УДК 519.854.2; 004.272.2

ПАРАЛЛЕЛЬНАЯ РЕАЛИЗАЦИЯ СУБГРАДИЕНТНОГО АЛГОРИТМА ДЛЯ МАКСИМИЗАЦИИ ДВОЙСТВЕННОЙ ФУНКЦИИ ЛАГРАНЖА В ЗАДАЧЕ О P-МЕДИАНЕ

И. Л. Васильев[1], А. В. Ушаков[1]

Рассматривается алгоритм поиска нижних оценок для оптимального значения в задаче о p-медиане, основанный на построении релаксации Лагранжа, а также максимизации двойственной функции с помощью субградиентного метода. Предлагается эффективная схема распараллеливания такого алгоритма, включающая в себя процедуру каскадной сборки данных между процессами. Разработанный алгоритм тестируется на широком наборе модельных примеров большой размерности, в том числе на задачах, размерность которых превосходит известную до настоящего времени из литературы. Полученные результаты подтверждают эффективность предложенной модели распараллеливания. Работа выполнена при частичной финансовой поддержке РФФИ (проекты 12–07–33045-мол_а_вед и 12–01–31198-мол_а), а также СО РАН (интеграционный проект 21).

Ключевые слова: задача о p-медиане, параллельное программирование, релаксация Лагранжа, субградиентный метод, MPI.

1. Введение и постановка задачи. Пусть дано множество возможных пунктов размещения предприятий $I = \{1, \ldots, m\}$, множество клиентов $J = \{1, \ldots, n\}$, а также положительные величины d_{ij}, задающие затраты на обслуживание клиента $j \in J$ из предприятия, расположенного в пункте $i \in I$. Задача о p-медиане состоит в размещении p предприятий из множества возможных пунктов I таким образом, чтобы суммарные затраты на обслуживание всех клиентов были минимальны. Отметим, что на практике часто полагают, что $I = J$, в этом случае задача может быть сформулирована на полном простом взвешенном ориентированном графе $G(I, A)$ с множеством вершин I и множеством дуг $A = \{ij : i \in I, j \in J, i \neq j\}$, причем каждой дуге приписывается вес d_{ij}, задающий расстояние между вершинами $i \in I$ и $j \in I$. Задача состоит в поиске p вершин графа $G(I, A)$, называемых медианами, таких, что сумма всех весов входящих дуг к немедианным вершинам от ближайшей медианы была минимальна.

Задача о p-медиане может быть представлена в виде задачи целочисленного линейного программирования. С этой целью вводятся бинарные переменные y_i и x_{ij}, соответствующие вершинам и дугам графа. Переменная y_i принимает значение 1, если вершина i является медианой, и 0 в противном случае; переменная x_{ij} равна 1, если вершина i является ближайшей медианой к j, и 0 в противном случае. Дополнительно обозначим через $\delta^-(j) = \{i \in I |\ ij \in A\}$ множество вершин, соединенных с вершиной j выходящей из них дугой, а через $\delta^+(i) = \{j \in I |\ ij \in A\}$ множество вершин, соединенных с вершиной i выходящими из нее дугами. С использованием введенных переменных и обозначений задача о p-медиане может быть записана в следующей форме:

$$\min_{(x,y)} \sum_{(i,j)\in A} d_{ij} x_{ij}, \tag{1}$$

$$\sum_{i\in\delta^-(j)} x_{ij} + y_j = 1 \quad \forall\, j \in I, \tag{2}$$

$$x_{ij} \leqslant y_i \quad \forall\, (i,j) \in A, \tag{3}$$

$$\sum_{i\in I} y_i = p, \tag{4}$$

$$y_i \in \{0,\ 1\} \quad \forall\, i \in I, \tag{5}$$

$$x_{ij} \in \{0, 1\} \quad \forall\, (i,j) \in A. \tag{6}$$

[1] Институт динамики систем и теории управления СО РАН, ул. Лермонтова, 134, 664033, Иркутск; И. Л. Васильев, доцент, вед. науч. сотр., e-mail: vil@icc.ru; А. В. Ушаков, программист, e-mail: aushakov@icc.ru

Ограничения (2) гарантируют, что каждая вершина j является либо медианой, либо имеет одну входящую дугу из медианной вершины. Неравенства (3) исключают существование выходящих дуг из немедианных вершин. Количество медиан определяется уравнением (4). Условия на целочисленность переменных заданы ограничениями (5) и (6).

Задача о p-медиане является широко известной и одной из базовых в области дискретных задач размещения и впервые сформулирована в [6]. Одной из основных трудностей в разработке эффективных алгоритмов ее решения является тот факт, что она NP-трудна [9]. Вопреки этому к сегодняшнему моменту был предложен ряд методов и подходов к ее решению — как точных, так и эвристических, наиболее полный обзор которых представлен в работах [11, 13]. Несмотря на столь большой интерес к задаче и разнообразие предложенных методов, размерность примеров, для которых удается найти решение, остается сравнительно небольшой, что довольно существенно сужает область приложений задачи, зачастую содержащих десятки миллионов переменных и ограничений. Так, самые большие задачи, которыми способны оперировать современные алгоритмы, имеют размерность порядка 90 000 вершин. Результаты для таких примеров были получены, например, в работах [2, 5, 8]. Отметим, что основной трудностью в увеличении размерностей решаемых задач служит существенный рост количества данных и параметров, которые необходимо хранить в оперативной памяти на этапе выполнения алгоритмов.

Другим важным моментом является тот факт, что задача NP-трудна, что значительно сказывается на скорости выполнения алгоритмов с ростом размерности решаемых задач. Одним из главных способов преодоления такого рода ситуаций является применение средств и методов популярной и бурно развивающейся области параллельных вычислений. Для задачи о p-медиане уже был предложен ряд различных параллельных алгоритмов: параллельный поиск с чередующимися окрестностями [3, 12], распределенный поиск [4], алгоритм подстановки вершин, алгоритм Ллойда (discrete Lloyd algorithm) [10, 18] и др., однако размерность решаемых задач остается по-прежнему небольшой.

В настоящей статье рассматривается проблема поиска нижних оценок оптимального значения в задаче о p-медиане очень большой размерности, существенно превышающей известную до настоящего времени из литературы. Разработанный алгоритм основан на методе релаксаций Лагранжа, включающем в себя построение двойственной функции Лагранжа, а также субградиентный метод ее максимизации и поиска наилучшей нижней оценки. Предлагается схема распараллеливания такого подхода, эффективность которой иллюстрируется на серии тестовых задач с количеством вершин до одного миллиона.

Статья организована следующим образом: в разделе 2 приводится краткое описание метода релаксаций Лагранжа для задачи о p-медиане. В разделе 3 представлена параллельная схема субградиентного алгоритма для максимизации двойственной функции Лагранжа. Раздел 4 посвящен описанию одной процедуры ускорения работы метода за счет иерархической или каскадной сборки векторов оценок Лагранжа. Результаты вычислительных экспериментов приведены в разделе 5.

2. Релаксация Лагранжа и субградиентный метод. Метод поиска нижних оценок оптимального значения, основанный на релаксации Лагранжа, довольно часто применяется к задаче о p-медиане. Отметим, что в рамках настоящей статьи мы не будем подробно останавливаться на теоретических аспектах построения релаксации Лагранжа, более подробно эти вопросы освещены, например, в [1, 2]. Рассмотрим наиболее популярный тип релаксации исходной задачи, получающийся за счет ослабления ограничений (2). Эти ограничения добавляются в целевую функцию с некоторыми весами $\lambda \in \mathbb{R}^m$, называемыми множителями Лагранжа:

$$\mathcal{L}(\lambda) = \min_{(x,y)} \left\{ \sum_{(i,j)\in A} d_{ij}x_{ij} - \sum_{j\in I} \lambda_j \left(\sum_{i\in\delta^-(j)} x_{ij} + y_j - 1 \right) : \text{при (3)–(6)} \right\}.$$

Функцию $\mathcal{L}(\lambda)$ называют двойственной функцией Лагранжа. При любом фиксированном наборе множителей $\lambda = (\lambda_1, \ldots, \lambda_m)$ значение двойственной функции является нижней оценкой оптимального значения исходной задачи.

Для каждой вершины $i \in I$ вводятся величины $\rho_i(\lambda) = \sum\limits_{j\in\delta^+(i)} \min\{0, d_{ij} - \lambda_j\} - \lambda_i$, называемые оценками Лагранжа, и предполагается, что $\rho_{i_1}(\lambda) \leqslant \ldots \leqslant \rho_{i_n}(\lambda)$, т.е. они упорядочены по возрастанию. Тогда оптимальное решение $(x_{ij}(\lambda), y_i(\lambda))$ релаксированной задачи записывается в виде

$$y_i(\lambda) = \begin{cases} 1, & i \in T(\lambda), \\ 0, & \text{в противном случае;} \end{cases} \qquad x_{ij}(\lambda) = \begin{cases} 1, & y_i = 1 \wedge d_{ij} - \lambda_j < 0, \\ 0, & d_{ij} - \lambda_j \geqslant 0. \end{cases}$$

Значение двойственной функции Лагранжа может быть вычислено в явном виде по формуле

$$\mathcal{L}(\lambda) = \sum_{i \in T(\lambda)} \rho_i(\lambda) + \sum_{i \in I} \lambda_i,$$

где $T(\lambda)$ — множество, состоящее из p вершин с минимальной оценкой Лагранжа.

Для поиска наилучшей нижней оценки оптимального значения необходимо максимизировать двойственную функцию Лагранжа, т.е. решить задачу нахождения $\max\limits_{\lambda \in \mathbb{R}^m} \mathcal{L}(\lambda)$. Поскольку двойственная функция Лагранжа является негладкой, то для ее максимизации используется субградиентный алгоритм, находящий максимум посредством итерационной формулы $\lambda^{k+1} = \lambda^k + \alpha_k g\big(\lambda^k\big)$, где $g\big(\lambda^k\big)$ — вектор субградиента, найденный на k-й итерации по формуле $g_j\big(\lambda^k\big) = 1 - \sum\limits_{i \in \delta^-(j)} x_{ij}\big(\lambda^k\big) - y_j\big(\lambda^k\big)$.

Шаг метода α_k вычисляется по эвристическому правилу $\alpha_k = \dfrac{\phi\Big(1.05\,BUB - \mathcal{L}\big(\lambda^k\big)\Big)}{\Big\|g\big(\lambda^k\big)\Big\|_2^2}$, где BUB — верхняя оценка оптимального значения, $\|\cdot\|_2$ — евклидова норма, а ϕ — параметр.

Проанализируем особенности реализации субградиентного алгоритма для рассматриваемого случая. Для повышения эффективности работы алгоритма используется один из вариантов метода генерации столбцов [2], состоящий в следующем. Предположим, что матрица расстояний $D = \{d(i,j)\}$ вычислена и каждый ее столбец упорядочен по возрастанию. Таким образом, для каждого $j \in J$ имеется перестановка $u(j)$, такая, что $d\big(u_1(j), j\big) \leqslant d\big(u_2(j), j\big) \leqslant \ldots \leqslant d\big(u_m(j), j\big)$. В этом случае значение двойственной функции Лагранжа и вектор субградиента для заданного набора множителей λ могут быть эффективно вычислены по следующей схеме.

1. Инициализация: $\rho(\lambda) := -\lambda$, $\mathcal{L}(\lambda) := 0$ и $j := 1$.
2. Вычислить $\mathcal{L}(\lambda) := \mathcal{L}(\lambda) + \lambda_j$ и положить $h := 1$.
3. Если $d\big(u_h(j), j\big) \geqslant \lambda_j$, то перейти на шаг 6.
4. Вычислить $\rho_{u_h(j)}(\lambda) := \rho_{u_h(j)}(\lambda) + d\big(u_h(j), j\big) - \lambda_j$.
5. Если $h < m$, то положить $h := h + 1$ и перейти на шаг 3.
6. Если $j < m$, то положить $j := j + 1$ и перейти на шаг 2.
7. Найти множество $T(\lambda)$ и вычислить $\mathcal{L}(\lambda) := \mathcal{L}(\lambda) + \sum\limits_{i \in T(\lambda)} \rho_i(\lambda)$.

Отметим, что до начала основного цикла необходимо найти вектор переменных $y(\lambda)$, что отражено на шаге инициализации.

1. Инициализация: найти вектор $y(\lambda)$: $y_i := 1$ для всех $i \in T(\lambda)$, положить $j := 1$.
2. Вычислить $g_j(\lambda) := 1 - y_j(\lambda)$, положить $h := 1$.
3. Если $d\big(u_h(j), j\big) - \lambda_j \geqslant 0$, то перейти на шаг 6, в противном случае на шаг 4.
4. Если $y_{u_h(j)}(\lambda) = 1$, то положить $g_j(\lambda) := g_j(\lambda) - 1$.
5. Если $h < m$, то положить $h := h + 1$ и перейти на шаг 3, в противном случае на шаг 6.
6. Если $j < m$, то положить $j := j + 1$ и перейти на шаг 2, в противном случае stop.

Как было отмечено в работе [2], для подсчета значения двойственной функции Лагранжа и вектора субградиента при условии, что каждый столбец матрицы расстояний отсортирован по возрастанию, нет необходимости "пробегать" весь столбец j, а достаточно рассмотреть только первые элементы, значения которых не превосходят величины λ_j, что позволяет существенно сократить время работы алгоритма. Именно такой подход представлен в описанных выше схемах. Однако основным недостатком остается большая потребность в оперативной памяти, поскольку в ходе работы алгоритма приходится хранить всю матрицу расстояний. Для преодоления этой проблемы используется вариант подхода, основанного на частичном хранении матрицы расстояний и предложенного в [2], суть которого состоит в том, что для каждого отсортированного столбца $j \in I$ алгоритм хранит в оперативной памяти лишь некоторое число первых (наименьших) элементов $n_0^j < n$, называемых активными. Если на некоторой итерации k алгоритма множитель λ_j^k превысит $d\big(u_{n_0^j}(j), j\big)$, то столбец j дополняется на заданное фиксированное количество элементов n_1. После обновления числа активных элементов $n_0^j := n_0^j + n_1$, алгоритм продолжает свою работу.

Для контролирования роста значений множителей Лагранжа, а следовательно и числа активных элементов, применяется метод стабилизации, предложенный в [8]. Для каждого множителя устанавливается

верхняя граница, вначале полагаемая равной первому (наименьшему) элементу в соответствующем столбце, т.е. $\lambda_j^0 = d\big(u_1(j), j\big)$. Затем на каждой итерации верхняя граница устанавливается так, чтобы $\lambda_j^k \leqslant ub_j^k$, где $ub_j^k = \min\limits_{k\in I}\Big\{d\big(u_k(j), j\big) : d\big(u_k(j), j\big) > \lambda_j^{k-1}\Big\}$. Говоря другими словами, каждый множитель Лагранжа не может превзойти сразу два элемента в столбце матрицы расстояний на одной итерации.

Суммируя вышесказанное, представим общую схему субградиентного алгоритма максимизации двойственной функции Лагранжа $\mathcal{L}(\lambda)$ для задачи о p-медиане.

0. Инициализация: положить $BUB \in \mathbb{R}$ и $BLB := -10^{30}$, задать параметры $\phi_0 := 2$, $\gamma := 2$, $\beta := 0$, положить $\lambda_j^0 := d\big(u_1(j), j\big)$, $j = 1 \ldots m$, $k := 0$.

1. Подсчитать вектор оценок Лагранжа $\rho\big(\lambda^k\big)$, найти множество $T\big(\lambda^k\big)$ и подсчитать значение двойственной функции Лагранжа $\mathcal{L}\big(\lambda^k\big)$.

2. Если $\mathcal{L}\big(\lambda^k\big) > BLB$, то положить $BLB := \mathcal{L}\big(\lambda^k\big)$ и $\beta := 0$.

3. Проверить критерий остановки: если $BUB - BLB < 10^{-5}$, то stop, иначе перейти на шаг 4.

4. Вычислить вектор $y\big(\lambda^k\big)$ и вектор субградиента $g\big(\lambda^k\big)$.

5. Если $\beta \geqslant 30$, то $\phi_k := \dfrac{\phi_k}{\gamma}$, $\beta := \beta + 1$.

6. Проверить критерий остановки: если $\phi_k < 0.005$, то stop, иначе перейти на шаг 7.

7. Вычислить шаг $\alpha_k = \dfrac{\phi_k\Big(1.05\,BUB - \mathcal{L}\big(\lambda^k\big)\Big)}{\Big\|g\big(\lambda^k\big)\Big\|_2^2}$.

8. Положить $\lambda^{k+1} := \lambda^k + \alpha_k\, g\big(\lambda^k\big)$, $k := k + 1$ и перейти на шаг 1.

Отметим, что при реализации алгоритма мы использовали набор значений параметров, идентичный представленному в [2].

3. Распараллеливание алгоритма. Идея параллельного субградиентного алгоритма состоит в следующем. Фактически на каждой итерации алгоритма подсчет новых значений множителей Лагранжа, вектора оценок Лагранжа и вектора субградиента может производиться частями разными процессами независимо друг от друга, а следовательно, процессам необходимо оперировать лишь некоторыми непересекающимися подгруппами столбцов матрицы расстояний. Таким образом, итерационный процесс может быть успешно распараллелен.

При реализации параллельного алгоритма использовался интерфейс обмена сообщениями MPI. Приложение MPI состоит из нескольких ветвей или процессов, выполняющихся одновременно и обменивающихся данными в виде сообщений, общих данных у процессов нет. В качестве схемы взаимодействия между процессами использовалась модель Master–Slave [16, 17], в рамках которой выделяется один процесс — Master, управляющий работой всех остальных, называемых Slave (или вычислительные) процессы. Заметим, что в представленной реализации управляющий процесс имеет номер 0, а вычислительные $1, \ldots, s$.

Перед началом основного цикла управляющий процесс считывает входные данные, состоящие из набора координат вершин графа, и устанавливает значения параметров алгоритма, таких как: количество вершин m, медиан p, начальное количество активных элементов в каждом столбце n_0^j, а также диапазон индексов столбцов, т.е. размер блока, который каждый из Slave-процессов будет обрабатывать, в форме (j_q, n_q), где j_q — индекс первого столбца в блоке для данного процесса, n_q — размер блока, а $q = 1, \ldots, s$ — номер процесса. По завершении он формирует и отправляет всем вычислительным процессам сообщения, содержащие не только все считанные координаты вершин, но и индивидуальный набор параметров. На основании полученных данных каждый Slave-процесс вычисляет свою собственную часть матрицы расстояний $D = \big\{d(i,j)\big\}$, $i = u_1(j), \ldots, u_{n_0^j}(j)$, $j = j_q, \ldots, j_q + n_q$, а также устанавливает начальные значения множителей Лагранжа $\lambda_j^0 := d\big(u_1(j), j\big)$.

Основной цикл алгоритма начинается с подсчета Slave-процессами части вектора оценок Лагранжа $\rho_i\big(\lambda^k\big)$, $i \in I$, на основании столбцов из своего блока. Поскольку каждый столбец матрицы расстояний отсортирован по возрастанию и хранится в памяти не полностью, то затруднительно определить заранее, какие компоненты вектора оценок Лагранжа и на каком процессе будут отличны от нуля (т.е. заполнены), что вынуждает каждый Slave-процесс передавать управляющему два вектора, один из которых содержит полученные значения оценок Лагранжа, а другой — соответствующие им индексы. Дополнительной информацией, передаваемой на этом этапе, является сумма множителей $\sum\limits_{i=j_q}^{j_q+n_q} \lambda_i^k$, необходимая для подсчета

значения двойственной функции Лагранжа $\mathcal{L}(\lambda^k)$.

Получив эти данные, Master-процесс собирает полный вектор оценок, сортирует его, подсчитывает $\mathcal{L}(\lambda^k)$ и текущую нижнюю оценку BLB, а также находит множество $T(\lambda^k)$, проверяя затем критерий остановки (шаги 2 и 3). При выполнении критерия всем вычислительным процессам отправляется сообщение о завершении работы, в противном случае каждый Slave-процесс получает сообщение, содержащее множество индексов $T(\lambda^k)$.

На основе полученного множества $T(\lambda^k)$ вычислительные процессы находят вектор $y(\lambda^k)$. Подсчитав вектор $y(\lambda^k)$ и используя свой блок столбцов, каждый Slave-процесс вычисляет часть вектора субградиента $g_j(\lambda^k)$, $j = j_q, \ldots, j_q + n_q$, а также норму $\left\|g(\lambda^k)\right\|_2$, которую отсылает управляющему процессу.

Просуммировав части нормы вектора субградиента, Master-процесс вычисляет параметр ϕ_k, а также проверяет критерий остановки (шаг 6). Если критерий выполняется, то каждому Slave-процессу отправляется сообщение о завершении работы алгоритма. В противном случае управляющий процесс подсчитывает и отправляет всем Slave-процессам шаг α_k.

Наконец, каждый процесс, используя полученный шаг, может подсчитать новые значения своей части множителей Лагранжа $\lambda_j^{k+1} = \lambda_j^k + \alpha_k\, g(\lambda^k)$, $j = j_q, \ldots, j_q + n_q$, и перейти на следующую итерацию алгоритма.

4. Процедура иерархической сборки оценок Лагранжа. Численные результаты, полученные для представленной параллельной схемы, показали, что при увеличении размерности задачи эффективность алгоритма существенно снижается, приводя зачастую к увеличению времени вычисления даже по сравнению с последовательным алгоритмом [7]. Причиной этому служит тот факт, что на каждой итерации вычислительные процессы вынуждены отправлять управляющему на этапе подсчета оценок Лагранжа существенный объем данных в виде двух массивов, размерность которых для каждого Slave-процесса, вообще говоря, заранее не известна. С ростом размерностей задачи и количества вычислительных процессов происходит замедление работы алгоритма, поскольку Master-процессу необходимо обрабатывать и принимать все передаваемые ему сообщения, объем которых значителен. Поскольку без найденного множества $T(\lambda^k)$ Slave-процессы не могут продолжать вычисления, им приходится ожидать сначала своей очереди на передачу, а затем отправки сообщений всеми остальными процессами и их обработки управляющим процессом.

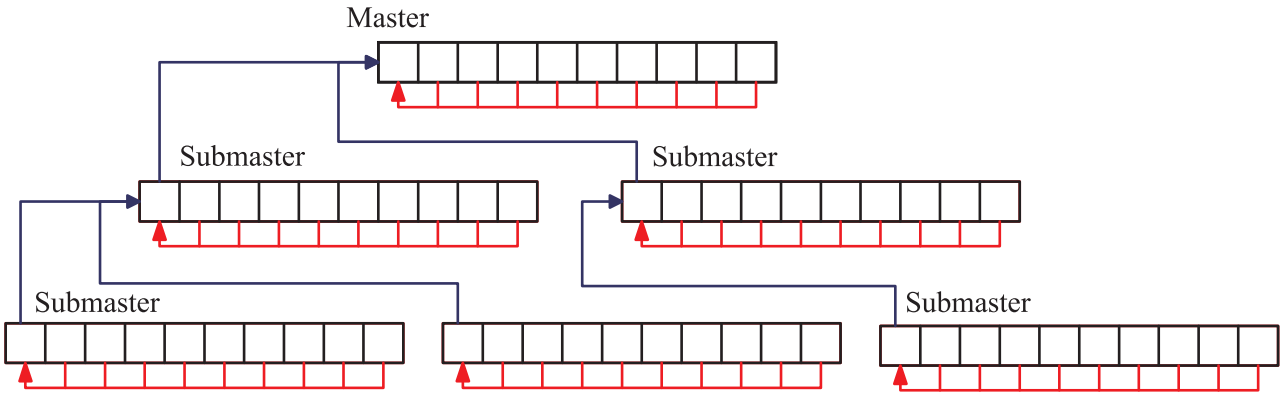

Рис. 1. Пример дерева иерархии

Для снижения нагрузки на Master-процесс и ускорения работы алгоритма была разработана схема иерархической, или каскадной сборки оценок Лагранжа и суммы множителей Лагранжа $\sum_{i=j_q}^{j_q+n_q} \lambda_i^k$, $q = 1, \ldots, s$, суть которой заключается в следующем. Все количество процессов последовательно разбивается на блоки фиксированной длины. Внутри каждого блока выделяется один главный процесс, называемый Submaster, отвечающий за сбор данных внутри своего блока. В реализованном варианте схемы таким процессом назначался имеющий наименьший номер в блоке. Далее все множество блоков процессов разбивается на уровни в виде корневого дерева, листья которого представляют собой блоки самого нижнего уровня. В качестве параметров такой схемы выступает желаемое количество уровней иерархии и размер создаваемых блоков. Вопреки классическому определению уровня вершин дерева мы будем полагать для удобства изложения, что корень находится на уровне 1. Сбор данных в схеме происходит последовательно: на первой стадии каждый Submaster производит прием и обработку сообщений от вычислительных

процессов из своего блока; после этого каждый Submaster отправляет собранную информацию Submaster-процессу, стоящему в дереве иерархии на один уровень выше, и так далее, пока данные не будут переданы в корень управляющему процессу (рис. 1).

Отметим, что формирование дерева в зависимости от выбранных параметров осуществляется по следующему принципу. На вершине иерархии (в корне дерева) всегда находится единственный блок, содержащий в качестве Submaster управляющий процесс, номер которого, очевидно, равен 0. Количество блоков на каждом уровне иерархии, или степень вершин $\deg(v)$ дерева, за исключением листьев и их родителей, определяется по формуле $\deg(v) = \min\left\{k \in \mathbb{N} : \sum\limits_{i=0}^{l-1} k^i \geqslant b\right\}$, где l — количество уровней иерархии, а b — общее количество блоков процессов.

Отметим, что степень вершин предпоследнего уровня иерархии, или родительских вершин листьев не задается по такому правилу из-за возможной неравномерной загрузки Submaster-процессов. Действительно, может возникнуть ситуация, представленная на рис. 2, в которой, несмотря на то что построенная иерархия блоков имеет три уровня, часть вершин дерева второго уровня не имеет потомков. Для избежания таких ситуаций блоки последнего уровня распределяются между Submaster-процессами предпоследнего уровня по возможности равномерно таким образом, чтобы $\left|\deg(u) - \deg(v)\right| \leqslant 1$, где u, v — любые вершины дерева иерархии, имеющие один уровень. Процедура построения иерархии блоков выполняется на управляющем процессе до начала основного цикла субградиентного алгоритма. Для формирования иерархии каждый вычислительный процесс получает информацию касательно его места в ней, состоящую только из двух параметров: количество получаемых сообщений с оценками Лагранжа, а также номер процесса, с которым происходит обмен данными.

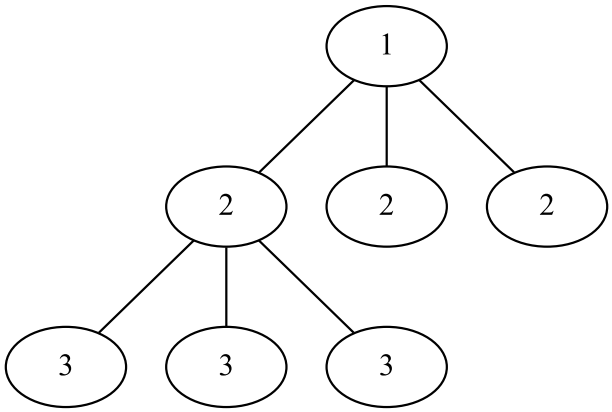

Рис. 2. Пример неравномерной загрузки Submaster-процессов

5. Результаты вычислительных экспериментов. Представленная выше схема параллельного субградиентного алгоритма была реализована на языке C++ с использованием библиотеки MPICH 1.2.7 и протестирована на вычислительном кластере Blackford, имеющем следующие характеристики: 20 вычислительных узлов, 40 процессоров Intel Xeon Quad-Core EM64T, 160 ядер; пиковая производительность 1.493 TFlops; суммарный объем оперативной памяти на узлах — 160 GB [15]. В качестве тестовых примеров использовались две задачи большой размерности, взятые из библиотеки TSP, размерностью 36 000 и 89 600 вершин, аналогичные тем, что использовались в [2, 8]. Задачи бóльших размерностей от 100 000 до 1 000 000 вершин были сгенерированы в соответствии с методом, предложенным в работе [14].

Верхняя оценка BUB, необходимая для подсчета шага алгоритма α_k, бралась из статьи [2]. Количество активных элементов n_0^j в столбце отсортированной матрицы расстояний выбиралось таким образом, чтобы их суммарное количество в оперативной памяти для одного процессорного ядра не превосходило 800 Mb. При необходимости число активных элементов в одном столбце увеличивалось на $n_1 = 100$.

Результаты вычислительных экспериментов, иллюстрирующие параллельное ускорение и эффективность параллельного алгоритма при различном количестве процессорных ядер для задач из библиотеки TSP, представлены в табл. 1 и 2. Напомним, что параллельное ускорение вычисляется по формуле t_s/t_i, где t_s — время работы последовательного алгоритма, а t_i — время, затраченное на решение задачи параллельным алгоритмом с использованием i процессоров. Эффективностью параллельного алгоритма называют величину, вычисляемую по формуле t_s/it_i. Отметим, что в столбце “Время” представлено полное время работы алгоритма, включая процедуру считывания входных данных, а также подсчет матрицы расстояний.

Анализируя полученные результаты, можно сделать вывод, что параллельное ускорение, достаточно близкое к линейному, было получено при увеличении числа процессорных ядер вплоть до 16 как для задачи с 36 000, так и с 89 600 вершинами. При дальнейшем увеличении числа процессорных ядер ускорение постепенно замедляется, практически останавливаясь при достижении 56 для примера с 36 000 вершин и при достижении 96 для задачи с 89 600 вершинами.

Столь высокая эффективность разработанного параллельного алгоритма для задач размерности в

Таблица 1

Параллельное ускорение и эффективность алгоритма для задачи с 36 000 вершинами

Количество ядер	Время, с	Ускорение	Эффективность
1	658.06	1	1
8	150.36	4.377	0.547
16	84.82	7.758	0.485
24	66.41	9.909	0.413
32	53.40	12.323	0.385
40	47.03	13.992	0.350
48	43.33	15.187	0.316
56	40.43	16.277	0.291
64	38.48	17.101	0.267
72	37.17	17.704	0.246
80	35.72	18.423	0.230
88	34.51	19.069	0.217

Таблица 2

Параллельное ускорение и эффективность алгоритма для задачи с 89 600 вершинами

Количество ядер	Время, с	Ускорение	Эффективность
1	3915.67	1	1
8	754.83	5.187	0.648
16	405.02	9.668	0.604
24	293.43	13.344	0.556
32	235.88	16.600	0.519
40	206.32	18.979	0.474
48	186.52	20.993	0.437
56	179.34	21.834	0.390
64	164.90	23.746	0.371
72	159.49	24.551	0.341
80	150.08	26.091	0.326
88	144.41	27.115	0.308
96	141.18	27.735	0.289
104	137.46	28.486	0.274
112	134.67	29.076	0.260
120	131.55	29.766	0.248
128	129.39	30.263	0.236
136	127.96	30.601	0.225
144	126.29	31.005	0.215
152	125.41	31.223	0.205

несколько десятков тысяч вершин дала основание для проведения тестирования на примере значительно большей размерности, содержащем один миллион вершин (10^{12} переменных) и сгенерированном по правилу, предложенному в статье [14]. Вычислительный эксперимент проводился на 80 процессорных ядрах вычислительного кластера. Относительная погрешность известной верхней оценки относительно нижней составила порядка 0.346%. Общее время вычисления, включая подсчет матрицы расстояний, составило 22 101 секунд (чуть более шести часов).

На основании проведенных вычислительных экспериментов можно сделать вывод, что предлагаемая параллельная схема субградиентного алгоритма для максимизации двойственной функции Лагранжа в задаче о p-медиане является достаточно эффективной и масштабируемой при увеличении размерности, позволяя успешно оперировать задачами, содержащими триллионы переменных. Более того, процедура иерархической, или каскадной сборки оценок Лагранжа обеспечивает дополнительную производитель-

ность алгоритма для больших тестовых задач, существенно ускоряя время работы по сравнению с разработанной ранее версией алгоритма [7]. Следующим логическим этапом исследований предполагается разработка подходов к поиску верхней оценки оптимального значения, что позволит успешно находить субоптимальные решения в задачах большой размерности.

СПИСОК ЛИТЕРАТУРЫ

1. *Васильев И.Л., Ушаков А.В.* Релаксации Лагранжа для нелинейной задачи о p-медиане // Известия ИГУ. Серия "Математика". 2011. **4**, № 2. 45–60.
2. *Avella P., Boccia M., Salerno S., Vasilyev I.* An aggregation heuristic for large scale p-median problem // Computers & Operations Research. 2012. **39**, N 7. 1625–1632.
3. *Crainic T.G., Gendreau M., Hansen P., Mladenović N.* Cooperative parallel variable neighborhood search for the p-median // Journal of Heuristics. 2004. **10**, N 3. 293–314.
4. *Garciá-López F.* Parallelization of the scatter search for the p-median problem // Parallel Computing. 2003. **29**, N 3. 575–589.
5. *García S., Labbé M., Marín A.* Solving large p-median problems with a radius formulation // INFORMS Journal on Computing. 2011. **23**, N 4. 546–556.
6. *Hakimi S.L.* Optimum distribution of switching centers in a communication network and some related graph theoretic problems // Operations Research. 1964. **13**, N 3. 462–475.
7. *Hanafi S., Salerno S., Ushakov A., Vasilyev I.* A parallel subgradient algorithm for Lagrangean dual function of the p-median problem // Studia Informatica Universalis. 2011. **9**, N 3. 105–124.
8. *Hansen P., Brimberg J., Urosević D., Mladenović N.* Solving large p-median clustering problems by primal-dual variable neighborhood search // Data Mining and Knowledge Discovery. 2009. **19**, N 3. 351–375.
9. *Kariv O., Hakimi S.L.* An algorithmic approach to network location problems; part 2. The p-medians // SIAM Journal on Applied Mathematics. 1979. **37**, N 3. 539–560.
10. *Ma L., Lim G.* GPU-based parallel computational algorithms for solving p-median problem // Proc. of the IIE Annual Conference. Reno (Nevada, USA), 2011. ID: 1110.
11. *Mladenović N., Brimberg J., Hansen P., Moreno-Pérez J.A.* The p-median problem: A survey of metaheuristic approaches // EJOR. 2007. **179**, N 3. 927–939.
12. *Moreno-Vega J.M.* The parallel variable neighborhood search for the p-median problem // Journal of Heuristics. 2002. **8**, N 3. 375–388.
13. *Reese J.* Solution methods for the p-median problem: An annotated bibliography // Networks. 2006. **28**, N 3. 125–142.
14. *Zhang T., Ramakrishnan R., Livny M.* BIRCH: an efficient data clustering method for very large databases // Journal of the American Statistical Association. 1996. **98**, N 463. 103–114.
15. Вычислительный кластер Blackford [Электронный ресурс] // Иркутский суперкомпьютерный центр СО РАН (http://hpc.icc.ru/hardware/blackford.php).
16. *Gropp W., Thakur R., Lusk E.* Using MPI-2: advanced features of the message passing interface. Cambridge: MIT Press, 1999.
17. *Васильев И.Л., Климентова К.Б., Орлов А.В.* Параллельный глобальный поиск равновесных ситуаций в биматричных играх // Вычислительные методы и программирование. 2007. **8**, № 1. 233–243.
18. *Lim G.J., Ma L.* GPU-based parallel vertex substitution algorithm for the p-median problem // Computers & Industrial Engineering. 2013. **64**, № 1. 381–388.

Поступила в редакцию
19.11.2012