ПАРАЛЛЕЛЬНАЯ РЕАЛИЗАЦИЯ МЕТОДА ДИНАМИЧЕСКОГО ПРОГРАММИРОВАНИЯ В ОБОБЩЕННОЙ ЗАДАЧЕ КУРЬЕРА

А.М. Григорьев, Е.Е. Иванко, А.Г. Ченцов, П.А. Ченцов

В работе рассматривается обобщенная задача коммивояжера – посещение мегаполисов – с условиями предшествования. В англоязычной литературе такая задача известна как generalized courier problem (по-видимому, не рассматриваемая другими авторами, в совокупности всех приведенных особенностей). Далее для краткости будем называть эту задачу обобщенной задачей курьера (ОЗК), посещаемые множества будем называть мегаполисами, а их элементы – городами.

Подобная задача естественным образом возникает при оптимизации перемещений исполнителя в агрессивной среде. Исполнителю следует обойти все мегаполисы (конечные множества точек) и выполнить «задания» на каждом мегаполисе.

Другим актуальным приложением можно считать оптимизацию распараллеливания вычислений в современных вычислителях. Здесь исполнитель – это вычислитель, а посещаемые точки – это подаваемые на вход вычислителя задачи, некоторые из которых обладают кластерной структурой и не допускают расщепления. Наконец, относительно новым приложением класса задач коммивояжера является исследование изменчивости. В этой задаче строятся максимально правдоподобные «эволюционные русла», по которым происходило развитие множества заданных на входе объектов с известной матрицей «попарной схожести» и наличием нерасщепляемой кластерной структуры.

Рассматриваемая ОЗК, как и обычная задача коммивояжера, является NP-трудной. Существует ряд подходов, использующих различные эвристические и смешанные стратегии решения ОЗК. Однако, учитывая, что решение ОЗК может быть связано с минимизацией вредных последствий для здоровья исполнителя-человека, особый интерес представляют точные решения. В настоящей работе кратко описан метод динамического программирования для решения ОЗМ, пригодный (при условии использования супервычислителя) для решения задач, содержащих не более 30 множеств и порядка 30 элементов в каждом из множеств. В такие рамки, например, укладывается большая часть реальных прикладных задач, связанных с обслуживанием АЭС.

1. Содержательная постановка задачи. В рассматриваемой задаче требуется оптимально обойти мегаполисы, каждый из которых представлен конечным числом точек, выполнив на каждом из мегаполисов внутренние работы. Между любыми двумя точками любых двух мегаполисов (не обязательно различных) задана стоимость перехода. Допускается лишь однажды зайти и однажды выйти из каждого мегаполиса. Внутренние работы в настоящей постановке заключаются в оптимальном обходе всех городов мегаполиса, начиная из точки входа и заканчивая в точке выхода. Иными словами требуется оптимально обойти все города всех мегаполисов, при этом для каждого мегаполиса посещение его городов должно происходить без отвлечения на города других мегаполисов. Формально требуется найти такую последовательность обхода мегаполисов и такие точки входа и выхода в каждый из них, для которых суммарная стоимость переходов между городами будет минимальна.

2. Математическая постановка задачи и МДП. Пусть $X = \{x_0\} \cup \bigcup\limits_{i=1}^{N} M_i$, где x_0 – точка старта, а $\{M_1, \ldots, M_N\}$ – совокупность мегаполисов, каждый из которых содержит конечное число городов, $M_p \cap M_q = \varnothing$ при $p \neq q$. Считаем, что $x_0 \notin \bigcup\limits_{i=1}^{N} M_i$. Пусть, кроме того, задано множество $\mathbf{K}, \mathbf{K} \subset \overline{1, N} \times \overline{1, N}$ адресных пар «отправитель-получатель» на множестве индексов мегаполисов. Далее считаем $\forall z = (a, b) \in \mathbf{K}$ $\mathrm{pr}_1(z) = a, \mathrm{pr}_2(z) = b$. Через $\Re_+[S]$ обозначим множество всех неотрицательных вещественнозначных функций, определенных на множестве $S, S \neq \varnothing$. Также входными данными считаем следующие функции стоимости

$$\mathbf{c} \in \mathsf{P}_+[X \times X],$$

$$(c_i)_{i \in \overline{1,N}} : \overline{1, N} \to \mathsf{P}_+[X \times X],$$

$$\mathbf{f} \in \mathsf{P}_+[X].$$

Здесь функция $\mathbf{c}$ используется для оценки внешних перемещений. Функции $c_1,\ldots,c_N$ используются для оценки внутренних работ (выполняемых на множествах $M_1,\ldots,M_N$ соответственно). С помощью $\mathbf{f}$ мы оцениваем соответствующие конечные состояния.

В соответствии с работами [1-3] введем ряд дополнительных обозначений

где

$$\mathbf{K}_1 \cong \{pr_1(z) : z \in \mathbf{K}\}$$

Зададим «слои», на которых далее рекурсивно будет рассчитываться функция Беллмана.

$$D_0 \cong \mathbf{M} \times \{\varnothing\} = \{(x,\varnothing) : x \in \mathbf{M}\}\ ;\ D_N \cong \{(x^0, \overline{1,N})\}.$$

Если $s \in \overline{1,N-1}$ и $K \in \Gamma_s$, то

$$\Delta_s[K] \cong \{(x,K) : x \in \mathrm{M}_s[K]\}$$

Тогда, рассматривая последнее множество как клетку пространства позиций, определяем (в «клеточном» виде) промежуточные слои.

$$D_s = \bigcup_{K \in \Gamma_s} \Delta_s[K]\ \forall s \in \overline{1,N-1}$$

В приведенных выше обозначениях для решения поставленной задачи в соответствии с [1-4] рассмотрим следующую самодостаточную рекурсивную процедуру. Пусть

$$\varsigma_0(x,\varnothing) \cong \mathbf{f}(x)\ \forall x \in X$$

где функция $\mathbf{f} : X \to [0,\infty[$ служит для оценки соответствующих конечных состояний. Пусть $m \in \overline{0,N-1}$ и все функции

$$\varsigma_i, i \in \overline{0,m}$$

построены. В частности, имеем функцию

$$\varsigma_m : D_m \to [0,\infty[$$

Тогда для построения

$$\varsigma_{m+1}$$

используем соотношение:

$$\varsigma_{m+1}(x,K) = \min_{j \in \mathbf{I}(K)} \min_{z \in M_j \times M_j} [\mathbf{c}(x, pr_1(z)) + c_j(z) + \varsigma_m(pr_2(z), K \setminus \{j\})]\ \forall (x,K) \in D_{m+1}.$$

(*)

После конечного числа регулярных шагов типа (*), получим все функции

$$\varsigma_i, i \in \overline{0,N}$$

$$V = \varsigma_N(x^0, \overline{1,N}) = \min_{j \in \mathbf{I}(1,N)} \min_{z \in M_j \times M_j} [\mathbf{c}(x^0, pr_1(z)) + c_j(z) + \varsigma_{N-1}(pr_2(z), \overline{1,N} \setminus \{j\})].$$

3. Параллельная реализация. Предполагаем, что доступны $\mathbf{n}$ процессоров; здесь $\mathbf{n} \in \mathsf{N}$ и $2 \le \mathbf{n}$. Для всех $s \in \overline{1,N-1}$ выполним разбиение семейства Γ_s в объединение $\mathbf{n}$ попарно непересекающихся подсемейств $\Gamma_i^{(s)}, i \in \overline{1,\mathbf{n}}$; так что $(\Gamma_i^{(s)})_{i \in \overline{1,\mathbf{n}}} : \overline{1,\mathbf{n}} \to \mathsf{P}(\Gamma_s)$ есть конечная последовательность, удовлетворяющая условиям

Здесь $\Pi(\Gamma_s)$ – $(\Gamma_s = \bigcup_{i=1}^{\mathbf{n}} \Gamma_i^{(s)}) \,\&\, (\Gamma_k^{(s)} \cap \Gamma_l^{(s)} = \varnothing \; \forall k \in \overline{1,\mathbf{n}} \; \forall l \in \overline{1,\mathbf{n}} \setminus \{k\})$.

семейство всех подмножеств Γ_s. В результате получим следующие конечные последовательности: $(\Gamma_i^{(1)})_{i\in\overline{1,\mathbf{n}}},\ldots,(\Gamma_i^{(N-1)})_{i\in\overline{1,\mathbf{n}}}$

Для шага с номером $s \in \overline{1,N-1}$ все списки $\Gamma_1^{(s)}$ передаются процессору Π_1, все списки $\Gamma_2^{(s)}$ передаются процессору Π_2 и так далее. Это означает, что Π_1 работает с клетками

$$\Delta_s[K], K \in \Gamma_1^{(s)},$$

Π_2 работает с клетками

$$\Delta_s[K], K \in \Gamma_2^{(s)},$$

и так далее. Тогда Π_1 вычисляет функции

$$\overline{\varsigma}_s[K], K \in \Gamma_1^{(s)},$$

Π_2 вычисляет функции

$$\overline{\varsigma}_s[K], K \in \Gamma_2^{(s)},$$

и так далее. Склеивая все такие клетки функции Беллмана, можно построить слой ς_s. Здесь используется следующее правило:

$$\varsigma_s(x,K) = \overline{\varsigma}_s[K](x) \; \forall K \in \Gamma_s \; \forall x \in \mathrm{M}_s[K].$$

Клетки

$$\overline{\varsigma}_s[K], K \in \Gamma_s,$$

передаются в общую память компьютера. При $s < N-1$ для реализации шага с номером $s+1$ каждый процессор $\Pi_1,\ldots,\Pi_{\mathbf{n}}$ извлекает необходимые значения функций

$$\overline{\varsigma}_s[K], K \in \Gamma_s.$$

Для построения

$$\overline{\varsigma}_{s+1}[K], K \in \Gamma_{s+1},$$

процессор Π_1 обрабатывает

$$\overline{\varsigma}_{s+1}[K], K \in \Gamma_1^{(s+1)},$$

Π_2 обрабатывает клетки

$$\overline{\varsigma}_{s+1}[K], K \in \Gamma_2^{(s+1)},$$

и так далее. Склеивая все клетки

$$\overline{\varsigma}_{s+1}[K], K \in \Gamma_{s+1},$$

мы определяем функцию ς_{s+1}. Последующие конструкции аналогичны.

4. Вычислительный эксперимент. В данном разделе рассматривается два варианта реализации решения задачи маршрутизации на плоскости с помощью компьютерной программы, которая была реализована на супервычислителе МВС «Уран». Для проверки изложенного в предыдущих разделах способа распараллеливания МДП создана программа для МВС на языке программирования С++ с использованием двух

различных параллельных технологий (MPI и OpenMP), с целью выявления наиболее подходящего способа решения поставленной задачи. Для вычислительного эксперимента используется модуль МВС «Уран» со следующими характеристиками:

1. Процессор Intel Xeon X5675 6-ти ядерный 3,07 ГГц - 2 шт;
2. Оперативная память - 192 ГБ;
3. Жесткий диск 160 ГБ SATA;
4. Сетевой интерфейс 10G Ethernet - 1 шт.

Для сравнения двух способов реализации параллельного алгоритма решения рассматривается задача со следующими параметрами.

Функция стоимости внешних перемещений ***c*** определяется как евклидово расстояние на плоскости; функция ***f*** полагается тождественно равной нулю. Количество мегаполисов N=30 и число адресных пар

|**K**|=25. Будем полагать, что все множества M_j имеют одно и то же количество элементов: | M_j |=24 $\forall j \in \overline{1,30}$. Внутренние работы на каждом множестве M_j сводятся при выбранных пункте прибытия $x \in M_j$ и пункте отправления $y \in M_j$ к однократному посещению всех оставшихся точек этого множества. При этом функции $c_1,\ldots,c_{30}$ определяются следующим способом. Если $j \in \overline{1,30}, x \in M_j$ и $y \in M_j$, то $c_j(x,y)$ есть экстремум (значение) задачи коммивояжера в условиях, когда база (начальный пункт наблюдения) есть x, а конечный совпадает с y, при этом стоимость перемещений между узлами сетки j-го мегаполиса определяется евклидовым расстоянием. Иными словами, функции $c_j(x,y)$ выражают значения соответствующей множеству M_j внутренней метрической задачи коммивояжера с фиксацией двух пунктов наблюдения: начального пункта x и конечного y. Разумеется, с точки зрения решения совокупной задачи мы фактически ограничиваем исполнителя в части выполнения внутренних работ оптимальными вариантами поведения, что, однако, не приводит к потере качества в рассматриваемом случае аддитивного агрегирования затрат.

В качестве начального пункта отправления используется точка со следующими координатами $x^0 = (41.542, 62.362)$. Каждое множество M_j определяется узлами сетки, “равномерно заполняющими” круг, радиус которого принадлежит отрезку [4,5], а центр $a^{(i)}$ задается следующим образом:

$$\mathbf{K} = \{(1,3);(1,4);(1,5);(1,6);(1,7);(2,8);(2,9);(2,10);(2,11);(2,12);(1,13);(1,14);(2,15);$$
$$(3,16);(4,17);(6,18);(5,19);(9,20);(10,21);(11,22);(16,23);(19,24);(15,25);(10,26);(17,18)\}.$$

Оба варианта реализации справились с решением поставленной задачи, но показали разные результаты по скорости счета. Это обуславливается тем что, в случае использования библиотеки MPI, задаче необходимо на каждом слое МДП обмениваться информацией между процессорами. Таким образом, каждый процессор должен передать информацию, насчитанную на текущем слое, а так же получить от всех процессоров, участвующих в расчете задачи, вычисленные ими данные, для того, чтобы продолжить расчет следующего слоя функции Беллмана. В случае использования OpenMP работа процессоров, участвующих в вычислении, происходит с общей памятью, в связи с этим не происходит обмена насчитанной информацией между процессорами. Но при этом возникает одно существенное ограничение: нельзя использовать больше вычислительных ядер, чем установлено на одном модуле МВС. В случае использования MPI таких ограничений нет. Применительно к нашей задаче, где каждый процессор насчитывает большой объем информации и передает ее другим процессорам более целесообразно использовать алгоритм работы с общей памятью. Экспериментально установлено, что в случае использования MPI при увеличении числа процессоров более 8, время решения задачи начинает увеличиваться, то есть накладные расходы на обмен данными нивелируют выигрыш от увеличения скорости счета.

В результате решения такой задачи найден маршрут:

$(i_1 = 1, i_2 = 4, i_3 = 17, i_4 = 29, i_5 = 13, i_6 = 2, i_7 = 7, i_8 = 6, i_9 = 9, i_{10} = 10, i_{11} = 21, i_{12} = 3, i_{13} = 18,$
$i_{14} = 11, i_{15} = 22, i_{16} = 8, i_{17} = 5, i_{18} = 30, i_{19} = 20, i_{20} = 26, i_{21} = 15, i_{22} = 19, i_{23} = 28, i_{24} = 16, i_{25} = 14,$
$i_{26} = 27, i_{27} = 24, i_{28} = 23, i_{29} = 25, i_{30} = 12$ (соответствующая последовательность обхода узлов сетки для каждого множества не приводится) (см. Рисунок 1). Искомый экстремум V= 1674.8460111. Время счета с использованием метода программирования OpenMP 17348 секунд, с использованием библиотеки MPI 36275 секунд.

Приводимые ниже рисунки можно рассматривать как иллюстрацию возможных вариантов перемещения беспилотного летательного аппарата (или какого-то другого средства) с целью обзора лесных массивов на предмет выявления возможных аномалий (в частности, пожаров).

5. Некоторые добавления. Конструкция на основе МДП применялась для решения задач маршрутизации с особенностями (в сравнении с ОЗК). Так, в частности, в [5] рассматривалась постановка, в

которой функции $\mathbf{c}, c_1, \ldots, c_N$ включали зависимость от номера в очереди, что с формальной точки зрения отвечало введению в число аргументов упомянутых функций дискретного времени. Так, в частности, при $j \in \overline{1, N}$ значения функции c_j, оценивающей работы на мегаполисе M_j, имеют вид $c_j(x, y, t)$, где $x \in M_j, y \in M_j$, а $t \in \overline{1, N}$ играет роль момента времени, в который (с точностью до идеализации) осуществляется работа на M_j. Зависимость от t может быть обусловлена различными причинами (параметр t может играть роль «настоящего» времени, а может учитывать претензии некоторого клиента, заинтересованного в обслуживании M_j). Так или иначе, возникает вариант ОЗК с нестационарностью в определении функций стоимости $\mathbf{c}, c_1, \ldots, c_N$; в [5] построен вариант МДП для решения данной задачи. Важно, что в упомянутом варианте МДП удается, несмотря на возникающую нестационарность, «обойтись» использованием в качестве пространства позиций множеством $X \times \Pi(\overline{1, N})$. На основе МДП был построен и реализован в виде стандартной программы для ПЭВМ оптимальный алгоритм решения, включающий элементы параллельной структуры.

Для вычислений использовалась программа, написанная на Microsoft Visual C++ 2005. Вычисления производились на ПЭВМ с процессором Intel i7-2630QM с 8Гб оперативной памяти, работающей под управлением Windows 7. Указанный процессор содержит четыре ядра с технологией Hyper-threading, позволяющей на каждом ядре запускать два программных потока. Таким образом, в вычислениях при построении функции Беллмана использовались восемь программных потоков. При этом использовалась схема с общей памятью. Производился обход 27 мегаполисов, содержащих 50 городов. В задаче использовалось ограничение в виде условия предшествования, в которое входили 20 адресных пар. Одна из основных проблем при построении подобного алгоритма — распределение вычисляемых значений по процессорам. Дело в том, что некоторые значения вычисляются очень быстро, а некоторые требуют значительных вычислительных ресурсов. Может быть предложено множество подходов к решению данной задачи. Особенность указанной параллельной реализации состоит в том, что при расчете каждого слоя функции Беллмана распределение вычисляемых значений функции производилось так: на каждый из программных потоков отводился диапазон по 10000 значений, идущих с шагом в 80000 (восемь потоков) вплоть до завершения слоя. Опытным путем было установлено, что такое распределение позволяет максимально усреднить нагрузку на программные потоки. Простой потоков минимален. Таким образом, удалось избежать реализации сложного (и требовательного к ресурсам) алгоритма выравнивания нагрузки. Время счета - 58 минут 50 секунд.

Как видно из экспериментов, точное решение рассматриваемой задачи сложно в вычислительном плане. Задача обхода мегаполисов с предшествованиями (обобщенная задача курьера) по-видимому, не рассматривалась в публикациях помимо научного коллектива под руководством А.Г. Ченцова. Соответственно не рассматривались и эвристические алгоритмы для решения этой задачи. Несложно, однако, предложить жадный алгоритм решения данной задачи, адаптированный к условиям предшествования. На каждом шаге в таком алгоритме выбирается «ближайший» мегаполис и «ближайший» город мегаполиса из числа мегаполисов не являющихся «получателями» адресных пар, если только «отправитель» данной пары уже не посещен. Также к рассматриваемой обобщенной постановке можно адаптировать алгоритм [6], основанный на рекурсивном дроблении маршрута с учетом условий предшествования.

Работа поддержана грантами РФФИ 10-08-00484-а, 10-01-96020-р-урал-а, программой Президиума УрО РАН (проект 09-П-1-1014)

ЛИТЕРАТУРА:

1. Ченцов А.Г. Об оптимальной маршрутизации в условиях ограничений. Доклады Академии Наук, 2008, Т.423, №3, с. 303-307.
2. Ченцов А. А., Ченцов А. Г., Ченцов П. А. Экстремальная задача маршрутизации перемещений с ограничениями и внутренними потерями. Известия вузов. Математика, 2010, № 6, с. 64–81.
3. Ченцов, А. Г. Метод динамического программирования в экстремальных задачах маршрутизации с ограничениями. Известия РАН. Теория и системы управления, 2010, №3, с. 52-66.
4. 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. Ярославль, 2011, том 18, №3, с.101-124.
5. Ченцов А. Г., Ченцов П. А. Динамическое программирование в одной нестационарной задаче маршрутизации. Изв. Института математики и информатики УдмГУ. Выпуск 1(39), Ижевск, 2012, С. 151-154.