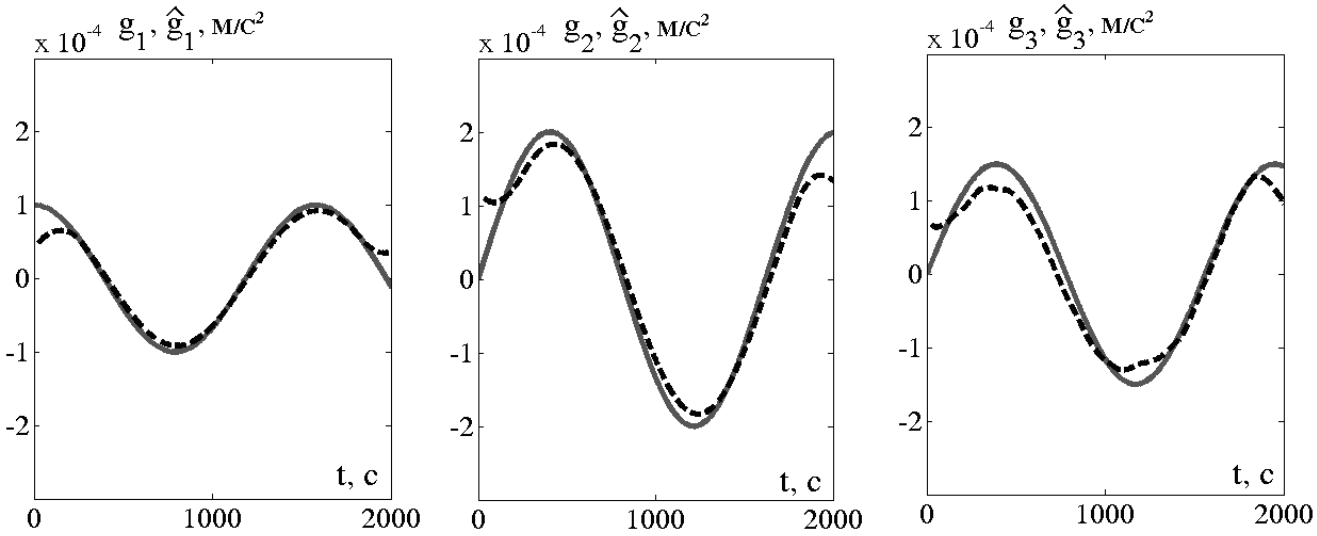


Рис. 2. Графики эволюции значений g_i (сплошные линии) и $\tilde{g}_i$ (пунктирные линии), $i = \overline{1,3}$.

ЛИТЕРАТУРА

1. *Девятисильный А.С., Числов К.А.* Модель корректируемой трехкомпонентной гравиинерциальной навигационной системы // Информатика и системы управления. – 2011. – № 4. – С.12-16.
2. *Ишлинский А.Ю.* Классическая механика и силы инерции. – М.: Наука, 1987.
3. *Калман Р., Фалб П., Арбиб М.* Очерки по математической теории систем. – М.: Мир, 1971.
4. *Андреев В.Д.* Теория инерциальной навигации. Корректируемые системы. – М.: Наука, 1967.
5. *Осипов Ю.С., Кряжимский А.В.* Задачи динамического обращения // Вестник РАН. – 2006. – Т.76, №7. – С. 615-624.

Статья представлена к публикации членом редколлегии Ю.Н. Кульчиным

E-mail:
Девятисильный Александр Сергеевич – devyatis@iacp.dvo.ru;
Числов Кирилл Александрович – kirillche@rambler.ru.

УДК 519.853.6

© 2012 г. **С.И. Мальковский,**
В.В. Пересветов, канд. физ.-мат. наук
(ВЦ ДВО РАН, Хабаровск)

РЕШЕНИЕ НЕЛИНЕЙНЫХ ТРАНСПОРТНЫХ ЗАДАЧ МЕТОДОМ РОЯ ЧАСТИЦ

Решаются транспортные задачи с нелинейными функциями стоимости произвольного вида. Предложен метаэвристический многороевый алгоритм метода роя частиц для приближенного решения поставленных задач. Приведены результаты вычислительных экспериментов по исследованию эффективности решения нелинейных транспортных задач различной сложности в параллельной реализации OpenMP разработанных алгоритмов.

Ключевые слова: транспортная задача, нелинейная функция стоимости, метод роя частиц, параллельные алгоритмы.

Введение

Необходимость в решении транспортных задач (ТЗ) возникает при планировании перевозок, когда нужно определить, от каких поставщиков и в каком количестве потребителям выгоднее получать продукт, как распределять заказы между предприятиями, размещать производства, склады и технические средства, проектировать различного рода сети (тепловые, электрические, связи).

В настоящей работе рассматриваются ТЗ с нелинейными функциями стоимости [1] произвольного вида. На практике такие задачи встречаются достаточно часто, так как производители продукции и перевозчики предоставляют при увеличении поставок различные, иногда весьма сложные, схемы скидок. Для некоторых других задач транспортного типа, – например, при проектировании технических сетей, единичная стоимость может, наоборот, возрастать, иногда скачками.

Среди приближенных методов решения нелинейных ТЗ в настоящее время применяются метаэвристические: метод имитации отжига [2], эволюционные и генетические алгоритмы [3 – 5], метод роя частиц [6] и др. В работе [7] был представлен алгоритм, разработанный для решения нелинейных ТЗ при помощи многороевого варианта метода роя частиц. Он основан на итерационных формулах, предложенных в [6] для решения линейных ТЗ замкнутого типа.

В настоящей работе усовершенствованы последовательные алгоритмы [7] решения нелинейных ТЗ. Для проверки эффективности разработанных алгоритмов были проведены испытания, аналогичные описанным в [8]: найдены решения ТЗ для различных типов нелинейной функции стоимости. Приведены результаты экспериментов по нахождению оптимального количества подроев для разработанного многороевого алгоритма метода роя частиц. Представлены также результаты расчетов для параллельной версии программы, созданной с использованием технологии OpenMP.

Постановка транспортной задачи

Пусть имеется m поставщиков (пунктов отправления) и n потребителей (пунктов назначения) однородного продукта. Количество продукта в пункте отправления i обозначим как a_i, $i = 1, 2, \ldots, m$. Потребность в продукте пункта назначения j обозначим как b_j, $j = 1, 2, \ldots, n$. При этом:

$$\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j. \qquad (1)$$

Стоимость перевозки продукта из пункта i в пункт j определяется выражением $c_{ij} f(x_{ij})$, где x_{ij} – объем перевезенного груза; c_{ij} – стоимость перемещения единицы объема (веса) груза для данного маршрута; $f(x_{ij})$ – нелинейная функция стоимости, зависящая от объема перевозимого груза. Матрица X с элементами x_{ij}, $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$ представляет собой совокупность объемов грузов и называется допустимым планом перевозок, если:

$$\sum_{j=1}^{n} x_{ij} = a_i, \quad i = 1, 2, \ldots, m; \qquad (2)$$

$$\sum_{i=1}^{m} x_{ij} = b_j, \quad j = 1, 2, ..., n; \tag{3}$$

$$x_{ij} \geq 0, \quad i = 1, 2, ..., m, \quad j = 1, 2, ..., n; \tag{4}$$

$$x_{ij} \in \mathrm{R}, \quad i = 1, 2, ..., m, \quad j = 1, 2, ..., n. \tag{5}$$

Множество допустимых планов перевозок обозначим через W. Общая стоимость перевозок записывается следующим образом:

$$Z(X) = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} f(x_{ij}). \tag{6}$$

Формулировка замкнутой нелинейной транспортной задачи: найти $X^{opt} = \arg\min_{X \in W} Z(X)$, где X^{opt} – оптимальный план перевозок.

Алгоритмы решения

Разработанный алгоритм базируется на многороевом варианте метода роя частиц, в котором весь рой разбивается на несколько подроев, в каждом из которых вычисления выполняются независимо от других. Через заданное число итераций подрои по топологии кольца обмениваются лучшими найденными решениями.

Общая схема работы алгоритма представлена на рис. 1, где изображены три основных блока, отвечающих за начальную инициализацию, вычисление координат и скоростей частиц и обмен решениями между подроями.

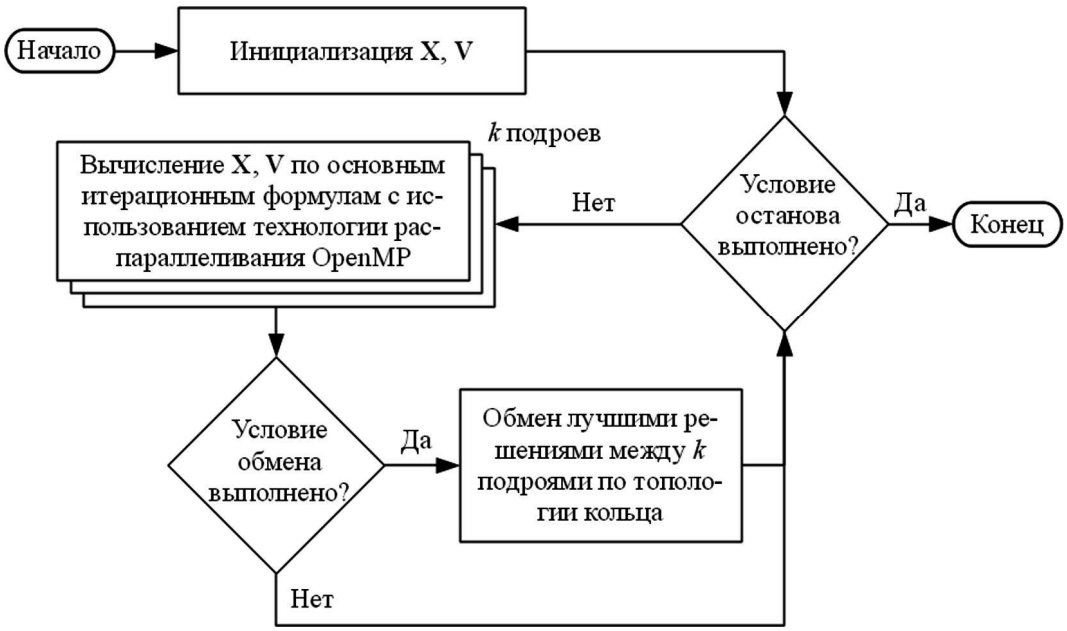


Рис. 1. Схема алгоритма поиска решения.

Работа алгоритма начинается с блока, в котором происходит создание и инициализация основных массивов данных и переменных. Создаются массивы $\mathbf{X}_j$, $\mathbf{V}_j$, $\mathbf{B}_j$, j = 1, 2, …, k, где k – число подроев, и массив $\mathbf{G}$. Массивы $\mathbf{X}_j$ содержат p частиц X_{ij}, i = 1, 2, ..., p, j = 1, 2, …, k которые, в свою очередь, являются приближенными решениями поставленной ТЗ – массивами размерности $m \times n$ с элементами x_{ij}. Каждому массиву $\mathbf{X}_j$ соответствует массив $\mathbf{V}_j$, в котором хранятся теку-

щие скорости частиц, и массив $\mathbf{B}_j$, хранящий лучшие решения, полученные частицами. В массиве **G** содержатся лучшие решения, найденные каждым подроем. Элементы этих массивов – V_{ij}, B_{ij}, G_j, i = 1, 2, ..., p, j = 1, 2, …, k также являются массивами размерности $m \times n$, соответствующей размерности ТЗ.

После того, как основные массивы созданы, происходит инициализация частиц во всех подроях некоторыми случайными решениями ТЗ. Рассмотрим алгоритм построения такого решения на примере одной из частиц.

Формирование случайного решения ТЗ начинается с присвоения всем элементам x_{ij} матрицы решения нулевых значений. Все x_{ij} помечаются как не просмотренные. После этого случайным образом выбирается строка i матрицы решения. Если все x_{ij} в этой строке уже просмотрены или в какой-либо из строк число не просмотренных элементов больше, чем в этой, случайным образом выбирается следующая строка. Иначе в этой строке осуществляется выбор не просмотренной переменной x_{ij}, которой соответствует наименьший по величине коэффициент c_{ij}. Значение этой переменной устанавливается равным меньшей из величин a_i, b_j.

Затем вносится поправка в величины a_i, b_j: $a_i = a_i - x_{ij}$, $b_j = b_j - x_{ij}$. После этого рассмотренная переменная помечается как просмотренная и случайным образом производится выбор следующей строки матрицы решения. Решение считается сформированным, если $b_j = 0$, j = 1, 2, …, n.

Во многих случаях оказывается эффективным применение дополнительного оператора мутации: к сформированным начальным решениям привносятся случайные изменения значений в точках $x_{i_0 j_0}$, $x_{i_0 j_1}$, $x_{i_1 j_0}$, $x_{i_1 j_1}$ матрицы решения, где индексы i_0, i_1, j_0 и j_1, выбираемые случайным образом, должны удовлетворять следующим условиям: $i_0 < i_1$, $j_0 < j_1$. Коррекция значений в этих точках осуществляется по формулам:

$$x_{i_0 j_0} = x_{i_0 j_0} + d\ ,\ x_{i_1 j_1} = x_{i_1 j_1} + d\ ,\ x_{i_0 j_1} = x_{i_0 j_1} - d\ ,\ x_{i_1 j_0} = x_{i_1 j_0} - d\ ,$$

где d – случайное число в интервале

$$[-\min(x_{i_0 j_0}, x_{i_1 j_1}), \min(x_{i_0 j_1}, x_{i_1 j_0})].$$

Этот оператор не нарушает целостность плана и применяется к каждому сформированному начальному решению до тех пор, пока значение целевой функции модифицированного решения не становится меньшим, чем у исходного. Применение данного оператора мутации при решении задач размерности 14×14 позволило уменьшить значения целевой функции начальных решений в среднем на 10%.

После того, как все начальные решения оказались сформированы, устанавливаются: $B_{ij} = X_{ij}$, i = 1, 2, …, p, j = 1, 2, …, k. Значения G_j, j = 1, 2, …, k инициализируются решениями с наименьшим $Z(X_{ij})$ из соответствующих подроев. Начальная скорость частицы i в подрое j вычисляется по формуле:

$$V_{ij}^l = j_1\left(B_{ij}^l - X_{ij}^l\right) + j_2\left(G_j^l - X_{ij}^l\right), \qquad (7)$$

где l – номер шага итерационного процесса (в (7) на нулевом шаге l = 0); j_1 – случайное число из интервала [0; 0,2], сгенерированное по равномерному закону распределения, если $B_{ij}^l \neq X_{ij}^l$ и $G_j^l \neq X_{ij}^l$. Если $B_{ij}^l \neq X_{ij}^l$ и $G_j^l = X_{ij}^l$, то $j_1 = 1$. Если

$B_{ij}^{l} = X_{ij}^{l}$ и $G_j^{l} \neq X_{ij}^{l}$, то $j_2 = 1$. При этом $j_1 + j_2 = 1$.

После выполнения начальной инициализации начинается итерационный процесс поиска решения, на каждом из шагов которого в соответствующем блоке алгоритма осуществляется вычисление скорости и положения частиц:

$$V_{ij}^{l+1} = l_1 V_{ij}^{l} + l_2 \left[j_1 \left(B_{ij}^{l} - X_{ij}^{l} \right) + j_2 \left(G_j^{l} - X_{ij}^{l} \right) \right], \quad (8)$$

$$X_{ij}^{l+1} = V_{ij}^{l+1} + X_{ij}^{l}. \quad (9)$$

В формуле (8) коэффициенты j_1 и j_2 вычисляются так же, как и в формуле (7). Коэффициент $l_1 = 1$, если $B_{ij}^{l} = X_{ij}^{l}$ и $G_j^{l} = X_{ij}^{l}$. В противном случае l_1 – это случайное число из интервала [0,8; 1,0], сгенерированное по равномерному закону распределения. Коэффициент l_2 задается следующим образом: $l_2 = 1 - l_1$.

После каждого вычисления текущих координат частицы полученное решение проверяется на соответствие условиям (2) – (4). Это требуется потому, что после применения формул (8) – (9) некоторые элементы матриц X_{ij} могут стать отрицательными. Из-за ограниченной точности расчетов может перестать выполняться условие (1). Если условия не выполняются – применяется один из восстанавливающих операторов. Первый из них аналогичен описанному в [6]: он модифицирует решение так, что все его элементы становятся положительными. Второй оператор для выполнения условия (1) изменяет значения одного из столбцов и одной из строк решения.

Затем, для предотвращения преждевременной сходимости алгоритма к локальным минимумам, используется оператор мутации, применяемый к 0,5% решений. Этот оператор действует следующим образом: из матрицы решения X случайным образом выделяется подматрица X' размерности $p \times q$, где $p < m$ и $q < n$ – случайные числа, сгенерированные по равномерному закону распределения. После этого вычисляются величины $a'_i = \sum_{j=1}^{q} x'_{ij}$, $i = 1, 2, \ldots, p$ и $b'_j = \sum_{i=1}^{p} x'_{ij}$, $j = 1, 2, \ldots, q$, где x'_{ij} – элемент матрицы X'. К матрице X' и величинам a'_i и b'_j применяется модифицированный алгоритм, используемый для генерации начальных решений задачи. Модификация заключается в том, что на каждом шаге этого алгоритма выбирается в случайном порядке не строка матрицы, а следующий не просмотренный элемент x'_{ij}.

После применения алгоритма мутации величины перевозимых грузов в подматрице X' оказываются перераспределены случайным образом. Использование оператора мутации при решении задач размерности 10×10 позволило уменьшить значения целевых функций получаемых решений в среднем на 10%.

При завершении каждого шага итерационного процесса обновляются значения B_{ij}, G_j.

Для всех B_{ij}:

вычисляем по формуле (6) значение целевой функции $Z(X_{ij})$;

если $Z(X_{ij}) < Z(B_{ij})$, то устанавливаем $B_{ij} = X_{ij}$.

Для каждого G_j, $j = 1, 2, \ldots, k$:

в подрое j ищем частицу X_{ij}, для которой $Z(X_{ij})$ минимально;

если $Z(X_{ij}) < Z(G_j)$, то устанавливаем $G_j = X_{ij}$.

В третьем блоке алгоритма при выполнении условия обмена происходит передача по топологии кольца между подроями лучших найденных решений:

$X_{tmp} = G_j$, при $j = 1$;

$G_j = G_{j+1}$, при $j = 1, 2, \ldots, k-1$;

$G_j = X_{tmp}$, при $j = k$.

Выход из итерационного процесса поиска решения осуществляется при выполнении условия останова: на протяжении заданного числа итераций не удается улучшить решение. При этом в качестве решения ТЗ выбирается такое G_j, для которого значение $Z(G_j)$ минимально.

Представленный выше алгоритм был реализован в виде параллельной программы. Разработка велась на языке C с использованием технологии OpenMP, применяющейся для программирования мультипроцессорных систем в модели общей памяти. Введенные спецкомментарии OpenMP указывают на проведение вычислений параллельно в циклах второго блока (рис. 1): рассчитываются текущие скорости и координаты всех частиц во всех подроях. После завершения тела распараллеленного цикла происходит неявная барьерная синхронизация, а затем, если выполняются условия обмена, происходит обмен лучшими решениями между подроями. Далее следует проверка выполнения условия останова.

Результаты вычислительных экспериментов

В рамках проведенных численных экспериментов решались ТЗ размерности 7×7 и 10×10 (с общим числом неизвестных 49 и 100), предложенные в [5]. Также решалась более сложная задача размерности 14×14, для которой в [9] представлены результаты целочисленного решения нелинейной ТЗ эволюционным методом, дополненным локальным поиском, с распараллеливанием на двумерной сетке MPI процессов. Эта задача была сформирована следующим образом: коэффициенты c_{ij}, используемые в формуле (6), и значения массива плана перевозок x_{ij} заполнялись случайными целыми числами, затем вычислялись a_i и b_j по формулам (2) и (3). При решении созданной таким образом задачи значения x_{ij} в дальнейшем уже не использовались.

При расчетах использовались нелинейные функции стоимости, представленные в табл. 1. Функция int() осуществляет приведение к целому числу путем отбрасывания дробной части. Во всех функциях параметры $P_A = P_B = 1000$. В функции A параметр $S = 2$, в остальных – $S = 5$. Функции A и B представляют собой аппроксимации арктангенсами ступенчатой A' и кусочно-линейной функции B' соответственно. Функция E имеет один максимум. Подробное описание нелинейных функций стоимости и их графики приводятся в [5], [8] и [7].

Результаты проведенных численных экспериментов по оценке скорости сходимости алгоритма приведены на рис. 2 и 3 для задач размерности 10×10 и 14×14 с нелинейной функцией B'.

Таблица 1

Функция A'	$f(x) = \begin{cases} \text{int}(x/2), & x < 10, \\ 5, & x \geq 10. \end{cases}$
Функция A	$f(x) = \sum_{n=1}^{N} \arctan(P_A(x - nS))/p + 0.5N,\ N = 5.$
Функция B'	$f(x) = \begin{cases} x/5, & x < 5, \\ 1, & 5 \leq x < 10, \\ x/5 - 1, & x \geq 10. \end{cases}$
Функция B	$f(x) = (x/S)[\arctan(P_B x)/p + 0.5] + (1 - x/S)[\arctan(P_B(x - S))/p + 0.5] + (x/S - 2)[\arctan(P_B(x - 2S))/p + 0.5].$
Функция C	$f(x) = x^2.$
Функция D	$f(x) = \sqrt{x}.$
Функция E	$f(x) = [1 + (x - 2S)^2]^{-1} + [1 + (x - 2.25S)^2]^{-1}. + [1 + (x - 1.75S)^2]^{-1}.$

На графиках отображены средние, максимальные и минимальные значения минимизируемых функций через каждые 20 шагов итерационного процесса поиска решения. Для всех представленных результатов численных экспериментов было проведено 100 независимых испытаний с итерационным процессом поиска решения, начиная с самого начала.

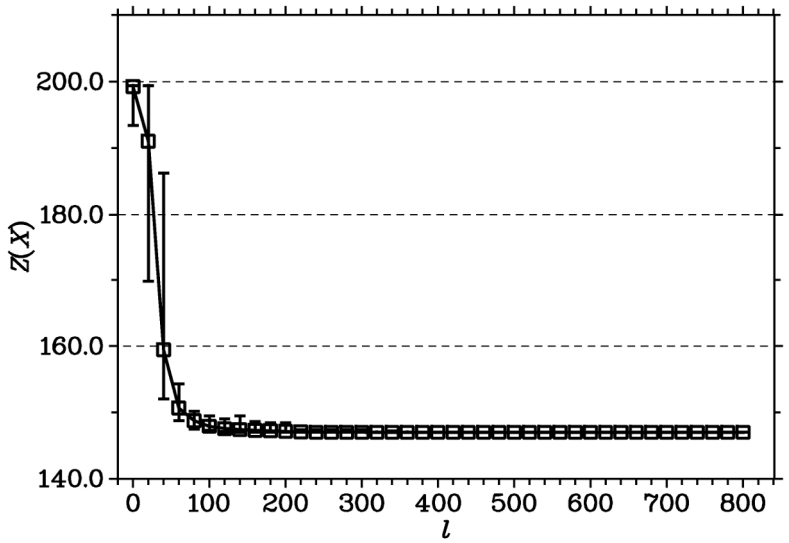


Рис. 2. Стоимость *Z(X)*, задача 10×10, функция *B'*.

Решение для задачи 10×10 быстро сходится (рис. 2), а разброс получаемых результатов является небольшим. На рис. 3 приведен аналогичный график для существенно более сложной задачи 14×14 с большим числом субоптимальных решений. Из этого графика видно, что разработанный алгоритм также показывает уверенную сходимость, но с достаточно большим разбросом значений целевой функции. При построении графиков, показанных на рис. 2 и 3, и далее в тексте,

если не будет указано иное, использовались следующие параметры алгоритма: количество групп – 20; частиц в каждой группе – $m \times n$; обмен лучшими решениями через каждые 20 шагов; общее число шагов – 800.

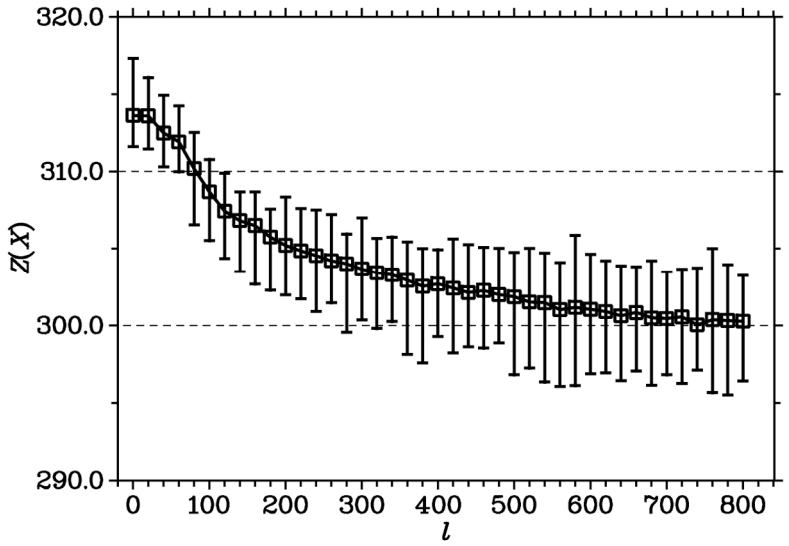


Рис. 3. Стоимость *Z(X)*, задача 14×14, функция *B′*.

Для определения оптимального числа групп k, при котором гарантируется нахождение решения с заданным значением целевой функции за наименьшее время, проводился следующий эксперимент: выполнялось по 100 независимых испытаний для каждого варианта настроек алгоритма, отличающихся числом групп, на которое разделяется фиксированное количество частиц, равное 2000. Вычисления останавливались, когда значение целевой функции становилось меньше некоторого порогового значения.

В табл. 2 приведено время нахождения решения с пороговыми значениями целевой функции Z_{min} для задачи 10×10. Величины Z_{min} были выбраны таким образом, чтобы они не превышали значений целевых функций лучших решений из [8] более чем на 1% (функция *D* – 4%).

Таблица 2

Функция	Z_{min}	$k = 2$	$k = 6$	$k = 12$	$k = 20$	$k = 40$
A′	173,1	86%	87%	96%	98%	7,3
A	174,1	8,3	7,4	7,8	8,6	8,9
B′	147,1	4,9	3,5	3,3	3,7	3,8
B	147,1	6,7	4,5	4,7	4,9	5,5
C	4445,7	49%	93%	1,8	1,9	2,0
D	403,1	0,06	0,07	0,07	0,06	0,06
E	71,9	1,3	1,2	1,3	1,4	1,5

Если при данных настройках алгоритма решение с искомым значением целевой функции не удавалось найти во всех 100 запусках программы, то вместо времени в этой таблице показана доля успешных запусков программы в процентах. Вычисления выполнялись непараллельной версией программы на узлах вычислительного кластера ВЦ ДВО РАН, оснащенных двумя четырехъядерными процессорами Intel Xeon E5450 3,00 ГГц.

Из табл. 2 видно, что увеличение числа подроев, на который разделяется рой, позволяет находить решения с меньшим значением целевой функции для функций A' и C. При этом для остальных нелинейных функций стоимости существует некоторое оптимальное число подроев, при котором искомое решение находится за минимальное время. Единственным исключением здесь является функция D, для которой решение с заданным значением целевой функции находится еще при генерации начальных решений.

При сравнении данных из табл. 2 с данными, приведенными в [8], можно видеть, что разработанная программа без распараллеливания решает задачу с функцией стоимости A в несколько раз быстрее. Решение задачи для функции B разработанный алгоритм также находит за меньшее время. Для остальных функций стоимости разработанная программа находит решение за сопоставимое или ненамного большее время по сравнению с другими методами, представленными в [8].

Далее представлены результаты проверки точности нахождения решений в сравнении с опубликованными данными. В табл. 3 показаны результаты решения задач размерности 7×7 и 10×10 для различных типов нелинейной функции.

Таблица 3

Функция	Задача 7×7			Задача 10×10		
	Z_{min}^{opt}	Z_{best}^{opt}	ΔZ^{opt}	Z_{min}^{opt}	Z_{best}^{opt}	ΔZ^{opt}
A'	0	0	0%	165,00	173,00	4,8%
A	4,26	4,26	0%	166,14	174,07	4,5%
B'	194,2	203,75	4,9%	147,00	159,79	8,7%
B	182,03	183,59	0,9%	146,99	146,99	0%
C	2535,31	2534,34	-0,04%	4401,79	4401,65	0%
D	533,91	480,16	-10%	384,14	388,91	1,2%
E	204,74	204,73	0%	71,66	71,66	0%

В столбцах Z_{min}^{opt} приведены лучшие результаты, полученные разработанным алгоритмом для ограниченного числа итераций. В столбцах Z_{best}^{opt} даны лучшие результаты, приведенные в [8]. Для каждого варианта задачи выполнялось по 100 испытаний, из результатов которых выбиралось лучшее решение.

При сравнении полученных результатов с результатами, приведенными в [8], можно видеть, что предложенный алгоритм находит решения, не уступающие решениям, полученным другими методами, для большинства функций стоимости. Только для функции D при размерности задачи 7×7 некоторые методы, приведенные в [8], получают меньшее значение целевой функции.

При этом для задачи 10×10 предложенный алгоритм позволяет находить заметно лучшие решения для функций стоимости A', A, B', D по сравнению с другими методами. Эта закономерность повышения точности решения задач при увеличении их размерности является важной, так как разработанный алгоритм предназначен прежде всего для получения приближенных решений сложных задач большой размерности.

Далее приведены результаты проведенных численных экспериментов для

параллельной реализации разработанного многороевого алгоритма. Для оценки масштабируемости, т.е. зависимости коэффициента ускорения решения задачи от числа задействованных процессорных ядер, проводилось решение ТЗ размерности 7×7, 10×10 и 14×14 (функция стоимости B') с использованием от 1 до 24 процессорных ядер. Эксперименты по масштабированию выполнялись на узлах, оснащенных четырьмя шестиядерными процессорами AMD Opteron Istanbul 8431 2,4 ГГц (24 ядра на узле). На каждом ядре выполнялся один OpenMP-поток. При этом искалось близкое к оптимальному решение ТЗ, при котором значение целевой функции не превышало 200,1 для задачи 7×7; 147,1 – для задачи 10×10 и 298,0 – для задачи 14×14. При нахождении такого решения вычислительный процесс останавливался, а время расчетов запоминалось и усреднялось по 200 запускам для задачи 7×7, по 100 запускам – для задач 10×10 и 14×14. Результаты испытаний приведены на рис. 4.

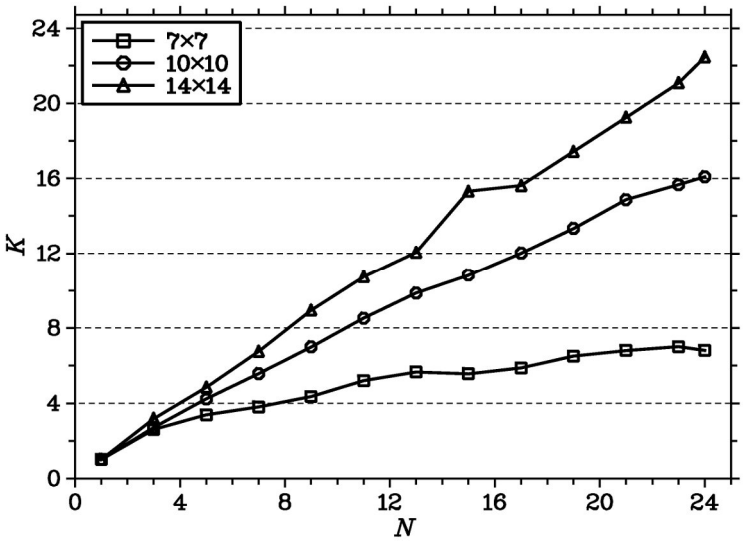
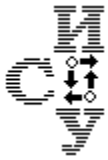


Рис. 4. Зависимость ускорения K от числа ядер N.

Ускорение работы программы рассчитывалось по формуле: $K = T_1 / T_N$, где T_1 – среднее время исполнения на одном ядре, а T_N – на N ядрах. Из рис. 4 видно, что ускорение в работе программы при добавлении новых ядер для задачи 14×14 больше, чем для задач 7×7 и 10×10. Это объясняется тем, что при увеличении размерности задачи возрастает относительная доля вычислений, выполняемых параллельно. Из этого следует, что чем больше размерность задачи, тем больше преимущество параллельных версий разработанного алгоритма.

Заключение

Представленный многороевый алгоритм метода роя частиц в последовательной программной реализации позволяет решать транспортные задачи с различными типами нелинейных функций стоимости со скоростью и точностью на уровнях, достигнутых другими методами. При решении задач большой размерности разработанный параллельный алгоритм показал хорошую масштабируемость и является одним из наиболее перспективных для решения сложных нелинейных транспортных задач.



ЛИТЕРАТУРА

1. *Триус Е. Б.* Задачи математического программирования транспортного типа. – М.: Советское радио, 1967.
2. *Altiparmak F., Karaoglan I.* An adaptive tabu-simulated annealing for concave cost transportation problems // Journal of the Operational Research Society. – 2008. – Vol. 59, № 3. – P. 331-341.
3. *Michalewicz Z.* Evolutionary computation techniques for nonlinear programming problems // International Transactions in Operational Research. – 1994. – Vol. 1, № 2. – P. 223-240.
4. *Sheng S., Dechen Z., Xiaofei X.* Genetic algorithm for the transportation problem with discontinuous piecewise linear cost function // International Journal of Computer Science and Network Security. – 2006. – Vol. 6, № 7A. – P. 182-190.
5. *Michalewicz Z.* Genetic algorithms + data structures = evolution programs. – Berlin, Heidelberg: Springer, 1996.
6. *Huang H., Hao Z.* Particle swarm optimization algorithm for transportation problems // Particle swarm optimization, ed. Aleksandar Lazinica, InTech. – 2009. – P. 275-290.
7. *Мальковский С. И., Пересветов В. В.* Метод роя частиц в решении нелинейных транспортных задач. Препринт ВЦ ДВО РАН. – № 169. – Хабаровск, 2011.
8. *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.
9. *Пересветов В. В.* Параллельные эволюционные алгоритмы целочисленного решения нелинейных транспортных задач // Информационные технологии и высокопроизводительные вычисления: материалы Международной науч.-практ. конф. – Хабаровск: Изд-во Тихоокеан. гос. ун-та, 2011. – С. 85-93.

Статья представлена к публикации членом редколлегии С.И. Смагиным

E-mail:
Мальковский Сергей Иванович – sergey.malkovsky@gmail.com
Пересветов Владимир Викторович – vvperesv@yandex.ru

УДК 004.438

© 2012 г. **Д.А. Потапов**
(Национальный исследовательский ядерный университет «МИФИ», Москва)

ДИНАМИЧЕСКАЯ ИНТЕГРАЦИЯ ФУНКЦИЙ MATLAB В КОМПЬЮТЕРНЫЕ СИСТЕМЫ МОДЕЛИРОВАНИЯ, РЕАЛИЗОВАННЫЕ НА ЯЗЫКЕ JAVA

Рассмотрена процедура динамической интеграции функций Matlab в Java, позволяющая осуществлять добавление новых моделей в экспертные системы, предложен метод преодоления проявляющихся при этом ограничений языков – таких как необходимость повторной компиляции проекта Java, преобразования функций в сценарии Matlab, а также передачи массива в качестве списка параметров в Java.

Ключевые слова: интеграция Matlab и Java, разработка экспертных систем, математическое моделирование, динамическая интеграция кода, рефлексия.