УДК 519.853.6

© 2014 г. **В.В. Пересветов,** канд. физ.-мат. наук
(ВЦ ДВО РАН, Хабаровск)

ЭВОЛЮЦИОННЫЕ АЛГОРИТМЫ И ЛОКАЛЬНЫЙ ПОИСК В РЕШЕНИИ НЕЛИНЕЙНЫХ ТРАНСПОРТНЫХ ЗАДАЧ

Решаются двухиндексные транспортные задачи с нелинейными функциями стоимости. Разработан гибридный эволюционный алгоритм с локальным поиском. Реализованы параллельные алгоритмы на двумерной сетке MPI процессов.
Ключевые слова: двухиндексная транспортная задача, нелинейная функция стоимости, эволюционный алгоритм, локальный поиск, параллельный алгоритм.

Введение

В работе рассматриваются нелинейные транспортные задачи (НТЗ) [1], в которых стоимости доставки грузов нелинейно зависят от их объемов. Необходимость решения НТЗ возникает, например, когда производители продукции и перевозчики предоставляют различные схемы скидок при увеличении объема поставок. Среди приближенных методов решения НТЗ в настоящее время применяются метаэвристические методы: эволюционные и генетические алгоритмы [2 – 4], метод роя частиц [5] и др. В [3] предложены параллельные гибридные метаэвристические алгоритмы решения многоэкстремальных двухиндексных целочисленных НТЗ, в которых эволюционный алгоритм (ЭА) дополнен локальным поиском (ЛП). В [4] такая схема алгоритмов используется для нахождения вещественных решений НТЗ.

В настоящей работе развиваются гибридные алгоритмы [4] нахождения вещественных решений НТЗ. Допускаются случаи, когда функции стоимости не являются непрерывно дифференцируемыми: они могут быть кусочно-линейными или иметь разрывы первого рода. Параллельная версия алгоритмов построена на двумерной сетке MPI процессов, что позволяет проводить распараллеливание по популяциям и внутри каждой популяции. Проведены вычислительные эксперименты по исследованию параллельных алгоритмов в зависимости от размерности задач и схем распараллеливания. Эффективность разработанных алгоритмов проверяется при решении НТЗ из [6].

Формулировка задачи

Пусть количество продукта в пункте отправления – a_i, $i=1,2,...,N^i$, а потребность продукта в пункте назначения – b_j, $j=1,2,...,N^j$. Объемы перемещае-

мых грузов x_{ij} удовлетворяют ограничениям:

$$\sum_{j=1}^{N^j} x_{ij} = a_i,\ i = 1,2,...,N^i; \quad (1)$$

$$\sum_{i=1}^{N^i} x_{ij} = b_j,\ j = 1,2,...,N^j; \quad (2)$$

$$x_{ij} \in R,\ x_{ij} \geq 0,\ i = 1,2,...,N^i,\ j = 1,2,...,N^j. \quad (3)$$

Для замкнутой транспортной задачи суммарный объем продукта в пунктах отправления равен общей потребности в пунктах назначения:

$$\sum_{i=1}^{N^i} a_i = \sum_{j=1}^{N^j} b_j. \quad (4)$$

Матрица X с элементами x_{ij}, $i = 1,2,...,N^i$, $j = 1,2,...,N^j$ представляет собой совокупность объемов грузов – план перевозок. Множество планов перевозок, удовлетворяющих (1) – (4), обозначим *W*.

Стоимость перевозки по маршруту *ij* зависит нелинейным образом от его объема $F_{ij}(x_{ij})$ и не является в общем случае непрерывно дифференцируемой функцией. Общая стоимость всех перевозок является целевой функцией (ЦФ) НТЗ и имеет вид:

$$Z(X) = \sum_{i=1}^{N^i} \sum_{j=1}^{N^j} F_{ij}(x_{ij}). \quad (5)$$

Требуется найти оптимальный план перевозок НТЗ:

$X^{opt} = \text{argmin}_{X \in W} Z(X)$.

В дальнейшем будет использоваться представление нелинейной функции стоимости перевозок по маршруту *ij* в виде [2,6]:

$$F_{ij}(x_{ij}) = c_{ij} f(x_{ij}),\ i = 1,2,...,N^i,\ j = 1,2,...,N^j, \quad (6)$$

где $f(x)$ – нелинейная типовая функция стоимости (функция формы); c_{ij} – коэффициент стоимости перемещения груза по маршруту *ij*.

Алгоритмы решения

Для решения многоэкстремальных НТЗ с недифференцируемой непрерывно ЦФ разработан гибридный алгоритм, в котором последовательно работают ЭА поиска глобального минимума и ЛП, уточняющий найденное ЭА приближенное решение на каждой итерации формирования нового поколения. Основная цель ЭА состоит в поиске окрестности решения НТЗ, в которой ЦФ достигает своего минимального значения. В случае многоэкстремальных задач таких окрестностей может быть несколько, а если ставится задача поиска приближенного решения, то и субоптимальные решения также должны быть целью поиска. ЛП проводит уточнение найденного приближенного решения. При решении рассматриваемых НТЗ ЛП позволяет значительно повысить итоговую скорость расчетов, так как содержит значительно меньше вычислительных операций, чем ЭА.

Общая схема алгоритмов построена на «островной с миграциями» модели. В ней создается S популяций, каждая из которых самостоятельно выполняет поиск решения. Через заданное число итераций в каждой популяции приближенное решение, имеющее наименьшее значение ЦФ, передается управляющей популяции, которая находит из всех лучшее и рассылает для включения в остальные популяции.

Оператор инициализации $I_q(X_0)$, где X_0 – заполненная нулевыми значениями матрица, генерирует планы $X_n \in W$, $n = 1,...,N^{pop}$; N^{pop} – размер популяции. В ЭА последовательно работают оператор рекомбинации – оператор скрещивания (кроссинговер) $C_q(X_n, X_m)$, $n \neq m$, $n,m = 1,...,N_{pop}$ и мутации $M_q(X_n)$. Оператор ЛП $L_q(X_n)$ и другие операторы имеют по три варианта построения алгоритмов $q = 1,2,3$. В процессе решения НТЗ выбор значения q осуществляется по «правилу рулетки». Вероятности выбора варианта p_q^{init}, p_q^{cros}, p_q^{mut}, p_q^{ls}, $q = 1,2,3$ являются настраиваемыми параметрами.

Популяции каждого поколения удовлетворяют требованию $X_n \in W$, $n = 1,...,N^{pop}$ (индекс номера поколения не используется, чтобы не перегружать запись), что существенно усложняет построение алгоритмов начальной инициализации, операторов скрещивания, мутации и ЛП по сравнению с их аналогами для обычных задач многомерной оптимизации с независимыми переменными. Требование $X_n \in W$ означает, что если $x^n_{i_1,j_1}$ изменяется на величину δ, то, как минимум, еще в трех элементах $x^n_{i_1,j_2}$, $x^n_{i_2,j_1}$, $x^n_{i_2,j_2}$, $i_1 \neq i_2$, $j_1 \neq j_2$ должны быть проведены согласованные изменения, чтобы сохранить целостность плана:

$$x^n_{i_1,j_1} = x^n_{i_1,j_1} + \delta, \quad x^n_{i_1,j_2} = x^n_{i_1,j_2} - \delta; \tag{7}$$

$$x^n_{i_2,j_1} = x^n_{i_2,j_1} - \delta, \quad x^n_{i_2,j_2} = x^n_{i_2,j_2} + \delta; \tag{8}$$

$$\delta^{\min} \le \delta \le \delta^{\max}; \tag{9}$$

$$\delta^{\min} = -\min(x^n_{i_1,j_1}, x^n_{i_2,j_2}, \min(a_{i_1}, b_{j_2}) - x^n_{i_1,j_2}, \min(a_{i_2}, b_{j_1}) - x^n_{i_2,j_1}),$$

$$\delta^{\max} = \min(x^n_{i_1,j_2}, x^n_{i_2,j_1}, \min(a_{i_1}, b_{j_1}) - x^n_{i_1,j_1}, \min(a_{i_2}, b_{j_2}) - x^n_{i_2,j_2}).$$

Справедливость (7), (8) можно проверить, при выполнении (9), непосредственно подставляя в (1) – (4).

Оператор селекции (репродукции) используют «правило рулетки», при котором чаще выбираются решения, имеющие наименьшие значения ЦФ. Из них формируются пары для проведения операторов скрещивания. Только два элитных решения из предыдущего поколения сразу передаются на этап ЛП. Первое элитное решение – лучшее в данной популяции (имеющее наименьшее значение ЦФ), второе – зависит от шага итерационного процесса. Если на данной итерации происходит обмен между популяциями, то лучшее решение всех популяций заменяет худшее в каждой популяции и становится вторым элитным решением. Если обмена между популяциями нет, то находящееся на втором месте в популяции решение становится вторым элитным решением.

В большинстве практически интересных случаев функция стоимости $f(x)$ является неубывающей. Преимущественное заполнение элементов плана большими значениями x_{ij}, которым соответствуют небольшие значения c_{ij}, позволяет, исходя из (5)-(6), чаще получать минимальные значения ЦФ. Для использования этого коэффициенты c_{ij} предварительно разбиваются не менее чем на 20 интервалов по их значениям – проводится упрощенная сортировка. Затем создаются векторы I_s, J_s, которые заполняются индексами элементов матрицы c_{ij}, при этом в начале расположены индексы, указывающие на минимальные значения c_{ij}, а далее – по их возрастанию. Векторы I_s, J_s многократно используются при инициализации и ЛП.

В первом варианте алгоритмов инициализации $I_1(X_0)$ предварительно формируются $a_i' = a_i$, $b_j' = b_j$, $i = 1,2,...,N^i$, $j = 1,2,...,N^j$. Затем случайным образом выбранный элемент матрицы X_0 заполняется значением $x_{ij}^n = \min(a_i', b_j')$ и вносятся поправки: $a_i' = a_i' - x_{ij}^n$, $b_j' = b_j' - x_{ij}^n$. Так повторяется, пока все a_i', b_j', $i = 1,2,...,N^i$, $j = 1,2,...,N^j$ не будут равны нулю. $I_1(X_0)$ выполняется для всей популяции $n = 1,...,N^{pop}$. Вариант $I_2(X_0)$ отличается от предыдущего тем, что в нем задается более высокая вероятность выбора элемента x_{ij}^n, если ему соответствует небольшое значение c_{ij}, для этого используются векторы I_s, J_s. В операторе $I_2(X_0)$ с более высокой вероятностью заполняются элементы матрицы плана, которым соответствуют малые значения c_{ij}. В результате работы $I_1(X_0)$ или $I_2(X_0)$ план $X_n \in W$ заполняется большим числом нулевых элементов. В $I_3(X_0)$ план формируется почти всегда ненулевыми вещественными значениями. С этой целью элементы заполняются значениями $x_{ij}^n = r\min(a_i', b_j')$, где r – случайное число в интервале [0,1].

Алгоритмы оператора скрещивания $C_1(X_{n_1}, X_{n_2})$, $n_1, n_2 = 1,...,N^{pop}$, $n_1 \neq n_2$ основаны на формуле расчета элементов матрицы:

$$x_{ij}^{cros_1(n_1,n_2)} = r x_{ij}^{n_1} + (1-r) x_{ij}^{n_2}, \qquad (10)$$

где r – случайное число в интервале [0,1]. Оператор $C_2(X_{n_1}, X_{n_2})$ учитывает значения ЦФ первого и второго решений Z^{n_1}, Z^{n_2}:

$$x_{ij}^{cros_2(n_1,n_2)} = R x_{ij}^{n_1} + (1-R) x_{ij}^{n_2}, \qquad (11)$$

$$R = \begin{cases} 0,5 + 0,5k_c, & Z^{n_1} < Z^{n_2}, \\ 0,5 - 0,5k_c, & Z^{n_1} \geq Z^{n_2}, \end{cases}$$

где $k_c = rk_0$; r – случайное число; $0 \leq k_0 \leq 1$ – заданный параметр. Если $k_0 = 0$, то $R = 0,5$. В $C_3(X_{n_1}, X_{n_2})$ $k_c = k_0$ элементы $x_{ij}^{cros_3(n_1,n_2)}$ также зависят от значений

Z^{n_1}, Z^{n_2}, но случайные числа не используются.

Непосредственным суммированием левой и правой части (10) или (11) можно показать, что если планы X_{n_1}, X_{n_2} удовлетворяют условиям целостности (1) – (4), то и планы, порождаемые операторами $C_q(X_{n_1}, X_{n_2})$, $q = 1,2,3$, также будут им удовлетворять.

Алгоритм оператора мутации $M_1(X_n)$, $n = 3,..., N^{pop}$ (два первых решения – элитные) основан на том, что условия целостности $X_n \in W$ не будут нарушены, если в элементах матрицы плана провести изменения (7)-(8). В $M_1(X_n)$ случайно выбираются два элемента с индексами i_1, j_1 и i_2, j_2. Если $\delta^{\min} \neq \delta^{\max}$, то случайно выбирается значение в пределах $\delta^{\min} \le \delta \le \delta^{\max}$, затем выполняется изменение элементов матрицы по формулам (7), (8) несколько раз (итераций). В первой альтернативной ветви алгоритма эти итерации проводятся независимо друг от друга, во второй – последовательно, начиная со второй итерации, первая пара индексов задается равной второй паре предыдущего шага (m): $i_1^{(m+1)} = i_2^{(m)}$, $j_1^{(m+1)} = j_2^{(m)}$, $m \ge 1$. Ветви выбираются случайным образом. В $M_2(X_n)$ случайным образом выбранные две строки с индексами i_1, i_2 обмениваются значениями $x_{i_1,j}^{mut_2(n)} = x_{i_2,j}^n$, $x_{i_2,j}^{mut_2(n)} = x_{i_1,j}^n$, $j = 1,..., N^j$, затем образованная матрица корректируется в элементах этих строк для восстановления целостности плана. Аналогичные операции во второй ветви алгоритма производятся для двух столбцов, также выбранных случайным образом. В $M_3(X_n)$ случайно выбранные столбцы или строки либо одновременно строки и строки матрицы X_n обнуляются (выбор ветви алгоритма случаен). Элементы после обнуления заполняются в случайно выбранной последовательности по алгоритму инициализации $I_1(X_0)$.

ЛП пытается улучшить найденное приближенное решение с помощью алгоритмов на основе перебора. В $L_1(X_n)$ случайно выбираются индексы i_1, j_1 и i_2, j_2. Далее проводится проверка возможности улучшения ЦФ по формулам (7)-(8). Для этого находится диапазон возможных изменений $\delta^{\min} \le \delta \le \delta^{\max}$. Если $\delta^{\min} \neq \delta^{\max}$, то в его концевых и промежуточных точках, число которых является параметром алгоритма, проверяется возможность уменьшения ЦФ. При появлении такой возможности план корректируется. Далее во вложенных циклах с помощью перебора значений индексов i_2, j_2 выполняются аналогичные действия $k^{ls} N^i N^j$ раз, где k^{ls} – параметр алгоритма. Вариант алгоритма $L_2(X_n)$ отличается от $L_1(X_n)$ выбором начальных индексов:

$$i_1 = I_s(r_1^2 N^i N^j),\ j_1 = J_s(r_1^2 N^i N^j),\ i_2 = I_s(r_2^2 N^i N^j),\ j_2 = J_s(r_2^2 N^i N^j),$$

где r_1, r_2 – случайные числа в интервале [0,1]. В этом варианте ЛП предпочтение оказывается начальным элементам плана, которым соответствуют небольшие значения c_{ij}. Алгоритм $L_3(X_n)$ также основан на учете значений c_{ij}. В нем на-

чальные элементы с индексами i_1, j_1 и i_2, j_2 выбираются случайно: соответствующие либо малым, либо большим c_{ij}.

Один из вариантов ЛП выполняется всегда для каждого $X_n \in W$, но операторы скрещивания и мутации, входящие в ЭА, выполняются не каждый раз, а с заданными вероятностями p^{cros} и p^{mut} соответственно. Использование ЛП с полным исключением ЭА имеет смысл только в параллельной версии алгоритмов, если сгенерированы большие популяции решений. В этом случае при начальной инициализации создается большое число разбросанных по пространству начальных решений – начальных точек, от которых происходит поиск решений с помощью алгоритмов ЛП. Однако с увеличением размерности задач эффективность ЛП без ЕА быстро снижается. Работа ЭА без ЛП существенно проигрывает при решении задач небольшой размерности, но может применяться для нахождения приближенных решений задач большой размерности.

Таким образом, алгоритмы ЭА и ЛП могут работать независимо друг от друга, но в указанной последовательности, как показали численные эксперименты, их применение наиболее эффективно.

MPI распараллеливание реализовано на 2-D сетке процессов $S \times P$, где S – число популяций, P – число групп решений внутри популяции. На рис. 1 показана схема передачи приближенных решений X и значений ЦФ $Z(X)$ между MPI процессами p_{sr}, $s = 1,2,...,S$, $r = 1,2,...,P$. Внутри популяций на каждой итерации происходит передача данных для организации параллельных расчетов, в каждой группе управляют этим процессы p_{s1}, $s = 1,2,...,S$. Через заданное число итераций между популяциями происходит обмен результатами поиска: решение, имеющее наименьшее значение ЦФ, передается процессами p_{s1}, $s = 1,2,...,S$ управляющему процессу p_{11}, затем лучшее из всех рассылается процессом p_{11} во все остальные популяции для замены худших решений. В общем случае, если на протяжении заданного числа итераций не происходит улучшения ЦФ, то управляющий процесс p_{11} принимает решение о завершении поиска решения. Решение о прекращении или продолжении расчетов рассылается подчиненным процессам на каждой итерации. Данные между процессами передаются с помощью двухточечных и коллективных MPI процедур.

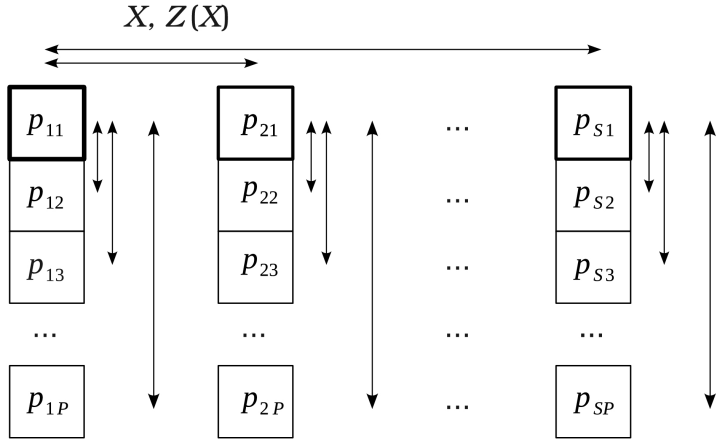

Рис. 1. Схема передачи данных на двумерной сетке MPI процессов.

Результаты вычислительных экспериментов

Для проверки эффективности разработанных алгоритмов были проведены эксперименты с использованием технологии распараллеливания MPI на узлах вычислительного кластера ВЦ ДВО РАН, оснащенных четырьмя шестиядерными процессорами AMD Opteron 8431 2,4 ГГц (24 ядра на узле), данные между узлами передавались в среде InfiniBand.

В [6] представлены результаты решения НТЗ размерностей 7×7 и 10×10 с нелинейными функциями **A**, **B**, **C**, **D**, **E**, **F** с помощью шести программ. Функции **A** и **B** представляют собой аппроксимации функциями арктангенса ступенчатой и кусочно-линейной функций. Функции **C** и **D** – функции возведения в квадрат и извлечения квадратного корня соответственно. Функция **E** имеет один максимум, функция **F** представляет собой осциллирующую функцию с возрастающей амплитудой. Эти функции и задачи описаны в [2, 5, 6].

В первой группе представленных ниже результатов численных экспериментов решались задачи [6], дополнительно также проведены вычисления с исходными прототипами функций **A** и **B**, обозначенных **a** и **b** соответственно, – ступенчатой и кусочно-линейной функциями [2].

В табл. 1 показаны результаты решения задачи 7×7.

Таблица 1

Функция	Z^{opt}	ε	T_1^{avg}	T_{mpi}^{avg}	T_{mpi}^{max}
a	0	10^{-6}	0,006	0,001	0,003
A	4, 26458	10^{-5}	30	1,5	3,5
b	179,400	10^{-6}	0,005	0,001	0,006
B	**179,385**	$5 \cdot 10^{-6}$	0,012	0,001	0,005
C	2535,29	10^{-6}	2,0	0,14	0,27
D	480,164	10^{-6}	0,002	0,001	0,002
E	**204,719**	10^{-6}	0,060	0,005	0,015
F	**39, 6725**	$2 \cdot 10^{-2}$	19	6,0	26

Поиск решения проводился до момента нахождения значения ЦФ Z^{opt} с заданной относительной погрешностью ε. Значения Z^{opt} найдены предварительно в результате всех проведенных вычислительных экспериментов. В столбце Z^{opt} показаны найденные минимальные значения ЦФ, они либо равны, либо меньше (выделены жирным шрифтом) приведенных в [6]. В столбце T_1^{avg} показаны средние значения времени решения в секундах для непараллельного решения, в столбцах T_{mpi}^{avg}, T_{mpi}^{max} – среднее и максимальное время для параллельного способа решения на 24 MPI процессах по результатам 200 запусков решения задач. Методика тестирования алгоритмов существенно отличается от используемой в [6], поэтому о результатах сравнения можно судить лишь приблизительно. Для функций **B**, **E** время непараллельного решения меньше, чем в [6] при меньших значениях Z^{opt}. Время получения решения для функции **D** значительно меньше, чем в [6], но больше для **A**, **C**. Использование функции **F** порождает НТЗ с большим

числом экстремумов, для функции **F** значение Z^{opt} (табл. 1) значительно меньше, чем в [6].

В табл. 2 приведены аналогичные результаты решения задачи 10×10. Для **D**, **F** находятся решения с меньшими значениями Z^{opt} при сопоставимом с [6] временем непараллельного решения. Время решения для функций **B**, **E** примерно на уровне [6], но для функции **C** оно больше, чем в [6].

Таблица 2

Функция	Z^{opt}	ε	T_1^{avg}	T_{mpi}^{avg}	T_{mpi}^{max}
a	158,000	$2 \cdot 10^{-2}$	0,46	0,029	0,26
A	**164,237**	10^{-3}	126	4,4	19
b	147,000	10^{-6}	0,022	0,002	0,018
B	146,987	10^{-6}	10	0,51	0,97
C	4401,65	10^{-6}	8,9	0,84	1,5
D	**377,247**	10^{-6}	0,033	0,002	0,015
E	71,6566	10^{-6}	1,0	0,058	0,19
F	**106,351**	10^{-4}	1,9	0,086	0,37

Для каждой функции (табл. 1, 2) подбирались значения параметров алгоритма, всюду $N^{pop} = 16$. Разработанные алгоритмы в случае недифференцируемых непрерывно функций стоимости **a** и **b** работают быстрее, чем с их аппроксимациями – функциями **A** и **B** соответственно (табл. 1, 2). Это связано с меньшим объемом вычислений ЦФ для функций **a** и **b**. Параллельный вариант алгоритмов (табл. 1, 2) значительно уменьшает время решения для задачи 10×10, но в случае задачи небольшой размерности 7×7 это ускорение заметно меньше, что можно объяснить сопоставимыми дополнительными затратами на MPI распараллеливание.

Во второй группе представленных ниже результатов вычисленных экспериментов решались специально сгенерированные для этой цели задачи размерности 7×7, 8×8, …, 20×20. Проведенные эксперименты показали, что некоторые задачи решаются быстро, а другие требуют значительного объема вычислений. Поэтому для объективной оценки эффективности алгоритмов нужно проводить вычислительные эксперименты на достаточно большой серии задач – было сгенерировано по 40 задач каждой размерности.

Коэффициенты c_{ij} и параметры a_i, b_j задавались случайными целыми числами: $1 \le c_{ij} \le 50$, $i = 1,2,...,N^i$, $j = 1,2,...,N^j$; $1 \le a_i \le N^i N^j / 1{,}28$, $i = 1,2,...,N^i$; $1 \le b_j \le N^i N^j / 1{,}28$, $j = 1,2,..., N^j - 1$. Элемент b_{N^j} находился из (4):

$$b_{N^j} = \sum_{i=1}^{N^i} a_i - \sum_{j=1}^{N^j - 1} b_j.$$

Таким способом было сгенерировано $14 \times 40 = 560$ задач. В ниже представленных результатах расчетов данные задачи решались с кусочно-линейной функцией стоимости **h**:

$$f_h(x) = \begin{cases} x, & x \le 5; \\ 0{,}9x + 0{,}5, & 5 < x \le 10; \\ 0{,}8x + 1{,}5, & 10 < x \le 15; \\ 0{,}7x + 3, & x > 15. \end{cases}$$

Эта функция является примером формирования функции стоимости с предоставлением скидок на поставляемую продукцию: скидка 10% на заказ продукции в объеме от 5 до 10; 20% – от 10 до 15; 30% – больше 15. Функция **h** не является непрерывно дифференцируемой.

На рис. 2 показаны зависимости среднего времени T^{avg} решения задач от их размерности. По горизонтальной оси отложено число неизвестных – значения $N_x = N^i N^j$. Точки 49, 64, … , 400 соответствуют задачам размерностей 7×7, 8×8, … , 20×20.

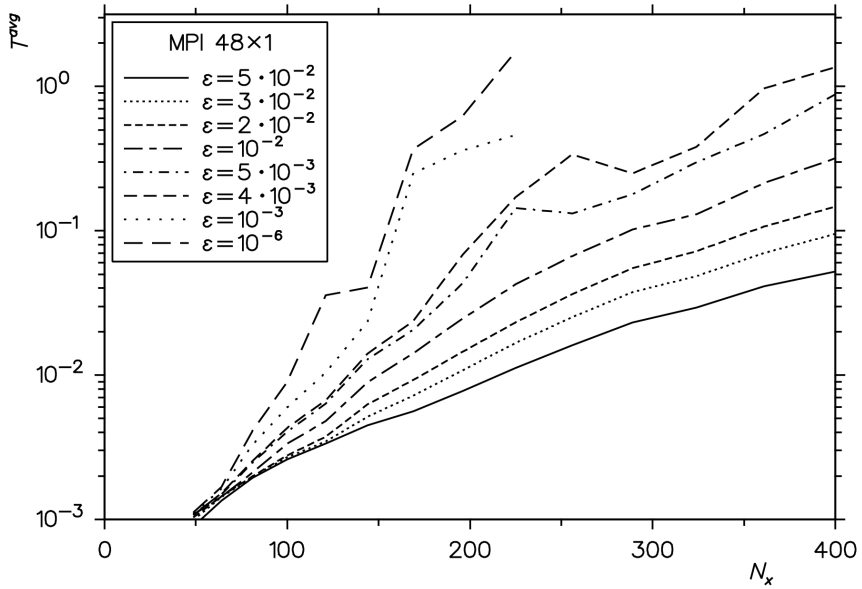

Рис. 2. Зависимость среднего времени решения от числа неизвестных.

На этом рисунке показаны зависимости T^{avg} при различных значениях относительной погрешности ε нахождения Z^{opt} с помощью параллельной версии программы на сетке MPI процессов 48×1. Для всех задач Z^{opt} найдены в предварительно проведенных экспериментах. Каждая задача решалась 250 раз. Таким образом, одна экспериментальная точка на графиках – результат усреднения $40 \times 250 = 10000$ запусков. Можно видеть, что при невысокой точности расчетов, но в большинстве случаев достаточной для практического применения, время решения с увеличением числа неизвестных возрастает умеренно. При $\varepsilon < 10^{-2}$ в отдельных точках имеется нарушение монотонности возрастания T^{avg} с ростом числа неизвестных. Это связано с тем, что среди 40 сгенерированных задач были отдельные задачи, которые требовали значительного времени нахождения ЦФ с такой точностью. В расчетах были использованы следующие значения параметров: $p_1^{init} = 0{,}1$; $p_2^{init} = 0{,}9$; $p_3^{init} = 0$; $p^{cros} = 0{,}06$; $p_1^{cros} = 0{,}2$; $p_2^{cros} = 0{,}4$; $p_3^{cros} = 0{,}4$; $p^{mut} = 0{,}9$; $p_1^{mut} = 0{,}1$; $p_2^{mut} = 0{,}6$; $p_3^{mut} = 0{,}3$; $k^{ls} = 0{,}2$; $p_1^{ls} = 0{,}1$; $p_2^{ls} = 0{,}7$; $p_3^{ls} = 0{,}2$.

С целью выбора способа распараллеливания для задач различной размерности проведено исследование влияния способа распараллеливания на зависимость T^{avg} от N_x. На рис. 3 показаны зависимости среднего времени решения T^{avg} рассматриваемых задач от их размерности при $\varepsilon = 10^{-2}$. Показаны зависимости для скалярной и параллельной версий программы на сетках MPI процессов 3×1, 6×4, 24×1, 24×4, 48×2. Каждая точка на графиках – результат усреднения времени решения задач T по $40\times250=10000$ запускам.

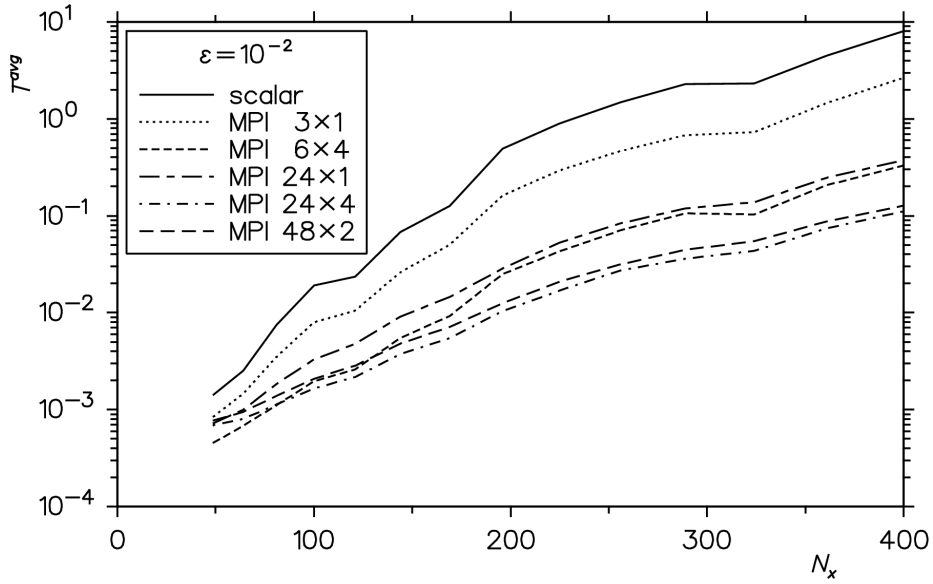

Рис. 3. Среднее время решения при различных способах распараллеливания.

Распараллеливание существенно уменьшает время решения T^{avg}, но различным образом для задач различной размерности. Для задач большой размерности наилучшие показатели достигнуты на сетке 24×4 MPI процессов, – например, для задачи 20×20 ускорение было больше 72, а эффективность распараллеливания – 0,76. Для задач небольшой размерности эффект от распараллеливания существенно меньше, так как дополнительные задержки MPI обменов заметно замедляют процесс решения. Максимальное ускорение для задач небольшой размерности получено на средней по размеру сетке 6×4 MPI процессов в пределах одного узла вычислительного кластера. При использовании двух и более узлов MPI обмены выполняются с увеличенными задержками в среде передачи данных InfiniBand.

На рис. 4 показаны зависимости числа успешно выполненных запусков R_p (в %) от времени решения T. Решалась задача 20×20 скалярной и параллельной версиями программы на сетках MPI процессов 3×1, 6×4, 24×1, 24×4, 48×2. Каждая кривая построена по результатам $40\times250=10000$ запусков с точностью $\varepsilon = 10^{-2}$ нахождения Z^{opt}.

Можно видеть, что при параллельном решении уменьшается среднее время решения, а минимальный разброс T наблюдается при максимальном числе «островов» S. Введение параллельности внутри популяции «островов» при сохранении общего числа процессов позволяет уменьшить среднее время решения, но не сокращает максимальное время решения задачи.

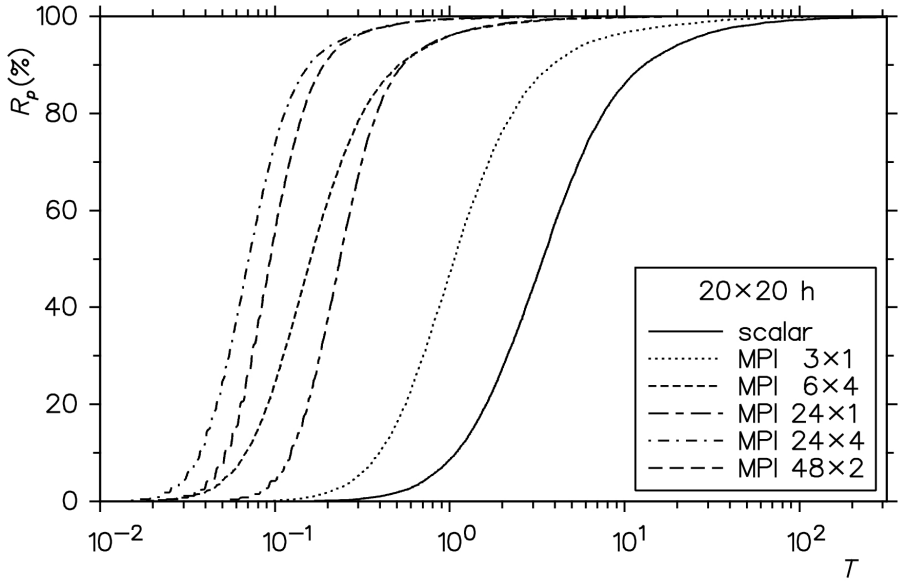

Рис. 4. Распределение числа успешных запусков по времени решения.

Заключение

Разработанный гибридный алгоритм эффективно решает нелинейные транспортные задачи с различными видами нелинейной функции стоимости. Параллельная версия алгоритма позволяет находить приближенные решения задач большой размерности с высоким уровнем масштабируемости при распараллеливании.

ЛИТЕРАТУРА

1. *Триус Е. Б.* Задачи математического программирования транспортного типа. – М.: Советское радио, 1967.
2. *Michalewicz Z.* Genetic algorithms + data structures = evolution programs. – Berlin, Heidelberg: Springer-Verlag, 1996.
3. *Пересветов В. В.* Параллельные эволюционные алгоритмы целочисленного решения нелинейных транспортных задач // Информационные технологии и высокопроизводительные вычисления: Материалы Междунар. науч.-практ. конф. – Хабаровск: Изд-во Тихоокеанского гос. ун-та, 2011. – С. 85-93.
4. *Пересветов В. В.* Эволюционные алгоритмы приближенного решения нелинейных транспортных задач // Информационные технологии и высокопроизводительные вычисления: Материалы Всероссийской науч.-практ. конф. – Хабаровск: Изд-во Тихоокеанского гос. ун-та, 2013. – С. 267-272.
5. *Мальковский С. И., Пересветов В. В.* Решение нелинейных транспортных задач методом роя частиц // Информатика и системы управления. – 2012. – № 2(32). – С. 54-64.
6. *Klansek U., Psunder M.* Solving the nonlinear transportation problem by global optimization // Transport: Research Journal of Vilnius Gediminas Technical University and Lithuanian Academy of Sciences. – 2010. – Vol. 25, № 3. – P.314-324.

Статья представлена к публикации членом редколлегии С.И. Смагиным

E-mail:
Пересветов Владимир Викторович – vvperesv@yandex.ru.