МАТЕМАТИКА 2014. Вып. 1

УДК 519.854.1, 519.854.2

© *Я. В. Салий*

ВЛИЯНИЕ УСЛОВИЙ ПРЕДШЕСТВОВАНИЯ НА ВЫЧИСЛИТЕЛЬНУЮ СЛОЖНОСТЬ РЕШЕНИЯ МАРШРУТНЫХ ЗАДАЧ МЕТОДОМ ДИНАМИЧЕСКОГО ПРОГРАММИРОВАНИЯ

В статье рассматривается общий случай маршрутной задачи дискретной оптимизации, осложненной условиями предшествования; изучается влияние условий предшествования на вычислительную сложность решений таких задач методом динамического программирования. Особенность применяемого метода динамического программирования заключается в его «экономичности»: подзадачи, не соблюдающие условия предшествования и, следовательно, не участвующие в оптимальном решении, не рассматриваются, что позволяет сберечь и вычислительную мощность, и память.

Этот метод с 2004 года используется А. Г. Ченцовым и его соавторами, но степень экономии ресурсов исследовалось мало. Мы предлагаем подход к решению этой проблемы, основанный на комбинаторном анализе числа подзадач, существенных в смысле условий предшествования. Применяя известные комбинаторные правила *сложения* и *произведения*, мы получили результат для важных частных случаев условий предшествования: а) «независимые» наборы условий предшествования; б) «цепь» условий предшествования — когда условия задают линейный порядок; в) случай, когда в графе предшествования нет неориентированных циклов, и исходящая степень любой вершины не превышает единицы. Последний случай представляет собой условия предшествования, встречающихся в практической задаче маршрутизации движений инструмента в машинах листовой резки и соответствует требованию вырезать внутренний контур прежде внешнего.

В связи с более сложной структурой случая в) по сравнению с остальными для него вместо аналитической формулы представлен алгоритм; алгоритм реализован на языке C++, зависимость его вычислительной сложности от числа связанных условиями предшествования объектов имеет не более чем квадратичный порядок. В дальнейшем мы предполагаем расширить область применения нашего подхода до более общих вариантов условий предшествования. Отметим также, что наш подход не зависит от критерия оптимальности, соответственно, может применяться для анализа сложности решения методом динамического программирования в произвольных маршрутных задачах с условиями предшествования.

Ключевые слова: условия предшествования, динамическое программирование, вычислительная сложность, маршрутные задачи, листовая резка.

Введение

Во многих практических маршрутных задачах приходится учитывать *условия предшествования* — требование посетить одни пункты ранее других. Такова, например, родственная классической задаче коммивояжера *задача курьера*[1]: посетить N городов по одному и только одному разу, совершив доставку согласно списку *адресных пар* $K \subset \overline{1,N}\times\overline{1,N}$, где в каждой паре $z \in K$ первый компонент $\mathrm{pr}_1(z)$ считается *отправителем*, а второй $\mathrm{pr}_2(z)$ — *получателем* и отправитель должен непременно посещаться перед получателем — считается, что он передает курьеру посылку, которую тот затем доставляет получателю, когда посещает его.

Одним из точных методов решения таких задач является предложенный Р. Беллманом *метод динамического программирования* (далее МДП), суть которого состоит в решении исходной задачи посредством решения нескольких менее трудных задач, в дискретной математике это обычно задачи меньшей размерности; МДП предложен Беллманом в работе [3], в применении к *задаче коммивояжера* — Беллманом в [4] и, независимо, Хелдом и Карпом в [5].

[1] В англоязычной литературе обычно Precedence Constrained TSP или Sequential Ordering Problem; последнее встречается в контексте теории расписаний.

Классический МДП не приспособлен для работы с ограничениями, так что от ограничений, включая условия предшествования, стараются в той или иной форме избавится, однако А.Г. Ченцову удалось сформулировать условие *допустимости* частичных маршрутов (подход описан в 2004 году в [6], полный обзор см. в [2]) — маршрутов в подзадачах[2], где требуется обходить не все пункты из полной, исходной задачи — таким образом, что оптимальное решение всех таких подзадач дает оптимальное решение полной задачи, соблюдающее условия предшествования, и при этом обходится без исследования *несущественных* (с точки зрения условий предшествования) подзадач, где *несущественность* понимается как отсутствие влияния решения подзадачи на оптимальный маршрут полной задачи. Это существенно сокращает количество подзадач, которые необходимо исследовать. Очевидно, что меньшее число подзадач ведет к меньшей вычислительной сложности решения полной задачи.

Точная оценка сокращения числа подзадач неизвестна, единственная попытка была предпринята в статье [7], где было рассчитано влияние единственного условия предшествования. В нашей статье предложен метод аналитического расчета степени сокращения перебора, позволяющий в случаях «простых» условий предшествования узнать, насколько сокращается число подзадач, не генерируя подзадачи непосредственно. Этот метод представляется ценным для теоретического исследования сложности решения задач с условиями предшествования методом динамического программирования; на практике же проще сгенерировать все допустимые подзадачи с помощью компьютерной программы, подсчитать их количество и сравнить с «полным» ($\sum\limits_{i=0}^{N}\left[(N-i)\cdot \mathrm{C}_N^{N-i}\right]$, где N — число пунктов; см. [7, раздел 5]), что связано с тем, что размерность задач, которые сто́ит решать посредством МДП, невелика. Отметим, что в контексте МДП число подзадач, которые необходимо решить, может оказаться едва ли не лучшим мерилом сложности вычислений, чем сама вычислительная сложность — число операций, поскольку число подзадач определяет объем памяти, необходимой для хранения массива значений функции Беллмана, а именно последний зачастую является ограничительным фактором в возможности решения задачи с помощью МДП («насыщение» объема памяти наступает раньше «насыщения» вычислительной мощности). Таким образом, исследование числа подзадач позволяет определить саму *возможность* разрешения задачи.

§ 1. Основные обозначения и определения

Всюду в дальнейшем $\triangleq$ обозначает равенство по определению. *Маршрутной задачей* назовем задачу, где требуется посетить N пунктов, пронумерованных числами из $\overline{1,N}$, по одному и только одному разу, начав обход с *базы*, пронумерованной 0; решением задачи является *маршрут* — перестановка $\alpha\in\mathbb{P}$ натуральных чисел $\overline{1,N}$ — номеров пунктов (где $\mathbb{P}$ — множество всех перестановок $\overline{1,N}$, см. [1, Приложение С.2]). *Оптимальным маршрутом* называется маршрут $\alpha\in\mathbb{P}$, доставляющий экстремум соответствующему критерию оптимальности; характер критерия не влияет на наши построения, поэтому мы не будем его вводить; достаточно, чтобы для критерия выполнялся принцип оптимальности Беллмана (см. [3]), то есть чтобы оптимум полной задачи мог быть получен из оптимумов подзадач меньшей размерности. *Подзадачей* назовем задачу, где требуется посетить не все пункты из $\overline{1,N}$, а только их часть — множество $K, K\subset\overline{1,N}$, называемое *списком заданий*, начав обход из пункта $x\in(\overline{1,N}\setminus K)$, доставив экстремум критерию оптимальности; для обозначения подзадач мы используем упорядоченные пары вида (x,K). Решением подзадачи будет оптимальный *частичный маршрут* — перестановка номеров пунктов, содержащихся в K.

Условия предшествования задаются упорядоченными парами $z\in\mathbb{K}\triangleq\overline{1,N}\times\overline{1,N}$, где $z=(\mathrm{pr}_1(z),\mathrm{pr}_2(z))$, а $\mathbb{K}$ — множество заданных упорядоченных пар. Первый компонент $\mathrm{pr}_1(z)$ пары z называется *отправителем*, а второй $\mathrm{pr}_2(z)$ — *получателем*. Отправитель должен непременно посещаться ранее получателя. Чтобы выделить упорядоченные пары, задающие условия предшествования, будем называть их *адресными парами*. «Соблюдение» условий предшествования описывается следующим образом: маршрут, соблюдающий условия предшествования,

[2] «Укороченных задачах» в терминологии [2].

называется *допустимым* маршрутом; $\{\alpha \in \mathbb{P} : \forall z \in \mathbb{K}\left(\alpha^{-1}(\mathrm{pr}_1(z)) < \alpha^{-1}(\mathrm{pr}_2(z))\right)\}$ есть множество всех таких маршрутов. На условия предшествования мы накладываем требование их «выполнимости», отсутствия «порочных кругов»: для всякого непустого множества $\mathbb{K}_0 \subset \mathbb{K}$, непременно $\exists z_0 \in \mathbb{K}_0 : \mathrm{pr}_1(z_0) \neq \mathrm{pr}_2(z)\ \forall z \in \mathbb{K}$. Тогда $\mathbb{A} \neq \varnothing$ (см. [2, часть 2]), то есть допустимые маршруты существуют. В задачах, осложненных условиями предшествования, *оптимальный* маршрут выбирается из множества *допустимых* $\mathbb{A}$, не из всего $\mathbb{P}$.

О том, как распространить условия предшествования на подзадачи, подробно см. в [2]. Мы воспользуемся определенным там понятием *существенного списка*; число этих списков мы и будем подсчитывать. Итак, пусть $\mathfrak{N}$ — семейство всех подмножеств $\overline{1,N}$ (читай — множество всех списков заданий) и для $K \in \mathfrak{N}$ мощность K обозначается $|K|$. Ранжируем списки заданий по мощности: $\mathfrak{N}_s \triangleq \{K \in \mathfrak{N} | s = |K|\} \forall s \in \overline{1,N}$. Отметим, что сюда входят и «пустой» список $\mathfrak{N}_0 = \{\varnothing\}$, и «полный» $\mathfrak{N}_N = \{\overline{1,N}\}$. Выделим и ранжируем по размерности *существенные* списки — те, которые в некотором собственном смысле соблюдают условия предшествования:

$$\begin{aligned} \mathcal{C}_s \triangleq \{K \in \mathfrak{N}_s | \forall z \in \mathbb{K}\ (\mathrm{pr}_1(z) \notin K) \vee (\mathrm{pr}_2(z) \in K)\} = \\ = \{K \in \mathfrak{N}_s | \forall z \in \mathbb{K}\ (\mathrm{pr}_1(z) \in K) \Rightarrow (\mathrm{pr}_2(z) \in K)\} \quad \forall s \in \overline{1,N}. \end{aligned} \tag{1}$$

То есть мы не допускаем случая, когда в списке заданий присутствует отправитель некоторой пары, но нет его получателя, тогда как «ненасыщенные» получатели могут присутствовать; доказано [2, часть 2], что решением всех существенных в этом смысле подзадач исчерпываются решения полной задачи.

В определении (1) *существенность* списка вводится в отношении множества всех заданных адресных пар K. Нам также понадобятся «сужения» условий предшествования. Назовем список заданий K *существенным в* $\mathbb{K}_1 \subset \mathbb{K}$, где $\mathbb{K}_1$ — собственное подмножество $\mathbb{K}$, если K остается существенным при замене в (1) $\mathbb{K}$ на $\mathbb{K}_1$.

Мы будем представлять список заданий в виде его «битовой маски» — N битов, пронумерованных слева направо с 1 по N, где 1 на i-м месте означает, что i-й пункт входит в список заданий; 0 соответственно означает, что не входит. Таким образом, полный список заданий и тривиальный список заданий (пустой список заданий — мы начинаем построение функции Беллмана с соответствующих ему подзадач) кодируются следующим образом:

$$\underbrace{11\ldots1}_{N \text{ единиц}}\ \underbrace{00\ldots0}_{N \text{ нулей}}.$$

Назовем *состоянием* множества пунктов $K_0 \subseteq \overline{1,N}$ его произвольное подмножество $K_1 \subseteq K_0$, кодированное способом, описанным выше, с дополнительным условием: на местах, соответствующих элементам $K_0 \backslash K_1$, в коде *состояния* стоят нули. Назовем состояние *допустимым*, если оно является *существенным списком* (см. (1)). Состояния множества пунктов суть списки заданий мощности от 0 до числа пунктов в этом множестве, которые можно составить из элементов этого множества и только из них. Поскольку далее мы подсчитываем число именно *допустимых* состояний, мы не будем отдельно оговаривать, что упоминаемое нами состояние *допустимо*. То есть любое упомянутое ниже состояние *допустимо*, если в явном виде не указано обратное. Мы также будем по привычке называть *допустимые* состояния и *существенными*, не различая эти термины.

В работе [5] найдена сложность решения замкнутой задачи коммивояжера методом динамического программирования. Считая существенными операции сложения и сравнения, авторы получают $(N-1)(N-2)2^{N-3} + (N-1)$ элементарных операций, где N — число городов.

Сущность нашего подхода заключается в изучении зависимости числа *существенных списков* от определяющих их *условий предшествования*.

Предположение 1. Мы полагаем отношение *числа существенных списков* заданий к *числу всех списков* заданий хорошей оценкой снижения вычислительной сложности МДП под действием условий предшествования.

Отметим, что при наличии условий предшествования существенные списки становятся еще более неравноценны по сложности вычисления: разным существенным спискам одной мощности может соответствовать неодинаковое число подзадач — не каждый пункт можно добавить к существенному списку, сохранив существенность; мы не можем добавлять «ненасыщенных» отправителей, тех, чьих получателей в списке еще нет. А добавляем мы к списку K новые пункты именно *с начала*, для добавления очередного пункта $x, x \in \overline{1,N} \setminus K$, необходимо сначала разрешить подзадачу (x, K), то есть посчитать стоимость обхода K из пункта x, что делается перебором оптимальных решений подзадач $(j, K \setminus \{j\})$, где $j \in K$ и $\forall z \in \mathbb{K}\,(\mathrm{pr}_1(z) \in K \Rightarrow j \neq \mathrm{pr}_2(z))$ (если пункт j является получателем и его отправитель присутствует в K, то мы не можем начать обход $\{K \setminus \{j\}\}$ с j) и учетом[3] стоимости перехода из x в j.

§ 2. Расчет числа существенных списков

В этом разделе мы укажем, сколько списков заданий позволяют «сэкономить» условия предшествования.

В отсутствие условий имеем 2^N списков заданий (включая тривиальный и полный): каждый пункт может входить в список (тогда на соответствующем месте в коде списка стоит «1») или не входить (тогда в коде на его месте стоит «0»), откуда по правилу произведения получаем 2^N, что соответствует числу подмножеств $\overline{1,N}$. В отсутствие условий предшествования списки заданий — произвольные подмножества множества всех пунктов (включая несобственные — пустое и множество всех пунктов, соответствующее полной задаче). Выясним, сколько списков останутся существенными в присутствии условий предшествования разного вида.

Будем называть *свободными* пункты, не ограниченные условиями предшествования, то есть те, которые ни в одной адресной паре не являются ни отправителем, ни получателем; остальные будем называть *связанными*. Для удобства обозначений в коде списков заданий мы всегда будем писать связанные пункты первыми, это не ограничивает общности, так как их всегда можно перенумеровать. Пункты, связанные произвольным множеством адресных пар $\mathbb{K}_0, \mathbb{K}_0 \subset \mathbb{K}$, объединим в множество $\widehat{\mathbb{K}}_0$: по определению, положим $\widehat{\mathbb{K}}_0 \triangleq \bigcup_{z\in\mathbb{K}_0}\{\mathrm{pr}_1(z) \cup \mathrm{pr}_2(z)\}$. Множеством *всех* связанных пунктов тогда будет $\widehat{\mathbb{K}} = \bigcup_{z\in\mathbb{K}}\{\mathrm{pr}_1(z) \cup \mathrm{pr}_2(z)\}$. Множество всех свободных пунктов обозначим $\widehat{\mathbb{K}}' \triangleq \overline{1,N} \setminus \widehat{\mathbb{K}}$.

Начнем с простейшего варианта — единственной адресной пары $(a, b), a \in \overline{1,N}, b \in \overline{1,N}$. Список заданий не будет *существенным*, если отправитель адресной пары входит в список заданий, а получатель — нет. То есть из четырех состояний $\overline{1,N}$, связанных с наличием/отсутствием (a) и (b) в списке заданий: $00{*}\cdots{*}$, $01{*}\cdots{*}$, $10{*}\cdots{*}$, $11{*}\cdots{*}$, где $*$ обозначает возможность как 0, так и 1 в соответствующей позиции, нам не подходят только состояния $10{*}\cdots{*}$. Этих *недопустимых* состояний, очевидно, 2^{N-2} — первые две позиции фиксированы, для остальных $N-2$ пунктов возможны оба варианта; очевидно, что это множество состояний, благодаря первым двум позициям, не может пересекаться с остальными тремя. Таким образом, из 2^N существенных списков остается $2^N - 2^{N-2} = \frac{3}{4}2^N$. Всего лишь одно условие предшествования сокращает их число уже на четверть!

Рассмотрим теперь «независимые условия предшествования».

Предложение 1. *Пусть $\mathbb{K}$ можно разбить на два множества адресных пар $\mathbb{K}_1$ и $\mathbb{K}_2$ так, что ни один пункт из тех, что присутствуют в $\mathbb{K}_1$, не встречается в $\mathbb{K}_2$, и наоборот: $\widehat{\mathbb{K}}_1 \cap \widehat{\mathbb{K}}_2 = \varnothing$. Пусть общее число связанных пунктов $|\mathbb{K}| = k = k_1 + k_2$, где $k_1 = |\widehat{\mathbb{K}}_1|$ и $k_2 = |\widehat{\mathbb{K}}_2|$, и пусть нам известно число a_1 состояний $\widehat{\mathbb{K}}_1$, существенных в $\mathbb{K}_1$, и число a_2 состояний $\widehat{\mathbb{K}}_2$, существенных в $\mathbb{K}_2$. Тогда общее число существенных списков будет $a_1 \cdot a_2 \cdot 2^{N-k}$.*

Доказательство. Дополним схему кодирования подзадач следующим соглашением: пункты из $\widehat{\mathbb{K}}_1$ будем писать первыми слева, за ними — пункты из $\widehat{\mathbb{K}}_2$ и далее — свободные;

[3] «Учет» здесь означает соответствующее критерию агрегирование затрат на обход $K \setminus \{j\}$ и переход из x в j; при аддитивном агрегировании затрат это будет просто сложение стоимости подзадачи $(j, K \setminus \{j\})$ и стоимости перехода из x в j.

в качестве разделителя используем символ «|». Тогда код произвольного состояния $\overline{1,N}$ фактически распадается на три позиции, первая из которых кодирует состояние $\widehat{\mathbb{K}}_1$; всего их, по условию, a_1, вторая — состояние $\widehat{\mathbb{K}}_2$, по условию их a_2, а третья — состояние множества *свободных* пунктов $\widehat{\mathbb{K}}'$, их получается 2^{N-k} — любая комбинация этих пунктов допустима:

$$\text{состояние } \widehat{\mathbb{K}}_1 \,|\, \text{состояние } \widehat{\mathbb{K}}_2 \,|\, \text{состояние } \widehat{\mathbb{K}}'.$$

Существенные списки исчерпываются всевозможными комбинациями этих состояний, число которых по правилу умножения (см., например, [1, Приложение С.1]) — произведение числа всех возможных состояний каждой из позиций, то есть $a_1 \cdot a_2 \cdot 2^{N-k}$, что и требовалось доказать. Таким образом, остается $\frac{a_1 \cdot a_2}{2^k}$ от исходного числа задач. □

Следствие 1. *Если условия предшествования представлены m «непересекающимися» адресными парами, то есть ни один пункт не входит более чем в одну адресную пару,*

$$\forall z \in \mathbb{K} \forall z_0 \in (\mathbb{K} \setminus \{z\})$$
$$\Big((\mathrm{pr}_1(z) \neq \mathrm{pr}_1(z_0))\&(\mathrm{pr}_1(z) \neq \mathrm{pr}_2(z_0))\&(\mathrm{pr}_2(z) \neq \mathrm{pr}_1(z_0))\&(\mathrm{pr}_2(z) \neq \mathrm{pr}_1(z_0))\Big),$$

то число существенных списков будет $3^{\frac{k}{2}} \cdot 2^{N-k}$, где $k = 2m$ — число связанных пунктов (то есть остается решить $3^{\frac{k}{2}}/2^k$ от исходного числа подзадач).

Доказательство. Каждая адресная пара порождает 4 состояния: $\{00, 01, 10, 11\}$, где слева записан отправитель, а справа — получатель. Из них недопустимым является только состояние 10: отправитель есть, получателя нет. Следовательно, каждой адресной паре соответствует 3 допустимых состояния. Адресных пар, по условию, $\frac{k}{2}$, свободных пунктов $N - k$. Получаем, по теореме умножения, $3^{\frac{k}{2}} \cdot 2^{N-k}$ существенных списков. Следствие доказано. □

Рассмотрим теперь фундаментальный пример «зависимых» условий предшествования, который, за его упорядоченность, мы называем «цепью». Пусть k пунктов линейно упорядочены условиями предшествования $\mathbb{K}$. То есть пункты с первого по $k-1$-й являются отправителями и пункты со второго по k-й являются получателями, передают строго по цепочке: i-й передает $i+1$, где $i \in \overline{1,k-1}$. Свободных пунктов, соответственно, $N - k$ штук.

Предложение 2. *Цепь из k пунктов, описываемая условиями $\mathbb{K}$, дает $(k+1) \cdot 2^{N-k}$ существенных списков.*

Доказательство. Наши рассуждения соответствуют логике генератора существенных списков. Начнем с пустого состояния множества $\widehat{\mathbb{K}}$ связанных пунктов: $0 \ldots 0$ (k нулей). Какой пункт можно к нему добавить? Очевидно, что получить *существенный* список можно только добавив последний пункт цепи — получателя, не являющегося отправителем, — пункт k. Добавляем, получая состояние $0 \ldots 01$ ($k-1$ нулей). Следующим мы можем добавить только $(k-1)$-й пункт — отправителя для k-го, получим состояние $0 \ldots 011$ ($k-2$ нулей). Очевидно, что на всех промежуточных этапах мы можем добавить лишь предыдущий в линейном порядке пункт. Таким образом, в нашем случае мы k раз добавляем по одному пункту, то есть всего допустимых состояний получается $k+1$. С учетом состояний множества свободных пунктов $\widehat{\mathbb{K}}'$ общим числом существенных списков тогда будет $(k+1) \cdot 2^{N-k}$, что и требовалось доказать. □

Таким образом, цепь длиной k пунктов (еще раз напомним, что между ними могут в произвольном порядке вставать *свободные* пункты) дает сокращение числа списков заданий в $\frac{k+1}{2^k}$ раз. В предельном случае $k = N$ мы получаем тривиальную задачу, в которой порядок фиксирован изначально и выбора маршрутов как такового нет, существенных списков же получается $k+1 = N+1$ штук, считая тривиальный и полный. Подзадач каждой размерности ровно по одной.

В общем случае граф предшествования — ориентированный *лес* (отсутствуют ориентированные циклы, связность не гарантируется); далее мы рассматриваем исключительно связный

случай — «дерево предшествования»; если способ «обсчитывания» деревьев известен, результат для леса получается из предложения 1. Отметим также, что несвязный случай можно превратить в связный добавлением фиктивного пункта $N+1$, являющегося получателем всех пунктов, ничего не отправляющих «действительным» пунктам $1 \dots N$, включая *свободные* пункты.

§ 3. Маршрутизация перемещений инструмента в машинах листовой резки

Подсчет существенных списков в общем случае кажется нам нетривиальным, однако у нас есть решение для одного важного частного случая, связанного с маршрутизацией движения инструмента в машинах листовой резки; оптимизация движения инструмента актуальна в связи с тем, что между вырезаемыми контурами инструмент перемещается на холостом ходу — быстрее, чем на рабочем ходу, но не совершая полезной работы; ценность минимизации времени холостого хода очевидна. Неоптимальная же маршрутизация, скажем необходимость постоянно двигаться из конца в конец листа, может очень отрицательно сказаться на производительности. Точный (посредством МДП) и приближенный метод решения этой задачи рассматривались в [8].

Не все варианты условий предшествования могут встретиться в задаче о листовой резке. Содержательно ограничения на условия предшествования формулируются так: «внутренний контур всегда вырезается прежде внешнего»; очевидно, что каждый внутренний контур лежит внутри одного и только одного внешнего, соответственно каждому отправителю соответствует один и только один получатель (тогда как несколько отправителей вполне могут отправлять одному получателю — внутри одного контура может быть несколько других). В терминологии теории графов исходящая степень любой вершины графа предшествования не превосходит 1.

Граф предшествования в нашем случае — *дерево*. Корнем дерева является *получатель*, который *ничего не отправляет* — у него в дереве не будет *предков*; такая вершина выбрана *корнем*, потому что наш алгоритм начинает работу из нее. Для каждого пункта *потомками* являются его отправители, а *предками* — получатели. Под движением «вниз» по дереву мы будем понимать переход от пункта к его потомкам. Для подсчета числа существенных списков мы представляем дерево объединением цепей, что достигается переходом «вниз» ко все более простым поддеревьям, которые рано или поздно вырождаются в цепи. То есть, начав с корня дерева, мы движемся вниз, пока не встретим «развилку» — пункт, у которого более одного *потомка*, либо «тупик» — пункт, у которого *потомков нет* (отправитель, не являющийся получателем). Если мы дошли до «тупика», то путь окончен — возвращаем число существенных списков для пройденной цепи «корень»–«тупик», то есть длину цепи + 1 (поскольку пустая цепь тоже допустима). Если же мы дошли до «развилки», то мы возвращаем длину цепи «корень»–«развилка» плюс произведение числа существенных списков в поддеревьях, корнями которых являются потомки «развилки». Отметим, что в этом случае состояние, отвечающее наличию всех пунктов цепи «корень»–«развилка» и отсутствию пунктов из поддеревьев, корнями которых являются потомки «развилки», входит в вышеуказанное «произведении состояний поддеревьев-потомков», поэтому мы начинаем подсчет с «длины цепи», а не «длины цепи+1», как это происходит в случае тупика.

Алгоритм реализован на `C++`. Он состоит из определения *корня* дерева предшествования (напомним, что *корнем* мы считаем *получателя*, который не является *отправителем*) и вызова функции подсчета числа состояний с *корнем* всего дерева в качестве исходных данных. Корень можно определить «технически», найдя пункт, не являющийся *отправителем*, а можно — «содержательно», начав с произвольного пункта и все время двигаясь «вверх» по дереву — переходя к очередному *получателю*; «содержательный» способ лучше, поскольку в *худшем* случае требует число операций, пропорциональное наибольшей длине пути от вершины до *корня*, то есть наибольшему *уровню* вершины дерева, тогда как «технический» способ в худшем случае потребует проверить *каждую* вершину дерева.

Условия предшествования представлены *ассоциативным массивом*, сопоставляющим каждому *пункту* список его *отправителей* и *получателей*. Интерес представляет функция *рекурсивного* подсчета числа состояний *поддерева* предшествования с *корнем* в данной вершине

(термины *вершина дерева предшествования* и *пункт* мы здесь не различаем, то есть все упомянутые в описании алгоритма *пункты* являются *связанными*, если явно не указано обратное). Выше она описана содержательно, ниже представлен псевдокод (где символ ⟵ обозначает присваивание значения).

```
   Дано: условия предшествования.
   Результат: число допустимых состояний дерева предшествования.
   Входные данные: вершина КорПод — корень поддерева.
   Выходные данные: число допустимых состояний поддерева.
 1 Функция ЧислоСостояний(вершина КорПод)
   начало блока
 2 |  V ⟵ КорПод;
 3 |  n ⟵ 1;                          // счетчик числа состояний; пустое уже посчитано
 4 |  пока у вершины V один потомок;          // идем до «тупика» или «развилки»
 5 |  выполнять
 6 |  |  увеличить n на 1;
 7 |  |  V ⟵ потомок V;                              // спускаемся вниз на 1 уровень
 8 |  конец цикла
 9 |  если у вершины V нет потомков;    // проверяем, «тупик» ли V или «развилка»
10 |  то
11 |  |  увеличить  n на 1;
12 |  |  конец работы: n — число допустимых состояний поддерева;
13 |  иначе
14 |  |  b ⟵ 1;                        // счетчик числа состояний потомков «развилки» V
15 |  |  для каждого потомка V выполнять
16 |  |  |  b  умножить на ЧислоСостояний(потомок V)
17 |  |  конец цикла
18 |  |  к числу n прибавить b;
19 |  |  конец работы: n — число допустимых состояний поддерева;
20 |  конец условия
21 конец блока
```

Алгоритм 1: Функция рекурсивного подсчета числа допустимых состояний связанных вершин

На рисунке 1 приведен абстрактный пример раскроя листа с соответствующими ему условиями предшествования и граф предшествования. Пример строился с тем, чтобы показать достаточно сложное дерево предшествования, в реальных задачах они, как правило, проще. Всего получается 10 пунктов, все зависимые. Из $2^{10} = 1024$ списков заданий, согласно нашему алгоритму, существенными являются только 92. Снижение сложности вычислений налицо.

Из раскроя условия предшествования составляются следующим образом: каждому контуру, кроме самого внешнего (на рис. 1 имеет номер 1), сопоставляется условие предшествования. Отправителем назначается сам контур, получателем — тот контур, внутри которого он лежит. Поскольку отношение предшествования здесь, как и обычно, полагается *транзитивным*, дополнительные условия предшествования на случай, если внешний по отношению к данному контур вложен в другой контур (здесь, например, к (8,3) можно добавить (8,2) и (8,1)), не несут новой информации и добавлять их незачем. В задаче маршрутизации перемещений инструмента машины листовой резки транзитивность отношения *вложенности* контуров очевидна; в общем случае ориентированного графа Ахо, Гэри и Ульман доказали [9], что (транзитивно) *минимальное представление*[4] (минимальное отношение, транзитивное замыкание кото-

[4] Англ. transitive reduction; в русскоязычном сегменте Интернета встречается перевод «транзитивное сокращение».

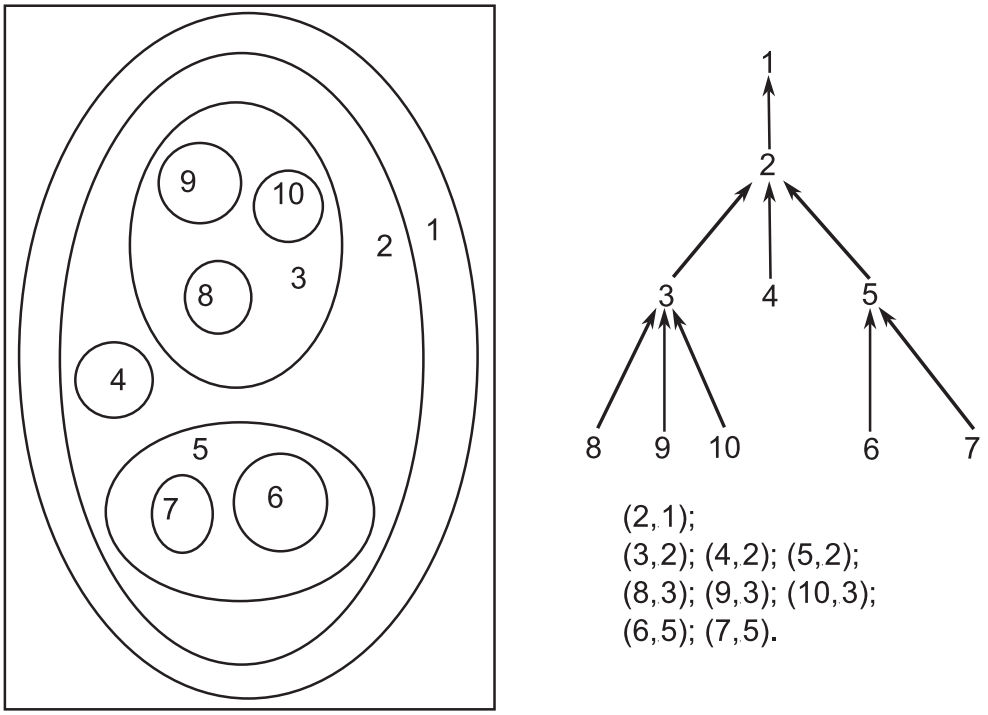

Рис. 1. Пример раскроя листа на несколько контуров и соответствующие ему граф предшествования и адресные пары

рого совпадает с данным отношением) сохраняет «информацию о путях» («path information»), что мы трактуем как тождественность множеств допустимых маршрутов, определяемых соответствующим графом предшествования, его *транзитивным замыканием* и *минимальным представлением*.

§ 4. Сложность алгоритма расчета числа допустимых состояний

Будем считать *существенными* в контексте *вычислительной сложности* операции *сравнение* (нужно для проверки числа потомков), *сложение* (включая унарное увеличение на 1), *умножение* и (рекурсивный) *вызов* функции подсчета числа допустимых состояний. Стоимость всех этих операций примем равной 1. Пусть n — число вершин в дереве предшествования, и пусть $\widehat{d^+}(i)$ — символ, равный числу $d^+(i)$ потомков вершины i, если у нее 2 и более потомков, и 0, если потомков 1 или менее. Подсчитаем число существенных операций.

1. Вызов функции подсчета происходит $\sum\limits_{i=1}^{n} \widehat{d^+}(i) + 1$ раз: исходный вызов плюс рекурсивный вызов для каждого потомка каждой «развилки».

2. Для каждого потомка «развилки» производится умножение, что дает $\sum\limits_{i=1}^{n} \widehat{d^+}(i)$ операций умножения.

3. Для каждой вершины счетчик числа состояний увеличивается на 1 и проверяется число ее потомков, что дает $2n$ операций. Эти действия совершаются единственный раз для каждой вершины.

4. Таким образом, суммарная сложность составляет порядка

$$2 \cdot \sum_{i=1}^{n} \widehat{d^+}(i) + 1 + 2n.$$

Приравняв число «развилок» и число потомков всех «развилок» к числу всех вершин, мы получим $2n^2 + 2n + 1$, то есть не более чем *квадратичный* порядок сложности.

§ 5. Заключение

В настоящее время труднорешаемые маршрутные задачи чаще всего решают приближенными, эвристическими методами. В отличие от точных методов они дают решение достаточно быстро; увы, для многих классов эвристических методов, включая популярные «интеллектуальные» методы, такие как *муравьиные* и *генетические* алгоритмы, точность приближения к оптимальному решению не гарантируется даже при решении задач, не обремененных условиями предшествования. Для эвристики условия предшествования — дополнительная трудность; между тем, как мы показали, они способны существенно снизить сложность точного метода динамического программирования. Соответственно, проанализировав задачу с ограничениями в виде условий предшествования, мы сможем сделать выбор и в пользу *точного* метода.

Очевидно, что при достаточно сильных условиях предшествования с помощью МДП можно за разумное время решать задачи большей размерности[5], чем обычно возможно. Таковы, например, задачи оптимальной маршрутизации перемещений инструмента в машинах листовой резки, для которой в нашей статье представлен алгоритм, позволяющий вычислить сложность. Примеры раскроя листового материала можно посмотреть у разработчиков пакетов автоматизации раскроя (например, САПР «СИРИУС», `http://www.haitek.ru/products/sirius.php`, «FieryCut» `http://exactcam.com/ru/` и др.); из них видно, что: а) в раскрое может быть несколько уровней вложенности контуров; б) размерность (число контуров) не слишком велика, соответственно, в некоторых случаях можно надеяться получить точное решение.

В дальнейшем мы надеемся обобщить наш подход до условий предшествования произвольного вида и планируем упростить анализ сложности посредством перехода к подсчету *существенных подзадач*. Текущий вариант подхода основан на подсчете *существенных списков заданий*; как мы уже говорили выше, *существенные списки заданий* имеют неодинаковую сложность в связи с тем, что им соответствует неодинаковое число *подзадач*. Поскольку наш подход никак не задействует критерий оптимальности, он может быть применен не только в разнообразных маршрутных задачах, но и в любых других, где вопрос оптимального порядка разрешается с использованием динамического программирования.

§ 6. Благодарности

Автор хотел бы поблагодарить своего научного руководителя А. Г. Ченцова за ценные советы и комментарии в процессе подготовки статьи; участников семинара Отдела управляемых систем ИММ УрО РАН за критику и комментарии к представленным в статье результатам; А. А. Петунина за консультацию по вопросам листовой резки.

СПИСОК ЛИТЕРАТУРЫ

1. Cormen T., Leiserson C., Rivest R., Stein C. Introduction to Algorithms, 3rd Ed. Cambridge. Massachussets: MIT Press, 2009. 1303 p.
2. Ченцов А.Г. Экстремальные задачи маршрутизации и распределения заданий: вопросы теории. М.–Ижевск: Институт компьютерных исследований, 2008. 240 с.
3. Bellman R. On the theory of dynamic programming // Proceedings of the National Academy of Sciences of the United States of America. 1952. Vol. 38. № 8. P. 716–719.
4. Bellman R. Dynamic programming treatment of the travelling salesman problem // Journal of the ACM. 1962. Vol. 9. № 1. P. 61–63.
5. Held M., Karp P. A dynamic programming approach to sequencing problems // Journal of the Society for Industrial & Applied Mathematics. 1962. Vol. 10. № 1. P. 196–210.

[5]Мы считаем, что условия предшествования следует учитывать в контексте снижения именно *размерности*, а не *сложности* задачи, ведь метод, пусть и на *меньшем* числе подзадач, работает точно так же.

6. Ченцов А.Г., Ченцов П.А. Маршрутизация с условиями предшествования (задача курьера): метод динамического программирования // Вестник УГТУ–УПИ. 2004. № 15. С. 148–151.
7. 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.
8. 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. Вып. 2 (169). С. 103–111.
9. Aho A.V., Garey M.R., Ullman J.D. The transitive reduction of a directed graph // SIAM Journal on Computing. 1972. Vol. 1. № 2. P. 131–137.

Поступила в редакцию 16.01.2014

Салий Ярослав Витальевич, ст. математик, отдел управляемых систем, Институт математики и механики имени Н. Н. Красовского УрО РАН, 620990, Россия, г. Екатеринбург, ул. С. Ковалевской, 16; ассистент, Институт математики и компьютерных наук, Уральский федеральный университет, 620000, Россия, г. Екатеринбург, пр. Ленина, 51.
E-mail: yvs314@gmail.com

Ya. V. Salii

On the effect of precedence constraints on computational complexity of dynamic programming method for routing problems

Keywords: precedence constraints, dynamic programming, computational complexity, sequential ordering problem, sheet cutting.

Mathematical Subject Classifications: 90C27, 90C39, 68Q25

We consider the general case of *Precedence Constrained TSP* (or a less general case of *Sequential Ordering Problem*) solved with a special kind of dynamic programming method that uses precedence constraints to significantly reduce the number of subproblems that must be solved to find the optimal solution of the original problem. Our aim is to quantify this reduction, which is necessary to clarify the influence of precedence constraints on computational complexity of dynamic programming solutions of such problems. This variety of the method of dynamic programming has been developed by A. G. Chentsov and his co-authors since 2004 but there was only one attempt at examining the influence of precedence constraints on complexity, which only described the influence of a single precedence constraint in the form of an "address pair" (sender, receiver).

Our approach to studying the complexity of this method is essentially the combinatorial analysis of the number of subproblems that are *feasible* in the sense of abiding by precedence constraints. Using the well-known combinatorial principles, *the rule of product* and *the rule of sum*, we established the estimates of complexity reduction for the following cases: a) "independent" sets of precedence constraints; b) "chains" of precedence constraints, when the precedence constraints define a linear ordering on the objects bound by them; c) precedence constraints expressed by an acyclic directed graph with outdegree (the number of receivers per sender) at most one. The latter case of precedence constraints is the one encountered in real-life problems of optimizing the route of the cutter in various machines used to cut sheet material. Since this is the most complex case of the three analyzed, instead of an analytic formula, we had to develop an algorithm (which we implemented in `C++`) to quantify the reduction; the computational complexity of the algorithm is less than quadratic with respect to the number of objects constrained by the precedence constraints. We intend to develop our approach to treat other cases of precedence constraints, eventually reaching the general case. We would also like to note that our method is optimization criterion-agnostic and thus applicable to many kinds of TSP, as long as they are precedence constrained and solvable by dynamic programming; in fact, our approach may be used to analyze the complexity of the dynamic programming method solution of any discrete optimization problem that deals with ordering subject to precedence constraints.

REFERENCES

1. Cormen T., Leiserson C., Rivest L., Stein C. *Introduction to Algorithms, 3rd Ed.*, Cambridge, Massachussets: MIT Press, 2009, 1303 p.

2. Chentsov A.G. *Ekstremal'nye zadachi marshrutizatsii i raspredeleniya zadanii: voprosy teorii* (Extremal problems of routing and scheduling: a theoretical approach) Moscow–Izhevsk: Institute of Computer Science, 2008, 240 p.

3. Bellman R. On the theory of dynamic programming, *Proc. Natl. Acad. Sci. USA*, 1952, vol. 38, no. 8, pp. 716–719.

4. Bellman R. Dynamic programming treatment of the travelling salesman problem, *Journal of the ACM*, 1962, vol. 9, no. 1, pp. 61–63.

5. Held M., Karp P. A dynamic programming approach to sequencing problems, *Journal of the Society for Industrial & Applied Mathematics*, 1962, vol. 10, no. 1, pp. 196–210.

6. Chentsov A.G., Chentsov P.A. Precedence-constrained routing (courier problem): a dynamic programming method, *Vestn. Ural. Gos. Tekh. Univ. – Ural. Politekh. Inst.*, 2004, no. 15, pp. 148–151 (in Russian).

7. Grigoriev A.M., Ivanko E.E., Chentsov A.G. Dynamic programming in a generalized courier problem with inner tasks: elements of a parallel structure, *Model. Anal. Inform. Sist.*, 2011, vol. 18, no. 3, pp. 101–124 (in Russian).

8. Petunin A.A., Chentsov A.G., Chentsov P.A. To the question about instrument routing in the automated machines of sheet cutting, *Nauch. Tekhn. Vedom. SPb Gos. Politekh. Univ. Inform. Telekom. Upr.*, St. Petersburg, 2013, issue 2 (169), pp. 103–111 (in Russian).

9. 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, pp. 131–137.

Received 16.01.2014

Salii Yaroslav Vital'evich, Senior Mathematician, Department of Control Systems, N. N. Krasovskii Institute of Mathematics and Mechanics, Ural Branch of the Russian Academy of Sciences, ul. S. Kovalevskoi, 16, Yekaterinburg, 620990, Russia;
Assistant Lecturer, Institute of Mathematics and Computer Science, Ural Federal University, pr. Lenina, 51, Yekaterinburg, 620000, Russia.
E-mail: yvs314@gmail.com