\}.$$

Замечание. Упорядоченные пары определяются по Куратовскому, $(a, b) \triangleq \{\{a\}, \{a, b\}\}$ [119, гл. 2, § 3]; упорядоченные *тройки* (a, b, c) и более длинные *кортежи* описываются в виде *биекции* множества натуральных чисел $1, \ldots, k$[1], где $k \in \mathbb{N}$ — длина кортежа, на множество элементов кортежа; например,

$$(a, b, c) \triangleq \{(1, a), (2, b), (3, c)\}.$$

Символ *вложения* $\subset$ понимается в *нестрогом* смысле: подмножества могут быть *несобственными*. *Семейством* называется множество, каждый элемент которого является множеством. Семейство всех (всех непустых) подмножеств некоторого множества K обозначаются $\mathcal{P}(K)$ ($\mathcal{P}'(K)$). Функции, аргументами и значениями которых являются *множества*, называются *операторами*; их аргумент указывается в квадратных скобках, например $\mathbf{E}[I]$. Подобная запись также используется для обозначения применения функции ко множеству аргументов, например, $v(s)$ — значение одного состояния, а $v[\mathcal{S}]$ — образ $\mathcal{S}$ под действием v, всевозможные значения v на множестве состояний $\mathcal{S}$, как укороченная запись выражения «v, примененная ко всем $s \in \mathcal{S}$».

[1] либо другого индексного множества; все рассматриваемые индексные множества конечны

Интервал *целых* чисел определяем и обозначаем как

$$p\,..\,q \triangleq \{a \in \mathbb{Z} \,|\, (a \geqslant p) \wedge (a \leqslant q)\};$$

отметим, что в случае $p > q$ данное определение корректно, и интервал становится *пустым множеством*.

Перестановкой элементов некоторого *конечного* множества K полагаем взаимно однозначную функцию (биекцию) K на себя. Множество всех биекций K (интерпретируемых *исключительно* как перестановки) обозначим $(\mathrm{bi})[K]$; сами перестановки, как правило, обозначаются строчными греческими буквами.

Множества, над которыми рассматриваются перестановки, конечны, следовательно, их элементы всегда можно пронумеровать числами $1\,..\,|K|$, таким образом, допустимо говорить о том, что некоторая перестановка (биекция) $\alpha \in (\mathrm{bi})[K]$ присваивает каждому элементу $k \in K$ некоторый номер $i_k \in 1\,..\,|K|$. Элемент K, находящийся на i-м месте в перестановке обозначается α_i.

Представление перестановки как *биекции* гарантирует, что для каждой перестановки $\alpha \in (\mathrm{bi})[K]$ существует *обратная* перестановка, композиция которой с постановкой α является *тождественной* функцией; перестановку, обратную к α обозначаем α^{-1}; при этом, α_k^{-1} обозначает индекс элемента $k \in K$ в перестановке α и $\alpha_{\alpha_k^{-1}} = k$.

В контексте TSP и ее обобщений, перестановки называются *маршрутами* и понимаются как (линейный) порядок посещения элементов K. См. подробное обсуждение *перестановок* в контексте задач *комбинаторной оптимизации* в [120, гл. 1].

1.1.2 Теоретико-порядковые определения

Фиксируем некоторое натуральное n и интервал $1\,..\,n$; пусть на нем определено отношение частичного порядка $<_P$. В некоторых определениях мы также используем *нестрогую* версию, $\leqslant_P$, где $a \leqslant_P b \triangleq (a <_P b) \vee (a = b)$ и $a \geqslant_P b \triangleq (a >_P b) \vee (a = b)$.

Операторы взятия множества *наибольших* и *наименьших* элементов некоторого множества K, $K \subset 1\,..\,n$, в силу отношения частичного порядка $<_P$ определим согласно [121, Def. 2.1.5, 2.3.2]:

$$\mathrm{Max}[K] \triangleq \{m \in K \mid \forall k \in K (m \geqslant_P k) \vee (m \parallel k)\}; \tag{1.1.1}$$

$$\mathrm{Min}[K] \triangleq \{m \in K \mid \forall k \in K (m \leqslant_P k) \vee (m \parallel k)\}; \tag{1.1.2}$$

здесь $\parallel$ — символ *несравнимости* в $<_P$:

$$a \parallel b \triangleq \neg(a \leqslant_P b) \wedge \neg(b \leqslant_P a).$$

Подмножество $K \subset 1\,..\,n$ называется *цепью*, если оно линейно упорядочено $(\forall a, b \in K (a \leqslant_P b) \vee (b \leqslant_P a))$ [122, Def. 1.27]. Подмножество $K \subset 1\,..\,n$ называется *антицепью*, если его элементы попарно несравнимы $(\forall a, b \in K (a \neq b) \to (a \parallel b))$ [122, Def. 1.29]. Наибольшая мощность антицепи в $<_P$ называется *шириной* частичного порядка $(1\,..\,n, <_P)$ и обозначается w [122, Def. 1.30].

Для описания состояний ДП ниже нам потребуются *порядковые идеалы — замкнутые вниз* подмножества [122, Def. 1.42]. В контексте отношения порядка $<_P$, определим семейство идеалов следующим образом:

$$\mathcal{I} \triangleq \left\{ I \in \mathcal{P}\big(1\,..\,n\big) \;\middle|\; \forall i \in I \; \forall j \in 1\,..\,n \; \big(j \leqslant_P i\big) \to \big(j \in I\big) \right\} \tag{1.1.3}$$

и упорядочим его по мощности: $\forall k \in 0\,..\,n \; \mathcal{I}_k \triangleq \{I \in \mathcal{I} \mid |I| = k\}$. Отметим, что *пустое множество* мы также полагаем (тривиальным) идеалом.

Двойственный объект, *порядковые фильтры*, определяется следующим образом:

$$\mathcal{F} \triangleq \Big\{ F \in \mathcal{P}(1\,..\,n) \;\Big|\; \forall i \in F\ \forall j \in 1\,..\,n\ \big(j \geqslant_P i\big) \to \big(j \in F\big) \Big\}.$$

Замечание. Употребление термина *фильтр* в теории порядков отличается от принятого в общей топологии (см., напр. [123, гл. 1, § 6], [124, § 1.6]), где фильтром называется *семейство* множеств, каждый элемент которого в некотором смысле обладает свойствами *порядкового* фильтра (в контексте порядка по *вложению* множеств $\subset$), в частности, замкнутостью[2] по конечным пересечениям — за исключением *пустого* множества $\varnothing$, которое обычно полагается порядковым фильтром [122, § 1.4.2], но не включается в фильтр в смысле [123, гл. 1, § 6].

Например, для произвольного частично упорядоченного множества $(X, <_Q;)$ над (непустым) множеством X множество всех *непустых* порядковых фильтров $\mathcal{F} \setminus \{\varnothing\}$ будет фильтром в смысле [123, гл. 1, § 6].

Здесь и ниже, мы опускаем прилагательное *порядковый* и пишем просто «идеал» («фильтр»).

Для *эффективного*[3] порождения всех идеалов, мы используем оператор существенного продолжения» $\mathbf{E}$, $\mathbf{E}\colon \mathcal{I} \setminus 1\,..\,n \to \mathcal{P}'(1\,..\,n)$, определенный по правилу

$$\mathbf{E}[I] \triangleq \{m \in 1\,..\,n \setminus I \mid I \cup \{m\} \in \mathcal{I}\}; \tag{1.1.4}$$

конструктивно он представляет собой взятие множества наименьших элементов дополнения до $1\,..\,n$ своего аргумента. Докажем это в форме следующей теоремы о характеризации [44, Theorem 1]:

Теорема 1.1. *Для всех $I \in \mathcal{I} \setminus 1\,..\,n$ и $m \in 1\,..\,n \setminus I$, следующие условия эквивалентны:*

(i) $I \cup \{m\} \in \mathcal{I}$ $(m \in \mathbf{E}[I])$;

(ii) $m \in \mathrm{Min}[1\,..\,n \setminus I]$.

Доказательство. Начнем с импликации (i)→(ii). Фиксируем $I \in \mathcal{I} \setminus 1\,..\,n$ и $m \in 1\,..\,n \setminus I$ такие что $I \cup \{m\} \in \mathcal{I}$. От противного, предположим $m \notin \mathrm{Min}\Big[1\,..\,n \setminus I\Big]$. Значит, должен существовать меньший элемент, $\exists m' \in 1\,..\,n \setminus I\colon (m' \neq m) \wedge (m' \leqslant_P m)$. Поскольку $m' \notin I \cup \{m\}$ и $m' \leqslant_P m$, множество $I \cup \{m\}$ не является идеалом, что *противоречит* предположению (i); импликация (i)→(ii) доказана.

Докажем импликацию в обратном направлении, (ii)→(i). Фиксируем идеал $I \in \mathcal{I} \setminus 1\,..\,n$ и $m \in \mathrm{Min}\Big[1\,..\,n \setminus I\Big]$; от противного, предположим, что $I \cup \{m\}$ — не идеал. Поскольку I — идеал по определению, должно существовать $m' \in 1\,..\,n \setminus I$ такое что $(m' \neq m) \wedge (m' \leqslant_P m)$. Однако второй конъюнкт в этом выражении *противоречит* предположению $m \in \mathrm{Min}\Big[1\,..\,n \setminus I\Big]$, что доказывает импликацию (ii)→(i), а с ней и всю теорему. □

[2] пересечение любых двух порядковых фильтров остается порядковым фильтром [122, § 1.4.2]; поскольку множество-носитель отношения порядка является порядковым фильтром по определению, с учетом замечания о второй аксиоме фильтра [123, гл. 1, § 6], получается, что порядковые фильтры замкнуты по *конечным* пересечениям

[3] *неэффективным* следует признать порождение всех подмножеств $1\,..\,n$ с последующим исключением тех, которые не являются идеалами; о других методах эффективного порождения идеалов см. [122, Appendix. A.2.2]. Сложность нашего метода подробнее обсуждается в доказательстве предложения 1.2

Двойственный $\mathbf{E}$ оператор «допустимых точек выхода из идеала»[4] $\mathbf{I}\colon \mathcal{I} \setminus \{\varnothing\} \to \mathcal{P}'(1\,..\,n)$ определяется следующим образом:

$$\mathbf{I}[I] \triangleq \left\{ m \in I \;\middle|\; I \setminus \{m\} \in \mathcal{I} \right\};$$

для него возможна аналогичная характеризация,

Теорема 1.2. *Для всех $I \in \mathcal{I} \setminus \{\varnothing\}$, $m \in 1\,..\,n$, следующие условия эквивалентны:*

(i) $I \setminus \{m\} \in \mathcal{I}$ $(m \in \mathbf{I}[I])$;

(ii) $m \in \mathrm{Max}[I]$.

Доказательство. Аналогично Теореме 1.1.

Для фильтров подобные операторы допускают двойственную характеризацию,

$$\mathbf{E}_{\mathrm{F}}[F] \equiv \mathrm{Max}[1\,..\,n \setminus F];$$
$$\mathbf{I}_{\mathrm{F}}[F] \equiv \mathrm{Min}[F].$$

1.2 Постановка задачи

1.2.1 Функция стоимости перемещений

Мы рассматриваем задачу типа *гамильтоновой цепи*[5] (см. [18]). Пусть $n \in \mathbb{N}$ обозначает количество городов, которые должен обойти агент. В их число *не включаются* некоторые фиксированные *точка старта* 0 (далее «база») и *точка финиша* (далее «терминал») $\mathbb{t} \triangleq n+1$; обозначения $\mathbb{t}$ и $n+1$ равнозначны и взаимозаменяемы. В части, не связанной с обобщенной задачей (см. [2, Ch. 13]), мы не различаем *города* и их *индексы*.

Затраты на перемещение между городами описываются *функцией стоимости* $\mathfrak{c}$, $\mathfrak{c}\colon \mathcal{P}(1\,..\,n) \times 0\,..\,n \times 1\,..\,n+1 \to \mathbb{R}$. Выражение $\mathfrak{c}(K,a,b)$ определяет затраты на перемещение из a в b при условии что во всех городах из K агент уже побывал после выхода из базы 0: либо $a \in K$, либо $(a=0) \wedge (K=\varnothing)$ — условимся не включать ни *базу*, ни *терминал* во множество пройденных городов; таким образом, стоимость перемещений из базы 0 в некий город b обозначается $\mathfrak{c}(\varnothing,0,b)$.

Замечание. Функция стоимости с зависимостью от множества *уже посещенных* городов более удобна в *прямом* динамическом программировании [16, 52]; эквивалентный вариант, более удобный в *попятном* [15, 25], введен в [53] и будет обсуждаться в разделе вычислительного эксперимента 1.5.

Общие, *агрегированные* затраты на посещение *всех* городов (стоимость решения) — целевая функция — определяются в виде функции маршрута,

$$\mathfrak{C}[\alpha] \triangleq \mathfrak{c}(\varnothing,0,\alpha_1) \oplus \mathfrak{c}(\{\alpha_1\},\alpha_1,\alpha_2) \oplus \ldots \oplus \mathfrak{c}(1\,..\,n \setminus \{\alpha_n\},\alpha_{n-1},\alpha_n) \oplus (1\,..\,n,\alpha_n,\mathbb{t}); \quad (1.2.1)$$

здесь α — некоторый маршрут, а $\oplus\colon \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ — операция *агрегирования затрат*.

Замечание. Мы определяем стоимости перемещений в виде вещественных чисел $\mathbb{R}$; между тем, ДП-решения не привязаны к $\mathbb{R}$ и могут применяться для любого другого вполне упорядоченного множества.

---

[4] соотв., допустимых точек *входа* в *фильтра*; по аналогии с выражением из [125], «интерфейс» соответствующего множества

[5] такая формулировка чаще встречается в Vehicle Routing Problem (VRP) [1]

Корректность *прямой* процедуры ДП обеспечивается *неубыванием* $\oplus$ по *первому аргументу*

$$\forall x, y, z \in \mathbb{R}\ \big(x \geqslant y\big) \to \big((x \oplus z) \geqslant (y \oplus z)\big), \tag{1.2.2}$$

и *лево*ассоциативностью $\oplus$

$$\forall x, y, z \in \mathbb{R}\ \big(x \oplus y \oplus z\big) = \big((x \oplus y) \oplus z\big). \tag{1.2.3}$$

Для *попятной* процедуры требуются, соответственно, *неубывание* по *второму аргументу* и *право*ассоциативность. Эти свойства обеспечивают аналог *разложимости* целевой функции [57, Ch. 9]. Отметим также, что авторам [53] удалось обойтись без ассоциативности операции агрегирования.

Данным свойствам удовлетворяет обычное арифметическое сложение $+$, агрегирующее затраты в классической TSP, а также агрегирование «узких мест»

$$a \oplus b = \max\{a, b\},$$

характерное для задачи коммивояжера на узкие места (Bottleneck TSP) [2, Ch. 15]; здесь и далее мы имеем в виду именно эти операции агрегирования затрат, хотя допустимо применение и других, например, умножения.

1.2.2 Условия предшествования

Условия предшествования формулизуем в виде строгого *частичного порядка* $P \triangleq \big(1\,..\,n, <_P\big)$ на множестве городов. Там, где это не приведет к путанице, мы не будем различать *отношение порядка* $<_P$ и сам порядок в виде пары $(1\,..\,n, <_P)$, называя и то и другое «порядком». Если $a <_P b$, то город a должен посещаться *прежде* города b.

Кроме отношения порядка [22, 23], условия предшествования также определяют через *ациклический орграф* [24] и *фундированное бинарное отношение* [25, 26]. В случае конечного числа городов, все три представления эквивалентны (например, в смысле совпадения допустимых маршрутов), что следует, в частности, из того, что отношение покрываемости определяет частичный порядок [27, гл. I, § 2, лемма 1]; см. также теоретико-графовую интерпретацию [28].

Множество *допустимых маршрутов* определяется следующим образом:

$$\mathbb{A} \triangleq \Big\{\alpha \in (\mathrm{bi})[1\,..\,n] \,\Big|\, \forall a, b \in 1\,..\,n\ \big(a <_P b\big) \to \big(\alpha_a^{-1} < \alpha_b^{-1}\big)\Big\}. \tag{1.2.4}$$

Сама задача состоит в отыскании *допустимого* маршрута, доставляющего минимум стоимости,

$$\mathfrak{C}[\alpha] \underset{\alpha \in \mathbb{A}}{\longrightarrow} \min. \tag{PSD-TSP-PC}$$

Обычная задача коммивояжера (TSP) формально является частным случаем (PSD-TSP-PC), следовательно, настоящая задача наследует ее NP-трудность.

1.3 Динамическое программирование для PSD-TSP-PC

Мы интерпретируем динамическое программирование как разложение, или «погружение», исходной, *полной* задачи, в семейство меньших задач, называемых *подзадачами* или *состояниями* ДП (также используется термин «позиции», происходящий из теории управления [25]), которые решаются рекурсивным образом так, что, если известны решения всех «предыдущих» подзадач, то текущая, исходя из этих сведений решается «легко», что

обеспечивается соблюдением *принципа оптимальности*, который можно понимать как «в оптимальном решении каждая подзадача решается оптимально».

Подходящее определение пространства состояний позволяет существенно сократить как пространственную, так и временную сложность ДП-решения; подробнее мы обсудим это в разделе 1.4, посвященном сложности.

1.3.1 Состояния в динамическом программировании

Состояниями ДП полагаем *упорядоченные пары* вида (I, x), где I — множество городов, *уже посещенных* агентом после выхода из базы 0, также называемое *списком заданий*,[6] а $x \in 1\,..\,n{+}1 \setminus I$ обозначает город, где агент в данный момент находится. Множество всех состояний описывается следующим образом:

$$\mathcal{S} \triangleq \Big\{(I,x) \in \mathcal{I} \setminus \{1\,..\,n\} \times 1\,..\,n \;\Big|\; x \in \mathbf{E}[I]\Big\} \cup \Big\{\big(1\,..\,n, \mathbb{t}\big)\Big\}; \tag{1.3.1}$$

стратифицируем его по мощности списков заданий, $\mathcal{S}_k \triangleq \{(I,x) \in \mathcal{S} \mid |I| = k\}$. Каждый элемент этого разбиения пространства состояний назовем *слоем*. Связь между состояниями аналогична связи между идеалами, выражаемой операторами $\mathbf{E}$ и $\mathbf{I}$ (см. теоремы 1.2,1.1); ввиду этой связи, будем говорить, что состояние $s' = (I', x')$ *покрывает* состояние $s = (I, x)$ если $\big(I \cup \{x\} = I'\big)$.

Отметим два особых слоя: $\mathcal{S}_0$ с пустами списками заданий задают начальные условия задачи; $\mathcal{S}_n$ содержит *единственное* состояние $(1\,..\,n, \mathbb{t})$, обозначающее *полную задачу*, где посещены все города и агент достиг терминала $\mathbb{t}$.

Цена (значение) состояния — оптимальное значение *целевой функции* $\mathfrak{C}$ на подзадаче, соответствующей данному состоянию, определяется следующим образом:

$$v(I,x) \triangleq \begin{cases} \mathfrak{c}(\varnothing, 0, x), & I = \varnothing; \\ \min\limits_{\alpha \in \mathbb{A}_I} \Big\{ \mathfrak{c}(\varnothing, 0, \alpha_1) \oplus \ldots \oplus \mathfrak{c}\Big(\overline{(\alpha_i)_{i=1}^{|I|-1}}, \alpha_{|I|-1}, \alpha_{|I|}\Big) \\ \qquad\qquad \oplus \mathfrak{c}\Big(\overline{(\alpha_i)_{i=1}^{|I|}}, \alpha_{|I|}, x\Big)\Big\}, & \text{иначе}; \end{cases}$$

здесь $\mathbb{A}_I$ — множество *допустимых* маршрутов на I (*частичных маршрутов* в терминологии [25]), определяемое аналогично (1.2.4):

$$\mathbb{A}_I \triangleq \Big\{\alpha \in (\mathrm{bi})[I] \;\Big|\; \forall a, b \in I \;\big(a <_P b\big) \to \big(\alpha_a^{-1} < \alpha_b^{-1}\big)\Big\};$$

отметим, что такие *частичные* маршруты являются *префиксами* «полных» допустимых маршрутов из $\mathbb{A}$.

Функция Беллмана определяет цену состояния через цены *покрываемых* состояний,

$$\mathrm{BF}(I,x) \triangleq \min_{m \in \mathbf{I}[I]} \big\{v(I \setminus \{m\}, m) \oplus \mathfrak{c}(I, m, x)\big\},$$

и *корректность* ДП означает, что для каждого состояния функция Беллмана *совпадает* с его ценой — исключая начальные условия $\mathcal{S}_0$, для которых нет смысла говорить о *покрываемых* ими состояниях.

Теорема 1.3. *Если операция $\oplus$ монотонно не убывает по первому аргументу и левоассоциативна, то для всех $(I,x) \in \mathcal{S} \setminus \mathcal{S}_0$ выполнено*

$$v(I,x) = \mathrm{BF}(I,x) = \min_{m \in \mathbf{I}[I]} \big\{v(I \setminus \{m\}, m) \oplus \mathfrak{c}(I, m, x)\big\}. \tag{1.3.2}$$

[6] термин используется, например, в [25]; предпочитаемый нами перевод на английский — «task set» [44,62,77]; хотелось избежать ассоциаций со структурой данных «список» (list), в которой элементы непременно упорядочены

Эта теорема, среди прочего, устанавливает корректность «экономичного» определения состояний (1.3.1): согласно (1.3.2), примененного к полной задаче, для получения *оптимального* решения достаточно знать цены состояний $\mathcal{S} \setminus \mathcal{S}_n$; соответственно, нет нужды использовать все состояния, определяемые в [15, 16]. Численное выражение этой экономии будет обсуждаться в разделе 1.4.

Отметим, что настоящая теорема [77, Theorem 1] фактически следует из [53, Теорема 1] или, скорее, некоторой двойственной ей теоремы — мы доказываем ее для *прямого* варианта ДП; тем не менее, мы полагаем возможным привести отдельное доказательство, поскольку оно (а) более кратко — не рассматриваются *обобщенные* задачи, использован язык теории порядков вместо языка фундированных отношений; (б) представлено в постановке задачи на минимум; (в) дано в прямом направлении.

Напомним, что в качестве операции $\oplus$ может выступать как обычное сложение (получим TSP-PC), так и операция взятия наибольшего из двух элементов (получим BTSP-PC, задачу коммивояжера с условиями предшествования на узкие места); допустимо и использование умножения и других операций, удовлетворяющих требованиям теоремы 1.3.

Доказательство. **Тривиальный случай: $|I| = 1$.** В тривиальном случае определения $v(I, x)$ и $\mathrm{BF}(I, x)$ *совпадают*: достаточно заметить, что для всех одноэлементных списков заданий (то есть, $\forall I \in \mathcal{I}_1$) выполнено $\mathbf{I}[I] \equiv I$.

Общий случай: $|I| = k$, $k \in 2\,..\,n$. Зафиксируем произвольное $k \in 2\,..\,n$ и состояние (I, x); докажем желаемое двумя частями.

Часть I. $\mathrm{BF}(I, x) \leqslant v(I, x)$. Фиксируем маршрут $\alpha \in \mathbb{A}_I$, доставляющий минимум в определении $v(I, x)$; обозначим последний его элемент $z \triangleq \alpha_k$. Рассмотрим «шаг назад» — список заданий $Q \triangleq I \setminus \{z\}$, состояние (Q, z) и маршрут $\check{\alpha} \triangleq (\alpha_1, \ldots, \alpha_{k-1})$, — ни что иное как α без своего последнего элемента. Поскольку $\alpha \in \mathbb{A}_I$, очевидно, $\check{\alpha} \in \mathbb{A}_Q$. По определению v и свойствам min, получаем

$$v(Q, z) \leqslant \mathfrak{c}(\varnothing, 0, \alpha_1) \oplus \ldots \oplus \mathfrak{c}\Big(\overline{(\alpha_i)_{i=1}^{k-2}}, \alpha_{k-2}, \alpha_{k-1}\Big) \oplus \mathfrak{c}\Big(\overline{(\alpha_i)_{i=1}^{k-1}}, \alpha_{k-1}, z\Big);$$

нельзя гарантировать, что $\check{\alpha}$ *оптимален* для (Q, z).

Из определения BF и свойств min, получаем

$$\mathrm{BF}(I, x) = \min_{m \in \mathbf{I}[I]} \big\{v(I \setminus \{m\}, m) \oplus \mathfrak{c}(I, m, x)\big\} \leqslant v(Q, z) \oplus \mathfrak{c}(Q \cup \{z\}, z, x);$$

аналогично, нет гарантии что именно z доставляет минимум в $\mathrm{BF}(I, x)$. Подставляя полученную выше оценку $v(Q, z)$ и учитывая неубывание $\oplus$ по первому аргументу (1.2.2), заключаем что

$$\begin{aligned} v(Q, z) \oplus \mathfrak{c}(Q \cup \{z\}, z, x) \leqslant \\ \mathfrak{c}(\varnothing, 0, \alpha_1) \oplus \ldots \oplus \mathfrak{c}\Big(\overline{(\alpha_i)_{i=1}^{k-1}}, \alpha_{k-1}, \alpha_k\Big) \oplus \mathfrak{c}\Big(\overline{(\alpha_i)_{i=1}^{k}}\alpha_k, x\Big) = v(I, x). \end{aligned}$$

То есть, $\mathrm{BF}(I, x) \leqslant v(I, x)$.

Часть II. $\mathrm{BF}(I, x) \geqslant v(I, x)$. Фиксируем город $z \in \mathbf{I}[I]$, доставляющий минимум $\mathrm{BF}(I, x)$ и маршрут $\beta \in \mathbb{A}_{I \setminus \{z\}}$, оптимальный для состояния $(I \setminus \{z\}, z)$. Рассмотрим маршрут $\hat{\beta} \triangleq (\beta_1, \ldots, \beta_{k-1}, z)$. Поскольку $z \in \mathbf{I}[I]$ и $(\beta_1, \ldots, \beta_{k-1}) \in \mathbb{A}_{I \setminus \{z\}}$ по определению, имеем $\hat{\beta} \in \mathbb{A}_I$.

Нет гарантии, что маршрут $\hat{\beta}$ оптимален для (I, x), следовательно,

$$\begin{aligned} v(I, x) \leqslant \mathfrak{c}(\varnothing, 0, \beta_1) \oplus \ldots \oplus \mathfrak{c}(I \setminus \{\beta_{k-1}, z\}, \beta_{k-2}, \beta_{k-1}) \\ \oplus \mathfrak{c}(I \setminus \{z\}, \beta_{k-1}, z) \oplus \mathfrak{c}(I, z, x). \end{aligned}$$

В то же время, первые k элементов правой части выражения представляют собой $v(I \setminus \{z\}, z)$ по выбору маршрута β и, ввиду левоассоциативности $\oplus$ (1.2.3), мы можем заменить их на $v(I \setminus \{z\}, z)$; наконец, учитывая что z доставляет минимум $\mathrm{BF}(I, x)$ по выбору, получаем следующую цепочку выражений: $v(I, x) \leqslant v(I \setminus \{z\}, z) \oplus \mathfrak{c}(I, z, x) = \mathrm{BF}(I, x)$, что завершает доказательство $\mathrm{BF}(I, x) \geqslant v(I, x)$ и всей теоремы. □

Восстановление оптимального маршрута

По завершении расчета (1.3.2), получаем глобальный экстремум — цену полной задачи $(1\,..\,n, \mathbb{t})$. Для *восстановления* решения, доставляющего этот экстремум, проводится поиск по пространству состояний сверху вниз: сначала отыскивается город α_n, доставляющий минимум $\mathrm{BF}(1\,..\,n, \mathbb{t})$ (исходя из (1.3.2)). Затем, в качестве α_{n-1} выбирается город, доставляющий минимум $\mathrm{BF}(1\,..\,n \setminus \{\alpha_n\}, \alpha_n)$, и так далее. Этот подход был впервые предложен в [16].

Теорема 1.3 гарантирует успешность такого поиска; очевидно, что поиск занимает меньше времени чем полная процедура генерации состояний и расчета функции Беллмана, следовательно, он не оказывает качественного влияния на порядок сложности алгоритма. В экспериментах (см., напр., [62, 77]) поиск занимал пренебрежимо малое время (не более 1–2 секунд).

Без поиска с восстановлением решения можно обойтись, сохраняя каждый раз при первичном расчете BF для каждого состояния город, доставивший экстремум; такое решение предлагается, например, в [52, 57]. Однако, в TSP-подобных задачах, основным ограничительным фактором для ДП является не временная, а пространственная сложность (см. раздел о сложности 1.4); таким образом, сохранение дополнительной информации в памяти может, наоборот, уменьшить наибольшую размерность разрешимых задач.

Отметим, что оптимальных маршрутов может быть более одного; чтобы выявить несколько оптимальных маршрутов, достаточно «разветвить» восстановление оптимального маршрута на каждом шаге α_k где минимум в BF доставляют несколько городов. Однако, чтобы гарантировать, что таким образом будут перечислены *все* оптимальные маршруты, необходимо, чтобы $\oplus$ была *строго* монотонной по первому аргументу; в противном случае, некоторые (но не все) глобально оптимальные маршруты могут быть неоптимальны *локально*.

Псевдокод точного ДП см. в листинге алгоритма 1.

Алгоритм 1 Псевдокод ДП для (PSD-)TSP-PC

Инициализация: $\mathcal{I}_0 = \{\varnothing\}$, $\mathcal{S}_0 = \{(\varnothing, x), x \in \mathbf{E}[\varnothing]\}$,
$\forall (\varnothing, x) \in \mathcal{S}_0\ v(\varnothing, x) = \mathfrak{c}(\varnothing, 0, x)$, $\mathcal{I}_1 = \mathbf{E}[\varnothing]$.
for all $k \in 1\,..\,n-1$ **do**
    for all $I \in \mathcal{I}_k$ **do**
        `Вычислить` $\mathbf{E}[I]$
        for all $x \in \mathbf{E}[I]$ **do**
            `Добавить` $I \cup \{x\}$ `в` $\mathcal{I}_{k+1}$
            `Добавить` (I, x) `в` $\mathcal{S}_k$
            `Вычислить` $v(I, x)$
`Вычислить` $v(1\,..\,n, \mathbb{t})$
`Восстановить решение по` v.

1.3.2 Усеченное динамическое программирование

Усеченное ДП (restricted dynamic programming) — основанная на ДП эвристика, впервые предложенная в [78] для решения TD-TSP (TSP с зависимостью функции стоимости от времени), затем использована в качестве «эвристического комплекса» (heuristic framework), дополненного теорией доминирования состояний (dominance analysis) для задач семейства CVRP [126]. Также она применялась автором диссертации к обобщенной задаче коммивояжера на узкие места с зависимостью от списка заданий (PSD-BGTSP-PC) [62], подробнее об этом см. в главе 2. Ниже будет обсуждаться применение этой эвристики к экземплярам TSP-PC из TSPLIB, обычным и с зависимостью от времени по [74] (TD-TSP-PC, также TD-SOP).

Эта эвристика в некотором смысле обобщает жадный алгоритм: вместо того, чтобы сохранять, рассчитав BF, весь очередной слой $\mathcal{S}_i$, в памяти вычислительного устройства сохраняются только $H \in \mathbb{N}$ *наилучших состояний*; при росте H эвристика приближается к *точному* ДП, достигая его по превышению H мощности наиболее населенного слоя $\mathcal{S}_k$. При $H = 1$, эта эвристика практически тождественна жадному алгоритму. Псевдокод эвристики указан в листинге алгоритма 2.

Ниже перечислены наиболее важные качества эвристики усеченного ДП:

- обеспечивает соблюдение ограничений
- временная и пространственная сложности *полиномиально* зависят от H и n
- достигает *точного* решения на предельном значении параметра H — когда он превосходит мощность наиболее населенного слоя пространства состояний
- легко реализуется на основе программного кода точного ДП

Алгоритм 2 Псевдокод *усеченного* ДП для (PSD-)TSP-PC. H — параметр эвристики, количество *сохраняемых* состояний

Инициализация: $\widetilde{\mathcal{I}}_0 = \{\varnothing\}$, $\widetilde{\mathcal{S}}_0 = \{(\varnothing, x), x \in \mathbf{E}[\varnothing]\}$,
$\forall (\varnothing, x) \in \widetilde{\mathcal{S}}_0\ \widetilde{v}(\varnothing, x) = \mathfrak{c}(\varnothing, 0, x)$, $\widetilde{\mathcal{I}}_1 = \mathbf{E}[\varnothing]$.
for all $k \in 1\,..\,n-1$ **do**
    for all $I \in \widetilde{\mathcal{I}}_k$ **do** ▷ для всех *сохраненных* на шаге k
        Вычислить $\mathbf{E}[I]$
        for all $x \in \mathbf{E}[I]$ **do**
            Вычислить $\widetilde{v}(I, x)$ ▷ $\widetilde{v}(I, x) \geqslant v(I, x)$, часть покрываемых могла *не сохраниться*
            if $|\widetilde{\mathcal{S}}_k| < H$ **then**
                Добавить (I, x) в $\widetilde{\mathcal{S}}_k$
            else
                Пусть $(J, y) = \operatorname{argmax}_{s \in \widetilde{\mathcal{S}}_k} \widetilde{v}(s)$
                if $\widetilde{v}(J, y) > \widetilde{v}(I, x)$ **then**
                    Удалить (J, y) из $\widetilde{\mathcal{S}}_k$
                    Добавить (I, x) в $\widetilde{\mathcal{S}}_k$
    for all $(I, x) \in \widetilde{\mathcal{S}}_k$ **do**
        Добавить $I \cup \{x\}$ в $\widetilde{\mathcal{I}}_{k+1}$
Вычислить $\widetilde{v}(1\,..\,n, \mathbb{t})$
Восстановить решение по $\widetilde{v}$.

1.4 Пространственная и временная сложность динамического программирования в PSD-TSP-PC

В настоящем разделе рассматривается сложность решения задач, подобных TSP-PC, точным ДП и эвристикой усеченного ДП; поскольку эти результаты не зависят от конкретной разновидности критерия агрегирования затрат и функции сложности, они применимы как непосредственно к TSP-PC, так и к ее обобщениям, включая те варианты зависимости функций стоимости от времени или от списка заданий, которые не изменяют структуры состояний, например, задачи (PSD-TSP-PC).

1.4.1 Опорные предположения для оценок сложности

В расчетах, касающихся временной сложности, мы предполагаем, что *теоретико-множественные* операции $\cup$, $\cap$ и $\setminus$ выполняются за *постоянное время* $\mathcal{O}(1)$; данное предположение верно в случае когда общее число городов n не превосходит разрядности процессора l (обычно $l = 64$ бита); в противном случае оно пропорционально $\lceil \frac{n}{l} \rceil$, в предположении программной реализации множеств поверх (массивов) целых чисел без знаков, где вышеуказанные теоретико-множественные операции представлены, соответственно, побитовыми AND, OR и OR затем XOR.

Совершая более сильное допущение, мы также предполагаем, что вычисление $v(I \setminus \{e\}, e) \oplus \mathfrak{c}(I, e, x)$, «тела» (BF), также занимает постоянное время; обозначим это время t_{BF}. Настоящее допущение может нарушаться когда функции стоимости перемещений $\mathfrak{c}$ не рассчитаны заранее; также t_{BF} зависит от того, используется ли целочисленная арифметика или арифметика с плавающей запятой; специфика операции агрегирования $\oplus$ также может влиять на это время.

Стоить заметить, что в прикладных задачах могут встречаться достаточно сложные функции стоимости перемещений (требующих задействования арифметики с плавающей запятой); например, в задачах маршрутизации инструмента в машинах листовой резки с ЧПУ [85] или минимизации облучения сотрудников АЭС [26]. Кроме того, t_{BF} может зависеть и от структуры данных, используемой для хранения значений функции Беллмана.

Как видно, например, из таблиц 1.2,1.5, зависимость от времени типа «traveling deliveryman» (см. раздел 1.5.4) где, в сравнении с обычной TSP-PC, требуется дополнительное (целочисленное) умножение, рост t_{BF} составляет не более 10%. В одном из наиболее наших длительных вычислений, закрывшем задачу `ry48p.3.sop`, вариант TD-SOP решался на 7% *дольше* чем обычная TSP-PC (02:18:51 к 02:10:05, ЧЧ:ММ:СС). Тем не менее, для большей ясности итогового выражения, мы выпустим t_{BF} из оценок сложности.

1.4.2 Точное динамическое программирование

Начнем с пространственной сложности — количества состояний ДП.

Предложение 1.1. *$|\mathcal{S}| \leqslant w|\mathcal{I}|$, где w — ширина P (мощность наибольшей по включению антицепи в P). Пространственная сложность* полиномиально *(по количеству городов n) зависит от количества идеалов $|\mathcal{I}|$.*

Доказательство. Состояния ДП (1.3.1) представляют собой «идеалы с выходом» (терминология [125]); последние связаны с идеалом оператором продолжения $\mathbf{E}$ (1.1.4), который выбирает *минимальные* элементы дополнения идеала; очевидно, что множество минимальных элементов образует *антицепь*, чья мощность, по определению, мажорируется w. Полиномиальность зависимости $|\mathcal{S}|$ от $|\mathcal{I}|$ очевидна из $w \leqslant n$. □

В общем случае *трудно* перечислить все идеалы, потому что существует («полиномиальной сложности») биекция между множеством идеалов и антицепей: каждая антицепь A однозначно определяет свой *главный идеал*

$$\downarrow A = \left\{ i \in 1\,..\,n \;\middle|\; \exists a \in A\colon i \leqslant_P a \right\};$$

в обратную сторону соответствие обеспечивается оператором множества наибольших элементов Max (1.1.1), в свою очередь, известно, что задача о перечислении всех антицепей #P-трудна[7] [106], откуда потребность в вычислительно простых оценках (см. [22,23]). Тем не менее, для многих важных классов порядков, подсчитать количество идеалов сравнительно легко [22, 23], [122, Appendix A.2.2].

Например, условия предшествования, возникающие в задачах маршрутизации инструмента в машинах листовой резки (см., напр., [85]), образуют лес (входящая степень вершины не более 1), количество идеалов в котором можно подсчитать за время кратное не более чем квадрату количества вершин [100].

Предложение 1.1 демонстрирует *экономичность* методов ДП, напрямую учитывающих *условия предшествования* по сравнению с попыткой использовать «обычное» ДП для TSP (без условий предшествования) [15, 16] для решения подобных задач. Например, следуя [16], состояния можно определить[8] как

$$\mathcal{S}^\sharp \triangleq \left\{ (K, x) \in \mathcal{P}(1\,..\,n) \setminus \{1\,..\,n\} \times 1\,..\,n \;\middle|\; x \in 1\,..\,n \setminus K \right\} \cup \left\{ \left(1\,..\,n, \Bbbt\right) \right\};$$

количество состояний $|\mathcal{S}^\sharp|$, по [16], оценивается[9] как $(n)2^{n-1}$, тогда как для TSP-PC при определении состояний по (1.3.1) имеем оценку $|\mathcal{S}| \leqslant w|\mathcal{I}|$ (предложение 1.1).

Очевидно, $w \leqslant n$, причем равенство достигается лишь в тривиальном случае *отсутствия* условий предшествования. В свою очередь, для «минимального» случая условий предшествования, выраженного одной *адресной парой* $P = (\{a <_P b\}, 1\,..\,n)$, где $a, b \in 1\,..\,n$ — некоторая пара городов, $|\mathcal{I}| = 3 \cdot 2^n$ [100] и $w = n - 1$. Отметим, что в этом случае оценка $|\mathcal{S}| < w|\mathcal{I}|$ уже *показывает* «экономию» в состояниях:

$$|\mathcal{S}| \leqslant w|\mathcal{I}| = (n-1)3 \cdot 2^{n-2} \leqslant \frac{n}{2} \cdot 2^{n-2} = |\mathcal{S}^\sharp|,$$

несмотря на то что она менее точна чем оценка [16] для случая без условий предшествования. Напомним, последняя выводится из

$$\sum_{k=1}^{n} k \cdot C_n^k = n \cdot 2^{n-1},$$

где C_n^k — количество сочетаний из n элементов по k (биномиальный коэффициент), выражающий количество подмножеств $1\,..\,n$ мощности k. В случае ДП для TSP-PC приходится просто домножать w на количество идеалов (аналогичная оценка для TSP без условий предшествования была бы $(n+1) \cdot 2^n$), поскольку для *количества идеалов* не известно подобного выражения; лучший (единственный известный автору диссертации) алгоритм *перечисления* всех идеалов заданной мощности предложен в [127] и имеет, для $\mathcal{I}_k$, сложность $O(|\mathcal{I}_k|n^3)$ — более высокую чем в алгоритмах, перечисляющих любой мощности, *все*

[7] в русском издании [19] этот класс обозначался KP.

[8] ниже мы приводим определение, сходное по структуре с (1.3.1); на самом деле, в [16] «интерфейс» *содержался* в списке заданий; нетрудно проверить, что на количество состояний эта перестановка не влияет

[9] здесь мы вновь отклоняемся от прямого цитирования [16]: там фиксированной точкой старта был город под номером 1, а не особая база 0, так что для соответствия нужно увеличить количество городов на единицу; кроме того, единственный элемент $\mathcal{S}_n^\sharp$ не фиксировался как состояние

идеалы $\mathcal{I}$ (см. [122, Appendix A.2.2]); см. также замечание об обобщенных биномиальных коэффициентах [122, pp. 126–127].

Для обсуждения *временной* сложности удобно ввести две леммы, связанные с процедурой генерации идеалов. Введем также особое обозначение для множества элементов, *предшествующих* данному: $\forall x \in 1\,..\,n$

$$\Downarrow x \triangleq \{m \in 1\,..\,n \mid m <_P x\};$$

фактически, это ни что иное как главный идеал [122, Section 1.4.2] синглетона, $\downarrow\{x\}$, откуда исключен сам элемент, $\Downarrow x = \downarrow\{x\} \setminus \{x\}$; то же самое представляют собой строки (или столбцы) матрицы смежности представления отношения порядка в виде ориентированного графа.

Лемма 1.1. *Для* $K \in \mathcal{P}(1\,..\,n)$, $|K| = k$,

$$\mathrm{Max}[K] = K \setminus \bigcup_{x \in K} \Downarrow x.$$

Если $\Downarrow k$ *рассчитаны заранее для всех* $k \in 1\,..\,n$, *то временная сложность оператора* $\mathrm{Max}[\cdot]$ *составляет* $\mathcal{O}(k)$.

Доказательство. Доказательство самоочевидно: *максимальный* элемент $m \in \mathrm{Max}[K]$ никак не может *предшествовать* какому бы то ни было $x \in K$. Что касается сложности, множество максимальных элементов получается последовательным удалением из K множеств $\Downarrow x$ для каждого $x \in K$, которых по условию k штук. Сложность вычитания на множествах предполагается постоянной[10]. □

Аналогично доказывается лемма о сложности $\mathrm{Min}[K]$.

Лемма 1.2. *Для* $K \in \mathcal{P}(1\,..\,n)$, $|K| = k$,

$$\mathrm{Min}[K] = K \setminus \bigcup_{x \in K} \Uparrow x,$$

где $\Uparrow x \triangleq \{m \in 1\,..\,n \mid m >_P x\}$. *Если* $\Uparrow k$ *рассчитаны заранее для всех* $k \in 1\,..\,n$, *то временная сложность оператора* $\mathrm{Min}[\cdot]$ *составляет* $\mathcal{O}(k)$.

Докажем теперь лемму о (временной) сложности генерации идеалов через оператор $\mathbf{E}$ с учетом характеризации по теореме 1.1.

Лемма 1.3. *Временная сложность порождения всех идеалов* $\mathcal{I}$ *по процедуре* $\mathcal{I}_{k+1} = \{I \cup \{m\} \mid (I \in \mathcal{I}_k) \wedge (m \in \mathbf{E}[I])\}$ *составляет* $\mathcal{O}(nw)$ *на каждый идеал (итого,* $\mathcal{O}(|\mathcal{I}| \cdot nw)$*); данные результат улучшает известную оценку* $\mathcal{O}(n^2)$ *на идеал (см. напр. "The Lawler Approach" в [128]).*

Доказательство. Вышеуказанная процедура состоит в применении $\mathbf{E}$ к каждому идеалу, начиная с $\varnothing \in \mathcal{I}_0$. Теорема 1.1 характеризует $\mathbf{E}[I]$ как $\mathrm{Min}[1\,..\,n \setminus I]$, сложность определения которого ввиду леммы 1.2 не превышает $\mathcal{O}(n)$. Каждый идеал $I \in \mathcal{I} \setminus \mathcal{I}_0$ может генерируется не один раз, а несколько — из каждого $\check{I} \in \mathcal{I}_{|I|-1}$ такого что $\exists y \in \mathbf{I}[I] \mid \check{I} = I \setminus \{y\}$, число которых ограничено w так как $\mathbf{I}[I] = \mathrm{Max}[I]$, а множество максимальных элементов есть *антицепь* по определению (1.1.1). Таким образом, сложность не превышает $\mathcal{O}(nw)$ на идеал. □

[10] см. общие предположения об операциях постоянной сложности в начале раздела 1.4

Предложение 1.2. *Временная сложность процедуры динамического программирования для (PSD-)TSP-PC составляет*

$$\mathcal{O}(|\mathcal{I}| \cdot nw);$$

кроме того, временная сложность полиномиально *зависит от пространственной.*

Доказательство. **Все состояния можно сгенерировать за время** $\mathcal{O}(|\mathcal{I}| \cdot nw)$: для каждого идеала $I \in \mathcal{I}$ достаточно один раз вызвать $\mathbf{E}[I]$ чтобы и «продолжить» его и выписать все соответствующие ему состояния. **Вычисление** $v[\mathcal{S}]$ **займет** $\mathcal{O}(|\mathcal{I}| \cdot nw)$: по предложению 1.1, каждому идеалу I соответствует *не более* w состояний; для каждого такого состояния $(I, x) \in \mathcal{S}_{|I|}$ в теле (BF) берется минимум по покрытым состояниям $\{(I \setminus \{m\}, m) \in \mathcal{S}_{|I|-1} \mid m \in \mathbf{I}[I]\}$, которых, согласно характеризации $\mathbf{I}[I] = \mathrm{Max}[I]$ (теорема 1.2), не более w, так как максимальные элементы образуют антицепь по определению $\mathrm{Max}[I]$ (1.1.1); также нужно учитывать стоимость расчета $\mathbf{I}[I]$, с помощью которого *определяются* покрываемые состояния ($\mathcal{O}(n)$), откуда получаем сложность расчета не более $(w \cdot w + n) \leqslant (2nw) = \mathcal{O}(nw)$ на идеал. Наконец, если элементы, доставляющие минимум в (BF), не заносятся в память, в худшем случае, **восстановление решения** потребует (повторно) вычислить $v[\mathcal{S}]$. Таким образом, получаем *трижды* $\mathcal{O}(|\mathcal{I}| \cdot nw)$ операций, что и требовалось доказать. □

Наиболее важные выводы из предложений 1.1 и 1.2 заслуживают представления виде самостоятельных следствий:

Следствие 1. *Временная сложность* полиномиально *зависит от пространственной — числа состояний* $|\mathcal{S}|$.

Следствие 2. Пространственная *сложность* полиномиально *зависит от количества идеалов* $|\mathcal{I}|$.

Следствие 3. *В случае* полиномиальной *зависимости количества идеалов* $|\mathcal{I}|$ *от количества городов* n*, точное динамическое программирование имеет* полиномиальную *временную сложность.*

Очевидно, что временная сложность *полиномиально* зависит от числа состояний; важное следствие этого факта — *полиномиальная разрешимость* ДП для TSP-PC в случае *полиномиальной* зависимости количества идеалов от общего количества городов n. Соответственно, для ДП и производных методов размерность задачи можно считать в виде числа состояний; очевидно, что число состояний в существенной мере определяется не только количеством городов но и характером частичного порядка, определяющего условия предшествования.

1.4.3 Усеченное динамическое программирование

Предложение 1.3. *Количество состояний при решении усеченным ДП не превышает* $H(n+1)$.

Доказательство. Очевидно в виду определения: в пространстве состояний $(n+1)$ слоев, на каждом не более H состояний. □

Предложение 1.4. *Временная сложность усеченного ДП не превышает*

$$\mathcal{O}\left(\left[H \cdot n \cdot \min\left\{\frac{n+1}{2}; w\right\}\right] \cdot (nw + \log_2 H)\right).$$

Доказательство. Рассмотрим каждый множитель по-отдельности:

$H \cdot n \cdot \min\left\{\frac{n+1}{2}; w\right\}$ — **общее количество *рассмотренных* состояний.** Пусть слои пространства состояний под номерами от 0 до k, $k \in 1\,..\,n-1$, уже обработаны; рассмотрим процедуру для слоя $k+1$. Есть два способа оценить (наибольшее) количество состояний, которые будут порождены H состояниями, оставленными на слое k предыдущим шагом. Начнем с варианта, не учитывающего *условия предшествования*: поскольку все списки заданий из $\mathcal{S}_k$ имеют мощность k, к ним могут быть добавлены не более $n-k$ городов, следовательно, на слое 0 может обрабатываться $H \cdot n$ состояний, на слое $1 - H \cdot (n-1)$ и так далее, итого[11] $H\frac{n(n+1)}{2}$ штук (сумма арифметической прогрессии). Другой способ основан на учете ширины w условий предшествования. Каждое состояние может покрываться не более чем w другими, при этом, каждый раз, на «предыдущем» слое находится не более H состояний, что приводит к суммарной оценке $n \cdot wH$ для n слоев под номерами от 1 до $n-1$. Эта оценка становиться *меньше* предыдущей когда $w < \frac{n+1}{2}$ (что выполняется для 18 из 41 задач в TSPLIB, смотрите ширину этих задач в [44, Table 1]). Общий положительный множитель n можно вынести из минимума.

$\mathcal{O}(nw)$ — **нужно на генерацию и расчет экстремума состояний.** Рассуждения аналогичны 1.2; отличие состоит в том, что приходится полагать, что $\mathcal{O}(nw)$ уходит на генерацию каждого *состояния*, а не идеала–списка заданий.

$\mathcal{O}(\log_2 H)$ **расходуется на «сортировку» каждого состояния.** В процессе обработки каждого слоя в памяти вычислительного устройства храняться лишь H наилучших (с наименьшим экстремумом) состояний. Сгенерировав новое состояние, нужно решить, оставлять ли его: если на текущий момент времени на текущем слое менее H состояний, новое сохраняется; если же их уже H штук, новое будет сохранено только если оно лучше чем *наихудшее* из текущих, которое будет отброшено. Для этого не требуется пересортировывать все состояния, достаточно лишь хранить их в *очереди с приоритетами*, где наивысший приоритет имеет *наихудшее* состояние — чтобы легче было его извлекать. Для широко известной и часто реализуемой очереди с приоритетами на *бинарной куче* (binary heap), время доступа к приоритетному элементу *постоянно*, а время добавления и удаления одного элемента — логарифм размера очереди (т. е. H) [129, Section 6.1][12]. □

1.4.4 Оценки количества идеалов

Статьи [22, 23] по сей день остаются лучшим справочным материалом в вопросах оценки количества идеалов (о результатах, полученных после 1993 см., напр., [122, Appendix A.2.2]); приведем наиболее общие (в смысле классов частичных порядков) аналитические[13] нижние и верхние оценки количества идеалов:

Количество идеалов можно оценить снизу выражением $2^w + n - w$ [23, p. 286].

Верхняя оценка, получаемая обобщением по неравенству Коши–Буняковского верхней оценки из разложения $P = (1\,..\,n, <_P)$ в цепи (где, по теореме Дилуорса [131], не менее w цепей), составляет $\left(\frac{n+w}{w}\right)^w$ [23, p. 286].

Оценка через неравенство Коши–Буняковского, несомненно, более грубая чем «хорошие» оценки по разложению в цепи — «плохим» разложением будет, например, тривиальное разложение в *одноэлементные* цепи, дающее тривиальную же оценку 2^n — однако,

[11] опустим из этих расчетов единственное состояние на слое n, $(1\,..\,n, \mathbb{t})$; формально, оно обрабатывается за $(nw)+w$: на генерацию уходит (nw) операций, затем рассчитывается BF на покрываемых состояниях, которых не более w; поскольку это единственное состояние на слое, сортировка не требуется. Таким образом, учет данного состояния не изменяет *порядка* зависимости сложности от n, w или H.

[12] в частности, стандарт C++ standard [130] требует обеспечения *логарифмической* сложности операций `push` и `pop`, см. [130, Sections 23.6.4.3, 25.4.6]

[13] в [22, 23] также проводилось статистическое исследование

какое разложение следует предпочесть? Поставим проблему об *оптимальном разложении в цепи*:

Проблема 1. По данному частичному порядку $P = (1\,..\,n, <_P)$, найти его разложение в цепи

$$C^* = \bigsqcup_{i=1}^{k} C_i^*,$$

доставляющее минимум $\prod_{i=1}^{k} |C_i^* + 1|$, где $k \leqslant n$, а $\bigsqcup$ обозначает *дизъюнктное* объединение.

Отметим, что настоящая проблема разрешена для порядков *размерности* 2 в [132], см. также обсуждение этого вопроса в [23]; однако тот метод основывается на характеристическом свойстве порядков размерности 2 и в общем случае не применим.

1.4.5 Расчет теоретико-порядковых характеристик

С целью изучить «индивидуальную» (пространственную) сложность экземпляров TSP-PC и других задач с условиями предшествования автор диссертации написал на Haskell программу, позволяющую определять некоторые, в широком смысле, *характеристики* частичных порядков, задающих условия предшествования: транзитивную редукцию, транзитивное замыкание, ширину и основанные на ней оценки сверху и снизу количества состояний ДП (для TSP-PC); результаты этого исследования были представлены на Четвертом русско-финском симпозиуме по дискретной математике и опубликованы в трудах этой конференции [44].

Бинарные отношения. Транзитивность и разновидности формализации условий предшествования

Остановимся подробнее на транзитивной редукции, транзитивном замыкании и потребности в них ввиду обсуждения различных формализаций условий предшествования.

Пусть $R \in 1\,..\,n \times 1\,..\,n$ — некоторое бинарное отношение на множестве городов, предназначенное представлять условия предшествования: если $(a, b) \in R$, то в любом маршруте город a *предшествует* городу b. Все упомянутые в настоящем подразделе отношения рассматриваются на *конечном* множестве $1\,..\,n \times 1\,..\,n$.

Отношение R называется *транзитивным*[14], если

$$\forall a, b, c \in 1\,..\,n \ \big((a, b) \in R \wedge (b, c) \in R\big) \to \big((a, c) \in R\big).$$

Транзитивным замыканием[15] отношения R называется наименьшее по включению транзитивное отношение R^+, содержащее R как подмножество,

$$R^+ \triangleq \inf\{H \subset 1\,..\,n \times 1\,..\,n \mid R \subset H, H \text{ транзитивно}\};$$

здесь inf понимается по включению $\subset$ на $1\,..\,n \times 1\,..\,n$.

Транзитивной редукцией[16] называется наименьший (по включению) представитель R^- класса эквивалентности по *совпадению* транзитивного замыкания

$$R^- \triangleq K \subset R : \big(K^+ = R^+\big) \wedge \big(\forall Q \subset K \ (Q^+ = K^+) \to (Q = K)\big).$$

[14] см., напр., [29, Def. 3.1.1]

[15] см., напр., [29, Def. 3.2.1]

[16] с англ. *transitive reduction* [28], также «транзитивно неприводимое ядро» (transitively irreducible kernel) [29, Ch. 3], см. подробнее [29, Def. 3.2.4]

Будем говорить, в рамках настоящей работы, что отношение R *фундировано* — по аналогии с аксиомой фундирования в теории множеств[17] — если

$$\forall R_0 \in \mathcal{P}'(R)\ \exists z \in R_0 : \mathrm{pr}_1(z) \neq \mathrm{pr}_2(z')\ \forall z \in R_0. \tag{1.4.1}$$

Данное условие [25, усл. 2.2.1] в вышеуказанном виде используется в работах А. Г.Ченцова и соавторов для обеспечения *существования* допустимых в смысле условий предшествований R маршрутов, определяемых следующим образом [25, ур. (2.1.5)]: маршрут $\alpha \in (\mathrm{bi})[1\,..\,n]$ *допустим*, если

$$\alpha_a^{-1} < \alpha_b^{-1}\ \forall (a,b) \in R.$$

В частности, свойство (1.4.1) не допускает наличия «рефлексивной компоненты» в отношении — пар вида (a,a), где $a \in 1\,..\,n$. Отметим без доказательства, что свойство (1.4.1) не отличается от свойства *ацикличности* (отсутствия циклов); на языке бинарных отношений, последнее можно сформулировать как «транзитивное замыкание не содержит рефлексивной компоненты», см. [29, Proposition 6.1.4]. Полное и общее доказательство корректности использования фундированных бинарных отношений для представления условий предшествования — в смысле существования допустимых маршрутов — доказана, например, в [25, предл. 2.2.2,2.2.3, теор. 2.2.1].

Следует также упомянуть, что *транзитивное замыкание* любого *фундированного* отношения будет *частичным порядком* [122, Theorem 2.23].

Транзитивную редукцию мы находили методами теории отношений [29] — через *композицию* исходного отношения с его транзитивным замыканием [134, p. 121]; последнее получалось через известный алгоритм Руа–Уоршолла [29, § 3.2.8] в форме для *функциональных* языков программирования, адаптированной из [134].

Отметим, что формат TSPLIB требует указывать условия предшествования *транзитивно замкнутыми*, однако, для сравнения сложности задач TSPLIB с задачами, где условия предшествования даны фундированным отношением[18,19], нужно привести их к общему «каноническому виду» — либо транзитивному замыканию, либо транзитивной редукции; последние по данному фундированному отношению определяются однозначно. Напомним, фундированные отношения используются как формализм для условий предшествования, например, в [26,58]. Отметим, что мощность как транзитивного замыкания так и редукции может использоваться в качестве некоторой меры «ограничительности» условий предшествования (то есть, сравнительного уменьшения количества ДП-состояний).

Ширина

Ширина определялась через максимальное паросочетание во вспомогательном двудольном графе (см. короткое описание [22, p. 107] и подробное [122, § 4.5], включая доказательство), которое находилось с использованием библиотеки `Data.Graph.MaxBipartiteMatching` языка программирования Haskell.

Плотность

Также следует упомянуть о *плотности* частичных порядков. Это безразмерная характеристика, предназначенная для оценки «ограничительности» частичного порядка, опре-

[17] то же аксиома регулярности, см., напр. [119, гл. 2, § 2], [133, § I.3, Axiom 2. Foundation]

[18] т. е. не взято транзитивное замыкание; *фундированность* (1.4.1) отношения гарантирует, что его транзитивное замыкание будет частичным порядком [122, Theorem 2.23], но само отношение может быть не этим частичным порядком, а некоторым *иным* членом класса эквивалентности по совпадению транзитивного замыкания

[19] например, экземпляры BGTSP-PC в разделе 2.1.8

деляемая следующим образом:

$$d(P) = \frac{|\leqslant_P|}{n(n-1)/2},$$

иными словами, частное мощности отношения порядка и мощности линейного порядка на том же носителе. Эта метрика изначально появилась в теории расписаний [102], под названием «силы» порядка, «order strength»; там же было предложено рассматривать значения плотности 0,25, 0,50 и 0,75 как определяющие, соответственно, «слабые», «средние» и «сильные» отношения порядка. Плотность используется как мера сложности экземпляров TSP-PC в [103, 104], однако, как показано в [22, 23], существуют как *сильные* классы порядков (плотность 0,75), для которых число идеалов зависит от n *экспоненциально*, так и *слабые* (плотность не более 0,25) классы, для которых число идеалов зависит от n лишь *полиномиально*, таким образом, плотность не всегда отражает сложность экземпляра задачи в контексте ДП. Возможно, следует рассмотреть другие характеристики частичных порядков [122, p. 35]; в частности, была предложена альтернативная плотности мера «близости» частичного порядка к линейному — «линейная невязка» (linear discrepancy) [135].

В Таблице 1.1 представлены расчитанные характеристики экземпляров TSP-PC из TSPLIB. Экземпляры, для которых не известно оптимального решения, отмечены **жирным** шрифтом. Экземпляры, для которых нами получены [77] точные ДП-решения помечены *курсивом* (см. подробнее далее). Представлены следующие характеристики:

- n количество «собственных» городов (исключая фиксированные *базу* и *терминал*)
- «RED.» мощность транзитивной редукции
- «CL.» мощность отношения порядка («транзитивного замыкания»)
- «LOG_S-LB» двоичный логарифм «нижней»[20] оценки числа состояний $w(2^w - n + w)$
- «LOG_S-UB» двоичный логарифм верхней оценки числа состояний $w\left(\frac{n+w}{w}\right)^w$.

Характеристики LOG_S-LB и LOG_S-UB представляют практическое средство оценки ДП-*разрешимости* данного экземпляра TSP-PC исходя из имеющейся оперативной памяти (предполагается, что доступная память исчисляется *десятками* ГБ): в 1 гигабайте 2^{30} байт; если нижняя оценка превосходит 40 («1 терабайт»), вероятно, не стоит пытаться решать данную задачу. Вместе с тем, если *верхняя* оценка составляет порядка 40, вероятно, что имеющейся памяти не хватит для точного ДП-решения.

Программа для расчета вышеуказанных характеристик написана автором на Haskell. Множества представлялись сортированными списками встроенного типа `Int`. Время счета не замерялось; тем не менее, отметим, что, за исключением самых больших экземпляров (порядка 300 городов), результат получался практически мгновенно. Двоичные логарифмы считались в пакете электронных таблиц Microsoft Excel, затем округлялись вверх до одного десятичного знака.

[20] строго говоря, это не нижняя оценка, поскольку некоторым идеалам может соответствовать менее w состояний; тем не менее, мы полагаем ее более иллюстративной чем формальную нижнюю оценку по числу идеалов (по одному идеалу на каждое состояние), которая превращается в точную оценку для случая *линейного* порядка

Таблица 1.1: Характеристики экземпляров TSP-PC из TSPLIB

НАЗВАНИЕ	n	RED.	CL.	DENSITY	WIDTH	LOG_S-LB	LOG_S-UB
br17.10.sop	16	10	15	0,12	10	13,4	17,2
br17.12.sop	16	12	22	0,18	9	12,2	16,5
ESC07.sop	7	6	7	0,33	5	7,5	8,7
ESC11.sop	11	3	5	0,09	9	12,2	13,6
ESC12.sop	12	7	11	0,17	10	13,4	14,7
ESC25.sop	25	9	11	0,04	19	23,3	27,3
ESC47.sop	47	10	32	0,03	41	46,4	50,6
ESC63.sop	63	95	233	0,12	53	58,8	65,7
ESC78.sop	78	77	283	0,09	32	37,1	62,1
ft53.1.sop	52	12	12	0,01	42	47,4	54,3
ft53.2.sop	52	25	30	0,02	34	39,1	50,7
ft53.3.sop	52	48	217	0,16	24	28,6	44,5
ft53.4.sop	52	63	759	0,57	13	16,8	33,9
ft70.1.sop	69	17	17	0,01	55	60,8	70,3
ft70.2.sop	69	35	48	0,02	44	49,5	65,4
ft70.3.sop	69	68	215	0,09	35	40,2	60,2
ft70.4.sop	69	86	1 325	0,56	16	20,1	42,6
kro124p.1.sop	99	25	33	0,01	78	84,3	98,5
kro124p.2.sop	99	49	68	0,01	65	71,1	92,9
kro124p.3.sop	99	97	266	0,05	43	48,5	79,6
kro124p.4.sop	99	131	2 305	0,48	22	26,5	58,6
p43.1.sop	42	9	11	0,01	36	41,2	45,4
p43.2.sop	42	20	34	0,04	26	30,8	40,8
p43.3.sop	42	37	96	0,11	21	25,4	37,7
p43.4.sop	42	50	496	0,58	13	16,8	30,8
prob.100.sop	98	41	41	0,01	57	62,9	88,1
prob42.sop	40	10	19	0,02	34	39,1	43,3
rbg048a.sop	48	192	447	0,40	32	37,1	47,4
rbg050c.sop	50	256	508	0,41	31	36	48
rbg109a.sop	109	622	5 329	0,91	12	15,7	43,6
rbg150a.sop	150	952	10 334	0,92	13	16,8	51,2
rbg174b.sop	174	1 113	13 955	0,93	22	26,5	73,9
rbg253a.sop	253	1 721	30 181	0,95	22	26,5	84,7
rbg323a.sop	323	2 412	48 202	0,93	47	52,6	145,5
rbg341a.sop	341	2 542	54 303	0,94	33	38,1	120,7
rbg358a.sop	358	3 239	56 536	0,88	55	60,8	165,8
rbg378a.sop	378	3 069	63 585	0,89	55	60,8	169,6
ry48p.1.sop	47	11	12	0,01	37	42,3	49
ry48p.2.sop	47	23	26	0,02	29	33,9	45,2
ry48p.3.sop	47	42	132	0,12	19	23,3	38,4
ry48p.4.sop	47	58	596	0,55	12	15,6	31,2

1.5 Вычислительный эксперимент. Задачи из TSPLIB

1.5.1 Программный комплекс, реализующий точное и усеченное динамическое программирование

Комплекс реализован в виде модульной программы на C++11, обеспечивающей точное и эвристическое решения обычной и обобщенной задачи коммивояжера с условиями предшествования с различными вариантами агрегирования затрат, в частности, арифметическая сумма реализует TSP-PC и ее обобщенные варианты, max реализует BTSP-PC и ее обобщенные варианты; также доступен вариант TD-TSP-PC с зависимостью типа *traveling deliveryman* [74]. Комплекс может применяться в 64-битных средах под управлением операционных систем семейств Windows (использовался компилятор Microsoft Visual C++ 12.0) и Linux (использовался компилятор gcc 6.3.0); совместимость между различными платформами достигается условной компиляцией с помощью директив препроцессора C. Ниже представлено его краткое описание, подробнее см. главу 4.

Универсальность по разновидностям решаемых задач достигается за счет разделения «геометрической» части задачи (расстояние между городами, условия предшествования) и признаков специфической задачи (конкретная разновидность функции агрегирования затрат). Также, благодаря конструкциям ДП из [44, 77], обеспечена «прозрачность по направлению решения»: за исключением начальных условий и определений операторов **E** и **I**, решение как в прямом так и в попятном направлении производит общая кодовая база, что упрощает обслуживание программного кода; если для точного ДП, за редким исключением, направление значения не имеет, то для усеченного ДП, в зависимости от конкретного экземпляра задачи, одно из направлений может оказаться более предпочтительным (см. ниже, в особенности, раздел 1.5.4). Использование элементов объектно-ориентированного подхода также позволяет частично объединить кодовую базы точного и усеченного ДП: вычисление значений функции Беллмана и восстановление оптимального маршрута по ее значением унифицированы.

В процессе решения задач, комплекс замеряет истекшее (физическое) время с точностью до 1 мс с помощью метода `steady_clock::now()` из библиотеки C++ `<std::chrono>` и расход оперативной памяти с помощью интерфейса C++ к ОС. При проведении вычислительного эксперимента, для пакетного запуска комплекса на различных исходных данных и последующего сбора выходных данных использовались стандартная пользовательская оболочка Unix-подобных операционных систем (/bin/sh) и стандартные утилиты (echo, grep, cut и др.).

1.5.2 Введение к вычислительном экспериментам

Здесь и далее, указывая расход машинной памяти, мы используем десятичные префиксы (КБ, МБ, ГБ) для обозначения двоичных порядков (1КБ = 1024 Б, 1МБ = 2^{20} Б и так далее). Расход памяти замерялся средствами интерфейса C++ к функционалу операционной системы по отслеживанию расхода оперативной памяти[21].

Мы запускали точное ДП и эвристику усеченного ДП на экземплярах TSP-PC из TSPLIB [76], как стандартных, так и с зависимостью функции стоимости от времени по типу «traveling deliveryman», как предложено в [74]; измерялся расход оперативной памяти и время выполнения[22]. Как точное, так и усеченное ДП программировались на C++; подход в целом совпадал с изложенным в [62], включая структуры данных; основные отличия заключалось в том, что

[21] Linux: VmRSS

[22] замерялось «физическое» (wall clock time), а не «процессорное» (CPU time) время

- все стоимости считались в целочисленной арифметике (мы использовали 64-битные целые без знака, `std::uint64_t`), как предполагается форматом TSPLIB для TSP-PC, вместо рассмотренной в [62] арифметики с плавающей запятой с числами типа `float`;

- подмножества $1..n$ кодировались не целыми числами без знака (`uint64_t`), а массивом бит (`std::bitset`).

Время исполнения алгоритма измерялось с *миллисекундной* точностью с помощью метода `steady_clock::now()` из библиотеки C++ `<std::chrono>`. В таблицах ниже, для *всех* экземпляров задач, время исполнения *усеченного* ДП усреднено по трем прогонам; для точного ДП усреднение по трем прогонам производилось только в случае если первый прогон завершался в течение 3 секунд. Для всех остальных экземпляров указано неусредненное время, округленное до целых секунд.

Отметим, что схема параллелизации для системы с общей памятью на основе конструкции `task` из OpenMP, в том виде, в каком она использовалась для PSD-BGTSP-PC [62] (об этой схеме и задаче см. раздел 2.1.6) не позволила улучшить время расчета в предварительных испытательных прогонах и потому не исследовалась далее. Наиболее вероятной причиной отсутствия прироста производительности мы полагаем более мелкое разбиение на задачи-`task`, при котором накладные расходы на распределение заданий доминировали над приростом производительности: в отличие от [62],

(a) исследовалась *необобщенная* задача, т. е. в одном блоке `task` оказывалось не более w вычислений, требующих t_{BF}, вместо $w \cdot m$, где m — число городов в мегаполисе

(b) вычисления производились в целых числах, а не числах с плавающей запятой, то есть, были существенно быстрее сами по себе

Усеченное ДП проверялось на всех задачах при значениях параметра $H \in \{10^k, k \in 0..5\}$, как в прямом, так и попятном виде. Эксперимент проводился на узлах суперкомпьютера «Уран» с центральным процессором Intel Xeon E5-2697 v4 и 256 ГБ оперативной памяти под управлением CentOS Linux 7 x64, использовался компилятор gcc 6.3.0.

Точное ДП применялось только к «перспективным» экземплярам задач, определяемым по критерию, указанному в разделе 1.4.5: двоичный логарифм верхней оценки числа состояний не выше 40. Также «перспективными» полагались экземпляры TSP-PC, для которых эвристика усеченного ДП при значении параметра не более чем $H = 10^5$ *достигала* известного оптимального значения. Эксперимент проводился на суперкомпьютере Уран (ИММ УрО РАН) на узлах с центральным процессором Intel Xeon E5-2697 v4 и 256 ГБ оперативной памяти под управлением CentOS Linux 7; использовался компилятор gcc 6.3.0. Отметим, что как время счета так и расход памяти для точного ДП совпали для прямого и попятного ДП; в таблицах ниже приводятся данные, полученные для *попятного* ДП.

В таблицах результатов *точного* ДП (Таблицы 1.2,1.5), названия экземпляров, для которых оптимальное решение *впервые* получено автором [77], отмечены **жирным** начертанием. В таблицах результатов *усеченного* ДП (Таблицы 1.3,1.4,1.6,1.7) приведено превышение результата эвристики при соответствующем параметре над наилучшим известным решением в процентах; нулевое значение этого превышения означает, что эвристика достигла известного оптимального значения либо наилучшей известной оценки сверху. Кроме того, в таблицах результатов *усеченного* ДП символом † отмечены экземпляры, для которых на момент написания не известно оптимального решения.

1.5.3 Задача коммивояжера с условиями предшествования / TSP-PC

В Таблице 1.2 приведены результаты решения «перспективных» экземпляров задач точным ДП. Единственным принципиальным достижением здесь является получение оптимального решения `ry48p.3.sop` [44]; однако, стоит отметить, что, учитывая данные по расходу памяти, некоторые недавно закрытые экземпляры (например `rbg150a.sop` [34], `ry48p.4` [47]) сравнительно легко разрешимы точным ДП. Трудно ожидать, что другие нерешенные экземпляры получится закрыть прямым применением ДП: `ft70.2.sop`, `kro124*.sop`, `prob.100.sop`, `rbg378a.sop` и `ry48p.2.sop`, по-видимому, требуют слишком большого объема памяти. Например, при попытке решить `kro124p.4.sop` доступные 256 ГБ оказались исчерпаны уже при расчете функции Беллмана на 24-м слое пространства состояний, которых всего 100 — в этой задаче 99 городов не считая базу и терминал.

Таблица 1.2: Точное ДП для TSP-PC

НАЗВАНИЕ	ЗНАЧЕНИЕ	ВРЕМЯ СЧЕТА (сек)	ПАМЯТЬ(МБ)
br17.10.sop	55	0,0821	9,07
br17.12.sop	55	0,0461	8,30
ESC07.sop	2 125	0,013	7,32
ESC11.sop	2 075	0,022	7,59
ESC12.sop	1 675	0,034	7,78
ESC25.sop	1 681	91	2 059,21
ft53.4.sop	14 425	2,035	73,87
ft70.4.sop	53 530	31	968,88
p43.3.sop	28 835	2 549	45 756,44
p43.4.sop	83 005	0,665	22,67
rbg109a.sop	1 038	0,345	13,12
rbg150a.sop	1 750	0,715	18,54
rbg174a.sop	2 033	165	2 972,50
rbg253a.sop	2 950	178	3 089,98
ry48p.3.sop	19 894	7 805	128 432,31
ry48p.4.sop	31 446	0,696	34,94

Можно предложить два общих подхода к увеличению размерности задач, разрешимых ДП:

экстенсивный Использовать схему параллелизации с распределенной памятью для задействования большего объема, чем доступен на одном (суперкомпьютерном) узле, например, [136, 137]. Вопрос о разложении TSP-PC также поднимался в [104].

интенсивный Найти эффективную оценку снизу и реализовать схему ветвей и границ в динамическом программировании [51]. Этот подход применялся, например, в [31]. Среди оценок снизу особо отметим [138].

В таблицах 1.3 и 1.4 приведены результаты, полученные усеченным ДП в попятной и прямой постановке соответственно. Во всех экземплярах, которые удалось решить точным ДП (см. таблицу 1.2), либо один, либо оба варианта эвристики достигли известных оптимальных значений при $H = 10^5$. Однако, нам не удалось улучшить известные верхние

оценки для открытых экземпляров. Для некоторых открытых экземпляров мы сделали попытку взять бóльшие значения H, однако, например, для `kro124p.4.sop`, наиболее «сильно ограниченного» из четырех вариантов, наилучшая известная верхняя оценка 76 103 была достигнута при $H = 2 \cdot 10^7$ (время счета порядка 3 часов, расход памяти порядка 80 ГБ) и не была улучшена при $H = 4 \cdot 10^7$ (время счета порядка 6 часов, расход памяти порядка 140 ГБ).

1.5.4 Задача коммивояжера с условиями предшествования и зависимостью от времени / TD-TSP-PC (TD-SOP)

Начнем с описания функции стоимости перемещений типа *traveling deliveryman* [74] в терминах функции стоимости *с зависимостью от списка заданий* $\mathfrak{c}$, $\mathfrak{c}\colon \mathcal{P}(1\,..\,n) \times 0\,..\,n \times 1\,..\,n+1 \to \mathbb{R}$. Напомним, что «собственные» города нумеруются от 1 до n и кроме них есть еще особые база 0 и терминал $n+1$; всего получается $n+2$ города, следовательно, $n+1$ перемещений-ребер маршрута. Пусть $\mathfrak{d}(i,j)$ обозначает расстояние между городами i и j. Тогда в прямой процедуре ДП $\mathfrak{c}$ определяется следующим образом:

$$\mathfrak{c}(K,i,j) \triangleq \begin{cases} (n+1)\mathfrak{d}(i,j) & (K=\varnothing) \wedge (i=0); \\ (n-|K|)\mathfrak{d}(i,j) & (K \in \mathcal{P}'(1\,..\,n)) \wedge (i \neq 0). \end{cases}$$

Для попятной процедуры используем двойственную функцию стоимости, зависящую от списка *не*выполненных заданий, аналогичную вводимой в [139], $\bar{\mathfrak{c}}\colon 0\,..\,n \times 1\,..\,n+1 \times \mathcal{P}(1\,..\,n) \to \mathbb{R}$, полагаем

$$\bar{\mathfrak{c}}(i,j,K) \triangleq \begin{cases} 1 \cdot \mathfrak{d}(i,j) & (K=\varnothing) \wedge (j=\mathbb{t}); \\ (|K|+2)\mathfrak{d}(i,j) & (K \in \mathcal{P}(1\,..\,n)) \wedge (j \neq \mathbb{t}). \end{cases}$$

В таблице 1.5 указано время счета и расход памяти при решении точным ДП. Как и ожидалось, удалось оптимально решить экземпляры TD-SOP, полученные из успешно решенных точным ДП экземпляров TSP-PC: нам удалось (впервые) закрыть `ft53.4.sop`, `ft70.4.sop`, `rbg109a.sop`, `rbg150a.sop`, `rbg174b.sop`, `rbg253a.sop` и `ry48p.3.sop`.

В таблицах 1.6 и 1.7 приведены результаты применения, соответственно, попятной и прямой процедуры эвристики усеченного ДП. Для всех экземпляров, закрытых в [74], исключая `p43.1.sop`, а также для тех, которые удалось закрыть автору (см. таблицу 1.5), прямой вариант эвристики достиг оптимальных значений при $H = 10^5$. Для всех экземпляров, не закрытых ни в [74], ни автором [77], прямой эвристикой удалось *улучшить* оценки сверху; исключением стала задача `rbg341a.sop`, в которой попятная эвристика оказалась не более чем на 1% лучше; таким образом, для открытых экземпляров, источником колонки BEST UB стала, за одним исключением, *прямая* процедура усеченного ДП.

Судя по сравнению с лучшими известными верхними оценками, функция стоимости типа *traveling deliveryman* [74] более подходит для прямой процедуры усеченного ДП: можно сказать, что локальные экстремумы «замечаемые» этой процедурой оказываются ближе к глобальному экстремуму, чем то же в *попятной* процедуре. Этого следовало ожидать, ведь в такой функции стоимости начальные перемещения «перевешивают» конечные, поскольку домножаются на бóльшие числа (напомним, что зависимость от времени типа traveling deliveryman фактически представляет собой домножение расстояние на номер перемещения в маршруте, считая с последнего).

Таблица 1.3: Усеченное ДП (*попятная* процедура) для TSP-PC, в сравнении с лучшими известными верхними границами [34]. Время t в секундах

	H=1		H=10		H=100		H=1000		H=10000		H=100000		
НАЗВАНИЕ	GAP,%	t	GAP,%	t	GAP,%	t	GAP,%	t	GAP,%	t	GAP,%	t	BEST UB
br17.10.sop	44	0,022	2	0,025	**0**	0,036	**0**	0,073	**0**	0,082	**0**	0,07	55
br17.12.sop	11	0,026	**0**	0,021	**0**	0,026	**0**	0,049	**0**	0,06	**0**	0,055	55
ESC07.sop	42	0,011	4	0,01	**0**	0,009	**0**	0,014	**0**	0,01	**0**	0,014	2 125
ESC11.sop	23	0,014	16	0,018	**0**	0,02	**0**	0,024	**0**	0,023	**0**	0,026	2 075
ESC12.sop	9	0,016	17	0,021	4	0,021	**0**	0,041	**0**	0,047	**0**	0,032	1 675
ESC25.sop	94	0,03	79	0,041	60	0,086	20	0,245	15	1,086	**0**	7,415	1 681
ESC47.sop	111	0,097	106	0,079	53	0,219	43	0,859	37	7,931	16	75,337	1 288
ESC63.sop	17	0,077	10	0,107	4	0,317	**0**	1,563	**0**	16,131	**0**	173,379	62
ESC78.sop	17	0,089	9	0,144	8	0,32	8	1,286	9	11,672	8	107,237	18 230
ft53.1.sop	52	0,074	27	0,111	15	0,225	13	0,977	9	8,85	7	78,711	7 531
ft53.2.sop	51	0,058	32	0,105	18	0,187	7	0,74	5	6,165	5	56,754	8 026
ft53.3.sop	43	0,081	18	0,094	15	0,158	11	0,508	1	2,846	**0**	24,642	10 262
ft53.4.sop	26	0,063	15	0,057	5	0,098	1	0,255	**0**	0,843	**0**	1,982	14 425
ft70.1.sop	20	0,089	13	0,126	8	0,417	7	2,076	5	19,236	4	177,209	39 313
ft70.2.sop†	22	0,173	13	0,15	12	0,346	9	1,525	5	13,273	5	120,429	40 419
ft70.3.sop	32	0,075	16	0,09	10	0,23	10	0,923	9	7,224	7	63,949	42 535
ft70.4.sop	20	0,079	8	0,082	4	0,145	2	0,47	1	1,869	**0**	14,283	53 530
kro124p.1.sop†	35	0,14	30	0,275	22	0,719	20	5,534	18	54,066	17	518,319	39 420
kro124p.2.sop†	32	0,124	20	0,203	21	0,646	19	4,518	13	39,067	11	395,117	41 336
kro124p.3.sop†	36	0,152	22	0,181	20	0,462	18	2,204	10	22,108	8	214,934	49 499
kro124p.4.sop†	40	0,202	18	0,171	11	0,247	10	0,769	6	5,501	5	50,246	76 103
p43.1.sop	3	0,053	3	0,091	1	0,138	1	0,563	1	3,981	1	36,359	28 140
p43.2.sop	6	0,063	4	0,052	2	0,118	1	0,486	1	2,509	1	21,269	28 480
p43.3.sop	5	0,047	3	0,062	2	0,107	2	0,301	1	1,433	1	12,645	28 835
p43.4.sop	3	0,045	1	0,07	1	0,08	**0**	0,146	**0**	0,489	**0**	0,459	83 005
prob.100.sop†	97	0,174	122	0,24	64	0,793	40	5,168	41	49,681	30	509,634	1 163
prob42.sop	88	0,051	58	0,059	27	0,144	19	0,61	6	4,557	3	40,799	243
rbg048a.sop	58	0,057	20	0,065	7	0,147	2	0,661	1	3,976	**0**	33,797	351
rbg050c.sop	39	0,06	24	0,062	16	0,134	6	0,431	4	2,579	2	21,127	467
rbg109a.sop	26	0,152	12	0,173	2	0,358	**0**	0,325	**0**	0,483	**0**	0,433	1 038
rbg150a.sop	20	0,2	6	0,177	2	0,275	1	0,446	**0**	0,622	**0**	0,49	1 750
rbg174b.sop	21	0,179	10	0,265	3	0,361	1	0,618	**0**	1,906	**0**	9,493	2 033
rbg253a.sop	19	0,405	7	0,421	3	0,65	1	1,414	1	2,984	**0**	14,53	2 950
rbg323a.sop	39	0,469	18	0,595	9	0,841	6	3,19	4	23,372	3	191,163	3 140
rbg341a.sop	58	0,376	21	0,525	10	0,839	3	2,722	1	17,527	1	121,751	2 568
rbg358a.sop	72	0,427	41	0,533	31	1,293	19	5,582	14	47,407	11	420,849	2 545
rbg378a.sop†	59	0,443	33	0,588	26	1,092	13	6,469	7	53,331	4	460,838	2 816
ry48p.1.sop	35	0,058	18	0,068	18	0,178	5	0,776	3	6,689	2	61,4	15 805
ry48p.2.sop†	31	0,062	18	0,064	16	0,153	6	0,688	5	5,09	4	46,263	16 666
ry48p.3.sop	39	0,086	33	0,068	12	0,134	5	0,416	3	2,39	1	21,939	19 894
ry48p.4.sop	27	0,114	18	0,073	8	0,107	1	0,213	**0**	0,617	**0**	0,822	31 446

Таблица 1.4: Усеченное ДП (*прямая* процедура) для TSP-PC, в сравнении с лучшими известными верхними границами [34]. Время t в секундах

	H=1		H=10		H=100		H=1000		H=10000		H=100000		
НАЗВАНИЕ	GAP,%	t	GAP,%	t	GAP,%	t	GAP,%	t	GAP,%	t	GAP,%	t	BEST UB
br17.10.sop	44	0,017	10	0,028	6	0,034	**0**	0,066	**0**	0,1	**0**	0,077	55
br17.12.sop	44	0,017	10	0,028	6	0,026	**0**	0,058	**0**	0,057	**0**	0,052	55
ESC07.sop	28	0,009	**0**	0,013	**0**	0,01	**0**	0,012	**0**	0,012	**0**	0,012	2 125
ESC11.sop	41	0,014	20	0,013	3	0,018	**0**	0,024	**0**	0,024	**0**	0,033	2 075
ESC12.sop	23	0,02	34	0,017	**0**	0,023	**0**	0,034	**0**	0,047	**0**	0,041	1 675
ESC25.sop	45	0,029	20	0,032	4	0,068	4	0,282	1	1,312	**0**	7,39	1 681
ESC47.sop	185	0,092	93	0,088	83	0,203	29	1,07	17	7,716	11	73,259	1 288
ESC63.sop	15	0,075	21	0,125	**0**	0,365	**0**	1,914	**0**	16,918	**0**	179,1	62
ESC78.sop	24	0,091	15	0,125	12	0,325	14	1,238	8	7,711	7	80,108	18 230
ft53.1.sop	33	0,056	43	0,114	29	0,269	13	1,188	9	9,26	8	91,906	7 531
ft53.2.sop	44	0,064	33	0,076	14	0,22	13	1,186	11	7,729	5	75,682	8 026
ft53.3.sop	48	0,063	30	0,101	29	0,159	21	0,576	9	3,412	5	35,383	10 262
ft53.4.sop	29	0,065	14	0,065	3	0,088	2	0,353	1	1,278	**0**	1,972	14 425
ft70.1.sop	16	0,075	15	0,156	9	0,518	7	2,447	7	23,447	5	219,542	39 313
ft70.2.sop†	22	0,1	10	0,176	9	0,435	8	1,935	4	17,225	2	168,168	40 419
ft70.3.sop	33	0,071	20	0,088	14	0,234	13	1,238	10	8,271	7	93,208	42 535
ft70.4.sop	17	0,088	9	0,102	5	0,136	3	0,452	2	2,089	1	13,97	53 530
kro124p.1.sop†	33	0,131	30	0,29	26	1,067	22	5,955	16	59,627	7	570,541	39 420
kro124p.2.sop†	35	0,111	24	0,244	25	0,907	16	4,749	5	47,476	3	464,848	41 336
kro124p.3.sop†	57	0,159	29	0,158	22	0,495	20	2,107	13	21,078	6	210,536	49 499
kro124p.4.sop†	30	0,133	16	0,118	17	0,271	6	1,002	3	5,87	3	53,813	76 103
p43.1.sop	10	0,046	4	0,076	4	0,126	3	0,653	3	3,606	3	33,06	28 140
p43.2.sop	10	0,043	5	0,05	3	0,114	2	0,448	1	2,42	1	20,855	28 480
p43.3.sop	6	0,051	4	0,061	3	0,094	2	0,319	1	1,542	1	11,811	28 835
p43.4.sop	3	0,044	2	0,067	1	0,067	1	0,155	**0**	0,518	**0**	0,442	83 005
prob.100.sop†	185	0,162	96	0,245	65	0,935	57	5,072	65	46,928	45	480,158	1 163
prob42.sop	89	0,04	71	0,07	32	0,161	19	0,786	15	4,367	5	38,177	243
rbg048a.sop	38	0,05	14	0,061	8	0,14	8	0,674	4	3,446	4	30,109	351
rbg050c.sop	20	0,051	24	0,078	15	0,148	12	0,579	9	2,781	1	24,5	467
rbg109a.sop	40	0,124	9	0,181	2	0,324	**0**	0,39	**0**	0,434	**0**	0,42	1 038
rbg150a.sop	27	0,195	7	0,242	2	0,289	1	0,536	**0**	0,689	**0**	0,631	1 750
rbg174b.sop	21	0,214	4	0,301	2	0,385	1	0,922	1	2,175	1	10,578	2 033
rbg253a.sop	21	0,333	5	0,426	3	0,633	1	1,475	**0**	3,729	**0**	16,27	2 950
rbg323a.sop	28	0,356	9	0,55	6	1,026	4	3,827	3	29,097	3	234,475	3 140
rbg341a.sop	47	0,4	18	0,535	10	1,027	6	3,088	3	19,48	2	133,97	2 568
rbg358a.sop	62	0,46	30	0,55	19	1,549	9	8,084	11	59,282	3	594,099	2 545
rbg378a.sop†	45	0,482	26	0,6	18	1,468	10	6,808	6	55,634	3	473,005	2 816
ry48p.1.sop	26	0,047	24	0,078	11	0,189	10	0,955	8	6,007	7	57,849	15 805
ry48p.2.sop†	26	0,062	22	0,072	25	0,156	7	0,782	6	4,639	3	41,846	16 666
ry48p.3.sop	37	0,055	25	0,08	5	0,134	4	0,439	3	2,272	1	20,48	19 894
ry48p.4.sop	34	0,084	17	0,075	5	0,105	1	0,278	**0**	0,851	**0**	0,808	31 446

Таблица 1.5: Точное ДП для TD-SOP

НАЗВАНИЕ	ЗНАЧЕНИЕ	ВРЕМЯ СЧЕТА (сек)	ПАМЯТЬ(МБ)
br17.10.sop	461	0,077	9,07
br17.12.sop	461	0,046	8,06
ESC07.sop	7 825	0,011	7,32
ESC11.sop	11 686	0,025	7,59
ESC12.sop	11 158	0,029	7,32
ESC25.sop	17 752	99	2 059,26
ft53.4.sop	383 057	2,196	73,88
ft70.4.sop	195 0951	34	968,88
p43.3.sop	455 810	2 628	45 756,44
p43.4.sop	109 3510	0,46	22,67
rbg109a.sop	59 115	0,301	13,12
rbg150a.sop	114 753	0,512	18,54
rbg174a.sop	180 869	177	2 972,50
rbg253a.sop	370 847	192	3 089,98
ry48p.3.sop	461 275	8 331	128 432,31
ry48p.4.sop	736 631	0,896	34,94

Таблица 1.6: Усеченное ДП (*попятная* процедура) для TD-SOP, в сравнении с наилучшими известными верхними оценками. Время в секундах

	H=1		H=10		H=100		H=1000		H=10000		H=100000		
НАЗВАНИЕ	GAP,%	t	GAP,%	t	GAP,%	t	GAP,%	t	GAP,%	t	GAP,%	t	BEST UB
br17.10.sop	53	0,024	26	0,029	23	0,026	**0**	0,058	**0**	0,09	**0**	0,079	461
br17.12.sop	53	0,018	23	0,021	17	0,025	**0**	0,047	**0**	0,055	**0**	0,053	461
ESC07.sop	100	0,009	38	0,009	**0**	0,01	**0**	0,009	**0**	0,01	**0**	0,011	7 825
ESC11.sop	65	0,013	40	0,018	19	0,018	**0**	0,024	**0**	0,024	**0**	0,024	11 686
ESC12.sop	24	0,014	47	0,016	19	0,03	**0**	0,031	**0**	0,032	**0**	0,041	11 158
ESC25.sop	220	0,038	212	0,061	170	0,065	99	0,235	51	1,168	16	8,178	17 752
ESC47.sop†	235	0,057	201	0,078	205	0,213	138	0,881	125	8,342	100	81,009	25 394
ESC63.sop	24	0,068	19	0,114	7	0,308	**0**	1,553	**0**	16,164	**0**	175,684	2 015
ESC78.sop†	76	0,131	63	0,185	63	0,299	60	1,351	61	11,216	59	108,963	619 785
ft53.1.sop†	120	0,104	77	0,117	79	0,27	57	1,006	52	8,72	47	85,993	171 616
ft53.2.sop†	104	0,091	61	0,102	71	0,249	42	0,762	34	6,775	29	62,172	192 482
ft53.3.sop†	77	0,059	67	0,064	41	0,156	32	0,589	18	3,327	15	29,267	264 829
ft53.4.sop	45	0,066	31	0,077	16	0,11	10	0,245	4	0,932	**0**	2,128	383 057
ft70.1.sop†	47	0,085	34	0,136	28	0,469	25	2,055	23	19,744	22	189,561	1 315 559
ft70.2.sop†	43	0,076	37	0,176	30	0,333	26	1,514	20	14,402	20	134,375	1 350 826
ft70.3.sop†	44	0,115	31	0,093	18	0,257	18	0,906	15	8,058	10	77,155	1 523 059
ft70.4.sop	25	0,069	12	0,082	9	0,121	7	0,385	5	1,842	2	15,297	1 950 951
kro124p.1.sop†	56	0,203	61	0,298	62	0,716	61	5,572	51	55,005	37	527,003	1 860 438
kro124p.2.sop†	67	0,124	50	0,201	42	0,782	40	4,43	31	44,933	30	431,892	1 930 003
kro124p.3.sop†	59	0,157	32	0,241	39	0,482	27	2,352	28	22,826	23	224,304	2 414 367
kro124p.4.sop†	55	0,139	30	0,133	23	0,23	19	0,791	15	5,674	10	53,267	3 823 232
p43.1.sop	833	0,07	813	0,073	782	0,171	774	0,633	748	3,85	724	37,454	116 170
p43.2.sop	398	0,059	355	0,052	343	0,123	341	0,497	327	2,524	310	21,907	203 035
p43.3.sop	111	0,072	97	0,055	88	0,088	82	0,327	291	1,497	272	12,743	455 810
p43.4.sop	84	0,044	67	0,047	42	0,068	37	0,169	3	0,531	**0**	0,492	1 093 510
prob.100.sop†	417	0,111	256	0,233	192	0,679	133	5,226	141	51,895	104	537,948	51 195
prob42.sop	219	0,043	125	0,054	81	0,149	65	0,623	56	4,671	39	42,184	4 315
rbg048a.sop†	87	0,052	46	0,073	33	0,159	22	0,657	18	4,28	15	36,476	8 002
rbg050c.sop†	100	0,053	81	0,06	61	0,182	48	0,46	36	2,695	27	24,507	9 236
rbg109a.sop	23	0,183	12	0,183	4	0,163	1	0,266	**0**	0,362	**0**	0,32	59 115
rbg150a.sop	33	0,202	13	0,162	4	0,214	1	0,552	1	0,68	**0**	0,592	114 753
rbg174b.sop	22	0,277	10	0,244	5	0,33	2	0,869	1	2,141	1	10,212	180 869
rbg253a.sop	23	0,257	10	0,287	5	0,422	2	1,025	1	3,164	1	15,296	370 847
rbg323a.sop†	41	0,418	19	0,489	11	0,903	8	3,262	5	24,697	3	197,597	511 755
rbg341a.sop†	57	0,363	22	0,447	12	0,759	4	2,868	2	18,17	**0**	128,529	431 816
rbg358a.sop†	58	0,45	29	0,648	19	1,068	15	5,56	10	46,906	8	433,96	497 657
rbg378a.sop†	51	0,51	28	0,565	21	1,069	16	6,371	9	55,199	5	486,266	555 774
ry48p.1.sop†	85	0,06	63	0,078	63	0,195	38	0,757	35	6,846	31	64,365	351 462
ry48p.2.sop†	72	0,05	50	0,089	36	0,219	34	0,7	30	5,273	26	48,407	384 622
ry48p.3.sop	79	0,049	77	0,075	33	0,127	26	0,448	18	2,41	9	21,023	461 275
ry48p.4.sop	44	0,105	33	0,081	22	0,101	8	0,221	4	0,662	**0**	0,842	736 631

Таблица 1.7: Усеченное ДП (*прямая* процедура) для TD-SOP, в сравнении с наилучшими известными верхними оценками. Время в секундах

	H=1		H=10		H=100		H=1000		H=10000		H=100000		
НАЗВАНИЕ	GAP,%	t	GAP,%	t	GAP,%	t	GAP,%	t	GAP,%	t	GAP,%	t	BEST UB
br17.10.sop	14	0,018	1	0,019	**0**	0,031	**0**	0,063	**0**	0,099	**0**	0,095	461
br17.12.sop	14	0,019	1	0,022	**0**	0,034	**0**	0,046	**0**	0,065	**0**	0,057	461
ESC07.sop	24	0,01	**0**	0,01	**0**	0,009	**0**	0,009	**0**	0,009	**0**	0,017	7 825
ESC11.sop	46	0,013	22	0,021	**0**	0,033	**0**	0,024	**0**	0,024	**0**	0,023	11 686
ESC12.sop	21	0,015	8	0,015	**0**	0,018	**0**	0,029	**0**	0,031	**0**	0,031	11 158
ESC25.sop	19	0,031	5	0,039	**0**	0,087	**0**	0,274	**0**	1,324	**0**	7,673	17 752
ESC47.sop†	117	0,049	44	0,069	25	0,229	1	1,091	1	7,891	**0**	74,715	25 394
ESC63.sop	3	0,064	7	0,101	**0**	0,332	**0**	1,893	**0**	17,148	**0**	181,865	2 015
ESC78.sop†	30	0,097	15	0,167	6	0,345	5	1,258	2	8,014	**0**	78,304	619 785
ft53.1.sop†	52	0,082	24	0,097	12	0,314	10	1,364	1	9,306	**0**	94,009	171 616
ft53.2.sop†	45	0,069	24	0,104	6	0,261	8	1,124	3	7,777	**0**	76,328	192 482
ft53.3.sop†	33	0,08	19	0,066	13	0,139	12	0,584	2	3,468	**0**	34,868	264 829
ft53.4.sop	29	0,054	13	0,064	3	0,09	**0**	0,3	**0**	1,216	**0**	2,112	383 057
ft70.1.sop†	19	0,089	9	0,144	2	0,519	1	2,466	1	23,48	**0**	224,221	1 315 559
ft70.2.sop†	24	0,083	4	0,155	3	0,418	1	1,961	1	17,135	**0**	169,155	1 350 826
ft70.3.sop†	24	0,073	9	0,096	4	0,249	4	1,149	3	8,508	**0**	88,883	1 523 059
ft70.4.sop	13	0,069	5	0,077	2	0,126	1	0,426	1	2,104	**0**	14,206	1 950 951
kro124p.1.sop†	15	0,168	9	0,363	6	1,117	3	6,1	1	59,256	**0**	572,816	1 860 438
kro124p.2.sop†	19	0,116	9	0,236	9	0,905	4	4,648	1	47,775	**0**	462,965	1 930 003
kro124p.3.sop†	39	0,147	16	0,237	9	0,464	8	2,093	3	21,191	**0**	210,482	2 414 367
kro124p.4.sop†	21	0,097	9	0,162	10	0,287	2	1,135	2	5,887	**0**	54,295	3 823 232
p43.1.sop	15	0,073	48	0,124	3	0,171	3	0,69	3	3,712	3	35,017	116 170
p43.2.sop	13	0,07	29	0,067	5	0,117	4	0,465	**0**	2,493	**0**	22,26	203 035
p43.3.sop	5	0,057	7	0,111	5	0,084	1	0,353	1	1,681	**0**	12,853	455 810
p43.4.sop	3	0,063	1	0,048	1	0,065	1	0,156	**0**	0,57	**0**	0,519	1 093 510
prob.100.sop†	50	0,109	18	0,207	9	0,929	6	4,808	4	46,862	**0**	487,404	51 195
prob42.sop	22	0,041	14	0,055	1	0,149	1	0,792	**0**	4,43	**0**	39,376	4 315
rbg048a.sop†	28	0,166	14	0,065	6	0,246	5	0,656	**0**	3,529	**0**	30,951	8 002
rbg050c.sop†	22	0,054	23	0,061	14	0,132	11	0,614	**0**	3,28	**0**	25,566	9 236
rbg109a.sop	39	0,14	8	0,177	1	0,175	**0**	0,292	**0**	0,383	**0**	0,316	59 115
rbg150a.sop	41	0,169	10	0,162	2	0,244	**0**	0,511	**0**	0,834	**0**	0,647	114 753
rbg174b.sop	19	0,186	5	0,283	1	0,357	1	0,971	**0**	2,537	**0**	11,045	180 869
rbg253a.sop	21	0,261	6	0,326	2	0,503	1	1,139	**0**	3,619	**0**	17,113	370 847
rbg323a.sop†	24	0,347	8	0,649	3	1,142	2	4,037	1	29,564	**0**	238,61	511 755
rbg341a.sop†	47	0,458	19	0,477	9	1,046	5	2,939	3	19,753	1	136,953	431 816
rbg358a.sop†	46	0,486	20	0,649	12	1,704	6	7,393	4	60,004	**0**	598,898	497 657
rbg378a.sop†	37	0,441	22	0,628	10	1,482	5	6,806	2	55,988	**0**	475,421	555 774
ry48p.1.sop†	14	0,059	7	0,082	1	0,193	1	0,972	**0**	5,941	**0**	58,359	351 462
ry48p.2.sop†	18	0,05	10	0,089	5	0,184	5	0,814	3	4,637	**0**	42,723	384 622
ry48p.3.sop	21	0,046	10	0,058	4	0,129	**0**	0,452	**0**	2,368	**0**	20,697	461 275
ry48p.4.sop	29	0,092	14	0,096	1	0,097	**0**	0,252	**0**	0,828	**0**	0,811	736 631

Глава 2

Обобщенная задача коммивояжера на узкие места с условиями предшествования и зависимостью от списка заданий / PSD-BGTSP-PC

В настоящем разделе мы представим наиболее обобщенную формулировку маршрутной задачи. Полностью она звучит как «обобщенная задача курьера с агрегированием затрат на узкие места, внутренними работами и зависимостью от списка невыполненных заданий»; перевод «Past Sequence-dependent Precedence Constrained Botteleneck Generalized Traveling Salesman Problem» (далее PSD-BGTSP-PC) представляется адекватным. Раздел основан на результатах [61] (доказательство корректности ДП, вычислительный эксперимент) и [62] (применение эвристики усеченного ДП, анализ ее сложности, схема параллелизации, вычислительный эксперимент). Определения и обозначения главы 1 действуют за исключением явно указанных случаев замены. Для большей аналогии с моделями А. Г. Ченцова (см., напр., [25]), ДП будет представлено в *попятной* формулировке, соответственно, существенные списки заданий будут *фильтрами*, а не *идеалами*; отметим, что, хотя модели [61,62] опубликованы в формализме *фундированных отношений*[1], в их изложении мы будем придерживаться формализма частично упорядоченных множеств, как в гл. 1. В оценки сложности эвристики усеченного ДП [62] внесены исправления, основанные на результатах [44,77]; эти исправления отмечены в тексте главы в явном виде.

2.1 Кластеры, мегаполисы и внутренние работы. Дополнительные определения и постановка задачи

2.1.1 Введение

В вариантах TSP, представленных выше, города полагались элементарными объектами, не имеющими структуры. В контексте *обобщенной* задачи коммивояжера города наделяются некоторой структурой. Точнее, рассматриваются *мегаполисы* (англ. *megalopolis*), или *кластеры* — неперекрывающиеся[2] конгломераты городов. В зависимости от нюансов постановки, ставится задача либо просто *посетить* каждый мегаполис, побывав в каком-то одном из его городов (см. [5, §1.5.3 Задача коммивояжера с выбором], [2, Ch.13,

[1] см. обсуждение в разделе 1.4.5

[2] исключение — перекрывающиеся конгломераты — описано в [140] для приложений к лазерной сварке

§1.6 Generalized Traveling Salesman Problem])[3], либо, прибыв в очередной мегаполис, посетить *все* составляющие его города до того как отправиться в следующий мегаполис; в последнем случае обычно говорят о *кластерной* задаче коммивояжера (англ. Clustered TSP), см. [20] и [5, п. 1.5.2].

Замечание. Есть разные причины возникновения *структуры*; их можно разделить на *внешнюю кластеризацию*, часто предназначенную сократить расчеты по сравнению с представлением в виде обычной TSP, см. напр. [37]; в [107] обычная TSP с определенного рода условиями предшествованиями, фактически, сводится к обобщенному варианту, для которого затем предлагается эффективный алгоритм динамического программирования[4], и *модельные соображения*, например, в задачах маршрутизации инструмента в машинах листовой резки [39, 85, 88, 141, 142] или маршрутизации сотрудников АЭС при демонтаже энергоблока с целью минимизировать дозовую нагрузку [50]. Отметим также применение внутренней структуры для отражения *состояния системы*: в задаче о перегрузке тепловыделяющих стержней (ТС) [143, § 1.5] один из городов обозначает состояние «рабочая штанга держит ТС», другой — «рабочая штанга свободна»; при этом накладываются ограничения на перемещения между городами внутри мегаполиса (см. подробнее ниже).

Постановка, охватывающая и кластерную и обобщенную задачу, по-видимому, впервые описана в [144] и далее развита, например, в [86, 89] в виде обобщенной задачи курьера *с внутренними работами*, средствами которых можно выразить требование посещать единственный город в каждом мегаполисе, или посетить все города мегаполиса до перемещения к следующему, или выполнить некоторые другие действия, стоимость которых зависит от расположения городов в каждом мегаполисе; именно эту постановку, исключая ограничения в виде условий предшествования, мы будем далее называть «обобщенной задачей коммивояжера» (Generalized TSP, GTSP), не оговаривая особо различие между GTSP и CTSP. Для демонстрации обобщенности, выпишем в явном виде выражения *внутренних работ* обобщенной и кластерной задачи:

Generalized TSP. Каждый мегаполис посещается единожды, т. е. посещается некоторый его город. Это условие обеспечивается следующими ограничениями на посещение: $\forall i \in 1\,..\,n\ \mathbb{M}_i \triangleq \big\{(b,b) : b \in M_i\big\}$, то есть, агент покидает мегаполис через тот же город, через который туда прибыл; внутренних работ в этом случае не требуется, поэтому мы присваиваем им нулевую стоимость: $\forall i \in 1\,..\,n\ \forall b \in \mathbb{M}_i\ \forall K \in \mathcal{F}\ c_i(b,b,K) = 0$.

Clustered TSP. Прибыв в мегаполис, нужно посетить все города прежде чем его покинуть; фактически, внутри мегаполиса ставится *незамкнутая* TSP с фиксированным пунктом выхода (SHCP (ii) [18]). Запретим покидать мегаполис через тот город, в который прибыли, $\forall i \in 1\,..\,n\ \mathbb{M}_i \triangleq \big\{(a,b) : (a,b \in M_i) \wedge (b \neq a)\big\}$[5], и присвоим внутренним работам между парой городов из мегаполиса стоимость оптимального решения соответствующего экземпляра TSP(-PC).

2.1.2 Исходные данные

Обобщенность задаче придает разбиение всего множества городов на *кластеры* или *мегаполисы* (англ. megalopolis), каждый из которых должен быть посещен единственный

[3] также встречается название «International TSP»

[4] подобные аргументы можно применить и к GTSP с особыми условиями предшествования [108], получив в некотором роде «мегаполис мегаполисов»

[5] Здесь предполагается, что городов в каждом кластере более одного; на *вырожденных* кластерах, естественно, допустимо и зайти и выйти через один и тот же город.

раз; при этом задача распадается на два уровня: «нижний», заключающийся в оптимизации работ внутри мегаполиса и «верхний» — маршрутизацию перемещений *между* мегаполисами. База обычно представляет собой «вырожденный» мегаполис, состоящий из одного единственного города. О разновидностях обобщенной задачи коммивояжера см. в [2, Ch. 13] и [5, пп. 1.5.2, 1.5.3]. Вновь отметим, что не все обобщенные задачи происходят исключительно из кластеризации исходного набора городов: двухуровневая структура может происходить из содержательной постановки задачи.

Мы не рассматриваем здесь процедуры разбиения множества городов на кластеры и в дальнейшем предполагаем множество городов и их разбиение на мегаполисы уже заданным: фиксируем непустое множество X и точку $x^0 \in X$ — *базу*. В постановке (PSD-TSP-PC) города и их индексы обозначались числами из $1\,..\,n$, $n \in \mathbb{N}$; настоящей постановке интервал $1\,..\,n$ содержит индексы *мегаполисов* — *конечных* множеств вида

$$M_1 \subset X, \ldots, M_n \subset X,$$

стесненных условиями

$$\big((x^0 \notin M_j)\ \forall j \in 1\,..\,n\big) \wedge \big(M_p \cap M_q = \varnothing\ \forall p \in 1\,..\,n\ \forall q \in 1\,..\,n \setminus \{p\}\big); \tag{2.1.1}$$

то есть, мегаполисы полагаются *дизъюнктными* подмножествами X, причем база от мегаполисов обособлена. В отличие от главы 1, мы рассматриваем *незамкнутую* задачу — без *терминала*. Отметим два исключения из этих правил: в [140] мегаполисы могут перекрываться; в [91] предложен набор экземпляров GTSP-PC, основанный на TSP-PC из TSPLIB, где каждый отдельный город в постановке TSP-PC превращен в мегаполис, соответственно, представлены *базовый мегаполис* и *конечный мегаполис*.

Дополнительно обеспечим возможность ограничить допустимые роли городов во внешних перемещениях между мегаполисами и работах внутри них. Введем отношения

$$\mathbb{M}_1 \in \mathcal{P}'(M_1 \times M_1), \ldots, \mathbb{M}_n \in \mathcal{P}'(M_n \times M_n). \tag{2.1.2}$$

При $j \in 1\,..\,n$, УП $z \in \mathbb{M}_j$ определяют допустимые варианты выполнения *внутренних работ* в мегаполисе $\mathbb{M}_j$: $\mathrm{pr}_1(z)$ определяет допустимый *пункт прибытия*, а $\mathrm{pr}_2(z)$ — *пункт отправления*.

Таким образом, мы рассматриваем вопрос об организации системы перемещений

$$\begin{aligned}\Big(x^0\Big) &\to \Big(\mathrm{pr}_1(z_1) \in M_{\alpha_1} \rightsquigarrow \mathrm{pr}_2\left(z_1\right) \in M_{\alpha_1}\Big) \to \\ &\to \Big(\mathrm{pr}_1\left(z_2\right) \in M_{\alpha_2} \rightsquigarrow \mathrm{pr}_2\left(z_2\right) \in M_{\alpha_2}\Big) \to \\ &\to \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \to \\ &\to \Big(\mathrm{pr}_1\left(z_n\right) \in M_{\alpha_n} \rightsquigarrow \mathrm{pr}_2\left(z_n\right) \in M_{\alpha_n}\Big).\end{aligned} \tag{2.1.3}$$

где α — допустимый маршрут по $1\,..\,n$, а УП $z_1, \ldots, z_n$ с ним согласуются — удовлетворяют условиям

$$z_1 \in \mathbb{M}_{\alpha_1}, \ldots, z_n \in \mathbb{M}_{\alpha_n}. \tag{2.1.4}$$

Объектами нашего выбора в (2.1.3) — решением задачи — являются перестановка α, именуемая *маршрутом*, и кортеж $((\varnothing, x^0), z_1, \ldots, z_n)$, согласованный с маршрутом и ограничениями (2.1.2) в смысле (2.1.4) и именуемый далее *трассой*. Напомним, что выбор α может быть стеснен *условиями предшествования*.

Введем обозначения для всех допустимых *пунктов прибытия* и *пунктов отправления* в каждом мегаполисе:

$$\begin{aligned}\mathbb{M}_j^{(\mathrm{in})} &\triangleq \{\mathrm{pr}_1(z) : z \in \mathbb{M}_j\}\ \forall j \in 1\,..\,n, \\ \mathbb{M}_j^{(\mathrm{out})} &\triangleq \{\mathrm{pr}_2(z) : z \in \mathbb{M}_j\}\ \forall j \in 1\,..\,n.\end{aligned} \tag{2.1.5}$$

В терминах мегаполисов и множеств пунктов прибытия и отправления (2.1.5) выделим три непустых конечных подмножества X:

$$\mathbb{X} \triangleq \{x^0\} \cup \left(\bigcup_{i=1}^{n} M_i \right), \tag{2.1.6}$$

$$\mathbb{X}_{\text{in}} \triangleq \left(\bigcup_{i=1}^{n} \mathbb{M}_i^{(\text{in})} \right) \cup \{\varnothing\}, \tag{2.1.7}$$

$$\mathbb{X}_{\text{out}} \triangleq \{x^0\} \cup \left(\bigcup_{i=1}^{n} \mathbb{M}_i^{(\text{out})} \right); \tag{2.1.8}$$

ясно, что $\mathbb{X}_{\text{in}} \subset \mathbb{X}$ и $\mathbb{X}_{\text{out}} \subset \mathbb{X}$. Данные три множества позволят давать несколько более точные и наглядные определения; отдельно отметим, что в общем случае $\mathbb{X} \neq X$: распространен вариант когда X является плоскостью $\mathbb{R} \times \mathbb{R}$ с евклидовым расстоянием, между тем, мы работаем лишь с ее конечным подмножеством $\mathbb{X}$. «Пустое» множество $\{\varnothing\}$ в $\mathbb{X}_{\text{in}}$ понадобится для обозначения ситуации, когда *не важно*, каким был пункт прибытия (в мегаполис).

2.1.3 Решения и критерий качества

Наряду с допустимым маршрутом $\alpha \in \mathbb{A}$, в *обобщенной* задаче надлежит выбрать еще *трассу*, или траекторию, представленную в (2.1.3) набором УП $z_1, \ldots, z_n$, дополненным начальной УП $(\varnothing, x^0)$. Обозначим символом $\mathcal{T}$ множество всех кортежей, соблюдающих условие (2.1.4) и содержащих базу и терминал:

$$(z_i)_{i=0}^{n} : 0\,..\,n \to \mathbb{X}_{\text{in}} \times \mathbb{X}_{\text{out}}.$$

Определим теперь множество всех трасс, согласующихся с некоторым маршрутом α:

$$\mathcal{T}_\alpha \triangleq \left\{ (z_i)_{i=0}^{n} \in \mathcal{T} \;\middle|\; \big(z_0 = (\varnothing, x^0)\big) \wedge \big(z_t \in \mathbb{M}_{\alpha_t} \;\forall t \in 1\,..\,n\big) \right\}. \tag{2.1.9}$$

Перейдем к определению критерия качества. Напомним определение существенных списков-фильтров (см. раздел 1.1.2):

$$\mathcal{F} \triangleq \left\{ F \in \mathcal{P}\big(1\,..\,n\big) \;\middle|\; \forall i \in F \;\forall j \in 1\,..\,n \;\big(j \geqslant_P i\big) \to \big(j \in F\big) \right\};$$

здесь они интерпретируются как множества (индексов) *мегаполисов*, еще не посещенных агентом. Связь между фильтрами, отличающимися мощностью на 1, выражается операторами

$$\mathbf{E}[F] \equiv \operatorname{Max}[1\,..\,n \setminus F]; \tag{2.1.10}$$

$$\mathbf{I}[F] \equiv \operatorname{Min}[F]; \tag{2.1.11}$$

здесь и далее в главе 2 мы опускаем индексы $_{\text{F}}$ у этих операторов: выше оговаривалось, что в настоящей главе рассматриваем исключительно *попятный* вариант ДП.

Напомним, $\mathbf{E}[F]$ — оператор *продолжения* фильтра, определяемый как «множество таких индексов, что при добавлении любого из них к F вновь получается *фильтр* F', на 1 больший по мощности». В свою очередь, $\mathbf{I}[F]$ — оператор *входа* в фильтр F, «множество индексов, которыми может „начинаться“ фильтр F», что эквивалентно «множеству индексов, таких что при изъятии одного из F полученное множество будет *фильтром* на 1 меньшей мощности (возможно, пустым)»; см. теоремы 1.2,1.1.

Введем $n+1$ функцию стоимости

$$\mathbb{c}\colon \mathbb{X}_{\text{out}} \times \mathbb{X}_{\text{in}} \times \mathcal{F} \to \mathbb{R};\ c_i\colon \mathbb{M}_i \times \mathcal{F} \to \mathbb{R},\ i \in 1\,..\,n. \tag{2.1.12}$$

Функция $\mathbb{c}(\cdot)$ отвечает за *внешние* перемещения между мегаполисами, включая перемещение из базы в первый мегаполис маршрута, тогда как n «нумерованных» функций стоимости оценивают *внутренние работы* в каждом мегаполисе.

Для краткости можно «*пред*агрегировать» внешнее перемещение с внутренними работами с помощью функции $\mathfrak{c}\colon \mathbb{X}_{\text{out}} \times \mathbb{X}_{\text{in}} \times \mathbb{X}_{\text{out}} \times \mathcal{F} \to \mathbb{R}$, определяемой по правилу[6]

$$\mathfrak{c}(x, z, K) \triangleq \mathbb{c}(x, \operatorname{pr}_1(z), K) + c_l(z, K),$$

где l — индекс мегаполиса, которому принадлежат города z.

Теперь мы можем, наконец, ввести критерий качества для PSD-BGTSP-PC. Для $\alpha \in \mathbb{A}$ и $(z_i)_{i=0}^n \in \mathcal{T}^{(\alpha)}$ полагаем

$$\begin{aligned} \mathfrak{C}^{(\alpha)}\left[(z_i)_{i=0}^n\right] &\triangleq \max_{t\in 0..n-1}\left[\mathbb{c}\Big(\operatorname{pr}_2(z_t), \operatorname{pr}_1(z_{t+1}), \overline{(\alpha_i)_{i=t+1}^n}\Big) + c_{\alpha_{t+1}}\Big(z_{t+1}, \overline{(\alpha_i)_{i=t+1}^n}\Big)\right] \\ &= \max_{t\in 0..n-1}\left[\mathfrak{c}\Big(\operatorname{pr}_2(z_t), z_{t+1}, \overline{(\alpha_i)_{i=t+1}^n}\Big)\right]. \end{aligned} \tag{2.1.13}$$

Данный критерий представляет собой «минимаксное» (также «на узкие места», англ. bottleneck) агрегирование суммарной стоимости переезда из предыдущего мегаполиса в текущий и выполнения внутренних работ в текущем.

Такой критерий соответствует вопросу о наименьшей возможной *емкости аккумулятора* электромобиля или электрокара, которая позволит объехать все мегаполисы с выполнением внутренних работ, соблюдая условия предшествования. При аддитивном агрегировании, решение задачи описывало бы кратчайший обход. Между тем, одним из наиболее важных препятствий широкому применению электромобилей и электрокаров является сравнительно малая емкость доступных аккумуляторов. В частности, в литературе рассматривается постановка VRP (Vehicle Routing Problem, см. [1]) для электромобилей — с учетом поездок к станциям зарядки [145].

Итак, задача PSD-BGTSP-PC выглядит следующим образом:

$$\begin{aligned} &\mathfrak{C}^{(\alpha)}\left[(z_i)_{i=0}^n\right] = \max_{t\in 0..n-1}\left[\mathfrak{c}\Big(\operatorname{pr}_2(z_t), z_{t+1}, \overline{(\alpha_i)_{i=t+1}^n}\Big)\right]; \\ &\mathfrak{C}^{(\alpha)}\left[(z_i)_{i=0}^n\right] \to \min,\ \alpha \in \mathbb{A},\ (z_i)_{i=0}^n \in \mathcal{T}_\alpha. \end{aligned} \tag{PSD-BGTSP-PC}$$

Введем *значение* (экстремум) задачи:

$$V \triangleq \min_{\alpha\in\mathbb{A}} \min_{(z_i)_{i=0}^n \in \mathcal{T}_\alpha} \mathfrak{C}^{(\alpha)}\left[(z_i)_{i=0}^n\right] \in [0, \infty[; \tag{2.1.14}$$

экстремум достигается на непустом[7] множестве оптимальных допустимых решений (далее ДР); при этом ДР

$$(\alpha^*, (z_i^*)_{i=0}^n),\ \alpha^* \in \mathbb{A}, (z_i^*)_{i=0}^n \in \mathcal{T}_{\alpha^*}$$

оптимально в задаче (PSD-BGTSP-PC) если $\mathfrak{C}^{(\alpha^*)}\left[(z_i^*)_{i=0}^n\right] = V$.

[6] второй аргумент $\mathfrak{c}$ — *пара городов* $z \in \mathbb{X}_{\text{in}} \times \mathbb{X}_{\text{out}}$; точнее, $z \in \mathbb{M}_i$ для некоторого индекса i, описывающая перемещение в ходе внутренних работ

[7] для любого частичного порядка $<_P$ существует *линейное продолжение*, см. [122, Theorems 2.23,2.29] — допустимый маршрут; отношения $\mathbb{M}_i$ предполагаются непустыми, см. (2.1.2)

2.1.4 Расширение основной задачи. Пространство состояний динамического программирования

Решение методом динамического программирования состоит в погружении задачи в некоторое семейство сходных задач и установления связи между решениями задач, более простых чем исходная, выраженной в уравнении Беллмана (см., напр., [57, Ch. 9]). В построениях ниже мы будем следовать конструкциям [144, гл. 4], [61, 62] с точностью до перевода на теоретико-порядковый язык.

Напомним необходимые определения.

Множество допустимых относительно порядка $<_P$ маршрутов (1.2.4):

$$\mathbb{A} \triangleq \left\{ \alpha \in (\mathrm{bi})[1\,..\,n] \,\middle|\, \forall a, b \in 1\,..\,n \ \left(a <_P b\right) \to \left(\alpha_a^{-1} < \alpha_b^{-1}\right) \right\}.$$

Множество допустимых маршрутов по списку заданий-фильтру F, $F \in \mathcal{F}$, определяется аналогично таковому для списка-идеала,

$$\mathbb{A}_F \triangleq \left\{ \alpha \in (\mathrm{bi})[F] \,\middle|\, \forall a, b \in F \ \left(a <_P b\right) \to \left(\alpha_a^{-1} < \alpha_b^{-1}\right) \right\}; \tag{2.1.15}$$

Состояниями[8] ДП, или *подзадачами*, полагаем *упорядоченные пары* вида (x, F), где F — множество (индексов) мегаполисов, *еще не посещенных* агентом (список заданий) после выхода из города $x \in \mathbb{X}_{\mathrm{out}}$, принадлежащего некоторому мегаполису M_i, $i \notin F$. Множество всех состояний определяется следующим образом:

$$\mathcal{S} \triangleq \left\{ (x, F) \in \mathbb{X}_{\mathrm{out}} \times \mathcal{F} \;\middle|\; \left(\exists m \in \mathbf{E}[F] \mid x \in \mathbb{M}_m^{(\mathrm{out})}\right) \right\} \cup \left\{ \left(x^0, 1\,..\,n\right) \right\}; \tag{2.1.16}$$

стратифицируем его по мощности списков заданий, $\mathcal{S}_k \triangleq \{(x, K) \in \mathcal{S} \mid |K| = k\}$; каждый элемент этого разбиения будем называть *слоем* (пространства состояний). Будем говорить, что состояние $s' = (x', K')$ *покрывает* состояние $s = (x, K)$ если $(K \cup \{m\}) = K'$, где $x \in M_m$.

Как и в *необобщенной* задаче, можно выделить два особых слоя: состояния $\mathcal{S}_0$ с пустыми списками заданий задают *начальные условия*; $\mathcal{S}_n$ содержит *единственное* состояние $(x^0, 1\,..\,n)$, обозначающее *полную задачу* — выйдя из *базы* x^0, посетить *все* мегаполисы M_i, $i \in 1\,..\,n$.

Опишем теперь (допустимые) *решения* частичных задач, данных состояниями. Для произвольного состояния $(x, K) \in \mathcal{S}_{|K|}$, как и для *полной задачи*, решение описывается парой маршрут-трасса, где трасса согласована с маршрутом. Введем множество таких *частичных* трасс, согласованных с некоторым *частичным* маршрутом $\alpha \in \mathbb{A}_K$:

$$\mathcal{T}_\alpha(x, K) \triangleq \left\{ (z_i)_{i=0}^{|K|} \colon 0\,..\,|K| \to \mathbb{X}_{\mathrm{in}} \times \mathbb{X}_{\mathrm{out}} \,\middle|\, \left(z_0 = (\varnothing, x)\right) \wedge \left(z_t \in \mathbb{M}_{\alpha_t} \ \forall t \in 1\,..\,|K|\right) \right\}. \tag{2.1.17}$$

Теперь мы можем определить «укороченный» критерий качества: при $x \in \mathbb{X}_{\mathrm{out}}$, $K \in \mathcal{F}$, $\alpha \in \mathbb{A}_K$ и $(z_i)_{i=0}^{|K|} \in \mathcal{T}_\alpha(x, K)$,

$$\mathfrak{C}_K^{(\alpha)}\left[(z_i)_{i=0}^{|K|}\right] \triangleq \max_{t \in 0..|K|-1} \left[\mathfrak{c}\left(\mathrm{pr}_2(z_t), z_{t+1}, \overline{(\alpha_s)_{s=1}^{|K|}} \right) \right]. \tag{2.1.18}$$

Данный критерий позволяет нам поставить *укороченные* задачи (далее «подзадачи»):

$$\mathfrak{C}_K^{(\alpha)}\left[(z_i)_{i=0}^{|K|}\right] \to \min, \alpha \in \mathbb{A}_K, (z_i)_{i=0}^{|K|} \in \mathcal{T}_\alpha(x, K), \tag{2.1.19}$$

[8] также называются *позициями* [25], по аналогии с теорией оптимального управления

для которых определено *значение* (экстремум)

$$v(x,K) \triangleq \begin{cases} 0, & (x,K) \in \mathcal{S}_0; \\ \min\limits_{\alpha\in\mathbb{A}_K} \min\limits_{(z_i)_{i=0}^{|K|}\in\mathcal{T}_\alpha(x,K)} \mathfrak{C}_K^{(\alpha)}\big[(z_i)_{i=0}^{|K|}\big], & (x,K) \notin \mathcal{S}_0; \end{cases} \tag{2.1.20}$$

здесь первый случай, $v(x,\varnothing) \triangleq 0\ \forall x \in \mathbb{M}_m^{(\mathrm{out})}\ \forall\{m\} \in \mathcal{F}_1$, служит *начальным условием*. Отметим, что в *попятной* процедуре это *начальное* условие описывает *конец* пути. В настоящем, *незамкнутом* случае оно «разрешает» бесплатно закончить трассу в произвольном допустимом городе (принадлежащем мегаполису с индексом из $\mathcal{F}_1$). В случае *замкнутой* задачи вместо нуля фигурировала бы стоимость перемещения из последнего города в начало пути, на базу: $v(x,\varnothing) \triangleq \mathbb{c}(x,x^0,\varnothing)\ \forall x \in \mathbb{X}_{\mathrm{out}}$, аналогично тому как это сделано в прямой процедуре для *необобщенной* задачи в главе 1.

Определение оптимального решения аналогично таковому в полной задаче: при $x \in \mathbb{X}_{\mathrm{out}}$ и $K \in \mathcal{F}$, допустимое решение $\big(\alpha^*,(z_i^*)_{i=0}^{|K|}\big)$, $\alpha^* \in \mathbb{A}_K, (z_i^*)_{i=0}^{|K|} \in \mathcal{T}_\alpha(x,K)$, *оптимально*, если $\mathfrak{C}_K^{(\alpha^*)}\big[(z_i^*)_{i=0}^{|K|}\big] = v(x,K)$.

Определение частичных задач (2.1.19) и их значений (2.1.20) *согласуется* с определением *полной*: достаточно положить $x = x^0$ и $K = 1\,..\,n$. При этом $\alpha \in \mathbb{A}$ и по определению трасс в полной (2.1.9) и частичной (2.1.17) задаче, $\mathcal{T}_\alpha = \mathcal{T}_\alpha(x^0,1\,..\,n)$, откуда $V = v(x^0,1\,..\,n)$.

Следующая теорема отражает *корректность* ДП: *значение* любой подзадачи, определенной непосредственно (перебором допустимых решений) *совпадает* со значением, полученным из *принципа оптимальности Беллмана* (перебором по покрываемым состояниям); следствием этой теоремы для состояний, определенных выше, является *экономичность* описываемой процедуры ДП: доказано, что оптимальное решение полной задачи $(x^0,1\,..\,n)$ получается расчетом значений функции Беллмана на множестве состояний $\mathcal{S}$, которое содержит только *допустимые* в контексте условий предшествования состояния; таким образом, рассматриваются не все состояния, которые потребовались бы для решения задачи *без условий предшествования*.

Теорема 2.1. *Для* $(x,K) \in \mathcal{S} \setminus \mathcal{S}_0$ *справедлив* ***принцип оптимальности:***

$$v(x,K) = \min_{j\in\mathbf{I}(K)} \min_{z\in\mathbb{M}_j} \max\Big\{\mathbb{c}\big(x,\mathrm{pr}_1(z),K\big) + c_j\big(z,K\big);\, v\big(\mathrm{pr}_2(z),K\setminus\{j\}\big)\Big\}. \tag{BF}$$

2.1.5 Доказательство принципа оптимальности для PSD-BGTSP-PC

Настоящее доказательство было опубликовано в [61] и принадлежит А. Г. Ченцову; ниже мы приводим его в переводе на теоретико-порядковый язык с некоторыми изменениями и комментариями, в частности, об *абстрактном* его характере, аналогично [53].

У Беллмана [15] доказательство корректности для *задачи коммивояжера* уместилось в одном небольшом абзаце. Аналогичное доказательство для PSD-TSP-PC в главе 1 (теорема 1.3) заняло порядка одной страницы. Потребность рассматривать и учитывать в PSD-BGTSP-PC, кроме маршрута, еще и *трассу*, приводит к несколько большему объему.

Фиксируем $x \in \mathbb{X}_{\mathrm{out}}$ и $K \in \mathcal{F}$. Мощность K обозначим k: $k \triangleq |K| \in 1\,..\,n$. Правая часть (BF) будет часто упоминаться в нашем доказательстве, обозначим ее $(\mathrm{BF}^{\mathrm{r}})$.

Общий план доказательства состоит в том, чтобы доказать сначала $v(x,K) \leqslant (\mathrm{BF}^{\mathrm{r}})(x,K)$, а затем $v(x,K) \geqslant (\mathrm{BF}^{\mathrm{r}})(x,K)$. Мы не используем *принцип математической индукции*, однако некоторые аналогии с ним будут прослеживаться. В частности, мы начнем с простого до тривиальности случая $k = 1$, для которого можно сразу доказать равенство, и только после этого рассмотрим общий случай $k \in 2\,..\,n$.

Простой случай. Для случая $k = 1$ справедливость (BF) практически очевидна: пусть $K = \{j\}$ ($K \in \mathcal{F}_1$); тогда маршрут во множестве $\mathbb{A}_K = \big\{\alpha \in \{j\} \to \{j\}\big\}$ только один, тривиальный. Соответствующие трассы $\mathcal{T}_\alpha(x, K)$ предоставляют лишь возможность выбора из допустимых пар пунктов прибытия в и отправления из j:

$$\mathcal{T}_\alpha(x, K) = \Big\{(z_i)_{i=0}^1 \colon 0\,..\,1 \to \mathbb{X}_{\text{in}} \times \mathbb{X}_{\text{out}} \,\Big|\, \big(z_0 = (\varnothing, x)\big) \wedge \big(z_1 = z\big), z \in \mathbb{M}_j\Big\}.$$

Соответственно, определения *значения* состояния (x, K) (2.1.20) и функции Беллмана $(\mathrm{BF}^{\mathrm{r}})(x, K)$ *совпадают*.

Общий случай. Доказательство $v(x, K) \geqslant (\mathrm{BF}^{\mathrm{r}})(x, K)$**.** Итак, $k \in 2\,..\,n$. Согласно (BF), для построения $v(x, K)$ средствами $(\mathrm{BF}^{\mathrm{r}})$ используются агрегированные данные о значении критерия, полученные на предыдущих этапах построения, которые выражены в $v\big(\operatorname{pr}_2(z), K \setminus \{j\}\big)$ $\forall j \in \mathbf{I}[K]$ $\forall z \in \mathbb{M}_j$; несомненно, $|K \setminus \{j\}| = k - 1$ $\forall j \in \mathbf{I}[K]$. По определению *экстремумов* частичных задач (2.1.20),

$$v\big(\operatorname{pr}_2(z), K \setminus \{j\}\big) = \min_{\alpha \in \mathbb{A}_{K\setminus\{j\}}} \min_{(z_i)_{i=0}^{n-1} \in \mathcal{T}_\alpha(\operatorname{pr}_2(z), K\setminus\{j\})} \mathfrak{C}^{(\alpha)}_{K\setminus\{j\}}\big[(z_i)_{i=0}^{k-1}\big]. \tag{2.1.21}$$

Выберем допустимое решение $\big(\alpha^*, (z_i^*)_{i=0}^k\big)$, доставляющее подзадаче (x, K) *экстремум*, $v(x, K) = \mathfrak{C}^{(\alpha^*)}_K\big[(z_i^*)_{i=0}^k\big]$. Введем обозначение $Q \triangleq K \setminus \{\alpha_1^*\}$ и распишем определение критерия качества для $\big(\alpha^*, (z_i^*)_{i=0}^k\big)$, отделив[9] первый элемент:

$$\mathfrak{C}^{(\alpha^*)}_K\big[(z_i^*)_{i=0}^k\big] = \max\Big\{\mathfrak{c}(x, z_1^*, K); \max_{t \in 1..n-1}\Big[\mathfrak{c}\big(\operatorname{pr}_2(z_t^*), z_{t+1}^*, \overline{(\alpha_s^*)_{s=t+1}^k}\big)\Big]\Big\}.$$

Рассмотрим $v\big(\operatorname{pr}_2(z_1^*), Q\big)$. Будет ли «хвост» ДР $\big(\alpha^*, (z_i^*)_{i=0}^k\big)$, полученный удалением из α^* и $(z_i^*)_{i=0}^k$ первого элемента и уменьшением индексов на 1, доставлять этот экстремум? Будет ли он хотя бы ДР для соответствующей *подзадачи*? Начнем с «хвоста» α^*, который обозначим $\check{\alpha}$ и зададим правилом $\check{\alpha}_t = \alpha_{t+1}^*$ $\forall t \in 1\,..\,k-1$.

Очевидно, $\alpha_1^* \in \mathbf{I}[\overline{(\alpha_s^*)_{s=1}^k}]$ (иначе нарушается условие в правой части определения $\mathbb{A}_K$), откуда $K \setminus \{\alpha_1^*\} = Q \in \mathcal{F}_{k-1}$. Следовательно, условие в правой части определения $\mathbb{A}_Q$ выполнено и для $\check{\alpha}$,

$$\forall a, b \in K \setminus \{\alpha_1^*\}\ (a <_P b) \to (\alpha_a^{-1} < \alpha_b^{-1}),$$

как частный случай соответствующего условия в определении $\mathbb{A}_K$ для α^*; таким образом, $\check{\alpha} \in \mathbb{A}_Q$.

Легко видеть, что «хвост» $(z_i^*)_{i=0}^k$, определенный по правилу

$$(\check{z}_i)_{i=0}^{k-1} = \Big\{\big(\check{z}_0 \triangleq (\varnothing, \operatorname{pr}_2(z_1^*))\big) \wedge \big(\check{z}_i \triangleq z_{i+1}^*\ \forall i \in 1\,..\,k-1\big)\Big\},$$

принадлежит $\mathcal{T}_{\check{\alpha}}(\operatorname{pr}_2(z_1^*), Q)$.

Теперь подставим полученные «хвосты» в определение экстремума $v\big(\operatorname{pr}_2(z_1^*), Q\big)$ и применим свойства операции взятия наименьшего элемента:

$$\begin{aligned} v\big(\operatorname{pr}_2(z_1^*), Q\big) &= \min_{\alpha \in \mathbb{A}_Q} \min_{(z_i)_{i=0}^{n-1} \in \mathcal{T}_\alpha(\operatorname{pr}_2(z_1^*), Q)} \mathfrak{C}^{(\alpha)}_Q\Big[(z_i)_{i=0}^{n-1}\Big] \\ &\leqslant \mathfrak{C}^{(\check{\alpha})}_Q\Big[(\check{z}_i)_{i=0}^{k-1}\Big] = \max_{t \in 0..n-2}\Big[\mathfrak{c}\big(\operatorname{pr}_2(\check{z}_t), \check{z}_{t+1}, \overline{(\check{\alpha}_s)_{s=t+1}^{k-1}}\big)\Big]. \end{aligned}$$

[9] Данное действие не изменит значение критерия, потому что операция *взятия максимума* ассоциативна, в частности, *право*ассоциативна

Вспомнив, что $\check{\alpha}$ есть «хвост» α^* и заменив индексы, получим

$$\overline{(\check{\alpha}_s)_{s=t+1}^{k-1}} = \overline{(\alpha^*_{s+1})_{s+1=t+1}^{k-1}} = \overline{(\alpha^*_l)_{l=t+2}^{k}};$$

поступив аналогично с «хвостом» $(z^*_i)_{i=0}^n$, увидим:

$$\mathfrak{C}_Q^{(\check{\alpha})}\Big[(\check{z}_i)_{i=0}^{n-1}\Big] = \max_{t\in 0..k-2}\Big[\mathfrak{c}\big(\operatorname{pr}_2(\check{z}_t), \check{z}_{t+1}, \overline{(\alpha^*_l)_{l=t+2}^{k}}\big)\Big] = \max_{\theta\in 1..k-1}\Big[\mathfrak{c}\big(\operatorname{pr}_2(z^*_\theta), z^*_{\theta+1}, \overline{(\alpha^*_l)_{l=\theta+1}^{k}}\big)\Big],$$

откуда следует оценка

$$v\big(\operatorname{pr}_2(z^*_1), Q\big) \leqslant \max_{\theta\in 1..k-1}\Big[\mathfrak{c}\big(\operatorname{pr}_2(z^*_\theta), z^*_{\theta+1}, \overline{(\alpha^*_l)_{l=\theta+1}^{k}}\big)\Big].$$

Поскольку $\alpha^*(1)$ и z^*_1 определены *корректно* — $\alpha^*(1) \in \mathbf{I}[K]$ и $z^*_1 \in \mathbb{M}_{\alpha^*(1)}$, применив определение операции взятия наименьшего элемента, подставим эту оценку $v\big(\operatorname{pr}_2(z^*_1), Q\big)$ «внутрь» (BF^{r}), сменив «немую» индексную переменную θ на более привычную t и учтя ассоциативность[10] и монотонное *неубывание* операции взятия наибольшего элемента по *второму* аргументу, получим

$$\begin{aligned}
(\mathrm{BF}^{\mathrm{r}})(x,K) &= \min_{j\in\mathbf{I}(K)}\min_{z\in\mathbb{M}_j}\max\Big\{\mathfrak{c}(x,z,K); v(\operatorname{pr}_2(z), K\setminus\{j\})\Big\}\\
&\leqslant \max\Big\{\mathfrak{c}(x,z^*_1,K); v\big(\operatorname{pr}_2(z^*_1), Q\big)\Big\}\\
&= \max\Big\{\mathfrak{c}(x,z^*_1,K); \max_{t\in 1..n-1}\Big[\mathfrak{c}\big(\operatorname{pr}_2(z^*_t), z^*_{t+1}, \overline{(\alpha^*_l)_{l=t+1}^{k}}\big)\Big]\Big\}\\
&= \mathfrak{C}_K^{(\alpha^*)}\Big[(z^*_t)_{t=0}^n\Big] = v(x,K).
\end{aligned}$$

Итак, мы *доказали* желаемое неравенство $v(x,K) \geqslant (\mathrm{BF}^{\mathrm{r}})(x,K)$. □.

Общий случай. Доказательство $v(x,K) \leqslant (\mathrm{BF}^{\mathrm{r}})(x,K)$. В предыдущем доказательстве мы начали с фиксации ДР, доставлявшего экстремум $v(x,K)$. Настоящее начнем с выбора $q \in \mathbf{I}[K]$ и $\mathbb{z} \in \mathbb{M}_q$, доставляющих минимум в $(\mathrm{BF}^{\mathrm{r}})(x,K)$; также введем обозначение $Q \triangleq K\setminus q$:

$$\max\Big\{\mathfrak{c}(x,\mathbb{z},K); v\big(\operatorname{pr}_2(\mathbb{z}), Q\big)\Big\} = \min_{j\in\mathbf{I}(K)}\min_{z\in\mathbb{M}_j}\max\Big\{\mathfrak{c}(x,\mathbb{z},K); v\big(\operatorname{pr}_2(z), K\setminus\{j\}\big)\Big\}. \tag{2.1.22}$$

Теперь фиксируем ДР $\big(\beta, (h_i)_{i=0}^{k-1}\big)$, $\beta \in \mathbb{A}_Q$, $(h_i)_{i=0}^{n-1} \in \mathcal{T}_\beta(\operatorname{pr}_2(\mathbb{z}), Q)$, доставляющее экстремум $v(\operatorname{pr}_2(\mathbb{z}), Q)$:

$$v(\operatorname{pr}_2(\mathbb{z}), Q) = \mathfrak{C}_Q^{(\beta)}\Big[(h_i)_{i=0}^{n-1}\Big] = \max_{t\in 0..n-2}\Big[\mathfrak{c}\big(\operatorname{pr}_2(h_t), h_{t+1}, \overline{(\beta_s)_{s=t+1}^{k-1}}\big)\Big]. \tag{2.1.23}$$

Теперь «присоединим» к $\big(\beta, (h_i)_{i=0}^{k-1}\big)$ элементы q и $\mathbb{z}$, обозначив полученный маршрут $\hat{\beta}$ и трассу $(\hat{h}_i)_{i=0}^k$:

$$\hat{\beta}\colon 1..k \to K, \qquad \big(\hat{\beta}_1 = q\big) \wedge \big(\hat{\beta}_i = \beta_{i-1}\ \forall j \in 2..k\big); \tag{2.1.24}$$

$$(\hat{h}_i)_{i=0}^k\colon 0..k \to \mathbb{X}_{\mathrm{in}} \times \mathbb{X}_{\mathrm{out}}, \quad \big(\hat{h}_0 = (\varnothing, x)\big) \wedge \big(\hat{h}_1 = \mathbb{z}\big) \wedge \big(\hat{h}_i = h_{i-1}\ \forall i \in 3..k\big). \tag{2.1.25}$$

Покажем, что новые объекты определены корректно, то есть, $\hat{\beta} \in \mathbb{A}_K$ и $(\hat{h}_i)_{i=0}^k \in \mathcal{T}_{\hat{\beta}}(x,K)$.

[10] здесь достаточно *право*ассоциативности

Корректность $\hat\beta$**.** Сначала покажем, что $\hat\beta$ является биекцией K. Маршрут $\beta\colon 1\,..\,k-1 \to Q$ является биекцией $Q = K \setminus q$. Очевидно, от сдвига индексов биективность не нарушилась: ограничение $\hat\beta$ на $2\,..\,k$ биективно отображает эти индексы на Q. Поскольку $q \notin Q$, определение $\hat\beta_1 = q$ также не нарушает биективности: q по определению $\hat\beta$ не может содержаться в $\{\hat\beta(t):\ t \in 2\,..\,k\}$. Теперь покажем *допустимость* $\hat\beta$ в качестве маршрута по K, $\hat\beta \in \mathbb{A}_K$.

Выпишем определение допустимости для $\hat\beta$ как маршрута по K (см. (2.1.15)): $\hat\beta \in \mathbb{A}_K$, если

$$\forall a, b \in K\ \big(a <_P b\big) \to \big(\hat\beta_a^{-1} < \hat\beta_b^{-1}\big);$$

отметим, что оно выполнено $\forall a, b \in Q$, поскольку $\beta \in \mathbb{A}_Q$; то есть, достаточно проверить его для $a = q$ и произвольных $b \in Q$ (случай $a = b$ тривиален). По выбору, $q \in \mathbf{I}[K]$; согласно *характеризации* (2.1.11), $q \in \mathrm{Min}[K]$, следовательно, $\forall b \in Q\ q <_P b$. При этом $\hat\beta_q^{-1} = 1$, то есть, для $a = q$ и произвольного $b \in Q$, консеквент импликации $\big(a <_P b\big) \to \big(\hat\beta_a^{-1} < \hat\beta_b^{-1}\big)$ истинен *всегда*, что завершает проверку $\hat\beta \in \mathbb{A}_K$.

Корректность $(\hat h_i)_{i=0}^k$**.** Рассмотрим $(h_i)_{i=0}^{k-1}$. По выбору, $(h_i)_{i=0}^{k-1} \in \mathcal{T}_\beta(\mathrm{pr}_2(\mathbb{z}), Q)$, следовательно, $h_0 = (\varnothing, \mathrm{pr}_2(\mathbb{z}))$ и $h_t \in \mathbb{M}_{\beta(t)}\ \forall t \in 1\,..\,k-1$; отметим, что $\mathbb{M}_{\beta(t)} = \mathbb{M}_{\hat\beta(t+1)}\ \forall t \in 1\,..\,k-1$: «хвост» $(\hat h_i)_{i=0}^k$ — трасса $(h_i)_{i=1}^{k-1}$ — определен *корректно* (с соблюдением условия допустимости пунктов прибытия и отправления, задаваемых отношениями (2.1.2) в том виде, в котором они входят в (2.1.17)); он «переносится» в $(\hat h_i)_{i=0}^k$ по правилу $\hat h_t = h_{t-1}\ \forall t \in 2\,..\,k-1$. Однако первый элемент $\hat h_1$ не может быть аналогичным образом заимствован из $h_0 = \big(\varnothing, \mathrm{pr}_2(\mathbb{z})\big)$: последний не несет информации о *пункте прибытия* в M_q, что вполне естественно, так как для решения задачи $(\mathrm{pr}_2(\mathbb{z}), Q)$ сведения о пункте *прибытия* в M_q уже не релевантны. Следовательно, стоит положить $\hat h_1 \triangleq \big(\mathrm{pr}_1(\mathbb{z}), \mathrm{pr}_2(\mathbb{z})\big)$; нулевым же элементом в $(\hat h_i)_{i=0}^k$ будет быть УП $(\varnothing, x)$ — это соответствует определению (2.1.17) для рассматриваемого состояния (x, K). Таким образом, мы получаем кортеж

$$\begin{aligned}&(\hat h_i)_{i=0}^k\colon \big(\hat h_0 = (\varnothing, x)\big) \wedge \big(\hat h_1 = \mathbb{z}\big) \wedge \big(\hat h_t = h_{t-1}\ \forall t \in 2\,..\,k\big);\\ &(\hat h_i)_{i=0}^k \in \mathcal{T}_{\hat\beta}(x, K).\end{aligned} \tag{2.1.26}$$

Продолжение доказательства. Итак, мы выяснили, что УП $\big(\hat\beta, (\hat h_i)_{i=0}^k\big)$ *является* допустимым решением рассматриваемой частичной задачи (x, K), откуда следует $v(x, K) \leqslant \mathfrak{C}_K^{(\beta)}\big[(\hat h_i)_{i=0}^k\big]$. По определению критерия качества решения,

$$\mathfrak{C}_K^{(\hat\beta)}\big[(\hat h_i)_{i=0}^n\big] = \max_{t\in 0..n-1}\Big[\mathfrak{c}\big(\mathrm{pr}_2(\hat h_t), \hat h_{t+1}, \overline{(\hat\beta_s)_{s=t+1}^k}\big)\Big].$$

Поскольку операция взятия максимума *ассоциативна*[11], равенство сохранится, если «отделить» первый элемент выражения выше, соответствующий $t = 0$:

$$\mathfrak{C}_K^{(\hat\beta)}\big[(\hat h_i)_{i=0}^n\big] = \max\Big\{\mathfrak{c}\big(\mathrm{pr}_2(\hat h_0), \hat h_1, \overline{(\hat\beta_s)_{s=1}^k}\big);\ \max_{t\in 1..k-1}\Big[\mathfrak{c}\big(\mathrm{pr}_2(\hat h_t), \hat h_{t+1}, \overline{(\hat\beta_s)_{s=t+1}^k}\big)\Big]\Big\}. \tag{2.1.27}$$

С учетом определений $\hat\beta$ (2.1.24) и $(\hat h_i)_{i=0}^n$ (2.1.26),

$$\mathfrak{c}\big(\mathrm{pr}_2(\hat h_0), \hat h_1, \overline{(\hat\beta_s)_{s=1}^k}\big) = \mathfrak{c}\big(x, z, K\big). \tag{2.1.28}$$

[11] здесь достаточно *право*ассоциативности

Покажем, что в правой части (2.1.27) «хвост» $\max\limits_{t\in 1..n-1}\big[\cdots\big]$ на самом деле равен $v\big(\mathrm{pr}_2(\mathbb{z}),Q\big)$. Рассмотрим

$$\max_{t\in 1..k-1}\Big[\mathfrak{c}\big(\mathrm{pr}_2(\hat{h}_t),\hat{h}_{t+1},\overline{(\hat{\beta}_s)_{s=t+1}^{k}}\big)\Big]. \tag{2.1.29}$$

Все элементы $\hat{\beta}_s$ $\forall s\in 2..k$ являются элементами β по правилу $\hat{\beta}_s=\beta_{s-1}$. Так, вместо $\overline{(\hat{\beta}_s)_{s=t+1}^{k}}$, для всех $t\in 1..k-1$, мы можем написать $\overline{(\beta_s)_{s=t}^{k-1}}$ и, внутри $\mathfrak{c}\big(\mathrm{pr}_2(\hat{h}_t),\hat{h}_{t+1},\overline{(\hat{\beta}_s)_{s=t+1}^{k}}\big)$, внутренние работы $c_{\hat{\beta}_{t+1}}(\cdot,\cdot)$ заменяются на $c_{\beta_t}(\cdot,\cdot)$. Аналогично можно перейти от $(\hat{h}_i)_{i=0}^{k}$ к $(h_i)_{i=0}^{k-1}$, ведь, начиная со второго элемента $\hat{h}_2$, все оставшиеся элементы этого кортежа совпадают с элементами $(h_i)_{i=0}^{k-1}$ по определению (2.1.26): $\hat{h}_t=h_{t-1}$ $\forall t\in 2..k$. Что касается $\hat{h}_1$, в данном выражении присутствует лишь его вторая компонента $\mathrm{pr}_2(\hat{h}_1)=\mathrm{pr}_2(\mathbb{z})$, которая совпадает со второй компонентой нулевого элемента h_0, поскольку $(h_i)_{i=0}^{k-1}\in\mathcal{T}_\beta(\mathrm{pr}_2(\mathbb{z}),Q)$. Таким образом, в (2.1.29) замена индекса $t\in 1..k-1$ на $\theta\in 0..k-2$ позволяет вернуться к записи через β и $(h_i)_{i=0}^{k-1}$:

$$\max_{t\in 1..n-1}\Big[\mathfrak{c}\big(\mathrm{pr}_2(\hat{h}_t),\hat{h}_{t+1},\overline{(\hat{\beta}_s)_{s=t+1}^{k}}\big)\Big]=\max_{\theta\in 0..n-2}\Big[\mathfrak{c}\big(\mathrm{pr}_2(h_\theta),h_{\theta+1},\overline{(\beta_s)_{s=\theta+1}^{k-1}}\big)\Big]. \tag{2.1.30}$$

Последнее есть ни что иное как $\mathfrak{C}_Q^{\beta}\big[(h_\theta)_{\theta=0}^{k-1}\big]=v\big(\mathrm{pr}_2(\mathbb{z}),Q\big)$ (2.1.23). С учетом этого факта и преобразования (2.1.28), получаем

$$v(x,K)\leqslant\mathfrak{C}_K^{(\hat{\beta})}\big[(\hat{h}_i)_{i=0}^{k}\big]=\max\Big\{\mathfrak{c}\big(x,z,K\big);v\big(\mathrm{pr}_2(\mathbb{z}),Q\big)\Big\}.$$

Поскольку $q\in\mathbf{I}[K]$, а $\mathbb{z}\in\mathbb{M}_q$ доставляют минимум $(\mathrm{BF}^{\mathrm{r}})(x,K)$ по выбору (см. (2.1.22)), получаем

$$v(x,K)\leqslant\min_{j\in\mathbf{I}[K]}\min_{z\in\mathbb{M}_j}\max\Big\{\mathbb{c}\big(x,\mathrm{pr}_1(z),K\big)+c_q\big(z,K\big);v\big(\mathrm{pr}_2(z),K\setminus j\big)\Big\},$$

то есть, $v(x,K)\leqslant(\mathrm{BF}^{\mathrm{r}})(x,K)$, что и требовалось доказать.

Доказав неравенства $v(x,K)\leqslant(\mathrm{BF}^{\mathrm{r}})(x,K)$ и $v(x,K)\geqslant(\mathrm{BF}^{\mathrm{r}})$, мы делаем вывод, что $v(x,K)=(\mathrm{BF}^{\mathrm{r}})$, и (BF) выполняется всегда. Теорема 2.1 доказана. □

Отметим, что в процессе доказательства из всех свойств операции взятия наибольшего элемента $(\max\{a,b\})$ в критериях качества для полной задачи (2.1.13) и подзадач (2.1.18), мы воспользовались лишь ее *право*ассоциативностью и *неубывании* по *второму* аргументу[12]. Следовательно, данное доказательство справедливо и для других способов агрегирования затрат, в которых операция *агрегирования* стоимости затрат также ассоциативна и не убывает по обоим аргументам, в частности, для аддитивного агрегирования затрат с внутренними работами и зависимостью от списка невыполненных заданий, ср., напр., [146]. Такая *абстрактность* по операции агрегирования впервые была предложена в [53]; отличие от настоящего доказательства заключается в отсутствии описания *внутренних работ* (то есть, нет прямой применимости к CTSP) и рассмотрении задачи на *максимум*, а не на минимум; кроме того, в [53] условия предшествования описывались на языке *фундированных* отношений, а не частичных порядков.

Процесс решения (PSD)-(G)TSP-PC динамическим программированием в итоге выглядит следующим образом:

1. Сгенерировать $\mathcal{S}$ — рекуррентно, начиная с $\mathcal{S}_0$, с помощью $\mathbf{E}$

[12] в последней цепочке выражений доказательства $v(x,K)\geqslant(\mathrm{BF}^{\mathrm{r}})(x,K)$

2. Последовательно рассчитать ограничения v на слои пространства состояний, вплоть до *экстремума* $v|_{\mathcal{S}_n}(x^0, 1\,..\,n) = V$,

$$v|_{\mathcal{S}_0} \longrightarrow v|_{\mathcal{S}_1} \longrightarrow \ldots \longrightarrow v|_{\mathcal{S}_n} \Rightarrow V.$$

3. По V и v восстановить *оптимальное решение*, аналогично разделу 1.3.1

2.1.6 Подход к параллельной реализации точного динамического программирования

Для параллельных систем с *общей* памятью идея реализации ниже впервые представлена в [105]: на основе API параллельных вычислений с общей памятью OpenMP (в реализации для C++) предложен параллельный алгоритм решения GTSP-PC (аддитивное агрегирование затрат, нет зависимости от списка заданий); аналогичная идея параллелизации, реализованная на *потоках* (threads) C#, для той же задачи, описана в [147]. Реализация этой схемы будет подробно разобрана ниже, на примере (PSD)-BGTSP-PC [62]; суть этой схемы заключается в распределении «заданий» по вычислению $v[\mathcal{S}_k]$ и генерации $\mathcal{S}_{k+1}$ между независимыми вычислителями, обладающими доступом к общей памяти, содержащей значения v и сами состояния $\mathcal{S}$. Далее мы будем называть эту схему «горизонтальной».

Для параллелизации по схеме с *распределенной* памятью также может применяться «вертикальная» схема *независимых вычислений*, предложенная в [148]; эта схема была реализована в [136] для TSP-PC, в комбинации с описанной выше *горизонтальной* схемой (на каждом индивидуальном вычислительном узле), в эксперименте наблюдалось *линейное* масштабирование (расчет на многих узлах по сравнению с расчетом на одном узле).

Также упомянем параллелизацию на основе «встречной» схемы ДП, впервые предложенной для TSP-PC в [52]: она состоит в одновременном решении задачи в двух направлениях ДП (прямом и попятном) и последующем «сращивании» оптимальных решений для «взаимодополняющих» состояний; решения в прямом и попятном направлениях не зависят друг от друга, соответственно, возможна параллельная реализация данной схемы и для систем с *распределенной* памятью [137]; также подобная процедура рассматривалась для *симметричной* замкнутой TSP (классический вариант TSP) [149].

Рассмотрим подробнее *горизонтальную* схему. Напомним, для каждого $i \in 1\,..\,n$, для генерации *следующего* слоя пространства состояний $\mathcal{S}_{i+1}$ и расчета $v[\mathcal{S}_{i+1}]$ достаточно знать, соответственно, F_{i+1} и $v[\mathcal{S}_i]$. Распределение работы по независимым вычислителям с общей памятью (средствами OpenMP) происходит на шаге обработки *текущего* множества существенных списков $\mathcal{F}_i$. Каждый существенный список $F \in \mathcal{I}$ заданий можно обработать — расчитать существенное продолжение $\mathbf{E}[F]$ и фильтры $F' \in \mathcal{F} \mid F' = F \cup \{x\}, x \in \mathbf{E}[F]$, получить связанные с ним состояния $(x, F) \in \mathcal{S}_i \mid x \in \mathbf{E}[F]$, рассчитать на них v — *независимо* от остальных, таким образом, в самих расчетах отсутствует «состояние гонки» (race conditions); некоторая синхронизация, правда, в любом случае потребуется для записи как полученных состояний, так и идеалов. В частности, возможно, что некий фильтр $F^\sharp \in \mathcal{F}_{i+1}$ будет порожден как из $F_1 \in \mathcal{F}_i$, так и некоторого не совпадающего с ним $F_2 \in \mathcal{F}_i$. Для синхронизации и избежания гонки используется директива `omp critical`, запрещающая одновременное исполнение кода несколькими нитями OpenMP, в область действия которой заносится операция записи свежеполученных фильтров в $\mathcal{F}_{i+1}$, состояний $\mathcal{S}_{i+1}$ и значений $v[\mathcal{S}_{i+1}]$.

Вопросы реализации. Наша основная структура данных — вложенные *хеш-таблицы* (см. [129, § III.11]), содержащие информацию о состояниях: в первом контейнере ключом

служит существенный список заданий F, представленный целым числом[13] или массивом бит (bitset[14]), которому, через посредство хеш-функции[15], сопоставляется еще одна хеш-таблица, где ключом служит уже *город* x — база состояния (x, F), которому сопоставляется *значение* $v(x, F)$, представляемое числом *с плавающей запятой* (floating point). Тип этой структуры данных в C++ выглядит следующим образом:

```
std::vector<std::unordered_map<uint32_t,
 std::unordered_map <uint16_t, float>>>,
```

где внешний массив `std::vector` хранит отдельные слои пространства состояний, а `std::uintXX_t` — целые `XX`-битные числа без знака. Таким образом, чтобы получить доступ к значению $v(x, F)$ нужно трижды записать оператор взятия элемента по индексу (subscript operator): `[|F|][F][x]`.

Вложенная структура с разбивкой по слоям позволяет сэкономить память за счет того что фильтры *не повторяются*: для всех состояний, соответствующих одному фильтру F, максимальное количество которых wm, где w — ширина условий предшествования P, а m — максимальное количество городов в мегаполисе, фильтр записывается единственный раз — в отличие от *плоской* структуры, описанной, например, в [78]. Отметим, что для необобщенных задач с небольшим количеством городов (до 64) плоская структура может оказаться выгоднее: для связи фильтра с соответствующими ему состояниями используется указатель, длина которого в 64-битных системах составляет те же 64 бита, соответственно, при описании состояния может оказаться дешевле лишний раз указать фильтр в явном виде.

Выбор *хеш-таблицы* в качестве базовой структуры связан с необходимостью в быстром доступе к элементам (в хеш-таблице это «амортизированная константа» — постоянное количество операций в среднем [129, § III.11]) при отсутствии потребности в оптимизации по удалению или перестановке элементов. Также был опробован *ассоциативный массив* `std::map`, обеспечивающий доступ к элементам за логарифм от их количества, но предварительные испытания показали рост общего времени работы алгоритма примерно в *четыре* раза, соответственно, был сделан выбор в пользу хеш-таблицы `std::unordered_map`.

Поскольку *хеш-таблицы* не обеспечивают доступ к элементам за *постоянное* время (только постоянное в среднем), не удалось применить наиболее характерную для OpenMP директиву `parallel for`, что привело к реализации на основе введенных в OpenMP 3.0 конструкций `task` — индивидуальных заданий. Ниже представлен псевдокод параллельного алгоритма с директивами OpenMP.

[13] наличие или отсутствие в нем некоторого города кодируется, соответственно, нулем или единицей в нужном двоичном разряде

[14] эта структура данных обеспечивает быстрый доступ к каждому биту и произвольное (фиксируемое перед компиляцией) максимальное количество битов, в отличие от встроенных целых типов данных, размер которых фиксирован

[15] используются стандартные, встроенные в реализацию C++; отметим, что для существенных списков-идеалов также есть также особые, например, «адресация по меткам» из [150], см. подробнее в [22, 23]

Алгоритм 3 Параллельный алгоритм с директивами OpenMP

```
1:  for all l ∈ 1..n−1 do
2:      #pragma omp parallel default (shared)
3:      #pragma omp single nowait
4:      for all K ∈ F_l do
5:          #pragma omp task untied firstprivate(K)
6:          Вычислить E[K]
7:          for all j ∈ E[K] do
8:              #pragma omp critical (expand)
9:              Добавить K ∪ {j} в F_{l+1}
10:             for all x ∈ M_j^(out) do
11:                 Вычислить v(x, K)
12:                 #pragma omp critical (costwrite)
13:                 foo[|K|][K][x]:=v(x, K)
```

Результаты применения данного алгоритма смотрите в разделе 2.1.8.

2.1.7 Эвристика усеченного динамического программирования

Данная эвристика впервые была предложена и применена для TD-TSP [78]; русскоязычное название предложено автором диссертации. Актуальное обсуждение этой техники представлено в [126], где предлагается использование данной эвристики как общего метода решения различного рода маршрутных задач (Vehicle Routing Problem, подробнее см. [1]). Суть эвристики состоит в том, чтобы на каждом слое пространства состояний ДП оставлять H, $H \in \mathbb{N}$, *наилучших* состояний по значению на них функции Беллмана (BF); параметр H мы далее будем называть *параметром глубины*. Когда H превосходит мощность самого населенного слоя, эвристика вырождается в обыкновенную, точную процедуру ДП, описанную выше. С другой стороны, при $H = 1$, эвристика приближается к обыкновенному *жадному* алгоритму. В главе 1 эта эвристика рассматривалась для *необобщенной* задачи (PSD-TSP-PC); ниже, на материале [62], мы рассматриваем ее в применении к *обобщенной* задаче (PSD-BGTSP-PC).

Здесь и далее все «усеченные» версии элементов точной процедуры ДП отмечены тильдой ~ над соответствующими символами. Рассмотрим усеченное уравнение Беллмана.

$$\begin{cases} \widetilde{v}(x, \varnothing) = 0, & (x, \varnothing) \in \widetilde{\mathcal{S}}_0 = \mathcal{S}_0; \\ \widetilde{v}(x, K) = \min\limits_{j \in \widetilde{\mathbf{I}}(K)} \min\limits_{z \in \widetilde{\mathbb{M}}_j} \max\Big\{ \mathfrak{c}(x, z, K);\ \widetilde{v}\big(\mathrm{pr}_2(z), K \setminus \{j\}\big) \Big\}, & (x, K) \in \widetilde{\mathcal{S}} \setminus \widetilde{\mathcal{S}}_0; \end{cases} \qquad (\widetilde{BF})$$

$$\widetilde{v}|_{\widetilde{\mathcal{S}}_0 = \mathcal{S}_0} \longrightarrow \widetilde{v}|_{\widetilde{\mathcal{S}}_1} \longrightarrow \ldots \longrightarrow \widetilde{v}|_{\widetilde{\mathcal{S}}_{n-1}} \longrightarrow \widetilde{v}|_{\widetilde{\mathcal{S}}_n = \mathcal{S}_n} \Rightarrow \widetilde{V}.$$

Основное отличие между (BF) и $(\widetilde{BF})$ заключается в ограничении $\mathbf{I}$ и $\mathbb{M}_j$: в точной процедуре ДП для каждого слоя $\mathcal{S}_1, \ldots, \mathcal{S}_n$ гарантируется существование $v(\mathrm{pr}_2(z), K \setminus \{j\})$ для всех $j \in \mathbf{I}[K]$ и $z \in \mathbb{M}_j$, минимизация по которым доставляет $v(x, K)$, усеченная же процедура этой гарантии существования не дает: вполне возможно, что некоторое состояние $\big(\bar{l}, K \setminus \{\bar{l}\}\big) \in \mathcal{S}_{s-1}$ *не оказалось* среди H наилучших, отобранных в $\widetilde{\mathcal{S}}_{s-1} \subset \mathcal{S}_{s-1}$; его (эвристическое) значение $\widetilde{v}\big(\bar{l}, K \setminus \{\bar{l}\}\big)$ не было сохранено и не может быть использовано в вычислении $\widetilde{v}(x, K)$. Отсюда проистекает потребность формально ввести $\widetilde{\mathbf{I}}[K] \subset \mathbf{I}[K]$ и $\widetilde{\mathbb{M}}_j \subset \mathbb{M}_j$, сохраняющие только такие $j \in \mathbf{I}[K]$ и $(z_{\text{in}}, z_{\text{out}}) \in \mathbb{M}_j$, что состояния $\big(z_{\text{out}}, K \setminus \{j\}\big)$ попадают в H лучших, составляющих $\widetilde{\mathcal{S}}_{s-1}$. Поскольку в усеченном уравнении Беллмана

минимизация производится по меньшим (по включению) множествам, очевидно, эвристическое значение состояния не может быть лучше точного,

$$v(x, K) \leqslant \widetilde{v}(x, K)\ \forall (x, K) \in \widetilde{\mathcal{S}}.$$

Таким образом, ($\widetilde{BF}$) доставляет *верхнюю* границу $\widetilde{V}$ значения V. Ниже представлен алгоритм усеченного ДП.

Алгоритм 4 Псевдокод *усеченного* ДП для (PSD-)BGTSP-PC. H — параметр эвристики, количество *сохраняемых* состояний

Инициализация: $\widetilde{\mathcal{F}}_0 = \mathcal{F}_0$, $\widetilde{\mathcal{S}}_0 = \mathcal{S}_0$, $\widetilde{\mathcal{F}}_1 = \mathcal{F}_1$.
▷ Мощность $\widetilde{\mathcal{S}}_0$ не ограничивается H, поскольку *все*
▷ состояния $s \in \mathcal{S}_0$ имеют равное, *нулевое* значение.
for all $l \in 1\,..\,n-1$ **do**
  for all $K \in \widetilde{\mathcal{F}}_l$ **do** ▷ для всех *сохраненных* на шаге l
    Вычислить $\mathbf{E}[K]$
    for all $j \in \mathbf{E}[I]$ **do**
      for all $x_{\text{out}} \in \mathbb{M}_j^{(\text{out})}$ **do**
        Вычислить $\widetilde{v}(x_{\text{out}}, K)$ ▷ $\widetilde{v}(x_{\text{out}}, K) \geqslant v(x_{\text{out}}, K)$
        if $|\widetilde{\mathcal{S}}_l| < H$ **then**
          Добавить (x_{out}, K) в $\widetilde{\mathcal{S}}_l$
        else ▷ $|\widetilde{\mathcal{S}}_l| = H$,
          Пусть $(y_{\text{out}}, K') = \operatorname{argmax}_{s \in \widetilde{\mathcal{S}}_l} \widetilde{v}(s)$
          if $\widetilde{v}(y_{\text{out}}, K') > \widetilde{v}(x_{\text{out}}, K)$ **then**
            Удалить (y_{out}, K') из $\widetilde{\mathcal{S}}_l$
            Добавить (x_{out}, K) в $\widetilde{\mathcal{S}}_l$
  for all $(x_{\text{out}}, K) \in \widetilde{\mathcal{S}}_l$ **do**
    Найти $m \in \mathbf{E}[K] : x \in \mathbb{M}_m^{(\text{out})}$
    Добавить $K \cup \{m\}$ в $\widetilde{\mathcal{F}}_{l+1}$
Вычислить $\widetilde{v}(0, 1\,..\,n)$
Восстановить решение по $\widetilde{v}$.

Процедура восстановления маршрута и трассы по *усеченной* функции Беллмана $\widetilde{v}$ отличается от аналогичной процедуры для точного ДП тем же, чем ($\widetilde{BF}$) отличается от (BF), поэтому мы опустим ее подробное описание.

Оценка временной сложности эвристики усеченного динамического программирования

Для оценки сложности нам понадобятся некоторые дополнительные параметры — введем их. Напомним, что число мегаполисов обозначено n, плюс есть еще *база* x^0. Пусть в каждом мегаполисе не более m городов. Символом t_{BF} обозначим время, за которое можно рассчитать суммарную стоимость внешних перемещений $\mathbb{c}\big(x, \operatorname{pr}_1(z)\big)$ и внутренних работ $c_i\big(\operatorname{pr}_1(z), \operatorname{pr}_2(z)\big)$; мы будем считать t_{BF} постоянным, сославшись на возможность рассчитать все стоимости внутренних работ заранее; будет также считать, что время на вычисление наибольшей из двух величин также заложено в t_{BF}.

Сколько операций требуется на расчет эвристического значения состояния (x, K)? Генерация существенного состояния K занимает не более n^2 операций (см. [122, Appendix. A.2.2]); автору удалось улучшить данную оценку до wn (см. лемму 1.3). Затем требуется собственно расчет $\widetilde{v}(x, K)$, для чего необходимо рассмотреть все $i \in \widetilde{\mathbf{I}}[K]$,

которых не может быть более w[16], и не более чем m^2 пар $(z_{\text{in}}, z_{\text{out}})$, образующих множество $\mathbb{M}_i$; для каждой такой пары требуется t_{BF} операций для расчета

$$\max\Big\{\mathfrak{c}(x,(z_{\text{in}},z_{\text{out}}),K);\widetilde{v}\big(z_{\text{out}},K\setminus\{i\}\big)\Big\}.$$

После расчета $\widetilde{v}(x,K)$ придется потратить еще $\log_2 H$ операций на «сортировку»[17] полученного значения (записи в и извлечение из очереди с приоритетами). Таким образом, поиск значения усеченной функции Беллмана ($\widetilde{BF}$) для произвольного состояния (x,K) требует не более чем $n^2wm^2t_{\text{BF}}\log_2 H$ операций.

Всего имеется $n+1$ слоев $\widetilde{\mathcal{S}}_0,\ldots,\widetilde{\mathcal{S}}_n$, на каждом из которых не более чем H состояний, за исключением первого $\widetilde{\mathcal{S}}_0$, не ограничиваемого H, но все равно содержащего не более wm состояний и последнего $\widetilde{\mathcal{S}}_n$, с единственным состоянием $\big(x^0,1\,..\,n\big)$. Таким образом, расчет всех значений $\widetilde{V}$ требует не более чем

$$(wm+(n-2)H+1)wnwm^2t_{\text{BF}}\log_2 H=\mathcal{O}(n^2w^2m^2t_{\text{BF}}\cdot H\log_2 H)\text{ операций.}\qquad(2.1.31)$$

Чтобы найти решение, в худшем случае требуется просмотреть все $(wm+(n-2)H+1)$ состояний, то есть проверить для каждого (x,K), могло ли оно привести к $v\big(y,K\cup\{j\}\big)$; фактически, это расчет $v\big(y,K\cup\{j\}\big)$ без необходимости генерировать соответствующее состояние, откуда получаем сложность $(wm+(n-2)H+1)nm^2t_{\text{BF}}$, не меняющую порядок (2.1.31). Отметим, что настоящая оценка не слишком точна — мы не учитывали, что каждый существенный список заданий K на самом деле генерируется только один раз для всех состояний, содержащих его, кроме того, может не потребоваться рассматривать все m^2 пар $(z_{\text{in}},z_{\text{out}})\in\mathbb{M}_i$, поскольку в процессе отбора H лучших состояний на предыдущем этапе для некоторых z_{out} могли оказаться исключенными все состояния, для которых этот город является базой.

Таким образом, доказано следующее предложение:

Предложение 2.1. *Временная сложность эвристики усеченного ДП имеет порядок*

$$\mathcal{O}(n^2w^2m^2t_{\text{BF}}\cdot H\log_2 H),$$

где n — количество мегаполисов, *w —* ширина *частичного порядка $<_P$, задающего условия предшествования, m — максимальное количество городов в мегаполисе, H — параметр эвристики, а t_{BF} — константа, описывающая время, требуемое на расчет*

$$\max\Big\{\mathfrak{c}(x,(z_{\text{in}},z_{\text{out}}),K);\widetilde{v}\big(z_{\text{out}},K\setminus\{i\}\big)\Big\}.$$

2.1.8 Вычислительный эксперимент

Настоящий вычислительный эксперимент опубликован в [62]; он подводит итог работам [60, 61, 112, 113]. Модельные задачи впервые сформулированы в [60]; в [112] представлена задача более высокой размерности (число мегаполисов доведено до 30), решенная на суперкомпьютере ИММ УрО РАН, а также параллельная реализация алгоритма. На конференции [113] было представлено решение модельной задачи с зависимостью стоимости перемещений от списка невыполненных заданий, в [61] опубликовано доказательство корректности, переработанный вариант которого приведен в разделе 2.1.5 (см. теорему 2.1); наконец, в [62] представлен пример применения эвристики усеченного ДП (см. раздел 2.1.7).

[16] в [62], где этот результат получен, вместо ширины w указано мажорирующее ее количество мегаполисов n

[17] затраты, связанные с этой операцией, в [62] были упущены

Все модельные задачи рассматривались на плоскости $X = [0, 1024] \times [0, 768] \subset \mathbb{R} \times \mathbb{R}$. Мегаполисы представлены кругами одинакового радиуса, на окружностях которых на равных угловых расстояниях расставлялись принадлежащие им города, число которых одинаково для всех мегаполисов; ограничений на вход и выход (2.1.2) не вводилось. Расположение мегаполисов генерировалось случайным образом — центры мегаполисов наудачу бросались на плоскость[18] X до тех пор пока все n мегаполисов не были расставлены так, чтобы их «несущие круги» не пересекались ни друг с другом, ни с краями плоскости. База размещалась в точке с координатами $(0, 0)$.

Стоимость перемещения между городами, расположенными в разных мегаполисах (*внешние перемещения*), отождествлялась с евклидовым расстоянием между ними; в задачах с зависимостью от списка заданий это расстояние домножалось на коэффициент

$$a(|K|) = 1 + \frac{n - |K|}{n}$$

(имеет смысл «накопления усталости» к концу пути), где n — число мегаполисов, а K — список заданий, входящий в некоторое состояние (x, K), для которого рассчитывается стоимость перехода между x и K.

Стоимость перемещения между городами одного мегаполиса (*внутренние работы*) считалось по манхеттенской норме $||x - y|| = |x_1 - y_1| + |x_2 - y_2|$ перемещения из города с координатами (x_1, x_2) в другой с координатами (y_1, y_2) через *центр* мегаполиса — «выключить рубильник» в мегаполисе.

Во всех задачах применялся один и тот же набор условий предшествования, формализуемый *фундированным* отношением, содержащим следующие «адресные пары»: (1,9); (1,10); (2,13); (2,27); (3,6); (3,19); (6,16); (7,10); (8,2); (9,27); (10,9); (11,19); (12,2); (13,15); (14,16); (14,25); (14,26); (15,16); (18,17); (18,27); (20,19); (21,20); (23,22); (24,22); (25,26).

Как показали дальнейшие исследования, в настоящем наборе адресных пар две — (1,9) и (14,26) — принадлежат *транзитивной* компоненте и могут быть удалены, потому как множество допустимых маршрутов, равно как и множество существенных списков заданий, полностью определяется *транзитивной редукцией*; подробнее см. главу 1. Таким образом, транзитивная редукция настоящих условий предшествования содержит 23 адресных пары: (1,10); (2,13); (2,27); (3,6); (3,19); (6,16); (7,10); (8,2); (9,27); (10,9); (11,19); (12,2); (13,15); (14,16); (14,25); (15,16); (18,17); (18,27); (20,19); (21,20); (23,22); (24,22); (25,26).

Транзитивное замыкание[19] содержит 42 адресные пары: (1,9); (1,10); (1,27); (2,13); (2,15); (2,16); (2,27); (3,6); (3,16); (3,19); (6,16); (7,9); (7,10); (7,27); (8,2); (8,13); (8,15); (8,16); (8,27); (9,27); (10,9); (10,27); (11,19); (12,2); (12,13); (12,15); (12,16); (12,27); (13,15); (13,16); (14,16); (14,25); (14,26); (15,16); (18,17); (18,27); (20,19); (21,19); (21,20); (23,22); (24,22); (25,26).

Наименование каждой задачи содержит ее основные характеристики и выглядит как «X–Y–Z–W», где X — число мегаполисов, Y — число городов в каждом мегаполисе, Z — число условий предшествования в виде адресных пар, и W — либо «SD» (Sequence Dependent) — есть зависимость от списка невыполненных заданий, либо «NO» — такой зависимости нет. В задачах 27–Y–25–NO мегаполисы расположены одинаково, меняется только количество городов в них. В задачах 30–25–25–SD и 30–25–25–NO мегаполисы и города расположены одинаково — отличия состоят исключительно в стоимости внешних перемещений, как описано выше.

[18] строго говоря, на «целочисленную сетку» $0\,..\,1024 \times 0\,..\,768$, то есть, их распределение было *дискретным* равномерным

[19] то есть формализация условий предшествования в виде *частичного порядка*

Параллельная реализация точного алгоритма

В таблицах 2.1 и 2.2 приведено время, требуемое на получение точного решения (MM:CC), а также безразмерное «ускорение» — отношение времени счета последовательной реализации ко времени счета параллельной реализации с соответствующим числом исполнителей. Все алгоритмы реализованы на C++11, для параллелизации использовался API OpenMP для систем с общей памятью.

Таблица 2.1: Выигрыш от параллелизации, по [112]

Задача	посл.	пар.-4	пар.-8
27–10–25–NO	04:58 — 1	01:25 — 3,506	00:44 — 6,773
27–20–25–NO	38:45 — 1	10:34 — 3,667	05:18 — 7,311
27–25–25–NO	73:32 — 1	20:27 — 3,596	10:13 — 7,197

Таблица 2.2: Выигрыш от параллелизации, по [62]

Задача	посл.	пар.-2	пар.-3	пар.-4
27–10–25–NO	02:57 — 1	01:27 — 2,034	01:00 — 2,95	00:46 — 3,848
27–20–25–NO	22:34 — 1	11:28 — 1,968	08:08 — 2,775	05:53 — 3,836
27–25–25–NO	42:56 — 1	22:23 — 1,918	14:48 — 2,901	11:23 — 3,772

Результаты свидетельствуют, что при использовании вплоть до 8 ядер наблюдается близкий к линейному прирост производительности. Исследование на большем числе ядер исполнителей не было произведено в связи с реализацией параллельной программы на основе системы с общей памятью и отсутствием доступа к системе с более чем 8 исполнителями с общей памятью. Более чем двухкратный выигрыш от использования двух ядер в задаче 27–25–25–NO, по-видимому, объясняется «накладными расходами» на структуры OpenMP при запуске на одном исполнителе.

Эвристический эксперимент

Было исследовано качество эвристики на наиболее трудных экземплярах задач: 27–25–25–NO, 30–25–25–NO, 30–25–25–SD, при варьировании параметра глубины эвристики H от 1, что соответствует жадному алгоритму, до 20 000, с несколькими произвольно выбранными промежуточными значениями. Последнее значение было также определено произвольным образом, как «наибольшее, не требующее излишне длительных вычислений на компьютере автора», использовался ПК с центральным процессором Intel Core-i5-3450 и 16 ГБ оперативной памяти под управлением 64-битной Windows 7; исходный текст программы на C++ компилировался средствами Microsoft Visual C++ 2013 со стандартными настройками «Release». Время счета (физическое, не процессорное) замерялось с точностью до 1 секунды с помощью функции `time` классической библиотеки `time.h`.

Отметим, что в двух задачах из трех эвристический алгоритм успешно обнаружил оптимальной решение, в третьем же подошел достаточно близко (на 13 % хуже оптимального); затраты оперативной памяти при этом оказались существенно ниже, чем для точного решения — так, для задач с 30-ю мегаполисами, точное решение потребовало более 15 ГБ (но менее 40 ГБ) оперативной памяти, тогда как эвристическое — не более 100 МБ.

Также интересно было наблюдать попадание эвристики в *локальный* экстремум — например, в 30–25–25–NO решение для $H = 100$ было лучше, чем для $H = 150$, аналогичное явление наблюдалось и в 27–25–25–NO (для тех же значений параметра) и для 30–25–25–SD на значениях $H = 250$ и $H = 1\,000$. См. сводку результатов в таблице 2.3.

Таблица 2.3: Усеченное динамическое программирование для PSD-BGTSP-PC

27–25–25–NO	Оптимальное значение: $V = 341{,}962$									
Параметр H	1	10	100	150	250	1 000	2 500	5 000	10 000	20 000
Время счета	0:02	0:02	0:02	0:03	0:03	0:10	0:33	1:33	4:52	15:37
Эвр. значение $\widetilde{V}$	1 192,12	1 182,04	602,763	836,27	839,793	613,632	613,632	534,167	419,913	386,139
Отношение $V/\widetilde{V}$	3,49	3,46	1,76	2,45	2,46	1,79	1,79	1,56	1,23	1,13
30–25–25–NO	Оптимальное значение: $V = 316{,}68$									
Параметр H	1	10	100	150	250	1 000	2 500	5 000	10 000	20 000
Время счета	0:03	0:03	0:03	0:04	0:04	0:12	0:44	1:58	6:14	20:17
Эвр. значение $\widetilde{V}$	1 072,81	966,038	784,037	798,953	606,083	497,729	391,787	391,787	316,68	316,68
Отношение $V/\widetilde{V}$	3,39	3,05	2,48	2,52	1,91	1,57	1,24	1,24	1	1
30–25–25–SD	Оптимальное значение: $V = 376{,}63$									
Параметр H	1	10	100	150	250	1 000	2 500	5 000	10 000	20000
Время счета	0:02	0:02	0:02	0:03	0:03	0:10	0:33	1:33	4:52	15:37
Эвр. значение $\widetilde{V}$	972,557	966,038	768,836	784,037	579,734	634,734	487,558	376,63	376,63	376,63
Отношение $V/\widetilde{V}$	2,58	2,56	2,04	2,08	1,54	1,69	1,29	1	1	1

Глава 3

Некоторые задачи без условий предшествования

Здесь собраны «прочие» результаты, не касающиеся напрямую основной темы диссертации — *динамического программирования* для маршрутных задач *с условиями предшествования*. Каждому результату поручен свой раздел.

3.1 Задача о перестановке однотипных объектов

Задача впервые поставлена в [151] для приложений к полевой работе специалистов-биологов, изучающих динамику популяций некоторых видов животных. Для оценки текущей популяции могут использоваться, в частности, *камеры-ловушки* (срабатывающие от датчика движения и т. п.); предполагается, что перестановка этих камер-ловушек способна увеличить статистическую значимость наблюдений. При этом, с одной стороны, камеры-ловушки сравнительно легки и компактны, соответственно, можно не ограничивать емкость «рюкзака», в котором они переносятся; с другой стороны, они достаточно дороги и трудоемки в обслуживании чтобы исключить возможность просто покрыть ими всю желаемую территорию.

Отсюда получается очень слабая версия 1-Commodity Pickup and Delivery TSP (1-PDTSP) [115], «задачи коммивояжера с получением и доставкой для одного товара»: с бесконечной емкостью и бинарными (0–1) спросом и предложением. В [151] алгоритм ДП был реализован на Perl, представлено *точное* решение для задачи с 8 камерами. В [116] Я. В. Салием представлен обновленный, более эффективный вариант ДП и его реализация на C++, а также программа расчета количества *состояний* представленного варианта ДП, реализованная на Haskell; Е. Е. Иванко представил жадный алгоритм, эффективность которого оценена в сравнении с точным ДП на предельной достигнутой размерности в 12 камер.

В сравнении с [115], где 1-PDTSP решается линейной релаксацией целочисленной программы (Mixed Integer-Linear Programming, MILP), оптимально решаются экземпляры задачи, содержащие до 75 городов (сравнимо с 37 камерами-ловушками; см. постановку ниже), за время не более 182 секунд, и на, вероятно, менее производительном компьютере, чем в [116]. Тем не менее, есть две причины все-таки использовать точное ДП и более простые эвристики чем основанные на релаксациях целочисленной модели: (а) в заявленном приложении нет потребности решать задачи большой размерности, 12 камер-ловушек вполне достаточно (см. подробнее в [151]), (б) нет потребности в закупке лицензий достаточно дорогих программных пакетов, эффективно решающих задачи линейного программирования (к примеру, IBM ILOG CPLEX).

3.1.1 Постановка задачи

Сохраняем самые общие обозначения раздела 1.1.1: символ $\triangleq$ обозначает равенство по определению, *упорядоченная пара* двух объектов a and b записывается в круглых скобках, (a, b), а *все подмножества* некоторого множества M обозначаются $\mathcal{P}(M)$.

Размерность задачи — число камер-ловушек — обозначим натуральным числом n. Множество *всех городов* обозначим $X \triangleq 0\,..\,2n = \{0\} \cup \mathcal{R} \cup \mathcal{B}$, где 0 обозначает *базу*[1], множество $\mathcal{R} \triangleq 1\,..\,n$, $|\mathcal{R}| = n$, содержит индексы точек, где размещены камеры, а множество $\mathcal{B} \triangleq n+1\,..\,2n$, $|\mathcal{B}| = n$ — индексы точек, куда их нужно переставить. Далее мы будем называть «точки, где размещены камеры» *красными*, а «точки, куда нужно переставить камеры» — *синими*. Термины «точка» и «город» взаимозаменяемы; мы также не различаем точки и их индексы — там, где это не приводит к путанице.

Стоимость перемещения между двумя точками определяет функция $\mathfrak{c}\colon X \times X \to \mathbb{R}$. Любой *экземпляр* задачи определяется множеством X и функцией стоимости перемещений $\mathfrak{c}$. *Решением* задачи $(X, \mathfrak{c})$ называем *перестановку* индексов $1\,..\,2n$ (маршрут); обозначим множество всех маршрутов $\mathbb{P}$. Маршруты обозначаем строчными греческими буквами; в выражении вида $\alpha_i = j$, натуральное число j — индекс точки, стоящей на i-м месте в маршруте α.

Поскольку *синие* точки можно посещать только неся хотя бы одну камеру «в рюкзаке», часть маршрутов непременно будет *недопустимыми*, например, маршруты, сначала посещающие все синие точки. Допустимые маршруты введем с помощью «оператора допустимой точки выхода» $\mathbf{I} : \mathcal{P}(X) \to \mathcal{P}(X)$, определяемого следующим образом:

$$\mathbf{I}[K] \triangleq \begin{cases} K, & |K \cap \mathcal{R}| > |K \cap \mathcal{B}|; \\ K \cap \mathcal{B}, & |K \cap \mathcal{R}| = |K \cap \mathcal{B}|. \end{cases}$$

Суть этого определения состоит в том, что *последней* в маршруте из базы 0 через K может быть точка *любого* цвета если в K *больше* красных чем синих точек, но если, наоборот, синих и красных точек в K *поровну*, то последней может быть только *синяя* точка. Очевидно, что ситуация, когда пройдено больше синих чем красных точек недопустима: в этом случае собранных по ходу движения камер не хватит, чтобы занять все вакантные места.

Посредством оператора $\mathbf{I}$ *допустимость* маршрута выражается «пошаговым» образом, через применение этого оператора ко всем префиксам[2]:

$$\mathbb{A} \triangleq \left\{ \alpha \in \mathbb{P} \,\middle|\, \forall s \in 1\,..\,2n \;\; \alpha(s) \in \mathbf{I}\big[\{\alpha(t) : t \in 1\,..\,s\}\big] \right\}.$$

Отметим, что такие маршруты являются *языками Дика*, количество которых определяется n-м *числом Каталана*, см. [152].

Наконец, качество *допустимого маршрута* $\alpha \in \mathbb{A}$ определяет следующая аддитивная целевая функция (критерий качества):

$$\mathfrak{C}[\alpha] \triangleq \left\{ \mathfrak{c}\big(0, \alpha_1\big) + \sum_{i=1}^{2n-1} \mathfrak{c}(\alpha_i, \alpha_{i+1}) + \mathfrak{c}\big(\alpha_{2n}, 0\big) \right\};$$

сама *задача* состоит в *минимизации* этого критерия качества посредством выбора наилучшего допустимого маршрута.

---

[1] с нее начинается обход; на нее требуется вернуться по завершении обхода: задача замкнутая

[2] формально, к множествам-«носителям» префиксов — образам маршрутов как *биекций*

3.1.2 Динамическое программирование

Мы используем *прямую* процедуру ДП, где состояние — множество *пройденных* начиная с выхода из базы точек и точка, где находится агент, после обхода этого множества. Корректность ДП для настоящей задачи доказана в [151].

Формально, *состояние* есть упорядоченная пара (K, x), где $K \subset 1\,..\,2n$ называется *списком заданий*, а $x \in X \setminus K$ — *терминалом*; каждое состояние описывает «подзадачу» оптимального обхода точек из множества K при старте из базы 0 и финише на терминале x. В отличие от обычной TSP, не все состояния *допустимы*: некоторые подзадачи невозможно решить, не нарушив условия «непустого рюкзака», например, состояния вида $(\mathcal{B}, x)$, $x \in \mathcal{R}$, где сначала посещаются все *синие* точки, а камеры на них установить не получается, потому что они еще не собраны с *красных* точек.

В [151] значение *недопустимых* состояний полагалось «машинно бесконечным» (некоторое число, большее стоимости любого возможного обхода). Можно сэкономить как процессорное время так и память, где хранятся состояния и их значения, полностью исключив недопустимые состояния их рассмотрения, для чего мы вводим представленные ниже формальное условие допустимости и процедуру генерации существенных состояний.

Назовем список заданий $K \subset 1\,..\,2n$ *допустимым* если синих точек в нем *не меньше* чем красных; соберем все такие списки заданий в семейство

$$\mathcal{G} \triangleq \Big\{ K \subset X \;\Big|\; |K \cap \mathcal{R}| \geqslant |K \cap \mathcal{B}| \Big\}.$$

Стратифицируем это семейство по мощности, $\mathcal{G}_i \triangleq \big\{ K \in \mathcal{G} \;\big|\; |K| = i \big\}$, $i \in 0\,..\,2n$, пустой список заданий из $\mathcal{G}_0 = \{\varnothing\}$ также полагается допустимым.

Не все состояния с допустимым списком заданий допустимы сами: терминал может быть выбран недопустимым образом. Очевидно, что если в некотором списке заданий $K \in \mathcal{G}$ красных точек *больше* чем синих, в качестве терминала можно выбрать точку любого цвета. Однако, если красных и синих точек *поровну*, все красные точки «используются» для расстановки камер в синие точки этого списка, таким образом, рюкзак пуст и нужно далее идти в *красную* точку. Это условие формально выражает *оператор допустимого продолжения* $\mathbf{E} : \mathcal{G} \to X$, определенный по правилу

$$\mathbf{E}[K] \triangleq \begin{cases} X \setminus K, & |K \cap \mathcal{R}| > |K \cap \mathcal{B}|; \\ \big(X \setminus K\big) \cap \mathcal{R}, & |K \cap \mathcal{R}| = |K \cap \mathcal{B}|; \end{cases}$$

на краевой случай $K = \mathcal{R} \cup \mathcal{B}$ мы не распространяем действие $\mathbf{E}$, его удобнее рассматривать отдельно. Также оператор допустимого продолжения можно выразить и через оператор допустимой точки выхода $\mathbf{I}$,

$$\mathbf{E}[K] = \Big\{ x \in (X \setminus K) \;\Big|\; x \in \mathbf{I}[K \cup \{x\}] \Big\}.$$

Наконец, назовем состояние (K, x) *допустимым* если $K \in \mathcal{G}$ и $x \in \mathbf{E}(K)$. Соберем все допустимые состояния в семейство

$$\mathcal{S} \triangleq \big\{ (K, x) \in \mathcal{P}(X) \times X \mid K \in \mathcal{G} \setminus \mathcal{G}_{2n},\ x \in \mathbf{E}[K] \big\} \cup \big\{ (1\,..\,2n, 0) \big\},$$

которое стратифицируем по мощности списков заданий: $\forall i \in 0\,..\,2n$, $\mathcal{S}_i \triangleq \big\{ (K, x) \in \mathcal{S} \mid i = |K| \big\}$; элементы этого разбиения будем называть *слоями* (пространства состояний). Отметим два особых слоя: $\mathcal{S}_{2n} = \big\{ (1\,..\,2n, 0) \big\}$ содержит состояние, соответствующее *полной* задаче, а $\mathcal{S}_0 = \big\{ (\varnothing, x) : x \in \mathbf{E}[\varnothing] \big\}$ содержит *тривиальные* подзадачи, соответствующие перемещению из базы 0 в некоторую *красную* точку x.

Для каждого состояния $(K, x) \in \mathcal{S}$, его *значение* — стоимость оптимального обхода K, начатого из базы 0 и законченного в x — определяется следующим рекурсивным *уравнением Беллмана*:

$$v(K,x) \triangleq \begin{cases} \mathfrak{c}(0,x), & (0,x) \in \mathcal{S}_0; \\ \min\limits_{y \in \mathbf{I}(K)} \Big\{ v\big(K \setminus \{y\}, y\big) + \mathfrak{c}(y,x) \Big\}, & (K,x) \in \mathcal{S} \setminus \mathcal{S}_0; \end{cases}$$

здесь тривиальные состояния $\mathcal{S}_0$ играют роль начального условия, а значение полной задачи представляется в виде $v\big(1\,..\,2n, 0\big)$; корректность такого определения уравнения Беллмана следует из корректности аналогичного уравнения в [151, раздел 2.1].

Ниже представлена процедура генерации всех допустимых состояний $\mathcal{S}$ снизу вверх (начиная с тривиальных $\mathcal{S}_0$) и расчета их значений:

1. Инициализация: $\mathcal{G}_0 = \{\varnothing\}$, $\mathcal{G}_1 = \mathcal{R}$, $\mathcal{S}_0 = \{\varnothing\} \times \mathcal{R}$. Начальные условия $v[\mathcal{S}_0]$ не требуют расчетов.

2. Для каждого $l \in 1\,..\,2n-1$

   Для каждого $K \in \mathcal{G}_l$, для каждого $x \in \mathbf{E}[K]$

   - Добавить $K \cup \{x\}$ в $\mathcal{G}_{l+1}$
   - Добавить (K, x) в $\mathcal{S}_l$. Рассчитать $v(K, x)$ по уравнению Беллмана

3. Расчитать значение полной задачи $v\big(\{1\,..\,2n\}, 0\big)$ по уравнению Беллмана

4. Восстановить оптимальный маршрут по значениям $v[\mathcal{S}]$

Процедура восстановления на шаге 4 описана, например, в [151, раздел 2.2] и аналогична процедуре в разделе 1.3.1.

3.1.3 Вычислительный эксперимент

Я. В. Салием был написан на C++ генератор экземпляров задачи, работающий следующим образом: на плоскости $[0, 1024] \times [\,.\,0, 768]$ база имеет координаты $(0, 0)$; прочие точки не совсем[3] равномерно брошены на эту плоскость через `std::uniform_int_distribution` из стандартой библиотеки C++ `std::random`; если (евклидово) расстояние между новой бросаемой на плоскость точкой и уже размещенными точками оказывалось менее 10, точка бросалась повторно, до тех пор пока это условие не было достигнуто. Затем рассчитывалось евклидово расстояние между всеми точками (число с плавающей запятой, тип `double`); в качестве $\mathfrak{c}$ принималось ближайшее *целое*, не меньшее полученного расстояния, и далее все вычисления производились в целых числах.

Проведено сравнение на 100 случайно сгенерированных экземплярах с $n = 12$ камерами-ловушками (25 точек, считая базу) между точным ДП и жадным алгоритмом. Конкретное значение $n = 12$ выбрано исходя их того, что это наибольшая размерность, для которой расход оперативной памяти не превосходил доступные 16 ГБ.

Время счета точного алгоритма составило от 85 до 94 секунд; время работы жадного алгоритма пренебрежимо мало. Превышение эвристического значения над оптимальным (отношение эвристического к оптимальному минус 1, в процентах) в среднем составило 28 %, минимальное оказалось 1 %, максимальное 81 % и медианное 26 %.

[3] координаты имеют *дискретное* равномерное распределение на $0\,..\,1024$ и на $0\,..\,768$, по горизонтали и вертикали, соответственно

3.1.4 Пространственная сложность точного динамического программирования

В нашей реализации алгоритма в 16 ГБ оперативной памяти помещались задачи размерностью не выше 12 камер-ловушек (расход составлял порядка 12 ГБ), соответственно, возникло желание аналитически определить пространственную сложность точного ДП (выражаемую количеством состояний, каждое из которых нужно хранить, вместе с его значением), и сравнить ее с аналогичным параметром для классической TSP, то есть, для n камер-ловушек с TSP для $2n$ городов — чтобы понять, стоит ли оптимизировать расход памяти в программной реализации, или сдаться на милость комбинаторного взрыва.

Нетрудно посчитать, сколько состояний в обычной TSP [16]. В отличие от настоящей задачи, все состояния допустимы; тогда, для n городов получается

$$C_{\mathrm{TSP}}(n) = \sum_{i=0}^{n-1} \binom{n}{i}(n-i) + \binom{n}{n}1.$$

Действительно, при $i \in 0\,..\,n-1$ имеем $|\mathcal{G}_i| = \binom{n}{i}$, и количество терминалов получается $n-i$; в $\mathcal{S}_n$ только одно состояние, $(1\,..\,n, 0)$.

В настоящей задаче такой расчет получается более сложным: (а) некоторые списки заданий недопустимы, (б) некоторые продолжения недопустимы (когда в списке заданий *поровну* красных и синих точек, продолжением являются только *красные* точки, не входящие в список). Нам не удалось красиво упростить итоговый результат, поэтому, вместо формальной общей формулы, мы приведем пример расчета для экземпляра задачи с *тремя* камерами-ловушками.

Рассмотрим существенные списки заданий $\mathcal{G}_0, \ldots, \mathcal{G}_6$. В $\mathcal{G}_0$ нет ни красных, ни синих точек $|\mathcal{G}_0| = \binom{3}{0}\binom{3}{0}$; очевидно, он продолжается всеми *красными* точками, то есть, $|\mathcal{S}_0| = |\mathcal{R}| = 3$. Далее, мощность $\mathcal{G}_1$ *нечетна*. Поскольку точка только одна, она должна быть *красной*, таким образом, $|\mathcal{G}_1| = \binom{3}{1}\binom{3}{0}$; во всех списках заданий $\mathcal{G}_1$ красных точек больше чем синих, так что продолжением может служить любая из оставшихся $2n-1$ штук, $|\mathcal{S}_1| = \binom{3}{1}\binom{3}{0} \cdot 5$. Перейдем к $\mathcal{G}_2$. Можно выбрать две красных точки, либо одну красную и одну синюю, $|\mathcal{G}_2| = \binom{3}{2}\binom{3}{0} + \binom{3}{1}\binom{3}{1}$. В первом случае, допустимо продолжение на все остающиеся $2n-2$ точек, однако, во втором продолжать обязательно *красной* точкой, которых остается $n-1$: $|\mathcal{S}_2| = \binom{3}{2}\binom{3}{0} \cdot 4 + \binom{3}{1}\binom{3}{1} \cdot 2$. Для $\mathcal{G}_3$ получаем $|\mathcal{G}_3| = \binom{3}{3}\binom{3}{0} + \binom{3}{2}\binom{3}{1}$ и $|\mathcal{S}_3| = \binom{3}{3}\binom{3}{0} \cdot 3 + \binom{3}{2}\binom{3}{1} \cdot 3$. Переходя к $\mathcal{G}_4$, имеем $|\mathcal{S}_4| = \binom{3}{3}\binom{3}{1} \cdot 2 + \binom{3}{2}\binom{3}{2} \cdot 2$; тогда, для $\mathcal{S}_5$, будет $|\mathcal{S}_5| = \binom{3}{3}\binom{3}{2} \cdot 1$, а $\mathcal{S}_6$ состоит из одного элемента, $|\mathcal{S}_6| = \binom{3}{3}\binom{3}{3} \cdot 1$.

Процедура расчета запрограммирована на Haskell; также рассчитано отношение между $C_{\mathrm{TSP}}(2n)$ и $|\mathcal{S}|$ для задачи с n камерами. Для одной камеры-ловушки оно составляет 1,666 (2 города в TSP, не считая базы), возрастает до 1,941 для 2 камер-ловушек (4 города в TSP, не считая базы), и имеет 6 девяток после запятой для 9 камер-ловушек (18 городов в TSP, не считая базы). Таким образом, представляется, что асимптотически эта задача «вдвое проще» TSP при решении динамическим программированием, что составляет не слишком радужный прогноз для повышения размерности экземпляров, решаемых по ДП — при потребности в большей размерности, нужно использовать другие методы, либо пользоваться эвристиками.

3.2 Ультраметрическая задача коммивояжера на узкие места

«Задача коммивояжера на узкие места» (Bottleneck TSP, BTSP) [93], также называемая «минимаксной задача коммивояжера» [153], состоит в минимизации стоимости наиболее

затратного из перемещений между городами в маршруте; см. обзор [2, Ch. 15], где, в частности, описана процедура сведения BTSP к TSP. О последних достижениях см. [94].

Критерий качества в BTSP и сама задача (в *незамкнутой* форме — без возвращения на базу) формулируются следующим образом:

$$\begin{aligned} &\mathfrak{C}[\alpha] = \max\left\{\mathfrak{c}\big(0,\alpha_1\big); \max_{i\in 1..n-1}\Big[\mathfrak{c}\big(\alpha_i,\alpha_{i+1}\big)\Big]\right\};\\ &\mathfrak{C}[\alpha] \xrightarrow[\alpha\in(\mathrm{bi})[1..n]]{} \min\,. \end{aligned} \tag{BTSP}$$

*Ультра*метрикой называется метрика $\mathfrak{c}$, удовлетворяющая дополнительно к обычным аксиомам еще *усиленному неравенству треугольника*

$$\forall a,b,c\in 1\,..\,n\;\; \mathfrak{c}(a,c)\leqslant \max\{\mathfrak{c}(a,b);\mathfrak{c}(b,c)\}. \tag{STI}$$

О приложениях см. [109, 154, 155].

TSP в ультраметрическом пространстве разрешима за полиномиальное время [156, 157], более того, оптимальное решение можно найти *жадным* алгоритмом [156]. Данный результат — вместе с замечанием научного руководителя о том, что и в ультраметрическом неравенстве треугольника (STI) и в BTSP используется max — подтолкнул автора к исследованию *ультраметрической* задачи коммивояжера на узкие места, результатом чего стал доклад на конференции [114] о тривиальности такой задачи.

В процессе написания диссертации автор обнаружил, что результат, идейно схожий с Теоремой 3.1 о тривиальности BTSP в ультраметрическом пространстве, уже был доказан в качестве характеризации *ультраметрик* [109, Proposition 3.1]; вместе с тем, в [109] никак не упоминается BTSP, соответственно, автор оставляет за собой приоритет в области ее тривиализации в ультаметрическом пространстве.

Ультраметрической называем BTSP, в которой функция стоимости перемещений $\mathfrak{c}$, для которой выполнены *аксиомы метрики* [158, Appendix B3], подчиняется *усиленному неравенству треугольника* (STI), см. подробнее [158, p. 2] (strong triangle inequality; также известно как *ультраметрическое неравенство*).

Важным следствием этого неравенства является *принцип равнобедренного треугольника*, фактически, означающий, что треугольники бывают двух видов: равносторонний и равнобедренный с основанием, меньшим боковых сторон

$$\forall a,b,c\in 1\,..\,n\;\; \big(\mathfrak{c}(a,b)\neq\mathfrak{c}(b,c)\big)\to\big(\mathfrak{c}(a,c)=\max\big\{\mathfrak{c}(a,b);\mathfrak{c}(b,c)\big\}\big); \tag{$\triangle$}$$

доказательство этого принципа обычно опускают, предлагая читателю в качестве упражнения, поэтому автор полагает уместным его привести.

Доказательство ($\triangle$)**.** Фиксируем произвольные $a,b,c\in 1\,..\,n$. Без ограничения общности предположим $\mathfrak{c}(a,b)<\mathfrak{c}(b,c)$. Покажем, что для $\mathfrak{c}(a,c)$ (STI) обращается в равенство. Действительно, $\mathfrak{c}(a,b)<\mathfrak{c}(b,c)$ влечет $\max\{\mathfrak{c}(a,b);\mathfrak{c}(b,c)\}=\mathfrak{c}(b,c)$, откуда, с учетом (STI) для $\mathfrak{c}(a,c)$, следует $\mathfrak{c}(a,c)\leqslant\mathfrak{c}(b,c)$. С другой стороны, согласно (STI), $\mathfrak{c}(b,c)\leqslant\max\big\{\mathfrak{c}(a,b);\mathfrak{c}(a,c)\big\}=\mathfrak{c}(a,c)$, поскольку $\mathfrak{c}(a,b)<\mathfrak{c}(b,c)$ по предположению. Таким образом,

$$\big(\mathfrak{c}(a,c)\leqslant\mathfrak{c}(b,c)\big)\wedge\big(\mathfrak{c}(b,c)\leqslant\mathfrak{c}(a,c)\big),$$

откуда следует $\mathfrak{c}(a,c)=\mathfrak{c}(b,c)$, что и требовалось доказать. □.

С помощью ($\triangle$) и принципа математической индукции мы покажем тривиальность (BTSP) в ультраметрическом пространстве, а именно, тот факт что для любого маршрута значением задачи (BTSP) будет стоимость *наиболее дорогого* из всех возможных

перемещений. Этот факт — следствие нижеследующей теоремы, для удобства доказательства которой мы введем некоторые термины, действующие внутри данного подраздела и сходные с терминологией теории графов.

Ребром назовем УП $(i,j) \in 0\,..\,n \times 1\,..\,n$; каждому ребру сопоставим его *вес* $\mathfrak{c}(i,j)$; поскольку $\mathfrak{c}$ удовлетворяет аксиомам ультраметрики, $\mathfrak{c}(i,j) = \mathfrak{c}(j,i)$, если только $i \neq 0$ ($\mathfrak{c}$ не определена для перемещения *на* базу — рассматриваем *незамкнутую* задачу).

Теорема 3.1. *Пусть $\alpha \in (\mathrm{bi})[1\,..\,n]$ — произвольный маршрут, $a \in 1\,..\,n$ и $b \in 1\,..\,n$ — произвольные города, посещаемые коммивояжером. Без ограничения общности предположим, что $\alpha_a^{-1} < \alpha_b^{-1}$: город a посещается ранее города b. Тогда в маршруте α между a и b найдется ребро (p,q), $p \in \alpha_a^{-1}\,..\,\alpha_b^{-1} - 1$, $q \in \alpha_a^{-1} + 1\,..\,\alpha_b^{-1}$, стоящее между a и b, вес которого не меньше чем вес ребра (a,b):*

$$\exists p \in \alpha_a^{-1}\,..\,\alpha_b^{-1} - 1\ \exists q \in \alpha_a^{-1} + 1\,..\,\alpha_b^{-1} : \big(q = p+1\big) \wedge \big(\mathfrak{c}(p,q) \leqslant \mathfrak{c}(a,b)\big). \tag{3.2.1}$$

Доказательство. Если a и b стоят в маршруте подряд ($\alpha_b^{-1} = \alpha_a^{-1} + 1$), то требованиям (3.2.1) удовлетворяет само ребро (a,b). Рассмотрим далее случай $\alpha_b^{-1} > \alpha_a^{-1} + 1$. Проведем математическую индукцию по числу городов, стоящих между a и b в маршруте α. Очевидно, это число изменяется от 1 до $n-2$.

Б. И. Один промежуточный город, $\alpha_b^{-1} = \alpha_a^{-1} + 2$. Обозначим город, стоящий между a и b символом l: $l \triangleq \alpha_{\alpha_a^{-1}+1}$. Тогда в маршруте между a и b находятся ребра (a,l) и (l,b). Рассмотрим треугольник $\triangle(a,l,b)$. Записав ($\triangle$) для ребер (a,b) и (b,l), получим

$$\big(\mathfrak{c}(a,b) \neq \mathfrak{c}(b,l)\big) \to \big(\mathfrak{c}(a,l) = \max\{\mathfrak{c}(a,b); \mathfrak{c}(b,l)\}\big).$$

То есть, или $\mathfrak{c}(b,l) = \mathfrak{c}(a,b)$, или $\mathfrak{c}(a,l) \geqslant \mathfrak{c}(a,b)$. База индукции доказана.

Ш. И. k промежуточных городов, $k \in 2\,..\,n-2$; $\alpha^{-1}(b) = \alpha^{-1}(a) + k + 1$. Пусть предположение индукции (3.2.1) выполнено для $k-1$ промежуточных городов. Покажем, что оно выполняется и для k. В таком случае, маршрут между a и b проходит через города, стоящие в маршруте α на местах

$$\alpha_a^{-1} + 1, \alpha_a^{-1} + 2, \ldots, \alpha_a^{-1} + k - 1 = \alpha_b^{-1} - 2, \alpha_a^{-1} + k = \alpha_{b-1}^{-1}.$$

Рассмотрим другой маршрут $\hat{\alpha}$, который совпадает с α всюду за исключением положения города $l \triangleq \alpha_{\alpha_a^{-1}+k}$, который в $\hat{\alpha}$ «исключен» из перемещения между a и b, и город b сдвинут на один шаг назад:

$$\forall s \in \alpha_a^{-1}\,..\,\alpha_a^{-1} + k - 1\ (\hat{\alpha}_s = \alpha_s) \wedge (\hat{\alpha}_b^{-1} = \alpha_a^{-1} + k = \alpha_b^{-1} - 1),$$
$$\big(\hat{\alpha}_l^{-1} < \hat{\alpha}_a^{-1}\big) \vee \big(\hat{\alpha}_l^{-1} > \hat{\alpha}_b^{-1}\big),$$

где остальная часть маршрута определяется произвольным образом; очевидно, что такой маршрут $\hat{\alpha}$ существует. В нем между a и b по построению расположено $k-1$ городов, значит, для него выполняется предположение индукции (3.2.1). Тогда возможны два варианта. Если ребро (p,q) веса не меньшего чем $\mathfrak{c}(a,b)$ приходится на ту часть маршрута $\hat{\alpha}$, которая совпадает с α, $q \leqslant \hat{\alpha}_a^{-1} + k - 1 = \alpha_a^{-1} + k - 1$, то шаг индукции доказан, потому что ребро соответствующего веса непосредственно находится в маршруте α. Рассмотрим второй случай, $p = \hat{\alpha}_a^{-1} + k - 1 = \alpha_a^{-1} + k - 1$, $q = \hat{\alpha}_a^{-1} + k = \alpha_a^{-1} + k + 1 = \alpha_b^{-1}$. К треугольнику $\triangle\big(p,l,q\big) = \triangle\big(\alpha_a^{-1} + k - 1, \alpha_a^{-1} + k, \alpha_a^{-1} + k + 1\big)$ применим тот же аргумент, что и в базе индукции: $\mathfrak{c}(p,q) = \mathfrak{c}(a,b)$ и, записав ($\triangle$) для ребра p,q, получим

$$\big(\mathfrak{c}(p,q) \neq \mathfrak{c}(q,l)\big) \to \big(\mathfrak{c}(p,l) = \max\{\mathfrak{c}(p,q); \mathfrak{c}(q,l)\}\big),$$

откуда следует

$$\Big(\mathfrak{c}(p,l) = \mathfrak{c}(p,q) \geqslant \mathfrak{c}(a,b)\Big) \vee \Big(\mathfrak{c}\big(\alpha_a^{-1}+k, \alpha_a^{-1}+k+1\big) = \mathfrak{c}(l,q) \geqslant \mathfrak{c}(p,q) \geqslant \mathfrak{c}(a,b)\Big),$$

что и требовалось доказать. Шаг индукции доказан, а с ним и Теорема 3.1. Нетрудно заметить, что данная теорема будет выполняться и в случае если вместо города a взять *базу*; мы не включили этот случай в доказательство чтобы избежать громоздких выкладок — маршруты по определению не содержат *базу*, пришлось бы вводить дополнительные конструкции и делать лишние оговорки. □

Следствие 4. *Теорема* (3.1) *выполняется и для* самого тяжелого *ребра*

$$(p^*, q^*) \triangleq \operatorname*{argmax}_{i\in 0..n, j\in 1..n\setminus\{i\}} \big\{\mathfrak{c}(i,j)\big\}.$$

Таким образом, если $\mathfrak{c}$ является ультраметрикой, то *самое тяжелое ребро* непременно встретится в *любом* маршруте и (BTSP) вырождается: целевая функция $\mathfrak{C}$ тождественно равна константе $\mathfrak{c}(p^*, q^*)$ на всем пространстве решений $(\mathrm{bi})[1\,..\,n]$.

Глава 4

Программный комплекс. Точное и усеченное динамическое программирования для задачи курьера и ее обобщений

Настоящая глава посвящена устройству и вопросам реализации комплекса программ, выполняющего решение точным и усеченным динамическим программированием различных обобщений TSP-PC по алгоритмам, описанным в главах 1,2. Комплекс [117] является логическим развитием менее универсального варианта [118], достаточно подробно описанного в [62]. Вычислительные эксперименты, в которых использовался комплекс [117], описаны в гл. 1; эксперименты, упомянутые в гл. 2, проводились с помощью различных версий комплекса [118].

В настоящей главе конструкции, относящиеся к языку программирования C++, набраны `машинописным` шрифтом, например, так:

```
std::cout<<"Hello, World!";
```

Также `машинописным` шрифтом выделены *расширения* файлов (предполагается, что *расширение* однозначно задает *тип* файла); например, `.sop` обозначает тип файлов, содержащий исходные данные TSP-PC в формате TSPLIB [76].

4.1 Используемые технологии

Комплекс реализован в виде модульной программы на C++, где «модули» представлены в виде традиционных для C++ заголовочных файлов `.h` и исходных текстов `.cpp`. В среде Microsoft Windows использовался компилятор Microsoft Visual C++ 14 (далее MS VC++), в среде CentOS Linux использовался компилятор C++ из набора gcc 6.3.0 (далее gcc). Машинное представление математических объектов и структур, описанных в главах 1,2 выполнено практически полностью средствами Standard Template Library (STL) — «массивом бит» `std::bitset`, контейнерами `std::unordered_map`, `std::map` и др.

«Массив бит» `std::bitset`. Обеспечивает машинное представление множеств. За счет высокой эффективности — теоретико-множественные операции представлены двоичной арифметикой — позволяет добиться высокой производительности комплекса в части, касающейся обработки условий предшествования (операторы **E**, **I**) и сравнительно малого расхода памяти. При производительности, сравнимой с «ручной» реализацией множеств в виде целых чисел без знака (в предварительных тестах, реализация над

64-битными целыми числами без знака `uint64_t` выполняла заданный набор теоретико-множественных операций всего лишь на *единицы* процентов быстрее, чем реализация на 64-битных `std::bitset<64>`), стандартный массив бит удобнее в работе, что снижает количество ошибок и упрощает отладку. В реализации, средствами которой выполнен вычислительный эксперимент главы 1 на задачах из TSPLIB используется размерность 384 (`std::bitset<384>`) — кратное 64 битам число, позволяющее представить все задачи из TSPLIB: больше всего городов содержит задача `rbg378a.sop`, их 378, не считая базы 0 и терминала $\mathfrak{t}$.

Кроме того, средствами `std::bitset` описаны и условия предшествования: с каждым городом (мегаполисом) ассоциированы множества городов, строго меньших его в отношении $<_P$ и строго больших его; фактически, имеем представление отношений $<_P$ и $>_P$ в виде матрицы смежности, однако, в отличие от программы, анализирующей условия предшествования (см. раздел 1.4.5), реляционная семантика напрямую не используется.

Хеш-таблица `std::unordered_map`. Используется как компонент структуры данных, в которой хранятся *состояния* ДП $\mathcal{S}$, упорядоченные по мощности *списка заданий*. Ключом в хеш-таблице является машинное представление множества в виде `std::bitset`, используется хеш-функция[1], предоставляемая STL; отметим, что, учитывая особенности ключей (теоретико-порядковые идеалы), можно ставить вопрос об использовании более эффективных (в смысле вычислительной сложности и расхода памяти) хеш-функций, например, можно предположить применение вариаций «алгоритма пометки» идеалов [150] (см. обсуждение разновидностей этого алгоритма в [22, 23, 128]).

Очередь с приоритетами `std::priority_queue`. Ключевой элемент реализации *усеченного* ДП: используется для отбора H наилучших состояний на каждом слое. Поскольку параметр эвристики H известен заранее, в «носителе» очереди с приоритетами (мы используем `std::vector`) сразу резервируется место на H элементов (метод `reserve`), что позволяет избежать временных затрат на увеличение максимальной емкости контейнера-носителя во время работы алгоритма. В качестве частичного порядка, задающего приоритет, используется сравнение состояний по значению («усеченной») функции Беллмана, при этом наибольший приоритет (и первое место в очереди) отдается *наихудшему* состоянию — с самым высоким значением, для упрощения его удаления из очереди.

Межплатформенная совместимость. Условная компиляция. Комплекс разрабатывался, в основном, в пакете Microsoft Visual Studio, при этом предполагалось использование его как в среде под управлением Microsoft Windows, так и в Linux-подобных операционных системах (использовался компилятор gcc и система сборки cmake). Для обеспечения единообразия и упрощения отладки в разных средах максимально использовались стандартные элементы C++, но в нескольких случаях пришлось прибегнуть к распространенному приему *условной компиляции*: согласно директивам разработчика, в зависимости от (обнаруживаемой) операционной системы, препроцессор C передает на компиляцию необходимый фрагмент исходного кода, специфический для запущенной операционной системы.

[1] как показывает эксперимент, в реализациях STL для MS VC++ и gcc используются *разные* хеш-функции для `std::bitset`, что приводит к разному же порядку хранения состояний. Последнее влияет на результат работы эвристики усеченного ДП; например, для `br17.10.sop` для $H = 1$ в попятном направлении программа, скомпилированная MS VC++, находит значение 63 (не более 15% хуже оптимального 55), а реализация под gcc находит значение 79 (не более 44% хуже оптимального), что дает основания для внедрения вероятностного подхода в усеченное ДП.

В частности, в среде Windows[2] для запроса объема оперативной памяти, выделенной процессу, используются заголовочные файлы `<Windows.h>` и `<Psapi.h>`, предоставляющие желаемую информацию в структуре данных типа `PROCESS_MEMORY_COUNTERS` в качестве элемента `WorkingSetSize` — тогда как в Linux-подобных ОС[3] аналогичные данные, в частности, можно считать из особой файловой системы procfs (строка VmRSS в файле `/proc/self/status`).

Аналогичный прием обеспечивает и совместимость между компиляторами: при форматировании вывода даты и времени через `std::put_time` приходится задействовать классическую библиотеку `<ctime>`; при этом, один и тот же функционал обеспечивает в MS VC++ функция под названием `gmtime_s(tm* foo, const time_t input)`, а в gcc — `gmtime_r(tm* foo, const time_t input)`, соответственно, в зависимости от обнаруженной ОС, используются разные названия функций и порядок аргументов.

Для получения «календарных», абсолютных даты и времени использовалась библиотека `<ctime>`; при этом, относительное время работы алгоритма в экспериментах замерялось средствами библиотеки `std::chrono`; последняя, в отличие от `<ctime>`, позволяет замерять время с более высокой точностью (в комплексе время измеряется с точностью до миллисекунд) платформонезависимым образом; использовался метод `std::chrono::steady_clock::now()`.

Вспомогательные элементы. Обработка символьных строк, потребность в которой возникает при считывании исходных данных, ведении журнала событий, в котором регистрируется начало расчета функции Беллмана на следующем очередном слое пространства состояний и т. п., форматирование выходных данных (найденный маршрут, его стоимость, время вычисления, расход памяти), реализована средствами STL-строк `std::string` и строковых потоков `std::stringstream`, чтение и запись файлов также реализована *потоками* `std::fstream`.

При проведении вычислительного эксперимента, для пакетного запуска комплекса на различных исходных данных и последующего сбора выходных данных использовались стандартная пользовательская оболочка Unix-подобных операционных систем (/bin/sh) и стандартные утилиты (echo, grep, cut и др.).

Элементы C++11. Наиболее важным для нашей реализации, несомненно, являются добавленные в STL в редакции C++11 хеш-таблица `std::unordered_map` и библиотека-«секундомер» `std::chrono`; кроме того, достаточно широко использовались автоматическое определение типов компилятором (ключевое слово `auto`), циклы по элементам контейнера «range-based `for`» и методы `emplace`, создающие элемент непосредственно при занесении его в контейнер — в отличие от методов `insert`, помещающих внутрь контейнера *уже* существующий элемент.

4.2 Функциональные возможности

Комплекс выполняет точное (ДП) и эвристическое (усеченное ДП) решение задачи коммивояжера с условиями предшествования с различными вариантами агрегирования затрат, в частности, арифметическая сумма реализует TSP-PC и ее обобщенные варианты, max реализует BTSP-PC и ее обобщенные варианты; также доступен вариант TD-TSP-PC с зависимостью типа *traveling deliveryman* [74]. Реализована возможность выполнения и точного ДП и усеченного ДП как в прямой (см. гл. 1, ср. [16, 22]), так и в попятной (см.

[2] при работе с MS VC++ обнаруживается директивой `#ifdef _WIN32`

[3] при работе с gcc обнаруживаются директивой `#ifdef __linux__`

гл. 2, ср. [15, 25, 53]) формулировках. Для точного ДП направление, за маловероятным исключением — потребности в «односторонне» ассоциативных и столь же односторонне неубывающих операциях агрегирования $\oplus$ — не имеет значения и является исключительно вопросом удобства реализации специфической операции агрегирования и функций стоимости перемещений, однако, для эвристики *усеченного* ДП выбор направления доставляет дополнительную (к выбору параметра алгоритма H) «степень свободы», способную повлиять на итоговый результат, см. выводы в разделе 1.5.

Функционал для работы с *обобщенными* вариантами задач, в которых рассматриваются кластеры (мегаполисы) из многих городов (см. гл. 2) встроен в машинное представление экземпляров задачи и процедуры решения — во внутреннем представлении «обычные» задачи суть обобщенные с тривиальными мегаполисами, содержащими по одному городу — но в текущей реализации не задействован; вычислительные эксперименты, описанные в гл. 2 проводились на предыдущих версиях комплекса, включая [118].

Универсальность по разновидностям решаемых задач достигается за счет разделения «геометрической» части задачи (расстояние между городами, условия предшествования) и признаков специфической задачи (конкретная разновидность функции агрегирования затрат). Также, благодаря конструкциям ДП из [44, 77], обеспечена «прозрачность по направлению решения»: за исключением начальных условий и определений операторов $\mathbf{E}$ и $\mathbf{I}$, решение как в прямом так и в попятном направлении производит общая кодовая база.

В процессе решения задач, комплекс замеряет истекшее (физическое) время с точностью до 1 мс с помощью метода `steady_clock::now()` из библиотеки C++ `<std::chrono>` и расход оперативной памяти с помощью интерфейса C++ к ОС.

Комплекс считывает исходные данные из файла типа `.sop`, описывающего экземпляр задачи TSP-PC в формате TSPLIB [76]. Считывание исходных данных из других форматов может быть реализовано отдельно.

Для запуска решения задачи используется интерфейс командной строки

```
filename.sop [-H breadth] [-d FWD | BWD] [-t TSP | BTSP | TD_TSPb | TD_TSPf]
```

единственный обязательный аргумент — название файла исходных данных (необязательные элементы обозначены в квадратных скобках);

ключ `-H` указывает на запуск эвристики *усеченного* ДП с соответствующим значением параметра (некоторого натурального числа); отсутствие этого ключа интерпретируется как требование запуска *точного* ДП

ключ `-d` определяет *направление* решения, `FWD` обозначает прямое, а `BWD` — попятное; в отсутствие ключа по умолчанию принимается *попятное* направление счета

ключ `-t` определяет *тип* задачи: функцию агрегирования затрат, функцию стоимости перемещений и т. п. В настоящее время доступны задача коммивояжера с условиями предшествования (тип `TSP`), задача коммивояжера с условиями предшествования на узкие места (тип `BTSP`), задача коммивояжера с условиями предшествования с зависимостью от времени типа traveling deliveryman [74] (типы `TD_TSPb`, `TD_TSPf`); в последнем подсчет стоимости перемещений зависит от направления решения (см. раздел 1.5.4). В отсутствие ключа `-t` по умолчанию предполагается обычная задача коммивояжера с условиями предшествования.

4.3 Состав и устройство программного комплекса

В настоящем разделе на концептуальном уровне описаны составные части программного комплекса, придающие ему гибкость, необходимую для решения разнообразных обоб-

щений TSP-PC, а также удобство доработки и поддержки. Большая часть технических подробностей опущена, и «физическое» разбиение на модули (в виде традиционных для C++ заголовочных файлов `.h` и исходных текстов `.cpp`) не всюду соответствует настоящей схеме.

4.3.1 Внутреннее представление экземпляра задачи

«Геометрия». Данные экземпляра задачи из TSPLIB

1. Число городов, распределение городов по мегаполисам. И города и мегаполисы нумеруются 16-битными целыми числами без знака (`std::uint16_t`); предполагается, что города, принадлежащие одному мегаполису идут по порядку, их индексы образуют некий целочисленный промежуток
2. Расстояние между городами — базовая единица стоимости перемещения между городами, дополнительно модифицируется в задачах с зависимостью от времени или списка невыполненных заданий
3. Условия предшествования
4. Функция: считывание исходных данных экземпляра TSPLIB из файла типа `.sop`

«Направление» решения (прямое/попятное)

1. Функция: оператор $\mathbf{I}$, в прямом направлении $\mathbf{I}[K] = \mathrm{Max}[K]$, в попятном — $\mathbf{I}[K] = \mathrm{Min}[K]$
2. Функция: оператор $\mathbf{E}$, в прямом направлении $\mathbf{E}[K] = \mathrm{Min}[1\,..\,n \setminus K]$, в попятном — $\mathbf{E}[K] = \mathrm{Max}[1\,..\,n \setminus K]$

«Целевая функция»

1. Функция: стоимость перемещения $\mathfrak{c}(x, z_{\mathrm{in}}, z_{\mathrm{out}}, K)$ (в попятном направлении), $\mathfrak{c}(z_{\mathrm{in}}, z_{\mathrm{out}}, x, K)$ (в прямом)
   (a) Функция: внутренние работы $z_{\mathrm{in}} \longrightarrow z_{\mathrm{out}}$
   (b) Функция: внешние перемещения[4] $x \longrightarrow z_{\mathrm{out}}$ (попятное направление), $z_{\mathrm{out}} \longrightarrow x$ (прямое)
   (c) Функция: агрегирование *внутренних* работ и *внешних* перемещений (в главе 2 — обычная арифметическая сумма)
2. Функция: агрегирование затрат $\oplus$

4.3.2 Внутреннее представление процесса решения

- Структура данных: состояния $\mathcal{S}$ и значения функция Беллмана на них.
  Со списком заданий K ассоциированы состояния $(K, x) : x \in \mathbf{E}[x]$ и их стоимости. Внешняя структура данных: хеш-таблица `std::unordered_map`, где ключ — машинное представление списка заданий K в виде `std::bitset`, а значение — ассоциативный массив `std::map`, где ключ — город-«интерфейс» x, а значение — $v(K, x)$. Кроме того, состояния упорядочены по мощности; таким образом, для доступа к значению

[4] используется например, для задания стоимостей типа traveling deliveryman [74]

функции Беллмана $v(K, x)$ на состоянии (K, x), нужно трижды применить оператор *взятия индекса* (subscript operator):

$$\texttt{\_[|K|][K][x]} = v(K, x),$$

где `_` обозначает опущенное имя структуры данных. Формально, тип структуры данных представляет собой

```
std::vector< std::unordered_map< std::bitset, std::map < t_city_tag,
 t_cost > > >
```

здесь `t_city_tag` — номер города (соотв., базы либо терминала — интерфейса состояния) представленный 16-битным целым числом без знака `uint16_t`, а `t_cost` — стоимость перемещений, представленная 32-битным целым числом `int32_t` (в экспериментах на задачах TSPLIB; в экспериментах главы 2 использовались числа с плавающей запятой `float`)

1. Функция: расчет $v(K, x)$ по уравнению Беллмана, из состояний, покрываемых[5] (K, x).
   Реализация инвариантна[6] по
   - точному/усеченному ДП: достигается за счет проверки *существования* покрываемого состояния на предыдущем слое (в случае усеченного ДП *не каждое* покрываемое состояние сохраняется)
   - направлению: достигается использованием *соответствующих* направлению операторов $\mathbf{I}$ и $\mathbf{E}$ (компонент «Направление» в предыдущем разделе) и функции стоимости перемещения (компонент «Целевая функция» в предыдущем разделе)
2. Функция: определение $\operatorname{argmin}_{m \in \mathbf{I}[K]} v(K, x)$, при восстановлении решения по функции Беллмана. Инвариантна, аналогично предыдущему пункту.

- Точное ДП
  Функция: обработка состояний, соответствующих списку заданий K на *текущем* слое
  - расчет $v(K, x)$, где состояния (K, x) сгенерированы при обработке *предыдущего* слоя
  - генерация *следующего* слоя (по схеме $K \cup \{m\}$, где $m \in \mathbf{E}[K]$)
- Усеченное ДП
  Функция: обработка состояний, соответствующих списку заданий K на *текущем* слое
  - Параметр эвристики H, $H \in \mathbb{N}$ — наибольшее число состояний на слое
  - Структура данных: очередь с приоритетами. В очереди стоят состояния *текущего слоя*, упорядоченные[7] по качеству — значению функции Беллмана — так, что *первым* всегда стоит *наихудшее* состояние, для облегчения его изъятия.

[5] см. раздел 1.3.1

[6] то есть, использует один и тот же программный код

[7] отметим, что это *частичный*, не полный порядок: состояния одинаковой стоимости получаются *не*сравнимыми; в текущей реализации в этом случае сохраняется состояние, уже попавшее в очередь. Фактически, этот порядок продолжается до полного *псевдослучайным* образом — в зависимости от того, в каком порядке находятся состояния в структуре данных, описанной выше. Вместо этого представляется возможным *случайно*, например, с вероятностью $\frac{1}{2}$, предпочитать *новое* состояние имеющемуся, что позволит рассматривать разнообразные решения «одинаковой жадности H».

Заключение

Итоги и рекомендации

Точное решение. Динамическое программирование

Исследования [44,77] демонстрируют пригодность рассматриваемого варианта точного ДП как метода решения «сильно нагруженных» (условиями предшествования) задач на примере решения экземпляров из TSPLIB, как в обычной постановке TSP-PC, так и в постановке с зависимостью от времени [74, TD-SOP], и конкурентоспособность с методом программирования в ограничениях (Constraint Programming) [74].

Например, экземпляр p43.4.sop впервые был оптимально решен в [138] за 20 часов и, независимо, в [43] за чуть более чем 4 часа; расход оперативной памяти в этих работах не замерялся; из недавних результатов, в [47] этот экземпляр решался за 88,49 секунд и в [34] за 11 секунд. Посредством ДП, автору удалось решить этот экземпляр менее чем за 1 секунду [77, Table 1] (в среднем по трем запускам 0,665 миллисекунд) с расходом памяти менее 23 МБ, откуда можно заключить что и на менее производительных вычислительных машинах[8] этот экземпляр тоже решался бы достаточно быстро. Из экземпляров, оптимальное решение которых было получено сравнительно недавно, можно отметить ft70.4.sop, впервые решенный в [34] за 249 секунд; наша реализация ДП находит оптимальное решение за 31 секунду с расходом памяти чуть менее 1 ГБ.

Таким образом, можно рекомендовать решать TSP-PC и PSD-TSP-PC посредством ДП согласно критерию, сформулированному в [44,77]:

- если оценка *сверху* двоичного логарифма числа состояний состояний не превосходит 40, экземпляр наверняка удастся разрешить (затратив не более 256 ГБ оперативной памяти);
- если оценка *снизу* двоичного логарифма числа состояний превосходит 40, экземпляр наверняка *не* удастся разрешить.

Отметим ключевые особенности ДП:

1. не требует использование коммерческих пакетов для решения задач линейного программирования (например, IBM ILOG CPLEX)
2. хорошо справляется с дополнительными ограничениями и обобщениями, будь то условия предшествования, группировка городов в кластеры-мегаполисы или зависимость от времени или списка невыполненных заданий
3. более универсально чем другие методы: модель с абстрактной функцией агрегирования позволяет единообразно и с близкой сложностью решать как обычные, аддитивные задачи, так и задачи с агрегированием *на узкие места* (например, BTSP-PC)

[8] вычислительный эксперимент в [77] проводился на достаточно современном на момент написания центральном процессоре Intel Xeon E5-2697 v4

4. ДП *полиномиально* по количеству идеалов, следовательно, (PSD)-TSP-PC *полиномиально* разрешима при полиномиальной зависимости $|\mathcal{I}|$ от количества городов n.

5. При расчетах с *плавающей запятой* нет потребности в генерации $\mathcal{I}$ быстрее чем за $\mathcal{O}(wn)$ на идеал, потому что временна́я сложность доминируется сложностью расчета значений функции Беллмана ($\mathcal{O}(wn)$), где n — количество городов, а w — ширина частичного порядка задающего условия предшествования.

Эвристика. Усеченное динамическое программирование

Эвристика «усеченное ДП» (restricted DP) достаточно эффективна для задач с «сильными» и «средними» условиями предшествования. Зачастую достигает оптимального решения существенно быстрее точного ДП и с сильно меньшим расходом памяти [62, 77].

- Всегда находит *допустимое* решение (важно когда много ограничений)
- Полиномиальна по H и n
- Решение стремится к точному при приближении H к $\operatorname{argmax}_{i\in 0..n}\{|\mathcal{S}_i|\}$

Перспективы дальнейших исследований

Точное решение (PSD)-(B)(G)TSP-PC

Можно предложить два общих подхода к увеличению размерности задач, разрешимых ДП:

экстенсивный Использовать «вертикальную» схему параллелизации с *распределенной* памятью для задействования большего объема, чем доступен на одном (суперкомпьютерном) узле, например, [136, 137]. Вопрос о разложении TSP-PC также поднимался в [104]. Кроме того, возможно, удастся ускорить решение за счет применения более совершенных генераторов порядковых идеалов [122, Appendix A.2.2].

интенсивный Найти эффективную оценку *снизу* и реализовать схему ветвей и границ в динамическом программировании [51]. Этот подход применялся, например, в [31]. Среди оценок снизу особо отметим [138].

Также предполагается распространить метод на задачу об ориентировании (Orienteering Problem, см. обзоры [2, Ch. 13], [159], [160]) с условиями предшествования.

Эвристика «усеченное динамическое программирование»

Представляется возможным получить гарантированные оценки *точности* эвристики, связанные с соотношением параметра эвристики H и мощностью «наиболее населенного» слоя пространства состояний, возможно, на стохастических моделях[9], ср. [161]. Также представляется возможным улучшить работу эвристики, совместив ее со схемой *встречного* ДП [137].

[9] например, по аналогии с оценками для жадного алгоритма в TSP [120, § 4.2]

Оценки сложности экземпляров TSP-PC

Представляется возможным выделить два направления развития:

техническое направление Дополнить оценки [44] верхними оценками из различных вариантов разложения в цепи; получить оценки для библиотеки экземпляров SOPLIB. Получить оценки размерности экземпляров TSP-PC в виде количества допустимых маршрутов (линейных продолжений)[10].

теоретическое направление Определить оптимальное разложение в цепи. Исследовать связь пространственной сложности (имеется в виду количество состояний ДП) и «линейной невязки» (linear discrepancy) [135], либо других теоретико-порядковых характеристик — в дополнение к ширине. Получить оценки на количество теоретико-порядковых идеалов заданной мощности, более легкие в вычислении чем прямое применение алгоритма порождения идеалов заданной мощности [127]. Формально описать верхнюю границу теоретического прироста производительности при использовании «вертикальной» схемы распараллеливания [136].

Создание библиотеки экземпляров GTSP-PC

Единственная на настоящий момент библиотека экземпляров GTSP-PC представлена в [91] и является производной некоторых (необобщенных) экземпляров TSP-PC из TSPLIB. Можно предложить несколько вариантов ее пополнения:

- Кластеризация экземпляров TSP-PC из TSPLIB [76] и SOPLIB [162] с сохранением имеющихся условий предшествования внутри кластеров и выявлением условий предшествования между кластерами исходя из имеющихся.
- Наделение случайными условиями предшествования экземпляров GTSP из библиотеки http://www.cs.nott.ac.uk/~pszdk/gtsp.html. О проблемах при генерации случайных условий предшествования (частичных порядков) см. [163, § 4.5]; подходы описаны, например, в [§ 1.6] [122].

[10] в [44], следуя [22, 23] представлены оценки, определяющие сложность ДП и основанные на количестве состояний ДП, которое, в свою очередь, зависит от количества порядковых идеалов при частичном порядке, задающем условия предшествования. Для оценки трудоемкости методов, пространство решений в которых — допустимые маршруты (некоторые варианты метода ветвей и границ), разумно оценить их количество

Литература

1. Vehicle routing: problems, methods, and applications / Ed. by P. Toth, D. Vigo. 2nd edition. Philadelphia: Math. Opt. Soc, Soc. of Ind. and Appl. Math., 2014.

2. The traveling salesman problem and its variations / Ed. by G. Gutin, A. P. Punnen. Dordrecht: Kluwer Academic Publishers, 2002. Vol. 12 of *Combinatorial optimization*.

3. Bellmore M., Nemhauser G. L. The traveling salesman problem: a survey // Operations Research. 1968. Vol. 16, no. 3. P. 538–558.

4. The traveling salesman problem: a guided tour of combinatorial optimization / Ed. by E. L. Lawler, J. K. Lenstra, A. R. Kan, D. B. Shmoys. New York: Wiley–Interscience, 1985. Vol. 3 of *Wiley–Interscience Series in Discrete Mathematics and Optimization*.

5. Меламед И. И., Сергеев С. И., Сигал И. Х. Задача коммивояжера. Вопросы теории // Автоматика и телемеханика. 1989. № 9. С. 3–33.

6. Меламед И. И., Сергеев С. И., Сигал И. Х. Задача коммивояжера. Точные методы // Автоматика и телемеханика. 1989. № 10. С. 3–29.

7. Меламед И. И., Сергеев С. И., Сигал И. Х. Задача коммивояжера. Приближенные алгоритмы // Автоматика и телемеханика. 1989. № 11. С. 3–26.

8. Laporte G. The traveling salesman problem: An overview of exact and approximate algorithms // European Journal of Operational Research. 1992. Vol. 59, no. 2. P. 231–247.

9. Reinelt G. The traveling salesman: computational solutions for TSP applications. Berlin: Springer-Verlag, 1994.

10. Laporte G., Osman I. H. Routing problems: A bibliography // Annals of Operations Research. 1995. Vol. 61, no. 1. P. 227–262.

11. The traveling salesman problem. A computational study / D. L. Applegate, W. J. Cook, R. E. Bixby, V. Chvátal. Princeton Series in Applied Mathematics. Princeton, NJ: Princeton Univ. Press, 2006.

12. Laporte G., Martín I. R. Locating a cycle in a transportation or a telecommunications network // Networks. 2007. Vol. 50, no. 1. P. 92–108.

13. Gendreau M., Ghiani G., Guerriero E. Time-dependent routing problems: A review // Computers & Operations Research. 2015. Vol. 64. P. 189–197.

14. Dantzig G., Fulkerson R., Johnson S. Solution of a large-scale traveling-salesman problem // Journal of the operations research society of America. 1954. Vol. 2, no. 4. P. 393–410.

15. Bellman R. Dynamic programming treatment of the travelling salesman problem // Journal of the ACM (JACM). 1962. Vol. 9, no. 1. P. 61–63.

16. Held M., Karp R. M. A dynamic programming approach to sequencing problems // Journal of the Society for Industrial & Applied Mathematics. 1962. Vol. 10, no. 1. P. 196–210.

17. An algorithm for the traveling salesman problem / J. D. Little, K. G. Murty, D. W. Sweeney, C. Karel // Operations research. 1963. Vol. 11, no. 6. P. 972–989.

18. Christofides N. The shortest Hamiltonian chain of a graph // SIAM Journal on Applied Mathematics. 1970. Vol. 19, no. 4. P. 689–696.

19. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982.

20. Chisman J. A. The clustered traveling salesman problem // Computers & Operations Research. 1975. Vol. 2, no. 2. P. 115–119.

21. Allahverdi A. The third comprehensive survey on scheduling problems with setup times/costs // European Journal of Operational Research. 2015. Vol. 246, no. 2. P. 345–378.

22. Steiner G. On the complexity of dynamic programming for sequencing problems with precedence constraints // Annals of Operations Research. 1990. Vol. 26, no. 1. P. 103–123.

23. Steiner G. On estimating the number of order ideals in partial orders, with some applications // Journal of statistical planning and inference. 1993. Vol. 34, no. 2. P. 281–290.

24. Escudero L. F. An inexact algorithm for the sequential ordering problem // European Journal of Operational Research. 1988. Vol. 37, no. 2. P. 236–249.

25. Ченцов А. Г. Экстремальные задачи маршрутизации и распределения заданий: вопросы теории. Ижевск: НИЦ «Регулярная и хаотическая динамика», 2008.

26. Ченцов А. А., Ченцов А. Г., Ченцов П. А. Элементы динамического программирования в экстремальных задачах маршрутизации // Проблемы управления. 2013. № 5. С. 12–21.

27. Гретцер Г. Общая теория решеток. М.: Мир, 1982.

28. Aho A. V., Garey M. R., Ullman J. D. The transitive reduction of a directed graph // SIAM Journal on Computing. 1972. Vol. 1, no. 2. P. 131–137.

29. Schmidt G., Ströhlein T. Relations and graphs: discrete mathematics for computer scientists. EATCS Monographs on Theoretical Computer Science. Berlin: Springer-Verlag, 1993.

30. Плотинский Ю. М. Обобщенная задача развозки // Автоматика и телемеханика. 1973. Т. 34, № 6. С. 100–104.

31. The Traveling Salesman Problem with Precedence Constraints / L. Bianco, A. Mingozzi, S. Ricciardelli, M. Spadoni // Papers of the 19th Annual Meeting/Vorträge der 19. Jahrestagung / Springer. 1992. P. 299–306.

32. A cutting plane approach to the sequential ordering problem (with applications to job scheduling in manufacturing) / N. Ascheuer, L. Escudero, M. Grötschel, M. Stoer // SIAM Journal on Optimization. 1993. Vol. 3, no. 1. P. 25–42.

33. Kubo M., Kasugai H. The precedence constrained traveling salesman problem // Journal of the Operations Research Society of Japan. 1991. Vol. 34, no. 2. P. 152–172.

34. Gouveia L., Ruthmair M. Load-dependent and precedence-based models for pickup and delivery problems // Computers & Operations Research. 2015. Vol. 63. P. 56–71.

35. Fiala Timlin M. T., Pulleyblank W. R. Precedence constrained routing and helicopter scheduling: heuristic design // Interfaces. 1992. Vol. 22, no. 3. P. 100–111.

36. Escudero L. F. A production planning problem in FMS // Annals of Operations Research. 1989. Vol. 17, no. 1. P. 69–103.

37. Dubowsky S., Blubaugh T. Planning time-optimal robotic manipulator motions and work places for point-to-point tasks // Robotics and Automation, IEEE Transactions on. 1989. Vol. 5, no. 3. P. 377–381.

38. Spieckermann S., Gutenschwager K., Voß S. A sequential ordering problem in automotive paint shops // International journal of production research. 2004. Vol. 42, no. 9. P. 1865–1878.

39. Петунин А. А. О некоторых стратегиях формирования маршрута инструмента при разработке управляющих программ для машин термической резки материала // Вестник Уфимского государственного авиационного технического университета. 2009. Т. 13, № 2. С. 280–286.

40. Ascheuer N. Hamiltonian path problems in the on-line optimization of flexible manufacturing systems. Ph.D. thesis: Technische Universität Berlin Germany. 1995.

41. Balas E., Fischetti M., Pulleyblank W. R. The precedence-constrained asymmetric traveling salesman polytope // Mathematical programming. 1995. Vol. 68, no. 1-3. P. 241–265.

42. Ascheuer N., Jünger M., Reinelt G. A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints // Computational Optimization and Applications. 2000. Vol. 17, no. 1. P. 61–84.

43. Gouveia L., Pesneau P. On extended formulations for the precedence constrained asymmetric traveling salesman problem // Networks. 2006. Vol. 48, no. 2. P. 77–89.

44. Salii Y. V. Order-theoretic characteristics and dynamic programming for Precedence Constrained Traveling Salesman Problem // Proceedings of the Fourth Russian Finnish Symposium on Discrete Mathematics / Ed. by J. Karhumäki, Y. Matiyasevich, A. Saarela. Vol. 26 of *TUCS Lecture Notes*. Turku Centre for Computer Science, 2017. P. 152–164. URL: http://urn.fi/URN:ISBN:978-952-12-3547-4.

45. Kalantari B., Hill A. V., Arora S. R. An algorithm for the traveling salesman problem with pickup and delivery customers // European Journal of Operational Research. 1985. Vol. 22, no. 3. P. 377–386.

46. Shobaki G., Jamal J. An exact algorithm for the sequential ordering problem and its application to switching energy minimization in compilers // Computational Optimization and Applications. 2015. Vol. 61, no. 2. P. 343–372.

47. Cire A. A., van Hoeve W.-J. Multivalued decision diagrams for sequencing problems // Operations Research. 2013. Vol. 61, no. 6. P. 1411–1428.

48. Decision Diagrams for Optimization / D. Bergman, A. A. Cire, W.-J. van Hoeve, J. N. Hooker. Artificial Intelligence: Foundations, Theory, and Algorithms. Cham: Springer, 2016.

49. Benoist T., Jeanjean A., Jost V. Call-Based Dynamic Programming for the Precedence Constrained Line Traveling Salesman // International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems / Springer. 2014. P. 1–14.

50. Ченцов А. Г., Ченцов П. А. Маршрутная задача с условиями предшествования (задача курьера): метод динамического программирования // Вестн. УГТУ-УПИ. 2004. № 15. С. 148–151.

51. Morin T. L., Marsten R. E. Branch-and-bound strategies for dynamic programming // Operations Research. 1976. Vol. 24, no. 4. P. 611–627.

52. Lawler E. L. Efficient implementation of dynamic programming algorithms for sequencing problems: Tech. Rep.: BW 106/79: Stichting Mathematisch Centrum, 1979.

53. Сесекин А. Н., Ченцов А. А., Ченцов А. Г. Маршрутизация с абстрактной функцией агрегирования стоимостей перемещений // Труды Института математики и механики УрО РАН. 2010. Т. 16, № 3. С. 240–264.

54. Маркелова Е. Ю., Рольщиков В. Е., Ченцов А. Г. Задача маршрутизации конечного числа переходов системы с абстрактной функцией агрегирования затрат. 1998. Деп. ВИНИТИ № 1577-И98.

55. Mitten L. G. Composition principles for synthesis of optimal multistage processes // Operations Research. 1964. Vol. 12, no. 4. P. 610–619.

56. Morin T. L. Monotonicity and the principle of optimality // Journal of Mathematical Analysis and Applications. 1982. Vol. 88, no. 2. P. 665–674.

57. Minoux M. Programmation mathèmatique. Théorie et algorithmes. 2^e edition. Lassay-les-Châteaux: Lavoisier, 2008.

58. Ченцов А. А., Ченцов А. Г. Экстремальная задача маршрутизации “на узкие места” с ограничениями в виде условий предшествования // Тр. Ин-та математики и механики УрО РАН. 2008. Т. 14, № 2. С. 129–142.

59. Сесекин А. Н., Ченцов А. А., Ченцов А. Г. Об одной задаче маршрутизации “на узкие места” // Труды Института математики и механики УрО РАН. 2010. Т. 16, № 1. С. 152–170.

60. Салий Я. В., Ченцов А. Г. Об одной маршрутной задаче на узкие места с внутренними работами // Вестник Тамбовского университета. Серия: Естественные и технические науки. 2012. Т. 17, № 3. С. 827–847.

61. Chentsov A. G., Salii Y. V. A model of “nonadditive” routing problem where the costs depend on the set of pending tasks // Vestnik YuUrGU. Ser. Mat. Model. Progr. 2015. Vol. 8, no. 1. P. 24–45.

62. Salii Y. V. Restricted Dynamic Programming Heuristic for Precedence Constrained Bottleneck Generalized TSP // Proceedings of the 1st Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists / Ed. by A. Sozykin, E. Akimova, D. Ustalov. Vol. 1513. CEUR Workshop Proceedings, 2015. P. 85–108. URL: http://ceur-ws.org/Vol-1513/#paper-10.

63. Kouvelis P., Yu G. Robust discrete optimization and its applications. Dordrecht: Kluwer Academic Publishers, 1997. Vol. 14 of *Nonconvex Optimization and Its Applications*.

64. Serov V. P. Optimal Feedback Strategy in The Game Variant of Generalized Travelling Salesman Problem1 // IFAC Proceedings Volumes. 2000. Vol. 33, no. 16. P. 635–640.

65. Aissi H., Bazgan C., Vanderpooten D. Min–max and min–max regret versions of combinatorial optimization problems: A survey // European Journal of Operational Research. 2009. Vol. 197, no. 2. P. 427–438.

66. On the maximum scatter traveling salesperson problem / E. M. Arkin, Y.-J. Chiang, J. S. Mitchell, S. S. Skiena et al. // SIAM Journal on Computing. 1999. Vol. 29, no. 2. P. 515–544.

67. Psaraftis H. N. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem // Transportation Science. 1980. Vol. 14, no. 2. P. 130–154.

68. Psaraftis H. N. Dynamic vehicle routing: Status and prospects // Annals of Operations Research. 1995. Vol. 61, no. 1. P. 143–164.

69. A review of dynamic vehicle routing problems / V. Pillac, M. Gendreau, C. Guéret, A. L. Medaglia // European Journal of Operational Research. 2013. Vol. 225, no. 1. P. 1–11.

70. Сихарулидзе Г. Г. Об одном обобщении задачи коммивояжера. I // Автоматика и телемеханика. 1971. № 8. С. 116–123.

71. Сергеев С. И. Гибридные системы управления и динамическая задача коммивояжера // Автоматика и телемеханика. 2008. № 1. С. 45–54.

72. Тонков Л. В., Ченцов А. Г. К вопросу оптимального выбора маршрута в условиях временного дисконтирования // Кибернетика и системный анализ. 1999. Т. 35, № 1. С. 95–105.

73. Ченцов А. Г., Ченцов П. А. Об одном нестационарном варианте обобщенной задачи курьера с внутренними работами // Вестник Южно-Уральского государственного университета. Серия «Математическое моделирование и программирование». 2013. Т. 6, № 2. С. 88–107.

74. Kinable J., Cire A. A., van Hoeve W.-J. Hybrid optimization methods for time-dependent sequencing problems // European Journal of Operational Research. 2017. Vol. 259, no. 3. P. 887–897.

75. Щербина О. А. Удовлетворение ограничений и программирование в ограничениях // Интеллектуальные системы. Теория и приложения. 2011. Т. 15, № 1-4. С. 53–170.

76. Reinelt G. TSPLIB: a library of sample instances for the TSP (and related problems) from various sources and of various types. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95. Accessed: 2018-03-01.

77. Salii Y. V. Revisiting Dynamic Programming for Precedence-Constrained Traveling Salesman Problem and Its Time-Dependent Generalization. Accessed: 05.03.2018; second revision submitted to European Journal of Operational Research, under peer review since 05.03.2018. URL: https://www.researchgate.net/publication/316521572.

78. Malandraki C., Dial R. B. A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem // European Journal of Operational Research. 1996. Vol. 90, no. 1. P. 45–55.

79. Koulamas C., Kyparisis G. J. Single-machine scheduling problems with past-sequence-dependent setup times // European Journal of Operational Research. 2008. Vol. 187, no. 3. P. 1045–1049.

80. Alkaya A. F., Duman E. A new generalization of the traveling salesman problem // Appl. Comput. Math. 2010. Vol. 9, no. 2. P. 162–175.

81. Allahverdi A., Gupta J. N., Aldowaisan T. A review of scheduling research involving setup considerations // Omega. 1999. Vol. 27, no. 2. P. 219–239.

82. A survey of scheduling problems with setup times or costs / A. Allahverdi, C. Ng, T. E. Cheng, M. Y. Kovalyov // European Journal of Operational Research. 2008. Vol. 187, no. 3. P. 985–1032.

83. Leon V. J., Peters B. A. Replanning and analysis of partial setup strategies in printed circuit board assembly systems // International Journal of Flexible Manufacturing Systems. 1996. Vol. 8, no. 4. P. 389–411.

84. Lee K., Lei L., Pinedo M. Production scheduling with history-dependent setup times // Naval Research Logistics (NRL). 2012. Vol. 59, no. 1. P. 58–68.

85. Chentsov P. A., Petunin A. A. Tool Routing Problem for CNC Plate Cutting Machines // IFAC-PapersOnLine. 2016. Vol. 49, no. 12. P. 645–650.

86. Ченцов А. Г. К вопросу о маршрутизации комплексов работ // Вестник Удмуртского университета. Математика. Механика. Компьютерные науки. 2013. № 1. С. 59–82.

87. Lokin F. Procedures for travelling salesman problems with additional constraints // European Journal of Operational Research. 1979. Vol. 3, no. 2. P. 135–141.

88. Castelino K., D'Souza R., Wright P. K. Toolpath optimization for minimizing airtime during machining // Journal of Manufacturing Systems. 2003. Vol. 22, no. 3. P. 173–180.

89. Чеблоков И. Б., Ченцов А. Г. Об одной задаче маршрутизации с внутренними работами // Вестник Удмуртского университета. Математика. Механика. Компьютерные науки. 2012. № 1. С. 96–119.

90. Scheduling twin yard cranes in a container block / A. H. Gharehgozli, G. Laporte, Y. Yu, R. de Koster // Transportation Science. 2014. Vol. 49, no. 3. P. 686–705.

91. Salman R. Algorithms for the Precedence Constrained Generalized Travelling Salesperson Problem: Master's thesis: Chalmers University of Technology. University of Gothenburg. 2015.

92. An industrially validated CMM inspection process with sequence constraints / R. Salman, J. S. Carlson, F. Ekstedt, D. Spensieri et al. // Procedia CIRP. 2016. Vol. 44. P. 138–143.

93. Garfinkel R. S., Gilbert K. C. The bottleneck traveling salesman problem: Algorithms and probabilistic analysis // Journal of the ACM (JACM). 1978. Vol. 25, no. 3. P. 435–448.

94. LaRusic John, Punnen Abraham P. The asymmetric bottleneck traveling salesman problem: Algorithms, complexity and empirical analysis // Computers & Operations Research. 2014. Т. 43. С. 20–35.

95. Коротаева Л. Н., Ченцов А. Г. Об одном обобщении задачи коммивояжера "на узкие места" // Журнал вычислительной математики и математической физики. 1995. Т. 35, № 7. С. 1067–1076.

96. Коротаева Л. Н., Сесекин А. Н., Ченцов А. Г. Об одной модификации метода динамического программирования в задаче последовательного сближения // Журнал вычислительной математики и математической физики. 1989. Т. 29, № 8. С. 1107–1113.

97. Arkin E. M., Hassin R. Approximation algorithms for the geometric covering salesman problem // Discrete Applied Mathematics. 1994. Vol. 55, no. 3. P. 197–218.

98. Alatartsev S., Stellmacher S., Ortmeier F. Robotic Task Sequencing Problem: A Survey // Journal of Intelligent & Robotic Systems. 2015. P. 1–20.

99. Коротаева Л. Н., Трухин М. П., Ченцов А. Г. К вопросу о маршрутизации соединений // Автоматика и телемеханика. 1997. № 12. С. 175–192.

100. Салий Я. В. Влияние условий предшествования на вычислительную сложность решения маршрутных задач методом динамического программирования // Вестник Удмуртского университета. Математика. Механика. Компьютерные науки. 2014. № 1. С. 76–86.

101. Brightwell G., Winkler P. Counting linear extensions // Order. 1991. Vol. 8, no. 3. P. 225–242.

102. Mastor A. A. An experimental investigation and comparative evaluation of production line balancing techniques // Management Science. 1970. Vol. 16, no. 11. P. 728–746.

103. Montemanni R., Smith D. H., Gambardella L. M. Ant colony systems for large sequential ordering problems // Swarm Intelligence Symposium, 2007. SIS 2007. IEEE / IEEE. 2007. P. 60–67.

104. A comparison of two exact algorithms for the sequential ordering problem / V. Papapanagiotou, J. Jamal, R. Montemanni, G. Shobaki et al. // Systems, Process and Control (ICSPC), 2015 IEEE Conference on / IEEE. 2015. P. 73–78.

105. Григорьев А. М., Иванко Е. Е., Ченцов А. Г. Динамическое программирование в обобщенной задаче курьера с внутренними работами: элементы параллельной структуры // Моделирование и анализ информационных систем. 2011. Т. 18, № 3. С. 101–124.

106. Provan J. S., Ball M. O. The complexity of counting cuts and of computing the probability that a graph is connected // SIAM Journal on Computing. 1983. Vol. 12, no. 4. P. 777–788.

107. Balas E., Simonetti N. Linear time dynamic-programming algorithms for new classes of restricted TSPs: A computational study // INFORMS Journal on Computing. 2001. Vol. 13, no. 1. P. 56–75.

108. Chentsov A., Khachay M., Khachay D. Linear time algorithm for Precedence Constrained Asymmetric Generalized Traveling Salesman Problem // IFAC-PapersOnLine. 2016. Vol. 49, no. 12. P. 651–655.

109. Leclerc B. Description combinatoire des ultramétriques // Mathématiques et Sciences humaines. 1981. Vol. 73. P. 5–37.

110. Reinelt G. TSPLIB—A traveling salesman problem library // ORSA journal on computing. 1991. Vol. 3, no. 4. P. 376–384.

111. Alkaya A. F., Duman E. Combining and solving sequence dependent traveling salesman and quadratic assignment problems in PCB assembly // Discrete Applied Mathematics. 2015. Vol. 192. P. 2–16.

112. Салий Я. В., Ченцов А. Г. О маршрутной задаче на узкие места с внутренними работами и условиями предшествования // Международная конференция «Дискретная оптимизация и исследование операций»: Материалы конференции (Новосибирск, 24 – 28 июня 2013 г.) / ИМ СО РАН, НГУ. 2013. С. 134.

113. Ченцов А. Г., Салий Я. В. Неаддитивная задача маршрутизации с условиями предшествования // Алгоритмический анализ неустойчивых задач: тез. докл. Всерос. конф. с междунар. участием, посвящ. памяти В. К. Иванова, Челябинск, 10 – 14 нояб. 2014 г. / Южно-Уральский государственный Университет; ИММ УрО РАН; УрФУ. 2014. С. 166–167.

114. Салий Я. В. Об ультраметрической задаче коммивояжера на узкие места // Современные проблемы математики. Тезисы Международной (43-й Всероссийской) молодёжной школы-конференции / ИММ УрО РАН. 2012. С. 287–289.

115. Hernández-Pérez H., Salazar-González J.-J. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery // Discrete Applied Mathematics. 2004. Vol. 145, no. 1. P. 126–139.

116. Salii Y. V., Ivanko E. E. Dynamic programming and greedy heuristic in Camera Trap Traveling Salesman Problem // Современные проблемы математики и ее приложений. 2016. P. 215–219. URL: http://ceur-ws.org/Vol-1662/.

117. Салий Я. В. Свидетельство о государственной регистрации программы для ЭВМ № 2018614740 «Модульная программа вычисления оптимальных и эвристических решений задачи курьера и ее обобщений методом динамического программирования». Федеральная служба по интеллектуальной собственности (Роспатент). Зарегистрирована 17.04.2018.

118. Салий Я. В. Свидетельство о государственной регистрации программы для ЭВМ № 2015661435 «Программа вычисления оптимального решения обобщенной задачи курьера на узкие места с условиями предшествования и зависимостью от списка невыполненных заданий методом динамического программирования». Федеральная служба по интеллектуальной собственности (Роспатент). Зарегистрирована 27.10.2015.

119. Куратовский К., Мостовский А. Теория множеств. М.: Мир, 1970.

120. Гимади Э. Х., Хачай М. Ю. Экстремальные задачи на множествах перестановок. Екатеринбург: Издательство УМЦ УПИ, 2016.

121. Schröder B. S. W. Ordered sets. Boston: Birkhäuser, 2003.

122. Caspard N., Leclerc B., Monjardet B. Finite ordered sets: concepts, results and uses. Encyclopedia of Mathematics and Its Applications no. 144. Cambridge: Cambridge University Press, 2012.

123. Бурбаки Н. Общая топология. Основные структуры. М.: Наука, 1968.

124. Engelking R. General Topology. Berlin: Heldermann Verlag, 1986. Vol. 6 of *Sigma Series in Pure Mathematics*.

125. zu Siederdissen C. H., Prohaska S. J., Stadler P. F. Dynamic programming for set data types // Brazilian Symposium on Bioinformatics / Springer. 2014. P. 57–64.

126. Restricted dynamic programming: a flexible framework for solving realistic VRPs / J. Gromicho, J. J. van Hoorn, A. Kok, J. Schutten // Computers & Operations Research. 2012. Vol. 39, no. 5. P. 902–909.

127. Wild M. Output-polynomial enumeration of all fixed-cardinality ideals of a poset, respectively all fixed-cardinality subtrees of a tree // Order. 2014. Vol. 31, no. 1. P. 121–135.

128. Kao E. P., Queyranne M. On dynamic programming methods for assembly line balancing // Operations Research. 1982. Vol. 30, no. 2. P. 375–390.

129. Introduction to algorithms / T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. 3rd edition. Cambridge, MA: MIT press, 2009.

130. Working Draft, Standard for Programming Language C++ (C++14): Tech. Rep.: N4140 / Ed. by R. Smith: International Standards Organization, 2014. 11. ISO C++ Working Group 21. URL: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4296.pdf.

131. Dilworth R. P. A decomposition theorem for partially ordered sets // Annals of Mathematics. 1950. P. 161–166.

132. Steiner G. Single machine scheduling with precedence constraints of dimension 2 // Mathematics of Operations Research. 1984. Vol. 9, no. 2. P. 248–259.

133. Kunen K. Set Theory. Revised edition. Milton Keynes: Lightning Source, 2013. Vol. 23 of *Studies in Logic*.

134. Berghammer R. A functional, successor list based version of Warshall's algorithm with applications // International Conference on Relational and Algebraic Methods in Computer Science / Springer. 2011. P. 109–124.

135. Tanenbaum P. J., Trenk A. N., Fishburn P. C. Linear discrepancy and weak discrepancy of partially ordered sets // Order. 2001. Vol. 18, no. 3. P. 201–225.

136. Chentsov A. G., Grigoryev A. M. A Scheme of Independent Calculations in a Precedence Constrained Routing Problem // International Conference on Discrete Optimization and Operations Research / Springer. 2016. P. 121–135.

137. Салий Я. В. О прямом и попятном динамическом программировании в маршрутных задачах с условиями предшествования и алгоритмах генерации допустимых подзадач // Тезисы докладов XV Всероссийской конференции «Математическое программирование и приложения» / ИММ УрО РАН, УрФУ. Информационный бюллетень Ассоциации математического программирования № 13. 2015. С. 168–169.

138. Hernádvölgyi I. T. Solving the sequential ordering problem with automatically generated lower bounds // Operations Research Proceedings 2003. Springer, 2004. P. 355–362.

139. Sesekin A. N., Chentsov A. A., Chentsov A. G. A generalized courier problem with the cost function depending on the list of tasks // Journal of Computer and Systems Sciences International. 2010. Vol. 49, no. 2. P. 234–243.

140. Dolgui A., Pashkevich A. Cluster-level operations planning for the out-of-position robotic arc-welding // International Journal of Production Research. 2006. Vol. 44, no. 4. P. 675–702.

141. Петунин А. А., Ченцов А. Г., Ченцов П. А. К вопросу о маршрутизации движения инструмента в машинах листовой резки с числовым программным управлением // Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление. 2013. Т. 169. С. 103–111.

142. Dewil R., Vansteenwegen P., Cattrysse D. Sheet Metal Laser Cutting Tool Path Generation: Dealing with Overlooked Problem Aspects // Key Engineering Materials / Trans Tech Publ. Vol. 639. 2015. P. 517–524.

143. Методы маршрутизации и их приложения в задачах повышения безопасности и эффективности эксплуатации атомных станций / В. В. Коробкин, А. Н. Сесекин, О. Л. Ташлыков, А. Г. Ченцов. М.: Новые технологии, 2012.

144. Ченцов А. А., Ченцов А. Г., Ченцов П. А. Экстремальная задача маршрутизации с внутренними потерями // Тр. Ин-та математики и механики УрО РАН. 2008. Т. 14, № 3. С. 183–201.

145. Schneider M., Stenger A., Goeke D. The electric vehicle-routing problem with time windows and recharging stations // Transportation Science. 2014. Vol. 48, no. 4. P. 500–520.

146. Chentsov A. A., Chentsov A. G., Chentsov P. A. Elements of dynamic programming in extremal routing problems // Automation and Remote Control. 2014. Vol. 75, no. 3. P. 537–550.

147. Кошелев Г. Н., Кошелева М. С. Параллельная реализация метода динамического программирования в задаче маршрутизации с ограничениями // Современные проблемы математики и её приложений: 46-я Междунар. мол. шк.-конф.: труды / ИММ УрО РАН; УрФУ. 2015. С. 110–115.

148. Ченцов А. Г. Одна параллельная процедура построения функции Беллмана в обобщенной задаче курьера с внутренними работами // Autom. Remote Control. 2012. № 3. С. 134–149.

149. Иванко Е. Е. Усеченный метод динамического программирования в замкнутой задаче коммивояжера с симметричной функцией стоимости // Труды ИММ УрО РАН. 2013. Т. 19, № 1. С. 121–129.

150. Schrage L., Baker K. R. Dynamic programming solution of sequencing problems with precedence constraints // Operations Research. 1978. Vol. 26, no. 3. P. 444–449.

151. Иванко Е. Е. Динамическое программирование в задаче перестановки однотипных объектов // Труды Института математики и механики УрО РАН. 2013. Т. 19, № 4. С. 125–130.

152. Labelle J., Yeh Y. N. Generalized Dyck paths // Discrete Mathematics. 1990. Vol. 82, no. 1. P. 1–6.

153. Сергеев С. И. Алгоритмы решения минимаксной задачи коммивояжера. I. Подход на основе динамического программирования // Автоматика и телемеханика. 1995. № 7. С. 144–150.

154. Хренников А. Ю. Неархимедов анализ и его приложения. М.: ФИЗМАТЛИТ, 2003. С. 216.

155. Владимиров В. С., Волович И. В., Зеленов Е. И. p-Адический анализ и математическая физика. М.: Физматлит, 1994.

156. Миссаров М. Д., Степанов Р. Г. О задачах комбинаторной оптимизации в ультраметричных пространствах // Теоретическая и математическая физика. 2003. Т. 136, № 1. С. 164–176.

157. Адигеев М. Г. О полиномиальной разрешимости ультраметрических версий некоторых NP-трудных задач // Информатика и её применения. 2014. Т. 8, № 2. С. 70–76.

158. Schikhof W. H. Ultrametric Calculus: an introduction to p-adic analysis. Cambridge: Cambridge University Press, 2007. Vol. 4 of *Cambridge Studies in Advanced Mathematics*.

159. Feillet D., Dejax P., Gendreau M. Traveling salesman problems with profits // Transportation Science. 2005. Vol. 39, no. 2. P. 188–205.

160. Gunawan A., Lau H. C., Vansteenwegen P. Orienteering problem: A survey of recent variants, solution approaches and applications // European Journal of Operational Research. 2016. Vol. 255, no. 2. P. 315–332.

161. Rosenkrantz D. J., Stearns R. E., Lewis P. M. An Analysis of Several Heuristics for the Traveling Salesman Problem // SIAM Journal on Computing. 1977. Vol. 6, no. 3. P. 563–581.

162. Montemanni R., Smith D. H., Gambardella L. M. A heuristic manipulation technique for the sequential ordering problem // Computers & Operations Research. 2008. Vol. 35, no. 12. P. 3931–3944.

163. Rardin R. L., Uzsoy R. Experimental evaluation of heuristic optimization algorithms: A tutorial // Journal of Heuristics. 2001. Vol. 7, no. 3. P. 261–304.